Linux Professional Institute

Open Source Essentials

Phiên bản 1.0
Tiếng Việt

050

Table of Contents

Linux Professional Institute

Chủ đề 051: Những nguyên tắc cơ bản về Phần mềm

051.1 Những nguyên tắc cơ bản về Phần mềm

Tham khảo các mục tiêu LPI

Open Source Essentials version 1.0, Exam 050, Objective 051.1

Khối lượng

2

Các lĩnh vực kiến thức chính

- Hiểu rõ khái niệm về mã nguồn và thực thi mã
- Hiểu rõ khái niệm về trình biên dịch và trình thông dịch
- Hiểu rõ khái niệm về thư viện phần mềm

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Mã nguồn
- Các chương trình thực thi
- Mã byte
- Mã máy
- Trình biên dịch
- Trình liên kết
- Trình thông dịch
- Máy ảo thời gian chạy
- Thuật toán
- Thư viện phần mềm
- Liên kết tĩnh và động

exam: "050" topic: "051" objective: "051.1" type: "lm-lesson" title: "051.1 Bài 1" menu: main: identifier: "lesson-050-051-051.1-1" parent: "objective-050-051-051.1" ---

Linux Professional Institute

Bài 1

Chứng chỉ:	Open Source Essentials
Phiên bản:	1.0
Chủ đề:	051 Những nguyên tắc cơ bản về Phần mềm
Mục tiêu:	051.1 Thành phần của Phần mềm
Bài học:	1 trên 1

Giới thiệu

Phần mềm mã nguồn mở và *tự do* — thường được viết tắt là FOSS (Free and Open Source Software) — đã trở thành một phần không thể thiếu trong cuộc sống hàng ngày dù chúng ta hầu như không hề nhận ra. Ví dụ: FOSS có thể đứng sau tất cả các hoạt động của chúng ta trên internet dưới một số hình thức: trên máy tính nơi chúng ta xem các trang web trong trình duyệt hoặc trên các máy chủ lưu trữ và phân phối các trang web này ngay khi chúng ta truy cập chúng.

Phần mềm là gì?

Tuy nhiên, trước khi tìm hiểu về toàn bộ các chi tiết cụ thể về phần mềm mã nguồn mở và tự do, chúng ta cần làm rõ *phần mềm* thực sự là gì. Hãy bắt đầu với một mô tả tổng quan: Phần mềm là bộ phận phi vật lý và phi vật chất của máy tính ở mọi hình thức. Phần mềm sẽ đảm bảo rằng các bộ phận vật lý (*phần cứng*) của máy tính có thể tương tác và máy tính có thể chấp nhận lệnh và thực thi các tác vụ.

Điện thoại thông minh, máy tính xách tay hay chính máy chủ trong trung tâm dữ liệu đều chỉ là những cỗ máy làm từ nhựa và kim loại khi tắt nguồn. Ngay sau khi được bật, phần mềm sẽ khởi động dưới dạng các chuỗi lệnh được mã hóa để điều khiển các thành phần riêng lẻ của máy và cho

phép người dùng tương tác với máy cũng như thực hiện các tác vụ cụ thể bằng cách gọi các ứng dụng riêng biệt.

Công việc của *nhà phát triển phần mềm* là phân tích các tác vụ mà máy tính phải thực hiện và chỉ rõ các nhiệm vụ ấy theo một cách sao cho máy tính có thể thực hiện được. Những công cụ được các nhà phát triển sử dụng cũng nhiều và đa dạng không kém gì so với các tác vụ mà phần mềm phải thực hiện.

Một số tác vụ phần mềm được kết nối rất chặt chẽ với phần cứng và kiến trúc của máy tính (ví dụ: việc chỉ định và quản lý bộ nhớ hoặc xử lý các tiến trình khác nhau). Do đó, *Lập trình viên hệ thống* sẽ làm việc sát sao với phần cứng.

Mặt khác, *các nhà phát triển ứng dụng* sẽ tập trung nhiều hơn vào người dùng và các chương trình ứng dụng cho phép người dùng thực hiện các tác vụ của họ một cách hiệu quả và trực quan. Ví dụ về một ứng dụng phức tạp chính là một chương trình xử lý văn bản cung cấp tất cả các chức năng định dạng văn bản trong các bảng chọn hoặc nút bấm và cũng có thể hiển thị văn bản dưới dạng sẵn sàng để in.

Nói chung, *thuật toán* chính là một cách để giải quyết vấn đề. Ví dụ: để tính giá trị trung bình, thuật toán thông thường sẽ là cộng một tập hợp các giá trị và chia tổng cho tổng số giá trị. Theo truyền thống, các thuật toán sẽ được thiết kế và thực hiện bởi các lập trình viên; ngày này, các thuật toán cũng có thể được tạo ra bởi trí tuệ nhân tạo.

Các khái niệm trong chương này có thể giúp chúng ta hiểu được những điểm mạnh cũng như những rủi ro của FOSS và có thể đưa ra những lựa chọn sáng suốt hay thậm chí là quyết định xem bản thân chúng ta có muốn trở thành một nhà phát triển phần mềm hay không.

Ngôn ngữ Lập trình

Ngôn ngữ lập trình là những ngôn ngữ nhân tạo có cấu trúc chặt chẽ được sử dụng để giúp máy tính biết phải làm gì. Các chương trình thường được viết bằng văn bản, nhưng một số ngôn ngữ sẽ được viết dưới dạng đồ họa. Các nhà phát triển phần mềm sẽ viết các chỉ dẫn (được gọi là *mã* - code) cho máy tính bằng chính các ngôn ngữ nhân tạo này. Tuy nhiên, phần cứng của máy tính sẽ không trực tiếp thực thi mã này. Phần cứng chỉ có thể thực thi trực tiếp một chuỗi mẫu bit được lưu trong bộ nhớ được gọi là *mã máy* hoặc *ngôn ngữ máy*. Tất cả các ngôn ngữ lập trình đều sẽ được chuyển đổi thành mã máy bằng một *trình biên dịch* hoặc được thông dịch bởi một chương trình mã máy khác được gọi là *trình thông dịch* để phần cứng thực thi các chỉ dẫn này.

Một số ngôn ngữ lập trình được sử dụng rộng rãi nhất hiện nay gồm có Python, JavaScript, C, C++, Java, C#, Swift và PHP. Mỗi ngôn ngữ lập trình này đều có những điểm mạnh và điểm yếu riêng và việc lựa chọn ngôn ngữ sẽ tùy thuộc vào mỗi dự án cũng như nhu cầu của nhà phát triển. Ví dụ:

Java là một lựa chọn phổ biến để phát triển các ứng dụng doanh nghiệp có quy mô lớn, trong khi Python thường được sử dụng để tính toán khoa học và phân tích dữ liệu.

Các nhà phát triển đã cho thấy khả năng sáng tạo đầy ấn tượng của mình trong việc thiết kế ngôn ngữ lập trình. Ban đầu, chúng chỉ là những *ngôn ngữ cấp thấp* tương tự như các chỉ dẫn trong máy tính. Các ngôn ngữ đã dần đạt đến *bậc cao*, có nghĩa là chúng có thể thể hiện các kết hợp mạnh mẽ giữa các chỉ dẫn bằng những thuật ngữ ngắn gọn. Một số ngôn ngữ sẽ phản ánh cách tư duy tự nhiên của con người trong khi vẫn duy trì được sự chặt chẽ cần thiết để hoạt động một cách chính xác.

Hiện có khoảng 400 ngôn ngữ lập trình đã được công nhận dù khá nhiều trong số đó chỉ được sử dụng trong các ứng dụng đặc thù hoặc ở trong các môi trường cũ. Mỗi ngôn ngữ đều được phát triển để giải quyết một số nhiệm vụ nhất định.

Đặc điểm, cú pháp và cấu trúc của Ngôn ngữ Lập trình

Việc lựa chọn ngôn ngữ lập trình có thể có ảnh hưởng một cách đáng kể đến hiệu suất, khả năng mở rộng và phát triển trong một dự án phần mềm. Những khía cạnh này thể hiện các yếu tố quan trọng của ngôn ngữ.

Đặc điểm của Ngôn ngữ Lập trình

Một số đặc điểm và tính chất chung của ngôn ngữ lập trình gồm có:

Tính đồng thời

Tính đồng thời biểu thị việc xử lý đồng thời nhiều tác vụ bằng cách chạy trên các bộ xử lý phần cứng khác nhau hoặc bằng cách xen kẽ việc sử dụng một bộ xử lý duy nhất của các tác vụ. Mức độ đồng thời được hỗ trợ bởi một ngôn ngữ lập trình có thể ảnh hưởng lớn đến hiệu suất và khả năng mở rộng của nó, đặc biệt là đối với các ứng dụng yêu cầu xử lý thời gian thực hoặc có lượng dữ liệu lớn. Mỗi phần công việc riêng biệt có thể được gọi là một *tiến trình* (process), *tác vụ* (task) hoặc *luồng* (thread).

Quản lý bộ nhớ

Quản lý bộ nhớ có nghĩa là phân bổ và giải phóng bộ nhớ trong một chương trình. Tùy thuộc vào ngôn ngữ hoặc môi trường thời gian chạy, việc quản lý bộ nhớ có thể được lập trình viên thực hiện thủ công hoặc được xử lý tự động. Quản lý bộ nhớ một cách phù hợp là yếu tố rất quan trọng để đảm bảo rằng chương trình có thể sử dụng bộ nhớ một cách hiệu quả và sẽ không bị hết bộ nhớ hoặc gây ra các sự cố khác. Nếu một chương trình không thể giải phóng bộ nhớ không được sử dụng, chương trình đó sẽ gây ra *rò rỉ bộ nhớ*, khiến cho mức sử dụng bộ nhớ tăng dần cho đến khi chương trình gặp sự cố hoặc có thể thấy rõ các hiệu ứng tiêu cực về hiệu suất.

Bộ nhớ chia sẻ

Bộ nhớ chia sẻ là một loại cơ chế giao tiếp giữa các tiến trình; nó cho phép nhiều tiến trình đọc và thao tác trên một vùng bộ nhớ chung. Bộ nhớ chia sẻ rất phổ biến đối với phần cứng (ví dụ như ổ đĩa) và cũng có thể là một cách hiệu quả để chia sẻ dữ liệu giữa các tiến trình. Tuy nhiên, cơ chế này đòi hỏi việc đồng bộ hóa và quản lý nghiêm ngặt để ngăn ngừa hỏng dữ liệu. Lỗi *điều kiện đua* (race condition) sẽ xảy ra nếu một tiến trình thực hiện thay đổi ngoài dự kiến đối với dữ liệu trong khi một tiến trình khác đang sử dụng dữ liệu đó.

Truyền tin

Truyền tin là một cơ chế giao tiếp giữa các tiến trình để cho phép chúng trao đổi dữ liệu và điều phối các hoạt động. Điều này thường được sử dụng trong lập trình đồng thời để đạt được giao tiếp giữa các tiến trình và có thể được thực hiện thông qua các cơ chế khác nhau như ổ nối, đường ống hoặc hàng đợi tin.

Thu gom rác

Thu gom rác là một kỹ thuật quản lý bộ nhớ tự động được một số ngôn ngữ lập trình sử dụng để lấy lại bộ nhớ không còn được sử dụng trong khi một tiến trình đang chạy. Điều này có thể giúp ngăn ngừa rò rỉ bộ nhớ và giúp các nhà phát triển viết mã một cách chính xác và hiệu quả hơn, nhưng nó cũng có thể gây ra tiêu hao hiệu năng chênh lệch và khiến việc kiểm soát hành vi chính xác của chương trình trở nên khó khăn hơn.

Kiểu dữ liệu

Kiểu dữ liệu sẽ xác định loại thông tin nào có thể được biểu diễn trong chương trình. Các kiểu dữ liệu có thể được xác định sẵn bằng ngôn ngữ hoặc do người dùng xác định và có thể bao gồm các số nguyên, số dấu phẩy động (nghĩa là xấp xỉ số thực), chuỗi, mảng và các loại khác.

Đầu vào và đầu ra (I/O)

Đầu vào và đầu ra là các cơ chế đọc và ghi dữ liệu vào và ra khỏi một chương trình. Đầu vào có thể đến từ nhiều nguồn khác nhau như nhấp chuột, đầu vào bàn phím, tệp hoặc kết nối mạng, trong khi đầu ra có thể được gửi đến nhiều đích khác nhau như màn hình, tệp hoặc kết nối mạng. I/O cho phép các chương trình tương tác với thế giới bên ngoài và trao đổi thông tin với các hệ thống khác.

Xử lý lỗi

Xử lý lỗi có nghĩa là phát hiện và phản hồi các lỗi xảy ra trong quá trình thực thi chương trình. Điều này bao gồm các lỗi như chia cho 0 và không tìm thấy tệp được yêu cầu. Xử lý lỗi cho phép các chương trình tiếp tục chạy ngay cả khi xảy ra lỗi và cải thiện độ tin cậy cũng như độ bền của chúng.

Các khái niệm được liệt kê ở trên chính là nền tảng để hiểu về cách hoạt động của ngôn ngữ lập

trình cũng như phương pháp viết mã hiệu quả và dễ bảo trì.

Cú pháp của Ngôn ngữ Lập trình

Cú pháp của ngôn ngữ lập trình đề cập đến các quy tắc viết câu lệnh và biểu thức chương trình. Điều quan trọng là cú pháp phải được xác định một cách rõ ràng và nhất quán để lập trình viên có thể viết và hiểu mã của họ một cách hiệu quả. Sau đây là các khối cơ bản của hầu hết các ngôn ngữ lập trình:

Quy trình và Hàm

Các quy trình và hàm được sử dụng để xác định các khối mã có thể tái sử dụng và có thể được gọi nhiều lần.

Biến

Biến đại diện cho các phần bộ nhớ và dữ liệu lưu trữ có thể được thao tác và truyền giữa các quy trình và hàm.

Toán tử

Toán tử là các từ khóa hoặc ký hiệu (chẳng hạn như `+` và `-`) dùng để gán giá trị cho các biến và thực hiện các phép toán số học.

Cấu trúc điều khiển

Nói chung, mã chương trình sẽ được thực thi theo thứ tự được viết, nhưng *các câu lệnh điều kiện* sẽ thay đổi luồng thực thi. Việc mã nào được thực thi tiếp theo sẽ dựa trên các điều kiện khác nhau như nội dung của bộ nhớ, trạng thái của bàn phím, các gói đến từ mạng, v.v. *Câu lệnh vòng lặp* (loop statement) là một dạng câu lệnh điều kiện đặc biệt và rất hữu ích trong việc thực hiện các thao tác tương tự nhau trên một chuỗi tập dữ liệu. Một *ngoại lệ*, hay chính là một cấu trúc điều khiển khác, sẽ gọi mã đặc biệt khi xảy ra lỗi.

Mỗi ngôn ngữ lập trình đều có thể có các cú pháp và hành vi khác nhau, và việc lựa chọn ngôn ngữ có thể có tác động lớn đến khả năng đọc và bảo trì mã.

Thư viện

Một ngôn ngữ lập trình tốt sẽ giúp người dùng dễ dàng phát triển chương trình và dễ dàng sử dụng lại mã có sẵn. Nhiều ngôn ngữ lập trình đều có cơ chế tổ chức các quy trình và hàm thành các phần có thể được tái sử dụng trong các chương trình khác.

Thư viện là nơi tập hợp các quy trình và hàm nhằm hỗ trợ một tính năng hoặc mục tiêu cụ thể và được kết hợp thành một tệp duy nhất. Việc có sẵn nhiều thư viện dễ sử dụng là một yêu cầu rất quan trọng khác đối với một ngôn ngữ lập trình tốt. Ví dụ: Python được công nhận rộng rãi là một

ngôn ngữ tốt để phát triển các chương trình liên quan đến AI vì nó có một số lượng lớn thư viện phù hợp với việc xử lý AI.

Với quy mô và độ phức tạp ngày càng tăng của các chương trình, thư viện giống những khối gạch có sẵn và đang ngày càng trở nên quan trọng hơn. Điều này đặc biệt đúng trong thế giới mã nguồn mở nơi mọi người đều có thể cảm thấy thoải mái với việc lấy mã do người khác tạo ra và sử dụng lại nó. Do đó, một hệ sinh thái thư viện đã được phát triển cho từng ngôn ngữ lập trình và các trình quản lý gói như `composer` cho PHP, `pip` cho Python và `gems` cho Ruby sẽ giúp cài đặt các thư viện một cách dễ dàng.

Thư viện cũng rất quan trọng đối với ngôn ngữ biên dịch. Việc kết hợp nhiều tệp nhị phân và thư viện đã được biên dịch sẵn để có được một tệp thực thi duy nhất được gọi là *liên kết* (linking) và công cụ thực hiện thao tác này được gọi là *trình liên kết* (linker). Có hai loại liên kết: *liên kết tĩnh* (static linking) nơi chỉ có mã thư viện cần thiết được đưa vào tệp thực thi của ứng dụng cuối cùng và *liên kết động* (dynamic linking) nơi một thư viện được cài đặt trong hệ thống sẽ được chia sẻ bởi tất cả các ứng dụng sử dụng thư viện đó. Hiện tại, liên kết động là phương pháp được ưa chuộng hơn và đặc trưng bởi các tệp thực thi ứng dụng nhỏ hơn và sử dụng ít bộ nhớ hơn khi chạy.

Hãy lưu ý rằng vì cùng một thư viện có thể được sử dụng bởi nhiều chương trình nên sự khác biệt giữa các phiên bản của một thư viện có thể còn là một vấn đề lớn hơn so với các ứng dụng. Ngoài lề một chút, chúng ta sẽ phải ghi nhớ cách xem số phiên bản. *Phân phiên bản ngữ nghĩa* (Semantic versioning) được sử dụng phổ biến và biểu thị các phiên bản bằng ba số được phân tách bằng dấu chấm. Một phiên bản điển hình có thể là `2.39.16` cho biết phiên bản chính là 2 (một con số có thể chỉ thay đổi vài năm một lần), phiên bản phụ là 39 nằm trong phiên bản chính (có thể cập nhật vài tháng một lần để chứa các thay đổi quan trọng về tính năng) và bản sửa đổi nhanh là 16 (có thể thay đổi do sửa một lỗi duy nhất). Các phiên bản và sửa đổi càng về sau sẽ có số càng cao hơn.

Một ví dụ rất đơn giản

Chúng ta hãy cùng xem một ví dụ *rất* đơn giản về một chương trình máy tính sử dụng ngôn ngữ Python để có thể mường tượng sơ bộ về một số yếu tố đã được đề cập tới.

Được trình bày bằng ngôn ngữ tự nhiên, chương trình phải thực hiện những việc sau: "Yêu cầu người dùng nhập một số và kiểm tra xem số này là chẵn (EVEN) hay lẻ (ODD). Cuối cùng là xuất kết quả."

Và đây là đoạn mã chúng ta có thể lưu trong tệp `simpleprogram.py`:

```
num = int(input("Enter a number: "))
if (num % 2) == 0:
   print("The given number is EVEN.")
```

```
else:
   print("The given number is ODD.")
```

Chỉ trong vài dòng mã này, chúng ta đã có thể tìm thấy nhiều đặc điểm và thành phần cú pháp đã được đề cập tới ở trên:

1. Trong dòng 1, chúng ta đã đặt *biến* `num` và gán cho nó một *giá trị* với *toán tử* `=`.
2. Giá trị được gán tương ứng với *đầu vào* của người dùng (thông qua `input()` *function*). Ngoài ra, hàm `int()` sẽ đảm bảo rằng đầu vào này sẽ được chuyển đổi thành *kiểu dữ liệu* số nguyên nếu có thể. Biểu thức được truyền cho hàm trong dấu ngoặc đơn được gọi là *tham số* (parameter) hoặc *đối số* (argument).
3. Nếu người dùng nhập một chuỗi ký tự, Python sẽ in ra lỗi. Việc in ra lỗi này là một phần của quá trình *xử lý lỗi*. Nếu một số thập phân được nhập vào, hàm `int()` sẽ chuyển đổi nó thành số cơ sở (ví dụ: `5,73` đến `5`).
4. Trong các dòng tiếp theo, *cấu trúc điều khiển* nêu ra một *điều kiện* với từ khóa `if` và `else` sẽ kiểm soát những gì sẽ xảy ra trong mỗi trường hợp (số chẵn hoặc số lẻ).
5. Đầu tiên, toán tử modulo `%` kiểm tra xem khi (`if`) chia số đã nhập cho 2 có cho ra giá trị 0 (tức là không có số dư) hay không — trong trường hợp này, số đó là số chẵn. Ký tự `==` được nhân đôi là toán tử so sánh "bằng" khác với toán tử gán `=` ở dòng 1.
6. Trong trường hợp khác (`else`), tức là khi chia cho 2 kết quả không bằng 0 thì số được nhập là số lẻ.
7. Trong cả hai trường hợp, hàm `print()` đều sẽ trả về kết quả dưới dạng *đầu ra* bằng văn bản.

Và đây là kết quả khi chúng ta chạy chương trình trên dòng lệnh:

```
$ python simpleprogram.py
Enter a number: 5
The given number is ODD.
```

Khi xem xét mức độ logic ngôn ngữ đã được sử dụng trong ví dụ đơn giản này, chúng ta sẽ biết được phần mềm phức tạp được phân phối trên hàng nghìn tệp có khả năng thực hiện những gì. Ví dụ: *hệ điều hành* (như Microsoft Windows, macOS hoặc Linux) sẽ khiến tất cả phần cứng của máy tính trở nên khả dụng, đồng thời đảm bảo rằng người dùng có thể cài đặt tất cả các ứng dụng khác mà họ mong muốn và sử dụng chúng cho công việc hoặc mục đích giải trí.

Mã máy, Hợp ngữ và Trình biên dịch mã

Như đã đề cập trước đây, phần cứng chỉ có thể thực thi trực tiếp một loạt mẫu bit được gọi là mã

máy. CPU sẽ đọc mẫu bit từ bộ nhớ theo đơn vị của một từ (8 đến 64 bit) và thực hiện lệnh tương ứng. Các lệnh riêng lẻ đều khá là đơn giản, ví dụ: sao chép nội dung của vị trí bộ nhớ A sang vị trí bộ nhớ B, nhân nội dung của vị trí bộ nhớ C với nội dung của vị trí bộ nhớ D hoặc đọc dữ liệu đã đến thiết bị tại địa chỉ X. Trong thời đại CPU 8 bit, một số người có thể ghi nhớ tất cả các mẫu bit được sử dụng trong mã máy và viết chương trình một cách trực tiếp. Ngày nay, số lượng mẫu chỉ dẫn đã tăng lên theo cấp độ lớn và việc cố gắng ghi nhớ tất cả các mẫu này là không thực tế.

Mã máy là một chuỗi các mẫu bit, có thể là 0 và 1, hoàn toàn không hề trực quan đối với con người. Để việc lập trình có thể trở nên trực quan hơn, *hợp ngữ* (assembly language) đã được tạo ra, trong đó các chỉ dẫn sẽ được đặt tên và có thể được chỉ định bằng chuỗi. Trong hợp ngữ, các chỉ dẫn tương ứng 1-1 với mã máy sẽ được viết từng lệnh một. Các chỉ dẫn có thể sẽ giống như sau:

```
move    [B], [A]           copy the contents of memory A to memory B
multi   R1, [C], [D]    multiply the contents of memory C by the contents of memory D
input   R1, [X]         read the data that has arrived at the device at address X
```

Một chỉ dẫn trong hợp ngữ sẽ tương ứng với một lệnh trong mã máy; lệnh này sẽ tương ứng với lệnh chính xác mà phần cứng có thể hiểu và thực hiện. Ưu điểm của hợp ngữ so với ngôn ngữ máy gồm có:

Cải thiện khả năng đọc và bảo trì

Hợp ngữ dễ đọc và viết hơn nhiều so với mã máy. Điều này sẽ giúp các lập trình viên dễ dàng hiểu, gỡ lỗi và duy trì mã của họ hơn.

Tự động hóa việc tính toán địa chỉ

Việc lập trình mã máy cũng có thể sử dụng khái niệm biến và hàm, nhưng mọi thứ phải được thể hiện dưới dạng *địa chỉ bộ nhớ*. Hợp ngữ cũng sẽ gán tên cho các địa chỉ bộ nhớ để giúp việc diễn đạt logic của chương trình trở nên dễ dàng hơn.

Vì hợp ngữ có quyền truy cập vào tất cả chức năng của phần cứng nên nó thường được sử dụng trong các tình huống sau:

Phần phụ thuộc kiến trúc của hệ điều hành

Việc sử dụng các chỉ dẫn chuyên dụng dành riêng cho một kiến trúc CPU (để truy cập vào các tính năng khởi tạo và bảo mật) chỉ có thể được thực hiện bằng hợp ngữ.

Phát triển các thành phần hệ thống cấp thấp

Hợp ngữ được sử dụng để phát triển các thành phần hệ thống cần tương tác trực tiếp với phần cứng của máy tính như trình điều khiển thiết bị, chương trình cơ sở (firmware) và hệ thống đầu vào/đầu ra cơ bản (BIOS). Đặc biệt, các thiết bị tốc độ cao yêu cầu đẩy hiệu suất phần cứng

đến giới hạn thường cần tới các trình điều khiển và chương trình cơ sở được lập trình bằng hợp ngữ.

Lập trình bộ vi điều khiển

Hợp ngữ cũng được sử dụng để lập trình bộ vi điều khiển - tức những máy tính nhỏ có công suất thấp được sử dụng trong nhiều hệ thống nhúng từ đồ chơi đến điều khiển công nghiệp. Một số bộ vi điều khiển chỉ có dung lượng bộ nhớ là vài trăm byte và thường được lập trình bằng hợp ngữ.

Mã hợp ngữ sẽ được chuyển đổi thành mã máy trước khi được thực thi bởi một ứng dụng có tên *trình biên dịch mã* (assembler). Trình biên dịch mã là công cụ lập trình lâu đời nhất và đã mang lại một số lợi thế không thể tưởng tượng được trong lập trình mã máy. Vì dễ gây nhầm lẫn nên đôi khi người ta conf gọi hợp ngữ là trình biên dịch mã.

Mã máy và hợp ngữ khác nhau ở các bộ xử lý phần cứng. Những ngôn ngữ này được gọi là “ngôn ngữ cấp thấp” vì chúng hoạt động trực tiếp trên phần cứng. Tuy nhiên, các khái niệm tính toán và đầu vào/đầu ra đều giống nhau trên tất cả các bộ xử lý. Nếu các khái niệm chung có thể được diễn đạt theo cách dễ hiểu hơn đối với con người thì hiệu quả lập trình có thể được cải thiện đáng kể. Đây là nơi "ngôn ngữ bậc cao" xuất hiện.

Ngôn ngữ biên dịch

Ngôn ngữ biên dịch là ngôn ngữ lập trình được dịch sang mã máy hoặc sang định dạng trung gian gọi là *bytecode*. Bytecode sẽ được thực thi trên máy tính đích bởi *máy ảo thời gian chạy*. Máy ảo sẽ dịch mã byte thành mã máy thích hợp cho mỗi máy tính. Bytecode cho phép các chương trình không phụ thuộc vào nền tảng và có thể chạy trên bất kỳ hệ thống nào có máy ảo tương thích.

Việc dịch mã nguồn được viết bằng ngôn ngữ lập trình bậc cao sang mã máy hoặc mã byte sẽ được thực hiện bởi *trình biên dịch*. Ví dụ về các ngôn ngữ được biên dịch tạo ra mã máy trực tiếp chính là C và C++. Các ngôn ngữ tạo ra bytecode gồm có Java và C#. Việc lựa chọn giữa mã máy và bytecode phụ thuộc vào các yêu cầu của dự án như hiệu suất, tính bất khả tri về nền tảng (không bị ràng buộc trên một nền tàng cụ thể) và tính dễ phát triển.

Ngôn ngữ thông dịch

Ngôn ngữ thông dịch là các ngôn ngữ lập trình được thực thi bởi một *trình thông dịch* thay vì được biên dịch thành mã máy. Trình thông dịch sẽ đọc mã nguồn và thực thi các câu lệnh có trong đó. Trình thông dịch có thể xử lý trực tiếp mã nguồn mà không cần chuyển đổi nó sang một định dạng tệp khác. Do đó, không giống như trình biên dịch là dịch toàn bộ chương trình thành mã máy trước khi thực thi, trình thông dịch sẽ đọc từng dòng mã và thực thi nó ngay lập tức. Việc này cho phép lập trình viên có thể xem được kết quả của từng dòng khi chúng được thực thi.

Ngôn ngữ thông dịch thường được sử dụng cho *tệp lệnh* - tức là các chương trình ngắn tự động hóa các tác vụ - dành cho giao diện dòng lệnh và để kiểm soát công việc và các tác vụ hàng loạt. Các tệp lệnh được viết bằng ngôn ngữ thông dịch có thể dễ dàng được sửa đổi và thực thi mà không cần qua biên dịch lại. Điều này khiến chúng phù hợp với các tác vụ đòi hỏi tạo mẫu nhanh hoặc lặp lại linh hoạt và nhanh chóng. Sự tiện lợi này cũng đi kèm với một số nhược điểm tiềm ẩn. Ví dụ: một chương trình được thông dịch sẽ chạy chậm hơn một chương trình được biên dịch tương đương.

Một số ví dụ về các ngôn ngữ thông dịch chính là Python, Ruby và JavaScript. Python được sử dụng rộng rãi trong tính toán khoa học, phân tích dữ liệu và máy học tự động (machine learning), trong khi Ruby thường được sử dụng trong phát triển trang web và tạo các tệp lệnh tự động hóa. JavaScript là một ngôn ngữ tệp lệnh phía máy khách được nhúng trong trình duyệt web để tạo các trang web động và tương tác.

Ngôn ngữ hướng Dữ liệu

Ngôn ngữ hướng dữ liệu là ngôn ngữ lập trình được tối ưu hóa để xử lý và thao tác với các số lượng dữ liệu lớn. Chúng được thiết kế để có thể xử lý hiệu quả các tập hợp dữ liệu lớn có cấu trúc hoặc không cấu trúc cũng như cung cấp một bộ công cụ để làm việc với cơ sở dữ liệu, cấu trúc dữ liệu và thuật toán để xử lý và phân tích dữ liệu.

Ngôn ngữ hướng dữ liệu được ứng dụng trong nhiều khía cạnh như khoa học dữ liệu, phân tích dữ liệu lớn, máy học tự động và lập trình cơ sở dữ liệu. Chúng rất phù hợp cho các nhiệm vụ liên quan đến xử lý và phân tích các số lượng dữ liệu lớn như làm sạch và chuyển đổi dữ liệu, trực quan hóa dữ liệu và lập mô hình thống kê.

Một số ví dụ về các ngôn ngữ hướng dữ liệu chính là SQL (viết tắt của Structured Query Language - Ngôn ngữ truy vấn có cấu trúc), R và MATLAB. SQL là ngôn ngữ tiêu chuẩn được sử dụng để quản lý cơ sở dữ liệu quan hệ và được sử dụng rộng rãi trong kinh doanh và công nghiệp. R là một ngôn ngữ lập trình và môi trường dành cho cho tính toán và đồ hoạ thống kê, đồng thời cũng được sử dụng rộng rãi trong khoa học dữ liệu và máy học tự động. MATLAB là một môi trường tính toán số và ngôn ngữ lập trình được ứng dụng trong nhiều tác vụ như xử lý tín hiệu, xử lý hình ảnh và tài chính tính toán.

Mô hình Lập trình

Bên cạnh các đặc điểm cụ thể của ngôn ngữ lập trình, *mô hình lập trình* (programming paradigm) sẽ xác định cách tiếp cận giải pháp cụ thể. Chúng ta có thể coi mô hình như một chiến lược cơ bản để tiếp cận một nhiệm vụ tùy thuộc vào các yêu cầu và điều kiện cụ thể.

Một ví dụ để dễ hình dung chính là việc xây dựng một ngôi nhà: việc thợ xây xây tường bằng gạch

hay bằng các khối bê tông đúc sẵn được lắp ráp tại chỗ là một quyết định cơ bản tùy thuộc vào yêu cầu và hoàn cảnh. Chúng ta muốn ngôi nhà có những đặc điểm gì? Nó nằm ở đâu? Nó có được kết nối với những ngôi nhà khác hay không?

Theo một cách tương tự, các mô hình sẽ đặt ra phương hướng lập trình: ví dụ như một dự án phần mềm liệu có thể được chia thành các phần nhỏ riêng biệt hay không và bằng cách nào. Mỗi ngôn ngữ lập trình sẽ phù hợp nhất với một số mô hình cụ thể. Vì vậy, việc lựa chọn mô hình có liên quan chặt chẽ đến việc lựa chọn ngôn ngữ lập trình.

Các mô hình sau đây rất phổ biến trong lập trình:

Lập trình hướng đối tượng (Object-oriented programming - OOP)

OOP được dựa trên khái niệm về *đối tượng* - tức là các thành phần của *hạng* (class) bao hàm cả dữ liệu và hành vi. Ví dụ: một ngôn ngữ có thể cung cấp một hình chữ nhật như một hạng để giúp lập trình viên hiển thị hộp khung trên màn hình.

OOP tập trung vào thao tác dữ liệu ở cấp độ đối tượng. OOP sẽ giúp việc viết mã trở nên dễ bảo trì, tái sử dụng và mở rộng hơn và được sử dụng rộng rãi trong các phần mềm máy tính để bàn, trò chơi video và ứng dụng web. Java, C# và Python chính là các ví dụ về ngôn ngữ lập trình hướng đối tượng.

Lập trình thủ tục (Procedural programming)

Lập trình thủ tục sẽ thực hiện tác vụ thông qua các *quy trình* hoặc các khối mã có thể được thực thi theo một thứ tự cụ thể. Điều này sẽ giúp chúng ta dễ dàng viết mã có cấu trúc và dễ theo dõi, nhưng nó cũng có thể dẫn đến việc mã trở nên kém linh hoạt và khó bảo trì hơn khi quy mô và độ phức tạp của dự án tăng lên. C, Pascal và Fortran là các ví dụ về các ngôn ngữ lập trình thủ tục.

Ngày nay, các cách tiếp cận khác nhằm phát triển phần mềm đang được sử dụng; một số ngôn ngữ nhất định sẽ phù hợp hơn với các cách tiếp cận này. Ngoài ra, giao diện kéo và thả sẽ cho phép những người không phải lập trình viên có thể viết chương trình và gần đây nhiều dịch vụ trực tuyến đã bắt đầu có khả năng tạo ra mã với trí tuệ nhân tạo thông qua các chỉ dẫn bằng ngôn ngữ đơn giản.

Tóm lại, mỗi một mô hình lập trình đều có điểm mạnh và điểm yếu riêng và việc lựa chọn mô hình thường phụ thuộc vào nhu cầu của dự án, kinh nghiệm và sở thích của nhà phát triển cũng như các hạn chế của nền tảng và môi trường phát triển. Việc hiểu về các loại mô hình khác nhau sẽ giúp chúng ta chọn ra được các mô hình phù hợp với nhu cầu của mình và cũng có thể giúp chúng ta viết mã một cách hiệu quả hơn.

Bài tập Hướng dẫn

1. Mục đích của hàm là gì?

2. Ưu điểm của bytecode so với tệp mã máy là gì?

3. Ưu điểm của tệp mã máy so với bytecode là gì?

Bài tập Mở rộng

1. Một số nhược điểm của việc chia chương trình thành một số lượng lớn các tiến trình hoặc tác vụ là gì?

2. Bạn đã tìm thấy một số gói mã nguồn mở được cung cấp ở các phiên bản khác nhau. Các gói này cung cấp các tính năng mà bạn cần cho chương trình của mình. Các tiêu chí để lựa chọn một gói là gì?

3. Ngoài OOP và các mô hình phát triển thủ tục, có những phương pháp phát triển phần mềm nào khác và ngôn ngữ lập trình nào hỗ trợ tốt nhất cho từng phương pháp?

Tóm tắt

Trong bài học này, chúng ta đã tìm hiểu phần mềm là gì và nó được phát triển như thế nào với sự trợ giúp của các ngôn ngữ lập trình. Các ngôn ngữ lập trình khác nhau không chỉ về cú pháp mà còn về cách quản lý tài nguyên phần cứng hoặc cách xử lý cấu trúc dữ liệu.

Các ngôn ngữ lập trình cũng khác nhau về cách mã nguồn (con người có thể đọc được) sẽ được trình thông dịch hoặc trình biên dịch chuyển đổi thành mã máy cuối cùng để máy tính xử lý như thế nào.

Mô hình lập trình xác định chiến lược của các dự án phần mềm và do đó cũng sẽ quyết định việc lựa chọn ngôn ngữ lập trình phù hợp tùy thuộc vào yêu cầu và quy mô của dự án tương ứng.

Đáp án Bài tập Hướng dẫn

1. Mục đích của hàm là gì?

   Hàm gói gọn một số hoạt động phổ biến nhất định, chẳng hạn như việc xuất ra một chuỗi. Bằng việc tạo một hàm, chúng ta có thể cho phép chương trình của mình và các chương trình khác thực hiện hàm đó một cách thuận tiện và lặp đi lặp lại mà không cần phải viết mã riêng cho nó.

2. Ưu điểm của bytecode so với tệp mã máy là gì?

   Tệp bytecode có thể chạy trên nhiều máy tính khác nhau nơi máy ảo sẽ biến mã thành mã máy. Ví dụ: JavaScript chạy trong nhiều trình duyệt trên nhiều loại máy tính.

3. Ưu điểm của tệp mã máy so với bytecode là gì?

   Mã máy sẽ chạy một cách nhanh nhất có thể. Bytecode sẽ chạy chậm hơn vì máy ảo phải biến nó thành mã máy khi chạy bytecode.

Đáp án Bài tập Mở rộng

1. Một số nhược điểm của việc chia chương trình thành một số lượng lớn các tiến trình hoặc tác vụ là gì?

   Khi một chương trình được chia thành các tiến trình, chúng sẽ phải giao tiếp với nhau. Nếu chúng làm việc chung trên nhiều dữ liệu thì các tiến trình có thể tiêu tốn chênh lệch trong việc trao đổi dữ liệu và bảo vệ dữ liệu khỏi nhiều thay đổi đồng thời (điều kiện đua). Các tiến trình cũng sẽ phải chịu mức chênh lệch khi chúng khởi động và kết thúc. Càng có nhiều tiến trình thì chương trình và các tương tác của nó càng trở nên phức tạp hơn, do đó khó có thể tìm ra lỗi hơn.

   Mô hình hàm có xu hướng giúp việc chia chương trình thành nhiều tiến trình trở nên dễ dàng hơn vì tính bất biến. Dữ liệu bất biến sẽ không bị ảnh hưởng bởi các điều kiện đua.

2. Bạn đã tìm thấy một số gói mã nguồn mở được cung cấp ở các phiên bản khác nhau. Các gói này cung cấp các tính năng mà bạn cần cho chương trình của mình. Các tiêu chí để lựa chọn một gói là gì?

   Kiểm tra các báo cáo lỗi và tư vấn bảo mật cho các gói vì một số gói sẽ có rất nhiều lỗi và thậm chí là không an toàn. Đôi khi phiên bản mới nhất không phải là phiên bản tốt nhất vì có thể nó đã có một lỗ hổng bảo mật mới nào đó.

   Hãy xem qua diễn đàn nơi các nhà phát triển nói về gói để biết chắc rằng nó vẫn đang được duy trì. Chương trình của bạn có thể sẽ cần được sử dụng trong một thời gian dài và bạn cũng sẽ muốn gói này luôn vững chắc và sẵn sàng theo thời gian.

   Hãy thử các gói khác nhau để kiểm tra hiệu suất cũng như tính chính xác của chúng.

   Hầu hết các gói đều phụ thuộc vào các hàm có trong các gói khác (*phần phụ thuộc*); do đó, điểm yếu ở một trong các gói phụ thuộc có thể sẽ ảnh hưởng đến chương trình của bạn.

3. Ngoài OOP và các mô hình phát triển thủ tục, có những phương pháp phát triển phần mềm nào khác và ngôn ngữ lập trình nào hỗ trợ tốt nhất cho từng phương pháp?

   Ngoài OOP và các mô hình phát triển thủ tục, chúng ta còn có các loại khác như sau:

   Lập trình hàm nhấn mạnh việc sử dụng các hàm và khái niệm toán học (như lambda và toán tử bao đóng) để viết mã dựa trên việc đánh giá các biểu thức thay vì thực thi các câu lệnh. Lập trình hàm xử lý các hàm như những công dân hạng nhất — để chương trình có thể thao tác chúng — và nhấn mạnh *tính bất biến*, hoặc làm việc với các biến không thể được thay đổi sau thiết lập ban đầu. Điều này giúp việc suy luận và kiểm tra mã cũng như viết các ứng dụng đồng

thời và song song trở nên dễ dàng hơn. Erlang, Haskell, Lisp và Scheme là các ví dụ về ngôn ngữ lập trình hàm.

Các ngôn ngữ mệnh lệnh tập trung vào các câu lệnh cần thiết để kiểm soát luồng chuyển đổi của chương trình từ và sang các trạng thái khác nhau.

Ngôn ngữ khai báo mô tả những hành động cần thực hiện và logic đằng sau các chỉ dẫn. Thứ tự thực hiện các chỉ dẫn sẽ không được chỉ định. Các chỉ dẫn, lệnh gọi hàm và các câu lệnh khác này có thể được trình biên dịch sắp xếp lại và tối ưu hóa, miễn là chúng vẫn duy trì logic cơ bản.

Lập trình tự nhiên là một mô hình lập trình sử dụng ngôn ngữ tự nhiên hoặc các cách biểu diễn thân thiện với con người khác để mô tả hành vi cần có của chương trình. Ý tưởng ở đây là làm cho việc lập trình có thể tiếp cận được với những người không được đào tạo chính quy về khoa học máy tính. Scratch và Alic là các ví dụ về ngôn ngữ lập trình tự nhiên.

051.2 Kiến trúc của Phần mề

Tham khảo các mục tiêu LPI

Open Source Essentials version 1.0, Exam 050, Objective 051.2

Khối lượng

2

Các lĩnh vực kiến thức chính

- Hiểu rõ các khái niệm về máy khách và máy chủ
- Hiểu rõ các khái niệm về máy khách cấu hình tối thiểu và máy khách cấu hình tối đa
- Hiểu rõ các khái niệm về nguyên khối và các dịch vụ vi mô cũng như những sự khác biệt chính giữa chúng
- Hiểu rõ các khái niệm về Giao diện Lập trình Ứng dụng (API)
- Hiểu rõ khái niệm về các thành phần phần mềm và sự tích hợp hoặc tách biệt của chúng (dịch vụ, mô-đun, API)

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Máy khách và máy chủ
- Máy khách cấu hình tối thiểu và máy khách cấu hình tối đa
- Ứng dụng web
- Ứng dụng một trang
- Kiến trúc nguyên khối
- Kiến trúc dịch vụ vi mô
- Giao diện Lập trình Ứng dụng (API)
- API RESTful

Linux Professional Institute

Bài 1

Chứng chỉ:	Open Source Essentials
Phiên bản:	1.0
Chủ đề:	051 Những nguyên tắc cơ bản về Phần mềm
Mục tiêu:	051.1 Kiến trúc của Phần mềm
Bài học:	1 trên 1

Giới thiệu

Internet cũng như các ứng dụng di động và dựa trên web đã có mặt ở khắp nơi trong thế giới hiện đại của chúng ta. Những công cụ này được phần lớn dân số thế giới sử dụng một cách liền mạch và hỗ trợ mọi hoạt động từ nhắn tin tức thời đến các hoạt động phức tạp hơn như mua thiết bị khai thác cho các công ty lớn.

Đằng sau tất cả các giao diện và dịch vụ trực tuyến trông có vẻ đơn giản này là cả một *kiến trúc* — hay một công trình sắp đặt các thành phần liên kết của phần mềm — mà chúng ta thường không quá coi trọng. Chúng ta cần tìm hiểu một chút về kiến trúc này để có thể hiểu được cách tất cả các thành phần của Internet khớp với nhau và cách phần mềm có thể mang lại giá trị thực sự cho người dùng.

Trong bài học này, chúng ta sẽ xem xét một số kiến trúc phần mềm đứng đằng sau các ứng dụng web — chính là các hệ thống phần mềm dựa trên máy chủ — và cách chúng được sử dụng trong

các hệ thống rất quen thuộc với hầu hết mọi người dùng.

Máy chủ và Máy khách

Tại một thời điểm nào đó khi sử dụng các hệ thống trực tuyến, rất có thể người dùng nào cũng đều đã gặp phải thông báo giống như Màn hình mẫu hiển thị khi phản hồi đang được tiến hành.

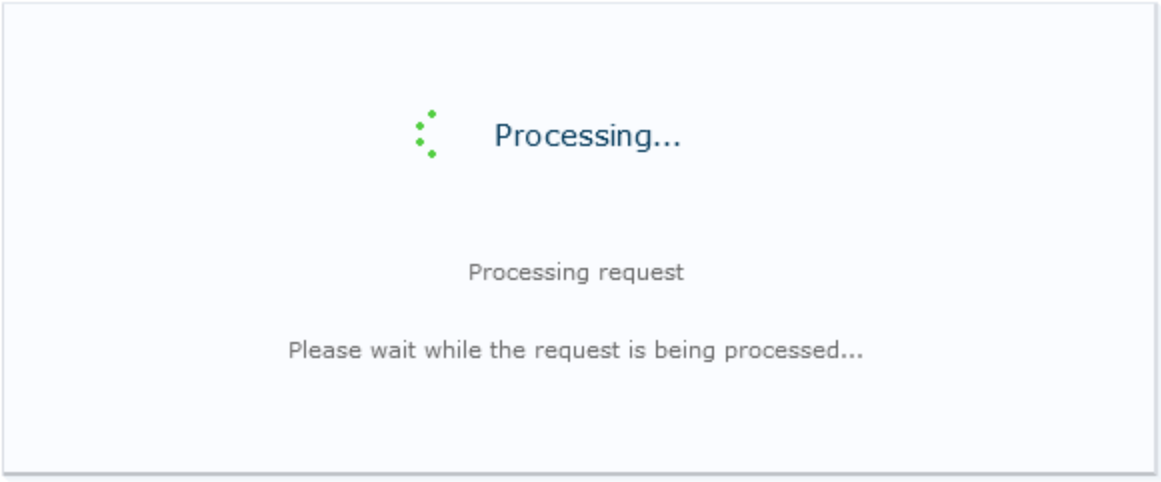

Figure 1. Màn hình mẫu hiển thị khi phản hồi đang được tiến hành

Hãy cùng nhìn lại và xem xét bối cảnh nơi chúng ta đã nhìn thấy thông báo này. Giả sử khi chúng ta đang cố gắng truy cập vào tài khoản ngân hàng của mình thông qua trang web ngân hàng. Khi truy cập trang web ngân hàng trên máy tính xách tay của mình, chúng ta sẽ sử dụng một loại phần mềm (tức là ứng dụng) được gọi là trình duyệt web (chẳng hạn như Google Chrome hay Firefox). Trong trường hợp này, trình duyệt trên máy tính của chúng ta sẽ gửi yêu cầu đến một máy tính khác lưu trữ trang web này. Loại máy tính đặc biệt này được gọi là *máy chủ*. Nó được thiết kế đặc biệt để hoạt động 24/7 nhằm phục vụ các yêu cầu mới đến từ khắp mọi nơi trên thế giới.

Như vậy có nghĩa là máy chủ chính là một máy tính giống như máy tính chúng ta sử dụng để làm việc, chơi trò chơi điện tử và thực hiện các bài tập lập trình. Tuy nhiên, có một điểm khác biệt chính là máy chủ thường sử dụng tất cả tài nguyên của nó cho ứng dụng phần mềm chạy trên nó. Trong ví dụ này, phần mềm là một ứng dụng web - một chương trình máy tính chạy trên máy chủ.

Trong Màn hình mẫu hiển thị khi phản hồi đang được tiến hành, máy chủ đã nhận được yêu cầu từ trình duyệt và ứng dụng đang chạy trên máy chủ đang xử lý thao tác kết quả. Hoạt động này có

thể truy vấn một cơ sở dữ liệu để lấy thông tin chi tiết của người dùng trong ngân hàng hoặc liên lạc với một máy chủ khác để xác minh mức chiết khấu đặc biệt cho khoản vay tiếp theo của người dùng.

Trong ví dụ này, chúng ta sẽ gọi trình duyệt đang chạy trên máy của chúng ta là *ứng dụng khách*, hay nói một cách đơn giản là *máy khách*. Máy khách sẽ tương tác với máy chủ từ xa.

Giao tiếp mạng giữa máy khách và máy chủ có thể diễn ra bên trong một mạng doanh nghiệp hoặc trên mạng toàn cầu mà chúng ta gọi là internet. Đặc điểm chung của tương tác giữa máy khách và máy chủ là máy chủ có thể thiết lập nhiều mối quan hệ với nhiều máy khách. Hãy nghĩ về ví dụ trước: trang web của một ngân hàng được lưu trữ trên máy chủ có thể đáp ứng hàng nghìn yêu cầu mỗi phút từ nhiều điểm, mỗi điểm đều có người dùng đang cố gắng truy cập vào tài khoản ngân hàng cá nhân của họ.

Không phải trường hợp nào cũng đều sẽ có cấu trúc giống như một trình duyệt tương tác với một máy chủ thực hiện hầu hết mọi tiến trình xử lý. Trong một số trường hợp, máy khách có thể sẽ là thành phần xử lý chính; khái niệm này được gọi là *máy khách cấu hình tối đa* (*fat client* hay *thick client*). Trong đó, máy khách sẽ lưu trữ và xử lý phần lớn các tác vụ thay vì dựa vào tài nguyên của máy chủ. Ngược lại, trong ví dụ về ngân hàng, trình duyệt lại là một *máy khách cấu hình tối thiểu* (*thin client*) sẽ dựa vào máy chủ để tính toán và trả về thông tin qua mạng.

Một ví dụ về máy khách cấu hình tối đa là ứng dụng trò chơi điện tử trên máy tính để bàn. Trong đó, phần lớn việc lưu trữ và xử lý dữ liệu sẽ được thực hiện cục bộ sử dụng GPU, RAM, CPU và dung lượng ổ đĩa của máy tính để xử lý thông tin. Ứng dụng như vậy hiếm khi dựa vào máy chủ bên ngoài, đặc biệt nếu đó là trò chơi ngoại tuyến.

Cả hai cách tiếp cận đều có ưu và nhược điểm: Đối với máy khách cấu hình tối đa, tính ổn định của mạng không phải là vấn đề so với máy khách cấu hình tối thiểu dựa trên máy chủ từ xa, nhưng việc cập nhật phần mềm có thể khó áp dụng hơn và máy khách cấu hình tối đa sẽ yêu cầu nhiều tài nguyên máy tính hơn. Đối với một máy khách cấu hình tối thiểu, tiêu hao thấp hơn chính là một lợi thế lớn. Đối với cả hai loại máy khách, việc cung cấp dữ liệu cá nhân cho ứng dụng của bên thứ ba có thể sẽ trở thành vấn đề.

Ứng dụng Web

Ứng dụng web là phần mềm chạy trên máy chủ, xử lý các tương tác của người dùng và được liên hệ với máy khách cấu hình tối đa thông qua một mạng máy tính. Không phải tất cả các trang web đều được coi là ứng dụng web: các trang web tĩnh đơn giản không có tính tương tác không được coi là ứng dụng web vì máy chủ không chạy ứng dụng để xử lý các hành động do máy khách yêu cầu.

Có nhiều ứng dụng web có thể được chia thành hai nhóm: *ứng dụng một trang* (SPA) và *ứng dụng đa trang* (MPA). SPA chỉ có một trang web nơi tất cả việc trao đổi và tải dữ liệu diễn ra mà không cần chuyển hướng người dùng đến một trang web khác bên trong ứng dụng. Một MPA, trái ngược với SPA, sẽ có nhiều trang web. Một phiên thay đổi dữ liệu có thể làm mới cùng một trang web bắt nguồn của hành động hoặc chuyển hướng người dùng đến một trang web khác.

Xét trong ví dụ trước khi người dùng muốn kiểm tra các giao dịch tài khoản gần đây nhất của họ trên trang web ngân hàng trực tuyến. Hãy tưởng tượng rằng một giao dịch sẽ xảy ra sau khi trang web được tải. Nếu ứng dụng web ngân hàng là SPA, giao dịch mới sẽ tự động được hiển thị trên cùng một trang web mà không chuyển hướng người dùng đến một trang mới. Nếu người dùng muốn kiểm tra khoản vay của mình, thông tin mới cũng sẽ được hiển thị trên cùng một trang để tránh việc phải chuyển hướng người dùng đến một trang web mới. Việc thay đổi trang mà không chuyển hướng người dùng sẽ giúp việc điều hướng trở nên mượt mà hơn.

Đối với MPA, khi người dùng yêu cầu trang web về khoản vay, máy chủ sẽ phải chuyển hướng người dùng đến một vị trí mới - tức là một trang web khác.

Một ví dụ rất nổi tiếng về ứng dụng web SPA chính là Google Mail (Gmail). Nó sẽ không chuyển hướng người dùng đến các trang hoàn toàn mới (ví dụ khi người dùng muốn hiển thị thư mục thư rác); ứng dụng sẽ chỉ đơn giản là làm mới các khu vực cụ thể của màn hình hiển thị nơi hiển thị tất cả các tin nhắn rác trong khi vẫn ở lại trên một trang web đó.

Trái lại, một ứng dụng MPA nổi tiếng chính là Amazon.com - một gã khổng lồ trong lĩnh vực thương mại điện tử - với mỗi một mặt hàng đều được đặt trên một trang web riêng biệt.

Một lợi thế của MPA so với SPA là dễ dàng thu thập và đo lường dữ liệu phân tích trang web. Điều này rất quan trọng trong việc giúp các nhà phát triển tối ưu hóa các kết quả tìm kiếm trên internet.

Thông thường, một ứng dụng web sẽ được chia thành hai phần riêng biệt: *giao diện người dùng* (frontend) và *giao diện máy chủ* (backend). Giao diện người dùng là tầng xem nơi người dùng tương tác với trình duyệt bằng cách sử dụng các thành phần trang với các thao tác nhấp, chọn hoặc nhập. Đây là nơi dữ liệu từ máy chủ sẽ được nhận, định dạng và hiển thị cho người dùng thông qua trình duyệt.

Giao diện máy chủ thường chiếm phần lớn hơn trong ứng dụng web. Nó bao gồm logic nghiệp vụ, bộ xử lý giao tiếp, phần lớn việc xử lý dữ liệu và lưu trữ dữ liệu. Lưu trữ dữ liệu sẽ sử dụng một hệ thống quản lý cơ sở dữ liệu riêng biệt được kết nối với giao diện máy chủ.

Giao diện người dùng và giao diện máy chủ sẽ giao tiếp với nhau. Các yêu cầu dữ liệu sẽ được giao diện người dùng chuyển tiếp đến giao diện máy chủ và dữ liệu do máy chủ trả về sẽ được giao diện người dùng nhận, định dạng và hiển thị cho người dùng.

Trong ví dụ đơn giản của chúng ta về việc tìm nạp giao dịch mới nhất trong tài khoản ngân hàng của người dùng, hành động này được diễn giải bởi giao diện người dùng của ứng dụng đang chạy trong trình duyệt trên màn hình của người dùng. Yêu cầu này sau đó được gửi qua internet đến máy chủ của ứng dụng, xác thực xem người dùng có được phép thực hiện hành động hay không, tìm nạp dữ liệu và trả danh sách giao dịch trở lại giao diện người dùng được tải trong trình duyệt. Trình duyệt sau đó sẽ định dạng và hiển thị dữ liệu cho người dùng.

Giao diện Lập trình Ứng dụng (API)

Không có phần mềm hữu ích nào là không giao tiếp với các thành phần bên trong và bên ngoài. Vậy làm thế nào để máy khách có thể giao tiếp với ứng dụng web? Làm cách nào giao diện người dùng có thể gửi dữ liệu đến máy chủ?

Các ứng dụng phần mềm giao tiếp với nhau thông qua *giao diện lập trình ứng dụng* (API) chạy trên *các giao thức* giao tiếp internet cơ bản. Các giao thức là các tiêu chuẩn và quy tắc được phát triển để đảm bảo rằng hai hoặc nhiều ứng dụng có thể trao đổi lệnh và dữ liệu.

Lợi ích chính của API là tách rời các phần khác nhau của ứng dụng, đồng thời cho phép chúng hợp tác với nhau để xử lý dữ liệu. API cũng tập trung luồng dữ liệu vào các kênh được xác định trước, hoạt động giống như một chiếc cổng có trách nhiệm đảm bảo rằng mọi thứ đều đến và đi bằng cùng một cách. Trong các ứng dụng web, API rất quan trọng đối với chức năng của ứng dụng vì chúng cho phép người dùng tương tác, cung cấp thông tin đã xử lý, yêu cầu lưu trữ dữ liệu và nhiều tác vụ khác. Ví dụ: khách hàng có thể sử dụng API để yêu cầu các hành động sẽ được thực thi trên máy chủ.

Hãy quay lại ví dụ về ứng dụng web ngân hàng. Để đăng nhập vào tài khoản thông qua ứng dụng web, người dùng thường sẽ nhập dữ liệu như tên người dùng và mật khẩu vào các trường văn bản nhất định và nhấp vào nút “Đăng nhập”. Trình duyệt sẽ lấy thông tin này và gọi một API máy chủ. Ứng dụng web chạy trên máy chủ từ xa sẽ nhận dữ liệu người dùng, xác thực người dùng, xác minh quyền truy cập của người dùng và cuối cùng gửi phản hồi trở lại trình duyệt. Để người dùng có thể giao tiếp với máy chủ, bắt buộc cả máy khách và máy chủ đều phải gửi dữ liệu qua lại. Đây chính là những gì API cho phép.

Có thể thấy được ứng dụng web của ngân hàng sẽ không để lộ các thông tin nhạy cảm khác; nó sẽ chỉ hiển thị cho người dùng các trường được phép và cần thiết cho tương tác mong muốn. Phần còn lại đều sẽ được ẩn đối với người dùng.

Giao tiếp giữa các API có thể dựa trên các thiết kế và giao thức rất khác nhau. *Giao thức truyền siêu văn bản* (HTTP) cho đến nay là giao thức được sử dụng thường xuyên nhất trong các ứng dụng web. *Siêu văn bản* là văn bản có liên kết đến các văn bản khác, đây là khái niệm nền tảng của các liên kết trong trang web HTML. Từ đây, các siêu liên kết sẽ tạo thành cơ sở để xây dựng các trang

web và hiển thị chúng trong trình duyệt.

HTTP được thiết kế cho các ứng dụng máy khách-máy chủ nơi tài nguyên được yêu cầu từ máy chủ và sau đó được trả về máy khách qua mạng bằng cấu trúc được xác định trước do giao thức HTTP chỉ định.

Đối với một ứng dụng web có cấu trúc, các kỹ sư phần mềm sẽ thiết kế ứng dụng với các phần hoặc mô-đun riêng biệt. Sự phân tách trách nhiệm này cho phép các tác vụ và trách nhiệm được xác định rõ ràng, tạo điều kiện cho việc phát triển nhanh hơn và bảo trì tốt hơn.

Hãy lấy ví dụ về một ứng dụng có hai mô-đun bên trong: một mô-đun triển khai logic nghiệp vụ và mô-đun còn lại dựa vào sự tích hợp với bên thứ ba. Bên thứ ba này là một công ty bên ngoài cung cấp API của họ cho một số mục đích cụ thể — chẳng hạn như dự báo thời tiết. Nếu máy chủ thời tiết ngừng hoạt động, chúng ta sẽ không thể lấy thông tin chi tiết về thời tiết. Nếu dữ liệu này quan trọng đối với kết quả xử lý cuối cùng và không có nguồn thay thế cho dữ liệu thì người dùng có thể sẽ nhất thời gặp phải một số vấn đề khá đau đầu.

Hãy tưởng tượng rằng nhà cung cấp bên thứ ba này đã được thay thế và nhà cung cấp mới có cách khác để xử lý cùng một API. Việc tách các mô-đun có nghĩa là các nhà phát triển chỉ có một mô-đun phải cập nhật. Logic nghiệp vụ trong mô-đun còn lại hoàn toàn không cần phải động tới, hoặc nhiều nhất là yêu cầu cập nhật ở mức tối thiểu.

Nhu cầu về cấu trúc tiến trình rõ ràng cũng ảnh hưởng đến việc thiết kế các API để làm cho chúng dễ sử dụng hơn. Khái niệm *chuyển trạng thái đại diện* (Representational State Transfer - REST) là kiểu kiến trúc với một bộ hướng dẫn để thiết kế và triển khai quyền truy cập vào dữ liệu trong một ứng dụng.

Có sáu nguyên tắc REST. Để đơn giản hoá, chúng ta sẽ cùng tìm hiểu về ba điều có liên quan nhất đến bài học này:

Tách rời máy khách-máy chủ

Máy khách chỉ nên biết URI tài nguyên và giao tiếp với máy chủ qua đó. Nguyên tắc này sẽ khiến giao tiếp trở nên linh hoạt hơn. Ví dụ: khi máy chủ của ứng dụng được tái cấu trúc hoặc có thay đổi lớn về kiến trúc đối với cơ sở dữ liệu phụ trợ thì giao diện người dùng không cần phải được cập nhật cùng lúc. Nó chỉ đơn giản là tiếp tục gửi các yêu cầu HTTP tương tự đến máy chủ.

Phi trạng thái

Mỗi yêu cầu mới sẽ độc lập với những yêu cầu trước đó. Không phải ngẫu nhiên mà giao thức HTTP được sử dụng rộng rãi cho các ứng dụng tuân theo nguyên tắc REST bởi HTTP sẽ không biết gì về các yêu cầu HTTP trước đó; đối với mỗi yêu cầu mới, tất cả thông tin cần thiết đều phải được gửi đi để xử lý yêu cầu một cách chính xác. Ví dụ: một ứng dụng web thực hiện

nguyên tắc này sẽ không biết máy khách liệu có đăng nhập (được xác thực) hay không. Vì vậy, đối với mỗi yêu cầu HTTP, máy khách sẽ phải gửi mã thông báo xác thực. Máy chủ có thể sử dụng mã thông báo này để xác minh xem yêu cầu nên bị chặn hay được xử lý.

Một trong những ưu điểm chính của nguyên tắc này là dễ dàng mở rộng quy mô hơn vì máy chủ có thể xử lý hàng triệu yêu cầu mà không cần kiểm tra chi tiết người dùng.

Kiến trúc phân lớp

Ứng dụng bao gồm nhiều lớp và mỗi lớp có thể có logic và mục đích riêng như bảo mật hoặc thu thập dữ liệu. Máy khách có thể sẽ không bao giờ biết được có bao nhiêu lớp tồn tại hoặc liệu chúng có đang giao tiếp trực tiếp với một lớp cụ thể bên trong ứng dụng hay không.

Các API tuân theo các nguyên tắc REST được gọi là *RESTful* và trong web hiện đại, thiết kế REST được nhiều ứng dụng web tuân theo. Mặc dù API RESTful không cần triển khai các nguyên tắc này bằng giao thức HTTP nhưng nó vẫn gần như được sử dụng phổ quát trong mô hình REST nhờ tính mạnh mẽ, đơn giản và phổ biến trong môi trường web toàn cầu.

Các kiểu Kiến trúc

Có hàng chục kiểu cách và tiêu chuẩn kiến trúc đang tồn tại nhằm tổ chức các cấu trúc của ứng dụng web. Giống như hầu hết mọi thứ trong thế giới máy tính, không có cái gì được gọi là "toàn diện", chỉ có một tập hợp ưu và nhược điểm cho mỗi mô hình. Một mô hình quan trọng được gọi là *kiến trúc dịch vụ vi mô* đã được tạo ra để thay thế cho *kiến trúc nguyên khối* đã cũ.

Kiến trúc dịch vụ vi mô là một mô hình phần mềm bao gồm nhiều *dịch vụ* phụ thuộc lẫn nhau và cùng nhau tạo thành ứng dụng cuối cùng. Kiến trúc này nhằm mục đích phân cấp cơ sở mã: Nhiều lớp phần mềm được chia thành nhiều ứng dụng nhỏ hơn để có thể bảo trì tốt hơn (Kiến trúc dịch vụ vi mô).

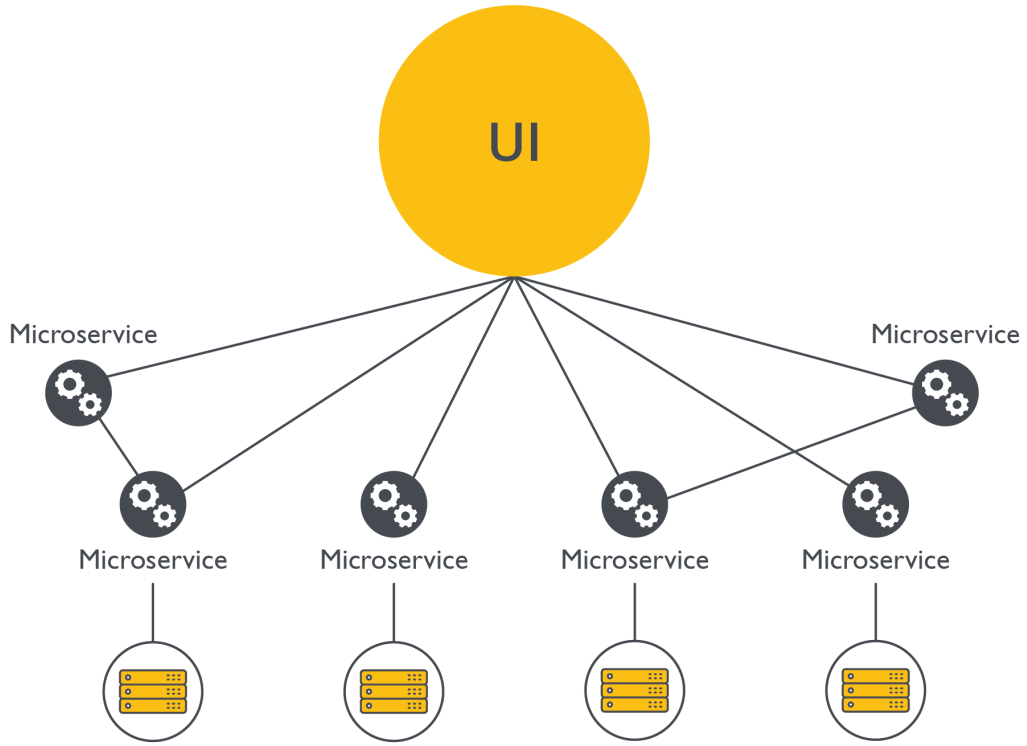

Figure 2. Kiến trúc dịch vụ vi mô

Ngược lại, một kiến trúc nguyên khối sẽ chứa tất cả các dịch vụ và tài nguyên của ứng dụng trong một ứng dụng (Kiến trúc nguyên khối).

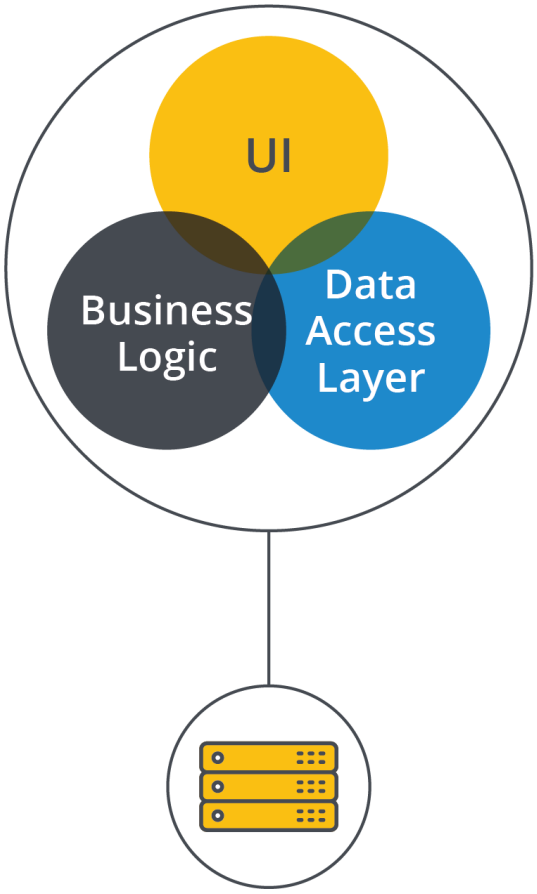

Figure 3. Kiến trúc nguyên khối

Các hình ảnh cho thấy mô hình dịch vụ vi mô được phân cấp và ứng dụng dựa trên nhiều dịch vụ, mỗi dịch vụ đều có cơ sở dữ liệu, cơ sở mã và thậm chí là cả tài nguyên máy chủ riêng. Đúng như tên gọi, mỗi dịch vụ vi mô đều phải nhỏ hơn so với phó bản nguyên khối chịu trách nhiệm về tất cả các dịch vụ của nó.

Ứng dụng nguyên khối sẽ đóng gói tất cả tài nguyên của nó thành một đơn vị duy nhất. Tất cả các logic nghiệp vụ, dữ liệu và cơ sở mã đều sẽ được tập trung thành một khối lớn. Đây chính là lý do tại sao nó được gọi là “nguyên khối.”

Người dùng hầu như sẽ không chú ý việc ứng dụng web đang chạy với mô hình nguyên khối hay dịch vụ vi mô; sự lựa chọn về vấn đề này nên được minh bạch. Ví dụ: ứng dụng ngân hàng có thể là một khối nguyên khối nơi tất cả caclogic nghiệp vụ liên quan đến thanh toán, giao dịch, khoản vay, v.v. đều nằm trong cùng một cơ sở mã chạy trên một hoặc nhiều máy chủ. Mặt khác, nếu ứng dụng ngân hàng sử dụng kiểu dịch vụ vi mô, nó có thể có một dịch vụ vi mô dành riêng cho việc xử lý các khoản thanh toán và một dịch vụ vi mô khác chỉ để phát hành các khoản vay. Dịch vụ vi

mô phát hành khoản vay sẽ gọi thêm một dịch vụ vi mô khác để phân tích xác suất người nộp đơn không trả được nợ. Ứng dụng có thể có tới hàng ngàn các dịch vụ nhỏ hơn.

Phương thức nguyên khối tiêu hao nhiều kinh phí trì hơn khi ứng dụng phát triển lớn hơn, đặc biệt là khi có nhiều đội ngũ viết mã trong cùng một cơ sở mã. Với các tài nguyên phần mềm tập trung, rất có thể một đội nhóm sẽ thay đổi điều gì đó dẫn đến làm hỏng phần ứng dụng của một nhóm khác. Đây có thể là một vấn đề thực sự đau đầu đối với các đội ngũ làm việc lớn với số lượng nhóm làm việc cao.

Về mặt này, phương thức dịch vụ vi mô linh hoạt hơn nhiều vì mỗi dịch vụ chỉ được quản lý bởi một nhóm. Tất nhiên, một nhóm cũng có thể quản lý nhiều dịch vụ. Việc thay đổi mã có thể được thực hiện dễ dàng và việc cạnh tranh tài nguyên sẽ không phải là một vấn đề thực sự. Vì mỗi dịch vụ đều được kết nối với nhau nên bất kỳ điểm lỗi nào cũng có thể có tác động tiêu cực đến toàn bộ ứng dụng. Hơn nữa, vì có nhiều phiên bản cơ sở dữ liệu, máy chủ và API bên ngoài giao tiếp với nhau nên khả năng phục hồi của toàn bộ ứng dụng chỉ đạt ngang tầm với dịch vụ vi mô yếu nhất của nó.

Một ưu điểm của phương pháp nguyên khối là có một nguồn dữ liệu tập trung giúp dễ dàng tránh trùng lặp dữ liệu hơn. Cách tiếp cận này cũng làm giảm mức tiêu thụ tài nguyên đám mây vì một máy chủ lớn hơn cần ít tài nguyên máy tính hơn nhiều so với các máy chủ phi tập trung. Một ứng dụng dịch vụ vi mô có kích thước tương đương sẽ đặt lên đám mây một gánh nặng lớn hơn.

Bài tập Hướng dẫn

1. Sự khác biệt chính giữa máy khách cấu hình tối thiểu và máy khách cấu hình tối đa là gì?

2. Có đúng không khi cho rằng mọi trang web đều là một ứng dụng web?

3. Mô hình REST là gì?

4. Mô hình ưa thích để phát triển các ứng dụng web lớn và hiện đại với nhiều nhóm phát triển là gì? Hãy giải thích tại sao.

5. Giao thức được sử dụng phổ biến nhất để trao đổi dữ liệu giữa các ứng dụng web là gì?

6. Hãy nêu hai nhược điểm của ứng dụng đa trang so với ứng dụng một trang.

7. Hãy mô tả một lợi thế của hệ thống nguyên khối so với hệ thống dịch vụ vi mô và một lợi thế của hệ thống dịch vụ vi mô so với hệ thống nguyên khối.

Bài tập Mở rộng

1. Vào năm 2021, NASA Perseverance Rover (xe rover tự hành) đã hạ cánh xuống Sao Hỏa với một trong số các mục tiêu là xác định xem liệu sự sống có từng tồn tại trên Sao Hỏa hay không. Mặc dù Rover có thể được điều khiển từ xa từ trên Trái đất nhưng nó cũng có thể tự điều khiển trong hầu hết các tình huống. Tại sao việc mô phỏng một chiếc rover như một máy khách cấu hình tối đa lại là một ý tưởng hay?

2. Hãy xét một chiếc ô tô cá nhân tự lái, hiện đại và có kết nối với máy chủ bên ngoài để trao đổi dữ liệu. Nó nên là một máy khách cấu hình tối đa hay tối thiểu?

Tóm tắt

Bài học này đã giải thích các khái niệm cốt lõi của kiến trúc phần mềm cho các ứng dụng web. Bài học đã lý giải cách chúng được cấu trúc và tổ chức phổ biến cũng như những điểm khác biệt chính giữa mô hình nguyên khối và mô hình dịch vụ vi mô. Bài học cũng đã đề cập đến các khái niệm về máy chủ và máy khách cũng như những kiến thức cơ bản về giao tiếp ứng dụng web giữa máy khách và các chương trình phần mềm khác.

Đáp án Bài tập Hướng dẫn

1. Sự khác biệt chính giữa máy khách cấu hình tối thiểu và máy khách cấu hình tối đa là gì?

   Máy khách cấu hình tối đa không yêu cầu một kết nối liên tục với máy chủ từ xa để cung cấp thông tin quan trọng cho máy khách đang chạy. Máy khách cấu hình tối thiểu phụ thuộc rất nhiều vào thông tin được xử lý bởi nguồn bên ngoài. Một điểm khác biệt nữa là máy khách cấu hình tối đa sẽ chịu trách nhiệm xử lý phần lớn dữ liệu, do đó yêu cầu nhiều tài nguyên máy tính hơn so với máy khách cấu hình tối thiểu.

2. Có đúng không khi cho rằng mọi trang web đều là một ứng dụng web?

   Không. Có những trang web không phải là ứng dụng phần mềm. Ứng dụng web tương tác với người dùng, người dùng có thể nhập dữ liệu và sử dụng các chức năng web trong thời gian thực. Các trang web đơn giản như quảng cáo cho một sự kiện xã hội với chức năng giống như một biểu ngữ web không phải là ứng dụng web. Những trang web không tương tác này dễ bảo trì hơn và yêu cầu tài nguyên máy tính rất nhỏ để lưu trữ và phân phối các trang của chúng. Một ứng dụng web sẽ yêu cầu nhiều tài nguyên máy tính hơn, máy chủ và các chức năng xử lý người dùng mạnh hơn (chẳng hạn như quyền truy cập hạn chế và lưu trữ dữ liệu vĩnh viễn).

3. Mô hình REST là gì?

   Mô hình REST là mô hình kiến trúc phần mềm cung cấp cho ứng dụng một chỉ dẫn phát triển để cải thiện khả năng sử dụng, tính rõ ràng và dễ bảo trì hơn. Một trong những nguyên tắc được nêu trong bộ nguyên tắc REST là *kiến trúc phân lớp* được sử dụng chủ yếu để gắn kết và giảm sự phụ thuộc của các thành phần bên trong của các API khác nhau.

4. Mô hình ưa thích để phát triển các ứng dụng web lớn và hiện đại với nhiều nhóm phát triển là gì? Hãy giải thích tại sao.

   Mô hình phần mềm dịch vụ vi mô cung cấp một khuôn khổ linh hoạt nơi các nhóm sẽ cộng tác trên cùng một ứng dụng phần mềm, mang lại khả năng hoạt động đồng thời dễ dàng hơn cho hai hoặc nhiều nhóm duy trì một ứng dụng web lớn. Vì là khuôn khổ phi tập trung nên mỗi nhóm đều có thể cập nhật một miền nghiệp vụ cụ thể mà không cần phải cập nhật các thành phần khác.

5. Giao thức được sử dụng phổ biến nhất để trao đổi dữ liệu giữa các ứng dụng web là gì?

   HTTP là giao thức được sử dụng nhiều nhất để trao đổi dữ liệu và lệnh giữa máy chủ và máy khách.

6. Hãy nêu hai nhược điểm của ứng dụng đa trang so với ứng dụng một trang.

Khi người dùng kích hoạt một số hành động, ứng dụng đa trang sẽ tải lại tất cả các thành phần trong trang web thay vì chỉ cập nhật các thành phần đã thay đổi. Hiệu suất sẽ bị ảnh hưởng từ thiết kế này. Một nhược điểm khác của MPA là khả năng tương tác với người dùng kém hơn: mỗi lần tải trang đều sẽ làm mất đi một chút tính thân thiện với người dùng. Trái với đó, trong một SPA, hiệu ứng hình ảnh lại có thể được liên tục.

7. Hãy mô tả một lợi thế của hệ thống nguyên khối so với hệ thống dịch vụ vi mô và một lợi thế của hệ thống dịch vụ vi mô so với hệ thống nguyên khối.

   Hệ thống nguyên khối có thể giúp quản trị dữ liệu dễ dàng hơn vì dữ liệu sẽ được đặt trong một cơ sở dữ liệu lớn thay vì nằm rải rác trong nhiều cơ sở dữ liệu. Mặt khác, một ứng dụng dịch vụ vi mô có thể cải thiện việc phát triển và bảo trì mã; nhiều nhóm cộng tác có thể làm việc theo các logic nghiệp vụ khác nhau mà không cản trở tiến độ của các nhóm còn lại.

Đáp án Bài tập Mở rộng

1. Vào năm 2021, NASA Perseverance Rover (xe tự hành) đã hạ cánh xuống Sao Hỏa với một trong số các mục tiêu là xác định xem liệu sự sống có từng tồn tại trên Sao Hỏa hay không. Mặc dù xe tự hành có thể được điều khiển từ xa từ trên Trái đất nhưng nó cũng có thể tự điều khiển trong hầu hết các tình huống. Tại sao việc mô phỏng một chiếc xe tự hành như một máy khách cấu hình tối đa lại là một ý tưởng hay?

   Thời gian tín hiệu liên lạc được gửi từ Trái đất và nhận được trên Sao Hỏa có thể khác nhau tùy thuộc vào vị trí của các hành tinh đó, nhưng việc này có thể mất tới 20 phút. Do đó, việc chỉ huy và điều khiển một chiếc xe tự hành ở xa đang chuyển động là không thể, đặc biệt là khi tính đến các tình huống bất ngờ. Lý tưởng nhất ở đây là xe tự hành nên tự điều khiển trong hầu hết các tình huống. Điều này có thể đạt được bằng cách sử dụng đào tạo (máy học) trí tuệ nhân tạo (AI) để xe tự hành trở nên độc lập hơn với các lệnh thủ công. Để biến điều này thành hiện thực và ít phụ thuộc hơn vào các tín hiệu từ xa, xe tự hành dự kiến sẽ phải có tài nguyên riêng và phần lớn các quy trình tính toán chạy cục bộ, phù hợp với định nghĩa về một máy khách cấu hình tối đa.

2. Hãy xét một chiếc ô tô cá nhân tự lái, hiện đại và có kết nối với máy chủ bên ngoài để trao đổi dữ liệu. Nó nên là một máy khách cấu hình tối đa hay tối thiểu?

   Một chiếc xe tự hành có thể ủy thác việc xử lý dữ liệu nặng cho một máy chủ bên ngoài và đáng tin cậy, nhưng điều này sẽ dễ xảy ra các giai đoạn ngoại tuyến khi cần xử lý dữ liệu quan trọng. Do đó, phương tiện tự hành bắt buộc phải tự xử lý phần lớn các nhiệm vụ - và điều này đòi hỏi nó phải là một máy khách cấu hình tối đa với một lượng tài nguyên lớn.

051.3 Giải pháp Tại chỗ và Điện toán Đám mây

Tham khảo các mục tiêu LPI

Open Source Essentials version 1.0, Exam 050, Objective 051.3

Khối lượng

1

Các lĩnh vực kiến thức chính

- Hiểu rõ các khái niệm về điện toán tại chỗ và điện toán đám mây
- Hiểu rõ các mô hình hoạt động đám mây phổ biến
- Hiểu về các loại dịch vụ đám mây phổ biến
- Hiểu rõ các lợi ích và rủi ro chính của điện toán đám mây và cơ sở hạ tầng CNTT tại ch

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Điện toán đám mây
- Cơ sở hạ tầng CNTT tại chỗ
- Trung tâm dữ liệu
- Đám mây công cộng, riêng tư và lai
- Cơ sở hạ tầng dưới dạng dịch vụ (IaaS), Nền tảng dưới dạng dịch vụ (PaaS), Phần mềm dưới dạng dịch vụ (SaaS)
- Mô hình chi phí
- Bảo mật
- Quyền sở hữu dữ liệu
- Tính khả dụng của dịch vụ

Linux Professional Institute

Bài 1

Chứng chỉ:	Open Source Essentials
Phiên bản:	1.0
Chủ đề:	051 Những nguyên tắc cơ bản về Phần mềm
Mục tiêu:	051.1 Giải pháp Tại chỗ và Điện toán Đám mây
Bài học:	1 trên 1

Giới thiệu

Ngày nay, mọi người đều đang nói về đám mây. Các doanh nghiệp đang đánh giá các dịch vụ từ nhiều nhà cung cấp — bao gồm cả các công ty lớn như Amazon.com và Microsoft — để xem đám mây có thể cung cấp được những gì.

Nhưng ý tưởng về điện toán đám mây chỉ mới mới xuất hiện từ cuối thập niên 2000, và thuật ngữ này là một thuật ngữ rất mơ hồ — mơ hồ đến mức có rất nhiều người, kể cả Tổ chức Phần mềm Tự do (Free Software Foundation), đều không khuyến khích việc sử dụng thuật ngữ này.

Tuy nhiên, ý tưởng về điện toán đám mây bao hàm một số khái niệm cụ thể khá hữu ích và là một xu hướng quan trọng trong điện toán hiện đại. Bài học này sẽ nói về khái niệm đám mây và, ở một mức độ cao hơn, cách nó hoạt động cả về mặt kỹ thuật lẫn tài chính. Chúng ta sẽ cùng xem xét cả về lợi ích lẫn rủi ro của đám mây. Trong quá trình đó, chúng ta sẽ dần hiểu được tại sao chủ đề này lại liên quan đến mã nguồn mở.

Giải pháp Tại chỗ và Điện toán Đám mây

Nếu phải làm việc hoặc học tập trong một tổ chức có nhiều hệ thống máy tính đơn lẻ, tổ chức đó

gần như chắc chắn sẽ phải có một *trung tâm dữ liệu*: một khu vực riêng để chứa các máy chủ của tổ chức, thường có khóa và có điều hòa. Trung tâm dữ liệu *tại chỗ* (hoặc *on-premise*) chỉ đơn giản là một trung tâm mà tổ chức duy trì để sử dụng cho riêng mình.

Có một số lựa chọn thay thế để vận hành một trung tâm dữ liệu tại chỗ. Trong nhiều thập kỷ trước khi xuất hiện xu hướng mà chúng ta gọi là "điện toán đám mây", các doanh nghiệp sẽ thay mặt khách hàng thiết lập các trung tâm dữ liệu để lưu trữ máy tính. Do đó, chúng ta có thể cấp phép cho năm máy chủ của doanh nghiệp và cung cấp cho doanh nghiệp các thông số kỹ thuật khác nhau về CPU, bộ nhớ và bộ lưu trữ, sau đó tải phần mềm của mình lên máy chủ. Mô hình kinh doanh này là *lưu trữ từ xa* (remote hosting).

Lưu trữ từ xa cho chúng ta rất nhiều lợi thế. Tổ chức có thể thuê ngoài các chuyên gia quản trị về lĩnh vực mua và thiết lập máy chủ cho doanh nghiệp lưu trữ từ xa; việc này có thể được thực hiện với số lượng lớn. Nói cách khác, chúng ta sẽ tránh được trách nhiệm đối với việc phải có một cơ sở hạ tầng CNTT tại chỗ. Hoạt động kinh doanh dịch vụ lưu trữ từ xa có thể đảm bảo an ninh vật lý, đồng thời giúp khách hàng thoát khỏi nỗi lo lắng này. Cuối cùng, việc thiết lập một máy chủ mới cho doanh nghiệp cung cấp dịch vụ lưu trữ từ sẽ xa nhanh hơn nhiều so với việc mua, vận chuyển và thiết lập máy chủ tại chỗ.

Một tổ chức cũng có thể duy trì một trung tâm dữ liệu tại chỗ, đồng thời cấp phép cho nhiều máy chủ hơn trong môi trường từ xa để phục hồi sau sự cố hoặc cung cấp thêm năng lực tính toán trong thời gian sử dụng cao điểm.

Hãy nhớ rằng việc tính toán từ xa sẽ cần tới một mạng nhanh và đáng tin cậy. Chúng ta sẽ khám phá về vấn đề này cùng với những lợi ích và điểm yếu đi kèm ở một phần khác.

Tất cả những lợi ích của việc lưu trữ từ xa cũng áp dụng cho điện toán đám mây. Điện toán đám mây khác biệt về mặt kỹ thuật vì không có máy tính vật lý cụ thể nào được dành riêng cho tổ chức. Thay vào đó, nhà cung cấp đám mây sẽ chạy nhiều hệ thống cho nhiều máy khách trên mỗi một hệ thống vật lý và sử dụng thêm một lớp phần mềm được gọi là *máy ảo*.

Từ đây, dịch vụ đám mây sẽ vận hành một trung tâm dữ liệu giống như bất kỳ một tổ chức nào khác, nhưng nó lại phục vụ các tổ chức khác thay vì (hoặc bên cạnh) chính nó. Trung tâm dữ liệu sẽ lưu trữ hàng ngàn máy tính vật lý. Trên mỗi máy tính vật lý, nó sẽ chạy một hệ điều hành (thường được gọi là *trình giám sát máy ảo* - hypervisor) hỗ trợ nhiều máy ảo. Mỗi máy ảo có thể được sinh ra và bị xóa đi một cách nhanh chóng. Mỗi một máy ảo sẽ hỗ trợ một hệ điều hành do một máy khách điều hành (Điện toán đám mây).

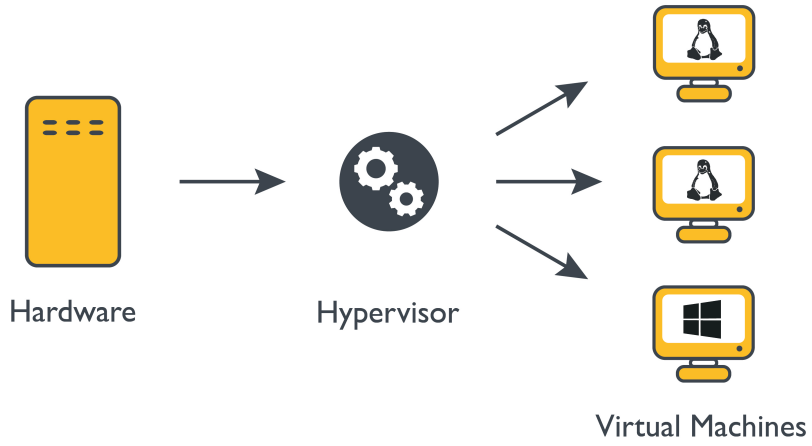

Figure 4. Điện toán đám mây

Vậy phần mềm tự do và mã nguồn mở sẽ xuất hiện từ điểm nào? Khi một doanh nghiệp đang vận hành một số lượng lớn máy tính và triển khai hệ điều hành một cách nhanh chóng, điều quan trọng nhất là không bị sa lầy vào việc xử lý giấy phép. Mặc dù có các mô hình cấp phép cho hệ điều hành độc quyền trên đám mây nhưng chúng phức tạp hơn việc chỉ chạy một máy ảo và hệ điều hành mã nguồn mở. Ngoài ra, mã nguồn mở cũng thường sẽ miễn phí.

Sự khởi đầu của điện toán đám mây thường quay về thời điểm ra mắt Amazon Web Services (AWS) của Amazon.com vào năm 2006. Hiện tại có hàng chục công ty đám mây bao gồm cả các dịch vụ từ các nhà cung cấp lớn như Microsoft, Google, Alibaba và IBM. Dù vậy, AWS vẫn là sản phẩm lớn nhất. Các nhà cung cấp cạnh tranh gay gắt với nhau trong việc phát triển các tính năng và dịch vụ mới bởi họ đều có thế mạnh ngang nhau về mức chi phí và độ tin cậy.

Ưu điểm của điện toán đám mây được xây dựng dựa trên lợi ích của điện toán từ xa. Chi phí của nó thấp hơn vì một hệ thống vật lý có thể chạy nhiều máy chủ cho nhiều khách hàng và có thể bận rộn liên tục. Một khách hàng cần nhiều sức mạnh tính toán một cách nhanh chóng để tăng mức sử dụng có thể tạo ra các hệ thống mới trong vài giây. Các hệ thống có thể được quản lý tự động thông qua giao diện lập trình ứng dụng (API). Chúng ta sẽ xem xét kỹ hơn về các lợi ích và rủi ro ở các phần sau.

Các Mô hình vận hành Đám mây phổ biến

Trước khi tìm hiểu chi tiết các mô hình hoạt động, chúng ta cần lưu ý rằng nhiều công ty đã áp dụng các mô hình đám mây giúp thay đổi hoàn toàn cách lập trình và cung cấp dịch vụ của chính họ. Thay vì cập nhật mỗi ứng dụng một hoặc hai lần một năm, các công ty sẽ cho phép cập nhật

nhanh chóng. Họ có thể làm điều này vì đám mây cho phép họ tắt máy ảo và khởi động các máy mới với phiên bản mới của ứng dụng gần như ngay lập tức. Các tổ chức cũng có thể tăng và giảm quy mô một cách nhanh chóng; vì vậy mà họ thường chia ứng dụng thành nhiều phần mô-đun đôi khi được gọi là *các dịch vụ vi mô*.

Nhưng trong phần này, chúng ta sẽ chỉ tập trung vào các mô hình đám mây thông thường.

Mô hình chi phí cho đám mây rất khác với chi phí cho giải pháp tại chỗ. Các trung tâm dữ liệu tại chỗ sẽ yêu cầu mua máy chủ một lần duy nhất cùng với các chi phí thường lệ về điện, điều hòa không khí và quản trị. Những thứ này sẽ biến mất khi chúng ta cấp phép hệ thống từ một nhà cung cấp đám mây. Thay vào đó, chúng ta sẽ phải trả phí cho những gì mình sử dụng. Các nhà cung cấp đám mây chia việc sử dụng máy tính của khách hàng thành các khoảng thời gian và tính phí cho họ theo từng khoảng thời gian đó. Họ cũng sẽ tính phí theo lượng dữ liệu khách hàng lưu trữ trên hệ thống của mình.

Cho đến nay, chúng ta đã nói về các nhà cung cấp năng lực tính toán cho khách hàng; đây được gọi là *đám mây công cộng*. Nhưng chúng ta cũng có thể có một *đám mây riêng tư*. Một số tổ chức lớn sẽ vận hành các trung tâm dữ liệu tại chỗ của riêng họ giống như một đám mây. Họ chỉ cung cấp dịch vụ cho các phòng ban hoặc phân khu của riêng mình, nhưng họ sẽ coi mỗi phòng ban của mình như một khách hàng của nhà cung cấp đám mây. Trung tâm dữ liệu sẽ theo dõi lượng thời gian tính toán, dữ liệu, v.v. được mỗi bộ phận sử dụng và tính phí cho mức sử dụng đó.

Có rất nhiều lý do khác nhau khiến nhiều tổ chức sử dụng nhiều loại dịch vụ đám mây, chẳng hạn như để bảo vệ khỏi lỗi của nhà cung cấp, lưu giữ dữ liệu ở một khu vực địa lý nhất định hoặc tận dụng các tính năng đặc biệt do một nhà cung cấp cụ thể cung cấp. Ngoài ra, việc duy trì cả trung tâm dữ liệu tại chỗ và máy chủ trên đám mây là một việc rất phổ biến; phương pháp này được gọi là *điện toán lai* (hybrid computing).

Khách hàng đăng ký dịch vụ đám mây có thể chọn khu vực địa lý để chạy. Ví dụ: Amazon hiện cung cấp các khu vực ở Miền Tây Hoa Kỳ, Miền Đông Hoa Kỳ, một số khu vực ở Châu Âu và nhiều khu vực khác ở mọi nơi trên thế giới. Thông thường, khách hàng sẽ chọn khu vực gần với mình nhất. Tuy nhiên, có nhiều tổ chức sẽ muốn hoạt động ở nhiều khu vực khác nhau vì họ có phạm vi quốc tế. Các tổ chức đôi khi sẽ cần lưu giữ dữ liệu ở một nơi cụ thể để tuân thủ Quy định bảo vệ dữ liệu chung của Châu Âu (GDPR) hoặc Luật bảo vệ thông tin cá nhân của Trung Quốc (PIPL).

Mỗi khu vực thường được chia nhỏ thành các *vùng* hoặc *vùng khả dụng*. Khách hàng được khuyến nghị nên chạy ở nhiều vùng trong trường hợp thảm họa có thể khiến một vùng bị hỏng.

Mặc dù đám mây nổi bật với tính năng chia sẻ máy tính vật lý giữa nhiều khách hàng nhưng một số nhà cung cấp có thể dành riêng một máy tính duy nhất cho một khách hàng cụ thể có quan ngại về vấn đề bảo mật. Vì không có tổ chức nào khác sử dụng máy tính này nên khách hàng sẽ cảm thấy an toàn hơn khi chạy các dịch vụ nhạy cảm và tải dữ liệu của họ lên đám mây. Tùy chọn này

đã đưa điện toán đám mây đến gần hơn với dịch vụ lưu trữ từ xa kiểu cũ.

Các loại Dịch vụ Đám mây phổ biến

Ở các cấp độ khác nhau, điện toán đám mây sẽ có một vẻ ngoài nhất định và nhắm đến các loại người dùng khác nhau.

Cơ sở hạ tầng dưới dạng dịch vụ (IaaS) là loại hình mà quản trị viên hệ thống thường xử lý. IaaS chỉ cung cấp phần cứng và phần mềm hỗ trợ máy ảo. Quản trị viên hệ thống của máy khách có quyền tải hệ điều hành và các ứng dụng mong muốn của họ vào máy ảo. Quản trị viên hệ thống sẽ xử lý gần như mọi thứ giống như với trung tâm dữ liệu tại chỗ.

Nền tảng dưới dạng dịch vụ (PaaS) là một phát minh gần đây hơn được sử dụng chủ yếu bởi các lập trình viên. Ở đây, lập trình viên sẽ không phải lo lắng về hệ điều hành và không phải tải các thư viện mà chương trình sử dụng. Tất cả những thứ này đều sẽ được nhà cung cấp đám mây cung cấp. Lập trình viên sẽ chỉ tải lên các chức năng chạy trên nền tảng. Một khái niệm liên quan chính là điện toán *phi máy chủ* (serverless).

Phần mềm dưới dạng dịch vụ (SaaS) là một ứng dụng chạy trên hệ thống đám mây. Mỗi khi người dùng đăng nhập vào một trang mạng xã hội, đặt mua một mặt hàng từ cửa hàng trực tuyến, truy cập trang web để nhập giờ làm việc của họ vào hệ thống theo dõi công việc hoặc nhập biểu mẫu trên trang web của chính phủ, họ đều đang sử dụng SaaS. Phần lớn của ứng dụng đang chạy trên hệ thống từ xa và phần duy nhất của ứng dụng đang chạy trên máy tính của người dùng chính là trang web được trình duyệt của họ hiển thị.

Cơ sở dữ liệu dưới dạng dịch vụ (DaaS) thường được thêm vào các danh mục trên. Dịch vụ dữ liệu S3 của Amazon thực ra chính là dịch vụ đám mây đầu tiên. Việc cung cấp DaaS có thể đơn giản là một phiên bản của máy chủ cơ sở dữ liệu phổ biến như MySQL, Oracle hoặc MongoDB chạy trên đám mây. Các nhà cung cấp đám mây lớn cũng cung cấp các cơ sở dữ liệu độc quyền chỉ chạy trong các dịch vụ đám mây của họ. Trong mọi trường hợp, người dùng sẽ đọc và ghi cơ sở dữ liệu như thể họ có nó trên hệ thống của mình.

Các biến thể khác của các danh mục cơ bản đó được cung cấp bởi một số công ty (chẳng hạn như Bảo mật dưới dạng Dịch vụ).

Lợi ích và rủi ro chính của Điện toán Đám mây và Cơ sở Hạ tầng CNTT tại chỗ

Trước khi xem xét một cách chi tiết về điện toán đám mây, chúng ta hãy thử làm một phép loại suy. Việc vận hành một trung tâm dữ liệu tại chỗ cũng giống như việc mua một ngôi nhà. Nếu tầng hầm của ngôi nhà bị ngập nước hoặc lò hơi ngừng hoạt động, chúng ta sẽ cần tìm người sửa chữa.

Ngược lại, lưu trữ từ xa và điện toán đám mây giống như việc thuê một căn hộ: chủ nhà sẽ phải chịu trách nhiệm sửa chữa lò hơi đó chứ không phải chúng ta. Hơn nữa, trên đám mây, chúng ta có thể nhanh chóng thêm và xóa các tải dữ liệu giống như việc thay đổi căn hộ chúng ta thuê sẽ nhanh hơn việc thay đổi ngôi nhà mà mình sở hữu.

Trong một căn hộ cho thuê, chủ nhà thậm chí sẽ cung cấp các thiết bị và cả đồ nội thất. Theo cách hình dung tương tự của chúng ta, điều này cũng giống như việc các nhà cung cấp đám mây đem lại các dịch vụ đa dạng cho khách hàng (chẳng hạn như các cơ sở dữ liệu và bản phân tích).

Bây giờ, chúng ta có thể xem xét các lợi ích và rủi ro của việc sử dụng đám mây thay vì/ bên cạnh việc sử dụng trung tâm dữ liệu của riêng mình.

Tính linh hoạt có lẽ là lý do thuyết phục nhất để chuyển sang đám mây. Nếu là một nhà bán lẻ cần chạy nhiều máy chủ hơn vào dịp Giáng sinh hoặc một nhân viên kế toán thuế thực hiện gần như toàn bộ các công việc của chúng ta vào mùa thuế, chúng ta sẽ muốn đám mây khởi động các máy chủ mới ngay lập tức và sau đó xóa các máy ảo của chúng sau.

Chi phí có thể sẽ thấp hơn trên đám mây vì một số lý do. Chúng ta đang chia sẻ một máy chủ vật lý với nhiều ứng dụng khác, nhờ đó mà máy tính có thể được sử dụng một cách hiệu quả hơn. Bởi vì các nhà cung cấp đám mây rất lớn nên họ có thể đạt được tính kinh tế nhờ quy mô trong việc mua hàng, quản trị, làm mát và các yêu cầu cơ sở hạ tầng khác. Cuối cùng, khách hàng sẽ được giải phóng khỏi nhiều nhiệm vụ quản trị—mặc dù việc quản trị hệ thống không hề bị loại bỏ. Khách hàng vẫn cần tới quản trị viên hệ thống để tạo và tải lên phần mềm của họ (được gọi là các *phiên bản* trên đám mây), ủy quyền cho người dùng cũng như các tác vụ khác liên quan đến hoạt động kinh doanh. Quản trị viên hệ thống phải tìm hiểu về API của nhà cung cấp và các quy tắc sử dụng dịch vụ; đây cũng là một khoản tiêu tốn cần được tính vào kế hoạch của người sử dụng dịch vụ.

Mặt khác, người dùng phải cẩn thận với mức độ sử dụng đám mây của mình. Việc theo dõi năng lượng tính toán máy tính đang sử dụng có thể sẽ tương đối khó khi người dùng có thể nhanh chóng khởi động máy chủ, đặc biệt là nếu họ tự động hóa quy mô. Người dùng có thể sẽ nhận được một hóa đơn lớn không mấy dễ chịu vào kỳ thanh toán.

Đám mây có hiệu quả carbon tốt hơn so với việc chạy máy tính của chúng ta không? Nghiên cứu cho thấy các nhà cung cấp dịch vụ đám mây có thể vận hành hệ thống của họ hiệu quả hơn nhiều so với bất cứ người dùng nào. Tuy nhiên, chúng ta sẽ phải liên lạc với các hệ thống đó qua mạng; việc này cần tới rất nhiều điện năng để cung cấp năng lượng cho tất cả các thiết bị mạng. Thật không may, đám mây trên thực tế đã làm tăng lượng khí thải carbon của chúng ta.

Tính khả dụng của dịch vụ đôi khi sẽ tốt hơn đối với đám mây. Chắc chắn khi phụ thuộc vào trung tâm dữ liệu tại chỗ của riêng mình, chúng ta sẽ dễ gặp phải nhiều loại vấn đề, từ thiên tai đến những kẻ phá hoại nội bộ độc hại. Thế nhưng các trung tâm dữ liệu trên đám mây cũng có thể ngừng hoạt động. Vì vậy, chúng ta nên tận dụng các vùng sẵn có khác nhau và phân tán rủi ro của

mình. Có những công cụ sẽ cho phép chúng ta chuyển dịch vụ của mình từ vùng bị lỗi sang vùng đang hoạt động.

Nếu sử dụng các dịch vụ do nhà cung cấp đám mây cung cấp (chẳng hạn như cơ sở dữ liệu trên đám mây), chúng ta sẽ dễ gặp lỗi trong dịch vụ đó. Tất nhiên, chúng ta cũng có thể gặp lỗi trong phần mềm mà mình tải vào hệ thống.

Rủi ro lớn hơn khi sử dụng dịch vụ của nhà cung cấp là bị khóa. Chúng ta thường có thể tìm thấy một công cụ chuyển đổi tự động để di chuyển dữ liệu của mình ra khỏi hệ thống của nhà cung cấp và sang hệ thống mới, nhưng công cụ này có thể sẽ không thể thực hiện được công việc một cách hoàn chỉnh.

Vấn đề an ninh có thể sẽ tốt hơn trên đám mây vì nhân viên của nhà cung cấp có thể có nhiều chuyên môn hơn nhân viên bảo mật của người dùng. Mặt khác, các nhà cung cấp đám mây đều rất lớn và nổi tiếng nên cũng rất dễ trở thành các mục tiêu bị tấn công. Ngoài ra, việc bổ sung thêm một phần mềm -— trình giám sát máy ảo điều khiển các máy ảo -— sẽ gây ra một mối nguy hiểm tiềm ẩn mới. Các nhà nghiên cứu đã tìm thấy các lỗ hổng trong trình giám sát máy ảo.

Mặc dù khách hàng vẫn là chủ sở hữu hợp pháp đối với dữ liệu của mình nhưng về mặt lý thuyết, việc lưu trữ dữ liệu trên đám mây sẽ khiến dữ liệu dễ bị tổn thương hơn. Thông thường, máy khách sẽ mã hóa dữ liệu để bảo vệ dữ liệu trong trường hợp bị đột nhập. Các quy định về quyền riêng tư (chẳng hạn như GDPR đã đề cập ở trên) sẽ yêu cầu dữ liệu phải được lưu trữ trong trung tâm dữ liệu ở một khu vực được coi là an toàn.

Cuối cùng, hầu hết các cuộc tấn công bảo mật đều bắt đầu ở cấp độ cao (chẳng hạn như gửi email có chứa phần mềm độc hại cho một nhân viên không hề đáng ngờ). Việc khách hàng đang chạy tại chỗ hay trên đám mây thực chất là không quan trọng. Nhưng kẻ xâm nhập độc hại chiếm đoạt tài khoản của một nhân viên sẽ không tiến xa hơn được trừ khi chúng có thể lợi dụng các lỗ hổng trong máy chủ của khách hàng; một lần nữa, chúng ta vẫn chưa rõ liệu việc chạy trên đám mây có tạo ra nhiều khác biệt hay không vì hầu hết các lỗ hổng là được tìm thấy trong phần mềm chứ không phải trong dịch vụ đám mây.

Cuối cùng, chúng ta cần xem xét về chi phí băng thông và mạng. Khách hàng và có thể cả nhân viên của khách hàng sẽ phải liên lạc với các máy chủ cách xa hàng trăm dặm. Nếu kết nối mạng không đáng tin cậy hoặc chậm, hiệu suất của máy chủ đám mây sẽ kém hơn trung tâm dữ liệu tại chỗ. Nhưng ngày nay, mọi người đều kết nối với những người làm việc từ xa, các dịch vụ SaaS và các hệ thống khác ở xa về mặt địa lý. Hiệu suất mạng của khách hàng sẽ ảnh hưởng đến hầu hết mọi việc họ làm cho dù họ có ở trên đám mây hay không.

Bài tập Hướng dẫn

1. Tại sao máy tính vật lý trong trung tâm đám mây được sử dụng hiệu quả hơn máy tính trong trung tâm dữ liệu tại chỗ truyền thống?

2. Đám mây lai là gì?

3. Loại điện toán đám mây nào thường yêu cầu quản trị viên hệ thống ở phía máy khách làm việc nhiều nhất?

4. Bạn nên làm cách nào để bảo vệ dịch vụ của mình khỏi việc bị ngừng hoạt động khi sử dụng một nhà cung cấp đám mây?

Bài tập Mở rộng

1. Hãy so sánh các loại chi phí khác nhau mà bạn sẽ gặp phải khi chạy máy chủ trên đám mây so với tại chỗ.

2. Bạn đang hoạt động ở khu vực Trung Đông nhưng có nhiều khách hàng ở khu vực Châu Âu và Đông Á. Hãy mô tả nơi bạn sẽ đặt dịch vụ của mình trong một dịch vụ đám mây.

Tóm tắt

Bài học này trình bày cách thức hoạt động của điện toán đám mây và sự cân bằng giữa việc sử dụng đám mây và các hệ thống đang chạy trong trung tâm dữ liệu tại chỗ của riêng mọi người dùng. Chúng ta đã tìm hiểu các mô hình kinh doanh và chi phí khác nhau, bao gồm sự khác biệt giữa các đám mây công cộng, riêng tư và đám mây lai. Chúng ra cũng đã tìm hiểu về các loại dịch vụ đám mây chính khác nhau và mục đích sử dụng của từng loại.

Đáp án Bài tập Hướng dẫn

1. Tại sao máy tính vật lý trong trung tâm đám mây được sử dụng hiệu quả hơn máy tính trong trung tâm dữ liệu tại chỗ truyền thống?

   Trên đám mây, mỗi máy tính đều có thể chạy nhiều phiên bản của hệ điều hành và thậm chí chạy các phiên bản được tải lên bởi các máy khách khác nhau. Do vậy mà máy tính trên đám mây được sử dụng thường xuyên hơn.

2. Đám mây lai là gì?

   Đám mây lai sử dụng cả trung tâm dữ liệu của nhà cung cấp đám mây và một hoặc nhiều trung tâm dữ liệu tại chỗ.

3. Loại điện toán đám mây nào thường yêu cầu quản trị viên hệ thống ở phía máy khách làm việc nhiều nhất?

   Cơ sở hạ tầng dưới dạng dịch vụ (IaaS) cần khách hàng phải thực hiện việc quản trị hệ thống cho các tác vụ như tạo và tải lên các phiên bản của hệ điều hành và ứng dụng.

4. Bạn nên làm cách nào để bảo vệ dịch vụ của mình khỏi việc bị ngừng hoạt động khi sử dụng một nhà cung cấp đám mây?

   Chọn một số vùng trong mỗi khu vực nơi bạn vận hành dịch vụ của mình vì rất khó có khả năng nhiều vùng sẽ bị lỗi cùng một lúc.

Đáp án Bài tập Mở rộng

1. Hãy so sánh các loại chi phí khác nhau mà bạn sẽ gặp phải khi chạy máy chủ trên đám mây so với tại chỗ.

   Trên đám mây, bạn sẽ trả tiền cho việc sử dụng CPU và lưu trữ dữ liệu cho từng khoảng thời gian do nhà cung cấp đo lường và không phải trả bất kỳ chi phí phần cứng nào. Đối với lưu trữ tại chỗ, bạn cần trả chi phí cố định của phần cứng cùng với các thiết bị khác như điều hòa không khí, cộng với các chi phí định kỳ như điện và bảo trì vật lý.

2. Bạn đang hoạt động ở khu vực Trung Đông nhưng có nhiều khách hàng ở khu vực Châu Âu và Đông Á. Hãy mô tả nơi bạn sẽ đặt dịch vụ của mình trong một dịch vụ đám mây.

   Sử dụng khu vực Trung Đông cho văn phòng của bạn và khách hàng Trung Đông. Một khu vực ở Châu Âu để tuân thủ Quy định bảo vệ dữ liệu chung (GDPR). Bạn có thể sẽ cần một khu vực ở Trung Quốc để tuân thủ Luật bảo vệ thông tin cá nhân (PIPL) của Trung Quốc. Trong mọi trường hợp, việc có các khu vực Đông Á và Châu Âu sẽ rất có giá trị trong việc đạt được hiệu suất tốt hơn khi tương tác với khách hàng ở những khu vực đó.

   Trong mỗi khu vực, hãy chọn một số vùng để tránh được sự cố phát sinh từ một khu vực duy nhất.

Linux Professional Institute

Chủ đề 052: Giấy phép dành cho Phần mềm Mã nguồn Mở

052.1 Các khái niệm về Giấy phép Phần mềm Mã nguồn Mở

Tham khảo các mục tiêu LPI

Open Source Essentials version 1.0, Exam 050, Objective 052.1

Khối lượng

3

Các lĩnh vực kiến thức chính

- Hiểu rõ các định nghĩa về phần mềm mã nguồn mở và phần mềm tự do
- Nắm được kiến thức về các loại phần mềm tự do khác
- Nắm được kiến thức về các sự kiện quan trọng trong lịch sử của phần mềm mã nguồn mở
- Hiểu giấy phép là gì và các quyền nằm trong phạm vi của giấy phép
- Hiểu về cách các phần mềm hiện có có thể được sử dụng để tạo ra các tác phẩm phái sinh
- Hiểu về tính tương thích và không tương thích của giấy phép
- Hiểu về cấp phép kép và cấp phép có điều kiện
- Hiểu về hậu quả của hành vi vi phạm giấy phép
- Hiểu về các nguyên tắc của luật bản quyền và luật bằng sáng chế cũng như cách chúng bị ảnh hưởng bởi giấy phép phần mềm mã nguồn mở

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Định nghĩa phần mềm tự do của Tổ chức Phần mềm Tự do (FSF)
- Định nghĩa phần mềm mã nguồn mở của Sáng kiến Mã nguồn Mở (OSI)
- Giấy phép
- Hợp đồng
- Phần mềm công cộng

- Phần mềm miễn phí
- Phần mềm chia sẻ
- Người quản lý giấy phép
- Quyền sử dụng, sửa đổi và phân phối mã và phần mềm
- Các sản phẩm phái sinh và tái sử dụng mã
- Phần mềm mã nguồn đóng/độc quyền
- Phân phối có trả phí
- Phân phối phần mềm đã sửa đổi và chưa sửa đổi
- Lưu trữ phần mềm dưới dạng dịch vụ trả phí
- Khả năng tương thích giấy phép
- Cấp phép kép và đa cấp phép
- Cấp phép có điều kiện
- Bằng sáng chế phần mềm
- Cấp phép bằng sáng chế rõ ràng/ ngầm định

---

exam: "050" topic: "052" objective: "052.1" type: "lm-lesson" title: "052.1 Bài 1" menu: main: identifier: "lesson-050-052-052.1-1" parent: "objective-050-052-052.1" ---

Linux Professional Institute

Bài 1

Chứng chỉ:	Open Source Essentials
Phiên bản:	1.0
Chủ đề:	052 Giấy phép dành cho Phần mềm Mã nguồn Mở
Mục tiêu:	052.1 Các khái niệm về Giấy phép Phần mềm Mã nguồn Mở
Bài học:	1 trên 1

Giới thiệu

Bất kỳ một nhà phát triển phần mềm nào cũng sẽ đề cao việc một vấn đề đã có sẵn những phương án giải quyết hiệu quả. Nếu giải pháp đó lại sẵn có ở trên mạng, một phản xạ dễ hiểu sẽ là dùng thử — sao chép hoặc liên kết mã nguồn hiện có vào cơ sở mã của chính mình, kiểm tra nó và nếu nó hoạt động thì sẽ để luôn nó ở đó — và sau đó là quên nó đi.

Tuy vậy, chúng ta cũng phải thận trọng ở một số điểm. Thông thường, mã nguồn phần mềm sẵn có trên mạng internet đều được bảo vệ bởi *giấy phép* mã nguồn mở và tự do (FOSS). Ở chương này, chúng ta sẽ đi vào một số khái niệm cơ bản về FOSS từ góc nhìn pháp lý và lý do tại sao chúng ta không nên coi mã nguồn FOSS là một điều hiển nhiên, mà trái lại là phải tuân thủ chặt chẽ các điều kiện cấp phép của chúng.

Để làm cơ sở cho việc nắm rõ nghĩa vụ pháp lý của người dùng đối với phần mềm, hãy hiểu rằng giấy phép chi phối hầu hết tất cả các phần mềm. Giấy phép là một loại hợp đồng. Tất cả các phần mềm (bao gồm cả các trang web) phải được phân phối kèm theo phiên bản giấy phép bằng văn bản (thường được gọi là “điều khoản và điều kiện”). Một số chương trình và trang web sẽ yêu cầu

người dùng đánh dấu vào một ô thông báo xác nhận rằng họ đã đọc các điều khoản và điều kiện của phần mềm (và người dùng nên làm vậy — dù trong hầu hết mọi trường hợp chúng ta đều không hề thực hiện). Dù thế nào, người dùng cũng đều phải ngầm chấp nhận giấy phép khi họ sử dụng phần mềm.

Định nghĩa về Phần mềm Mã nguồn Mở và Phần mềm Tự do

Phần mềm mã nguồn mở và tự do đã xuất hiện từ khá lâu và các thuật ngữ này thường đi đôi với nhau. Tuy nhiên, có một số khác biệt giữa cái mà một số cá nhân tuyên bố là "phần mềm tự do" và ý nghĩa chính thức của *phần mềm tự do* cũng như *phần mềm mã nguồn mở*. Cảnh báo tiết lộ nội dung: "Mã nguồn mở" không chỉ có nghĩa là bất kỳ ai cũng có thể xem mã nguồn — định nghĩa này sẽ là của phần mềm "mã nguồn có sẵn". Phần mềm mã nguồn mở và tự do có thể làm được nhiều hơn thế.

Phần mềm mã nguồn mở và tự do trái ngược với các loại *độc quyền* hoặc *mã nguồn đóng* vì chúng không cung cấp tất cả các quyền tự do sẽ được thảo luận tới trong bài học này.

Có lẽ định nghĩa được trích dẫn nhiều nhất về phần mềm tự do là:

> Think of "free speech," not "free beer."
>
> *Dịch nghĩa: Coi nó là "tự do" như trong "tự do ngôn luận" chứ không phải trong "`vào cổng tự do`*".*
>
> *Bản dịch đã được thay đổi để phù hợp với văn phong và nhận thức trong tiếng Việt.
>
> — Richard Stallman, Selling Free Software

Chúng ta sẽ khám phá những điều thú vị trong tuyên bố trên ở các phần sau.

Tự do như trong Tự do Ngôn luận: Quyền Tự do thực sự dành cho Người dùng

Một mặt, các nhà phát triển có thể đưa ra một quyết định đơn giản là bỏ qua những phần thưởng cũng như những khó khăn khi bán phần mềm của họ, và chỉ cho đi mà không cần đền bù. Trong tiếng Anh, phần mềm này là "tự do" theo nghĩa là không ai phải trả tiền cho nó mà sẽ *tự do như ra vào một công viên công cộng**. Việc phân phối miễn phí này không liên quan gì đến các điều kiện cấp phép bổ sung và sẽ khiến người dùng luôn phải xem xét một cách kỹ lưỡng hơn (chẳng hạn như có thể họ đã không được cấp các quyền cần thiết để thay đổi hoặc tái phân phối phần mềm).

Mặt khác, các nhà phát triển cũng có thể lựa chọn biến phần mềm của họ thành "phần mềm tự do" theo định nghĩa của Stallman. Thuật ngữ này (được ông đặt ra vào những năm 1980) đề cập

đến các quyền tự do thiết yếu của người dùng có thể được tóm tắt như sau:

1. Quyền chạy phần mềm
2. Quyền nghiên cứu phần mềm
3. Quyền trao nó cho người khác (*tái phân phối*)
4. Quyền tái phân phối các bản sao của các phiên bản đã được sửa đổi

"Phần mềm tự do" trong định nghĩa này được ủng hộ bởi Tổ chức Phần mềm Tự do (FSF) nơi đã đưa ra thuật ngữ *copyleft* (không có ý nghĩa pháp lý mà thiên về thể hiện sự cân nhắc mang tính triết lý) để mô tả các đặc điểm của giấy phép trên phần mềm tự dọ.

Phần mềm Mã nguồn mở

Bên cạnh phong trào phần mềm tự do, cộng đồng những người ủng hộ mã nguồn mở đã nổi lên vào cuối những năm 1990 như một cách làm cho phần mềm tự do trở nên dễ hiểu và phổ biến hơn đối với những người nằm ngoài phong trào. Một tổ chức phi lợi nhuận là *Sáng kiến Mã nguồn mở* (OSI) đã được thành lập vào năm 1998 và Linus Torvalds - tác giả đầu tiên của hạt nhân Linux - đã ủng hộ khái niệm này. Sáng kiến Mã nguồn mở đã chính thức hóa *Định nghĩa Mã nguồn mở* bao gồm các tiêu chí sau:

1. *Tái phân phối tự do*: Người ta được tự do quyết định cách tái phân phối chương trình dù là miễn phí hay là để bán, miễn là không yêu cầu phí bản quyền hoặc phí giấy phép. Quyền tự do này bao gồm việc kết hợp chương trình này vào một chương trình khác.
2. *Tính khả dụng của mã nguồn*, có thể là trực tuyến hoặc được cung cấp cùng với phần mềm.
3. Cho phép sáng tạo và phân phối các *sản phẩm phái sinh* và bản sửa đổi.
4. *Tính toàn vẹn mã nguồn của tác giả*: Việc sửa đổi có thể bị hạn chế nếu người nhận được phép sửa đổi chương trình thông qua các bản vá tại thời điểm xây dựng và phân phối các bản vá này cùng với mã nguồn. Việc phân phối phần mềm *được xây dựng* từ mã nguồn đã qua sửa đổi có thể là không bị hạn chế.
5. *Không phân biệt đối xử với cá nhân hoặc nhóm*: Ví dụ: một giấy phép sử dụng phần mềm "chỉ dành cho giáo viên" sẽ không đáp ứng được Định nghĩa về Mã nguồn mở.
6. *Không phân biệt đối xử giữa các mục đích*: Ví dụ: không hạn chế mục đích thương mại.
7. *Phân phối giấy phép*: Bất cứ ai nhận chương trình đều sẽ có giấy phép gốc giống nhau.
8. *Giấy phép không được dành riêng cho một sản phẩm.*
9. *Giấy phép không được hạn chế phần mềm khác*: Ví dụ: phần mềm khác đi kèm với phần mềm này có thể có một giấy phép khác.

10. *Giấy phép phải trung lập trên phương diện công nghệ.*

Tính Tự do và Mã nguồn mở

Tổ chức Phần mềm Tự do (FSF) đã không ủng hộ thuật ngữ "mã nguồn mở" và nhấn mạnh rằng nó đã lấp đi mục tiêu chính là tính tự do. Vì vậy, trong khi những người ủng hộ phần mềm tự do và phần mềm mã nguồn mở dường như đang theo đuổi và ủng hộ cùng một khái niệm thì các phong trào lại có những động cơ riêng biệt. Đơn giản mà nói, những người ủng hộ phần mềm tự do nhấn mạnh đến quyền của các nhà phát triển và người dùng, trong khi những người ủng hộ mã nguồn mở lại ưu tiên việc sử dụng rộng rãi cũng như thành công của phần mềm.

Có khá nhiều phần mềm tự do đạt đủ điều kiện để được coi là mã nguồn mở; trong khi đó, có nhiều giấy phép cũng được coi là mã nguồn mở (được OSI phê duyệt) nhưng lại không tự do theo như FSF.

Bởi sự khác biệt giữa các mục tiêu và động cơ liên quan đến mã nguồn mở và tự do nhiều hơn là nội dung của các giấy phép, và vì cả hai thuật ngữ vẫn được sử dụng thường xuyên, nhiều người ủng hộ tham chiếu cả hai định nghĩa này bằng cách sử dụng các cụm từ là FOSS (Free and Open Source Software) hoặc FLOSS (*Free/Libre and Open Source Software*). Thuật ngữ “libre” được dùng để chỉ sự tự do trong nhóm ngôn ngữ Rôman.

Các loại Phần mềm Tự do tính phí khác

Ngoài các danh mục rộng rãi về FOSS và phần mềm độc quyền, chúng ta còn có một loạt các chiến lược phân phối khác. Một trong số các chiến lược này là:

Phần mềm chia sẻ (Shareware)

Thuật ngữ này thường đề cập đến các phần mềm độc quyền có mục đích thương mại nhưng có thể được sử dụng miễn phí với chức năng hạn chế cho đến khi người dùng quyết định mua phiên bản đầy đủ.

Phần mềm miễn phí (Freeware)

Thuật ngữ này mô tả phần mềm được phân phối miễn phí và không có giới hạn về việc sử dụng, nhưng cũng không nhất thiết phải tuân theo định nghĩa chính thức về phần mềm tự do. Phần mềm miễn phí trong nhiều trường hợp đều là phần mềm độc quyền và mã nguồn thường không được phát hành.

Phần mềm mã nguồn có sẵn (Source available software)

Đôi khi các nhà phát triển phần mềm độc quyền sẽ cung cấp mã nguồn của họ (ví dụ như nhằm mục đích tạo điều kiện cho việc báo cáo lỗi) nhưng lại áp đặt việc mua lại giấy phép độc quyền

như một điều kiện để sử dụng mã nguồn trong các dự án khác. Không nên nhầm lẫn tính khả dụng có các giới hạn nghiêm ngặt này với phần mềm mã nguồn mở đích thực.

Phần mềm mã nguồn chia sẻ (Shared source software)

Thuật ngữ này được Microsoft giới thiệu vào năm 2001 khi công ty này quyết định cung cấp một số mã nguồn phần mềm của họ lên mạng để nghiên cứu và thử nghiệm. Đừng nhầm lẫn định nghĩa có phạm vi rất hẹp này với phần mềm chia sẻ hoặc phần mềm mã nguồn có sẵn.

Phần mềm thuộc phạm vi công cộng (Public-domain software)

Đây là phần mềm mà các tác giả đã từ bỏ mọi quyền bản quyền. Định nghĩa này không áp dụng ở tất cả các khu vực pháp lý (đặc biệt với những khu vực mà tác giả được cấp quyền "droit d'auteur`" hoặc “quyền tác giả” như ở Pháp hoặc Đức). Các giấy phép như “Unlicense” đã được giới thiệu nhắm tới tác dụng tương tự. Hơn nữa, phần mềm cũng có thể thuộc phạm vi công cộng khi hết thời hạn bản quyền.

Các nguyên tắc của Luật Bản quyền và những ảnh hưởng đến từ Giấy phép Phần mềm Mã nguồn mở

Điều đầu tiên và quan trọng nhất: Nếu không có sẵn thông tin giấy phép cho một tệp hoặc một dự án mã nguồn nhất định, chúng ta không thể mặc định rằng tệp hoặc dự án đó không được bảo vệ bản quyền. Thực tế thì ngược lại, ít nhất là vì hầu hết các quốc gia trên thế giới đều đã ký kết *Công ước Berne* kể từ năm 1887.

Trong công ước này, các quốc gia ký kết đã đồng ý rằng một tác phẩm văn học hoặc nghệ thuật sẽ lập tức được bảo vệ bản quyền ngay khi nó bắt đầu tồn tại (hay nói cách khác là ngay khi nó được "định hình" để trở thành một phương tiện). Điều đó có nghĩa là tác giả sẽ không phải đăng ký hoặc xin bản quyền. Họ vẫn có thể làm việc này ở một số quốc gia và việc đăng ký có thể sẽ trở nên cần thiết khi đối mặt với các hành vi vi phạm ở một số khu vực pháp lý bao gồm cả Hoa Kỳ.

Ngoài ra, các bên ký kết đều đồng ý tôn trọng bản quyền của bất kỳ tác giả nào thuộc về một quốc gia ký kết khác; tính đến tháng 11 năm 2022, số lượng các quốc gia này chiếm tới 181 trong tổng số 195 quốc gia trên thế giới.

Tuy nhiên, không phải bất cứ thứ gì được tạo ra cũng đều sẽ được bảo vệ bản quyền. Để đủ điều kiện để được coi là một *tác phẩm* được bảo vệ bản quyền, nó cần phải đáp ứng một số tiêu chí cơ bản. Ví dụ như các sự kiện và ý tưởng không thể được bảo vệ bản quyền, nhưng một văn bản lý giải một ý tưởng lại có thể được bảo vệ nếu nó đạt đến một mức độ *nguyên bản* nhất định. Mức độ nguyên bản ở từng nơi trên thế giới có thể sẽ khác nhau, nhưng rào cản bảo hộ sẽ rất thấp trong nhiều trường hợp. Nhiều khu vực pháp lý hiện đang xem xét mức độ mà trí tuệ nhân tạo có thể được sử dụng để tạo ra một tác phẩm xứng đáng với cái được gọi là tính nguyên bản và từ đó cũng

xứng đáng được bảo vệ bản quyền.

Tùy thuộc vào khu vực pháp lý, các chương trình máy tính sẽ được bảo vệ quyền dưới dạng các tác phẩm văn học: tức là bản quyền không áp dụng cho ý tưởng hoặc thuật toán mà áp dụng cho việc triển khai của nó trong mã nguồn.

Bản quyền trao cho tác giả các quyền độc quyền (bên cạnh những quyền khác) để sao chép, sửa đổi, cấp phép lại, phân phối và phát hành tác phẩm. Việc tiếp nhận tác phẩm là miễn phí; vì vậy, người ta không cần giấy phép để đọc một cuốn sách hay nghe một bài hát trên radio, miễn là việc đó không yêu cầu phải tạo ra một bản sao vĩnh viễn.

Bởi quyền của tác giả ra đời mà không cần phải đăng ký nên bất kỳ ai muốn sao chép, sửa đổi, cấp phép lại, phân phối hoặc phát hành tác phẩm của tác giả khác đều phải xin phép trước. Đây là nơi các giấy phép phát huy tác dụng như những hợp đồng giữa tác giả của tác phẩm và những người muốn thực hiện một số quyền độc quyền của tác giả.

Giấy phép FOSS có thể được cung cấp cho bất kỳ ai mà không cần trả phí. Bất cứ ai cũng có thể tạo giấy phép của riêng mình và đăng ký để được OSI hoặc FSF phê duyệt. Tuy nhiên, việc sử dụng giấy phép FOSS sẵn có rất được khuyến khích vì nó đã được đại chúng chấp nhận; hơn thế nữa, nội dung của giấy phép và các nghĩa vụ của nó cũng đã trở nên quen thuộc đối với những người sử dụng phần mềm được trang bị đầy đủ kiến thức. Trên thực tế, chỉ một số ít trong số vô vàn các giấy phép được FSF hoặc OSI phê duyệt là được sử dụng phổ biến.

Nguyên tắc của Luật Sáng chế

Ngược lại với bản quyền (không bảo vệ các ý tưởng), bằng sáng chế sẽ bảo vệ các phát minh (ý tưởng) mà không (hoặc chưa) cần ý tưởng của chúng phải được "định hình" thông qua máy móc hay các quy trình. Một điểm khác biệt nữa là các nhà phát minh sẽ phải nộp đơn xin cấp bằng sáng chế một cách rõ ràng và đăng ký chúng với cơ quan cấp bằng sáng chế của quốc gia nơi yêu cầu bảo hộ.

Để tránh đi quá sâu vào luật sáng chế và các yêu cầu của nó, chúng ta sẽ chỉ tập trung vào một vấn đề trọng tâm: việc bảo vệ bằng sáng chế yêu cầu (bên cạnh các tiêu chí khác) một ý tưởng có một khía cạnh kỹ thuật nhất định. Từ trước tới nay, những ý tưởng về một công thức nấu ăn mới hoặc một trò chơi cờ bàn (board game) mới không đủ điều kiện để được bảo hộ bằng sáng chế; trong khi đó, các ý tưởng về một chiếc máy nấu ăn mới hoặc một máy chơi trò chơi mới lại có thể có đủ điều kiện để được bảo hộ.

Trong bối cảnh của lập trình máy tính, câu hỏi đặt ra là liệu phần mềm có được bảo hộ bởi bằng sáng chế hay không. Điều này phụ thuộc vào cả vùng pháp lý và các ứng dụng cụ thể. Ví dụ, ở Đức, các chương trình máy tính như vậy thường không được bảo hộ bởi bằng sáng chế. Tuy nhiên, nếu

các chương trình được kết hợp với một đối tượng vật lý — ví dụ như khi phần mềm điều khiển hệ thống phanh tự động trong ô tô — thì nó có thể yêu cầu bảo hộ bằng sáng chế cho ứng dụng bao gồm thành phần vật lý của phanh chứ không phải phần mềm và nhờ vậy có thể đạt đủ điều kiện để được bảo hộ.

Ở các khu vực pháp lý khác (chẳng hạn như Hoa Kỳ), bằng sáng chế có thể được cấp cho các chương trình máy tính như vậy, tùy thuộc vào sự phát triển của án lệ.

Khái niệm về bảo hộ với bằng sáng chế cần được ghi nhớ khi thực hiện cấp phép FOSS. Một số giấy phép (chẳng hạn như GPLv3) cho phép cấp bằng sáng chế về phần mềm FOSS bằng cách cấp quyền rõ ràng cho việc sử dụng phần mềm. Ngược lại, một số giấy phép thậm chí còn không đề cập đến các bằng sáng chế (chẳng hạn như giấy phép Điều khoản BSD-3) và các giấy phép khác lại loại trừ chúng một cách rõ ràng (chẳng hạn như các giấy phép Tài sản Sáng tạo Công cộng - Creative Commons). Tuy nhiên, trong một số trường hợp nhất định, việc cấp bằng sáng chế có thể được ám chỉ trong giấy phép hoặc có thể được ghi vào giấy phép.

NOTE Các giấy phép FOSS khác nhau sẽ được đề cập đến trong các bài học tiếp theo.

Đặc biệt là khi xử lý phần mềm nhúng (như phần mềm trong thiết bị âm thanh), chúng ta cần đặc biệt chú ý đến các bằng sáng chế liên quan đến phần mềm (ví dụ như tiến hành kiểm tra bằng sáng chế của tác giả phần mềm).

Hợp đồng cấp phép

Như đã đề cập từ trước, giấy phép là hợp đồng giữa tác giả của tác phẩm và người muốn thực hiện một số quyền độc quyền của tác giả. Một số giấy phép FOSS mở rộng hơn (chẳng hạn như Giấy phép Công cộng GNU phiên bản 2 và phiên bản 3) sẽ bao gồm cả các điều khoản về việc chấm dứt hợp đồng. Các giấy phép FOSS luôn bao gồm các khoản cấp quyền liên quan đến các quyền tự do trung tâm. Thông thường, các quyền sau đây sẽ được nêu rõ hoặc có thể được ngầm hiểu trong văn bản giấy phép:

> ...quyền sử dụng, sao chép, sửa đổi, hợp nhất, phát hành, phân phối, cấp phép lại và/hoặc bán bản sao của Phần mềm...
>
> — MIT license

Người quản lý giấy phép là những cá nhân — hoặc trong nhiều trường hợp là các tổ chức (như FSF) — quản lý các phiên bản của giấy phép. Ví dụ: mục 9 của GPLv2 tuyên bố FSF là người quản lý giấy phép:

> Tổ chức Phần mềm Tự do đôi khi có thể phát hành các phiên bản sửa đổi và/hoặc phiên bản mới của Giấy phép Công cộng Chung. Các phiên bản mới như vậy sẽ có tinh thần tương tự

> như phiên bản hiện tại nhưng có thể có các chi tiết khác nhau để giải quyết các vấn đề hoặc mối quan ngại mới.
>
> — Giấy phép Công cộng Chung GNU phiên bản 2

Một số người quản lý giấy phép cũng sẽ phát hành các câu hỏi thường gặp (FAQs) để giúp trả lời các câu hỏi liên quan đến giấy phép. Những câu hỏi này có thể sẽ rất hữu ích trong việc trao đổi giữa người được cấp phép và người cấp phép.

Phân phối

Việc phân phối phần mềm (đặc biệt là ở dạng mã nhị phân hoặc mã nguồn) là khía cạnh trung tâm của hầu hết các giấy phép FOSS vì việc chỉ sử dụng phần mềm FOSS thường không kích hoạt bất kỳ nghĩa vụ cấp phép nào:

> Các hoạt động khác ngoài việc sao chép, phân phối và sửa đổi không được bao gồm trong Giấy phép này mà nằm ngoài phạm vi của nó. Hành vi chạy Chương trình không bị hạn chế...
>
> — Giấy phép Công cộng Chung GNU phiên bản 2

Hầu hết các nghĩa vụ cấp phép chỉ phát sinh khi phân phối — tức là chuyển một bản sao đã qua sửa đổi hoặc chưa qua sửa đổi, có thể là trên phương tiện hữu hình như CD hoặc thông qua việc tải xuống. Trong một số trường hợp, các điều kiện cấp phép cũng sẽ được kích hoạt nếu phần mềm đang chạy trên một máy chủ (tức là chưa bao giờ phân phối mã nguồn) trong khi người dùng tương tác với phần mềm.

Việc phân phối phần mềm thực thi có thể sẽ yêu cầu phân phối mã nguồn, văn bản giấy phép và thông báo bản quyền, tùy thuộc vào loại giấy phép FOSS. Một số giấy phép sẽ yêu cầu phải đưa thông báo sửa đổi vào mã nguồn nếu mã sửa đổi đang được phân phối.

Việc phân phối cũng có thể kích hoạt các hiệu ứng copyleft — tức là khi phân phối phần mềm được cấp phép GPLv3 đã được sửa đổi, mã nguồn của toàn bộ phần mềm đã sửa đổi có thể sẽ phải được phát hành theo GPLv3.

Mặc dù FOSS có thể được bán nhưng nó lại thường không bị áp đặt phí cấp phép. Điều này có nghĩa là một ai đó có thể bán phần mềm theo giấy phép GPLv3 dưới dạng nhị phân, nhưng vì mã nguồn đang sẵn có (hoặc phải được cung cấp) nên những người quan tâm đến phần mềm luôn có thể chọn lấy các nguồn (có thể là từ một người khác mà không mất phí) và xây dựng phần mềm từ các nguồn đó.

Sản phẩm phái sinh

Khi một nhóm nhà phát triển kết hợp mã từ dự án của người khác vào dự án của riêng họ, kết quả của nó có thể là một sản phẩm *dẫn xuất* hoặc *phái sinh*. Các chi tiết khác nhau sẽ tùy theo từng dự án và từ giấy phép này sang giấy phép khác, và bởi vì các chi tiết này đòi hỏi tính tinh vi ở một mức nhất đinh với các kỹ thuật phát triển phần mềm, có thể nói là các nhà phát triển bắt buộc phải đảm bảo rằng phương thức kết hợp mã mà họ sử dụng tuân thủ theo đúng giấy phép.

Vấn đề quan trọng nhất liên quan đến sản phẩm phái sinh là đối với một số giấy phép (như GPL), sản phẩm phái sinh phải được phát hành theo cùng một giấy phép. Những giấy phép như vậy được gọi là giấy phép *tương hỗ* (reciprocal).

Ý nghĩa thực tế trước mắt của một yêu cầu tương hỗ là ở chỗ, nếu chúng ta sử dụng mã GPL trong dự án của riêng mình theo cách làm cho mã của mình trở thành một nguồn phái sinh, chúng ta sẽ phải tiết lộ mã nguồn của mình và để người khác xây dựng sản phẩm từ đó. Yêu cầu cấp phép này được một số nhà phát triển rất yêu thích bởi họ muốn khuyến khích nhiều người sử dụng giấy phép miễn phí hơn. Tuy nhiên, giấy phép sẽ làm giảm đi sự hấp dẫn của GPL đối với một số nhà phát triển muốn sử dụng mã tự do.

Nhiều giấy phép mã nguồn mở và tự do khác không áp đặt yêu cầu đó. Chúng có xu hướng được gọi là giấy phép *linh hoạt* (permissive).

Hậu quả của việc vi phạm Giấy phép

Nếu tại thời điểm phân phối các điều kiện cấp phép không được đáp ứng (ví dụ: nếu phần mềm được cấp phép GPLv3 được phân phối dưới dạng nhị phân không kèm theo một mã nguồn), hợp đồng cấp phép sẽ bị vi phạm. Hậu quả của việc vi phạm giấy phép sẽ phụ thuộc vào giấy phép. Ví dụ: vi phạm giấy phép GPLv3 có thể dẫn đến việc chấm dứt giấy phép. Bất kỳ hành động nào khác đều sẽ cần có sự cho phép của tác giả (ví dụ như nếu phân phối phần mềm thì sẽ cấu thành hành vi vi phạm bản quyền).

Nếu một công ty đưa phần mềm được cấp phép GPLv3 vào một sản phẩm và vi phạm giấy phép thì khiếu nại có thể được đưa ra để yêu cầu công ty thu hồi sản phẩm của họ. Việc cố ý vi phạm bản quyền thậm chí có thể dẫn đến các cáo buộc hình sự.

Một số giấy phép sẽ có các điều khoản cụ thể cho phép người được cấp phép khắc phục vi phạm trong vòng 30 ngày kể từ ngày thông báo. Nếu người phân phối phần mềm tuân thủ theo giấy phép trong khoảng thời gian này thì giấy phép sẽ được khôi phục.

Khả năng tương thích và không tương thích của Giấy phép

Các dự án phần mềm lớn thường bao gồm các phần mềm tuân theo các giấy phép khác nhau, mỗi giấy phép sẽ chỉ rõ các yêu cầu riêng của nó. Những dự án như vậy có thể gặp phải trở ngại khi sử dụng các phần mềm yêu cầu giấy phép của nó được sử dụng trên các sản phẩm phái sinh. Điều này là do các điều khoản cấp phép của các giấy phép copyleft đó có thể sẽ khác nhau dẫn đến việc không tương thích. Việc phát hành hoặc phân phối một dự án phần mềm tích hợp các thành phần theo các giấy phép copyleft khác nhau có thể sẽ không thể thực hiện mà không vi phạm một trong các giấy phép.

Một số giấy phép copyleft sẽ liệt kê rõ ràng các giấy phép tương thích để giúp dễ dàng sử dụng thành phần được cấp phép copyleft theo một giấy phép copyleft khác.

Hầu hết các giấy phép linh hoạt đều sẽ tương thích với các giấy phép khác. Ví dụ: các thành phần được MIT cấp phép có thể được sử dụng trong dự án được cấp phép GPLv3 mà không gặp phải rủi ro vi phạm cấp phép. Tuy nhiên, trong mọi trường hợp, một thành phần được cấp phép GPLv3 sẽ không thể được sử dụng trong dự án được MIT cấp phép mà không vi phạm các điều khoản của GPLv3. Do đó, khả năng tương thích không phải lúc nào cũng sẽ hoạt động theo cả hai chiều.

Trước khi một dự án phần mềm được đưa ra thị trường, các nhà phát triển và cố vấn pháp lý của họ nên tiến hành kiểm tra kỹ lưỡng khả năng tương thích của giấy phép để tránh vi phạm giấy phép. Việc quản lý tuân thủ mã nguồn mở nên được tích hợp vào các giai đoạn đầu của quy trình phát triển phần mềm để tránh sự chậm trễ trong việc phân phối phần mềm (ví dụ như do các vấn đề không tương thích với giấy phép). Hãy nghiên cứu kỹ mã nguồn để tìm các giấy phép hiện hành (ví dụ như bằng cách sử dụng các công cụ quét phần mềm) và kiểm tra xem liệu tất cả các điều kiện cấp phép có được đáp ứng hay không.

Cấp phép kép và Đa giấy phép

Một số phần mềm sẽ có thể có sẵn theo nhiều giấy phép. Ví dụ: người cấp phép có thể chọn *cấp phép kép* cho dự án của họ theo cả giấy phép copyleft như GPLv3 và một giấy phép độc quyền. Ví dụ: giấy phép độc quyền có thể được yêu cầu nếu người được cấp phép tiềm năng kết hợp mã vào sản phẩm độc quyền của riêng họ. Mỗi nhà phát triển sẽ xác định xem họ có thể tuân thủ các điều kiện GPLv3 hay không hay sẽ phải có giấy phép độc quyền và rất có thể là phải trả phí cấp phép.

Như đã chỉ ra trước đó, một số phần mềm có thể bao gồm các thành phần tuân theo nhiều giấy phép khác nhau với các điều kiện cấp phép khác nhau. Mặc dù hầu hết các giấy phép thường không gây ra tình trạng không tương thích nhưng các giấy phép khác nhau có thể có các yêu cầu khác nhau đối với các phần khác nhau của phần mềm.

Bài tập Hướng dẫn

1. FOSS là cụm viết tắt của khái niệm nào?

2. Điều nào sau đây rõ ràng thuộc về các quyền tự do thiết yếu của người dùng đối với phần mềm miễn phí?

Chạy phần mềm	
Nghiên cứu phần mềm	
Sao chép phần mềm	
Thay đổi phần mềm	
Phát hành phần mềm	
Tái phân phối phần mềm	

3. Có phải mã nguồn được tìm thấy trên internet mà không có thông tin cấp phép thì bất kỳ ai cũng có thể sửa đổi và phân phối không? Hãy giải thích vì sao.

4. Phần mềm có thể được cấp bằng sáng chế không?

Có	
Không	
Tuỳ từng trường hợp	

5. Người quản lý giấy phép là ai?

Giống như người cấp phép	
Người có thể đề xuất các phiên bản giấy phép trong tương lai	
Người có thể chấm dứt giấy phép	
Người quản lý phần mềm của máy bay	

6. Khả năng tương thích của giấy phép có phải luôn luôn hoạt động theo cả hai chiều hay không?

Có, nếu hai giấy phép tương thích thì việc A được đưa vào B hay B được đưa vào A không thành vấn đề.	
Không, trong một số trường hợp, giấy phép A có thể được đưa vào dự án theo giấy phép B, nhưng giấy phép B có thể lại không được phép phân phối theo giấy phép A.	
Không, khả năng tương thích của giấy phép luôn luôn là một chiều.	

Bài tập Mở rộng

1. Giấy phép GPLv2 và LGPLv2.1 có tương thích không? Hãy đưa ra một lời giải thích ngắn gọn.

2. Phần mềm có thể được phát hành theo giấy phép Nội dung Mở (chẳng hạn như CC-BY) không?

Có, giấy phép CC có thể được áp dụng cho bất kỳ sản phẩm nào có bản quyền.	
Không, phần mềm chỉ có thể được bảo vệ bởi bằng sáng chế.	
Không, giấy phép CC không áp dụng cho phần mềm.	

3. Tại sao việc tái phân phối quan trọng đối với việc cấp phép FOSS?

Tóm tắt

Bài học này đã giới thiệu các khái niệm cơ bản về phần mềm mã nguồn mở và tự do cũng như các khái niệm cơ bản về bản quyền và bằng sáng chế. Nó giải thích một số nền tảng và lịch sử của cả phần mềm tự do và phần mềm mã nguồn mở, cũng như giúp phân biệt cả hai phần mềm này với các khái niệm cấp phép độc quyền. Định nghĩa Mã Nguồn Mở của OSI giúp phân loại các giấy phép thành các giấy phép mã nguồn mở.

Trong khi đã có hàng chục giấy phép FOSS tồn tại, bất cứ ai cũng đều có thể tự do đưa ra các giấy phép bổ sung.

Không nên đánh giá thấp khái niệm cấp phép: Giấy phép cung cấp các quyền xử lý phần mềm và các quyền này vốn chỉ dành riêng cho tác giả (khi không có giấy phép). Việc vi phạm giấy phép có thể dẫn đến tranh chấp pháp lý. Do đó, chúng ta nên tìm kiếm tư vấn pháp lý về việc tuân thủ giấy phép trước khi phần mềm được phân phối hoặc tích hợp vào mã của dự án khác.

Đáp án Bài tập Hướng dẫn

1. FOSS là cụm viết tắt của khái niệm nào?

   Free and Open Source Software (Phần mềm Mã nguồn mở và Tự do).

2. Điều nào sau đây rõ ràng thuộc về các quyền tự do thiết yếu của người dùng đối với phần mềm miễn phí?

Chạy phần mềm	X
Nghiên cứu phần mềm	X
Sao chép phần mềm	
Thay đổi phần mềm	X
Phát hành phần mềm	
Tái phân phối phần mềm	X

3. Có phải mã nguồn được tìm thấy trên internet mà không có thông tin cấp phép thì bất kỳ ai cũng có thể sửa đổi và phân phối không? Hãy giải thích vì sao.

   Không. Nếu đạt đến một mức độ nguyên bản nhất định, mã nguồn theo mặc định sẽ được bảo vệ bản quyền như một tác phẩm văn học. Vì việc sửa đổi và phân phối là quyền độc quyền của tác giả nên không ai ngoài tác giả có thể làm hoặc cho phép làm việc này.

4. Phần mềm có thể được cấp bằng sáng chế không?

Có	
Không	
Tuỳ từng trường hợp	X

5. Người quản lý giấy phép là ai?

Giống như người cấp phép	
Người có thể đề xuất các phiên bản giấy phép trong tương lai	X
Người có thể chấm dứt giấy phép	
Người quản lý phần mềm của máy bay	

6. Khả năng tương thích của giấy phép có phải luôn luôn hoạt động theo cả hai chiều hay không?

Có, nếu hai giấy phép tương thích thì việc A được đưa vào B hay B được đưa vào A không thành vấn đề.	
Không, trong một số trường hợp, giấy phép A có thể được đưa vào dự án theo giấy phép B, nhưng giấy phép B có thể lại không được phép phân phối theo giấy phép A.	X
Không, khả năng tương thích của giấy phép luôn luôn là một chiều.	

Đáp án Bài tập Mở rộng

1. Giấy phép GPLv2 và LGPLv2.1 có tương thích không? Hãy đưa ra một lời giải thích ngắn gọn.

   Nếu phần mềm A được cấp phép theo GPLv2 và phần mềm B được cấp phép theo LGPLv2.1 thì ta có thể sử dụng B trong A vì LGPLv2.1 có cho phép sử dụng theo các điều kiện của giấy phép GPLv2. Tuy nhiên, A không thể được sử dụng trong B theo các điều khoản của LGPLv2.1. Nếu A đang được sử dụng trong B thì toàn bộ phần mềm phải được cấp phép theo GPLv2; điều này là có thể vì LGPLv2.1 có cho phép sử dụng phần mềm theo GPLv2.0.

2. Phần mềm có thể được phát hành theo giấy phép Nội dung Mở (chẳng hạn như CC-BY) không?

Có, giấy phép CC có thể được áp dụng cho bất kỳ sản phẩm nào có bản quyền.	X
Không, phần mềm chỉ có thể được bảo vệ bởi bằng sáng chế.	
Không, giấy phép CC không áp dụng cho phần mềm.	

3. Tại sao việc tái phân phối quan trọng đối với việc cấp phép FOSS?

   Nhiều nghĩa vụ cấp phép FOSS sẽ được kích hoạt khi phân phối phần mềm. Nếu phần mềm không bao giờ được phân phối mà chỉ được thay đổi và sử dụng trong một hệ thống khép kín (chẳng hạn như một bộ phận của công ty) thì người ta có thể sử dụng phần mềm mà không cần tuân thủ các nghĩa vụ cấp phép.

052.2 Giấy phép Phần mềm Copyleft

Tham khảo các mục tiêu LPI

Open Source Essentials version 1.0, Exam 050, Objective 052.2

Khối lượng

3

Các lĩnh vực kiến thức chính

- Hiểu rõ khái niệm về giấy phép phần mềm copyleft
- Hiểu rõ các quyền được cấp bởi giấy phép phần mềm copyleft
- Hiểu rõ các nghĩa vụ do giấy phép phần mềm copyleft áp đặt
- Hiểu rõ các thuộc tính chính của các giấy phép phần mềm copyleft phổ biến
- Hiểu rõ tính tương thích của giấy phép phần mềm copyleft với các giấy phép phần mềm khác
- Nắm được kiến thức về các thuật ngữ "`giấy phép tương hỗ `" và "`giấy phép hạn chế `"

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Copyleft
- Phân phối
- Truyền tải
- Tivo hoá
- Giấy phép Công cộng GNU, phiên bản 2.0 (GPLv2)
- Giấy phép Công cộng GNU, phiên bản 3.0 (GPLv3)
- Giấy phép Công cộng GNU Ít hơn, Phiên bản 2 (LGPLv2)
- Giấy phép Công cộng GNU Ít hơn, Phiên bản 3 (LGPLv3)

- Giấy phép Công cộng GNU Affero, Phiên bản 3 (AGPLv3)
- Giấy phép Công cộng Eclipse (EPL), phiên bản 1.0
- Giấy phép Công cộng (EPL), phiên bản 2.0
- Giấy phép Công cộng Mozilla (MPL)

Linux Professional Institute

Bài 1

Chứng chỉ:	Open Source Essentials
Phiên bản:	1.0
Chủ đề:	052 Giấy phép dành cho Phần mềm Mã nguồn Mở
Mục tiêu:	052.1 Giấy phép Phần mềm Copyleft
Bài học:	1 trên 1

Giới thiệu

Tầm quan trọng của giấy phép đối với cả việc sử dụng lẫn phát triển phần mềm đã được nhắc tới ở bài trước. Do đó, không có gì đáng ngạc nhiên khi phần mềm tự do cũng được đặc tả bởi các phương pháp tiếp cận mới trong cấp phép ngay từ đầu: Các điều kiện cho việc sử dụng không hạn chế hoặc việc hợp tác phát triển phần mềm phải được định nghĩa một cách hợp pháp để có thể được bảo vệ và thực thi.

Nhận thức được rằng luật bản quyền được thể hiện một cách chặt chẽ trong hầu hết các hệ thống pháp luật trên toàn thế giới và không thể bị nghi ngờ hay thay thế, ngay từ những năm 1980, các nhà phát triển phần mềm đã áp dụng một cách tiếp cận nhằm tôn trọng các quy định về bản quyền nhưng đồng thời cũng bổ sung vào các quy định mới nhấn mạnh về nguyên tắc “tự do”: *copyleft*.

Copyleft và Giấy phép Công cộng Chung GNU (GPL)

Richard Stallman, khi đó là một nhà phát triển nổi tiếng tại MIT, đã thành lập Dự án *GNU* vào năm 1983 để phát triển thứ mà ông coi là một hệ điều hành “tự do”. Rất nhanh sau đó mọi người đều

nhận ra rằng mã do dự án phát triển phải được bảo vệ về mặt pháp lý để nó không thể bị các nhà cung cấp thương mại chiếm lấy và từ đó trở thành "không tự do".

Do đó, Stallman đã thành lập tổ chức phi lợi nhuận *Tổ chức Phần mềm Tự do* (FSF) vào năm 1985. Tổ chức này tóm tắt sứ mệnh của mình trên trang web của họ như sau: "Tổ chức Phần mềm Tự do đang làm việc để đảm bảo quyền tự do cho người dùng máy tính bằng cách thúc đẩy sự phát triển và sử dụng phần mềm và tài liệu tự do..."

Công cụ trọng yếu đối với sứ mệnh này là một giấy phép một mặt tôn trọng các điều luật hiện hành (đặc biệt là luật bản quyền) và mặt khác thực hiện các ý tưởng tự do của chính nó một cách trong sạch về mặt pháp lý. Kết quả chính là phiên bản đầu tiên của *Giấy phép Công cộng Chung GNU* (GPLv1) ra đời vào năm 1989. Giấy phép này và nhiều bài viết liên quan — chẳng hạn như "Phần mềm Tự do là gì?" do Stallman viết vào năm 1992 — đã làm rõ động cơ và giá trị của các nhà phát triển phần mềm tự do - những người mà giờ đây cũng tự coi mình là một "phong trào".

Cốt lõi của lập trình vẫn được hình thành từ "bốn quyền tự do thiết yếu" do Stallman đưa ra trong bài viết nói trên và được đánh số bắt đầu từ 0:

> - Quyền tự do chạy chương trình theo ý muốn, với bất kỳ mục đích nào (quyền tự do số 0).
> - Quyền tự do nghiên cứu cách chương trình hoạt động và thay đổi nó để máy tính của bạn hoạt động theo ý muốn (quyền tự do số 1). Quyền truy cập vào mã nguồn là điều kiện tiên quyết cho việc này.
> - Quyền tự do tái phân phối các bản sao để bạn có thể giúp đỡ những người khác (quyền tự do số 2).
> - Quyền tự do phân phối bản sao các phiên bản đã sửa đổi của bạn cho người khác (quyền tự do số 3). Bằng cách này, bạn có thể mang đến lợi ích cho cả cộng đồng từ những thay đổi của mình. Quyền truy cập vào mã nguồn là điều kiện tiên quyết cho việc này.
>
> — Richard Stallman, Phần mềm Tự do là gì?

Không giống như giấy phép dành cho các sản phẩm thương mại sẽ đặt ra các hạn chế về việc sử dụng ngay từ đầu, phần mềm tự do hướng tới sự tự do tối đa cho người dùng và các nhà phát triển.

Lời mở đầu của GNU GPLv1 tóm tắt điều này như sau:

> Cụ thể, Giấy phép Công cộng Chung được thiết kế để đảm bảo rằng bạn có quyền tự do cho hoặc bán các bản sao của phần mềm tự do, rằng bạn có thể nhận được mã nguồn hoặc lấy nó nếu muốn, và rằng bạn có thể thay đổi phần mềm hoặc sử dụng các phần của nó trong các chương trình tự do mới; và rằng bạn biết bạn có thể làm được những điều này.

> — Giấy phép Công cộng Chung GNU, phiên bản 1

Điều này có nghĩa là mọi người đều có quyền sử dụng, phân phối và sửa đổi phần mềm theo GPL mà không bị hạn chế (điều này có thể thực hiện được vì mã nguồn hoàn toàn công khai — hay nói cách khác là "mở") và lần lượt phân phối các bảnsửa đổi. Thậm chí bạn có thể yêu cầu trả phí khi chuyển nhượng phần mềm:

> Trên thực tế, chúng tôi khuyến khích những người tái phân phối phần mềm tự do tính phí nhiều nhất như họ mong muốn hoặc có thể. Nếu một giấy phép không cho phép người dùng tạo bản sao và bán chúng thì đó là giấy phép không tự do.... Các chương trình tự do đôi khi sẽ được phân phối miễn phí và đôi khi là với một mức giá đáng kể. Thường thì cùng một chương trình có thể được cung cấp theo cả hai cách từ những nơi khác nhau. Chương trình sẽ vẫn là tự do bất kể giá cả của nó, vì người dùng có quyền tự do sử dụng nó.
>
> — Tổ chức Phần mềm Tự do, Bán Phần mềm Tự do

Nhưng nếu các quyền tự do có phạm vi rộng như vậy thì phần mềm sẽ được bảo vệ theo giấy phép này ở mức độ nào (chẳng hạn như việc kết hợp vào các sản phẩm độc quyền)?

Đây là vai trò của nguyên tắc copyleft đã được đề cập trước đó mà GPL đã áp dụng trong phiên bản 1 ngay cả khi thuật ngữ này chưa được xác định một cách rõ ràng:

> Bạn không được sao chép, sửa đổi, cấp phép lại, phân phối hoặc chuyển giao Chương trình trừ khi được quy định rõ ràng theo Giấy phép Công cộng này.
>
> — Giấy phép Công cộng Chung GNU phiên bản 1

Điều này có nghĩa là tất cả các quyền tự do này được liên kết với điều kiện là người dùng phải bảo toàn các quyền tự do này trong mọi việc họ làm đối với phần mềm.

Do đó, Copyleft không chỉ đảm bảo các quyền tự do mà còn yêu cầu mọi người dùng phải cấp các quyền tự do này cho người khác. Điều này có thể đạt được bằng cách quy định rằng phần mềm phải tuân theo giấy phép copyleft (chẳng hạn như GPLv1 đời đầu), chỉ có thể được sửa đổi và tái phân phối nếu các sửa đổi đó được xuất bản dưới cùng các điều kiện - tức là theo cùng một giấy phép.

Do đó, lý tưởng của phần mềm tự do, cụ thể là việc sử dụng chung và phát triển phần mềm xa hơn, được ưu tiên hơn các nhu cầu cá nhân liên quan đến phần mềm. Nguyên tắc có đi có lại rất quan trọng: Những người sử dụng các quyền tự do cũng phải trao đi những quyền tự do đó. Do đó, các giấy phép Copyleft thường được gọi là giấy phép *tương hỗ*.

Cách tiếp cận hoàn toàn mới này đối với giấy phép phần mềm đã được chứng minh là hợp lý về mặt pháp lý và có thể thực hiện được trong phiên bản 1 của GPL; do đó, GPL chỉ trải qua hai lần

sửa đổi lớn trong gần 40 năm phát triển của thị trường CNTT hiện đại.

GPLv2 và GPLv3

Năm 1991, Tổ chức Phần mềm Tự do đã giới thiệu phiên bản 2 của Giấy phép Công cộng Chung GNU (GPLv2). Giấy phép này đã trở thành giấy phép phổ biến nhất cho các dự án phần mềm tự do trong nhiều năm. Ví dụ: lõi của hệ điều hành Linux ngày nay vẫn được cấp phép theo GPLv2.

So với phiên bản 1, phiên bản 2 chủ yếu liên quan đến các định nghĩa chính xác hơn để tránh bị mơ hồ. Ví dụ: phiên bản 2 giải thích chi tiết hơn nhiều ý nghĩa của "mã nguồn".

Một điều đáng quan tâm khác nữa là Mục 7 mới đã đặt ra nguyên tắc tự do và do đó hiệu lực của giấy phép là tuyệt đối và không cho phép bất kỳ một sự thỏa hiệp nào — ví dụ như việc tích hợp các thành phần ít tự do hơn vào phần mềm:

> Nếu bạn không thể đáp ứng đồng thời các nghĩa vụ của mình trong việc phân phối theo Giấy phép này và bất kỳ nghĩa vụ thích hợp nào khác thì hậu quả sẽ là bạn hoàn toàn không được phép phân phối Chương trình. Ví dụ: nếu một giấy phép bằng sáng chế không cho phép tái phân phối miễn phí bản quyền chương trình bởi tất cả những người nhận bản sao trực tiếp hoặc gián tiếp thông qua bạn thì cách duy nhất bạn có thể đáp ứng cả giấy phép đó và Giấy phép này là hạn chế hoàn toàn việc phân phối Chương trình.
>
> — Giấy phép Công cộng Chung GNU phiên bản 2

Phải đến 16 năm sau, vào năm 2007, FSF mới xuất bản một phiên bản mới của GPL để tính đến những cải tiến kỹ thuật — chẳng hạn như việc cung cấp các dịch vụ phần mềm qua internet — cũng như các vấn đề về tính tương thích với các giấy phép FOSS khác. Tuy nhiên, giấy phép vẫn duy trì tính ổn định về các tuyên bố cốt lõi và chỉ bổ sung thêm các chi tiết để làm rõ thêm. Chúng ta hãy xem xét kỹ hơn về một số bổ sung này.

Trong khi GPLv2 vẫn thường sử dụng khái niệm *phân phối* khi đề cập đến việc cung cấp phần mềm, GPLv3 lại chỉ rõ quy trình này bằng hai thuật ngữ mới: *truyền bá* (propagation) và *chuyển tải* (conveying). Lý do chính cho điều này là bởi thuật ngữ "phân phối" (distribution) được định nghĩa trong nhiều luật bản quyền trên toàn thế giới. Để tránh sự mơ hồ hoặc các xung đột có thể xảy ra, GPLv3 đã chọn các thuật ngữ mới này và định nghĩa chúng như sau:

> "Truyền bá" một sản phẩm có nghĩa là nếu làm bất cứ điều gì với tác phẩm đó trong khi không có sự cho phép thì bạn sẽ phải chịu trách nhiệm trực tiếp hoặc gián tiếp về hành vi vi phạm theo luật bản quyền hiện hành, ngoại trừ việc thực thi tác phẩm đó trên máy tính hoặc sửa đổi một bản sao riêng tư. Việc truyền bá bao gồm sao chép, phân phối (có hoặc không có sửa đổi), cung cấp cho công chúng và ở một số quốc gia còn bao gồm cả các hoạt động khác.

> "Chuyển tải" một sản phẩm có nghĩa là bất kỳ hình thức truyền bá nào cho phép các bên khác tạo hoặc nhận các bản sao. Việc chỉ tương tác với người dùng thông qua mạng máy tính mà không có sự chuyển giao bản sao thì không được coi là chuyển tải.
>
> — Giấy phép Công cộng Chung GNU phiên bản 3

Với số lượng đang ngày càng tăng lên một cách đáng kể của các sản phẩm phần mềm thương mại bị hạn chế phân phối bởi các nhà sản xuất thông qua các biện pháp kỹ thuật như mã đăng ký hoặc các thành phần phần cứng (*dongles*), đã có một số sáng kiến pháp lý quốc tế vào cuối những năm 1990 nhằm hình sự hóa việc lách luật của các biện pháp này. *Quản lý quyền kỹ thuật số* (DRM) (hay còn được những người phản đối gọi một cách ác ý là *quản lý hạn chế kỹ thuật số*) là một cái gai đối với FSF vì các biện pháp này về cơ bản mâu thuẫn với nhu cầu phân phối phần mềm tự do.

Để đáp lại, phiên bản 3 của GPL có chứa một phân đoạn tuyên bố rằng phần mềm tuân theo GPL sẽ không được sửa đổi khi tham chiếu đến các yêu cầu DRM pháp lý. Điều này cũng có nghĩa là phần mềm được cấp phép theo GPLv3 có thể sử dụng DRM nhưng những phần mềm khác cũng được phép lách các biện pháp đó.

Thuật ngữ *tivo hoá* (tivoization) cũng thường được sử dụng trong ngữ cảnh này. Từ này xuất hiện rõ ràng trong bản nháp đầu tiên của GPLv3 nhưng lại không được đưa vào phiên bản cuối cùng. Thuật ngữ này xuất phát từ công ty TiVo; công ty này đã sử dụng phần mềm được cấp phép GPLv2 trong máy ghi video kỹ thuật số của mình, nhưng đồng thời lại cũng ngăn chặn các phần mềm đã được sửa đổi khỏi việc được cài đặt và sử dụng trên thiết bị. Theo quan điểm của FSF, điều này mâu thuẫn với các nguyên tắc của GPL, và sau một số cuộc thảo luận, GPLv3 đã đưa điều này vào bằng một đoạn văn về cái gọi là *sản phẩm của người dùng*. Nó thường quy định rằng các sản phẩm sử dụng phần mềm được cấp phép GPLv3 cũng phải cung cấp thông tin về cách sửa đổi phần mềm này.

Một bổ sung nữa liên quan đến *bằng sáng chế* mà FSF về cơ bản đã bác bỏ đối với phần mềm vì lý do cản trở quyền tự do và đổi mới của chúng. Điều này đã được nêu trong phần mở đầu của GPLv3:

> Cuối cùng, chương trình nào cũng sẽ luôn bị đe dọa bởi các bằng sáng chế phần mềm. Các quốc gia không nên cho phép các bằng sáng chế hạn chế việc phát triển và sử dụng phần mềm trên các máy tính có mục đích chung; nhưng ở những nước làm như vậy, chúng tôi muốn tránh các nguy cơ đặc biệt mà bằng sáng chế áp đặt cho một chương trình tự do để có thể biến nó trở thành chương trình độc quyền. Để ngăn chặn điều này, GPL đảm bảo rằng các bằng sáng chế không thể được sử dụng để làm cho chương trình trở nên không tự do.
>
> — Giấy phép Công cộng Chung GNU phiên bản 3

Văn bản giấy phép cũng chứa một số đoạn về việc cho phép đưa mã theo bằng sáng chế thông qua

một "giấy phép bằng sáng chế không độc quyền, toàn cầu, miễn phí bản quyền" từ người cấp phép để bảo vệ người dùng mã đó khỏi các tranh chấp giữa người nắm giữ bằng sáng chế và người được cấp phép.

Giấy phép Công cộng Chung GNU Affero (AGPL)

Với sự gia tăng về khả năng truy cập và tốc độ của Internet, ngày càng có nhiều dịch vụ xuất hiện mà trong đó, phần mềm chỉ được cài đặt trên máy chủ của nhà cung cấp — *Nhà cung cấp dịch vụ ứng dụng* (ASP) — và khách hàng của họ sẽ tương tác với dịch vụ thông qua internet. Xu hướng này được gọi là *Phần mềm dưới dạng dịch vụ* (SaaS).

Trong những trường hợp như vậy, GPLv2 đã không cung cấp được một sự rõ ràng về việc liệu mã nguồn (có thể đã được nhà cung cấp sửa đổi) có nên được cung cấp hay không và bằng cách nào. Phiên bản 3 của GPL sẽ lấp được lỗ hổng này — hay còn được gọi là *lỗ hổng ASP* — bằng cách đề cập rõ ràng đến một giấy phép khác do FSF cấp năm 2007 trong mục 13: *Giấy phép Công cộng Chung GNU Affero* phiên bản 3 (GNU AGPLv3). Tên này được đặt theo tên của công ty Affero - nơi đã phát triển và xuất bản hai phiên bản đầu tiên của giấy phép này.

AGPLv3 này về cơ bản là tương ứng với GPLv3 nhưng có quy định rõ ràng về vấn đề ASP trong phần "tương tác mạng từ xa". Hơn nữa, cả hai giấy phép đều nêu rõ rằng chúng có thể được kết hợp với nhau mà không bị hạn chế.

Tóm lại, GNU AGPL là phần bổ sung bổ trợ cho GPL. AGPL mở rộng phạm vi của GPL bằng cách áp dụng copyleft cho phần mềm không còn được sử dụng trong cài đặt cục bộ mà chỉ ở dạng dịch vụ được truyền qua mạng.

Khả năng tương thích của Giấy phép Copyleft

Sự phát triển của phần mềm tự do được dựa trên sản phẩm của người khác — tức là tích hợp, sửa đổi và chia sẻ mã nguồn của những người khác. Nếu tất cả các phần của một phần mềm được sửa đổi hoặc mới được biên dịch đều có cùng một giấy phép copyleft (chẳng hạn như là GPLv3) thì điều này có thể thực hiện được mà không gặp bất kỳ vấn đề nào về pháp lý. Giấy phép chỉ đơn giản là yêu cầu kết quả được phân phối theo cùng một giấy phép.

Mọi thứ sẽ trở nên phức tạp hơn khi phần mềm có bao gồm các thành phần được cấp phép theo các giấy phép khác nhau. Chúng ta cần phải tính đến nhiều yếu tố trên phương diện này.

Các Sản phẩm Kết hợp và Phái sinh

Đôi khi phần mềm tự do được tạo ra trong những điều kiện rất khác biệt. Các thay đổi có thể là từ sửa lỗi đơn giản đến các dự án phức tạp với hàng triệu dòng mã. Bất kể là ở phạm vi nào, chúng ta

cũng đều sẽ có sự phân biệt cơ bản giữa hai loại sản phẩm khi nói đến vấn đề cấp phép: *phái sinh* và *kết hợp*.

Ví dụ: giả sử dự án phần mềm A bị thiếu một chức năng nhất định. Thay vì tự phát triển chức năng này từ đầu, sẽ hợp lý hơn khi người ta kết hợp mã từ một dự án B khác cung cấp chính xác chức năng này vào với A. Phần mềm từ B thậm chí không cần phải thay đổi cho việc này mà có thể chỉ cần được thêm vào A. Vì lý do này mà nó được gọi là một sản phẩm kết hợp. Nếu cả A và B đều có cùng giấy phép copyleft thì không có vấn đề gì đối với sản phẩm kết hợp này.

Nếu A và B có các giấy phép copyleft khác nhau, chúng ta sẽ cần phải thận trọng: Sự kết hợp giữa A và B đã là một sản phẩm riêng biệt chưa? Và nếu vậy thì nó có thể hoặc phải được cấp phép theo giấy phép nào? Hoặc có thể tránh được xung đột bằng cách đảm bảo rằng cả hai phần A và B vẫn tách biệt với các giấy phép tương ứng của chúng và không cấu thành một sản phẩm mới hay không?

Điều này sẽ càng trở nên khó khăn hơn với các sản phẩm phái sinh — tức là khi dự án A chỉ có thể sử dụng chức năng của B bằng cách đưa trực tiếp mã từ B vào mã của A. Việc tích hợp này sẽ tạo ra một sản phẩm phái sinh mới mà các thành phần của nó không thể tách rời được nữa.

Copyleft sẽ phát huy tác dụng đối với một sản phẩm phái sinh. Ví dụ: nếu A tuân theo giấy phép copyleft như GPLv3 thì sản phẩm phái sinh mới cũng sẽ phải tuân theo giấy phép GPLv3 theo nguyên tắc tương hỗ có qua có lại. Nhưng nếu B có giấy phép khác thì sao? Đây có phải là giấy phép copyleft và nó có tương thích với GPLv3 không? Hoặc, nếu đó là một loại giấy phép khác, B cũng có thể được phát hành theo một giấy phép khác — tức là được cấp lại — hay không? Hay thậm chí giấy phép của A và B liệu có loại trừ lẫn nhau trên phương diện phân loại hay không?

Mục đích của bài học này không phải là để liệt kê các sự kết hợp có thể xảy ra và các giải pháp khả thi. Điều quan trọng ở đây là phải minh họa được mức độ phức tạp của vấn đề xuất phát từ sự kết hợp của nhiều vấn đề rất khác nhau.

Về mặt kỹ thuật, điều đầu tiên chúng ta cần phải làm rõ là cách các thành phần khác nhau của phần mềm (trong ví dụ A và B) hoạt động cùng nhau như thế nào: Chúng có thể tách rời nhau hay có những quy trình mà việc thực thi của chúng không còn có thể được phân công rõ ràng cho phần này hoặc phần kia nữa?

Từ góc độ pháp lý, có nhiều câu hỏi được đặt ra về mối quan hệ giữa giấy phép của A và B. Chúng có tương thích với nhau không hay chỉ tương thích một phần/ một chiều? Có thể chọn cấp lại giấy phép hay không?

Ở đây chúng ta chỉ có thể hiểu sơ qua về mức độ phức tạp của những câu hỏi này chứ không thể giải quyết chúng. Những quyết định như vậy đòi hỏi phải có một nền tảng kiến thức pháp lý vững chắc. Do đó, các quyết định cấp phép cho các sản phẩm mới — đặc biệt là các sản phẩm kết hợp và

phái sinh — phải được đưa ra *sớm*, *cẩn thận* và luôn luôn là sau khi được *tư vấn pháp lý* một cách chi tiết.

Giấy phép Copyleft yếu hơn

Copyleft đã được chứng minh là cực kỳ mạnh mẽ trong nhiều thập kỷ qua, đặc biệt là ở dạng GPL trong phiên bản 2 và 3. Tuy nhiên, các yêu cầu kỹ thuật đặc biệt đã thúc đẩy FSF phản ứng bằng cách điều chỉnh các giấy phép của họ. Giấy phép Công cộng Chung GNU Affero là một ví dụ như vậy. Chúng ta sẽ thảo luận về các ví dụ khác trong các phần phụ sau.

Giấy phép Công cộng Chung Thu hẹp GNU (LGPL)

Một phương pháp thường được sử dụng trong phát triển phần mềm là sử dụng các mô-đun nhỏ cho các tác vụ tiêu chuẩn (chẳng hạn như mở hoặc lưu tệp). Các mô-đun này — hoặc tập hợp các mô-đun như vậy — được gọi là *thư viện phần mềm*. Chúng thường không phải là các ứng dụng thực thi độc lập mà là các quy trình lặp lại mà ứng dụng thực tế tích hợp theo yêu cầu. Quá trình tích hợp được gọi là *liên kết* và có thể được thực hiện với hai hình thức khác nhau.

Với *thư viện tĩnh*, ứng dụng thực tế (thông qua các chương trình phụ trợ và các bước trung gian) sẽ tích hợp chặt chẽ mã của các mô-đun vào tệp thực thi của ứng dụng. Mặt khác, với *thư viện động*, ứng dụng sẽ chỉ tích hợp một mô-đun khi được yêu cầu trong thời gian chạy và tải mô-đun đó vào bộ nhớ làm việc.

Điều này đặt ra một câu hỏi liên quan đến copyleft và vấn đề cấp phép: Ý nghĩa cấp phép của việc liên kết là gì? Ví dụ: hình thức tiếp quản mã này có yêu cầu một ứng dụng sử dụng thư viện theo GPL để chấp nhận chính GPL không? Và điều này có áp dụng cho cả liên kết tĩnh và liên kết động hay không?

Quá trình liên kết rất phổ biến trong việc phát triển phần mềm, và các vấn đề liên quan đến copyleft tương hỗ rộng đến nỗi FSF đã sớm xem xét giải pháp cấp phép cho việc liên kết thông qua *Giấy phép Công cộng Chung Thu hẹp GNU* (GNU Lesser General Public License - LGPL). LGPL được cập nhật cùng lúc với mỗi phiên bản mới của GPL. Tên của giấy phép trong phiên bản 1 là *Giấy phép Công cộng Chung Thư viện GNU*; chính bản thân cái tên này đã cho biết lý do ban đầu khiến người ta phải tạo ra giấy phép này. Trong phiên bản 2 và 3, “Thư viện” (Library) đã được thay thế bằng “Thu hẹp” (Lesser) để chỉ ra điều gì đang thực sự bị đe dọa, cụ thể là sự suy yếu về tính thực dụng của nguyên tắc copyleft.

Như vậy, một thư viện được cấp phép theo LGPL có thể được phần mềm sử dụng mà bản thân phần mềm này không tự động tuân theo copyleft. Đặc biệt là các dự án phần mềm tuân theo giấy phép *linh hoạt* (được đề cập tới trong bài học khác) sẽ được hưởng lợi từ sự thỏa hiệp này vì nó tạo ra một vòng bảo vệ pháp lý cho sự kết hợp giữa các phương pháp tiếp cận về bản chất là không

tương thích theo luật cấp phép.

Mọi thay đổi đối với phần mềm được cấp phép theo LGPL vẫn phải tuân theo copyleft, tức là chúng cũng phải tuân theo LGPL.

Tuy nhiên, thư viện được cấp phép LGPL phải có thể thay thế được để cho phép người dùng phần mềm thay thế thư viện bằng một phiên bản sửa đổi. Thông tin về các điều kiện thích hợp để thực hiện việc thay thế đó cũng phải được cung cấp.

Các giấy phép Copyleft khác

Các dự án và tổ chức FOSS khác cũng luôn tìm kiếm một khuôn khổ pháp lý tốt nhất cho nhu cầu của họ và từ đó phát triển giấy phép của riêng mình. Một ví dụ chính là *Tổ chức Mozilla* được thành lập vào năm 1998 và ngày nay được biết đến nhiều nhất với hai dự án mà nó hỗ trợ: trình duyệt internet Firefox và ứng dụng email Thunderbird.

Phiên bản 1 của *Giấy phép Công cộng Mozilla* (MPL) được phát hành vào năm 1998 và phiên bản 2.0 hiện tại (tính đến năm 2024) là vào năm 2012.

Giống như LGPL, MPL thường được coi là một giấy phép “copyleft yếu”. Trên thực tế, mục đích của nó là tìm cách để đạt được sự cân bằng giữa các yêu cầu nghiêm ngặt của copyleft và khả năng tích hợp với các sản phẩm thương mại. Nó có thể đạt được điều này và một số tiêu chí khác thông qua một nguyên tắc được gọi là *copyleft cấp độ tệp*: Nếu bạn thực hiện thay đổi đối với một tệp thuộc về phần mềm tuân theo MPL, bạn có thể tích hợp tệp này vào phần mềm độc quyền, miễn là chính tệp được sửa đổi đó vẫn thuộc MPL và do đó có thể truy cập được.

Một ví dụ khác về giấy phép copyleft yếu là *Giấy phép Công cộng Eclipse* (EPL) từ Tổ chức Eclipse. Phiên bản 2.0 hiện tại từ năm 2017 rất giống với MPL và thường được coi là giấy phép copyleft “thân thiện với doanh nghiệp” nhất. Tuy nhiên, các giấy phép copyleft khác nhau — và có những giấy phép khác ngoài những giấy phép được đề cập ở đây — thường phát sinh do những phát triển mang tính thời thế hơn là những khác biệt rõ ràng về mặt pháp lý.

Bài tập Hướng dẫn

1. GPL là cụm viết tắt của khái niệm nào?

2. Tại sao giấy phép copyleft còn được gọi là giấy phép tương hỗ?

3. Giấy phép copyleft FSF nào phù hợp với thư viện phần mềm?

4. Thuật ngữ nào thay thế thuật ngữ “phân phối” (distribution) trong GPLv3 và tại sao?

5. Giấy phép copyleft nào sau đây được cấp bởi Tổ chức Phần mềm Tự do?

GPL phiên bản 3	
AGPL phiên bản 1	
LGPL phiên bản 2	
MPL 2.0	
Phiên bản EPL 2	

Bài tập Khám phá

1. Giấy phép nào sau đây là giấy phép copyleft?

GPL phiên bản 2	
Giấy phép BSD 3 điều khoản	
LGPL phiên bản 3	
CC BY-ND	
EPL phiên bản 2	

2. Thông thường người ta có thể tạo ra một sản phẩm phái sinh kết hợp các phần của hai dự án phần mềm theo các giấy phép copyleft mạnh khác nhau không? Hãy đưa ra lý do.

3. Giấy phép nào sau đây có copyleft mạnh và giấy phép nào có copyleft yếu?

Giấy phép phân phối và phát triển chung (CDDL) 1.1	
GNU AGPLv3	
Giấy phép tương hỗ của Microsoft (MS-RL)	
Giấy phép Công cộng IBM (IPL) 1.0	
Giấy phép Sleepycat	

4. Hãy mô tả một số vấn đề tương thích có thể phát sinh khi kết hợp một phần mềm tuân theo giấy phép copyleft yếu với một phần mềm tuân theo giấy phép không phải copyleft.

Tóm tắt

Bài học này đề cập đến các đặc điểm của giấy phép phần mềm tuân theo nguyên tắc copyleft. Được phát triển vào những năm 1980 bởi Tổ chức Phần mềm Tự do, Giấy phép Công cộng Chung GNU (GPL) hiện là giấy phép phổ biến nhất với tính bản quyền mạnh mẽ. Mặc dù có nhiều bản sửa đổi trước phiên bản 3 hiện tại, các yêu cầu cơ bản của nó hầu như không thay đổi: Các quyền tự do được cấp bởi giấy phép để sử dụng, tái phân phối và sửa đổi phần mềm mà không bị hạn chế phải luôn được duy trì. Điều này có nghĩa là phần mềm đã được sửa đổi chỉ có thể được phân phối với cùng các điều kiện (tức là cùng một giấy phép).

Những cải tiến kỹ thuật cũng như mong muốn có được sự thoải mái về mặt pháp lý khi hợp tác với các dự án khác đã thúc đẩy FSF phát triển các giấy phép bổ sung hoặc thay thế. Ví dụ: Giấy phép Công cộng Chung Thu hẹp (LGPL) của GNU có tính đến liên kết tĩnh hoặc động của các thư viện phần mềm thường được sử dụng trong việc phát triển phần mềm. Giấy phép Công cộng Chung GNU Affero (AGPL) đáp ứng xu hướng kỹ thuật hướng tới việc sử dụng phần mềm dưới dạng dịch vụ qua mạng (đặc biệt là internet) — tức là không phải trong các cài đặt cục bộ.

Ngoài FSF, các dự án khác như Tổ chức Mozilla và Tổ chức Eclipse cũng đã phát triển các giấy phép copyleft. Những giấy phép này cũng tìm kiếm sự thỏa hiệp giữa việc đảm bảo các quyền tự do được cấp bởi giấy phép và một mối quan hệ đơn giản hơn với các phần mềm tuân theo các giấy phép khác thông qua một giấy phép copyleft yếu hơn.

Về nguyên tắc, khả năng tương thích của giấy phép là một vấn đề quan trọng đối với giấy phép copyleft bởi nguyên tắc phát triển phần mềm hợp tác và tự do phải được dung hòa với các điều kiện pháp lý được xác định bởi giấy phép tương ứng. Trong bất kỳ hình thức kết hợp phần mềm nào theo các giấy phép khác nhau, các điều kiện pháp lý cũng phải được xem xét kỹ lưỡng trong từng trường hợp cụ thể.

Đáp án Bài tập Hướng dẫn

1. GPL là cụm viết tắt của khái niệm nào?

   General Public License (Giấy phép Cộng cộng Chung).

2. Tại sao giấy phép copyleft còn được gọi là giấy phép tương hỗ?

   Nguyên tắc copyleft yêu cầu các quyền tự do được cấp bởi một giấy phép cũng phải được cấp cho người khác mà không bị hạn chế. Ví dụ: nếu bạn thực hiện một thay đổi đối với phần mềm tuân theo GPL, bạn có nghĩa vụ cung cấp những thay đổi này cho những người khác theo cùng điều kiện phù hợp với nguyên tắc tương hỗ có qua có lại.

3. Giấy phép copyleft FSF nào phù hợp với thư viện phần mềm?

   Giấy phép Công cộng Chung Thu hẹp GNU (GNU LGPL).

4. Thuật ngữ nào thay thế thuật ngữ “phân phối” (distribution) trong GPLv3 và tại sao?

   Các thuật ngữ “chuyển tải” (convey) và “truyền bá” (propagate) thay thế cho thuật ngữ “phân phối” (distribute). Cơ sở của điều này là thuật ngữ “phân phối” được gắn chặt chẽ trong luật bản quyền quốc tế. Để tránh xung đột hoặc hiểu lầm, GPL trong phiên bản 3 không sử dụng thuật ngữ này.

5. Giấy phép copyleft nào sau đây được cấp bởi Tổ chức Phần mềm Tự do?

GPL phiên bản 3	X
AGPL phiên bản 1	
LGPL phiên bản 2	X
MPL 2.0	
Phiên bản EPL 2	

Đáp án Bài tập Mở rộng

1. Giấy phép nào sau đây là giấy phép copyleft?

GPL phiên bản 2	X
Giấy phép BSD 3 điều khoản	
LGPL phiên bản 3	X
CC BY-ND	
EPL phiên bản 2	X

2. Thông thường người ta có thể tạo ra một sản phẩm phái sinh kết hợp các phần của hai dự án phần mềm theo các giấy phép copyleft mạnh khác nhau không? Hãy đưa ra lý do.

   Không. Các giấy phép có copyleft mạnh thường yêu cầu các phiên bản được sửa đổi phải tuân theo cùng một giấy phép. Điều này cũng loại trừ việc cấp lại giấy phép. Do đó, sự kết hợp của các giấy phép khi cả hai đều tuân theo những nguyên tắc này tạo nên một mâu thuẫn không thể giải quyết được.

3. Giấy phép nào sau đây có copyleft mạnh và giấy phép nào có copyleft yếu?

Giấy phép phân phối và phát triển chung (CDDL) 1.1	Mạnh
GNU AGPLv3	Yếu
Giấy phép tương hỗ của Microsoft (MS-RL)	Yếu
Giấy phép Công cộng IBM (IPL) 1.0	Yếu
Giấy phép Sleepycat	Mạnh

4. Hãy mô tả một số vấn đề tương thích có thể phát sinh khi kết hợp một phần mềm tuân theo giấy phép copyleft yếu với một phần mềm tuân theo giấy phép không phải copyleft.

   Trong khi một giấy phép copyleft mạnh yêu cầu phần mềm đã sửa đổi phải được phân phối theo cùng một giấy phép, đối với các giấy phép copyleft yếu thì các điều kiện lại khá “dễ chịu”. Tuy nhiên, bất kỳ sự kết hợp nào với các giấy phép khác đều đặt ra những câu hỏi cơ bản về tính tương thích. Các khía cạnh quan trọng cần được xem xét khi trả lời những câu hỏi này: Các phần gốc của hai dự án phần mềm có tách biệt rõ ràng trong sản phẩm mới hay không và nếu cần, liệu chúng có thể vẫn được cấp phép theo các cách khác nhau hay không? Sự kết nối giữa hai dự án phần mềm gốc thực sự diễn ra ở cấp độ nào? Mã nguồn có được đưa vào trực tiếp hay nó “chỉ” được liên kết động với phần mềm (thư viện) khác? Một trong hai giấy phép có thường

cho phép cấp phép lại không? Tất cả các câu hỏi như vậy chỉ có thể được trả lời sau khi phân tích chính xác các giấy phép tương ứng và với chuyên môn pháp lý của một chuyên gia.

052.3 Giấy phép phần mềm cho phép

Tham khảo các mục tiêu LPI

Open Source Essentials version 1.0, Exam 050, Objective 052.3

Khối lượng

3

Các lĩnh vực kiến thức chính

- Hiểu rõ khái niệm về giấy phép phần mềm linh hoạt
- Hiểu rõ các quyền được cấp bởi giấy phép phần mềm linh hoạt
- Hiểu rõ các nghĩa vụ do giấy phép phần mềm linh hoạt áp đặt
- Hiểu rõ các thuộc tính chính của các giấy phép phần mềm linh hoạt phổ biến
- Hiểu rõ tính tương thích của giấy phép phần mềm linh hoạt với các giấy phép phần mềm khác

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Giấy phép BSD 2 điều khoản
- Giấy phép BSD 3 điều khoản
- Giấy phép MIT
- Giấy phép Apache, phiên bản 2.0

Bài 1

Chứng chỉ:	Open Source Essentials
Phiên bản:	1.0
Chủ đề:	052 Giấy phép dành cho Phần mềm Mã nguồn Mở
Mục tiêu:	052.3 Giấy phép Phần mềm Linh hoạt
Bài học:	1 trên 1

Giới thiệu

Giấy phép *linh hoạt* (permissive) hiện là giấy phép mã nguồn mở được sử dụng rộng rãi nhất, trái ngược với các giấy phép *hạn chế* như Giấy phép Công cộng Chung GNU (GPL). Giấy phép phần mềm linh hoạt có xu hướng đơn giản và linh hoạt, đồng thời cung cấp nhiều quyền tự do cho tác giả của chúng — có lẽ đây là giấy phép có quyền tự do rộng rãi nhất trong số các giấy phép mã nguồn mở.

Loại giấy phép này trao các quyền tự do rộng rãi cho các nhà phát triển phần mềm liên quan đến việc sử dụng, sửa đổi và tái phân phối phần mềm, miễn là họ có công nhận tác giả gốc. Ví dụ: một người thường có thể phân phối một sản phẩm phái sinh theo giấy phép mã nguồn đóng, miễn là sản phẩm đó có bao gồm phần công nhận về tác giả của sản phẩm gốc.

Nguyên tắc đằng sau logic này là mục đích phổ biến phần mềm ở mức tối đa nhất. Giấy phép linh hoạt khẳng định sẽ mang lại lợi ích cho các cá nhân và cộng đồng bằng cách tạo điều kiện thuận lợi cho việc sử dụng phần mềm vì mục đích thương mại, đồng thời bảo vệ các quyền đo cho tác giả gốc thông qua việc ghi nhận và công nhận họ.

Không phải ngẫu nhiên mà các giấy phép phần mềm linh hoạt đầu tiên và quan trọng nhất lại được phát triển trong lĩnh vực học thuật (chẳng hạn như các giấy phép giống BSD của Đại học Berkeley ở California hay Giấy phép MIT/X11 của Viện Công nghệ Massachusetts). Chúng được gọi là giấy phép mã nguồn mở học thuật. Ảnh hưởng của các giấy phép này ở trong ngành lớn đến mức chúng đã đặt nền móng cho các giấy phép phần mềm linh hoạt tương tự vượt ra ngoài bối cảnh học thuật như Giấy phép Apache từ Tổ chức Phần mềm Apache.

Quyền và Nghĩa vụ của Giấy phép Phần mềm Linh hoạt

Nhìn chung, các giấy phép phần mềm linh hoạt phổ biến nhất sẽ cung cấp cho người được cấp phép quyền không giới hạn để:

Sử dụng phần mềm

Bất kỳ ai cũng có thể sử dụng phần mềm được cấp giấy phép phần mềm linh hoạt (từ người dùng cá nhân, công ty thương mại đến cơ quan công quyền) và cho bất kỳ mục đích nào, dù là cá nhân hay chuyên nghiệp.

Sửa đổi phần mềm

Phần mềm có thể được cải tiến, điều chỉnh hoặc thậm chí tích hợp dưới dạng một thành phần (ví dụ: thư viện) vào các phần mềm khác.

Tái phân phối phần mềm

Nếu phần mềm được sửa đổi dưới dạng một sản phẩm phái sinh, nó có thể được tái phân phối theo các giấy phép khác nhau, bao gồm cả các giấy phép độc quyền.

Nghĩa vụ duy nhất giấy phép phần mềm linh hoạt yêu cầu thường là nghĩa vụ ghi công: Người được cấp phép được yêu cầu chỉ ra tên của tác giả gốc của phần mềm trong các sản phẩm phái sinh và kèm theo bản sao văn bản giấy phép trong mọi lần tái phân phối phần mềm.

Các tính năng của Giấy phép Phần mềm Linh hoạt quan trọng nhất

Phần này sẽ giải thích sự khác biệt giữa giấy phép MIT/X11, các giấy phép BSD phổ biến nhất và giấy phép Apache 2.0; tất cả những giấy phép này hiện nay đều đang được sử dụng rộng rãi.

Giấy phép MIT/X11

Giấy phép MIT/X11 (text: https://opensource.org/license/mit) còn được gọi là giấy phép X11; đây là một trong những giấy phép học thuật đã được đề cập ở trên. Giấy phép này được lấy tên từ phần mềm Hệ thống X Window do Viện Công nghệ Massachusetts phát triển vào năm 1987. Giấy phép

này là một trong những giấy phép phần mềm linh hoạt lâu đời và phổ biến nhất, một phần vì ngôn ngữ đơn giản và rõ ràng của nó.

Giấy phép này cấp cho người được cấp phép các quyền để:

- Sử dụng, sửa đổi và phân phối phần mềm
- Thương mại hóa phần mềm (có hoặc không có sửa đổi)
- Phát hành các sản phẩm phái sinh theo một giấy phép khác (ngay cả dưới dạng phần mềm mã nguồn đóng)

Nghĩa vụ duy nhất của người được cấp phép là đưa thông báo bản quyền và văn bản giấy phép vào mã nguồn của phần mềm hoặc các sản phẩm phái sinh từ nó.

Giấy phép MIT/X11 được coi là tương thích với tất cả các giấy phép copyleft quan trọng nhất, cả yếu và mạnh. Đặc biệt, giấy phép MIT/X11 còn tương thích với

- Giấy phép Công cộng Chung GNU (GPL), phiên bản 2.0 và 3.0
- Giấy phép Công cộng Chung GNU Thu hẹp (LGPL), phiên bản 2.0 và 3.0
- Giấy phép Công cộng Mozilla (MPL)

Điều đó có nghĩa là các sản phẩm phái sinh của phần mềm được cấp phép ban đầu theo giấy phép MIT/X11 có thể được tái phân phối theo một trong các giấy phép nêu trên hoặc được đưa vào các dự án được phát hành theo một trong các giấy phép nêu trên.

Giấy phép BSD 2 Điều khoản

Giấy phép BSD 2 Điều khoản (text: https://opensource.org/license/bsd-2-clause) bắt nguồn từ giấy phép BSD nguyên bản được tạo ra vào năm 1980 bởi Đại học Berkeley ở California dưới dạng giấy phép cho BSD - hệ điều hành giống Unix của họ.

Giấy phép này nổi tiếng vì tính đơn giản và, đúng như tên gọi của nó, chỉ bao gồm hai điều khoản ngắn. Về cơ bản, nó cũng giống với giấy phép MIT/X11 ở điểm yêu cầu có phần ghi công tác giả trong phần mềm. Ngoài ra, Giấy phép BSD 2 ĐIều khoản còn yêu cầu người được cấp phép đưa văn bản giấy phép vào tài liệu và các tư liệu khác được cung cấp khi tái phân phối mã.

Tóm lại, giấy phép BSD 2 Điều khoản cấp các quyền để:

- Sử dụng, sửa đổi và phân phối phần mềm
- Thương mại hóa phần mềm (có hoặc không có sửa đổi)
- Phát hành các sản phẩm phái sinh theo một giấy phép khác (ngay cả dưới dạng phần mềm mã

nguồn đóng)

Người được cấp phép có nghĩa vụ:

- Đính kèm thông báo bản quyền và văn bản giấy phép trong mã nguồn và mã nhị phân của phần mềm hoặc các sản phẩm phái sinh của nó
- Đính kèm thông báo bản quyền và văn bản giấy phép trong tài liệu hoặc các tài liệu liên quan được cung cấp cùng với phần mềm

Giấy phép BSD 2 Điều khoản được coi là tương thích với tất cả các giấy phép copyleft phổ biến, cả yếu và mạnh. Đặc biệt, Giấy phép BSD 2 Điều khoản tương thích với:

- Giấy phép Công cộng Chung GNU (GPL), phiên bản 2.0 và 3.0
- Giấy phép Công cộng Chung GNU Thu hẹp (LGPL), phiên bản 2.0 và 3.0
- Giấy phép Công cộng Mozilla (MPL)

Do đó, các sản phẩm phái sinh của phần mềm được cấp phép ban đầu theo giấy phép BSD 2 Điều khoản có thể được tái phân phối theo một trong các giấy phép nêu trên hoặc được đưa vào các dự án được phát hành theo một trong các giấy phép nêu trên.

Giấy phép BSD 3 Điều khoản

Giấy phép BSD 3 Điều khoản (text: https://opensource.org/license/bsd-3-clause) là một biến thể khác của giấy phép BSD ban đầu và chỉ bao gồm ba điều khoản. Đặc điểm chính để phân biệt giấy phép này với giấy phép BSD 2 Điều khoản họ hàng là "điều khoản không chứng thực" nhằm ngăn chặn việc tên của tác giả gốc bị lợi dụng để phân phối hoặc bán các sản phẩm phái sinh.

Giấy phép BSD 3 Điều khoản cấp cho người được cấp phép các quyền để:

- Sử dụng, sửa đổi và phân phối phần mềm
- Thương mại hóa phần mềm (có hoặc không có sửa đổi)
- Phát hành các sản phẩm phái sinh theo một giấy phép khác (ngay cả dưới dạng phần mềm mã nguồn đóng)

Người được cấp phép có nghĩa vụ:

- Đính kèm thông báo bản quyền và văn bản giấy phép trong mã nguồn và mã nhị phân của phần mềm hoặc các sản phẩm phái sinh của nó
- Đính kèm thông báo bản quyền và văn bản giấy phép trong tài liệu hoặc các tài liệu liên quan khác được cung cấp cùng với phần mềm

- Không trích dẫn tên của người giữ bản quyền hoặc những nhà phát triển đóng góp để xác nhận hoặc quảng bá các sản phẩm có nguồn gốc từ phần mềm này mà không có sự cho phép cụ thể trước bằng văn bản

Để giải thích nghĩa vụ thứ hai — "điều khoản không chứng thực" — hãy tưởng tượng một tình huống mà trong đó một công ty hoặc cá nhân tạo ra một sản phẩm phái sinh bắt nguồn từ phần mềm được cấp phép theo giấy phép phần mềm linh hoạt. Trong trường hợp không có điều khoản không chứng thực này, người được cấp phép có thể quảng bá sản phẩm mới bằng cách chỉ ra rằng nó là một sản phẩm phái sinh từ phần mềm của một nhà phát triển nổi tiếng nhằm hưởng lợi từ uy tín của họ.

Giống như người anh em của mình, giấy phép BSD 3 Điều khoản được coi là tương thích với tất cả các giấy phép copyleft quan trọng nhất, cả yếu và mạnh. Đặc biệt, giấy phép BSD 3 Điều khoản còn tương thích với:

- Giấy phép Công cộng Chung GNU (GPL), phiên bản 2.0 và 3.0
- Giấy phép Công cộng Chung GNU Thu hẹp (LGPL), phiên bản 2 và 3
- Giấy phép Công cộng Mozilla (MPL)

Do đó, các sản phẩm phái sinh của phần mềm được cấp phép ban đầu theo giấy phép BSD 3 Điều khoản có thể được tái phân phối theo một trong các giấy phép nêu trên hoặc được đưa vào các dự án được phát hành theo một trong các giấy phép nêu trên.

Giấy phép Apache 2.0

Giấy phép Apache đầu tiên được phát triển bởi Tổ chức Phần mềm Apache - một tổ chức phi lợi nhuận được thành lập vào năm 1999 - để phân phối phần mềm của họ và giúp dễ dàng đưa nó vào các dự án khác. Sứ mệnh của giấy phép này là đảm bảo sự hợp tác trong phát triển phần mềm và áp dụng triết lý mã nguồn mở trên tinh thần thân thiện với doanh nghiệp.

Giấy phép Apache 2.0 (text: https://www.apache.org/licenses/LICENSE-2.0) có lẽ là giấy phép phổ biến nhất trong số các giấy phép phần mềm linh hoạt; nó có các điểm khác biệt so với các giấy phép đã được nhắc đến trước đó ở một số khía cạnh liên quan. Khía cạnh chính liên quan đến việc phân bổ quyền sáng chế cho từng người được cấp phép (bao gồm điều khoản liên quan đến việc chấm dứt giấy phép sáng chế) nhằm hạn chế và ngăn chặn mọi yêu cầu bồi thường thiệt hại do vi phạm bằng sáng chế.

Ngoài ra, có hai nghĩa vụ khác rất quan trọng. Người được cấp phép phải phát hành bất kỳ một bộ phận hoặc thành phần nào của phần mềm (tuân theo cùng một giấy phép Apache 2.0) mà không qua sửa đổi. Hơn nữa, người được cấp phép phải đặt một thông báo được làm nổi trong bất kỳ tệp đã sửa đổi nào để cho biết rằng người được cấp phép đã thay đổi những tệp đó.

Tổ chức Phần mềm Tự do — đơn vị khuyến khích và thúc đẩy việc sử dụng các giấy phép hạn chế — khuyến nghị rằng Apache 2.0 là giấy phép phần mềm linh hoạt tốt nhất để phân phối một số lượng nhỏ phần mềm và thư viện.

Giấy phép Apache 2.0 cấp cho người được cấp phép:

- Quyền sử dụng, sửa đổi và phân phối phần mềm
- Quyền thương mại hóa phần mềm (có hoặc không có sửa đổi)
- Quyền phát hành các sản phẩm phái sinh theo giấy phép khác (kể cả dưới dạng phần mềm mã nguồn đóng)
- Một giấy phép bằng sáng chế để sử dụng, bán, nhập khẩu hoặc chuyển giao phần mềm được cấp bằng sáng chế
- Một giấy phép bằng sáng chế để sử dụng, bán, nhập khẩu hoặc chuyển giao phần mềm có thể vi phạm quyền sáng chế của một nhà phát triển đóng góp

Người được cấp phép có nghĩa vụ:

- Đặt các thông báo được làm nổi trong bất kỳ tệp sửa đổi nào để cho biết rằng các tệp đã bị sửa đổi
- Công bố tất cả các phần chưa qua sửa đổi của phần mềm gốc theo giấy phép Apache 2.0
- Đính kèm thông báo bản quyền và văn bản giấy phép trong mã nguồn của phần mềm và các sản phẩm phái sinh của nó
- Bao gồm thông báo bản quyền và văn bản giấy phép trong tài liệu và các tư liệu liên quan khác được cung cấp cùng với phần mềm

Giấy phép Apache 2.0 được coi là chỉ tương thích với một số giấy phép copyleft quan trọng nhất. Đặc biệt, giấy phép Apache 2.0 tương thích với:

- Giấy phép Công cộng Chung GNU (GPL), phiên bản 3.0 (không bao gồm phiên bản 2.0)
- Giấy phép Công cộng Chung GNU Thu hẹp (LGPL), phiên bản 3.0 (không bao gồm phiên bản 2.0)
- Giấy phép Công cộng Mozilla (MPL), phiên bản 2.0 (không bao gồm phiên bản 1.1)

Điều đó có nghĩa là các sản phẩm phái sinh của phần mềm được cấp phép ban đầu theo giấy phép Apache 2.0 có thể được tái phân phối theo một trong các giấy phép tương thích nêu trên hoặc được đưa vào các dự án được phát hành theo một trong các giấy phép tương thích nêu trên.

Khả năng tương thích hạn chế giữa giấy phép Apache 2.0 và các giấy phép mã nguồn mở khác chủ yếu là do sự hiện diện của các điều khoản liên quan đến việc cấp giấy phép bằng sáng chế cho người được cấp phép.

Giấy phép Phần mềm Linh hoạt liên quan đến các Giấy phép Mã nguồn Mở khác

Hiện tại, chúng ta đã có một cái nhìn tổng quan về các tính năng phổ biến và thiết yếu của các giấy phép phần mềm linh hoạt; chúng ta có thể tiếp tục tìm hiểu xem chúng khác với phần mềm công cộng và giấy phép copyleft như thế nào.

So sánh với Phần mềm Công cộng

Sự khác biệt đầu tiên giữa giấy phép phần mềm linh hoạt và việc phát hành phần mềm thuộc phạm vi công cộng nằm ở chính sự tồn tại của chúng. Phạm vi công cộng không phải là một giấy phép thực tế mà chỉ là một cách để phát hành phần mềm. Nói cách khác, theo định nghĩa, các sản phẩm thuộc phạm vi công cộng sẽ không có giấy phép.

Sự khác biệt tiếp theo nằm ở các nghĩa vụ mà giấy phép linh hoạt áp đặt lên người được cấp phép. Trên thực tế, người dùng các sản phẩm thuộc phạm vi công cộng không có bất kỳ một nghĩa vụ nào. Tuy nhiên, bất kỳ ai sử dụng, sửa đổi hoặc tái phân phối phần mềm theo giấy phép phần mềm linh hoạt đều phải tuân thủ nghĩa vụ ghi công tác giả đã được giải thích từ trước: họ được yêu cầu phải đính kèm tên của tác giả gốc và bản sao văn bản giấy phép trong phần mềm.

Hậu quả của nghĩa vụ ghi công tác giả là không có sản phẩm phái sinh nào có nguồn gốc từ phần mềm tuân theo giấy phép phần mềm linh hoạt có thể được phát hành dưới phạm vi công cộng vì điều này sẽ vi phạm (hoặc phá vỡ) quyền được ghi công của tác giả ban đầu.

So sánh với Copyleft

Sự khác biệt giữa giấy phép phần mềm linh hoạt và giấy phép copyleft hạn chế sẽ phức tạp hơn, đặc biệt là khi chúng ta xem xét sự khác biệt giữa các giấy phép copyleft khác nhau. Như đã thấy trong các bài học trước, một điểm khác biệt chính phân biệt các giấy phép copyleft *mạnh* (bao gồm Giấy phép Công cộng Chung GNU phiên bản 2.0 và 3.0) với các giấy phép copyleft *yếu* (bao gồm Giấy phép Công cộng Chung GNU Thu hẹp phiên bản 2.0 và 3.0 và Giấy phép Công cộng Mozilla).

Sự khác biệt chính giữa giấy phép hạn chế và giấy phép linh hoạt nằm ở nguyên tắc copyleft vốn là cốt lõi của giấy phép hạn chế. Để phù hợp với nguyên tắc này, người được cấp phép sửa đổi phần mềm theo giấy phép copyleft và sau đó phân phối nó sẽ nhất thiết phải phát hành — toàn bộ hoặc một phần — sản phẩm phái sinh dưới cùng một giấy phép với phần mềm gốc. Việc không tuân thủ nghĩa vụ này sẽ dẫn đến vi phạm bản quyền và đi kèm theo hậu quả pháp lý.

Ngược lại, giấy phép phần mềm linh hoạt không áp đặt nghĩa vụ như vậy. Những người sử dụng hoặc dự định phát hành phần mềm theo loại giấy phép này có thể tự do quyết định giấy phép cho sản phẩm phái sinh của mình (bao gồm cả giấy phép độc quyền).

Giấy phép Copyleft mạnh

Sự khác biệt giữa giấy phép phần mềm linh hoạt và giấy phép copyleft mạnh rõ rệt hơn so với giấy phép copyleft yếu.

Sự khác biệt cơ bản là các giấy phép copyleft mạnh yêu cầu các sản phẩm phái sinh phải được phát hành theo cùng một giấy phép với phần mềm gốc. Điều này cũng sẽ được áp dụng nếu phần mềm được cấp phép copyleft được tích hợp vào một dự án khác: Toàn bộ sản phẩm phái sinh phải được phát hành theo cùng một giấy phép copyleft mạnh.

Giấy phép Copyleft yếu

Không giống như các giấy phép copyleft mạnh, các giấy phép Copyleft yếu chỉ yêu cầu một phần về việc phát hành sản phẩm phái sinh theo cùng một giấy phép. Cụ thể hơn, người được cấp phép tái phân phối phần mềm được phát hành theo giấy phép copyleft yếu phải áp dụng giấy phép tương tự cho phần sản phẩm được bắt nguồn từ phần mềm gốc của họ. Ví dụ: một thư viện dựa trên thư viện được cấp phép theo LGPL cũng phải được cấp phép theo LGPL.

Các giấy phép copyleft yếu chính xác là được sinh ra để mang các tính năng của chúng đến gần hơn với các tính năng của giấy phép phần mềm linh hoạt. Tuy nhiên, chúng được phân biệt với các giấy phép phần mềm linh hoạt ở chỗ giấy phép phần mềm linh hoạt không áp đặt nghĩa vụ duy trì cùng một giấy phép đối với sản phẩm phái sinh hoặc tích hợp trong bất kỳ trường hợp nào.

Bài tập Hướng dẫn

1. Hai nghĩa vụ thường được áp đặt bởi giấy phép phần mềm linh hoạt là gì?

2. Giấy phép nào sau đây *không* phải là giấy phép phần mềm linh hoạt?

Giấy phép Apache 2.0	
Giấy phép LGPL	
Giấy phép MIT/X11	
Giấy phép BSD 3 Điều khoản	

3. Điều gì phân biệt giấy phép phần mềm linh hoạt với giấy phép copyleft?

Phần mềm được phát hành theo giấy phép copyleft không thể được phân phối, trong khi phần mềm tuân theo giấy phép phần mềm linh hoạt thì có thể.	
Các sản phẩm bắt nguồn từ phần mềm tuân theo giấy phép copyleft không thể được phát hành theo giấy phép độc quyền, trong khi phần mềm tuân theo giấy phép phần mềm linh hoạt thì có thể.	
Giấy phép Copyleft chỉ được công nhận về mặt pháp lý ở Hoa Kỳ, trong khi giấy phép phần mềm linh hoạt được công nhận trên toàn cầu.	

4. Giấy phép phần mềm linh hoạt nào sẽ cấp giấy phép bằng sáng chế để sử dụng, bán, nhập khẩu và chuyển giao phần mềm được cấp bằng sáng chế?

Giấy phép Apache 2.0	
Giấy phép BSD 2 Điều khoản	
Giấy phép MIT/X11	
Giấy phép BSD 3 Điều khoản	

5. Khi phần mềm được phát hành theo giấy phép MIT/X11, bạn có thể phân phối phần mềm đó theo giấy phép độc quyền và bán sản phẩm phái sinh dựa trên phần mềm đó không?

Bài tập Mở rộng

1. Bạn đang sửa đổi phần mềm được phân phối theo giấy phép Apache 2.0 và muốn tái phân phối sản phẩm phái sinh theo giấy phép độc quyền. Bạn nên làm theo những bước nào để tuân thủ các nghĩa vụ cấp phép Apache 2.0?

2. Hãy kể tên ít nhất ba ví dụ về các dự án phổ biến được phát hành theo giấy phép phần mềm linh hoạt.

3. Tại sao bạn không thể phân phối theo giấy phép LGPL 2.0 các phần mềm có bao gồm các thành phần ban đầu được phát hành theo giấy phép MIT/X11, giấy phép Apache 2.0 và giấy phép BSD 2 Điều khoản? Giấy phép copyleft yếu nào khác có thể được sử dụng để phát hành phần mềm?

Tóm tắt

Trong bài học này, chúng ta đã học về: * Giấy phép phần mềm linh hoạt là gì, các quyền và nghĩa vụ mà chúng cung cấp * Sự khác biệt giữa giấy phép phần mềm linh hoạt và các giấy phép mã nguồn mở khác * Các tính năng phổ biến nhất của giấy phép phần mềm linh hoạt * Khả năng tương thích của giấy phép phần mềm linh hoạt với các giấy phép mã nguồn mở khác

Đáp án Bài tập Hướng dẫn

1. Hai nghĩa vụ thường được áp đặt bởi giấy phép phần mềm linh hoạt là gì?

   Nghĩa vụ nêu rõ tên tác giả gốc của phần mềm và nghĩa vụ đính kèm một bản sao văn bản giấy phép trong sản phẩm phái sinh.

2. Giấy phép nào sau đây *không* phải là giấy phép phần mềm linh hoạt?

Giấy phép Apache 2.0	
Giấy phép LGPL	X
Giấy phép MIT/X11	
Giấy phép BSD 3 Điều khoản	

3. Điều gì phân biệt giấy phép phần mềm linh hoạt với giấy phép copyleft?

Phần mềm được phát hành theo giấy phép copyleft không thể được phân phối, trong khi phần mềm tuân theo giấy phép phần mềm linh hoạt thì có thể.	
Các sản phẩm bắt nguồn từ phần mềm tuân theo giấy phép copyleft không thể được phát hành theo giấy phép độc quyền, trong khi phần mềm tuân theo giấy phép phần mềm linh hoạt thì có thể.	X
Giấy phép Copyleft chỉ được công nhận về mặt pháp lý ở Hoa Kỳ, trong khi giấy phép phần mềm linh hoạt được công nhận trên toàn cầu.	

4. Giấy phép phần mềm linh hoạt nào sẽ cấp giấy phép bằng sáng chế để sử dụng, bán, nhập khẩu và chuyển giao phần mềm được cấp bằng sáng chế?

Giấy phép Apache 2.0	X
Giấy phép BSD 2 Điều khoản	
Giấy phép MIT/X11	
Giấy phép BSD 3 Điều khoản	

5. Khi phần mềm được phát hành theo giấy phép MIT/X11, bạn có thể phân phối phần mềm đó

theo giấy phép độc quyền và bán sản phẩm phái sinh dựa trên phần mềm đó không?

Có thể.

Đáp án Bài tập Mở rộng

1. Bạn đang sửa đổi phần mềm được phân phối theo giấy phép Apache 2.0 và muốn tái phân phối sản phẩm phái sinh theo giấy phép độc quyền. Bạn nên làm theo những bước nào để tuân thủ các nghĩa vụ cấp phép Apache 2.0?

   Bạn nên:

   - Chèn vào mỗi tệp đã qua sửa đổi một thông báo xác nhận rằng các tệp đã được sửa đổi.
   - Phát hành tất cả các phần chưa sửa đổi của phần mềm gốc theo giấy phép Apache 2.0.
   - Đính kèm thông báo bản quyền và văn bản giấy phép trong mã nguồn của phần mềm hoặc các sản phẩm phái sinh của nó.
   - Đính kèm thông báo bản quyền và nội dung giấy phép trong tài liệu và các tư liệu liên quan khác được cung cấp cùng với việc phân phối phần mềm.

2. Hãy kể tên ít nhất ba ví dụ về các dự án phổ biến được phát hành theo giấy phép phần mềm linh hoạt.

   - Angular web framework — Giấy phép MIT/X11
   - Ruby on Rails — Giấy phép MIT/X11
   - Apache HTTP Server — Giấy phép Apache 2.0
   - Kubernetes — Giấy phép Apache 2.0

3. Tại sao bạn không thể phân phối theo giấy phép LGPL 2.0 các phần mềm có bao gồm các thành phần ban đầu được phát hành theo giấy phép MIT/X11, giấy phép Apache 2.0 và giấy phép BSD 2 Điều khoản? Giấy phép copyleft yếu nào khác có thể được sử dụng để phát hành phần mềm?

   Mặc dù giấy phép MIT/X11 và giấy phép BSD 2 Điều khoản tương thích với giấy phép LGPL 2.0 nhưng giấy phép Apache 2.0 thì không. Phần mềm có bao gồm các thành phần được phát hành theo giấy phép MIT/X11, giấy phép Apache 2.0 và giấy phép BSD 2 Điều khoản có thể được phát hành theo giấy phép LGPL 3.0 vì nó tương thích với tất cả các giấy phép phần mềm linh hoạt đó.

Linux Professional Institute

Chủ đề 053: Giấy phép dành cho Nội dung Mở

053.1 Các khái niệm về Giấy phép dành cho Nội dung Mở

Tham khảo các mục tiêu LPI

Open Source Essentials version 1.0, Exam 050, Objective 053.1

Khối lượng

2

Các lĩnh vực kiến thức chính

- Hiểu rõ về các loại nội dung mở
- Hiểu rõ những gì cấu thành nội dung có bản quyền
- Hiểu rõ về các sản phẩm phái sinh của các tài liệu có bản quyền
- Hiểu rõ nhu cầu về giấy phép nội dung mở
- Nắm được kiến thức về thương hiệu

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Tài liệu
- Hình ảnh
- Sản phẩm nghệ thuật
- Bản đồ
- Âm nhạc
- Video
- Thiết kế phần cứng và các thông số kỹ thuật
- Cơ sở dữ liệu
- Luồng dữ liệu (Data streams)

- Nguồn cấp dữ liệu (Data feeds)

Linux Professional Institute

Bài 1

Chứng chỉ:	Open Source Essentials
Phiên bản:	1.0
Chủ đề:	053 Giấy phép dành cho Nội dung Mở
Mục tiêu:	053.1 Các khái niệm về Giấy phép dành cho Nội dung Mở
Bài học:	1 trên 1

Giới thiệu

Có một tình huống giả định như sau: Frank là một nhà văn muốn mang những câu chuyện của mình đến với tất cả mọi người trên internet với một số điều kiện nhất định. Anh muốn các tác phẩm của mình hoàn toàn miễn phí và bất cứ ai cũng có thể sử dụng những câu chuyện đó mà không cần xin phép. Song song với đó, anh cũng muốn các tác phẩm của mình không bị mang đi sử dụng cho các mục đích thương mại. Cuối cùng, anh muốn mọi người ghi công tác giả xứng đáng cho mình nếu những câu chuyện đó được sử dụng cho bất kỳ mục đích gì. Frank muốn làm được điều này mà không cần phải dính líu đến các thủ tục pháp lý phức tạp. Sau này chúng ta sẽ hiểu được tại sao một nhà văn lại muốn làm như vậy.

Emma là một nhà làm phim, đồng thời cô cũng dạy một số sinh viên về cách sản xuất phim ngắn. Bên cạnh nhiều kiến thức khác, học sinh cần có một câu chuyện để tạo ra tác phẩm của mình. Emma lại tình cờ biết về những câu chuyện của Frank và cho rằng chúng rất phù hợp với lớp học của mình.

Frank có thể làm gì để cho phép Emma sử dụng những câu chuyện của mình? Tình huống này và nhiều tình huống tương tự khác có thể được giải quyết bằng cách sử dụng mô hình *cấp phép cho*

nội dung mở. Ngày nay, có hàng triệu các tác phẩm được bảo vệ bản quyền có thể được bất cứ một người nào sử dụng lại mà không cần phải trả phí cấp phép hoặc được sự đồng ý rõ ràng từ người nắm giữ bản quyền; tất cả những điều này đều là nhờ vào việc phân phối các tác phẩm theo giấy phép nội dung mở. Các tài nguyên như Wikipedia, Flickr, OpenStreetMap, Unsplash và Jamendo là một số ví dụ về các nền tảng sử dụng mô hình giấy phép này.

Theo một định nghĩa rộng hơn, *nội dung mở* ám chỉ bất kỳ một tác phẩm nào (ví dụ như phim, nhạc, hình ảnh, văn bản, cơ sở dữ liệu, bộ dữ liệu, tài liệu, bản đồ và thiết kế phần cứng) được phép sử dụng và tái phân phối tự do theo luật bản quyền. Điều này cũng bao gồm các tác phẩm mà phương án bảo vệ ban đầu của chúng đã hết thời hạn và do đó trở thành tác phẩm thuộc phạm vi công cộng. Một định nghĩa hẹp hơn sẽ là một tác phẩm đã được tác giả chỉ định tuân theo một giấy phép nội dung mở cho phép sử dụng và phân phối tác phẩm mà không cần bất kỳ khoản thanh toán hoặc sự cho phép nào. Do đó, mô hình nội dung mở không đối lập với luật bản quyền mà ngược lại sẽ bổ sung cho nó.

Nguyên tắc cơ bản về Bản quyền

Vì các giấy phép nội dung mở được dựa trên bản quyền nên chúng ta sẽ cần tìm hiểu về một số khía cạnh của luật bản quyền để hiểu tại sao các giấy phép đó lại cần thiết và chúng cho phép những gì.

Bản quyền là một phần của một danh mục pháp lý được gọi là *sở hữu trí tuệ*. Bản quyền sẽ bảo vệ các tác phẩm gốc bằng cách cấp cho tác giả các quyền hợp pháp độc quyền để kiểm soát các hành vi sử dụng nhất định của các cá nhân hoặc tổ chức khác trên tác phẩm của họ. Nói chung, điều này có nghĩa là không ai khác có thể sao chép, phân phối, biểu diễn công khai, phỏng theo hoặc thực hiện hầu hết mọi hành vi khác ngoài việc xem hoặc đọc tác phẩm mà không có sự cho phép của người nắm giữ bản quyền.

Các nguyên tắc cơ bản mà chúng ta sẽ xem xét trong phần này bao gồm việc những sản phẩm như thế nào thì có thể có bản quyền, cùng với việc ai sẽ kiểm soát các quyền và cấp quyền sử dụng lại tác phẩm có bản quyền như Emma muốn làm trong ví dụ ở trên.

Như Tổ chức Sở hữu Trí tuệ Thế giới đã tuyên bố rõ ràng:

> Bản quyền bảo vệ hai loại quyền. Quyền kinh tế cho phép chủ sở hữu quyền nhận được phần thưởng tài chính từ việc người khác sử dụng tác phẩm của họ. Quyền đạo đức cho phép tác giả và nhà sáng tạo thực hiện một số hành động nhất định để duy trì và bảo vệ mối liên kết của họ với tác phẩm. Tác giả hoặc nhà sáng tạo có thể là chủ sở hữu các quyền kinh tế hoặc các quyền đó có thể được chuyển giao cho một hoặc nhiều chủ sở hữu quyền tác giả. Có nhiều nước sẽ không cho phép chuyển giao quyền đạo đức. Bản quyền sẽ chỉ bảo vệ hình thức thể hiện các sự thật hoặc ý tưởng. Nó không cho phép người giữ bản quyền sở hữu hoặc

> kiểm soát độc quyền ý tưởng đó. Ví dụ: một hình minh họa có thể có bản quyền nhưng ý tưởng tạo ra nó thì không.
>
> — Tổ chức Sở hữu Trí tuệ Thế giới, Hiểu về Bản quyền và Các Quyền liên quan

Các quyền kinh tế được bảo vệ bởi bản quyền sẽ tồn tại rất lâu, thường là hàng thập kỷ sau khi nhà sáng tạo đã qua đời. Quyền đạo đức thì sẽ không bao giờ hết hạn.

Bản quyền cấp quyền cho các tác phẩm sáng tạo, chẳng hạn như các tác phẩm văn học và nghệ thuật đáp ứng một tiêu chuẩn nhất định về tính nguyên bản. Tác phẩm phải là sự sáng tạo của nhà sáng tạo ra nó và không được sao chép từ tác phẩm khác (có nhiều cuộc tranh luận đang diễn ra về việc trí tuệ nhân tạo có thể tham gia đến đâu vào một tác phẩm để tác phẩm đó vẫn được coi là nguyên bản).

Bản quyền chỉ bảo vệ hình thức thể hiện sự thật hoặc ý tưởng. Nó không cho phép người giữ bản quyền sở hữu hoặc kiểm soát độc quyền ý tưởng đó. Ví dụ: một hình minh họa có thể có bản quyền nhưng ý tưởng tạo ra nó thì không.

Bản quyền sẽ được tự động áp dụng khi một tác phẩm được tạo ra và cố định ở một dạng hữu hình nào đó, chẳng hạn như một tác phẩm nghệ thuật kỹ thuật số hoặc một bài hát. Điều này có nghĩa là các quyền sẽ được cấp cho nhà sáng tạo mà không cần đăng ký chính thức tác phẩm.

Những người thúc đẩy việc cấp phép nội dung mở muốn thúc đẩy một nền văn hóa tự do và sự phát triển của một nền tảng kỹ thuật số chung. Họ nhận thấy hệ thống bản quyền quá hạn chế và cứng nhắc đối với cả người dùng và nhà sáng tạo. Bằng cách tạo ra các giấy phép tiêu chuẩn dễ sử dụng, những người ủng hộ nội dung mở đã đơn giản hóa việc sử dụng và phân phối các tác phẩm được bảo vệ bởi bản quyền.

Bản quyền bảo vệ những gì?

Luật bản quyền ở mỗi quốc gia đều sẽ có những điểm khác biệt. Tuy nhiên, có một số thỏa thuận quốc tế nhằm mục đích tiêu chuẩn hóa luật bản quyền. Các tác phẩm văn học và nghệ thuật gốc có thể có bản quyền (ví dụ như các tác phẩm về nghệ thuật, âm nhạc, nhiếp ảnh, phim ảnh, truyền hình, văn học và lập trình). Các quy tắc cụ thể để quyết định nội dung nào sẽ có bản quyền và mức độ nguyên bản của tác phẩm đạt đến đâu ở mỗi khu vực cũng sẽ khác nhau.

Đôi khi những danh mục này có thể sẽ rất chung chung và áp dụng cho những tác phẩm vừa mang yếu tố sáng tạo vừa mang tính chức năng chặt chẽ. Ví dụ: một bộ phim ngắn sẽ được coi là một tác phẩm nghệ thuật. Tuy nhiên, một số phần của nó có thể sẽ không có bản quyền nếu chúng không đáp ứng được tiêu chuẩn về tính nguyên bản.

Sản phẩm phái sinh

Hãy cùng tiếp tục với tình huống giả định từ đầu bài học: Frank cuối cùng cũng đã quyết định phát hành các câu chuyện của mình theo giấy phép nội dung mở để chúng có thể được sử dụng theo các điều khoản mà anh ấy đã chọn. Vì các câu chuyện đều được cấp giấy phép đó nên Emma đã sử dụng chúng trong lớp của mình và quyết định chiếu một số phim ngắn của học sinh trong một liên hoan phim. Trong trường hợp này, nội dung do học sinh tạo ra có thể được coi là *sản phẩm phái sinh*.

Sản phẩm phái sinh hoặc *chuyển thể* là các sản phẩm dựa trên các tác phẩm hiện hữu (chẳng hạn như bản dịch, dàn dựng âm nhạc, kịch, hư cấu, phiên bản điện ảnh, ghi âm, tái tạo nghệ thuật hoặc bất kỳ hình thức nào khác mà trong đó tác phẩm gốc được chuyển thể hoặc phỏng theo). Sản phẩm phái sinh sẽ trở thành một tác phẩm thứ hai, tách biệt về hình thức so với tác phẩm thứ nhất. Việc chuyển đổi, sửa đổi hoặc điều chỉnh tác phẩm phải được thể hiện một cách rõ ràng để thể hiện được tính nguyên bản và từ đó được bảo vệ bởi bản quyền.

Các Tính năng cấp phép Nội dung Mở phổ biến

Mỗi một giấy phép nội dung mở đều sẽ khẳng định bản quyền của tác giả và rằng nếu không có giấy phép từ tác giả thì bất kỳ người nào sử dụng tác phẩm đều sẽ vi phạm bản quyền. Do đó, những giấy phép như vậy hoạt động trong khuôn khổ hệ thống bản quyền toàn cầu chứ không hề đi ngược lại.

Giấy phép nội dung mở cũng đảm bảo rằng tác giả của tác phẩm phải được ghi công xứng đáng. Nếu người nhận tác phẩm phân phối nó cho bên thứ ba, họ sẽ phải đảm bảo rằng tác giả gốc sẽ được thừa nhận và ghi nhận. Ngoài ra, khi người nhận sửa đổi tác phẩm, sản phẩm phái sinh sẽ phải đề cập rõ ràng về tác giả của bản gốc và nơi có thể tìm thấy bản gốc.

Không giống như hầu hết các giấy phép bản quyền áp đặt các điều kiện hạn chế đối với việc sử dụng tác phẩm, giấy phép nội dung mở cho phép người dùng có một sự tự do nhất định bằng cách cấp quyền cho họ. Một số quyền này là quyền chung cho hầu hết tất cả các giấy phép nội dung mở (chẳng hạn như quyền sao chép tác phẩm và quyền phân phối tác phẩm). Tùy thuộc vào giấy phép, người dùng cũng có thể có quyền sửa đổi, tạo, biểu diễn, trưng bày và phân phối các sản phẩm phái sinh.

Giấy phép nội dung mở có thể kiểm soát các sản phẩm phái sinh. Các giấy phép này thường bao gồm quyền tạo sản phẩm phái sinh và phân phối nó trên bất kỳ một phương tiện truyền thông nào. Nếu một người cấp phép cho một bức tranh theo giấy phép nội dung mở thì quyền lấy nó làm cơ sở cho một bức tranh khác cũng có thể được cấp. Các quyền này có thể được cấp với điều kiện là những người khác có thể sử dụng sản phẩm phái sinh một cách tự do giống như tác phẩm gốc.

Do đó, giấy phép nội dung mở thường đảm bảo rằng các sản phẩm phái sinh sẽ được cấp phép theo các điều khoản và điều kiện của cùng một giấy phép nội dung mở. Nhưng nghĩa vụ này không được áp dụng khi tác phẩm được đưa vào tuyển tập. Ví dụ: nếu một người tạo ra một album gồm các bài hát, một trong số đó được cấp phép theo giấy phép nội dung mở, thì không nhất thiết tất cả các bài hát đều phải được cấp phép theo cùng các điều khoản.

Một khía cạnh quan trọng khác của giấy phép nội dung mở là kiểm soát việc sử dụng một tác phẩm với mục đích thương mại. Mọi người có thể cấp phép cho tác phẩm của mình theo giấy phép nội dung mở, đồng thời hạn chế các quyền đối với các mục đích phi thương mại. Ngoài ra, mọi người cũng có thể cấp tất cả các quyền, bao gồm cả quyền sử dụng tác phẩm cho mục đích thương mại.

Giấy phép nội dung mở không ngăn cản mọi người kiếm tiền từ công việc của họ. Nếu tác phẩm tuân theo giấy phép phi thương mại, nhà xuất bản hoặc tổ chức thương mại khác có thể xuất bản tác phẩm đó bằng cách thỏa thuận với chủ sở hữu bản quyền trước khi thực hiện việc đó. Nói cách khác, ngay cả sau khi phát hành một tác phẩm trên cơ sở phi thương mại, tác giả vẫn có thể bán bản quyền cho một tổ chức vì lợi nhuận với điều kiện là không có sự loại trừ nào đối với việc tiếp tục sử dụng phi thương mại.

Tầm quan trọng của Giấy phép Nội dung Mở

Một số lợi ích của mô hình nội dung mở là gì và tại sao mọi người lại sử dụng giấy phép nội dung mở để phân phối tác phẩm sáng tạo của họ thay vì dựa vào mô hình bản quyền truyền thống?

Giấy phép nội dung mở cho phép các tác phẩm được lưu hành rộng rãi hơn nếu chúng bị hạn chế theo mặc định. Do đó, các nghệ sĩ mới có thể được hưởng lợi từ những giấy phép này vì họ có thể trở nên nổi tiếng hơn và được nhiều người công nhận hơn. Bằng cách từ bỏ một số quyền kiểm soát nhất định (và có thể cả doanh thu đi kèm với chúng), nhà sáng tạo nội dung có thể được mời tham gia nhiều chương trình hơn, nhận được nhiều tài trợ hoặc cơ hội hợp tác hơn, v.v.

Giấy phép nội dung mở khá thực tế trong thời đại internet bởi các nhà sáng tạo có thể phát hành tác phẩm của họ mà không cần dựa vào các cá nhân hoặc tổ chức khác. Ví dụ: một nhiếp ảnh gia muốn trưng bày ảnh của họ có thể phát hành chúng trên trang web cá nhân. Bằng cách này, họ có thể tự khẳng định mình và được công chúng biết đến một cách rộng rãi hơn mà không cần đăng ký với một đại lý hoặc phòng trưng bày để thực hiện việc này.

Bằng cách lựa chọn một giấy phép phù hợp, chủ sở hữu bản quyền có thể tối đa hóa việc phân phối và duy trì quyền kiểm soát việc thương mại hóa tác phẩm của mình. Nếu mọi người muốn sử dụng tác phẩm cho mục đích thương mại, nhà sáng tạo nội dung có thể giữ quyền cấp hoặc từ chối cấp phép. Ngay cả khi những người khác bị cấm sử dụng vì mục đích thương mại, chủ sở hữu quyền vẫn có thể sử dụng tác phẩm của mình vì mục đích thương mại.

Bên cạnh khả năng phân phối tác phẩm rộng rãi hơn rất nhiều, các giấy phép nội dung mở cũng làm tăng sự chắc chắn về mặt pháp lý cho người dùng và giảm đáng kể chi phí giao dịch pháp lý.

Chúng ta cũng có thể sử dụng các mô hình tài trợ không phụ thuộc vào việc sử dụng giấy phép phi thương mại. Ví dụ: nhiều nghệ sĩ và nhà sáng tạo sử dụng phương thức huy động vốn từ cộng đồng để tài trợ cho tác phẩm của họ trước khi phát hành nó theo một giấy phép linh hoạt. Những người khác lại sử dụng một mô hình mà trong đó nội dung cơ bản sẽ được miễn phí, nhưng các tính năng bổ sung như phiên bản in hoặc quyền truy cập đặc biệt vào trang web chỉ dành riêng cho thành viên sẽ chỉ được dành cho các khách hàng trả phí.

Giấy phép nội dung mở cho phép mọi người phân phối tác phẩm của họ cho bất kỳ ai và trên bất kỳ phương tiện và định dạng nào như trang web, bản sao, đĩa CD hoặc sách mà không bị hạn chế.

Giấy phép nội dung mở sẽ tự động thiết lập giấy phép giữa tác giả và người dùng. Nếu không có giấy phép nội dung mở, việc chia sẻ tác phẩm qua một nguồn trực tuyến khác sẽ yêu cầu một thỏa thuận hợp đồng cá nhân giữa tác giả và người dùng.

Nhãn hiệu và Bản quyền

Giống như bản quyền, nhãn hiệu (trademarks) cũng là một loại tài sản trí tuệ. Nhưng luật bảo vệ nhãn hiệu khác với luật bảo vệ bản quyền. Nhãn hiệu sẽ bảo vệ thương hiệu, các từ liên quan như tên thương hiệu, logo, biểu tượng và thậm chí cả âm thanh và màu sắc được sử dụng để phân biệt loại hàng hóa và dịch vụ cụ thể với các loại hàng hóa và dịch vụ khác. Bất kỳ yếu tố đặc biệt nào được sử dụng để quảng bá và phân biệt doanh nghiệp cũng như dịch vụ hoặc sản phẩm được bán với người khác đều có thể là một nhãn hiệu.

Người nắm giữ nhãn hiệu thường có thể ngăn người khác sử dụng các sản phẩm đã được đăng ký nhãn hiệu này nếu điều đó có thể gây ra nhầm lẫn cho công chúng. Luật về nhãn hiệu sẽ giúp các nhà sản xuất hàng hóa và dịch vụ bảo vệ danh tiếng của họ cũng như bảo vệ công chúng bằng cách cung cấp cho họ một cách đơn giản để phân biệt họ với các sản phẩm và dịch vụ tương tự. Nhãn hiệu có thể được đăng ký chính thức hoặc được tự động bảo vệ theo luật chung. Điều này có nghĩa là một nhãn hiệu sẽ tồn tại ngay khi nó được sử dụng, nhưng một nhãn hiệu tuân theo luật thông thường sẽ không mang lại sự bảo vệ pháp lý giống như một nhãn hiệu đã được đăng ký. Đó là lý do tại sao nhiều công ty lại đăng ký nhãn hiệu của riêng họ.

Bản quyền và nhãn hiệu có thể cùng lúc tồn tại. Ví dụ: Wikipedia là một nhãn hiệu đã được đăng ký và biểu tượng của nó cũng được bảo vệ theo luật bản quyền dưới dạng một tác phẩm nghệ thuật hoặc tác phẩm gốc.

Bài tập Hướng dẫn

1. Giấy phép nội dung mở có thể được sử dụng để tránh bản quyền không?

2. Nội dung như thế nào sẽ được coi là nội dung mở?

3. Sản phẩm phái sinh là gì?

Bài tập Mở rộng

1. Bạn sẽ làm gì nếu muốn phát hành một tác phẩm và cho phép mọi người sử dụng nó cho bất kỳ mục đích nào, miễn là họ tái phân phối tác phẩm đó theo cùng các quyền và điều kiện?

2. Bạn có thể phân phối các sản phẩm phái sinh thương mại dựa trên các tác phẩm được xuất bản theo giấy phép nội dung mở không?

Tóm tắt

Trong bài học này, bạn đã học về:

- Nguyên tắc cơ bản của luật bản quyền
- Nội dung nào có thể được coi là nội dung có thể được bảo vệ bản quyền
- Sản phẩm phái sinh hoặc chuyển thể là gì
- Các tính năng chung được chia sẻ bởi các giấy phép nội dung mở
- Các loại nội dung mở
- Tầm quan trọng và lợi ích của mô hình cấp phép nội dung mở
- Bản quyền và nhãn hiệu là các loại hình sở hữu trí tuệ

Đáp án Bài tập Hướng dẫn

1. Giấy phép nội dung mở có thể được sử dụng để tránh bản quyền không?

   Giấy phép nội dung mở không thể được sử dụng để tránh bản quyền. Giấy phép nội dung mở là một loại giấy phép bản quyền cấp một số quyền theo những điều kiện nhất định, đồng thời duy trì các quyền độc quyền của chủ sở hữu bản quyền.

2. Nội dung như thế nào sẽ được coi là nội dung mở?

   Nội dung mở có thể là bất kỳ một tác phẩm có bản quyền nào được xuất bản theo giấy phép nội dung mở. Nội dung này có thể là phim, nhạc, hình ảnh, văn bản, cơ sở dữ liệu, tài liệu, bản đồ và thiết kế phần cứng cũng như các dạng tác phẩm sáng tạo khác. Các loại giấy phép này cấp các quyền như sử dụng, tái phân phối, tạo sản phẩm phái sinh và sử dụng thương mại hoặc phi thương mại mà không cần bất kỳ sự cho phép đặc biệt nào.

3. Sản phẩm phái sinh là gì?

   Sản phẩm phái sinh — hay còn được gọi là bản chuyển thể — là một tác phẩm dựa trên một tác phẩm sẵn có, trong đó tác phẩm gốc đã được chuyển đổi hoặc chuyển thể. Các bản dịch, bản soạn nhạc, kịch, phiên bản điện ảnh, bản ghi âm và tái tạo tác phẩm nghệ thuật là những ví dụ về sản phẩm phái sinh.

Đáp án Bài tập Mở rộng

1. Bạn sẽ làm gì nếu muốn phát hành một tác phẩm và cho phép mọi người sử dụng nó cho bất kỳ mục đích nào, miễn là họ tái phân phối tác phẩm đó theo cùng các quyền và điều kiện?

   Bạn có thể cung cấp tác phẩm theo giấy phép copyleft. Bằng cách này, tất cả các sản phẩm phái sinh của bản gốc cũng phải được để mở và phát hành theo cùng một giấy phép hoặc bất kỳ một giấy phép tương thích nào khác.

2. Bạn có thể phân phối các sản phẩm phái sinh thương mại dựa trên các tác phẩm được xuất bản theo giấy phép nội dung mở không?

   Điều này phụ thuộc vào giấy phép mà tác phẩm gốc tuân theo. Nếu giấy phép đó quy định rằng bạn có thể sử dụng sản phẩm phái sinh cho bất kỳ mục đích nào, thậm chí là về mặt thương mại, thì bạn có thể.

053.2 Giấy phép Tài sản Sáng tạo Công cộng (Creative Commons)

Tham khảo các mục tiêu LPI

Open Source Essentials version 1.0, Exam 050, Objective 053.2

Khối lượng

2

Các lĩnh vực kiến thức chính

- Hiểu rõ khái niệm về giấy phép Tài sản Sáng tạo Công cộng
- Hiểu rõ về các loại giấy phép Tài sản Sáng tạo Công cộng và sự kết hợp của chúng
- Hiểu rõ về các quyền được cấp bởi giấy phép Tài sản Sáng tạo Công cộng
- Hiểu các nghĩa vụ do giấy phép Tài sản Sáng tạo Công cộng áp đặt

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Tài sản Phạm vi Công cộng (CC0)
- Tài sản Sáng tạo Công cộng Ghi công (CC BY)
- Tài sản Sáng tạo Công cộng Ghi công-Chia sẻ thống nhất (CC BY-SA)
- Tài sản Sáng tạo Công cộng Ghi công-Phi thương mại (CC BY-NC)
- Tài sản Sáng tạo Công cộng Ghi công-Phi thương mại-Chia sẻ thống nhất (CC BY-NC-SA)
- Tài sản Sáng tạo Công cộng Ghi công-Không phái sinh (CC BY-ND)
- Tài sản Sáng tạo Công cộng Ghi công-Phi thương mại-Không phái sinh (CC BY-NC-ND)

Linux Professional Institute

Bài 1

Chứng chỉ:	Open Source Essentials
Phiên bản:	1.0
Chủ đề:	053 Giấy phép dành cho Nội dung Mở
Mục tiêu:	053.2 Giấy phép Tài sản Sáng tạo Công cộng (Creative Commons)
Bài học:	1 trên 1

Giới thiệu

Thành công của phần mềm tự do từ những năm 1990 song song với sự xâm chiếm của Internet đã đánh thức khát vọng có được những điều kiện tương tự trong các lĩnh vực khác để hỗ trợ sự phát triển và phân phối các sản phẩm sáng tạo — tức các tác phẩm về cơ bản là được bảo hộ bản quyền. Những mong muốn này phản ánh một loạt các lợi ích khác nhau.

Các nghệ sĩ sáng tạo sẽ muốn tự mình xác định các điều kiện để theo đó các tác phẩm của họ có thể được người khác sử dụng. Tuy nhiên, các nghệ sĩ đồng thời cũng sẽ quan tâm đến việc sử dụng tác phẩm của những người khác cho bản thân mình, có thể là bằng cách trích dẫn, chỉnh sửa hoặc phỏng theo giống như các quá trình sáng tạo vẫn luôn cần tới. Đối với người tiếp nhận tác phẩm, các câu hỏi được đặt ra xoay quanh vấn đề tiềm năng sử dụng (ví dụ như liệu tác phẩm có thể được sao chép hoặc truyền lại dưới một hình thức nào hay không).

Ngoài việc làm rõ những câu hỏi thực tế như vậy, các quy định cũng phải được thực hiện theo một cách phù hợp với các chuẩn mực pháp lý hiện hành - chủ yếu là luật bản quyền; điều này đã trở nên khó khăn hơn do luật bản quyền được điều chỉnh theo nhiều cách khác nhau trên phạm vi quốc tế.

Cuối cùng nhưng không kém phần quan trọng, các quy tắc phải dễ hiểu và dễ áp dụng để đẩy nhanh quá trình sáng tạo và mang lại cho nhà sáng tạo cũng như người tiếp nhận sự chắc chắn về mặt pháp lý.

Nguồn gốc và Mục tiêu của Tài sản Sáng tạo Công cộng

Các giấy phép đầu tiên trong lĩnh vực phần mềm tự do, trên hết là Giấy phép Công cộng chung GNU, đã thực hiện thành công việc tiên phong trong vấn đề cung cấp một khuôn khổ pháp lý đáng tin cậy cho các quy trình hoàn toàn mới xoay quanh sự phát triển hợp tác của phần mềm.

Năm 2001, một nhóm do giáo sư luật Lawrence Lessig từ Trường Luật Stanford dẫn đầu đã thành lập một tổ chức phi lợi nhuận có tên là *Tài sản Sáng tạo Công cộng* (Creative Commons - CC) lấy cảm hứng từ phần mềm tự do. Mục tiêu của tổ chức được tóm tắt trên trang web của dự án như sau:

> Tài sản Sáng tạo Công cộng là một tổ chức phi lợi nhuận toàn cầu cho phép chia sẻ và tái sử dụng tính sáng tạo và kiến thức thông qua việc cung cấp các công cụ pháp lý miễn phí.
>
> — Creative Commons, Website (FAQ)

Với thuật ngữ "công cộng" đề cập đến một hình thức hoạt động kinh tế cộng đồng đã được ghi nhận từ thời Trung Cổ, tổ chức này nhấn mạnh tới một nhu cầu sâu sắc và có thể được kiểm chứng về mặt lịch sử của con người đối với một hành động tập thể hướng tới lợi ích chung. Do đó, nhiệm vụ tự đặt ra của họ là tạo ra một khuôn khổ pháp lý hiện đại cho công việc sáng tạo. Tài sản Sáng tạo Công cộng chuyển các nhu cầu và giải pháp của phong trào phần mềm tự do sang các lĩnh vực sáng tạo khác như văn học, nghệ thuật biểu diễn (hội họa, đồ họa, nhiếp ảnh, video, v.v.) và âm nhạc, cũng như các tài liệu và sản phẩm khoa học — nói một cách đại khái thì là tất cả các sản phẩm có bản quyền.

Tương tự như Tổ chức Phần mềm Tự do, Tài sản Sáng tạo Công cộng cũng chọn thực hiện các mục tiêu của mình thông qua *giấy phép*: tập hợp các thông số kỹ thuật được tiêu chuẩn hóa và ràng buộc về mặt pháp lý mà nhà sáng tạo đã chỉ định rõ ràng cho tác phẩm của họ, thường là trước khi họ "phát hành" tác phẩm ra công chúng — tức là xuất bản chúng.

Nguyên tắc "bảo lưu mọi quyền" được quy định trong luật bản quyền của Hoa Kỳ được coi là quá hạn chế khi xét đến tiềm năng kỹ thuật đang ngày càng tăng cao sẵn có cho các quy trình sáng tạo. Bản thân các nhà sáng tạo thường cũng không nắm rõ được mức độ cho phép để xây dựng tác phẩm của mình dựa trên tác phẩm của người khác mà không vượt quá giới hạn của luật bản quyền và, trong trường hợp xấu nhất, có thể khiến bản thân họ bị truy tố. Tài sản Sáng tạo Công cộng đối lập lại với khuôn khổ hà khắc này bằng nguyên tắc "một số quyền được bảo lưu": những nhà sáng tạo sẽ nêu tên những quyền mà họ muốn bảo lưu — và từ bỏ tất cả các quyền khác.

Sự kết hợp của vỏn vẹn bốn điều kiện (mô-đun) cơ bản đơn giản sẽ tạo ra tổng cộng sáu giấy phép mà từ đó tác giả có thể chọn và cấp giấy phép thích hợp cho tác phẩm của mình.

Mô-đun giấy phép Tài sản Sáng tạo Công cộng

Bốn mô-đun vừa được đề cập tương ứng với bốn quyết định cơ bản đơn giản mà tác giả đưa ra liên quan đến việc sử dụng và phân phối tác phẩm. Mỗi mô-đun có thể được biểu thị bằng chữ viết tắt gồm hai chữ cái và một ký hiệu. Định nghĩa tương ứng sẽ rất ngắn gọn và dễ hiểu ngay cả đối với những người không có chuyên môn về pháp lý; điều này góp phần rất lớn vào sự phổ biến của các giấy phép CC. Tuy nhiên, trong một số trường hợp cá biệt, các điều khoản cũng có thể dẫn đến những câu hỏi chưa được giải quyết hoặc những vùng xám.

Ghi công Tác giả (BY)

> Bạn phải ghi công tác giả một cách phù hợp, cung cấp liên kết tới giấy phép và cho biết liệu các thay đổi đã được thực hiện hay chưa. Bạn có thể làm như vậy theo bất kỳ cách nào, nhưng không được theo bất kỳ cách nào ám chỉ rằng người cấp phép đã xác nhận bạn hoặc việc sử dụng của bạn.
>
> — Creative Commons, Attribution

Từ tiếng Anh “by” chỉ ra rằng tác giả phải được nêu tên, tức là tác phẩm phải được ghi rõ ràng là của tác giả nếu nó được sử dụng, sao chép hoặc chuyển giao dưới bất kỳ hình thức nào. Mô-đun Ghi công tác giả (Attribution - BY) là mô-đun bắt buộc duy nhất trong *tất cả* các giấy phép CC. Nói cách khác, không có tác phẩm nào tuân theo sáu giấy phép CC tiêu chuẩn thoát ly khỏi việc ghi công tác giả.

Yêu cầu đơn giản này có thể gây ra những vấn đề thực tế trong các dự án hợp tác được phát triển thông qua internet. Ví dụ: trong các sản phẩm bắt nguồn từ bách khoa toàn thư trực tuyến Wikipedia, thật khó để có thể “ghi công tác giả một cách phù hợp” với hàng nghìn nhà đóng góp.

Phi Thương mại (NC)

> Bạn không được sử dụng tài liệu cho mục đích thương mại.
>
> — Creative Commons, NonCommercial

Mục đích của mô-đun *Phi thương mại* (NonCommercial) thoạt nhìn có vẻ rất rõ ràng: nhằm ngăn chặn việc một tác phẩm có sẵn miễn phí bị người khác chiếm đoạt vì mục đích thương mại. Điều này thường liên quan đến các sản phẩm được phát triển bằng nguồn vốn công (chẳng hạn như tại các trường đại học). Chúng được "bảo vệ" khỏi việc thương mại hóa để đảm bảo chất lượng và tính sẵn có miễn phí lâu dài của mình.

Trên thực tế, mô-đun NC thường gây ra nhiều sự không chắc chắn vì nó không hề rõ ràng trong từng trường hợp khi việc sử dụng thực sự có mang tính thương mại. Ví dụ: nếu các giáo viên sử dụng tài liệu tuân theo một giấy phép CC với mô-đun NC và dạy ở một trường tư thục hoặc một trường đại học tư mà sinh viên phải trả tiền để theo học thì tức là họ đã nằm trong một vùng xám về mặt pháp lý.

Nhiều nhà phê bình mô-đun NC cũng bác bỏ lập luận rằng các tài liệu tự do đã trở thành "không tự do" thông qua việc sử dụng thương mại vì các tác phẩm vẫn đang *đồng thời* có sẵn miễn phí.

Không Phái sinh (ND)

> Nếu bạn phối lại, biến đổi hoặc xây dựng dựa trên tài liệu, bạn không được phân phối tài liệu đã sửa đổi.
>
> — Creative Commons, NoDerivs

Phái sinh (Derivatives, viết tắt là *Derivs*) nghĩa là các tác phẩm/ sản phẩm phái sinh (đã được đề cập ở các bài học trước). Trong bối cảnh của Tài sản Sáng tạo Công cộng, thuật ngữ này cũng đề cập đến việc tạo ra các sản phẩm mới trên cơ sở các sản phẩm hiện có được bảo vệ quyền tác giả. Để lấy ví dụ, sản phẩm phái sinh có thể là tài liệu học tập mà giáo viên điều chỉnh cho phù hợp với bài học của mình hoặc bản dịch của một cuốn tiểu thuyết sang một ngôn ngữ khác. *Không phái sinh* loại trừ một cách rõ ràng việc phân phối các bản sửa đổi hoặc chuyển thể kiểu này: tác phẩm chỉ có thể được truyền đạt dưới hình thức xuất bản bởi chính tác giả.

Chia sẻ Tương tự (SA)

> Nếu bạn phối lại, biến đổi hoặc xây dựng dựa trên tài liệu, bạn phải phân phối phần đóng góp của mình theo cùng một giấy phép giống như bản gốc.
>
> — Tài sản Sáng tạo Công cộng, ShareAlike

Mô-đun *Chia sẻ Tương tự* (ShareAlike) chỉ ra rằng một tác phẩm chỉ có thể được chia sẻ trong cùng các điều kiện với giấy phép đã được chỉ định. SA được coi là đối trọng với nguyên tắc copyleft trong các giấy phép phần mềm tự do; nó đảm bảo rằng các quyền tự do do giấy phép cấp cũng được áp dụng mà không có sự thay đổi đối với các sản phẩm phái sinh.

Mô-đun SA đặc biệt thú vị khi kết hợp với các mô-đun khác vì yêu cầu của SA cũng sẽ cập nhật các điều kiện được xác định khác như chúng ta sẽ thấy khi thảo luận về các giấy phép khác nhau.

Các Giấy phép cốt lõi của Tài sản Sáng tạo Công cộng

Sự kết hợp giữa các mô-đun vừa được giới thiệu ở trên sẽ tạo ra tổng cộng sáu giấy phép. Chỉ có

sáu vì không phải sự kết hợp nào giữa chúng cũng khả thi. Ví dụ: yêu cầu chia sẻ ngay cả các sản phẩm đã được sửa đổi trong cùng các điều kiện (ShareAlike) và việc cấm chuyển thể (NoDerivs) sẽ loại trừ lẫn nhau. Do đó, không có giấy phép nào có cả hai mô-đun này. Vì quyền tác giả cũng là một phần của tất cả các giấy phép nên kết quả là chúng ta có *các giấy phép cốt lõi* sẽ được giới thiệu sau đây.

Danh sách các giấy phép này cũng có thể được hình dung như một thước đo từ "nhiều quyền tự do" đến "ít quyền tự do", vì tác giả sẽ trả lời các câu hỏi về khả năng sử dụng hoặc các hạn chế liên quan đến tác phẩm theo từng bước và từ đó sẽ đạt được một giấy phép đúng như mong muốn.

Tài sản Sáng tạo Công cộng Ghi công (CC BY)

Việc ghi công tác giả đã được nhắc tới như một thành phần bắt buộc của tất cả các giấy phép và vì vậy, giấy phép đầu tiên — *Tài sản Sáng tạo Công cộng Ghi công* (CC Attribution hoặc CC BY) — chỉ bao gồm mô-đun này. Do đó, một tác phẩm có thể được phát hành cho bất kỳ hình thức sử dụng, sửa đổi và phân phối nào, miễn là đáp ứng được yêu cầu ghi công tác giả.

Figure 5. CC BY Icon

Ví dụ: nếu một tác giả đã xuất bản một bài thơ theo CC BY, nhà xuất bản có thể đưa nó vào một tập thơ mà không cần hỏi ý kiến tác giả cũng như không có bất kỳ nghĩa vụ tài chính nào và thậm chí có thể phân phối nó qua các hiệu sách, miễn là bài thơ này được đề rõ ràng là tác phẩm của tác giả.

Tài sản Sáng tạo Công cộng Ghi công-Chia sẻ Tương tự (CC BY-SA)

Tài sản Sáng tạo Công cộng Ghi công-Chia sẻ Tương tự tương ứng với CC BY nhưng cũng bổ sung cả các yêu cầu của *Chia sẻ Tương tự* — tức là phân phối các phiên bản đã sửa đổi của tác phẩm dưới cùng các điều kiện.

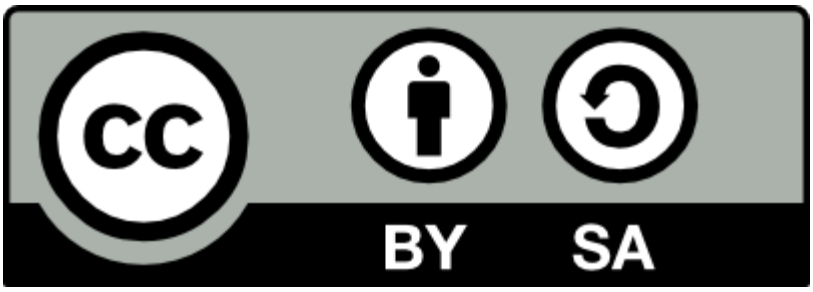

Figure 6. CC BY-SA Icon

Ví dụ: nếu một ca sĩ đặt bài hát của mình theo giấy phép CC BY-SA, một ban nhạc khác có thể hát lại hoặc phỏng theo bài hát này, miễn là họ cũng phát hành phiên bản bài hát của mình theo giấy phép CC BY-SA hoặc một giấy phép khác tương thích với nó.

Tài sản Sáng tạo Công cộng Ghi công-Phi thương mại (CC BY-NC)

Tài sản Sáng tạo Công cộng Ghi công-Phi thương mại là giấy phép đầu tiên trong chuỗi liên kết việc sử dụng tác phẩm với một lệnh cấm bằng cách loại trừ việc phân phối và chuyển thể/ phóng tác cho mục đích thương mại.

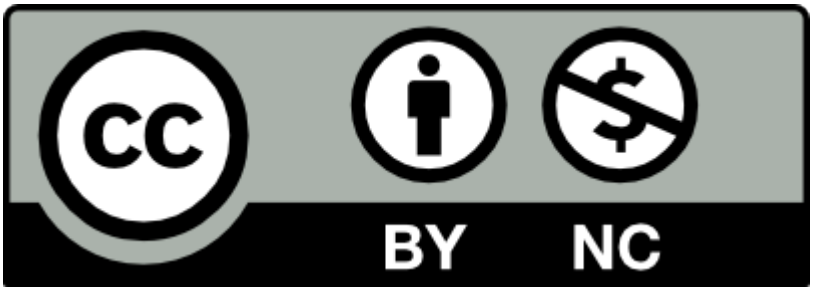

Figure 7. CC BY-NC Icon

Trong ví dụ, một ban nhạc sẽ được phép chuyển thể bài hát của ca sĩ nhưng không được sử dụng bản chuyển thể này cho mục đích thương mại (chẳng hạn như bán bài hát đó trên đĩa nhạc của chính họ hoặc chơi nó tại các buổi hòa nhạc mà họ được trả phí). Tuy nhiên, ban nhạc sẽ được phép cấm bổ sung việc chuyển thể phiên bản của họ — tức là xuất bản bản chuyển thể theo giấy phép CC BY-NC-ND.

Tài sản Sáng tạo Công cộng Ghi công-Phi thương mại-Chia sẻ Tương tự (CC BY-NC-SA)

Trong trường hợp của *Tài sản Sáng tạo Công cộng Ghi công-Phi thương mại-Chia sẻ Tương tự*, việc cấm sử dụng thương mại có liên quan đến việc phân phối trong cùng điều kiện. Tiếp tục với ví dụ của chúng ta: mặc dù ban nhạc được phép hát lại bài hát của ca sĩ nhưng không được phép thêm ND vào bản hát lại vì bản hát lại đó phải được phân phối dưới cùng các điều kiện.

Figure 8. CC BY-NC-SA Icon

Tài sản Sáng tạo Công cộng Ghi công-Không phái sinh (CC BY-ND)

Giống như Phi thương mại, *CC Ghi công-Không phái sinh* cũng dựa trên một lệnh cấm — trong trường hợp này là lệnh cấm sửa đổi hoặc chuyển thể/ phỏng theo một tác phẩm. Do đó, ban nhạc trong ví dụ của chúng ta sẽ không được phép phân phối phiên bản hát lại bài hát của ca sĩ.

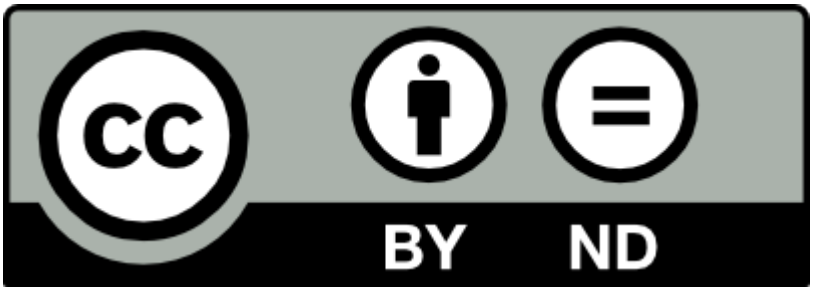

Figure 9. CC BY-ND Icon

Việc sử dụng giấy phép CC có điều khoản ND (tức là các giấy phép CC BY-ND và giấy phép CC BY-NC-ND sẽ được giới thiệu tiếp theo đây) được nhìn nhận với một thái độ hoài nghi, đặc biệt là trong lĩnh vực khoa học, vì chúng có thể cản trở việc trao đổi kiến thức cần thiết cho tiến bộ khoa học. Ngay cả trên trang web Tài sản Sáng tạo Công cộng, các giấy phép có mô-đun ND cũng được mô tả là không tương thích với nguyên tắc truy cập mở phổ biến trong khoa học. Để lấy một ví dụ về xu hướng hạn chế quá mức của ND, người ta đã trích dẫn rằng ngay cả các bản dịch của các bài báo khoa học cũng được coi là các sản phẩm phái sinh và do đó sẽ bị cấm.

Tài sản Sáng tạo Công cộng Ghi công-Phi thương mại-Không phái sinh (CC BY-NC-ND)

Giấy phép CC hạn chế nhất — *Tài sản Sáng tạo Công cộng Ghi công-Phi thương mại-Không phái sinh* — kết hợp cả việc cấm sử dụng thương mại và việc phân phối các sản phẩm phái sinh. Nói một cách tích cực, điều này có nghĩa là một tác phẩm chỉ có thể được sao chép và phân phối dưới hình thức do tác giả xuất bản và chỉ được sử dụng cho mục đích phi thương mại.

Figure 10. CC_BY-NC-ND_Icon

Do đó, ca sĩ trong ví dụ của chúng ta có thể ngăn một ban nhạc hát lại bài hát của cô ấy (ND), nhưng đồng thời bài hát đó cũng có thể có nguy cơ bị loại khỏi bất kỳ nền tảng nào tạo ra doanh thu từ khách hàng hoặc quảng cáo.

Tài sản Sáng tạo Công cộng Không (CC0) và Nhãn Phạm vi Công cộng

Chúng ta đã biết rằng luật bản quyền được quy định rất khác nhau giữa các quốc gia và khu vực pháp lý khác nhau. Ví dụ: theo luật pháp của Hoa Kỳ, chủ sở hữu quyền hoàn toàn có thể từ bỏ bản quyền của mình. Bằng cách đó, họ có thể chuyển tác phẩm của mình sang cái gọi là *Phạm vi công cộng* - tức là sang mục đích sử dụng chung và không hạn chế.

Tuy nhiên, ở hầu hết các nước Châu Âu, ngoài các quyền sử dụng thuần túy, bản quyền còn bao gồm các quyền cá nhân thường không thể chuyển nhượng và do đó tác giả không thể từ bỏ. Nếu một tác giả muốn phát hành một tác phẩm ra cho công chúng, họ sẽ từ bỏ mọi quyền sử dụng nhưng vẫn là tác giả.

Ngoài ra, có những tác phẩm mà việc sử dụng nó không bị hạn chế một chút nào. Các tác phẩm này có thể bao gồm những tác phẩm đã hết thời hạn bảo hộ pháp lý (chẳng hạn như cuốn tiểu thuyết 70 năm sau khi tác giả qua đời) hoặc những tác phẩm về nguyên tắc chưa bao giờ được bảo vệ (chẳng hạn như các văn bản pháp luật).

Tài sản Sáng tạo Công cộng cung cấp hai *công cụ phạm vi công cộng* để xác định các tác phẩm có sẵn cho công chúng mà không bị hạn chế.

Tài sản Sáng tạo Công cộng Không (CC0) hầu như không có điều kiện nào cho việc sao chép, phân phối và sửa đổi tác phẩm. Kể cả việc ghi công tác giả bắt buộc trong tất cả các giấy phép CC cũng bị bỏ qua. Giấy phép này cho phép chủ sở hữu bản quyền đánh dấu tác phẩm của họ thuộc phạm vi công cộng trong nhiều hệ thống pháp luật nhất có thể.

Figure 11. CC0 Icon

Hãy lấy một tài liệu học tập được phát triển cho mục đích giảng dạy làm ví dụ. Tác giả hoặc các tác giả có thể sử dụng nhãn CC0 để đảm bảo rằng tài liệu này có thể được sử dụng bởi bất kỳ ai mà không có bất kỳ hạn chế nào: nó có thể được sao chép toàn bộ hoặc một phần, phân phối miễn phí hoặc có tính phí và được tích hợp toàn bộ hoặc một phần vào các tài liệu khác. Các tác giả ban đầu không cần phải được nêu tên ở bất cứ đâu.

Để xác định các tác phẩm không bị hạn chế về bản quyền, Tài sản Sáng tạo Công cộng khuyến nghị sử dụng *Nhãn Phạm vi Công cộng*:

> Nhãn Phạm vi công cộng hoạt động như một chiếc thẻ hoặc nhãn hiệu; nó cho phép các tổ chức truyền đạt rằng tác phẩm không còn bị hạn chế bởi bản quyền và có thể được sử dụng tự do bởi những người khác.
>
> — Creative Commons, Public Domain Mark

Figure 12. Public Domain Mark Icon

Chọn Giấy phép và Đánh dấu Nhãn cho Tác phẩm

Chúng ta đã biết rằng Tài sản Sáng tạo Công cộng nhắm tới việc đơn giản hoá nhất có thể cho cả người cấp phép (nhà sáng tạo nội dung) và người được cấp phép (người tiếp nhận tác phẩm). Nói một cách cụ thể, tổ chức này cung cấp rất nhiều hỗ trợ, từ việc lựa chọn giấy phép thích hợp cho đến việc đánh dấu nhãn cho tác phẩm và sử dụng nó.

Bộ chọn Giấy phép CC (CC License Chooser) trên trang web của tổ chức đưa người dùng từng bước một tiến đến một đề xuất giấy phép thích hợp bằng cách đặt ra các câu hỏi đơn giản liên quan đến

mục đích sử dụng mong muốn đối với tác phẩm mà tác giả trả sẽ phải lời (CC License Chooser).

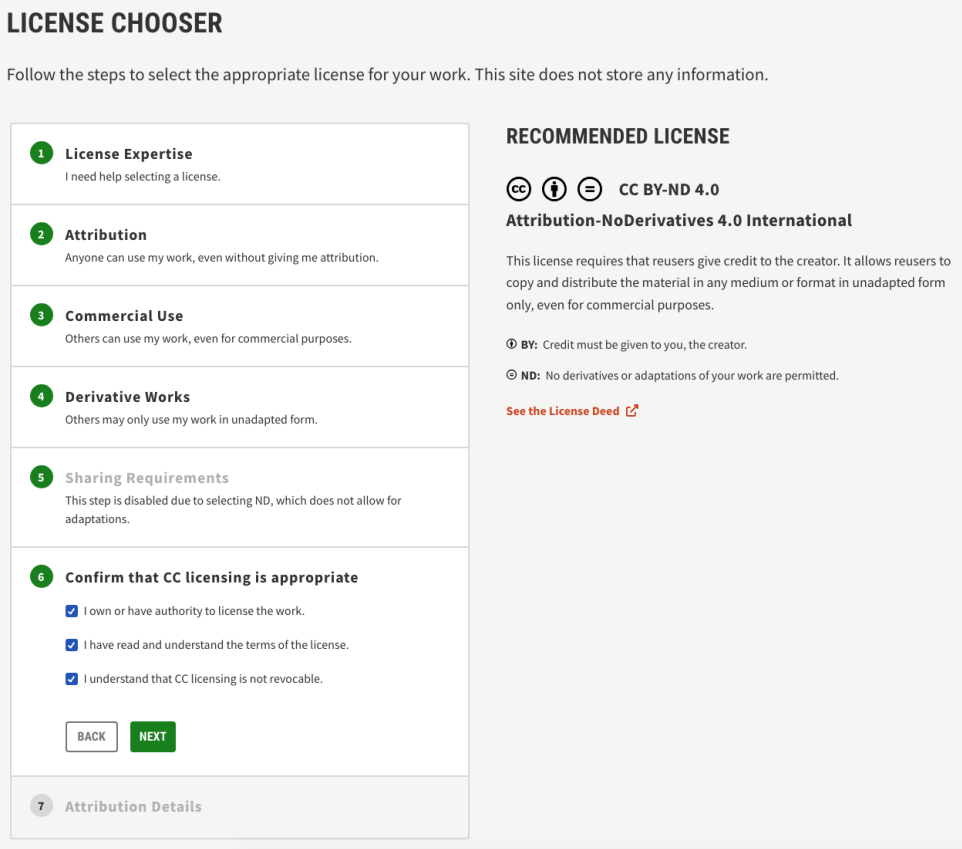

Figure 13. CC License Chooser

Nếu giấy phép được đề xuất phù hợp với ý định của tác giả, họ có thể theo đó đánh dấu nhãn cho tác phẩm của mình. Ví dụ: Bộ chọn Giấy phép cung cấp mã HTML mà tác giả có thể thêm trực tiếp vào trang web hiển thị tác phẩm (Mã HTML có liên kết đến giấy phép).

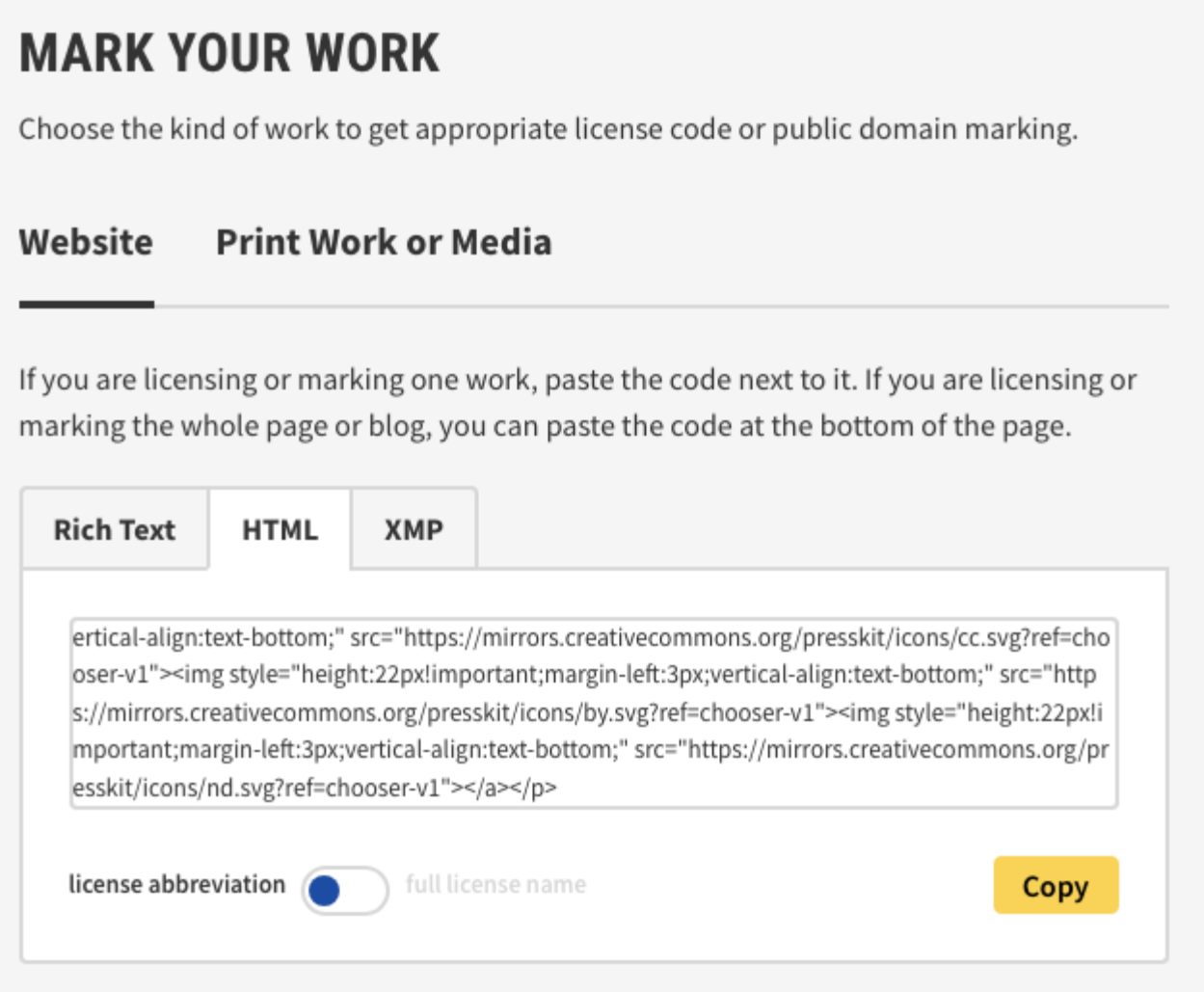

Figure 14. Mã HTML có liên kết đến giấy phép

Kết quả sau đó sẽ là một thông báo được chuẩn hóa với liên kết tới giấy phép tương ứng (Thông báo giấy phép có liên kết đến giấy phép).

Figure 15. Thông báo giấy phép có liên kết đến giấy phép

Các biểu tượng dành riêng cho giấy phép (trong đó các mô-đun đi kèm có thể được nhìn thấy trực tiếp) đã được giới thiệu ở trên. Chúng đã tự thiết lập bản thân mình thành các "điểm bắt mắt" trong việc xác định nội dung miễn phí trên internet.

Giấy phép Quốc tế và Giấy phép chuyển đổi

Ngay từ đầu, Tài sản Sáng tạo Công cộng đã tập trung vào việc phát triển các giấy phép mang tính tổng quát nhất có thể - tức là có giá trị trên toàn thế giới - được xác định tương ứng bằng việc bổ sung cụm từ “quốc tế” (trước đây là “chưa được chuyển đổi” - unported) . Mặt khác, có những biến thể có tính đến đặc thù của hệ thống pháp luật của một khu vực hoặc quốc gia và được gọi là “chuyển đổi” (ported). Ví dụ: ngoài giấy phép *CC BY-SA 3.0 Chưa chuyển đổi* (CC BY-SA 3.0 Unported), chúng ta còn có một phiên bản đặc biệt dành cho nước Đức dưới tên *CC BY-SA 3.0 Germany*.

Với phiên bản 4.0 hiện đang có hiệu lực của giấy phép CC, Tài sản Sáng tạo Công cộng đang nỗ lực tiêu chuẩn hóa hơn nữa và (cho đến nay) đã loại bỏ hoàn toàn các phiên bản chuyển đổi. Trong phiên bản 4.0, giấy phép “quốc tế” được khuyến nghị một cách mạnh mẽ:

> Chúng tôi khuyến nghị sử dụng giấy phép quốc tế phiên bản 4.0. Đây là phiên bản giấy phép được cập nhật mới nhất của chúng tôi, được soạn thảo sau khi tham khảo ý kiến rộng rãi với mạng lưới các chi nhánh toàn cầu của chúng tôi và được tạo ra để có hiệu lực quốc tế. Hiện tại không có các chuyển thể 4.0 và theo kế hoạch thì cũng sẽ có rất ít (nếu có).
>
> — Creative Commons, FAQ

Bài tập Hướng dẫn

1. Mô-đun nào của hệ thống giấy phép Tài sản Sáng tạo Công cộng là một thành phần có mặt trong toàn bộ sáu giấy phép Tài sản Sáng tạo Công cộng?

2. Mô-đun nào của hệ thống cấp phép Tài sản Sáng tạo Công cộng yêu cầu phân phối một tác phẩm theo cùng các điều kiện?

3. CC0 có gì khác so với sáu giấy phép CC cốt lõi?

4. Sự khác biệt giữa giấy phép “quốc tế” và “chuyển đổi” là gì?

Bài tập Mở rộng

1. Có thể đặt một bức ảnh chụp một tác phẩm thuộc phạm vi công cộng theo giấy phép CC không?

2. Liệu một tác giả có thể xuất bản một cuốn tiểu thuyết của mình trong đó có sử dụng toàn bộ bản sonnet của Shakespeare theo giấy phép CC không?

3. Một người dùng phát hành ảnh của chính mình trên trang web của họ theo một giấy phép CC. Liệu cô ấy có thể ngăn chặn việc phát tán hình ảnh này bằng cách viện dẫn quyền cá nhân của mình không?

Tóm tắt

Được thành lập vào năm 2001, tổ chức Tài sản Sáng tạo Công cộng hướng tới mục đích hỗ trợ việc chia sẻ và sử dụng các tác phẩm sáng tạo — tức là các tác phẩm có bản quyền — với sự trợ giúp của các công cụ pháp lý miễn phí. Trọng tâm của họ là bốn mô-đun (Ghi công, Phi thương mại, Không phái sinh và Chia sẻ Tương tự) được kết hợp trong sáu giấy phép được gọi là các giấy phép cốt lõi và có thể được tác giả lựa chọn cho các tác phẩm của họ.

Đáp án Bài tập Hướng dẫn

1. Mô-đun nào của hệ thống giấy phép Tài sản Sáng tạo Công cộng là một thành phần có mặt trong toàn bộ sáu giấy phép Tài sản Sáng tạo Công cộng?

   Mô-đun “Ghi công tác giả’, viết tắt là "`BY.”

2. Mô-đun nào của hệ thống cấp phép Tài sản Sáng tạo Công cộng yêu cầu phân phối một tác phẩm theo cùng các điều kiện?

   Mô-đun “Chia sẻ Tương tự”, viết tắt là “SA.”

3. CC0 có gì khác so với sáu giấy phép CC cốt lõi?

   Với CC0, một tác giả sẽ không được đảm bảo một số quyền nhất định như với các giấy phép CC khác; họ từ bỏ *tất cả* mọi quyền đối với tác phẩm trong phạm vi khả năng của khu vực pháp lý tương ứng. Nguyên tắc "không bảo lưu quyền" này tương ứng với việc chỉ định tác phẩm thuộc *phạm vi công cộng*.

4. Sự khác biệt giữa giấy phép “quốc tế” và “chuyển đổi” là gì?

   Cho đến phiên bản 3.0, Tài sản Sáng tạo Công cộng đã cung cấp các biến thể giấy phép có tính đến các đặc điểm cụ thể của khu vực tài phán quốc gia hoặc khu vực. Kể từ phiên bản 4.0, CC đã loại bỏ cái gọi là phiên bản “chuyển đổi” này và đề xuất phiên bản “quốc tế” hợp lệ trên toàn cầu của giấy phép tương ứng.

Đáp án Bài tập Mở rộng

1. Có thể đặt một bức ảnh chụp một tác phẩm thuộc phạm vi công cộng theo giấy phép CC không?

   Điều này phụ thuộc vào việc bức ảnh đó có đáp ứng các tiêu chí để bản thân nó được bảo vệ bản quyền hay không. Nói cách khác, bản thân bức ảnh phải được công nhận là một sản phẩm sáng tạo và đáng được bảo vệ (ví dụ như thông qua phối cảnh, nền, bộ lọc, v.v.). Nếu đáp ứng được, nó có thể được đặt theo giấy phép CC và sau đó các điều kiện của giấy phép CC sẽ có thể được áp dụng. Tuy nhiên, phần nguyên bản — tức là mô típ thực tế — vẫn thuộc phạm vi công cộng.

2. Liệu một tác giả có thể xuất bản một cuốn tiểu thuyết của mình trong đó có sử dụng toàn bộ bản sonnet của Shakespeare theo giấy phép CC không?

   Có, nhưng chỉ những phần sáng tạo của chính tác giả trong cuốn tiểu thuyết mới phải tuân theo giấy phép CC do tác giả chọn. Phần Sonnet vốn thuộc phạm vi công cộng vẫn sẽ thuộc phạm vi công cộng.

3. Một người dùng phát hành ảnh của chính mình trên trang web của họ theo một giấy phép CC. Liệu cô ấy có thể ngăn chặn việc phát tán hình ảnh này bằng cách viện dẫn quyền cá nhân của mình không?

   Bằng cách xuất bản một bức ảnh theo một giấy phép CC, chủ sở hữu thường đồng ý với việc phân phối nó. Giới hạn sẽ là khi việc sử dụng bức ảnh vi phạm quyền cá nhân của chủ sở hữu (chẳng hạn như khi hình ảnh bị bóp méo nghiêm trọng hoặc được sử dụng cho mục đích quảng cáo hoặc trong bối cảnh chính trị mà không có sự đồng ý của chủ sở hữu). Quyền cá nhân của chủ sở hữu không bị ảnh hưởng bởi quy định về quyền sử dụng của giấy phép CC; do đó, trong những trường hợp như thế này, cô ấy có thể khởi kiện việc sử dụng hình ảnh của mình.

053.3 Các Giấy phép dành cho Nội dung Mở khác

Tham khảo các mục tiêu LPI

Open Source Essentials version 1.0, Exam 050, Objective 053.3

Khối lượng

1

Các lĩnh vực kiến thức chính

- Hiểu rõ về việc cấp phép cho tài liệu
- Hiểu rõ về việc cấp phép cho các tập dữ liệu và cơ sở dữ liệu
- Hiểu rõ về các quyền được cấp bởi các giấy phép dành cho nội dung mở
- Hiểu rõ về các nghĩa vụ do các giấy phép dành cho nội dung mở áp đặt

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Giấy phép Tài liệu Tự do GNU, phiên bản 1.3 (GFDL)
- Giấy phép Cơ sở dữ liệu mở Open Data Commons (ODbL)
- Thỏa thuận Cấp phép Dữ liệu Cộng đồng — Linh hoạt, phiên bản 1.0 (CDLA)
- Thỏa thuận Cấp phép Dữ liệu Cộng đồng — Chia sẻ, phiên bản 1.0 (CDLA)
- Truy cập mở

Bài 1

Chứng chỉ:	Open Source Essentials
Phiên bản:	1.0
Chủ đề:	053 Giấy phép dành cho Nội dung Mở
Mục tiêu:	053.3 Các Giấy phép dành cho Nội dung Mở khác
Bài học:	1 trên 1

Giới thiệu

Các bài học trước đã giới thiệu về khái niệm nội dung mở và tập trung vào các giấy phép Tài sản Sáng tạo Công cộng. Tuy nhiên, ngoài các giấy phép Tài sản Sáng tạo Công cộng, chúng ta còn có các giấy phép dành cho nội dung mở khác, một số trong đó có liên quan đến các dạng nội dung cụ thể và cũng thường xuyên được sử dụng cho các "sản phẩm phụ" (chẳng hạn như tài liệu) của các dự án mã nguồn mở.

Giấy phép dành cho Tài liệu (của Phần mềm)

Ngoài bản thân mã, các kho lưu trữ phần mềm lớn thường có chứa cả các tài liệu hoặc hướng dẫn sử dụng phần mềm. Các tài liệu này thường chứa nhiều yếu tố có bản quyền khác nhau như văn bản giải thích, đồ họa minh họa hoặc ví dụ về mã. Các giấy phép mã nguồn mở áp dụng cho nội dung của một kho lưu trữ thường cũng được áp dụng cho các văn bản đó (nếu không có thông tin nào khác đi kèm). Ví dụ: GPLv3.0 áp dụng cho "bất kỳ một tác phẩm có bản quyền nào" và không chỉ giới hạn ở mã.

Tuy nhiên, các tài liệu thường sẽ phải tuân theo các điều kiện cấp phép khác nhau. Có nhiều lý do

cho việc này vì tài liệu được tạo ra và sử dụng theo các cách khác so với mã. Ví dụ: các điều kiện cấp phép nghiêm ngặt của một giấy phép copyleft mạnh dành cho các chương trình máy tính có thể sẽ ngăn cản việc áp dụng các phần riêng lẻ của tài liệu cho các chương trình khác. Ý nghĩa của việc “mã nguồn” được cung cấp cho tài liệu được cấp phép GPLv3.0 cũng chưa được rõ ràng. Ngoài ra, việc phân phối các phiên bản in của tài liệu hoặc sách hướng dẫn cho các chương trình vốn là thông lệ đã có từ lâu. Trong trường hợp này, sẽ hiệu quả hơn nếu chúng ta tính đến các giấy phép cụ thể dành cho tài liệu (ví dụ như khi một số nghĩa vụ cấp phép có quan hệ mật thiết với số lượng bản sao được phân phối nhất định).

Các giấy phép Tài sản Sáng tạo Công cộng thường được sử dụng cho tài liệu; tuy nhiên, chúng ta cũng có các giấy phép copyleft đặc biệt dành cho tài liệu, chẳng hạn như *Giấy phép Tài liệu Miễn phí* (FDL) đã giải thích mục đích của chính nó ngay trong lời mở đầu:

> Mục đích của Giấy phép này là tạo ra một cuốn sổ tay, sách giáo khoa hoặc các tài liệu hữu ích và có chức năng khác theo nghĩa “tự do”: đảm bảo cho mọi người quyền tự do hiệu quả để sao chép và tái phân phối nó, có hoặc không có sửa đổi, cả với mục đích thương mại lẫn phi thương mại. Thứ hai, Giấy phép này giữ cho tác giả và nhà xuất bản một phương thức để được ghi nhận công sức đối với tác phẩm của họ trong khi không phải chịu trách nhiệm về những sửa đổi do người khác thực hiện.
>
> Giấy phép này là một loại giấy phép "copyleft", có nghĩa là bản thân các sản phẩm phái sinh của tài liệu đó cũng phải tự do theo ý nghĩa tương tự. Nó bổ sung cho Giấy phép Công cộng Chung GNU — một giấy phép copyleft được thiết kế dành cho phần mềm tự do.
>
> Chúng tôi đã thiết kế Giấy phép này để sử dụng nó cho các sản phẩm sách hướng dẫn sử dụng cho phần mềm tự do vì phần mềm tự do cũng cần phải đi kèm với tài liệu tự do: một chương trình tự do phải đi kèm với các sách hướng dẫn cung cấp các quyền tự do giống với phần mềm. Nhưng Giấy phép này không giới hạn ở các sản phẩm hướng dẫn sử dụng phần mềm; nó có thể được sử dụng cho bất kỳ sản phẩm văn bản nào, bất kể chủ đề của nó là gì hay liệu nó có được xuất bản dưới dạng sách in hay không. Chúng tôi đề xuất Giấy phép này sử dụng chủ yếu cho các tác phẩm có mục đích hướng dẫn hoặc tham khảo.
>
> — Free Documentation License, Preamble

Giấy phép này rất sát sao đối với các sản phẩm tài liệu (ví dụ ngay ở việc nó cũng đề cập đến cản các phiên bản in). Đặc biệt, một giấy phép tài liệu copyleft sẽ không ảnh hưởng một cách rõ rệt đến mã mà nó được phân phối cùng. Mặc dù cũng có thể phân phối tài liệu theo GPLv3.0 cùng với mã được MIT cấp phép: vì trong trường hợp này có hai sản phẩm độc lập nên giấy phép GPLv3.0 của tài liệu sẽ không dẫn đến việc mã phải được cấp phép theo GPLv3.0. Tuy nhiên, một giấy phép dành riêng cho tài liệu sẽ khiến phương diện này trở nên rõ ràng hơn.

Giấy phép dành cho Cơ sở Dữ liệu

Ngoài ra, chúng ta còn có các giấy phép đặc biệt dành cho các cơ sở dữ liệu. Bên cạnh nhiều vấn đề khác, các giấy phép này còn tính đến việc sử dụng một cơ sở dữ liệu làm tài nguyên cho một chương trình và các tính năng đặc biệt về bảo vệ bản quyền cho cơ sở dữ liệu.

Thuật ngữ "Cơ sở dữ liệu" được áp dụng như thế nào?

Trong luật bản quyền, một số hệ thống pháp luật (kể cả ở Châu Âu) có sự phân biệt giữa các sản phẩm cơ sở dữ liệu — giống như các tác phẩm được bảo vệ bản quyền khác (hình ảnh, văn bản, v.v.) — được hưởng sự bảo vệ trên cơ sở *sáng tạo trí tuệ cá nhân* (chọn lọc và sắp xếp các thành phần) và cơ sở dữ liệu mà việc sản xuất (mua sắm, xác minh, trình bày) cần tới một khoản *đầu tư đáng kể* (quyền cơ sở dữ liệu).

Trong cả hai trường hợp, đối tượng bảo hộ là một tập hợp các yếu tố độc lập (chẳng hạn như các tác phẩm hoặc hạng mục dữ liệu riêng biệt) được sắp xếp một cách có hệ thống hoặc có phương pháp và có thể được truy cập điện tử riêng lẻ hoặc bằng các phương tiện khác. Khả năng bản quyền của bản thân các phần tử không quá quan trọng nên chúng cũng có thể chỉ là các giá trị số học.

Do đó, một bộ sưu tập dữ liệu có thể được bảo vệ bản quyền dưới dạng sản phẩm cơ sở dữ liệu hoặc theo quyền cơ sở dữ liệu (ở châu Âu). Để một sản phẩm cơ sở dữ liệu được bảo vệ, việc sắp xếp, lựa chọn các thành phần phải thể hiện sự sáng tạo trí tuệ của cá nhân. Điều này khó có thể xảy ra đối với các bộ sưu tập dữ liệu được sử dụng làm cơ sở cho phần mềm - nhưng chúng lại có thể được bảo vệ theo quyền cơ sở dữ liệu. Để có được quyền này, tác giả cần phải đầu tư đáng kể vào việc thu thập và sắp xếp các thành phần. Khoản đầu tư này cũng có thể bao gồm cả thời gian và nguồn nhân lực.

Hiện có nhiều giấy phép khác nhau dành cho cơ sở dữ liệu và bộ sưu tập dữ liệu bao quát việc sử dụng dữ liệu tốt hơn các giấy phép phần mềm mã nguồn mở "cổ điển". Chúng ta hãy cùng xem sự khác biệt thông qua một ví dụ.

Là một phần của dự án nghiên cứu học thuật, tất cả các bài thơ của một tác giả sẽ được sưu tầm và phát hành trên một trang web nơi chúng được sắp xếp theo thứ tự bảng chữ cái theo tiêu đề và có thể được truy cập riêng lẻ. Tuy nhiên, dự án yêu cầu rằng bản quyền liên quan đến việc phân phối và sao chép các bài thơ phải được lấy từ tác giả. Phí cấp phép sẽ được trả cho việc này.

Phí cấp phép có thể cấu thành một "khoản đầu tư đáng kể." Vì các bài thơ có thể được truy xuất riêng lẻ và được sắp xếp một cách có hệ thống (theo bảng chữ cái) nên cơ sở dữ liệu phục vụ các bài thơ có thể được bảo vệ theo quyền cơ sở dữ liệu. Tuy nhiên, các thành phần (các bài thơ) không được lựa chọn một cách cụ thể (việc lựa chọn được thực hiện trên cơ sở hoàn thiện) và sự

sắp xếp không mang tính sáng tạo mà hoàn toàn mang tính hệ thống. Do đó, không có sự sáng tạo trí tuệ cá nhân liên quan đến cơ sở dữ liệu; vậy nên cơ sở dữ liệu này không đủ tiêu chuẩn để được coi là một sản phẩm cơ sở dữ liệu.

Mặt khác, nếu các bài thơ được sắp xếp và lựa chọn theo những tiêu chí nhất định - chẳng hạn như ghi lại những mô típ nhất định trong tác phẩm của tác giả trong giai đoạn sáng tạo - thì tức là có thể tồn tại một sản phẩm sáng tạo trí tuệ cá nhân, và từ đó tuyển tập thơ này sẽ được bảo hộ như một sản phẩm cơ sở dữ liệu.

Tuy nhiên, hai thể loại này được xử lý khác nhau theo luật bản quyền, đặc biệt là về thời hạn bảo vệ: cơ sở dữ liệu theo quyền cơ sở dữ liệu chỉ được bảo vệ 15 năm sau khi sản xuất/phát hành, trái ngược với thời hạn 70 năm sau khi tác giả qua đời đối với các sản phẩm cơ sở dữ liệu.

Phân định Dữ liệu

Dữ liệu trong cơ sở dữ liệu (trong ví dụ của chúng ta là các bài thơ) phải luôn được xem xét một cách tách biệt khỏi tổng thể cơ sở dữ liệu vì chúng có thể (nhưng không nhất thiết) đủ điều kiện để được bảo vệ theo quyền riêng của chúng và do đó có thể sẽ phải tuân theo các điều kiện cấp phép khác nhau. Ví dụ: tuyển tập thơ có thể sẽ phải tuân theo Giấy phép Cơ sở dữ liệu Mở (xem ở bên dưới) trong khi các bài thơ riêng lẻ lại phải tuân theo giấy phép Tài sản Sáng tạo Công cộng hoặc một giấy phép độc quyền.

Chính sách Phân định dành cho Hệ thống Quản lý Cơ sở Dữ liệu (DBMS)

Ngoài ra, phần mềm quản lý dữ liệu (*hệ thống quản lý cơ sở dữ liệu* hay DBMS) không được coi là một phần của cơ sở dữ liệu theo luật bản quyền; nó được coi là một phương tiện truy cập dữ liệu phải được xem xét riêng theo luật bản quyền. Các ứng dụng như vậy — chẳng hạn như MySQL, PostgreSQL, MariaDB, v.v. — phải được xem xét một cách riêng biệt và được bảo vệ như các chương trình máy tính.

Bộ dữ liệu Đào tạo Mô hình Học Máy

Chúng ta có thể tải xuống nhiều bộ sưu tập dữ liệu khác nhau từ các trang web như hugface.co hoặc kaggle.com để đào tạo các mô hình học máy. Các bộ sưu tập như vậy (còn được gọi là *bộ dữ liệu* hay datasets) thường tuân theo quyền cơ sở dữ liệu với điều kiện là đã có một "khoản đầu tư đáng kể" được đưa vào trong quá trình sản xuất của chúng (thường không dễ thấy được từ bên ngoài).

Thuật ngữ "bộ dữ liệu" có phần hơi mơ hồ vì nó có thể chỉ ám chỉ tới một dòng trong cơ sở dữ liệu — tức là một mục nhập hoặc một phần tử có thể xác định — hoặc ám chỉ tới các tập hợp dữ liệu hoặc các bộ dữ liệu.

Giấy phép Cơ sở Dữ liệu Mở (ODbL)

Những sự khác biệt này cũng có thể thấy rõ trong *Giấy phép Cơ sở Dữ liệu Mở* (ODbL) được sử dụng rộng rãi trong lĩnh vực cơ sở dữ liệu. ODbL được phát triển bởi Dữ liệu Chung (Open Data Commons) - một dự án của Tổ chức Tri thức Mở (Open Knowledge Foundation). Ngay trong phần mở đầu, giấy phép này đã phân biệt rõ ràng giữa việc bảo vệ cơ sở dữ liệu và việc bảo vệ nội các dung riêng lẻ của cơ sở dữ liệu. Giấy phép chỉ đề cập đến việc bảo vệ cơ sở dữ liệu, còn phần nội dung cũng có thể được cấp phép hoàn toàn độc lập với cơ sở dữ liệu.

Tất nhiên, chúng ta vẫn có thể chọn cùng một giấy phép cho cơ sở dữ liệu và nội dung (ví dụ như khi chúng ta biên soạn một bộ sưu tập các tác phẩm được cấp phép theo CC-BY-4.0 trong một cơ sở dữ liệu cũng được cấp phép theo CC-BY-4.0).

Giấy phép tốt nhất dành cho một cơ sở dữ liệu phụ thuộc vào mục tiêu của nhà sản xuất cơ sở dữ liệu. Để tiếp tục với ví dụ về tập thơ của chúng ta: nếu để cơ sở dữ liệu thơ được sửa đổi theo ý muốn và cũng có thể được phân phối theo các điều kiện cấp phép khác thì ODbL sẽ không phải là một lựa chọn phù hợp. Điều này là do ODbL có chứa quyền copyleft cho cơ sở dữ liệu — tức là một cơ sở dữ liệu bắt nguồn từ cơ sở dữ liệu được cấp phép ODbL (ví dụ: một cơ sở dữ liệu áp dụng tất cả các thành phần và mẫu sắp đặt các thành phần của cơ sở dữ liệu gốc và thêm các thành phần khác) chỉ có thể được tái phân phối theo cùng các điều kiện. Trong trường hợp này, một giấy phép Tài sản Sáng tạo Công cộng có quyền chỉnh sửa/ biên tập (ví dụ như CC-BY) là một lựa chọn tốt hơn.

ODbL cũng nêu rõ rằng giấy phép này không áp dụng cho các chương trình máy tính quản lý cơ sở dữ liệu (tức là DBMS) theo bất kỳ một cách nào: “Giấy phép này không áp dụng cho các chương trình máy tính được sử dụng để tạo hoặc vận hành Cơ sở Dữ liệu.”

Tuy nhiên, giấy phép quy định rằng một tác phẩm được tạo ra từ nội dung của cơ sở dữ liệu và có sẵn cho tất cả mọi người sử dụng (cái được gọi là *sản phẩm chế tạo*) ít nhất phải chỉ ra được rằng cơ sở dữ liệu nào (không phải DBMS) được sử dụng cho tác phẩm đó và bản thân cơ sở dữ liệu đó có thể được sử dụng theo các điều kiện của ODbL. Một ví dụ về *sản phẩm chế tạo* chính là bản đồ đường phố được tạo ra từ các tọa độ được thu thập trong cơ sở dữ liệu.

Nếu một trong những giấy phép copyleft đã biết (dành riêng cho phần mềm) như GPLv3.0 được sử dụng cho cơ sở dữ liệu, chúng ta có thể hiểu rằng một chương trình máy tính sử dụng cơ sở dữ liệu (ví dụ như cho một hệ thống lưu trữ) chỉ có thể được phân phối theo các điều khoản của GPLv3.0. Mặt khác, giấy phép của DBMS có khả năng độc lập với giấy phép của cơ sở dữ liệu vì DBMS không được coi là một “sản phẩm phái sinh” của cơ sở dữ liệu; nó chỉ cho phép truy cập và chỉnh sửa và vì thế nên nó là một sản phẩm độc lập.

Thỏa thuận Cấp phép Dữ liệu Cộng đồng (CDLA)

Thỏa thuận Cấp phép Dữ liệu Cộng đồng (CDLA) là một dự án của Tổ chức Linux tập trung vào việc cấp phép dữ liệu (khác với phần mềm). Việc thu thập dữ liệu đào tạo cho các hệ thống học máy cũng thuộc phạm vi chủ đề của CDLA.

> Các cộng đồng của chúng tôi muốn phát triển các thỏa thuận cấp phép dữ liệu có thể cho phép chia sẻ dữ liệu tương tự như những gì chúng tôi đã làm với phần mềm mã nguồn mở. Kết quả sẽ chính là sự hợp tác trên quy mô lớn về các giấy phép chia sẻ dữ liệu theo khuôn khổ pháp lý mà chúng tôi gọi là Thỏa thuận Cấp phép Dữ liệu Cộng đồng (CDLA).
>
> Các giấy phép này sẽ thiết lập nên một khuôn khổ cho việc chia sẻ dữ liệu cộng tác mà chúng tôi đã chứng minh là có hiệu quả trong các cộng đồng phần mềm mã nguồn mở.
>
> — CDLA website (cdla.dev)

Do đó, CDLA được dựa trên một khuôn khổ tính đến các cấp độ khác nhau của các vấn đề cấp phép, đặc biệt là khi dữ liệu được biên soạn từ nhiều nguồn bởi nhiều nhà phát triển đóng góp khác nhau. Trong ví dụ của chúng ta, đây có thể là một bộ sưu tập mà trong đó, các tác giả khác nhau đóng góp những bài thơ của riêng họ để xây dựng nên một cơ sở dữ liệu thơ chung.

CDLA phân biệt giữa “cấp phép vào” (inbound) cho dữ liệu được thêm vào bộ sưu tập và “cấp phép ra” (outbound) cho việc sử dụng dữ liệu. Ở một cấp độ khác, nó chỉ định các yêu cầu đối với tổ chức lưu trữ hoặc nắm giữ dữ liệu.

Để tiếp tục với ví dụ về các bài thơ: một tác giả muốn đóng góp bài thơ của mình sẽ được hướng dẫn chi tiết bởi các “giấy phép vào”. “Giấy phép ra” sẽ liên quan đến những điều kiện mà các biên tập viên của toàn bộ tập thơ đã thỏa thuận với nhau.

Tương tự như các giấy phép mã nguồn mở dành cho phần mềm, CDLA cũng có sẵn các danh mục *linh hoạt* và *chia sẻ* mà chúng ta sẽ cùng tìm hiểu một cách ngắn gọn sau đây. Hãy truy cập vào các trang web kaggle.com hoặc kissingface.co đã được đề cập tới ở trên để có thể hiểu một cái nhìn cơ bản về các giấy phép được sử dụng cho cơ sở dữ liệu hoặc tập dữ liệu. Ví dụ: bạn có thể sử dụng bộ lọc để tìm ra các bộ sưu tập dữ liệu được cung cấp theo CDLA.

Thỏa thuận Cấp phép Dữ liệu Cộng đồng — Linh hoạt

Phiên bản 2.0 của CDLA linh hoạt đã có từ năm 2021; nó ngắn gọn và rõ ràng hơn nhiều so với phiên bản 1.0.

Phiên bản 1.0 cấp quyền sử dụng và xuất bản dữ liệu được cấp phép, bao gồm cả việc xuất bản “dữ liệu nâng cao” (có bao gồm các chỉnh sửa hoặc bổ sung vào bộ dữ liệu). Khi chuyển giao hoặc phát hành dữ liệu, người dùng sẽ phải tuân thủ các nghĩa vụ cấp phép cơ bản; bên cạnh các nghĩa

vụ khác, người dùng sẽ phải chuyển giao văn bản cấp phép, đánh dấu các thay đổi và giữ lại thông báo bản quyền. Dữ liệu cũng có thể được chuyển giao theo các điều kiện cấp phép bổ sung hoặc các điều kiện cấp phép khác.

Trong phiên bản 2.0, cụm từ “dữ liệu nâng cao” không còn được sử dụng nữa; thay vào đó, việc cấp quyền đã được diễn đạt một cách chính xác hơn:

> Người nhận dữ liệu có thể sử dụng, sửa đổi và chia sẻ Dữ liệu do Nhà cung cấp dữ liệu cung cấp theo thỏa thuận này nếu Người nhận dữ liệu đó tuân thủ các điều khoản của thỏa thuận này.
>
> — Community Data License Agreement - Permissive, version 2.0

Điều kiện duy nhất để chia sẻ dữ liệu trong phiên bản 2.0 là phải chuyển tiếp văn bản giấy phép hoặc một liên kết đến văn bản giấy phép.

Thỏa thuận Cấp phép Dữ liệu Cộng đồng — Chia sẻ

Phiên bản chia sẻ của CDLA (hiện chỉ có trong phiên bản 1.0) có chứa một quyền copyleft cho dữ liệu — tức là việc chuyển giao (“phát hành”) “dữ liệu nâng cao” chỉ có thể được cho phép trong cùng các điều kiện.

> Dữ liệu (bao gồm cả Dữ liệu nâng cao) phải được phát hành theo Thỏa thuận này và tuân thủ theo Mục 3 này;
>
> — Community Data License Agreement -- Sharing, version 1.0

Truy cập Mở

Một thuật ngữ thường được sử dụng liên quan đến nội dung mở là *truy cập mở* (open access). Mặc dù thuật ngữ này không được định nghĩa pháp lý, tuyên bố của Tổ chức Sáng kiến Truy cập Mở Budapest (BOAI) đã nêu rõ ý định đằng sau truy cập mở. Sáng kiến này xuất phát từ một hội nghị ở Budapest năm 2001 và nó đã tóm tắt những nỗ lực quốc tế dành cho truy cập mở:

> Tài liệu cần được truy cập miễn phí trực tuyến là tài liệu mà các học giả cung cấp cho thế giới và không mong đợi nhận được lợi nhuận. Danh mục này ban đầu bao gồm các bài báo trên tạp chí được bình duyệt của họ, nhưng nó cũng bao gồm bất kỳ một bản in trước chưa được bình duyệt nào mà họ có thể muốn đưa lên mạng để nhận được nhận xét hoặc thông báo cho các đồng nghiệp về những phát hiện quan trọng trong nghiên cứu. Có nhiều mức độ và loại quyền truy cập rộng rãi và dễ dàng hơn đối với loại tài liệu này. Khi nói đến “truy cập mở” đối với loại tài liệu này, chúng tôi đang nói đến tính khả dụng miễn phí của nó trên mạng internet công cộng cho phép mọi người dùng bất kỳ đọc, tải xuống, sao chép, phân phối, in,

> tìm kiếm hoặc liên kết đến toàn văn của các bài viết này, thu thập chúng để lập chỉ mục, chuyển chúng dưới dạng dữ liệu cho phần mềm hoặc sử dụng chúng cho bất kỳ mục đích hợp pháp nào khác mà không có rào cản về tài chính, pháp lý hoặc kỹ thuật nào khác ngoài những rào cản không thể tách rời khỏi việc truy cập vào chính bản thân internet. Hạn chế duy nhất đối với việc sao chép và phân phối cũng như vai trò duy nhất của bản quyền trong lĩnh vực này là trao cho tác giả quyền kiểm soát tính toàn vẹn của tác phẩm của họ và quyền được ghi nhận và trích dẫn đúng cách.
>
> — Declaration of the Budapest Open Access Initiative

Do vậy, truy cập mở đề cập cụ thể đến quyền truy cập miễn phí vào tài liệu khoa học và các nội dung khác trên internet. Phong trào truy cập mở bắt nguồn từ những năm 1990 với mục đích cung cấp các ấn phẩm khoa học cho công chúng. Do đó, truy cập mở không đề cập đến một giấy phép cụ thể hoặc một số điều kiện cấp phép nhất định; nó là một khái niệm nằm trong việc cấp phép.

Khi một bài viết được xuất bản với quyền truy cập mở, bước đầu tiên là cần chọn một giấy phép nội dung mở (chẳng hạn như một trong các giấy phép Tài sản Sáng tạo Công cộng). Tuy nhiên, việc lựa chọn giấy phép CC không phải là một việc bắt buộc đối với quyền truy cập mở. Trải qua nhiều năm, một danh mục các chiến lược truy cập mở khác nhau đã được phát triển, trong đó cũng tính đến việc cấp phép kép — ví dụ: cấp giấy phép toàn diện cho nhà xuất bản và tác giả cũng đồng thời cung cấp nội dung có thể tải xuống trên trang web của riêng họ.

Cả BOAI và *Tuyên bố Berlin* ra đời hai năm sau đó đều là những cột mốc quan trọng trong phong trào truy cập mở. Phần mở đầu của Tuyên bố Berlin đã nêu rõ:

> Theo tinh thần của Tuyên bố Sáng kiến Truy cập Mở Budapest, Hiến chương ECHO và Tuyên bố Bethesda về Xuất bản Truy cập Mở, chúng tôi đã soạn thảo Tuyên bố Berlin nhằm quảng bá Internet như một công cụ chức năng dành cho cơ sở tri thức khoa học toàn cầu và phản ánh hình ảnh của con người; bên cạnh đó, nó cũng nhằm chỉ ra rõ các biện pháp mà các nhà hoạch định chính sách nghiên cứu, các viện nghiên cứu, các cơ quan tài trợ, thư viện, lưu trữ và bảo tàng cần phải xem xét.
>
> — Berlin Decalaration 2003

Bài tập Hướng dẫn

1. Các giấy phép mã nguồn mở “cổ điển” có thể được sử dụng cho tài liệu và cơ sở dữ liệu không?

Có	
Không	
Tùy thuộc vào từng trường hợp, không có câu trả lời rõ ràng	

2. Việc sử dụng các giấy phép riêng biệt đối với tài liệu liệu có hợp lý không? Tại sao?

3. Việc sử dụng các giấy phép riêng biệt đối với có sở dữ liệu liệu có hợp lý không? Tại sao?

4. Giấy phép nào sau đây phù hợp cho tài liệu?

CC-BY-4.0	
FDL	
ODbL	
GPL-3.0	
BSD 3 Điều khoản	

5. Thuật ngữ “truy cập mở” mang ý nghĩa gì?

Bài tập Mở rộng

1. Có giấy phép cụ thể nào dành cho phông chữ không?

2. Hãy giải thích ngắn gọn hai chiến lược hoặc mô hình truy cập mở.

3. “Định nghĩa mở” có nghĩa là gì?

Tóm tắt

Ngoài các giấy phép Tài sản Sáng tạo Công cộng phổ biến, chúng ta còn có các loại hình cấp phép nội dung khác được thiết kế riêng biệt hơn cho từng loại dữ liệu tương ứng: đối với tài liệu phần mềm hoặc cơ sở dữ liệu, chúng ta có các giấy phép FDL hoặc ODbL có tính đến những đặc điểm của các loại sản phẩm này. Bài học này đã cung cấp một cái nhìn tổng quan về một số giấy phép này cũng như về khái niệm truy cập mở.

Đáp án Bài tập Hướng dẫn

1. Các giấy phép mã nguồn mở “cổ điển” có thể được sử dụng cho tài liệu và cơ sở dữ liệu không?

Có	X
Không	
Tùy thuộc vào từng trường hợp, không có câu trả lời rõ ràng	

2. Việc sử dụng các giấy phép riêng biệt đối với tài liệu liệu có hợp lý không? Tại sao?

   Có, vì chúng thường chứa cả các nội dung khác ngoài chương trình. Việc quy định cách xử lý bản in của tài liệu chính là một ví dụ.

3. Việc sử dụng các giấy phép riêng biệt đối với có sở dữ liệu liệu có hợp lý không? Tại sao?

   Có, vì nó có thể sẽ rất hữu ích. Cụ thể là một giấy phép cơ sở dữ liệu riêng biệt có thể đạt được hiệu ứng copyleft cho cơ sở dữ liệu mà không để copyleft ảnh hưởng đến phần mã xung quanh. Ngoài ra, giấy phép cơ sở dữ liệu sẽ giải quyết các điều khoản cụ thể mà luật bản quyền cung cấp để bảo vệ cơ sở dữ liệu.

4. Giấy phép nào sau đây phù hợp cho tài liệu?

CC-BY-4.0	X
FDL	
ODbL	
GPL-3.0	
BSD 3 Điều khoản	

5. Thuật ngữ “truy cập mở” mang ý nghĩa gì?

   Thuật ngữ này mô tả quyền truy cập miễn phí vào tài liệu khoa học và các nội dung khác trên Internet.

Đáp án Bài tập Mở rộng

1. Có giấy phép cụ thể nào dành cho phông chữ không?

   Có, ví dụ như Giấy phép Phông chữ Mở SIL (OFL). Giấy phép này có chứa quyền copyleft dành riêng cho phông chữ, nhưng quyền này sẽ không có hiệu lực nếu phông chữ không bị thay đổi khi sử dụng.

2. Hãy giải thích ngắn gọn hai chiến lược hoặc mô hình truy cập mở.

   Trong chiến lược truy cập mở "Vàng", ấn phẩm sẽ được nhà xuất bản cung cấp trực tiếp theo giấy phép nội dung mở, thường (nhưng không luôn luôn) là sử dụng giấy phép Tài sản Sáng tạo Công cộng.

   Mô hình truy cập mở "Xanh" được áp dụng khi nhà xuất bản không phân phối ấn phẩm theo quyền truy cập mở, nhưng tác giả lại được trao cơ hội cung cấp ấn phẩm để tải xuống (chẳng hạn như trên trang web của họ) theo một giấy phép mở.

3. "Định nghĩa mở" có nghĩa là gì?

   "Định nghĩa mở" được xây dựng bởi Tổ chức Tri thức Mở là một cách hiểu chung về thuật ngữ "mở" trong "dữ liệu mở", "kiến thức mở" và "nội dung mở". Định nghĩa này mô tả cụ thể các quyền tự do cơ bản phải được cấp nếu một giấy phép được coi là "mở" theo đúng định nghĩa. Những quyền này bao gồm quyền truy cập, sử dụng, chỉnh sửa hoặc sửa đổi và chia sẻ.

Linux Professional Institute

Chủ đề 054: Mô hình kinh doanh Mã nguồn Mở

054.1 Mô hình kinh doanh Phát triển Phần mềm

Tham khảo các mục tiêu LPI

Open Source Essentials version 1.0, Exam 050, Objective 054.1

Khối lượng

2

Các lĩnh vực kiến thức chính

- Hiểu rõ về các mô hình kinh doanh và luồng doanh thu chung cho các tổ chức cung cấp dịch vụ liên quan đến phần mềm mã nguồn mở cũng như nội dung mở
- Hiểu rõ tác động của giấy phép đối với mô hình kinh doanh của nhà cung cấp dịch vụ
- Hiểu rõ các mục tiêu cấp độ dịch vụ và thỏa thuận cấp độ dịch vụ
- Hiểu rõ về nhu cầu bảo vệ an ninh và quyền riêng tư
- Nắm được cấu trúc chi phí và đầu tư cần thiết cho các mô hình kinh doanh dịch vụ phần mềm mã nguồn mở

Partial list of the used files, terms and utilities

- Phát triển có trả phí
- Phần mở rộng và phần bổ sung có trả phí
- Freemium
- Phiên bản doanh nghiệp và phiên bản cộng đồng
- Phân phối tự lưu trữ
- Đăng ký theo dõi
- Hỗ trợ khách hàng

Linux Professional Institute

Bài 1

Chứng chỉ:	Open Source Essentials
Phiên bản:	1.0
Chủ đề:	054 Mô hình kinh doanh Mã nguồn Mở
Mục tiêu:	054.1 Mô hình kinh doanh Phát triển Phần mềm
Bài học:	1 trên 1

Giới thiệu

Theo truyền thống, các công ty phân phối phần mềm độc quyền (hay còn được gọi là phần mềm *mã nguồn đóng*) có mô hình kinh doanh tương đối đơn giản: họ bán giấy phép sử dụng phần mềm dưới dạng trar phí một lần hoặc liên tục theo mô hình *đăng ký*.

Tuy nhiên khi trải qua nhiều thập kỷ, thị trường phần mềm đã thay đổi hoàn toàn và khách hàng ngày nay chỉ muốn trả một vài đô la cho một ứng dụng để nhận được các bản cập nhật miễn phí trọn đời.

Kết quả là cách tiếp cận cũ của phần mềm độc quyền không còn mang lại lợi nhuận nữa, và ngay cả các công ty từng xây dựng toàn bộ mô hình kinh doanh của mình dựa trên việc bán giấy phép phần mềm cũng đã đa dạng hóa mô hình kinh doanh của họ bằng cách tích hợp nhà cung cấp, hỗ trợ, dịch vụ, phần mềm dưới dạng dịch vụ (SaaS), dịch vụ lưu trữ đám mây/ảo hoá và các sản phẩm phần cứng chạy phần mềm (điện thoại, máy tính xách tay, máy chủ, máy in, ô tô, đồng hồ thông minh, v.v.).

Ngược lại, giấy phép phần mềm mã nguồn mở và tự do trao cho tất cả mọi người quyền sử dụng, nghiên cứu, sửa đổi và tái phân phối phần mềm mà không phải trả phí. Vì vậy, các doanh nghiệp

dựa trên mã nguồn mở chưa bao giờ có lựa chọn kinh doanh trên giấy phép sử dụng phần mềm. Người dùng không thể mong chờ việc lấy tác phẩm của người khác từ một dự án mã nguồn mở và kiếm được lợi nhuận khổng lồ bằng cách cung cấp sản phẩm mang phạm vi công cộng đó cho khách hàng của mình; lý do là bởi khách hàng cũng có thể nhận được phần mềm tương tự mà không phải trả phí trực tiếp từ dự án. Thay vào đó, các công ty cung cấp phần mềm mã nguồn mở và tự do sử dụng các mô hình kinh doanh thay thế. Các mô hình kinh doanh phần mềm thành công nhất thường kết hợp nhiều dạng giá trị khách hàng.

Trong bài học này, chúng ta sẽ cùng xem xét lý do tại sao một công ty lại chọn một số mô hình kinh doanh phần mềm này cho các phần mềm mã nguồn mở cũng như những gì họ phải đánh đổi cho những lựa chọn này.

Mục tiêu và lý do phát hành Phần mềm hoặc Nội dung tuân theo Giấy phép mở

Có nhiều lý do khiến các nhà phát triển chọn xây dựng doanh nghiệp dựa trên mã nguồn mở: phần mềm có thể đáp ứng một nhu cầu, giải quyết một vấn đề hoặc cung cấp một số tính năng mà nhà phát triển cung cấp cho khách hàng. Một công ty có thể tiết kiệm rất nhiều thời gian và tiền bạc bằng cách chia sẻ công việc trên phần mềm với các nhà phát triển khác thay vì viết lại từ đầu cùng một phần mềm. Việc tham gia vào dự án là một khoản đầu tư được kỳ vọng sẽ mang lại kết quả là các phần mềm có thể sử dụng được, ổn định, đáng tin cậy, an toàn và được bảo trì tốt.

Nhiều nhà phát triển mã nguồn mở có kỳ vọng sẽ nhận được các bản sửa lỗi từ khách hàng của họ. Một số khách hàng thậm chí còn đưa những đóng góp của họ lên một tầm cao hơn bằng cách thêm các tính năng mới, chuyển phần mềm sang một môi trường mới hoặc thực hiện những cải tiến quan trọng khác. Cả việc sửa lỗi và cải tiến ở cấp độ cao hơn đều sẽ nâng cao phần mềm gốc. Nếu khách hàng đóng góp những thay đổi đó lại cho các nhà phát triển ban đầu thì những thay đổi này có thể khiến mã trở nên càng thu hút các khách hàng tiềm năng khác hơn, ngoài ra còn khuyến khích việc ứng dụng mã.

Nhưng một công ty không thể mong chờ nhận được sự đóng góp và chú ý chỉ bằng cách đưa mã của mình lên các trang như GitHub hoặc GitLab. Khuyến khích các nhà phát triển tham gia dự án là một chuyện, nhưng giữ cho họ nhiệt tình với dự án lại là chuyện khác — vì vậy, đừng đánh giá thấp khối lượng công việc để có được sự đóng góp từ cộng đồng. Một công ty cần nỗ lực tuyển dụng các nhà phát triển, cố vấn cho họ, động viên họ và kiểm tra lỗi trong mã của họ.

Một lý do khác để mở mã là tạo ra một tiêu chuẩn ngành. Việc chia sẻ mã mở có thể mang lại khả năng tương tác tốt hơn cũng như các lợi ích khác. Nhóm phát triển mã có kiến thức chuyên môn cao có thể cho phép họ định hướng tương lai của dự án và được trả tiền để hỗ trợ những người khác sử dụng mã.

Một số công ty mở mã của họ để nhằm tạo niềm tin cho khách hàng. Trước hết, khách hàng biết rằng họ có thể tiếp tục sử dụng và cải tiến mã nếu công ty thất bại hoặc quyết định thâm nhập một thị trường khác và từ bỏ mã. Thứ hai, khách hàng có thể tự mình kiểm tra chất lượng và tính bảo mật của mã hoặc nhờ một chuyên gia độc lập thực hiện rà soát; điều này thường không thể thực hiện được đối với phần mềm độc quyền đóng.

Cuối cùng, một công ty có thể đã áp dụng một cơ sở mã đã là mã nguồn mở sẵn và lên kế hoạch xây dựng hoạt động kinh doanh thương mại dựa trên đó. Giấy phép có thể sẽ yêu cầu công ty phân phối mã nguồn của các thay đổi cho bất kỳ người dùng cuối nào của chương trình.

Tóm lại, một số lý do nên để mở mã của một công ty là:

- Xây dựng dựa trên một dự án phần mềm tự do sẵn có
- Hưởng lợi từ sự đổi mới và đóng góp của khách hàng
- Tạo niềm tin nơi khách hàng
- Quảng bá mã theo một tiêu chuẩn ngành

Các Mô hình Kinh doanh phổ biến và Dòng doanh thu

Trong nhiều thập kỷ qua, ngành công nghiệp phần mềm đã phát hiện ra nhiều mô hình kinh doanh thích hợp cho các doanh nghiệp độc quyền cũng như mã nguồn mở. Ở phần này, chúng ta sẽ tập trung vào những mô hình phổ biến nhất được sử dụng hiện nay trong số các nhà phát triển mã nguồn mở.

Một mô hình kinh doanh phổ biến trong cả phần mềm độc quyền và mã nguồn mở là tính phí hỗ trợ. Khách hàng có thể sẽ gặp khó khăn khi cài đặt và định cấu hình phần mềm, sửa lỗi, cần thêm các tính năng mới để sử dụng riêng và quản lý hệ thống của họ. Nhân viên phát triển phần mềm là một nguồn lực hỗ trợ tuyệt vời. Trên thực tế, nhân viên này có thể đã cung cấp nhiều hỗ trợ miễn phí trên các diễn đàn thảo luận về phần mềm.

Do đó, trong mô hình hỗ trợ, khách hàng sẽ trả tiền cho nhà cung cấp để cài đặt phần mềm mã nguồn mở trên phần cứng của khách hàng (được gọi là phân phối *tự lưu trữ*) hoặc có quyền truy cập vào phần mềm trên phần cứng của nhà cung cấp (mô hình dựa trên đám mây). Hợp đồng hỗ trợ đảm bảo rằng khách hàng sẽ nhận được sự trợ giúp kịp thời từ công ty cho những nhu cầu phức tạp của họ. Các công ty sử dụng mô hình này thường cung cấp dịch vụ trả trước và khách hàng sẽ phải trả tiền định kỳ.

Một mô hình kinh doanh khác dành cho phần mềm độc quyền — thực ra là một tập hợp các mô hình chồng chéo — được gọi là *freemium* - một thuật ngữ được phổ biến rộng rãi vào năm 2009 bởi tác giả kiêm biên tập viên Chris Anderson; nó dựa trên một mô hình kinh doanh cũ hơn nhiều

được gọi là *dao cạo và lưỡi dao* (razor and blades). Trong mô hình dao cạo và lưỡi dao, khách hàng sẽ trả rất ít tiền cho sản phẩm cơ bản (ví dụ như một chiếc dao cạo râu hoặc máy in) và sau đó trả thêm tiền cho các bộ phận cần thiết được thiết kế dễ hao mòn (lưỡi dao cạo hay hộp mực cho máy in).

Trong mô hình freemium, khách hàng có thể sử dụng miễn phí một sản phẩm ở một mức độ nhất định và được khuyến khích trả tiền cho nhiều tính năng hơn nếu họ thấy sản phẩm đó hữu ích. Chúng ta có thể đã biết đến các trang tin tức trực tuyến cung cấp một số lượng bài báo miễn phí nhất định và yêu cầu độc giả đăng ký theo dõi để có được toàn quyền truy cập vào trang web của họ. Đây là một ví dụ điển hình về mô hình freemium. Một ví dụ khác là một nền tảng trò chơi cung cấp trò chơi cơ bản miễn phí nhưng sẽ tính tiền cho các tính năng phụ cụ thể.

Nhiều công ty phần mềm độc quyền khác lại căn cứ mô hình freemium của họ theo thời gian: dùng thử phần mềm miễn phí trong ba tháng và sau đó trả tiền để tiếp tục sử dụng phần mềm đó.

Các công ty phần mềm độc quyền đôi khi sẽ sử dụng một phiên bản khác của mô hình freemium được gọi là *lõi mở* (open core). Trong mô hình lõi mở, một số tính năng cơ bản nhất định ("lõi" của sản phẩm) sẽ là nguồn mở và các tính năng độc quyền bổ sung sẽ có giấy phép.

Thông thường, phần mở sẽ có đầy đủ chức năng nhưng sẽ phù hợp nhất cho các nhà nghiên cứu hoặc người dùng cá nhân và có thể trở nên khó quản lý khi có nhiều người chia sẻ nó trong toàn doanh nghiệp. Do đó, các tính năng độc quyền có thể bao gồm các giao diện web thuận tiện cho việc quản trị, các công cụ kế toán và cộng tác cũng như những yếu tố đặc biệt quan trọng khác đối với các trang web lớn.

Mặc dù các mô hình kinh doanh lõi mở sử dụng một số phần mềm mã nguồn mở nhưng khách hàng vẫn đủ thông minh để nhận ra rằng thực chất toàn bộ sản phẩm vẫn là độc quyền. Vì vậy, trong khi lõi mở có thể có lợi thế trong việc xây dựng dự án dựa trên mã nguồn mở hiện có, nó lại không cung cấp bất kỳ lợi thế nào khác của phần mềm mã nguồn mở. Mô hình kinh doanh lõi mở không tạo dựng được niềm tin với khách hàng, không truyền cảm hứng cho họ đóng góp vào phần mềm cơ sở, không cung cấp cho khách hàng quyền truy cập để kiểm tra mã, xây dựng mạng lưới các nhà phát triển lành nghề bên ngoài công ty chính hoặc làm việc theo tiêu chuẩn ngành. Tệ hơn nữa, mô hình phát triển lõi mở có cùng chi phí phát triển mà không có những lợi ích để khiến nó trở nên đáng giá. Các công ty thử nghiệm mô hình kinh doanh lõi mở thường thất vọng về kết quả và nhiều công ty sau đó đã chuyển sang mô hình độc quyền thuần túy.

Các công ty cũng có thể xây dựng một dịch vụ web dựa trên cơ sở mã nguồn mở hoặc độc quyền trong mô hình *Phần mềm dưới dạng dịch vụ* (SaaS). Khách hàng sẽ thanh toán theo tháng hoặc theo năm.

Nhiều công ty điều hành dịch vụ web SaaS hoặc cung cấp ứng dụng di động sẽ kiếm tiền thông qua quảng cáo. Một số công ty cũng thu thập dữ liệu về người dùng thông qua dịch vụ hoặc ứng

dụng và bán dữ liệu này cho các bên thứ ba mong muốn sử dụng dữ liệu đó để quảng cáo. Tuy nhiên, quảng cáo có thể dễ gây khó chịu. Việc bán dữ liệu hiện đang gây tranh cãi và một số quốc gia đã đặt ra các hạn chế trong việc thu thập dữ liệu.

Cuối cùng, các công ty có thể phát triển và phát hành phần mềm mã nguồn mở mà không cần cố gắng thu lại bất kỳ một loại doanh thu nào từ nó. Mô hình này dành cho các công ty có doanh thu từ những thứ khác ngoài phần mềm. Ví dụ: các công ty ô tô sẽ hợp tác để tạo ra một lượng lớn các phần mềm miễn phí để chạy trên ô tô của họ (ngày nay đã được vi tính hóa hoàn toàn).

Các công ty phần mềm lớn có được doanh thu từ các dịch vụ độc quyền như Google và Amazon đôi khi sẽ phát hành phần mềm quản trị hoặc các công cụ hữu ích khác dưới dạng mã nguồn mở vì những công cụ này không phải là trọng tâm trong hoạt động kinh doanh cốt lõi của họ. Các tổ chức phát hành mã theo giấy phép mở được hưởng lợi từ phản hồi, báo cáo lỗi và các cải tiến tính năng do cộng đồng mã nguồn mở cung cấp.

Sử dụng Phần mềm Mã nguồn Mở trong các Công nghệ và Dịch vụ khác

Có rất nhiều công ty kết hợp phần mềm mã nguồn mở vào sản phẩm hoặc nền tảng web của họ. Suy cho cùng, phần mềm mã nguồn mở chính là phần mềm miễn phí! Nhưng đó không phải là lý do quan trọng nhất (và có lẽ thậm chí không phải là lý do chính đáng) để ứng dụng chúng. Quan trọng hơn, phần lớn các phần mềm này đều có chất lượng cao (mặc dù các nhóm phát triển nên xem xét cẩn thận trước khi ứng dụng) và thậm chí có thể trở thành một tiêu chuẩn ngành.

Một ưu điểm khác của phần mềm mã nguồn mở là nó vẫn sẽ sẵn có nếu các nhà phát triển hoặc cộng đồng xung quanh nó rời đi.

Tuy nhiên, phần mềm mã nguồn mở và tự do cũng đi kèm với trách nhiệm. Phần này sẽ nhanh chóng liệt kê các vấn đề chính cần được xem xét trước khi ứng dụng chúng.

Vấn đề đầu tiên áp dụng cho bất kỳ phần mềm hoặc công cụ nào của bên thứ ba đang được xem xét áp dụng: liệu nó có đáp ứng được nhu cầu của người dùng hiện tại và khi công ty phát triển hay không? Phần mềm có được hỗ trợ bởi một cộng đồng mạnh mẽ có thể cung cấp hỗ trợ kỹ thuật và tiếp tục phát triển phần mềm không? Phần mềm có lỗ hổng bảo mật nghiêm trọng nào không?

Tiếp theo, các nhà quản lý sẽ phải xem xét trách nhiệm của công ty khi họ kết hợp một dự án mã nguồn mở vào phần mềm của mình. Nếu các nhà phát triển xây dựng mã mới dựa trên một mã nguồn mở, họ nên kiểm tra xem mình được yêu cầu phải làm gì theo như giấy phép của phần mềm mã nguồn mở đó. Một số giấy phép sẽ yêu cầu nhà phát triển phân phối mã nguồn của những thay đổi của chính họ cho người dùng cuối theo cùng giấy phép tự do giống như cơ sở mã gốc.

Ngay cả khi một nhà phát triển không bắt buộc phải phân phối mã nguồn đã qua sửa đổi của họ và đang tạo ra một sản phẩm độc quyền dựa trên mã nguồn mở, nhà phát triển đó có thể sẽ muốn đóng góp lại một số thứ nhất định, chẳng hạn như các bản sửa lỗi hay cải tiến mã. Bằng cách đóng góp những bản sửa lỗi và cải tiến này, nhà phát triển đóng góp đã cho phép dự án kết hợp các đóng góp (nếu các nhà phát triển lõi lựa chọn như vậy) và duy trì chúng. Các nhà phát triển không có đóng góp có thể sẽ phải áp dụng lại các bản sửa lỗi và cải tiến mỗi khi họ nâng cấp lên một phiên bản mới của mã gốc.

Vì sự phụ thuộc này cũng như một số các lý do khác, các nhân viên công ty nên cân nhắc việc trở thành thành viên tích cực của cộng đồng phát triển mã. Các nhà phát triển có thể tìm hiểu thêm về mã bằng cách tham gia và từ đó có thể giúp đặt ra định hướng phát triển trong tương lai. Tất nhiên, tổ chức sẽ phải trả tiền cho thời gian mà các nhà phát triển dành cho các hoạt động cộng đồng. Đối với một công ty nghiêm túc trong việc sử dụng phần mềm tự do (free), việc này lại không hề miễn phí (free).

Những cân nhắc về Phần mềm Mã nguồn Mở từ góc độ của Khách hàng

Việc mở mã rất có lợi cho khách hàng vì nhiều lý do, nhưng nó cũng ám chỉ một mối quan hệ khác giữa công ty và khách hàng. Khách hàng nên hiểu rõ lợi ích cũng như ý nghĩa của mối quan hệ này.

Lợi ích chính đối với khách hàng (đã được đề cập tới trước đó trong bài học này) là họ có thể tin tưởng hơn vào phần mềm. Họ biết rằng nó sẽ không biến mất. Nhiều công ty độc quyền khi đóng cửa hoặc đột ngột chuyển sang một hướng đi mới sẽ buộc khách hàng của họ gấp rút chuyển sang một sản phẩm khác mà thậm chí có thể không thật sự tương thích với nhu cầu của khách hàng. Trái với đó, phần mềm mã nguồn mở không phụ thuộc vào bất kỳ một tổ chức nào. Nếu đó là một dự án quan trọng, những thành viên khác trong cộng đồng sẽ tiếp tục dự án khi các nhà phát triển ban đầu rời đi.

Khách hàng cũng có thể kiểm tra chất lượng và tính bảo mật của phần mềm mã nguồn mở. Họ có thể tự xác định xem việc thêm các tính năng của riêng mình hoặc chuyển sang một môi trường mới có dễ dàng hay không.

Vì mã nguồn trong một dự án mã nguồn mở được mở cho tất cả mọi người nên nhiều nhà phát triển có thể tự mình làm quen với nó. Đối với một dự án sở hữu một cộng đồng đông đảo đứng đằng sau, sẽ có rất nhiều người cung cấp hỗ trợ trên diễn đàn. Thông thường, việc thuê người hỗ trợ phần mềm hoặc quản lý và sử dụng nó trong công ty sẽ tương đối dễ dàng vì các nhân viên mới đều đã biết về mã.

Khách hàng là những người phụ thuộc vào mã để làm việc hàng ngày; họ sẽ rất yên tâm khi biết

rằng mình có thể tự sửa lỗi hoặc thuê ai đó làm việc này. Một lỗi lạ chỉ ảnh hưởng đến một số khách hàng có thể sẽ phải mất rất nhiều năm mới được nhóm phát triển cốt lõi khắc phục; điều này thường xuyên khiến cho người sử dụng phần mềm độc quyền rất khó chịu. Ngoài ra, khi có sẵn mã nguồn, chúng ta có thể xác định được lỗi và đề xuất cách khắc phục sớm hơn.

Còn mối quan hệ giữa công ty và khách hàng thì sao? Công ty có thể chọn thiết lập một mối quan hệ giống hệt với một mối quan hệ được cung cấp bởi một công ty phần mềm độc quyền điển hình: cung cấp cho khách hàng các bản cập nhật thường xuyên và một hợp đồng hỗ trợ. Khách hàng sẽ không bao giờ phải xem tới mã nguồn hoặc tham gia các diễn đàn cộng đồng.

Tuy nhiên, hầu hết khách hàng sẽ được hưởng lợi từ những cơ hội mà chỉ các dự án mã nguồn mở mới có thể mang lại. Họ có thể cho phép các nhà phát triển của riêng họ đóng góp cả thông tin lẫn mã. Những sự đóng góp và tham gia như vậy sẽ giúp nhân viên hiểu sâu hơn, giúp họ tuyển dụng nhân viên mới hiệu quả hơn và mang lại cho họ tiếng nói trong sự phát triển ở tương lai.

Cơ cấu Chi phí và Đầu tư

Nhu cầu về lập trình viên hiện đang tăng cao; vì vậy, bất kỳ một nỗ lực phát triển phần mềm nào cũng sẽ rất tốn kém. Nhu cầu về những người biết rõ về các dự án mã nguồn mở lớn thậm chí còn cao hơn vì những cơ sở mã đó được sử dụng rất rộng rãi.

Một dự án mã nguồn mở có thể mang lại các tiềm năng về mặt tiết kiệm chi phí vì các tổ chức và cá nhân khác nhau có thể cùng kết hợp trong một cơ sở mã chung. Nhưng việc thực hiện một dự án như vậy lại sẽ gây ra những chi phí mới.

Nếu một công ty mở một dự án được phát triển nội bộ, công ty đó sẽ phải đầu tư nhắm tới mục đích phục vụ một cộng đồng lớn hơn (có thể là trên toàn thế giới). Một số chức năng đáp ứng nhu cầu kinh doanh hạn hẹp của công ty có thể sẽ phải được xem xét lại để chúng cũng có thể phục vụ các hoạt động kinh doanh khác.

Nếu chất lượng của mã thấp hoặc khó xử lý, có khả năng công ty không nên mở mã đó vì những cộng tác viên cũng như các khách hàng tiềm năng sẽ quan ngại về vấn đề chất lượng. Dù sao thì các nhà phát triển cũng cần phải khắc phục những sự cố này vì phần mềm được mã hóa kém rất dễ hỏng hóc: khó cập nhật các tính năng mới và có xu hướng phát triển ra các lỗi phức tạp khó sửa. Những vấn đề này được gọi là *món nợ kỹ thuật* và chúng nên được khắc phục càng sớm sẽ càng tốt cho tất cả mọi người dùng.

Cuối cùng, chúng ta sẽ phải ngạc nhiên khi biết một công ty có thể thường xuyên nhúng các bí mật thương mại, mật khẩu, tài liệu tham khảo cá nhân của các cá nhân hoặc thông tin nhạy cảm khác vào mã nguồn. Các nhà phát triển sẽ phải đầu tư thời gian để loại bỏ những lỗ hổng như vậy. Dù thế nào thì mật khẩu, khóa API, chứng chỉ và thông tin xác thực để truy cập đám mây không bao

giờ nên nằm ở trong mã; chúng phải được quản lý bên ngoài thông qua một dịch vụ an toàn.

Giả sử một công ty đã mở mã của mình và hy vọng sẽ thu được lợi ích từ những đóng góp từ bên ngoài. Một số khách hàng sẽ trả tiền cho việc bảo trì cũng như các tính năng mới. Những khách hàng khác có thể sẽ đưa các nhà phát triển của riêng họ vào dự án, nhưng nhóm phát triển cốt lõi cũng sẽ phải phân bổ thời gian để hỗ trợ họ. Nhóm phát triển cốt lõi cần hướng dẫn những người ở bên ngoài về mã và các tiêu chuẩn mã hóa liên quan. Nhóm này cũng nên hướng dẫn những nhà phát triển đóng góp bên ngoài về những gì cần thêm vào và nơi để gửi lại nhận xét của họ về mã.

Hãy để mắt tới những người có năng lực ở bên ngoài nhóm phát triển. Những cá nhân này có thể là những tân binh đáng giá cho đội ngũ nòng cốt. Họ có thể sở hữu một lượng kiến thức đáng kể và sẽ muốn tham gia vào nhóm.

Khi một nhóm nhà phát triển được trả tiền trong khi những người khác đang tình nguyện cống hiến thời gian của họ, các tình nguyện viên này có thể sẽ cảm thấy mình đang bị lợi dụng hoặc đặt ra câu hỏi về lý do tại sao họ nên đóng góp. Mỗi một nhà phát triển đóng góp, dù là cá nhân tình nguyện hay tổ chức, đều phải trải qua những kiểu suy nghĩ tương tự để quyết định lý do tại sao họ lại đóng góp.

Mã tự do không hề miễn phí, ngay cả khi nó không mang chi phí cấp phép. Nó chỉ đơn giản là một mô hình phát triển khác biệt.

Bài tập Hướng dẫn

1. Mã nguồn mở cải thiện niềm tin của khách hàng vào phần mềm như thế nào?

2. Tại sao các công ty bị cám dỗ bởi việc sử dụng một mô hình kinh doanh lõi mở, và tại sao mô hình đó lại không mang lại các lợi ích của phần mềm mã nguồn mở?

3. Phân phối tự lưu trữ là gì?

4. Những loại trợ giúp điển hình mà các nhà phát triển cung cấp cho những nhà phát triển đóng góp bên ngoài của các dự án mã nguồn mở của họ là gì?

Bài tập Mở rộng

1. Bạn đang xem xét việc xây dựng một sản phẩm độc quyền dựa trên một dự án mã nguồn mở. Bạn sẽ xem xét những yếu tố nào để đưa ra quyết định?

2. Bạn đã xây dựng một số sản phẩm tính phí đăng ký dựa trên một dự án mã nguồn mở. Bạn sẽ làm gì nếu giấy phép mã nguồn mở yêu cầu bạn đóng góp toàn bộ mã của mình lại cho dự án? Ngoài ra, bạn đã thêm hỗ trợ cho một số giao thức truyền thông mới vào dự án mã nguồn mở để hỗ trợ các nhu cầu độc quyền của mình. Nếu được lựa chọn, bạn có yêu cầu dự án mã nguồn mở tích hợp hỗ trợ giao thức này vào mã lõi của họ không?

Tóm tắt

Trong bài học này, chúng ta đã học được giá trị của việc mở mã nguồn. Chúng ta đã xem xét các mô hình kinh doanh phổ biến được sử dụng ngày nay và tầm quan trọng của việc hiểu về ảnh hưởng của giấy phép mã. Chúng ta cũng đã tìm hiểu về các vấn đề cần cân nhắc khi kết hợp mã nguồn mở vào doanh nghiệp của mình cũng như việc mã nguồn mở mang lại lợi ích như thế nào cho khách hàng của mình. Chúng ta cũng đã học về việc chi phí phát triển mã nguồn mở khác với chi phí phát triển phần mềm độc quyền như thế nào.

Đáp án Bài tập Hướng dẫn

1. Mã nguồn mở cải thiện niềm tin của khách hàng vào phần mềm như thế nào?

   Khách hàng có thể kiểm tra chất lượng và tính bảo mật của mã. Họ cũng có thể chắc chắn rằng mình vẫn có thể tiếp tục sử dụng mã nếu các nhà phát triển ban đầu ngừng hỗ trợ mã đó.

2. Tại sao các công ty bị cám dỗ bởi việc sử dụng một mô hình kinh doanh lõi mở, và tại sao mô hình đó lại không mang lại các lợi ích của phần mềm mã nguồn mở?

   Thoạt nhìn, lõi mở có vẻ như sẽ cung cấp tất cả mọi lợi ích của phần mềm mã nguồn mở, đồng thời cho phép một công ty vẫn sử dụng mô hình kinh doanh độc quyền bán giấy phép sử dụng phần mềm. Trên thực tế, mô hình kinh doanh lõi mở không mang lại được những lợi thế của mã nguồn mở trong khi vẫn phải chịu tất cả các chi phí gia tăng của việc phát triển mã mở.

3. Phân phối tự lưu trữ là gì?

   Phân phối tự lưu trữ có nghĩa là một phiên bản phần mềm có thể chạy trên thiết bị của chính khách hàng.

4. Những loại trợ giúp điển hình mà các nhà phát triển cung cấp cho những nhà phát triển đóng góp bên ngoài của các dự án mã nguồn mở của họ là gì?

   Các nhà phát triển của một dự án thường đào tạo và cố vấn cho những nhà phát triển đóng góp bên ngoài, cũng như kiểm tra xem những đóng góp của họ có đáp ứng các tiêu chuẩn về chất lượng và mã hóa của nhóm hay không.

Đáp án Bài tập Mở rộng

1. Bạn đang xem xét việc xây dựng một sản phẩm độc quyền dựa trên một dự án mã nguồn mở. Bạn sẽ xem xét những yếu tố nào để đưa ra quyết định?

   Đầu tiên, hãy quyết định xem phần mềm mã nguồn mở có đáp ứng được nhu cầu của bạn hay không. Hãy kiểm tra chất lượng của nó, các lỗi bảo mật đã được báo cáo công khai và tiềm lực của cộng đồng xung quanh nó. Hãy kiểm tra giấy phép cẩn thận để xem liệu nó có yêu cầu bạn chia sẻ các sửa đổi của mình trên mã hay không. Hãy xác định mức độ mà bạn mong muốn tham gia vào cộng đồng của dự án mã nguồn mở cũng như vai trò của các nhà phát triển của bạn trong cộng đồng đó.

2. Bạn đã xây dựng một số sản phẩm tính phí đăng ký dựa trên một dự án mã nguồn mở. Bạn sẽ làm gì nếu giấy phép mã nguồn mở yêu cầu bạn đóng góp toàn bộ mã của mình lại cho dự án? Ngoài ra, bạn đã thêm hỗ trợ cho một số giao thức truyền thông mới vào dự án mã nguồn mở để hỗ trợ các nhu cầu độc quyền của mình. Nếu được lựa chọn, bạn có yêu cầu dự án mã nguồn mở tích hợp hỗ trợ giao thức này vào mã lõi của họ không?

   Nếu dự án không yêu cầu bạn chia sẻ những thay đổi về mã, có lẽ bạn sẽ không chia sẻ để có thể tính phí đăng ký dễ dàng hơn cho một sản phẩm độc quyền. Nếu bạn phải chia sẻ mã của mình, hãy cung cấp tính năng đăng ký cho một bản phân phối tự lưu trữ. Ngay cả khi bạn không cần chia sẻ mã của mình, bạn có thể sẽ muốn yêu cầu dự án tích hợp hỗ trợ của bạn cho các giao thức truyền thông để bạn không phải triển khai lại phần hỗ trợ đó mỗi khi cài đặt một bản phát hành mới của mã nguồn.

054.2 Mô hình Kinh doanh của Nhà cung cấp Dịch vụ

Tham khảo các mục tiêu LPI

Open Source Essentials version 1.0, Exam 050, Objective 054.2

Khối lượng

2

Các lĩnh vực kiến thức chính

- Hiểu rõ về các mô hình kinh doanh và luồng doanh thu chung cho các tổ chức cung cấp dịch vụ liên quan đến phần mềm mã nguồn mở cũng như nội dung mở
- Hiểu rõ tác động của giấy phép đối với mô hình kinh doanh của nhà cung cấp dịch vụ
- Hiểu rõ các mục tiêu cấp độ dịch vụ và thỏa thuận cấp độ dịch vụ
- Hiểu rõ về nhu cầu bảo vệ an ninh và quyền riêng tư
- Nắm được cấu trúc chi phí và đầu tư cần thiết cho các mô hình kinh doanh dịch vụ phần mềm mã nguồn mở

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Dịch vụ lưu trữ
- Đám mây
- Tư vấn
- Đào tạo
- Kinh doanh phần cứng
- Hỗ trợ người dùng
- Điều khoản Dịch vụ (ToS)
- Mục tiêu Cấp độ Dịch vụ (SLO)

- Thỏa thuận Cấp độ Dịch vụ (SLA)
- Thỏa thuận xử lý dữ liệu

Linux Professional Institute

Bài 1

Chứng chỉ:	Open Source Essentials
Phiên bản:	1.0
Chủ đề:	054 Mô hình kinh doanh Mã nguồn Mở
Mục tiêu:	054.2 Mô hình Kinh doanh của Nhà cung cấp Dịch vụ
Bài học:	1 trên 1

Giới thiệu

Nhà cung cấp dịch vụ là xương sống của bất kỳ một công ty hoặc tổ chức nào hoạt động theo bất kỳ phương thức nào thông qua mạng. Từ Nhà cung cấp dịch vụ Internet (ISP) cung cấp quyền truy cập vào mạng internet toàn cầu đến các ứng dụng chuyên biệt như email hoặc trình quản lý khách hàng, chính các dịch vụ dựa trên phần mềm đã giúp các hoạt động kinh doanh thực tế trở nên khả thi.

Phần mềm mã nguồn mở là một thành phần của nhiều mô hình kinh doanh nhà cung cấp dịch vụ khác nhau. Ví dụ: một hệ điều hành mã nguồn mở có thể tạo thành cơ sở cho một doanh nghiệp dịch vụ lưu trữ. Một công cụ điều phối mã nguồn mở có thể là xương sống của một doanh nghiệp Cơ sở hạ tầng dưới dạng Dịch vụ (IaaS) hoặc Nền tảng dưới dạng Dịch vụ (PaaS) (chẳng hạn như nhà cung cấp đám mây công cộng hoặc nền tảng ảo hoá). Một ứng dụng mã nguồn mở có thể được cung cấp trực tuyến dưới dạng Phần mềm dưới dạng Dịch vụ (SaaS). Một số nhà cung cấp dịch vụ cũng cung cấp các dịch vụ bổ sung liên quan đến phần mềm mã nguồn mở như tư vấn, đào tạo hoặc hỗ trợ khách hàng.

Dòng Doanh thu

Khi dịch vụ phần lớn hoặc hoàn toàn dựa vào phần mềm mã nguồn mở, chúng ta có thể gọi đó là một *mô hình kinh doanh mã nguồn mở*. Vì phần mềm mã nguồn mở có thể được tải xuống và sử dụng mà không phải trả phí nên các mô hình kinh doanh dịch vụ mã nguồn mở đã phát triển các sản phẩm và dịch vụ cụ thể dựa trên tỷ lệ phần mềm tự do trong sản phẩm thực tế bên cạnh nhiều yếu tố khác.

Nếu toàn bộ phần mềm đều tuân theo một giấy phép tự do, mô hình kinh doanh có thể bao gồm cả việc *lưu trữ* — tức là cung cấp phần mềm trên một nền tảng đáng tin cậy và an toàn. Mô hình này sẽ loại bỏ công sức và rủi ro trong quá trình vận hành máy chủ từ phía khách hàng. Đổi lại, khách hàng sẽ trả phí cho nhà cung cấp dịch vụ để sử dụng. Chi phí có thể liên quan đến số lượng tài khoản người dùng, khối lượng dữ liệu được lưu trữ / chuyển giao hoặc các khoảng thời gian nhất định. Trong một số trường hợp, nhiều nhà cung cấp khác nhau sẽ cung cấp các dịch vụ trả phí cho cùng một phần mềm mã nguồn mở.

Mô hình kinh doanh phổ biến này thường liên quan đến việc *đăng ký* (subscriptions) nơi khách hàng trả phí hàng tháng hoặc hàng năm để truy cập dịch vụ. Quyền truy cập thường được kết hợp với các dịch vụ hoặc chức năng bổ sung. Nhà cung cấp có thể cung cấp nhiều cấp đăng ký, tính phí cao hơn cho nhiều tính năng hơn, nhiều người dùng hơn, nhiều dịch vụ toàn diện hơn hoặc đảm bảo mức độ tin cậy cao hơn. Một số nhà cung cấp cũng có thể cung cấp một mức dịch vụ cơ bản miễn phí - một biến thể của mô hình “freemium”.

Một nguồn doanh thu phổ biến khác cho mô hình kinh doanh của nhà cung cấp dịch vụ mã nguồn mở là cung cấp dịch vụ *hỗ trợ*. Hỗ trợ ở đây bao gồm việc tư vấn, cung cấp tài liệu và hướng dẫn cho người dùng khi gặp sự cố trong quá trình sử dụng phần mềm. Nếu khách hàng mong muốn lưu trữ phần mềm tự do trên máy chủ của riêng họ, chỉ riêng việc cài đặt đã thường gây ra nhiều khó khăn vì người dùng có thể sẽ cần cài đặt các công cụ và thư viện hỗ trợ khác, hoặc họ không biết đặt chúng ở đâu để tất cả các thành phần có thể hoạt động cùng nhau hoặc gặp sự cố trên phần cứng và hệ điều hành của mình. Do đó, dịch vụ hỗ trợ và đào tạo từ nhà cung cấp dịch vụ là rất có giá trị.

Các dịch vụ bổ sung (chẳng hạn như các tính năng bổ sung về chức năng hoặc liên quan đến bảo mật) cũng có thể được phát triển và cung cấp cụ thể cho từng khách hàng. Các tính năng hoặc điều chỉnh tuân theo các yêu cầu cụ thể của khách hàng đối với phần mềm như vậy là một sự bổ sung cầu kỳ và có khả năng tăng sinh lợi cho mô hình kinh doanh. Sau đó, nhà cung cấp dịch vụ và khách hàng sẽ quyết định xem các phần bổ sung này sẽ được đưa trở lại vào dự án cơ bản dưới dạng phần mềm tự do hay sẽ trở thành phần mềm độc quyền.

Một số dự án mã nguồn mở có cung cấp *mô hình cấp phép kép*. Phần mềm sẽ có sẵn theo giấy phép phần mềm tự do và đáp ứng nhu cầu của rất nhiều người dùng. Tuy nhiên, một số người dùng sẽ

muốn kết hợp phần mềm này vào một sản phẩm độc quyền; điều này sẽ không thể thực hiện được nếu nó tuân theo một giấy phép tương hỗ. Do đó, một giấy phép dựa trên bản quyền tiêu chuẩn đã được cung cấp cho những người dùng này. Mô hình cấp phép kép sẽ hiệu quả nhất đối với sản phẩm cơ sở dữ liệu vì có nhiều công ty muốn cung cấp một cơ sở dữ liệu giàu tính năng và hiệu quả dưới dạng một phần của một sản phẩm phần mềm độc quyền.

Trong *mô hình doanh thu quảng cáo*, khách hàng sẽ không phải trả tiền để truy cập vào dịch vụ. Thay vào đó, nhà cung cấp dịch vụ sẽ hiển thị quảng cáo cho khách hàng sử dụng dịch vụ và tính phí các công ty muốn đặt quảng cáo của mình. Các nhà cung cấp dịch vụ mã nguồn mở thường tránh sử dụng các dịch vụ quảng cáo độc quyền do lo ngại về quyền riêng tư của người dùng. Tuy nhiên, các dịch vụ quảng cáo mã nguồn mở thay thế sẽ không tham gia vào việc theo dõi quảng cáo xâm lấn.

Trong *mô hình dựa trên hoa hồng*, nhà cung cấp dịch vụ sẽ nhận được phần trăm hoặc một khoản phí để quản lý các giao dịch thông qua dịch vụ trực tuyến của họ. Mô hình này là cơ sở cho hầu hết các trang web thương mại điện tử, trang web đặt vé du lịch và bộ xử lý thanh toán. Rất nhiều dự án mã nguồn mở trong không gian này là các nền tảng mang mục đích chung được nhà cung cấp dịch vụ tự lưu trữ hoặc được cung cấp dưới dạng dịch vụ dựa trên hoa hồng thứ cấp nơi trang web hướng đến khách hàng sẽ trả phần trăm hoặc một khoản phí cho một phiên bản được lưu trữ của nền tảng mã nguồn mở.

Một *mô hình kinh doanh nội dung mở* sẽ liên quan đến việc tạo và phân phối nội dung theo các giấy phép như Tài sản Công cộng Chung để những người khác có thể tự do sử dụng, sửa đổi và chia sẻ nội dung đó. Người dùng được khuyến khích đóng góp, cải thiện nội dung theo thời gian. Mặc dù bản thân nội dung là tự do nhưng doanh thu có thể được tạo ra thông qua các sản phẩm hoặc dịch vụ bổ sung như quyên góp, hiển thị quảng cáo, cung cấp các tính năng freemium, bán hàng hóa liên quan hoặc cung cấp dịch vụ tư vấn, đào tạo hoặc hỗ trợ.

Lựa chọn giữa Dịch vụ Lưu trữ và Tự Lưu trữ

Dịch vụ lưu trữ là loại hình dịch vụ lý tưởng đối với các tổ chức đang tìm kiếm tính dễ sử dụng, chi phí ban đầu thấp, khả năng mở rộng và quản lý bảo mật chuyên nghiệp mà không cần chuyên môn sâu về kỹ thuật. Tuy nhiên, các dịch vụ này cũng đi kèm với những nhược điểm tiềm ẩn về khả năng kiểm soát, tùy chỉnh, quyền riêng tư dữ liệu và sự phụ thuộc vào nhà cung cấp.

Các dịch vụ lưu trữ thường yêu cầu thiết lập tối thiểu và nhà cung cấp dịch vụ sẽ xử lý vấn đề bảo trì, cập nhật và bảo mật hệ thống. Họ thường cung cấp các trung tâm dữ liệu toàn cầu mang lại hiệu suất và độ tin cậy tốt hơn cho người dùng quốc tế. Nhà cung cấp dịch vụ có thể tăng hoặc giảm tài nguyên tuỳ theo nhu cầu mà không cần quá nhiều sự can thiệp từ phía khách hàng. Hơn nữa, các dịch vụ lưu trữ thường có các nhóm bảo mật chuyên nghiệp để quản lý và ứng phó với các mối đe dọa, và các nhà cung cấp dịch vụ có thể cung cấp dịch vụ tuân theo nhiều tiêu chuẩn

quy định khác nhau.

Mặt khác, các dịch vụ lưu trữ sẽ lưu trữ dữ liệu nhạy cảm của khách hàng từ xa; điều này có thể gây ra các mối lo ngại về quyền riêng tư cũng như tính tin cậy của dữ liệu. Một môi trường lưu trữ chia sẻ có thể sẽ gây ra thêm nhiều rủi ro bảo mật. Các sự cố ngừng hoạt động hoặc khoảng thời gian ngừng hoạt động của nhà cung cấp dịch vụ sẽ ảnh hưởng trực tiếp đến tính khả dụng của dịch vụ của khách hàng. Việc phụ thuộc vào một nhà cung cấp duy nhất có thể sẽ gây ra vấn đề nếu nhà cung cấp thay đổi các điều khoản hoặc ngừng cung cấp dịch vụ.

Do đó, các dịch vụ tự lưu trữ sẽ phù hợp với các tổ chức có đầy đủ chuyên môn kỹ thuật và nguồn lực cần thiết để quản lý cơ sở hạ tầng của chính họ. Tuy nhiên, việc tự lưu trữ sẽ khiến chi phí ban đầu tăng cao, cần được bảo trì liên tục và đưa ra thách thức về khả năng mở rộng. Tóm lại, việc lựa chọn giữa dịch vụ lưu trữ hay tự lưu trữ phụ thuộc vào nhu cầu, khả năng kỹ thuật, ngân sách và ưu tiên của từng khách hàng đối với vấn đề kiểm soát, tùy chỉnh và quyền riêng tư đối với dữ liệu.

Tác động của Giấy phép

Về cơ bản, giấy phép phần mềm mã nguồn mở rất phù hợp với mô hình kinh doanh của nhà cung cấp dịch vụ vì khách hàng không mong đợi sở hữu phần mềm cung cấp dịch vụ. Thay vào đó, họ sẽ trả tiền để truy cập vào dịch vụ.

Mặt khác, việc sử dụng phần mềm mã nguồn mở trong các dịch vụ như IaaS, SaaS hoặc PaaS sẽ gây ra các vấn đề pháp lý thường không được làm rõ. Chỉ một số ít giấy phép — đáng chú ý nhất là *Giấy phép Công cộng Chung GNU Affero* (GNU Affero GPL) — quy định rõ ràng về việc sử dụng phần mềm tự do "trong trường hợp đó là phần mềm máy chủ mạng".

Với các giấy phép phổ biến khác — cả giấy phép copyleft hạn chế hơn như Giấy phép Công cộng Chung GNU và các giấy phép linh hoạt như giấy phép BSD — các định nghĩa về "sao chép", "truyền tải" hoặc "phân phối" khi phần mềm được sử dụng qua mạng có một sự linh hoạt nhất định. Quyết định về việc có nên cung cấp mã nguồn cho ứng dụng hay có yêu cầu ghi công hay không và dưới hình thức nào phụ thuộc vào các định nghĩa này. Mối liên hệ với các thành phần phần mềm độc quyền hoặc việc cấp phép cho các thành phần bổ sung do chính mình phát triển cũng phải được làm rõ về mặt pháp lý. Vì những lý do này, trong bối cảnh các mô hình kinh doanh của nhà cung cấp dịch vụ, chúng ta phải làm rõ các vấn đề cấp phép liên quan đến việc sử dụng phần mềm tự do.

Các nhà cung cấp dịch vụ nên công khai liệt kê tất cả phần mềm mã nguồn mở mà họ sử dụng cùng với các giấy phép tương ứng ngay cả khi họ không có nghĩa vụ pháp lý về phương diện này.

Các cân nhắc về vấn đề Bảo mật và Bảo vệ Quyền Riêng tư

Sự thành công của mô hình kinh doanh nhà cung cấp dịch vụ phụ thuộc rất nhiều vào mức độ thoả mãn của trải nghiệm khách hàng, đặc biệt là đối với phần mềm mã nguồn mở nơi khách hàng có quyền tự do tự lưu trữ phần mềm hoặc chọn nhà cung cấp dịch vụ cạnh tranh cho cùng một phần mềm. Do đó, lợi thế cạnh tranh chính của nhà cung cấp dịch vụ mã nguồn mở chính là trải nghiệm khách hàng mà họ cung cấp: có thể là thuận tiện hơn, tiết kiệm chi phí hơn, đáng tin cậy hơn, an toàn hơn hoặc hiệu suất cao hơn so với tự lưu trữ. Tuy nhiên, sự đánh đổi ở đây là khách hàng sẽ phải chia sẻ dữ liệu với nhà cung cấp dịch vụ; điều này có thể gây ra nhiều mối lo ngại về quyền riêng tư đối với dữ liệu cá nhân hoặc dữ liệu nhạy cảm.

Về mặt bảo mật, khách hàng nên đảm bảo rằng nhà cung cấp dịch vụ có một quy trình mạnh mẽ để áp dụng các bản vá bảo mật kịp thời. Chúng ta cũng cần đánh giá cách dịch vụ quản lý các phần phụ thuộc vào các dự án mã nguồn mở khác để đảm bảo tất cả các phần mềm đều an toàn và được cập nhật. Khách hàng cũng nên đánh giá các chính sách bảo mật của nhà cung cấp bao gồm cách họ xử lý mã hóa dữ liệu, kiểm soát truy cập và phản hồi sự cố.

Tính minh bạch của phần mềm mã nguồn mở cho phép cộng đồng xem xét mã một cách kỹ lưỡng; điều này có thể tăng cường tính bảo mật thông qua việc xác định và sửa các lỗ hổng. Vì vậy, khách hàng nên tìm kiếm các đánh giá hoặc bản rà soát về phần mềm của các bên thứ ba có uy tín hoặc các chuyên gia bảo mật trong cộng đồng. Các cuộc rà soát này sẽ xác nhận rằng nhà cung cấp có tuân theo các thông lệ và tiêu chuẩn mã hóa an toàn trong quá trình phát triển hay không, cũng như xét xem nhà cung cấp có sử dụng các đường ống CI/CD với các phiên kiểm tra bảo mật tích hợp để phát hiện sớm các lỗ hổng hay không.

Về vấn đề dữ liệu riêng tư, khách hàng nên xác minh rằng dịch vụ chỉ thu thập và lưu trữ dữ liệu cần thiết cho hoạt động của mình để giảm thiểu rủi ro bị lộ dữ liệu. Dịch vụ nên mã hóa dữ liệu cả khi truyền và khi lưu trữ với mã hóa mạnh. Cơ chế kiểm soát truy cập phải đảm bảo rằng chỉ những nhân viên được ủy quyền mới có thể truy cập vào các dữ liệu nhạy cảm.

Khách hàng nên kiểm tra xem dịch vụ có cung cấp cho người dùng quyền kiểm soát dữ liệu của họ hay không, chẳng hạn như khả năng xóa hoặc xuất dữ liệu. Dịch vụ phải tuân thủ các quy định về quyền riêng tư có liên quan — chẳng hạn như Quy định Bảo vệ Dữ liệu Chung (GDPR) của Liên minh châu Âu và Đạo luật Quyền Riêng tư của Người Tiêu dùng California (CCPA) — và cung cấp tính minh bạch về các hoạt động xử lý dữ liệu của mình.

Một điều quan trọng là khách hàng phải kiểm tra xem nhà cung cấp có chính sách và quy trình rõ ràng để ứng phó với vi phạm dữ liệu — bao gồm cả các yêu cầu thông báo — hay không. Khách hàng cũng nên xem xét chính sách bảo mật của nhà cung cấp để hiểu cách dữ liệu được thu thập, sử dụng, chia sẻ và bảo vệ, đồng thời xem xét bất kỳ bên thứ ba nào mà nhà cung cấp có thể chia sẻ dữ liệu tới và đảm bảo rằng họ cũng tuân thủ các tiêu chuẩn bảo mật một cách nghiêm ngặt.

Nhìn chung, việc xem xét phản hồi và đánh giá của cộng đồng rất có giá trị đối với việc đánh giá mức độ uy tín và đáng tin cậy của phần mềm và nhà cung cấp dịch vụ. Một cộng đồng hoặc tổ chức mạnh mẽ nên tích cực duy trì và hỗ trợ dự án mã nguồn mở. Khách hàng nên xác minh rằng nhà cung cấp có một quy trình sao lưu dữ liệu thường xuyên và kế hoạch phục hồi thảm họa mạnh mẽ, đồng thời kiểm tra xem dịch vụ có sử dụng dữ liệu dự phòng để đảm bảo tính khả dụng và độ tin cậy hay không.

Các nhà cung cấp dịch vụ nên biết rằng khách hàng sẽ đánh giá các hoạt động bảo mật và bảo toàn quyền riêng tư, danh tiếng trong cộng đồng, việc tuân thủ các quy định và tính minh bạch trong việc xử lý dữ liệu của họ để từ đó nghiêm túc thực hiện các bước tuân thủ theo các thông lệ một cách tốt nhất.

Thỏa thuận chung giữa Nhà cung cấp Dịch vụ và Khách hàng

Các nhà cung cấp dịch vụ thường áp dụng các điều khoản và thỏa thuận pháp lý để tạo thành xương sống pháp lý vận hành mối quan hệ giữa công ty và khách hàng của mình. Các thỏa thuận này giúp đảm bảo giao tiếp được rõ ràng, đặt ra kỳ vọng, xác định trách nhiệm và cung cấp sự bảo vệ pháp lý cho cả hai bên.

Điều khoản dịch vụ (Terms of Service - ToS) cho một dịch vụ — còn được gọi là *Điều khoản và Điều kiện* (Terms and Conditions - T&C) hoặc *Điều khoản sử dụng* (Terms of Use) — là một thỏa thuận pháp lý giữa nhà cung cấp dịch vụ và khách hàng hoặc người dùng dịch vụ đó. ToS nêu rõ các quy tắc, trách nhiệm và giới hạn chi phối việc sử dụng dịch vụ. Nó bảo vệ cả nhà cung cấp dịch vụ và khách hàng bằng cách xác định rõ ràng những gì mỗi bên có thể và không thể làm trong khi cung cấp hoặc sử dụng dịch vụ. Một số yếu tố chung của thỏa thuận ToS là xác nhận rằng người dùng đồng ý tuân thủ các điều khoản trong đó, hướng dẫn về hành vi được chấp nhận và các hoạt động bị cấm, thông tin chi tiết về người sở hữu nội dung do người dùng tạo hoặc tải lên, thông tin về việc tạo tài khoản, bảo mật và chấm dứt, quy định về trọng tài hoặc hòa giải để giải quyết xung đột và giới hạn về trách nhiệm pháp lý của nhà cung cấp dịch vụ đối với thiệt hại.

Chính sách bảo mật là một tuyên bố nêu rõ cách nhà cung cấp dịch vụ thu thập, sử dụng, lưu trữ và bảo vệ dữ liệu của người dùng. Một số yếu tố thường thấy của một chính sách bảo mật là các loại dữ liệu mà nhà cung cấp dịch vụ thu thập và phương pháp thu thập của họ, cách họ sử dụng dữ liệu, các điều kiện mà nhà cung cấp dịch vụ có thể chia sẻ dữ liệu với bên thứ ba và thông tin về quyền của người dùng liên quan đến dữ liệu của họ (chẳng hạn như quyền truy cập, chỉnh sửa và xóa).

Một *Thỏa thuận mức dịch vụ* (Service Level Agreement - SLA) là một thỏa thuận chính thức được ghi chép lại giữa nhà cung cấp dịch vụ và khách hàng; trong đó nêu rõ mức dịch vụ được mong đợi, bao gồm các số liệu hiệu suất cụ thể, trách nhiệm và kỳ vọng. SLA xác định các tiêu chuẩn cung cấp dịch vụ, cung cấp một khuôn khổ rõ ràng để đo lường và quản lý hiệu suất dịch vụ. Một

số yếu tố chung của SLA có bao gồm một mô tả chi tiết về các dịch vụ được cung cấp, các mục tiêu cụ thể về hiệu suất dịch vụ như thời gian hoạt động và thời gian phản hồi, các phương pháp đo lường và báo cáo hiệu suất, hậu quả khi không đạt được các mục tiêu hiệu suất và các thủ tục để xem xét và cập nhật SLA.

Mục tiêu mức dịch vụ (Service Level Objective - SLO) là thành phần chính của Thỏa thuận mức dịch vụ; trong đó có nêu rõ các mục tiêu hoặc mục đích có thể đo lường được đối với hiệu suất dịch vụ. Các mục tiêu này định lượng mức dịch vụ được mong đợi giữa nhà cung cấp dịch vụ và khách hàng và được sử dụng để đảm bảo rằng dịch vụ luôn đáp ứng các tiêu chuẩn đã thỏa thuận. Một số thành phần chung của SLO bao gồm các số liệu cụ thể mà nhà cung cấp dịch vụ sẽ đo lường để đánh giá hiệu suất dịch vụ (ví dụ như thời gian hoạt động, thời gian phản hồi, thời gian giải quyết và thông lượng), các giá trị mong muốn hoặc được kỳ vọng cho từng số liệu hiệu suất, các kỹ thuật và công cụ được sử dụng để đo lường và giám sát các số liệu hiệu suất, các thành phần hoặc khía cạnh cụ thể của dịch vụ mà SLO đảm bảo (ví dụ như phần cứng, phần mềm, thành phần mạng hoặc các tiến trình cụ thể) cũng như hậu quả hoặc phần thưởng dựa trên việc dịch vụ có đáp ứng được các mục tiêu mức dịch vụ hay không (ví dụ như hình phạt tài chính, tín dụng dịch vụ hoặc khoản thưởng hiệu suất).

Thỏa thuận xử lý dữ liệu (Data Processing Agreement - DPA) là một hợp đồng giữa khách hàng (bên kiểm soát dữ liệu) và nhà cung cấp dịch vụ (bên xử lý dữ liệu) tham gia vào việc xử lý dữ liệu cá nhân. DPA nêu chi tiết trách nhiệm và nghĩa vụ của cả hai bên để đảm bảo tuân thủ luật bảo vệ dữ liệu (chẳng hạn như GDPR). Ở một số khu vực pháp lý, DPA giữa nhà cung cấp dịch vụ và khách hàng sẽ là một yếu tố bắt buộc.

Cấu trúc Chi phí và Đầu tư

Chi phí đáng kể nhất trong mô hình kinh doanh của nhà cung cấp dịch vụ là chi phí cố định liên quan đến việc lưu trữ các dịch vụ như mua phần cứng máy chủ, thanh toán cho không gian trung tâm dữ liệu, điện để cấp điện và làm mát máy chủ, băng thông mạng và lương cho nhân viên triển khai và bảo trì. Một số nhà cung cấp dịch vụ cũng có thể có các khoản chi phí biến động cho giấy phép phần mềm hoặc dịch vụ của bên thứ ba. Những khách hàng chọn tự lưu trữ dịch vụ sẽ trực tiếp chịu các khoản chi phí này.

Bằng cách lựa chọn phần mềm mã nguồn mở, các nhà cung cấp dịch vụ và khách hàng có thể loại bỏ chi phí cấp phép hoặc giảm bớt so với phần mềm độc quyền vì, theo định nghĩa, phần mềm mã nguồn mở không tính phí cấp phép phần mềm. Việc sử dụng phần mềm mã nguồn mở để xây dựng dịch vụ cũng có thể tiết kiệm thời gian cũng như tài nguyên phát triển; việc bảo trì và cập nhật do cộng đồng thúc đẩy cũng có thể giảm bớt một số khoản chi phí dài hạn. Tuy nhiên, điều quan trọng ở đây là phải xem xét tổng chi phí sở hữu (Total Cost of Ownership - TCO) của phần mềm mã nguồn mở; con số này có thể bao gồm cả chi phí hỗ trợ, bảo trì và tích hợp nếu có.

Bài tập Hướng dẫn

1. Hãy nêu một số cách mà nhà cung cấp dịch vụ có thể có doanh thu mà không cần phải tính phí người dùng dịch vụ.

2. Hãy nêu ra một số lý do có thể cho phép nhà cung cấp dịch vụ cung cấp phần mềm của mình để khách hàng tự lưu trữ.

3. Tại sao một nhà cung cấp dịch vụ lại phát hành phần mềm của mình theo giấy phép GNU Affero GPL?

Bài tập Mở rộng

1. Chính sách Sử dụng được Chấp nhận (Acceptable Use Policy - AUP) là gì?

2. Thỏa thuận Cấp phép Người dùng Cuối (End User License Agreement - EULA) là gì?

Tóm tắt

Bài học này đã nói về các mô hình kinh doanh của nhà cung cấp dịch vụ dựa trên phần mềm mã nguồn mở và nhiều dòng doanh thu khác nhau bao gồm hình thức đăng ký theo dõi, doanh thu quảng cáo, tỷ lệ theo giờ và các mô hình dựa trên hoa hồng. Bài học này đã nêu bật lên tác động của các giấy phép phần mềm mã nguồn mở đối với các nhà cung cấp dịch vụ, nhấn mạnh đến lợi ích và nghĩa vụ của họ. Các cân nhắc về bảo mật, quyền riêng tư và độ tin cậy của dịch vụ cùng với tầm quan trọng của Điều khoản Dịch vụ (ToS), Thỏa thuận Mức Dịch vụ (SLA) và Thỏa thuận Xử lý Dữ liệu (DPA) cũng đã được đề cập.

Đáp án Bài tập Hướng dẫn

1. Hãy nêu một số cách mà nhà cung cấp dịch vụ có thể có doanh thu mà không cần phải tính phí người dùng dịch vụ.

   Dịch vụ có thể cung cấp quảng cáo để nhà quảng cáo phải trả tiền. Dịch vụ có thể tính hoa hồng cho các nhà cung cấp cung cấp sản phẩm hoặc dịch vụ trên trang web (chẳng hạn như trong lĩnh vực bán lẻ hoặc du lịch). Nhà cung cấp dịch vụ có thể bán dữ liệu khách hàng - một hoạt động mà nhiều khách hàng sẽ phản đối và cần được giải thích trong Điều khoản Dịch vụ.

2. Hãy nêu ra một số lý do có thể cho phép nhà cung cấp dịch vụ cung cấp phần mềm của mình để khách hàng tự lưu trữ.

   Khách hàng tiềm năng có thể dùng thử phần mềm để kiểm tra các tính năng và độ tin cậy trước khi đăng ký với nhà cung cấp dịch vụ. Khách hàng cũng có thể sẽ tìm thấy và thậm chí là đề xuất sửa lỗi.

3. Tại sao một nhà cung cấp dịch vụ lại phát hành phần mềm của mình theo giấy phép GNU Affero GPL?

   Affero GPL yêu cầu các nhà cung cấp dịch vụ khác phải phát hành bất kỳ thay đổi nào mà họ thực hiện đối với phần mềm theo cùng một giấy phép. Điều khoản này ngăn cản các đối thủ cạnh tranh lợi dụng nhà cung cấp dịch vụ bằng cách thêm một vài cải tiến mà họ giữ độc quyền và sau đó quảng cáo phiên bản dịch vụ của họ là vượt trội hơn bản gốc.

Đáp án Bài tập Mở rộng

1. Chính sách Sử dụng được Chấp nhận (Acceptable Use Policy - AUP) là gì?

   Chính sách Sử dụng được Chấp nhận (AUP) là chính sách xác định mục đích sử dụng dịch vụ được chấp nhận và không được chấp nhận nhằm ngăn chặn việc lạm dụng và đảm bảo một môi trường an toàn và đáng tin cậy. Một số yếu tố chung của AUP có bao gồm một mô tả về các hành động cụ thể không được phép (như gửi thư rác hoặc các hành vi gian lận), nghĩa vụ của người dùng trong việc sử dụng dịch vụ một cách có trách nhiệm, các hành động mà nhà cung cấp dịch vụ có thể thực hiện trong trường hợp vi phạm (chẳng hạn như đình chỉ tài khoản của người dùng) và các thủ tục báo cáo và giải quyết vi phạm.

2. Thỏa thuận Cấp phép Người dùng Cuối (End User License Agreement - EULA) là gì?

   Thỏa thuận Cấp phép Người dùng Cuối (EULA) là một hợp đồng pháp lý giữa nhà cung cấp phần mềm độc quyền và người dùng cuối (không phải là một công ty hoặc tổ chức) cấp cho người dùng quyền sử dụng phần mềm theo những điều kiện nhất định. Một số yếu tố phổ biến của EULA bao gồm phạm vi và giới hạn của giấy phép (ví dụ như phạm vi sử dụng cá nhân hoặc giấy phép không thể chuyển nhượng), các hành vi bị cấm (ví dụ như kỹ thuật đảo ngược hoặc tái phân phối), các điều kiện mà theo đó giấy phép có thể bị chấm dứt, quyền sở hữu và chính sách bảo vệ quyền sở hữu trí tuệ. Các dịch vụ mã nguồn mở hoàn toàn sẽ thay thế một giấy phép phần mềm mã nguồn mở thay cho một EULA độc quyền.

054.3 Tuân thủ và Giảm thiểu Rủi ro

Tham khảo các mục tiêu LPI

Open Source Essentials version 1.0, Exam 050, Objective 054.3

Khối lượng

3

Các lĩnh vực kiến thức chính

- Hiểu rõ cách thức đảm bảo tuân thủ giấy phép
- Hiểu rõ cách duy trì thông tin về giấy phép
- Hiểu rõ khái niệm Văn phòng Chương trình Mã nguồn Mở
- Hiểu rõ ý nghĩa của bản quyền, bằng sáng chế và nhãn hiệu đối với các mô hình kinh doanh mã nguồn mở
- Nắm được các rủi ro pháp lý liên quan đến các mô hình kinh doanh nguồn mở
- Nắm được các rủi ro tài chính liên quan đến các mô hình kinh doanh mã nguồn mở

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Phân tích Thành phần Phần mềm (SCA)
- Danh mục Vật liệu Phần mềm (SBOM)
- Trao đổi dữ liệu gói phần mềm (SPDX)
- OWASP CycloneDX
- Văn phòng Chương trình Mã nguồn Mở (OSPO)
- Bảo hành sản phẩm
- Trách nhiệm sản phẩm
- Quy định xuất khẩu

- Tác động của việc sáp nhập và mua lại

Bài 1

Chứng chỉ:	Open Source Essentials
Phiên bản:	1.0
Chủ đề:	054 Mô hình kinh doanh Mã nguồn Mở
Mục tiêu:	054.3 Tuân thủ và Giảm thiểu Rủi ro
Bài học:	1 trên 1

Giới thiệu

Có một câu nói đùa thường nghe về mã nguồn mở rằng “Phần mềm Tự do nhưng lại không thể dùng một cách 'tự do'” ("Free software doesn’t come for free"). Mặc dù phần mềm mã nguồn mở và tự do cho phép chúng ta thoả thích đổi mới và sáng tạo nhưng người dùng và nhà phát triển vẫn sẽ phải tuân thủ một số quy tắc. Bài học này sẽ đề cập đến sự tuân thủ và những rủi ro trong phần mềm mã nguồn mở.

Bài học này sẽ liệt kê các bước cơ bản mà nhà phát triển cần tuân thủ về giấy phép, rủi ro khi sử dụng phần mềm mã nguồn mở và cách theo dõi cũng như lập danh mục phần mềm mà tổ chức đang sử dụng. Chúng ta sẽ xem cách một Văn phòng Chương trình Mã nguồn Mở (Open Source Program Office - OSPO) chuyển các tác vụ này thành chính sách và thúc đẩy chúng như thế nào.

Các Yêu cầu trong việc Phát hành Phần mềm dựa trên các Thành phần Mã nguồn Mở

Nếu một tổ chức sử dụng phần mềm trong phạm vi nội bộ, các giấy phép phần mềm mã nguồn mở và tự do sẽ không áp đặt bất kỳ một yêu cầu nào. Tổ chức có thể xác định các quy tắc của riêng

mình để ngăn ngừa các lỗ hổng và đảm bảo rằng mọi người đều sử dụng cùng một thành phần cũng như nhiều lý do khác. Những điều trên nằm ngoài phạm vi của bài học này. Bất kỳ yêu cầu nào mà các giấy phép mã nguồn mở hoặc chính tổ chức áp đặt đối với việc sử dụng phần mềm đều phải được ghi lại và tổ chức sẽ phải cung cấp quy trình đào tạo về các chính sách của mình.

Ngay cả khi tổ chức chạy phần mềm trên máy chủ web của mình và cung cấp các dịch vụ có tương tác với người dùng bên ngoài tổ chức, hầu hết các giấy phép phần mềm mã nguồn mở và tự do đều sẽ không áp đặt bất kỳ yêu cầu gì. Tuy nhiên, Giấy phép Công cộng Chung GNU Affero sẽ yêu cầu tuân thủ các quy định copyleft đối với phần mềm chạy dưới dạng dịch vụ.

Các yêu cầu pháp lý của mỗi một giấy phép phần mềm mã nguồn mở và tự do phổ biến đều đã được trình bày đầy đủ trong các bài học khác; vì vậy, phần này sẽ chỉ cung cấp một danh sách các hành vi mà các nhà phát triển và nhà quản lý nên thực hiện để tuân thủ:

Kiểm tra các thành phần mã nguồn mở

Tổ chức cần có các chính sách và quy trình đào tạo rõ ràng về việc sẽ sử dụng phần mềm mã nguồn mở nào và vị trí nơi triển khai phần mềm đó trong sản phẩm. Các chính sách như vậy bao gồm việc tuân thủ cũng như các vấn đề khác như đảm bảo an ninh và việc tham gia vào cộng đồng phát triển phần mềm.

Giữ nguyên giấy phép gốc trong mã

Các nhà phát triển không được xóa giấy phép khỏi thành phần mà họ sử dụng. Việc đưa giấy phép vào mã nguồn thường là một yêu cầu bắt buộc trong giấy phép. Trên thực tế, việc xóa giấy phép có thể bị coi là đạo nhái vì nhà phát triển đang vờ như chính họ là người đã viết mã. Việc xóa giấy phép cũng có thể dẫn đến nhiều rắc rối nghiêm trọng sau này vì tổ chức có thể đã vi phạm giấy phép khi đưa mã vào sản phẩm của họ.

Theo dõi giấy phép

Tổ chức phải duy trì một danh mục hiển thị giấy phép của từng tệp hoặc gói mà tổ chức sử dụng để đảm bảo rằng chúng tuân thủ các giấy phép.

Ghi công Tác giả

Gần như tất cả các giấy phép đều yêu cầu ghi nhận công lao của chủ sở hữu bản quyền (thường được gọi là “tác giả”) khi thảo luận và quảng bá phần mềm. Mỗi giấy phép sẽ giải thích những gì cần đưa vào thông báo này: thông thường, chúng sẽ yêu cầu một thông báo bản quyền tiêu chuẩn có (thời gian) năm và tên của chủ sở hữu bản quyền. Thông báo này phải xuất hiện trong mọi tài liệu liên quan đến sản phẩm sử dụng thành phần bao gồm cả quảng cáo và trang web dành cho thiết bị hoặc chương trình.

Không ám chỉ xác thực

Một số giấy phép BSD sẽ yêu cầu người phân phối sản phẩm có chứa các thành phần phải đảm

bảo rằng họ không ám chỉ rằng chủ sở hữu bản quyền đã xác thực sản phẩm này. Ngay cả khi yêu cầu không được nêu ra, việc tuân thủ nó là một hành vi thuộc phạm trù đạo đức.

Cung cấp mã nguồn cho các giấy phép copyleft

Nếu một tổ chức phân phối một tệp nhị phân chứa các thành phần copyleft — cho dù đó là bản phân phối phần mềm hay trên thiết bị — tổ chức đó sẽ phải cung cấp cho công chúng mã nguồn của các thành phần đó. Nếu tổ chức này thay đổi mã và tái phân phối sản phẩm phái sinh ở dạng nhị phân thì mã nguồn cho phiên bản đã được thay đổi (*phái sinh*) cũng phải được cung cấp cho công chúng. Việc phân phối có thể được thực hiện thông qua bất kỳ phương tiện nào giúp công chúng có thể dễ dàng truy cậpđược (như đăng mã trên trang web hoặc cung cấp mã trên một đĩa cứng hoặc ổ USB).

Không đưa các thành phần copyleft vào các sản phẩm độc quyền

Nếu tổ chức đưa các thành phần copyleft vào một sản phẩm, trong một số trường hợp, tổ chức sẽ phải phát hành toàn bộ sản phẩm theo cùng một giấy phép copyleft. Các điều kiện kích hoạt yêu cầu này sẽ khác nhau tùy theo từng giấy phép. Tuy nhiên, đôi khi chúng ta sẽ không thể làm rõ được hình thức sử dụng nào sẽ kích hoạt yêu cầu copyleft. Vì vậy, ngoại trừ các tình huống rõ ràng — như sử dụng thư viện mã nguồn mở (sẽ không kích hoạt yêu cầu tương hỗ của copyleft) — các nhà phát triển cần rất cẩn thận khi sử dụng các thành phần copyleft trừ khi tổ chức đã chuẩn bị phát hành sản phẩm của mình theo cùng một giấy phép.

Ghi lại các thay đổi trên mã

Khi phân phối các phiên bản mã đã qua sửa đổi, hãy nêu rõ những thay đổi mà tổ chức đã thực hiện.

Cấp Quyền Bằng Sáng chế

Một số giấy phép phần mềm mã nguồn hoặc tự do sẽ yêu cầu cấp quyền sử dụng bằng sáng chế đối với phần mềm. Do đó, nếu một tổ chức phát hành mã theo giấy phép như vậy và nắm giữ bằng sáng chế đối với một số quy trình trong mã, tổ chức đó sẽ không thể tính phí hoặc áp dụng các biện pháp kiểm soát khác dựa trên bằng sáng chế của mình đối với bất kỳ ai sử dụng hoặc điều chỉnh mã.

Tôn trọng Nhãn hiệu

Một số phần mềm mã nguồn mở sẽ được bảo hộ nhãn hiệu. Nhãn hiệu có thể bao gồm các từ và cụm từ, logo, hình ảnh cùng nhiều yếu tố khác. Một mặt, một tổ chức sẽ phải xử lý nhãn hiệu đúng cách đối với bất kỳ một phần mềm nào mà tổ chức đó sử dụng: một hành vi vi phạm phổ biến chính là nhại lại, thay đổi hoặc chỉ đơn giản là hiển thị nhãn hiệu mà không có sự cho phép. Mặt khác, nếu tổ chức đó muốn sử dụng nhãn hiệu, họ sẽ phải tuân thủ các yêu cầu của chủ sở hữu nhãn hiệu; điều này có thể loại trừ việc sửa đổi phần mềm.

Rủi ro của Phần mềm Mã nguồn Mở

Bài học này nói về sự tuân thủ; vì vậy, chúng ta sẽ tập trung chủ yếu vào những rủi ro trong phương diện này và chỉ một số ít trong các phương diện khác.

Vi phạm Giấy phép

Giấy phép phần mềm mã nguồn mở và tự do phải được coi trọng ngang với các giấy phép và hợp đồng khác. Các tổ chức thực thi chúng (như Tổ chức Bảo vệ Quyền Tự do Phần mềm) sẽ hoạt động thay mặt cho các dự án copyleft nhỏ lẻ không có cơ chế thực thi riêng.

Các tổ chức này thường coi việc kiện cáo như giải pháp cuối cùng. Hầu hết các vi phạm thường đều là vô tình và có thể dễ dàng giải quyết thông qua việc phổ cập kiến thức. Tuy vậy, việc một tổ chức bị coi là thiếu hiểu biết và vô cảm đối với các cộng đồng mà phần mềm của tổ chức đó phụ thuộc vào vẫn sẽ gây ảnh hưởng đến danh tiếng của chính tổ chức đó.

Có rất nhiều vụ kiện thực sự đã xảy ra khi một người dùng phần mềm từ chối hợp tác một cách trắng trợn và khi phần mềm và bên bị đơn đủ quan trọng.

Ngay cả khi một tổ chức không phải đối mặt với một vụ kiện, thiệt hại mà hành vi vi phạm giấy phép gây ra cho mối quan hệ của tổ chức đó với cộng đồng cũng như là danh tiếng của tổ chức có thể là rất lớn. Một dự án có thể đi lệch khỏi quỹ đạo bởi một chuyện đơn giản như việc các câu hỏi của nhà phát triển không được trả lời trên các diễn đàn dành riêng cho phần mềm mà họ đang cố gắng sử dụng.

Nó có thể sẽ trở thành một thảm họa đối với một cá nhân nào đó khi họ tìm thấy phần mềm copyleft trong một sản phẩm độc quyền. Để tuân thủ giấy phép, các nhà phát triển sẽ phải loại bỏ toàn bộ mã copyleft hoặc là giải phóng sản phẩm của họ theo giấy phép copyleft. Việc cung cấp phần mềm tự do với mã nguồn có sẵn có thể gây ra những tác động nghiêm trọng đến các mô hình kinh doanh dựa trên phí cấp phép hoặc các phương thức khác nhằm nắm giữ độc quyền giá trị của sản phẩm.

Niềm tin và Uy tín

Chúng ta đã thấy rằng việc vi phạm giấy phép có thể gây ra các thiệt hại rất lớn. Các loại hành vi khác làm mất lòng tin và danh tiếng cũng có thể gây ra nhiều rủi ro.

Có một số tổ chức phát hành phần mềm theo giấy phép phần mềm mã nguồn mở hoặc tự do, sau đó tại một thời điểm nào đó lại thông báo rằng các phiên bản trong tương lai sẽ được phát hành theo giấy phép độc quyền. Sự thay đổi này có thể sẽ khá hấp dẫn vì nhiều dự án nguồn mã mở không có đủ khách hàng và từ đó không có được đủ hỗ trợ về mặt tài chính.

Nhưng những thay đổi về giấy phép như vậy sẽ gây ra bất mãn cho cả phía khách hàng lẫn phía các nhà phát triển bên ngoài đóng góp vào dự án. Đôi khi, các nhà phát triển bên ngoài sẽ sử dụng phiên bản tự do và tiếp tục phát triển độc lập — phương hướng phát triển được gọi là *phân nhánh* (fork) dự án. Phiên bản tự do có thể sẽ trở nên cạnh tranh hơn so với phiên bản độc quyền của công ty và thu hút khách hàng rời xa công ty.

Việc tham gia diễn đàn dự án cũng có thể có rủi ro. Một công ty phải học cách để trở thành một công dân tốt. Các vấn đề phổ biến gồm có:

- Cố gắng sử dụng các đóng góp về tài chính hoặc mã của một công ty làm đòn bẩy để buộc dự án đi theo hướng mà các nhà phát triển không mong muốn
- Đưa ra quá nhiều yêu cầu đối với cộng đồng: chẳng hạn như gây áp lực với quá nhiều yêu cầu về tính năng hoặc thậm chí là quá nhiều câu hỏi
- Có hành vi thô lỗ theo nhiều cách khác nhau và vi phạm quy tắc ứng xử của dự án
- Cố gắng chi phối hoạt động quảng cáo hoặc lợi dụng sự thành công của dự án theo những cách không phù hợp

Đầu tư không được đền đáp

Các mô hình kinh doanh sẽ được thảo luận trong bài học khác; phần này sẽ chỉ tập trung lưu ý đến rủi ro tài chính khi tham gia vào các dự án mã nguồn mở.

Các công ty có thể bắt đầu hoặc tham gia các dự án mã nguồn mở với kỳ vọng thu được doanh thu thông qua một số cơ chế như hợp đồng hỗ trợ, hợp đồng SaaS, thu thập dữ liệu hoặc thậm chí là quyên góp và tài trợ. Thật không may, những người thực hiện các dự án mã nguồn mở thường thấy được rằng chúng ít sinh lợi hơn so với mong đợi. Tất nhiên bất kỳ doanh nghiệp nào cũng phải chấp nhận rủi ro khi bắt đầu một dự án mới, nhưng mã nguồn mở lại là lĩnh vực đặc biệt khó để tìm được một nguồn thu nhập đáng tin cậy.

Mã nguồn mở có vẻ bền vững nhất khi nó hỗ trợ một số hình thức thu nhập khác, chẳng hạn như việc bán thiết bị phần cứng, ô tô, máy in, v.v.

Phân nhánh

Như chúng ta đã thấy, các thành viên của một dự án đôi khi sẽ không thể thống nhất về lộ trình, tư cách lãnh đạo hoặc các khía cạnh khác của dự án và tạo ra một dự án thay thế trên cùng một cơ sở mã. Việc phân nhánh (forking) cũng có thể được thực hiện bởi một tổ chức để tạo ra một phiên bản chuyên biệt của phần mềm. Ví dụ: Android đã sử dụng một phiên bản Linux chuyên biệt do Google duy trì (Google cũng đóng góp rất nhiều cho phiên bản cốt lõi của Linux nhưng không phải lúc nào chúng cũng được chấp nhận).

Các tổ chức có thể tạo ra một nhánh vì nhiều lý do khác nhau. Họ có thể gửi các thay đổi của mình cho dự án cốt lõi và bị từ chối vì các nhà phát triển khác không thích chúng. Trong một dự án có giấy phép linh hoạt hoặc một dự án chỉ được sử dụng trong tổ chức, họ có thể sẽ muốn giữ bí mật về các thay đổi của mình.

Rủi ro của việc phânnhánh là các nhà phát triển của dự án nhánh sẽ phải chịu trách nhiệm bảo trì toàn bộ sản phẩm. Nếu các bản sửa lỗi quan trọng hoặc các tính năng mới được thêm vào dự án cốt lõi, các nhà phát triển của dự án nhánh sẽ phải sao chép các thay đổi hoặc từ bỏ các lợi ích từ chúng. Dần dần khi thời gian trôi qua và các dự án ngày càng tách biệt nhau, việc theo kịp các thay đổi trong dự án cốt lõi sẽ trở nên ngày càng khó khăn hơn.

Giấy phép không tương thích

Như đã giải thích trong các bài học khác, một số giấy phép phần mềm mã nguồn mở và tự do sẽ có các điều khoản không tương thích. Các thành phần như vậy không thể được sử dụng cùng nhau trong một sản phẩm.

Vấn đề này thường phát sinh trong quá trình sáp nhập hoặc mua lại. Có thể là mỗi công ty đã sử dụng phần mềm với một giấy phép cụ thể và nếu họ muốn kết hợp mã trong các dự án của mình, họ phải đảm bảo rằng các giấy phép tương thích với nhau.

Trách nhiệm Sản phẩm

Nhiều giấy phép phần mềm mã nguồn mở (và các điều khoản dịch vụ cho nhiều phần mềm và dịch vụ độc quyền) từ chối triệt để mọi trách nhiệm pháp lý đối với việc sử dụng phần mềm. Nói cách khác, giấy phép hoặc các điều khoản dịch vụ không cung cấp bất kỳ hình thức bảo hành hoặc đảm bảo nào.

Các nhà cung cấp phần mềm hiếm khi phải chịu trách nhiệm về các vấn đề liên quan đến phần mềm của họ, nhưng khách hàng đôi khi vẫn sẽ kiện họ hoặc thử các hình thức trừng phạt khác. Tòa án sẽ không cần phải công nhận các điều khoản “không bảo hành”.

Ngày càng có nhiều luật lệ và quy định áp đặt trách nhiệm lên các nhà sáng tạo phần mềm. Có thể thấy được những hạn chế pháp lý này — hoặc ít nhất là các đề xuất về hạn chế — hiện đang đặc biệt nhắm tới trí tuệ nhân tạo; tuy vậy, đôi khi những hạn chế này cũng được áp dụng trong một phạm vi rộng hơn.

Quy định Xuất khẩu

Có rất nhiều quốc gia hạn chế xuất khẩu hàng hóa và phần mềm. Đối với phần mềm, các quy định như vậy thường áp dụng cho các sản phẩm bảo mật và cụ thể là việc sử dụng mật mã. Hoa Kỳ từng

kiểm soát rất chặt chẽ bất kỳ phần mềm nào có chứa mật mã, nhưng các biện pháp kiểm soát đã được nới lỏng đối với các dự án mã nguồn mở.

Các công ty nên lưu ý đến các quy định xuất khẩu tại nơi họ đang phát triển phần mềm trong trường hợp các quy định này ảnh hưởng đến việc bán và phân phối sản phẩm của họ.

Danh mục Vật liệu Phần mềm: Hiểu về thứ chúng ta đang sử dụng

Có nhiều sản phẩm thường đi kèm với một danh sách vật liệu — hiểu đơn giản là một danh sách tất cả các thành phần của chúng. Ví dụ: khi chúng ta mua một sản phẩm tiêu dùng, sản phẩm đó có thể bao gồm một tờ giấy liệt kê tất cả các bộ phận từ lớn đến nhỏ (thậm chí đến từng chiếc đai ốc và bu lông). Danh sách này có thể có chứa các thông tin hữu ích về các bộ phận như kích thước hay số vật liệu để người dùng có thể dễ dàng xác định và đặt mới nếu cần.

Các dự án mã nguồn mở hiện đều có một *Danh mục Vật liệu Phần mềm* (Software Bill of Materials - SBOM, phát âm là “S-bomb”). Một SBOM sẽ liệt kê ít nhất là thông tin về các gói, tên tệp, giấy phép, tác giả hoặc chủ sở hữu và số phiên bản. Nhiều SBOM còn đưa ra cả thông tin về các lỗ hổng bảo mật.

Các dự án sẽ tạo ra một SBOM cho mỗi một bản phát hành bằng các công cụ tự động. Người dùng có thể quét SBOM để tìm ra thông tin mà họ cần. Ví dụ: việc trích xuất giấy phép sẽ giúp một tổ chức nhanh chóng quyết định xem các thành phần có tương thích với mã của riêng họ hay không và liệu việc sử dụng các thành phần cho mã của họ có hợp pháp hay không.

Tương tự như vậy, việc trích xuất thông tin phiên bản trên mỗi gói sẽ giúp các tổ chức khám phá xem các phần khác nhau của sản phẩm có phụ thuộc vào các phiên bản khác nhau của một thư viện cụ thể hay không. Việc sử dụng hai phiên bản của một thư viện ít nhất cũng sẽ gây ra tình trạng "phình" (bloat). Nó cũng có thể gây ra lỗi hoặc nhầm lẫn nếu một thành phần được xây dựng sai phiên bản.

Tiêu chuẩn dành cho SBOM

Vì môi trường máy tính sử dụng số lượng thành phần rất lớn (đôi khi là hàng chục nghìn) và thay đổi thường xuyên, SBOM phải có cấu trúc cực kỳ tốt để thông tin có thể được trích xuất tự động và nhanh chóng. Ở phần này, chúng ta sẽ tìm hiểu hai định dạng phổ biến nhất trong thế giới mã nguồn mở:

Trao đổi Dữ liệu Gói Phần mềm (Software Package Data Exchange - SPDX)

Định dạng này biểu diễn từng gói và toàn bộ nội dung của gói đó dưới dạng cấu trúc cây. Định dạng này ghi lại các mối quan hệ phụ thuộc giữa các tệp và các mối quan hệ khác. Các con trỏ

thông tin — được gọi là các "snippets" — có thể được tạo và sau đó sử dụng trong toàn bộ tài liệu để thông tin chỉ cần được xác định ở một nơi. Định dạng này được phát triển bởi Tổ chức Linux (Linux Foundation).

CycloneDX

Đây là một định dạng lớn hơn với các trường chi tiết hơn, đặc biệt là thông tin về lỗ hổng. Định dạng này được tạo ra bởi *Dự án Bảo mật Ứng dụng Toàn cầu Mở* (OWASP) và được sử dụng phổ biến trong quân đội cũng như các tổ chức khác tập trung vào vấn đề bảo mật. Ví dụ: mục nhập dành cho một tệp có thể bao gồm tên lỗ hổng bảo mật, nguồn thông tin về lỗ hổng, mục tiêu bị ảnh hưởng, v.v. Định dạng này cũng được thiết kế cho các triển khai đám mây có chứa tới hàng nghìn các hệ thống khác nhau.

Cả SPDX và CycloneDX đều tạo ra các cấu trúc phân cấp ở nhiều định dạng khác nhau bao gồm JSON và XML. Có nhiều công cụ cho từng tiêu chuẩn, vừa để tạo định dạng, vừa để quét các tệp đã tạo để tìm kiếm thông tin. Các trang kho lưu trữ kiểm soát phiên bản phổ biến sẽ cho phép các nhà phát triển tạo SBOM ở một trong các định dạng này chỉ với một cú nhấp chuột.

Phân tích Thành phần Phần mềm

Để biết một tổ chức đang sử dụng phần mềm nào, chúng ta cần phải đánh giá phần mềm của tổ chức đó. Đánh giá này sẽ bao gồm cả thông tin về nguồn gốc của phần mềm, các lỗ hổng mà phần mềm có, các giấy phép mà phần mềm sử dụng cũng như nhiều yếu tố khác giúp các cá nhân hoặc tổ chức quyết định xem có nên sử dụng phần mềm hay không. Nhiệm vụ này được gọi là *Phân tích Thành phần Phần mềm* (Software Composition Analysis - SCA); hiện này có rất nhiều công cụ có sẵn phục vụ việc quét một cách chi tiết các SBOM và chính bản thân phần mềm.

Một số công cụ sẽ trích xuất thông tin giấy phép để giúp tổ chức quyết định xem phần mềm có an toàn để đưa vào sản phẩm hay không, và liệu các thành phần có giấy phép tương thích với nhau hay không. Một số công cụ thậm chí còn so sánh các đoạn mã với các thư viện của nhiều dự án mã nguồn mở phổ biến để tìm hiểu xem mã có được lấy từ các dự án mã nguồn mở mà không ghi công tác giả một cách phù hợp hay không.

Việc chạy các công cụ này đặc biệt quan trọng trong quá trình sáp nhập hoặc mua lại. Công ty mua lại có thể sẽ thấy được rằng phần mềm mà họ muốn mua lại có giá trị không được như mong đợi vì nó có chứa các thành phần mã nguồn mở áp đặt nhiều yêu cầu can thiệp vào mục đích sử dụng dự kiến của nó trong công ty/ tổ chức mới.

Chính sách Chính thức và Tuân thủ

Các quy trình và kỹ thuật được mô tả trong bài học này nên được xây dựng bởi các nhà quản lý cùng với sự đóng góp của các nhà phát triển, luật sư và những cá nhân có kiến thức. Ở phần này,

chúng ta sẽ tìm hiểu một số quyết định về chính sách liên quan đến phần mềm mã nguồn mở mà các tổ chức cần đưa ra. Chúng ta sẽ kết thúc với một mô tả về *Văn phòng Chương trình Mã nguồn Mở* (OSPO) — một yếu tố hỗ trợ quan trọng và một nguồn thông tin dành cho các chính sách.

Nguyên tắc Quản lý việc Sử dụng và Đóng góp của Phần mềm Mã nguồn Mở

Các tổ chức có thể được hưởng lợi rất nhiều từ việc sử dụng và đóng góp vào phần mềm mã nguồn mở, nhưng họ cũng cần phải làm việc này theo một số cách nhất định để bảo vệ lợi ích của chính họ cũng như mang lại lợi ích cho các dự án mã nguồn mở. Các chính sách nên được xác định cho:

- Nơi sử dụng phần mềm mã nguồn mở (hoạt động điều hành, dự án nội bộ, sản phẩm hướng tới khách hàng, v.v.)
- Điều làm cho một dự án mã nguồn mở phù hợp với tổ chức: tính năng, hiệu suất, tính bảo mật, lộ trình mở rộng trong tương lai và khả năng tồn tại của cộng đồng
- Các bản quyét phân tích thành phần phần mềm và nơi lưu trữ tài liệu kết quả
- Phần thưởng cho các nhà phát triển đóng góp vào phần mềm mã nguồn mở (bao gồm cả việc tham gia vào các diễn đàn cộng đồng)
- Hướng dẫn đóng góp cho các dự án (bao gồm cách trở thành đại diện công khai của công ty)
- Yêu cầu về tài liệu để các nhà phát triển bên ngoài nhóm cốt lõi có thể hiểu được cách đóng góp và tương tác

OSPO có thể phối hợp xây dựng các chính sách này với các quy trình đào tạo để đảm bảo việc tuân thủ.

Thỏa thuận đối với các Nhà phát triển Đóng góp

Các dự án mã nguồn mở cần đảm bảo rằng các đóng góp đều hợp pháp. Ví dụ: mã không được lấy từ một số sản phẩm độc quyền hoặc một dự án mã nguồn mở có giấy phép không tương thích. Nhiều dự án mã nguồn mở sẽ yêu cầu các nhà phát triển cung cấp một tài liệu được gọi là *Thỏa thuận Cấp phép của Nhà phát triển Đóng góp* (Contributor License Agreement - CLA) để đảm bảo tính hợp pháp của những đóng góp từ họ.

Các nhà phát triển tại một tổ chức đóng góp vào các dự án này có thể yêu cầu luật sư của tổ chức xem xét và phê duyệt CLA. Do đó, các luật sư nên hiểu được tính thích đáng của các điều khoản CLA và sẵn sàng cho việc kiểm tra chúng.

CLA nhận về nhiều lời chỉ trích cả về cả tính phức tạp của chúng lẫn những lỗ hổng mà chúng để lại khiến các dự án sử dụng mã được đóng góp theo cách mà các nhà phát triển đóng góp không mong muốn.

Nhiều dự án sẽ yêu cầu nhà phát triển ký một tài liệu ngắn được gọi là *Chứng chỉ Nguồn gốc của Nhà phát triển* (Developer Certificate of Origin - DCO) thay vì CLA để chứng thực rằng nhà phát triển có quyền đóng góp mã. DCO (sẽ được thảo luận tới trong bài học khác) thường không được thông qua luật sư.

Một số dự án sẽ chỉ yêu cầu các nhà phát triển đóng góp ký chuyển bản quyền mã của họ cho dự án trong *Thỏa thuận Chuyển nhượng Bản quyền* (Copyright Assignment Agreement - CAA). Tuy nhiên, nhiều nhà phát triển đóng góp lại không thích cách làm này vì họ không thể sử dụng mã trong một số dự án độc quyền của riêng họ cũng như vì nó trao quá nhiều quyền cho dự án.

Văn phòng Chương trình Mã nguồn Mở (OSPO)

Các dự án mã nguồn mở thường kết hợp các yếu tố xã hội, kỹ thuật, pháp lý và tổ chức khác biệt so với bình thường. Hiện tại, kiến thức về các yếu tố này ở trong các tổ chức vẫn chưa được lan toả đến tất cả các nhà quản lý, nhà phát triển, luật sư và những người khác để họ đều có thể hiểu một cách sâu sắc về chúng. Vì vậy, việc tạo ra *Văn phòng Chương trình Mã nguồn Mở* (OSPO) như một tụ điểm trung tâm cho các hoạt động vận động, hoạch định chính sách, thực thi chính sách và đào tạo sẽ rất có ích cho các tổ chức.

Là một loại phòng ban đa năng, OSPO có thể thực hiện nhiều vai trò như:

Quảng bá Mã nguồn mở

OSPO có thể khiến các khái niệm đằng sau phong trào mã nguồn mở và các đề xuất sử dụng mã nguồn mở trong toàn bộ tổ chức trở nên phổ biến. Nó có thể nhắc nhở các nhà phát triển tìm kiếm các giải pháp mã nguồn mở để giải quyết các vấn đề và khuyến khích họ áp dụng các phần mềm phù hợp. Nó cũng có thể thúc giục các nhà quản lý dành thời gian cho các nhà phát triển tham gia vào các dự án mã nguồn mở và công nhận sự tham gia đó khi đánh giá các nhà phát triển để tăng lương và thăng chức.

Định nghĩa chính sách

OSPO có thể huy động các nhà quản lý để tạo chính sách và thiết lập các kho để lưu trữ cho chúng.

Thực thi chính sách

OSPO có thể nhắc nhở các nhà phát triển quét phần mềm và SBOM và tuân thủ các quy tắc của công ty về việc sử dụng mã nguồn mở. OSPO có thể lưu giữ tài liệu về các phần mềm đã áp dụng và các khía cạnh khác của việc sử dụng trong công ty.

Đào tạo

OSPO có thể giải thích cho các nhà phát triển cách đánh giá và tham gia vào các dự án mã nguồn mở, giải thích cho các luật sư về chi tiết của giấy phép và các cân nhắc pháp lý khác,

đồng thời giúp công ty hiểu được những thay đổi về mặt tổ chức và văn hóa có thể tạo điều kiện thuận lợi cho việc sử dụng phần mềm mã nguồn mở.

Có rất nhiều tài liệu và nguồn trực tuyến về việc tạo OSPO. Một tổ chức nhỏ có thể giao nhiệm vụ này cho một bên thứ ba có chuyên môn trong lĩnh vực.

Bài tập Hướng dẫn

1. Tại sao một nhà phát triển lại không nên lấy mã mà họ thích từ một dự án mã nguồn mở và đưa nó vào mã của mình mà không giữ giấy phép?

2. Nhà phát triển sẽ ký những loại tài liệu nào khi đóng góp cho một dự án mã nguồn mở?

3. Bạn tìm thấy một số phần mềm mã nguồn mở có thể tạo nên một cơ sở tuyệt vời cho trang web bán lẻ của mình. Bạn có thể lấy logo của mình chồng lên logo đã phổ biến sẵn của các phần mềm đó để tạo ra một hình ảnh bắt mắt cho trang web của mình hay không?

Bài tập Mở rộng

1. Những cân nhắc nào có thể sẽ khiến bạn phân nhánh một dự án mã nguồn mở để sử dụng cho riêng mình bất chấp những nhược điểm của việc phân nhánh?

2. Khi tìm thấy một số phần mềm mã nguồn mở đáp ứng được nhu cầu của mình, những lý do nào có thể khiến bạn từ chối chúng?

3. Bạn muốn sử dụng công cụ ghi nhật ký copyleft trong sản phẩm độc quyền của mình. Có cách nào để kết hợp chúng mà không cần phải cung cấp sản phẩm của mình theo giấy phép copyleft không?

Tóm tắt

Bài học này đã đề cập đến nhiều cân nhắc cần được lưu ý trước khi sử dụng phần mềm mã nguồn mở và tự do. Bài học này đã giải thích cách tuân thủ giấy phép, các rủi ro về pháp lý, danh tiếng và tài chính cũng như cách xác định các chính sách quan trọng và thực thi chúng trong tổ chức.

Đáp án Bài tập Hướng dẫn

1. Tại sao một nhà phát triển lại không nên lấy mã mà họ thích từ một dự án mã nguồn mở và đưa nó vào mã của mình mà không giữ giấy phép?

   Đây thường là một hành vi vi phạm giấy phép mã nguồn mở và cũng là hành vi đạo nhái và vi phạm bản quyền. Trên thực tế, một tổ chức có thể bị buộc phải tuân thủ theo giấy phép bởi hậu quả từ việc tổn hại danh tiếng hoặc thậm chí là việc mô hình kinh doanh bị huỷ hoại hoàn toàn khi mã copyleft bị trộn lẫn với mã độc quyền.

2. Nhà phát triển sẽ ký những loại tài liệu nào khi đóng góp cho một dự án mã nguồn mở?

   Thỏa thuận Cấp phép của Nhà phát triển đóng góp là một văn bản pháp lý cấp quyền cho tổ chức mã nguồn mở sử dụng mã. Chứng chỉ Nguồn gốc của Nhà phát triển là một văn bản ngắn hơn và đơn giản hơn thống nhất rằng nhà phát triển có quyền đóng góp mã. Thỏa thuận Chuyển nhượng Bản quyền sẽ cấp mọi quyền cho tổ chức tiếp nhận.

3. Bạn tìm thấy một số phần mềm mã nguồn mở có thể tạo nên một cơ sở tuyệt vời cho trang web bán lẻ của mình. Bạn có thể lấy logo của mình chồng lên logo đã phổ biến sẵn của các phần mềm đó để tạo ra một hình ảnh bắt mắt cho trang web của mình hay không?

   Nếu dự án đã đăng ký nhãn hiệu cho logo của mình, việc thay đổi mà bạn muốn thực hiện có thể sẽ xâm phạm đến nhãn hiệu. Ngay cả khi logo không được đăng ký nhãn hiệu, việc thay đổi của bạn cũng có thể bị coi là thiếu tôn trọng cũng như dễ gây nhầm lẫn.

Đáp án Bài tập Mở rộng

1. Những cân nhắc nào có thể sẽ khiến bạn phân nhánh một dự án mã nguồn mở để sử dụng cho riêng mình bất chấp những nhược điểm của việc phân nhánh?

   Nếu đã hài lòng với trạng thái hiện tại của mã, có thể bạn sẽ không cần phải theo dõi các thay đổi của dự án cốt lõi. Có thể bạn sẽ muốn tạo ra một sản phẩm có khác biệt đáng kể so với mục đích sử dụng mà dự án cốt lõi hướng đến, và do đó sẵn sàng tách rời dự án cốt lõi. Mã có thể đủ giá trị đối với kế hoạch kinh doanh của bạn để nhóm của bạn sẵn sàng chịu toàn bộ trách nhiệm cho việc phát triển và bảo trì.

2. Khi tìm thấy một số phần mềm mã nguồn mở đáp ứng được nhu cầu của mình, những lý do nào có thể khiến bạn từ chối chúng?

   Có thể là mã có chứa quá nhiều lỗi và lỗ hổng bảo mật, cộng đồng nhà phát triển của dự án đang không hoạt động tốt, bạn không thích hướng phát triển của mã hoặc giấy phép không tương thích với các mã khác trong sản phẩm của bạn.

3. Bạn muốn sử dụng công cụ ghi nhật ký copyleft trong sản phẩm độc quyền của mình. Có cách nào để kết hợp chúng mà không cần phải cung cấp sản phẩm của mình theo giấy phép copyleft không?

   Việc tích hợp mã cho công cụ vào mã của bạn có thể kích hoạt yêu cầu tương hỗ của copyleft, tùy thuộc vào giấy phép hiện hữu. Công cụ ghi nhật ký phải được tách biệt khỏi sản phẩm của bạn để sản phẩm của bạn không bị coi là một sản phẩm phái sinh. Ví dụ: việc chạy công cụ copyleft như một tiến trình riêng biệt và giao tiếp với nó thông qua việc truyền tin nhắn có thể được coi là khá an toàn.

Linux Professional Institute

Chủ đề 055: Quản lý Dự án

055.1 Mô hình Phát triển Phần mềm

Tham khảo các mục tiêu LPI

Open Source Essentials version 1.0, Exam 050, Objective 055.1

Khối lượng

3

Các lĩnh vực kiến thức chính

- Hiểu được sự liên quan và mục tiêu của quản lý dự án trong phát triển phần mềm
- Nắm được cơ bản về mô hình phát triển phần mềm thác nước
- Nắm được cơ bản về phát triển phần mềm linh hoạt (bao gồm Scrum và Kanban)
- Nắm được khái niệm về DevOps

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Các giai đoạn trong dự án thác nước (thiết kế phân tích yêu cầu, phân tích kinh doanh, thiết kế phần mềm, phát triển, kiểm thử, vận hành)
- Các vai trò trong dự án thác nước (quản lý dự án, nhà phân tích kinh doanh, kiến trúc sư phần mềm, nhà phát triển, kiểm thử viên)
- Tổ chức các dự án Scrum (phiên tăng tốc và lập kế hoạch tăng tốc, danh mục công việc và tăng tốc, họp scrum hàng ngày, đánh giá phiên tăng tốc và họp khái quát phiên tăng tốc)
- Các vai trò trong dự án Scrum (chủ sở hữu sản phẩm, nhà phát triển, chuyên gia scrum)
- Tổ chức các dự án Kanban (bảng Kanban)

Linux Professional Institute

Bài 1

Chứng chỉ:	Open Source Essentials
Phiên bản:	1.0
Chủ đề:	055 Quản lý Dự án
Mục tiêu:	055.1 Mô hình Phát triển Phần mềm
Bài học:	1 trên 1

Giới thiệu

Trong cuộc sống của chúng ta luôn có đầy rẫy các dự án. Một dự án có thể chỉ đơn giản là việc lên kế hoạch cho một đêm xem phim, một chuyến đi, một sự kiện gia đình, một tuần lễ dành cho trẻ em hoặc một kế hoạch cá nhân nhằm cải thiện các hoạt động vì sở thích. Không quan trọng bạn muốn thực hiện việc gì: bạn cần phải lên kế hoạch và hành động. Chúng ta thường sẽ tạo ra một số kịch bản với các kế hoạch B, kế hoạch C, v.v. Trong thế giới phát triển phần mềm cũng y như vậy.

Các Vai trò trong Phát triển Phần mềm

Để quản lý dự án thành công cần đến nhiều kỹ năng khác nhau; vì vậy, việc tìm những cá nhân có khả năng đảm nhiệm các vai trò được xác định là rất hữu ích. Các phần dưới đây sẽ trình bày một số vai trò thường thấy trong các dự án phần mềm.

Quản lý Dự án

Quản lý dự án là một nhiệm vụ cực kỳ phức tạp. Một số tình huống thoạt nhìn có vẻ rất dễ dàng — nhưng để khiến chúng đi đúng hướng thì cần có một bề dày kinh nghiệm cũng như một sự cẩn trọng đúng mực. Người thực hiện việc này chính là *quản lý dự án*.

Trách nhiệm của người quản lý dự án bao gồm việc đảm bảo dự án được hoàn thành đúng thời hạn trong phạm vi ngân sách và đạt được mức chất lượng cho phép. Ngay cả khi họ không viết một dòng mã nào, người quản lý dự án vẫn sẽ là người chịu trách nhiệm và trả lời về kết quả của nhóm. Họ sẽ phải thống nhất với khách hàng về thời hạn cũng như trao đổi với những người khác để đảm bảo mọi thứ đều sẽ tuân theo đúng trình tự.

Chuyên viên Phân tích Kinh doanh và Kỹ sư Phân tích Yêu cầu

Trong phát triển phần mềm không có chỗ quản lý vi mô — ít nhất là trong một môi trường làm việc lành mạnh. *Các chuyên viên phân tích kinh doanh* (Business analysts - BA) và *các kỹ sư phân tích yêu cầu* (Requirement Engineer - RE) — dù là trong nội bộ hoặc nằm ngoài nhóm dự án — là cánh tay phải đắc lực của các nhà quản lý dự án. Họ chịu trách nhiệm tạo ra nguồn tài liệu phong phú và rõ ràng về các yêu cầu. Họ cũng là cầu nối giữa khách hàng và các nhà phát triển. Khách hàng sẽ giao tiếp thông qua ngôn ngữ thông thường; BA/RE sẽ đảm bảo việc cung cấp tài liệu một cách rõ ràng và minh bạch để các nhà phát triển có thể thực hiện công việc của họ một cách hiệu quả.

Hãy tưởng tượng bạn được yêu cầu mang một chiếc "bánh lớn" đến một bữa tiệc. "Bánh lớn" có nghĩa là gì? Mười lát hay hai mươi lát? Có thể đó là bánh dùng cho một đám cưới lớn và nó phải có tới một trăm lát. BA/RE sẽ phải xác định các yêu cầu — ví dụ như về số lượng — một cách chính xác để các nhà phát triển biết được rằng một ứng dụng cần đáp ứng những nhu cầu như thế nào (ví dụ như một số lượng người dùng cụ thể).

Dù mới chỉ ở giai đoạn đầu của việc xác định các yêu cầu nhưng chúng ta đã có thể thấy được tầm quan trọng của việc giao tiếp rõ ràng. Giao tiếp là điều cần thiết đối với bất kỳ một công việc chung nào, nhưng có lẽ nó còn quan trọng hơn cả đối với một dự án mã nguồn mở vì các nhà phát triển hoạt động trên từng phần của dự án có thể có các xuất phát điểm rất khác nhau. Một số người có thể là sinh viên hoạt động trên danh nghĩa cộng tác viên, một số khác lại là phụ huynh, một người đã nghỉ hưu hoặc đang làm nhiều công việc khác nhau, v.v. Mặc dù vậy, tiến độ phải diễn ra một cách suôn sẻ trong môi trường này — vì vậy, giao tiếp không được kìm hãm tiến độ.

Nhà phát triển, Kiến trúc sư và Kiểm thử viên

Để triển khai các yêu cầu, một dự án phần mềm cần *các nhà phát triển* (developer) tạo ra sản phẩm dựa trên các tài liệu và quyết định thiết kế. Công việc của họ được đánh giá bởi các *kiến trúc sư* (architect) với lượng kiến thức chuyên môn và lĩnh vực sâu rộng nhất trong số những người tham gia dự án. Mỗi nhà phát triển — đặc biệt là các nhà phát triển cao cấp — đều sẽ chịu trách nhiệm về mã của riêng họ; tuy nhiên, các quyết định lớn sẽ được đưa ra hoặc phê duyệt bởi các kiến trúc sư.

Đôi khi, một số cá nhân sẽ được chỉ định cụ thể với danh nghĩa là *kiểm thử viên* (tester). Những

người này sẽ chịu trách nhiệm với mọi loại thử nghiệm, việc ghi chép kết quả và tạo phiếu lỗi.

Lên kế hoạch và Lên lịch

Chúng ta đã thảo luận xong về các vai trò cơ bản. Bây giờ, hãy cùng tìm hiểu về các giai đoạn của một dự án: lập kế hoạch, lên lịch, triển khai, bảo trì/kiểm tra và phân phối. Một số giai đoạn có thể sẽ không giống nhau tuỳ theo mô hình phát triển phần mềm, nhưng việc lập kế hoạch và lên lịch là các chủ đề chung mà chúng ta sẽ thảo luận trước khi đi sâu hơn vào các mô hình khác nhau.

Việc *lên kế hoạch* sẽ diễn ra sau khi BA/RE cung cấp danh sách các yêu cầu. Việc lên kế hoạch sẽ bao gồm một hoặc nhiều cuộc họp về những gì nhóm muốn đạt được, cách họ tính toán để hoàn thành và thời gian thực hiện. Thông thường, một số phân tích rủi ro cũng sẽ diễn ra trong các cuộc họp này.

Sau đó, các chuyên viên hoạch định sẽ phải *lên lịch* cho công việc và tạo ra *các mốc quan trọng*. Tại mỗi mốc quan trọng, nhóm sẽ biết rằng một phần lớn và quan trọng trong dự án đã được hoàn thành. Các thành phần hoặc tính năng dài hạn có thể sẽ mất nhiều tháng để thực hiện; vì vậy, các chuyên viên hoạch định phải tính đến các kỳ nghỉ, ngày lễ và — đặc biệt là trong mùa lạnh — một số khoảng thời gian nghỉ ốm để đảm bảo thời gian được phân phối một cách chính xác và tránh việc vội vã hoặc rối loạn khi thời hạn đến gần. Với một lịch trình tốt, các nhà phát triển sẽ cảm thấy thoải mái hơn và có thể cung cấp các giải pháp chất lượng cao đúng hạn mà không có sự cố.

Các Công cụ phổ biến

Hai công cụ có lợi cho tất cả các mô hình và việc quản lý dự án nói chung là *hệ thống kiểm soát phiên bản* (Version Control System - VCS) và nền tảng *quản lý vòng đời ứng dụng* (Application Lifecycle Management - ALM). Công cụ kiểm soát phiên bản sẽ được thảo luận sâu hơn trong một bài học khác; ở đây, chúng ta sẽ cùng tìm hiểu một chút về ALM.

ALM là một khuôn khổ toàn diện quản lý vòng đời của một ứng dụng phần mềm từ giai đoạn phát triển ban đầu cho đến giai đoạn bảo trì và cuối cùng là ngừng hoạt động. ALM tích hợp con người, tiến trình và công cụ để tăng cường sự cộng tác cũng như hiệu quả nhằm đảm bảo tất cả các giai đoạn của vòng đời phần mềm đều được phối hợp tốt và phù hợp với các mục tiêu kinh doanh. Cách tiếp cận này giúp duy trì chất lượng, hiệu suất và độ tin cậy của các ứng dụng trong suốt vòng đời của chúng.

Sau khi đã có một cái nhìn tổng quan về các vai trò và giai đoạn vòng đời phần mềm, hãy cùng tìm hiểu sâu hơn về các mô hình.

Mô hình Thác nước (Waterfall)

Đây là một trong những phương pháp luận sớm nhất và đơn giản nhất trong phát triển phần mềm. Chúng ta có thể hình dung nó như một chiếc cầu thang; một bậc mới cần phải được hoàn thành trước khi ta có thể tiến lên phía trước một bước. Tuy nhiên, mô hình này chỉ đi theo một hướng nên nó chỉ phù hợp với các dự án có yêu cầu rõ ràng và ít hoặc không có thay đổi dự kiến.

Tính đơn giản của mô hình này có thể là một lợi thế; tuy nhiên, phương pháp luận cứng nhắc lại có thể là một nhược điểm khi dự án cần đến sự linh hoạt và khả năng thích ứng. Và ngày nay, trong khi chúng ta đang làm việc trên dự án của mình thì mọi thứ xung quanh thường có xu hướng liên tục thay đổi.

Như đã giải thích ở các phần trước, để bắt đầu phát triển, một dự án cần chuẩn bị tốt các yêu cầu cùng với một kế hoạch hoàn chỉnh và một lịch trình thống nhất cho công việc với các mốc quan trọng. Trong mô hình thác nước, những yếu tố trên là kết quả của việc phân tích yêu cầu và phân tích kinh doanh.

Phân tích Yêu cầu

Trong giai đoạn đầu này, tất cả các yêu cầu của dự án sẽ được thu thập và ghi chép lại. Công việc này bao gồm các cuộc thảo luận sâu rộng giữa quản lý dự án và các bên liên quan để hiểu được nhu cầu và kỳ vọng của họ. Kết quả của quá trình này sẽ là một tài liệu đặc tả yêu cầu toàn diện phác thảo những gì hệ thống nên làm.

Phân tích kinh doanh

Các chuyên gia phân tích kinh doanh sẽ xem xét các yêu cầu từ góc độ kinh doanh để đảm bảo chúng phù hợp với các mục tiêu của tổ chức. Các BA sẽ thực hiện các nghiên cứu khả thi, đánh giá rủi ro và phân tích chi phí-lợi ích để xác định xem dự án có khả thi và đáng giá hay không. Những hiểu biết thu được sẽ được sử dụng để tinh chỉnh phạm vi và mục tiêu của dự án.

Thiết kế Phần mềm

Thiết kế phần mềm là bước mà kiến trúc sư tạo ra bản thông số kỹ thuật thiết kế chi tiết để các nhà phát triển tuân theo. Với thông số kỹ thuật này, nhóm có thể bắt đầu triển khai các yêu cầu — nói cách khác là bắt đầu giai đoạn tiếp theo: phát triển dự án.

Phát triển

Giai đoạn phát triển là nơi phép thuật xảy ra — chính là thời điểm mã được viết. Thực hiện theo tài liệu và các quyết định thiết kế một cách cần mẫn, các nhà phát triển sẽ xây dựng nên công

trình của họ. Sau khi các nhà phát triển rà soát và kiểm tra mã của họ cùng với các nhà phát triển hoặc kiến trúc sư khác, giai đoạn này có thể được coi là đã hoàn thành.

Kiểm thử

Theo sau giai đoạn Phát triển sẽ là giai đoạn kiểm thử/thử nghiệm do kiểm thử viên quản lý. Có nhiều loại kiểm thử nghiệm khác nhau (được mô tả trong bài học khác) như kiểm thử đơn vị, kiểm thử tích hợp, kiểm thử hệ thống và kiểm thử chấp nhận.

Mục tiêu của những cuộc kiểm thử này là phát hiện ra các vấn đề trước khi phát hành và cho phép các nhà phát triển khắc phục chúng kịp thời — để tránh việc phát hiện sau khi dự án được triển khai và công bố và khiến cho người dùng cảm thấy khó chịu và thất vọng vì lỗi.

Vận hành

Sau khi các cuộc kiểm thử được thông qua và lỗi đã được sửa, dự án có thể được phát hành — tức là triển khai vào môi trường sản xuất. Giai đoạn này có thể bao gồm việc cài đặt, cấu hình và đôi khi thậm chí là đào tạo người dùng. Sau khi dự án đã sẵn sàng sử dụng, nó sẽ tiến vào giai đoạn bảo trì nơi nhóm có thể xử lý các vấn đề từ người dùng và cung cấp các bản cập nhật nếu cần.

Đánh giá Mô hình Thác nước

Chúng ta có thể nhận thấy điều gì khi đọc về mô hình này? Liệu nó có hiệu quả hoặc có thiếu sót gì không? Hãy cùng tìm hiểu về các ưu và nhược điểm của mô hình này.

Mô hình thác nước có những lợi ích sau:

Cấu trúc rõ ràng: Quy trình tuyến tính dễ hiểu, dễ quản lý và dễ theo dõi.

Hồ sơ chi tiết

việc ghi lại một cách chi tiết và đầy đủ ở mỗi giai đoạn sẽ giúp nhóm hiểu được điều gì là cần thiết trong quá trình phát triển và bảo trì trong tương lai.

Khả năng dự đoán

Các mốc thời gian rõ ràng và được xác định sẽ giúp cung cấp một dòng thời gian và ngân sách chính xác.

Quản lý dự án đơn giản hơn

Nhờ luồng tuyến tính và trực tiếp, mô hình này tương đối dễ quản lý.

Mô hình thác nước có những nhược điểm sau:

Sự cứng nhắc

Tính không linh hoạt tuyến tính của mô hình khiến việc triển khai bất kỳ thay đổi nào sau khi giai đoạn yêu cầu hoàn tất đều sẽ trở nên khó khăn.

Kiểm thử muộn

Trong giai đoạn phát triển, việc kiểm thử diễn ra một cách tùy tiện hoặc bị thiếu; điều này có thể dẫn đến việc phát hiện muộn các vấn đề quan trọng.

Giả định về các yêu cầu hoàn hảo

Mô hình này đòi hỏi và giả định rằng mọi yêu cầu đều phải chính xác và rõ ràng; điều này có thể nói là rất không thực tế. Mô hình này không tạo ra điều kiện để kiểm thử trong quá trình phát triển để đảm bảo rằng các yêu cầu đều chính xác và được các nhà phát triển hiểu rõ.

Thiếu phản hồi

Khách hàng chỉ có thể đưa ra nhận xét về sản phẩm trong giai đoạn cuối cùng. Vì vậy, nếu có một (hoặc rất nhiều) khía cạnh nào đó không đáp ứng được kỳ vọng của họ, vấn đề sẽ chỉ xuất hiện ở giai đoạn cuối cùng.

Phát triển phần mềm Linh hoạt, Scrum và Kanban

Để khám phá tập hợp mô hình phát triển phần mềm hiện đại này, chúng ta hãy bắt đầu với từ *linh hoạt* (agile): nó có nghĩa là gì? Tính linh hoạt (agility) thể hiện khả năng phản ứng và di chuyển nhanh chóng, trong cả thế giới vật lý lẫn phi vật lý. Tính chất của mục tiêu này cũng tương tự trong phát triển phần mềm.

Phát triển phần mềm linh hoạt là một phương pháp luận xây dựng dựa trên sự phát triển lặp, tính linh hoạt và sự cộng tác. Nó tập trung vào việc cung cấp những cải tiến nhỏ trong khi luôn thu thập phản hồi và triển khai chúng trong quá trình phát triển. Phương pháp này cho phép nhà phát triển phản ứng, điều chỉnh và phản hồi một cách nhanh chóng, tiết kiệm thời gian và đảm bảo rằng dự án có thể đáp ứng được kỳ vọng của người dùng và khách hàng.

Phong trào linh hoạt được phát động vào năm 2001 thông qua *Tuyên ngôn về Phát triển phần mềm Linh hoạt* trực tuyến, trong đó đã liệt kê các nguyên tắc như sau:

> Chúng tôi đang khám phá những cách tốt hơn để phát triển phần mềm thông qua chính việc thực hiện và giúp đỡ những người khác thực hiện nó.
>
> Thông qua công việc này, chúng tôi đánh giá cao:
>
> - *Các cá nhân và việc tương tác* hơn là quy trình và công cụ
> - *Phần mềm hoạt động tốt* hơn là tài liệu toàn diện

> - *Sự hợp tác của khách hàng* hơn việc đàm phán hợp đồng
> - *Phản hồi với những thay đổi* hơn là tuân theo một kế hoạch
>
> Nghĩa là trong khi các mục bên phải cũng có giá trị, chúng tôi coi trọng các mục ở bên trái hơn.
>
> — Manifesto for Agile Software Development

Scrum

Scrum là một trong những khuôn khổ phổ biến nhất — cũng có thể là phổ biến nhất — trong phát triển phần mềm linh hoạt. Scrum được xây dựng dựa trên các khối sau:

> Nói một cách ngắn gọn, Scrum sẽ cần một Chuyên gia Scrum (Scrum Master) tạo ra một môi trường nơi:
>
> 1. Chủ sở hữu sản phẩm ra lệnh công việc cho một vấn đề phức tạp vào Danh mục Công việc.
> 2. Nhóm Scrum biến một phần công việc thành một Giá trị gia tăng trong một Phiên tăng tốc.
> 3. Nhóm Scrum và các bên liên quan kiểm tra kết quả và điều chỉnh cho phiên tăng tốc tiếp theo.
> 4. Lặp lại
>
> — The 2020 Scrum Guide

Nhưng ai là chuyên gia scrum và ai là chủ sở hữu sản phẩm? Danh mục Công việc và Phiên tăng tốc là gì? Hãy cùng tìm hiểu sâu hơn.

Phiên tăng tốc và Lên Kế hoạch Tăng tốc

Phiên tăng tốc (Sprint) là một giai đoạn phát triển thường kéo dài từ hai đến bốn tuần; trong đó, một số nhiệm vụ và/hoặc tính năng đã được thống nhất trước đó sẽ được hoàn thành. Để đạt được sự đồng thuận về các nhiệm vụ, *kế hoạch tăng tốc* sẽ được đưa vào. Quá trình này là nơi nhóm đưa ra quyết định về những nhiệm vụ nào có thể được hoàn thành trong phiên tăng tốc sắp tới.

Những lựa chọn này được dựa trên các ưu tiên do *chủ sở hữu sản phẩm* (Product Owner - PO) đặt ra. Chủ sở hữu sản phẩm là người kiểm soát dự án và thường cũng là người cung cấp kinh phí cho dự án.

Danh mục Công việc và Dang mục Công việc Phiên Tăng tốc

Như vừa giải thích, PO sẽ quản lý danh sách các ưu tiên, trong đó có tất cả các công việc được

mong muốn trong dự án. Trong Scrum, danh sách này được gọi là *Danh mục Công việc* (Product Backlog). Mỗi *Danh mục Công việc Phiên Tăng tốc* (Sprint Backlog) sẽ chứa các tác vụ đã được chọn cho phiên tăng tốc hiện tại từ *danh mục công việc*. Hồ sơ phiên tăng tốc sẽ chứa các yếu tố mà nhóm đã chọn trong quá trình lập kế hoạch tăng tốc.

Scrum hàng ngày hay Họp ngắn

Các cuộc họp *Scrum hàng ngày* (daily Scrum) hoặc *họp ngắn* (stand-up) là những cuộc họp ngắn và hiệu quả nơi các nhóm phát triển sẽ thảo luận về tiến độ của mình, nêu bật các vấn đề và trở ngại cũng như chia sẻ về kế hoạch trong ngày. Một cuộc họp scrum thường được tổ chức vào đầu ngày làm việc.

Giá trị gia tăng

Giá trị gia tăng (Increment) là mục tiêu cần đạt được trong một phiên tăng tốc. Mỗi phiên tăng tốc sẽ phải tạo ra một hoặc nhiều giá trị gia tăng. Tính chính xác của nhiệm vụ hoặc các tính năng đạt được bằng giá trị gia tăng phải được xác minh thông qua việc kiểm thử.

Đánh giá Phiên Tăng tốc

Đánh giá phiên tăng tốc là một cuộc họp để kiểm tra kết quả của phiên tăng tốc. Những cuộc họp này thường không chỉ nói về các bản demo — mục tiêu chính là để nhận được phản hồi dựa trên kết quả của nhóm Scrum. Nhằm mục đích đạt được mục tiêu của sản phẩm, các bên liên quan sẽ đưa ra ý kiến và đề xuất về các điều chỉnh trong tương lai. Kết quả của những cuộc họp này có thể là những thay đổi hoặc bổ sung vào danh mục công việc.

Họp khái quát về Phiên tăng tốc

Mục tiêu của việc *họp khái quát về Phiên tăng tốc* là nâng cao chất lượng và hiệu quả (không chỉ là cho sản phẩm hiện tại). Cuộc họp khái quát nên trình bày những căng thẳng tiềm ẩn trong nhóm phát triển, bất kỳ sự thiếu thông tin nào dẫn đến chậm trễ quá trình, các mối phụ thuộc bên ngoài cần được xử lý để giảm bớt gánh nặng cho nhóm phát triển, v.v. Nhóm phát triển cũng sẽ thảo luận về những gì đã và đang diễn ra tốt đẹp, những vấn đề họ gặp phải và những giải pháp nào đã (hoặc chưa) được đưa ra để giải quyết những vấn đề đó.

Kết quả của cuộc họp sẽ là một danh sách các vấn đề kèm theo các giải pháp khả thi được giao cho những người chịu trách nhiệm.

Chuyên gia Scrum

Tất cả các cuộc họp này đều do *chuyên gia Scrum* (Scrum master) chủ trì. Mỗi nhóm Scrum đều cần có một Chuyên gia Scrum. Họ sẽ chịu trách nhiệm tạo điều kiện cho các quy trình Scrum và

giúp nhóm phát triển tiếp tục tiến tới mục tiêu sản phẩm trong khi đối mặt những vấn đề trong các phiên tăng tốc. Chuyên gia Scrum không chỉ dẫn dắt các quy trình mà còn đóng vai trò là người hướng dẫn để đảm bảo việc giao tiếp hiệu quả trong nhóm.

Chủ sở hữu Sản phẩm

Chủ sở hữu sản phẩm (Product Owner - PO) sẽ chịu trách nhiệm tối đa hóa giá trị của sản phẩm do nhóm Scrum tạo ra. Vai trò này trong các tổ chức và nhóm khác nhau có thể sẽ khác nhau. Ngay cả khi phân công nhiệm vụ, chủ sở hữu sản phẩm vẫn sẽ chịu trách nhiệm về các kết quả.

Sự thành công của sản phẩm đòi hỏi sự tôn trọng của toàn bộ tổ chức đối với các quyết định của chủ sở hữu sản phẩm được phản ánh trong danh mục công việc và cuộc họp đánh giá giá trị gia tăng của phiên tăng tốc. Chủ sở hữu sản phẩm là người duy nhất đại diện cho nhu cầu của nhiều bên liên quan khác nhau trong danh mục công việc. Bất kỳ ai muốn thay đổi danh mục công việc cũng phải thuyết phục được chủ sở hữu sản phẩm. Chủ sở hữu sản phẩm sẽ xác định và ưu tiên tầm nhìn và hồ sơ của sản phẩm cũng như đảm bảo tính minh bạch, tính rõ ràng và dễ hiểu của danh mục công việc.

Chuyên gia Scrum và chủ sở hữu sản phẩm sẽ hợp tác để thúc đẩy thành công của dự án. Quan hệ đối tác của họ sẽ giúp nhóm phát triển cung cấp các tính năng có giá trị cao một cách hiệu quả.

Nhà phát triển

Các nhà phát triển sẽ triển khai các yêu cầu theo danh mục công việc phiên tăng tốc đã được thống nhất. Họ có thể làm việc độc lập hoặc cộng tác trong một số trường hợp như lập trình theo cặp hoặc đánh giá mã của các nhà phát triển khác.

Đánh giá Mô hình Scrum

Scrum rất được ưa chuộng trong các nhóm phần mềm nhưng nó cũng có cả ưu lẫn nhược điểm.

Mô hình Scrum có các điểm lợi về:

Tính linh hoạt
: Các quy trình linh hoạt và có tính lặp lại cho phép các thay đổi dựa trên phản hồi.

Sự tham gia của khách hàng
: Phản hồi thường xuyên từ các bên liên quan sẽ đảm bảo sản phẩm có thể phù hợp với nhu cầu của khách hàng.

Chất lượng được cải thiện
: Việc thử nghiệm và đánh giá thường xuyên sẽ giúp cải thiện chất lượng sản phẩm.

Hợp tác nhóm

Mô hình này nhấn mạnh vào tinh thần làm việc nhóm và việc giao tiếp thông qua các cuộc họp thường kỳ và hàng ngày.

Mô hình Scrum gặp phải những vấn đề sau:

Tính kỷ luật bắt buộc

Scrum đòi hỏi sự tuân thủ nghiêm ngặt đối với các thông lệ của nó; điều này có thể là một thách thức lớn.

Phạm vi phát triển

Có nguy cơ ngày càng có nhiều yêu cầu của khách hàng được thêm vào và làm chậm tiến độ phát hành nếu danh mục công việc không được quản lý tốt.

Nhầm lẫn về vai trò

Các vai trò tiêu chuẩn như Chuyên gia Scrum và chủ sở hữu sản phẩm dễ bị nhầm lẫn hoặc triển khai kém.

Cường độ nguồn lực

Các cuộc họp hàng ngày và các đợt đánh giá thường xuyên có thể sẽ tốn rất nhiều thời gian.

Kanban

Kanban là một khuôn khổ linh hoạt phổ biến khác nhấn mạnh vào việc trực quan hóa công việc, quản lý luồng và cải tiến quy trình. Một trong những điểm khác biệt lớn giữa Scrum và Kanban là Kanban không yêu cầu các phiên lặp có độ dài cố định. Do đó, nó khiến cho việc phân phối liên tục trở nên linh hoạt hơn, việc cân bằng các ưu tiên công việc khác trở nên dễ dàng hơn.

Hãy cùng xem các nhóm phát triển sẽ cần phải chuẩn bị những gì cho một dự án Kanban trong các tiểu mục sau đây.

Bảng Kanban

Bảng Kanban là một công cụ trực quan dùng để theo dõi luồng công việc qua các giai đoạn. Các cột chính và quan trọng nhất là “Cần làm” (To Do), “Đang tiến hành” (In Progress) và “Hoàn thành” (Done). Có thể sẽ có thêm nhiều cột nữa, nhưng trên đây là ba cột chính bắt buộc phải có trong bảng. Hình ảnh trực quan này sẽ giúp các nhóm phát triển nắm bắt được các nút thắt và xem trạng thái của các nhiệm vụ chỉ trong chớp mắt.

Giới hạn Công việc đang tiến hành

Giới hạn công việc đang tiến hành (WIP limit) sẽ giúp nhóm phát triển tránh đa nhiệm và cải thiện sự tập trung. Với giới hạn này, sẽ có một thời điểm mà nhóm phát triển không thể thêm bất kỳ một nhiệm vụ nào vào cột "Đang tiến hành" nữa. Bằng cách này, các nhà phát triển sẽ không bị quá tải bởi các nhiệm vụ và có thể tập trung tốt hơn vào các nhiệm vụ mà họ đang thực hiện.

Thành viên Nhóm phát triển

Các thành viên trong nhóm có trách nhiệm hoàn thành nhiệm vụ và thiết lập một luồng thông suốt thông qua hệ thống Kanban. Họ sẽ hợp tác với nhau để thống nhất ưu tiên công việc, nêu bật và giải quyết mọi vấn đề phát sinh.

Quản lý Kanban

Quản lý Kanban sẽ giám sát quy trình Kanban để đảm bảo việc nhóm phát triển tuân thủ các nguyên tắc và thông lệ của Kanban. Người quản lý này sẽ tạo điều kiện cho các cuộc họp, theo dõi quy trình làm việc và giúp giải quyết mọi vấn đề có khả năng phát sinh.

Đánh giá Mô hình Kanban

Cốt lõi của mô hình này rất đơn giản. Chúng ta hãy cùng đánh giá ưu nhược của nó.

Mô hình Kanban có các điểm lợi sau:

Quy trình làm việc trực quan
: Bảng Kanban cung cấp khả năng hiển thị rõ ràng về trạng thái và tiến độ của công việc.

Tính linh hoạt
: Do không có các phiên lặp cố định, mô hình này hỗ trợ tính năng phân phối liên tục với khả năng thích ứng đáng nể.

Giới hạn nhiệm vụ
: Giới hạn công việc đang tiến hành sẽ giúp các thành viên trong nhóm phát triển không bị quá tải và từ đó cải thiện sự tập trung và hiệu quả của họ.

Cải tiến liên tục
: Mô hình này khuyến khích đánh giá và cải tiến quy trình thường xuyên.

Mô hình Kanban gặp phải những vấn đề sau:

Thiếu khung thời gian

Nếu không có thời hạn cụ thể, việc quản lý thời gian sẽ trở nên khó khăn.

Ít cấu trúc hơn

Mô hình có thể bị thiếu cấu trúc mà một số nhóm cần để duy trì tính tổ chức.

Tính kỷ luật bắt buộc

Việc quản lý bảng và giới hạn công việc đang tiến hành hiệu quả đòi hỏi sự chú ý cao.

Tiềm ẩn việc đơn giản hóa một cách quá mức

Phần mô tả tác vụ có thể sẽ đơn giản hóa quá mức các dự án phức tạp nếu không được triển khai cẩn thận.

DevOps

DevOps là một phương pháp phát triển phần mềm tích hợp các nhóm phát triển (Dev) và nhóm vận hành (Ops) để tăng cường tính cộng tác, hiệu quả và tốc độ phân phối. Mục tiêu chính của DevOps là rút ngắn vòng đời phát triển phần mềm và cung cấp dịch vụ phân phối liên tục với chất lượng phần mềm cao. DevOps nhấn mạnh vào tự động hóa, tích hợp và phân phối liên tục (Continuous Integration/ Continuous Delivery - CI/CD) cũng như sự cộng tác chặt chẽ giữa các nhóm (theo truyền thống là) bị cô lập.

Tự động hóa rất quan trọng trong DevOps đối với các tác vụ như tích hợp mã, kiểm thử, triển khai và quản lý cơ sở hạ tầng. Tự động hóa các tác vụ lặp đi lặp lại sẽ giúp giảm lỗi, tăng tốc quy trình và cho phép các nhóm phát triển tập trung vào các công việc mang tính chiến lược hơn.

Việc giám sát và ghi nhật ký liên tục là điều cần thiết để duy trì tình trạng và hiệu suất của hệ thống. Các hoạt động thường thấy của DevOps gồm có thiết lập hệ thống giám sát và ghi nhật ký toàn diện để phát hiện sự cố, phân tích xu hướng và cải thiện độ tin cậy của hệ thống.

Đánh giá Mô hình DevOps

Mặc dù DevOps được khuyến khích sử dụng cho nhiều loại dự án phần mềm hiện đại nhưng chúng ta vẫn cần phải cân nhắc đến các ưu và nhược điểm của nó.

Mô hình DevOps có các điểm lợi sau:

Tốc độ và hiệu suất

Mô hình này đẩy nhanh chu kỳ phát triển và triển khai thông qua tự động hóa.

Cải thiện sự cộng tác

Mô hình này thu hẹp khoảng cách giữa nhóm phát triển và nhóm vận hành.

Phân phối liên tục

Mô hình này đảm bảo phần mềm luôn ở trạng thái có thể triển khai, từ đó dẫn đến việc phát hành thường xuyên hơn.

Độ tin cậy cao

Việc kiểm thử và giám sát tự động sẽ giúp tăng cường độ tin cậy và giảm lỗi.

Mô hình DevOps gặp phải những vấn đề sau:

Sự thay đổi về văn hóa

Mô hình này đòi hỏi một sự thay đổi đáng kể về khía cạnh văn hóa và sự đồng thuận trong toàn tổ chức.

Độ phức tạp

Việc quản lý các quy trình CI/CD và cơ sở hạ tầng tự động có thể trở nên rất phức tạp.

Rủi ro bảo mật

Việc triển khai liên tục có thể gây ra lỗ hổng bảo mật nếu không được quản lý cẩn thận.

Quá tải công cụ

DevOps có thể trở nên quá tải do có quá nhiều công cụ và công nghệ liên quan.

Bài tập Hướng dẫn

1. Tính linh hoạt có nghĩa là gì?

2. Theo Scrum Guide, Scrum được định nghĩa như thế nào?

3. Tại sao Scrum có thể hoạt động tốt hơn mô hình thác nước về mặt phản hồi của khách hàng?

4. Những khía cạnh tích cực của DevOps là gì?

Bài tập Mở rộng

1. Tại sao mô hình thác nước không phải là mô hình tốt nhất để sử dụng trong một môi trường thay đổi nhanh chóng?

Điều gì có thể xảy ra khi vai trò của chuyên gia Scrum được triển khai kém hiệu quả?

+

Tóm tắt

Tầm quan trọng của việc quản lý dự án trong phát triển phần mềm, đặc biệt là trong các dự án mã nguồn mở, nằm ở khả năng cung cấp cấu trúc, tăng cường giao tiếp, xác định vai trò, quản lý rủi ro và đảm bảo chất lượng. Các yếu tố này rất quan trọng đối với việc hoàn thành thành công các dự án và thúc đẩy một môi trường phát triển hợp tác và hiệu quả.

Trong bài học này, chúng ta đã tìm hiểu về mô hình thác nước, phương pháp luận linh hoạt và DevOps. Kiến thức về các mô hình này là điều cần thiết để hiểu về việc quản lý dự án trong phát triển phần mềm. Không có bất cứ một phương thức đúng đắn duy nhất nào cả — mỗi dự án đều sẽ khác nhau. Trong bài học này, chúng ta đã tìm hiểu về các vai trò, quy trình, ưu và nhược điểm của nhiều phương pháp tiếp cận khác nhau. Những kiến thức này có thể sẽ có ích trong việc giúp người đọc hiểu và đưa ra một lựa chọn mô hình đúng đắn cho các dự án trong tương lai.

Đáp án Bài tập Hướng dẫn

1. Tính linh hoạt có nghĩa là gì?

   Tính linh hoạt thể hiện khả năng phản ứng và di chuyển nhanh chóng ở trong cả thế giới vật lý lẫn phi vật lý.

2. Theo Scrum Guide, Scrum được định nghĩa như thế nào?
   - Chủ sở hữu sản phẩm sắp xếp công việc cho một vấn đề phức tạp vào Danh mục Công việc (Product Backlog).
   - Nhóm Scrum biến một phần công việc thành một Giá trị gia tăng trong một Phiên tăng tốc.
   - Nhóm Scrum và các bên liên quan kiểm tra kết quả và điều chỉnh cho phiên tăng tốc tiếp theo.
   - Lặp lại
3. Tại sao Scrum có thể hoạt động tốt hơn mô hình thác nước về mặt phản hồi của khách hàng?

   Do phản hồi được thu thập trong quá trình phát triển nên sản phẩm cuối cùng có thể đáp ứng được mong đợi của khách hàng một cách tốt hơn.

4. Những khía cạnh tích cực của DevOps là gì?

   Tốc độ và hiệu quả — Cải thiện sự cộng tác — Phân phối liên tục — Độ tin cậy cao

Đáp án Bài tập Mở rộng

1. Tại sao mô hình thác nước không phải là mô hình tốt nhất để sử dụng trong một môi trường thay đổi nhanh chóng?

   Mô hình thác nước chỉ thu thập phản hồi vào cuối quy trình phát triển. Do đó, bất kỳ yêu cầu nào bị các nhà phát triển hiểu sai đều chỉ có thể được sửa vào giai đoạn cuối. Nếu môi trường thay đổi quá nhanh, chúng ta không nên sử dụng mô hình này vì nó không có quy định về phản hồi vào giữa chu kỳ phát triển.

Điều gì có thể xảy ra khi vai trò của chuyên gia Scrum được triển khai kém hiệu quả?

+ Một Chuyên gia Scrum được triển khai kém có thể gây cản trở khả năng cung cấp các sản phẩm chất lượng cao một cách hiệu quả của nhóm phát triển và có thể làm giảm lợi ích của việc áp dụng mô hình Scrum. Nhóm phát triển có thể sẽ không có được các hướng dẫn phù hợp về các nguyên tắc và thông lệ thực hành của Scrum dẫn đến việc áp dụng khuôn khổ này sai cách hoặc không nhất quán.

+ Nếu không có một Chuyên gia Scrum hiệu quả để loại bỏ các trở ngại, chúng có thể làm chậm tiến độ và giảm năng suất của nhóm phát triển. Tình trạng giao tiếp không hiệu quả giữa các thành viên trong nhóm phát triển và các bên liên quan có thể xảy ra dẫn đến các hiểu lầm và kỳ vọng không thống nhất. Nhóm phát triển có thể bị giảm tinh thần và động lực do các vấn đề chưa được giải quyết, thiếu sự hỗ trợ và việc tạo điều kiện không hiệu quả từ các sự kiện Scrum. Sự kém hiệu quả có thể phát sinh từ các cuộc họp được tiến hành kém, thiếu tập trung và việc lập kế hoạch và đánh giá phiên tăng tốc không hiệu quả.

055.2 Quản lý Sản phẩm / Phát hành

Tham khảo các mục tiêu LPI

Open Source Essentials version 1.0, Exam 050, Objective 055.2

Khối lượng

2

Các lĩnh vực kiến thức chính

- Hiểu rõ về các loại bản phát hành phổ biến
- Hiểu rõ về cách phân loại phiên bản phần mềm và lý do dẫn đến các bản phát hành lớn hoặc nhỏ
- Hiểu về vòng đời của một sản phẩm phần mềm từ khi lập kế hoạch, phát triển và phát hành cho đến khi ngừng sản xuất
- Hiểu rõ về tài liệu dành cho các phiên bản của sản phẩm

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Phiên bản Alpha và beta
- Ứng cử viên phát hành
- Đông băng tính năng
- Bản phát hành chính và phụ
- Phân Phiên bản Ngữ nghĩa
- Lộ trình và các mốc quan trọng
- Nhật ký thay đổi
- Hỗ trợ dài hạn (LTS)
- Kết thúc vòng đời (EOL)

- Khả năng tương thích ngược

Linux Professional Institute

Bài 1

Chứng chỉ:	Open Source Essentials
Phiên bản:	1.0
Chủ đề:	055 Quản lý Dự án
Mục tiêu:	055.2 Quản lý Sản phẩm / Phát hành
Bài học:	1 trên 1

Giới thiệu

Theo thời gian, hầu hết các phần mềm đều sẽ phát triển theo nhiều cách khác nhau. Trên thực tế, đây là một trong những điều tuyệt vời về phần mềm: dễ dàng thay đổi và chỉ cần một phiên chuyển mạng để cập nhật cho tất cả những người dùng đang sử dụng sản phẩm. Chính vì vậy, các nhà phát triển luôn thêm vào các tính năng mới, sửa lỗi và lỗ hổng bảo mật, chuyển phần mềm sang phần cứng mới và thêm giao diện vào các chương trình và dịch vụ phổ biến. Khi phần mềm thay đổi, các *phiên bản* hoặc *bản sửa đổi* mới sẽ được phát hành.

Bài học này sẽ nói về các công việc hậu cần của việc lập kế hoạch và thực hiện thay đổi đối với phần mềm, cách các nhóm phát triển cùng lúc xử lý nhiều phiên bản, quy ước đặt tên cho các phiên bản và các khía cạnh tổ chức khác của quá trình phát triển — một phần của các hoạt động *quản lý dự án*. Các công việc hậu cần áp dụng đặc biệt cho việc cân chỉnh thời gian, đặt tên và kiểm soát các bản phát hành được gọi là *quản lý phát hành*.

Tính chất của các Bản phát hành

Các bản phát hành sẽ khác nhau về tính ổn định, khả năng tương thích ngược và hỗ trợ. Chúng ta sẽ cùng xem xét từng khái niệm đó trong các phần sau đây.

Bản phát hành ổn định và không ổn định

Ổn định là một tính chất trọng yếu mà người dùng mong đợi từ phần mềm. Câu hỏi đầu tiên mà đa số người dùng sẽ đặt ra về một bản phát hành mới là "Nó ổn định đến đâu?" Nói cách khác, nó có thể hoạt động tốt và đáng tin cậy hay sẽ bị sập, làm hỏng dữ liệu hoặc tạo ra các kết quả không chính xác?

Tính ổn định có thể được tính trên một phạm vi rộng, và nhiều người dùng vẫn sẽ vui vẻ sử dụng một phần mềm ngay cả khi nó có lỗi. Tuy nhiên, các nhóm phát triển có xu hướng giữ cho mọi thứ đơn giản và nói theo kiểu nhị phân về các bản phát hành *ổn định* và *không ổn định*.

Tính ổn định nói đến cả lỗi của phần mềm lẫn khả năng thay đổi của nó theo cách mà người ngoài có thể nhìn thấy được. Các nhà phát triển có thể coi bản phát hành của một thư viện phần mềm là không ổn định vì họ đã thay đổi các hàm mà lập trình viên gọi (tức là các lập trình viên có thể sẽ phải viết lại phần mềm của họ cho bản phát hành tiếp theo); hoặc phần mềm cũng có thể bị coi là không ổn định theo quan điểm của một người dùng vì các nhà phát triển đã xóa đi một tính năng nào đó.

Vậy có lý do nào cho việc phát hành một phiên bản không ổn định hay không? Có: chúng vẫn có giá trị vì chúng cho phép người dùng dùng thử các tính năng mới và kiểm thử lỗi ngay từ đầu dự án.

Hầu hết các dự án đều phát hành các bản sửa đổi rất sớm của phần mềm cho các khách hàng chủ chốt dùng thử; chúng được gọi là các bản phát hành *alpha*. Các bản phát hành này không nên được sử dụng cho các công việc thực sự — chúng chỉ nên dành cho việc thử nghiệm. Trên thực tế, kiểm thử viên nên chạy các bản phát hành alpha trên các máy tính đang không thực hiện bất kỳ một công việc quan trọng nào vì lỗi trong các bản phát hành đó thực sự có thể làm hỏng các dữ liệu được lưu trữ trên máy tính.

Khi phần mềm gần như đã được coi là ổn định, dự án thường sẽ phát hành một bản sửa đổi thử nghiệm khác được gọi là bản phát hành *beta*. Những bản phát hành này vẫn nên chỉ để thử nghiệm chứ không sử dụng cho các công việc sản xuất.

Đối với cả phiên bản alpha và beta, các nhà phát triển đều có một quy trình báo cáo lỗi và theo dõi tiến độ sửa lỗi.

Một số dự án sẽ có thêm một giai đoạn nữa nằm giữa bản beta và các bản phát hành ổn định khi các nhà phát triển đã sửa tất cả các lỗi mà họ có thể và cảm thấy rằng sản phẩm đã được hoàn thiện. Họ gọi phiên bản này là một *ứng cử viên phát hành* và sẽ trình diện nó cho khách hàng hoặc các bên liên quan chủ chốt để họ có thể lập kế hoạch sử dụng các tính năng mới.

Cuối cùng, một số dự án sẽ đưa vào các tính năng chưa quá ổn định vì có một số người dùng quan

trọng yêu cầu. Các nhà phát triển muốn người dùng dùng thử các tính năng này và cho biết họ có thích chúng hay không trước khi các nhà phát triển hoàn thiện giao diện và đầu tư công sức để ổn định các tính năng.

Khả năng tương thích ngược

Hầu hết các dự án đều hướng đến tính năng *tương thích ngược* — nghĩa là họ cố gắng để không phải xóa các tính năng hoặc chức năng đã tồn tại trong các phiên bản cũ. Khả năng tương thích có thể tồn tại ở nhiều cấp độ. Ví dụ: nếu các nhà phát triển duy trì khả năng tương thích ngược cho giao diện lập trình ứng dụng (API) — tức là họ đang hứa hẹn rằng mã nguồn cũ vẫn có thể tiếp tục hoạt động nhưng có thể sẽ cần phải biên dịch lại. Nếu các nhà cung cấp phần cứng hoặc nhà phát triển hệ điều hành duy trì khả năng tương thích ngược cho giao diện ứng dụng nhị phân (ABI) — họ đang hứa hẹn thêm rằng các tệp thực thi cũ vẫn có thể chạy trên các phiên bản phần cứng mới.

Chúng ta sẽ xem xét ở cácphần sau cách các dự án xử lý tình trạng *thiếu* khả năng tương thích ngược khi họ quyết định rằng giao diện cũ thực sự không phù hợp với các tính năng hoặc môi trường mới mà họ cần hỗ trợ.

Hỗ trợ và Kết thúc vòng đời (EOL)

Phần mềm sẽ ngày càng tốt và hoàn thiện hơn khi các nhà phát triển khám phá ra các vấn đề và khắc phục chúng. Theo thời gian, nhiều tính năng có thể sẽ ngừng hoạt động do những thay đổi trong môi trường của phần mềm. Ngoài ra, có nhiều lỗi mới bị ẩn trước đây cũng sẽ được phát hiện.

Các phiên bản mới thường kết hợp bảo mật và sửa lỗi với các tính năng mới. Có thể một số người dùng sẽ không cần các tính năng mới (trên thực tế, chúng ta có thể sẽ thích một số các tính năng cũ mà các nhà phát triển đã xóa để nhường chỗ cho các tính năng mới hơn), nhưng mọi người vẫn thường nâng cấp lên phiên bản mới để có được các bản sửa lỗi bảo mật giúp bảo vệ họ khỏi các cuộc tấn công độc hại.

Một số người thực sự vẫn sẽ sử dụng các phiên bản phần mềm cũ và từ chối nâng cấp. Điều này thường là do phiên bản mới làm hỏng thứ gì đó đang hoạt động tốt trong môi trường của họ. Khi các công ty tính phí nâng cấp, một số khách hàng có thể từ chối nâng cấp vì họ không muốn trả thêm tiền. Trên thực tế, việc trả tiền nâng cấp rất hiếm khi xảy ra trong mã nguồn mở.

Các nhà phát triển sẽ hỗ trợ người dùng các phiên bản cũ bằng cách sửa lỗi trong các phiên bản cũ mà không thêm các tính năng làm hỏng phần mềm nếu có thể. Tất nhiên, các nhà phát triển không thể làm điều này mãi mãi vì nó làm tiêu tốn thời gian và năng lượng dành cho công việc mới; sẽ đến lúc họ phải từ chối sửa chữa phiên bản cũ và yêu cầu người dùng nâng cấp hoặc tự

sửa lỗi.

Việc sửa lỗi và các lỗ hổng bảo mật được gọi là *hỗ trợ* cho phần mềm. Điều này khác với việc hỗ trợ do bộ phận trợ giúp và những người khác — những người hướng dẫn người dùng để hiểu về phần mềm — cung cấp. Về mặt quản lý phát hành, một *bản phát hành được hỗ trợ* là một bản phát hành mà các nhà phát triển đồng ý hỗ trợ sửa lỗi, trong khi một *bản phát hành không được hỗ trợ* là một bản phát hành mà nhà phát triển từ chối việc sửa chữa.

Hãy lưu ý rằng khái niệm "hỗ trợ" chủ yếu được áp dụng cho phần mềm do các công ty hoặc tổ chức lớn tạo ra. Các dự án mã nguồn mở nhỏ hơn và phụ thuộc nhiều vào các tình nguyện viên thường chỉ có thể hứa hẹn sẽ cố gắng nhất có thể để sửa lỗi và đưa ra các phiên bản mới. Họ không thực sự cảm thấy cần phải sửa các phiên bản cũ. Bởi mã là mã nguồn mở, người dùng không muốn nâng cấp có thể trả tiền cho một người nào đó để họ sửa một phiên bản cũ.

Khi công tác hỗ trợ được triển khai, các nhà phát triển sẽ công bố một lịch trình cho biết họ sẽ hỗ trợ từng phiên bản trong bao lâu. Ngày kết thúc hỗ trợ được gọi là thời hạn *kết thúc vòng đời* (End of Life - EOL) của phần mềm. Người dùng có thể giữ phiên bản cũ cho đến EOL mà vẫn có thể nhận được sự hỗ trợ về các lỗi; nhưng sau EOL, họ sẽ buộc phải nâng cấp hoặc chấp nhận rủi ro.

Các dự án lớn (chẳng hạn như bản phân phối Debian của GNU/Linux) cung cấp các phiên bản *hỗ trợ dài hạn* (Long Term Support - LTS). Điều này đơn giản có nghĩa là các nhà phát triển sẽ tiếp tục sửa lỗi trong một số năm nhất định. Những khách hàng coi trọng tính ổn định có thể sẽ cảm thấy lo lắng khi cài đặt các phiên bản phần mềm có nhiều thay đổi trong trường hợp các thay đổi đó làm hỏng các quy trình mà họ tin tưởng. Những khách hàng như vậy, đặc biệt là các tổ chức lớn, thường rất yêu thích tính bảo mật của các phiên bản LTS.

Các Phiên bản của Phần mềm: Bản Chính, Bản Phụ và Bản vá

Chúng ta đã thấy được rằng việc lập phiên bản rất phức tạp: một số phiên bản sẽ sửa lỗi, trong khi những phiên bản khác lại bổ sung các tính năng; thêm vào đó, các phiên bản cũng có thể sẽ khác nhau về mức độ ổn định. Các nhà phát triển luôn cố gắng nói rõ về mức độ thay đổi trong từng phiên bản của phần mềm thông qua nhãn của phần mềm. Các quy ước được hầu hết các dự án áp dụng để gắn nhãn phiên bản được gọi là *phân phiên bản ngữ nghĩa*.

Hãy lấy câu chuyện về thời kỳ ban đầu của nhân Linux làm ví dụ. Linus Torvalds đã dán nhãn bản phát hành ổn định đầu tiên là 1.0. Sau khi cải thiện hạt nhân, ông đã dán nhãn các phiên bản tiếp theo là 1.1, 1.2, v.v. Số 1 ban đầu là *phiên bản chính* và các số sau dấu chấm biểu thị *phiên bản phụ*. Chúng ta có thể coi như phiên bản 1.2 có nhiều tính năng hơn và cung cấp nhiều giá trị hơn so với phiên bản 1.1.

Tuy nhiên, cũng có vô số các bản phát hành nhỏ đôi khi chỉ để sửa một vài lỗi. Để chứng minh

rằng những thay đổi có tác động rất nhỏ đến việc sử dụng hạt nhân, Torvalds đã đưa vào một số thứ ba đại diện cho cái gọi là *bản vá* (patch). Từ đó, 1.0 đã được nâng cấp lên 1.0.1, sau đó là 1.0.2, v.v.

Số bản vá sẽ bắt đầu lại từ 0 khi một phiên bản phụ mới được phát hành và số phiên bản phụ sẽ bắt đầu lại từ 0 khi một phiên bản chính mới được phát hành.

Các nhà phát triển sẽ gộp các phiên bản lại với nhau bằng cách sử dụng một ký tự “x” để chỉ ra rằng họ đang nói về nhiều phiên bản (chẳng hạn như 1.x cho tất cả các phiên bản nằm trong phiên bản chính 1).

Vậy sự khác biệt giữa một thay đổi dẫn đến một phiên bản phụ mới và một thay đổi đủ lớn để dẫn đến một phiên bản chính mới là gì? Thông thường, phải sau nhiều năm trôi quá thì một phiên bản chính mới mới được phát hành và nó sẽ phải cho thấy một sự nâng cấp đáng kể.

Nhìn chung, các nhà phát triển sẽ cố gắng duy trì khả năng tương thích ngược khi các phiên bản thay đổi. Nếu không thể duy trì khả năng tương thích ngược, các nhà phát triển sẽ nâng cấp phiên bản chính.

Đôi khi chúng ta thấy số phiên bản nhỏ hơn không, thường là 0.9. Số không đứng đầu cho biết phiên bản phần mềm ban đầu không được ổn định và chưa sẵn sàng để được sử dụng trong sản xuất. Khả năng tương thích ngược không thể được đảm bảo cho đến khi các nhà phát triển phát hành phiên bản bắt đầu bằng số 1.

Các phiên bản alpha thường được chỉ định bằng cách thêm ký tự “a” hoặc “alpha” đằng sau số phát hành (chẳng hạn như 3.6alpha). Tương tự, phiên bản beta sẽ được thêm ký tự “b” hoặc “beta” sau số phát hành.

Vòng đời Sản phẩm của Phần mềm

Công việc của một nhà phát triển chưa bao giờ là dễ dàng; mọi người luôn muốn một cái gì đó từ họ. Có thể là người này muốn lỗi trong bố cục màn hình được sửa ngay lập tức, hoặc lại có một người khác cho rằng lỗi trong tên tệp cần phải được ưu tiên hơn. Người dùng luôn đòi hỏi các tính năng mới và khi các nhà phát triển khác nhau cùng lúc phát triển các tính năng khác nhau, họ sẽ gặp phải tình trạng những thay đổi của nhóm phát triển này làm hỏng những thay đổi của một nhóm phát triển khác.

Quản lý dự án và các nhiệm vụ phụ được gọi là quản lý phát hành sẽ giải quyết các vấn đề này. Thông thường, một thành viên dự án cấp cao sẽ chịu trách nhiệm cho công việc quản lý này.

Giống như con người, các phiên bản phần mềm cũng sẽ trải qua một vòng đời. Mỗi phiên bản đều sẽ bắt đầu bằng một cuộc thảo luận về các tính năng mới và các thay đổi khác cần thiết, trải qua

các giai đoạn phát triển và thử nghiệm để cuối cùng được triển khai một cách có kế hoạch.

Kế hoạch và Lộ trình

Các nhà phát triển luôn cố gắng đề ra thật sớm những thay đổi mà họ muốn thực hiện đối với một sản phẩm. Trong các công ty thương mại, họ sẽ trao đổi với những người thực hiện công việc tiếp thị để lọc và tóm tắt những gì khách hàng mong muốn. Các dự án mã nguồn mở phụ thuộc nhiều hơn vào các ý tưởng do các nhà phát triển và người dùng gửi; những ý tưởng này được lưu lại trong một cơ sở dữ liệu được gọi là *trình theo dõi vấn đề* (issue tracker). Nếu cần sửa lỗi hoặc muốn có một tính năng mới, chúng ta sẽ phải điền một bản *báo cáo vấn đề* (issue). Sau đó, các nhà phát triển sẽ ưu tiên các thay đổi và quyết định xem ai sẽ xử lý thay đổi nào.

Vậy những thay đổi sẽ được lựa chọn và ưu tiên như thế nào? Đây có thể là một quá trình tương đối lộn xộn. Những người quản lý dự án giỏi trong các dự án mã nguồn mở luôn khuyến khích sự đóng góp rộng rãi, song song với đó vẫn đảm bảo sẽ đưa ra các quyết định chính xác. Một số dự án thậm chí còn tổ chức các hội nghị nơi những người tham gia sẽ thảo luận về các ưu tiên.

Một *lộ trình* (roadmap) được công bố sẽ nêu ra các kế hoạch cải tiến và thay đổi. Nó có thể trải dài suốt nhiều bản phát hành và nhiều năm trong tương lai. Mỗi bước sẽ được gọi là một *mốc thời gian*; mốc thời gian có thể có hoặc không liên quan đến ngày đích.

Lịch trình Phát hành

Có hai cách cơ bản để lập lịch trình phát hành: theo thời gian và theo tính năng. Một dự án có thể hẹn lịch phát hành theo các khoảng thời gian đều đặn — ví dụ như sáu tháng một lần — và phát hành bất kỳ những gì đã được hoàn thành tại thời điểm đó. Theo cách thứ hai, dự án có thể cam kết một số tính năng nhất định sẽ xuất hiện trong bản phát hành và để các nhà phát triển hoàn thành các tính năng đó trong thời gian mà họ cần.

Khi thời điểm phát hành đến gần, quản lý phát hành sẽ xác định ngày phát hành các bản alpha, beta và bản ổn định. Các thành viên trong nhóm phát triển luôn thường xuyên xem xét các báo cáo lỗi và cố gắng sắp xếp kế hoạch công việc của họ để đạt được các mốc thời gian đó.

Để hoàn thành một bản phát hành, một dự án phải ngừng tiếp nhận ý tưởng về các tính năng mới và tập trung vào việc làm cho các tính năng hiện có hoạt động tốt. Thời điểm này được gọi là phiên *đóng băng tính năng* (feature freeze).

Tài liệu dành cho các Phiên bản Sản phẩm

Lộ trình, như đã đề cập tới ở trên, sẽ giải thích các tính năng mà nhà phát triển dự định đưa vào mỗi bản phát hành. Bản phát hành sẽ được đi kèm với một danh sách các thay đổi được gọi là *nhật*

ký thay đổi (changelog). Nhìn chung, nhật ký thay đổi sẽ liệt kê các tính năng mới, các thay đổi đối với các tính năng hiện có, các tính năng đã bị xóa, các tính năng mà nhà phát triển dự định sẽ xóa trong tương lai (được gọi là các tính năng *lỗi thời*) và các bản sửa lỗi.

Do đó, nhật ký thay đổi có thể sẽ khá dài và chi tiết. Người dùng cần đặc biệt chú ý đến các tính năng đã bị xóa hoặc không còn được sử dụng nữa vì có thể họ sẽ phải thay đổi chương trình hoặc cách sử dụng sản phẩm của mình. Đương nhiên là các nhà phát triển sẽ cố gắng chỉ xóa những tính năng không ai cần tới nữa.

Mỗi bản phát hành đều sẽ thay đổi tài liệu sản phẩm để phù hợp với những thay đổi trong sản phẩm. Nhiệm vụ này có thể sẽ tốn khá nhiều thời gian và các nhà phát triển rất dễ bỏ lỡ một thay đổi nào đó hoặc chậm trễ trong quá trình tạo tài liệu.

Bài tập Hướng dẫn

1. Điều gì có thể giúp chúng ta phân biệt giữa một bản phát hành ổn định và một bản phát hành không ổn định?

2. Chúng ta có nên kỳ vọng rằng sẽ có sự thay đổi về tính năng giữa bản phát hành 2.6.14 và bản phát hành 2.6.15 không?

3. Chúng ta có nên kỳ vọng rằng sẽ có sự thay đổi về tính năng giữa bản phát hành 2.6.0beta và bản phát hành 2.6.0 không?

4. Tại sao chúng ta lại có thể biết rằng bản phát hành 1.0 sẽ không tương thích ngược với bản phát hành 0.9?

5. Nếu phát hiện ra một lỗi bảo mật trong một phiên bản sau một phiên đóng băng tính năng và trước thời điểm phát hành, bạn có thể sửa lỗi đó được không?

Bài tập Mở rộng

1. Giả sử bạn muốn tiếp tục sử dụng một phiên bản phần mềm mã nguồn mở sau EOL vì nó có một tính năng mà bạn cần. Bạn có thể làm gì để giữ cho nó vẫn có thể tiếp tục sử dụng được?

2. Một số tiêu chí cho phép một bản sửa lỗi hoặc một yêu cầu tính năng được chọn trong số những yêu cầu khác là gì?

Tóm tắt

Bài học này đã mô tả các đặc điểm chính để phân biệt các bản phát hành: tính ổn định, khả năng tương thích ngược và khả năng hỗ trợ. Chúng ta đã thảo luận về ý nghĩa của tên và số phiên bản và các khía cạnh chính của nhiệm vụ quản lý bản phát hành (bao gồm cả tài liệu).

Đáp án Bài tập Hướng dẫn

1. Điều gì có thể giúp chúng ta phân biệt giữa một bản phát hành ổn định và một bản phát hành không ổn định?

   Có hai cách để xác định các bản phát hành được coi là ổn định: một là chúng hoạt động tốt mà không bị sập hoặc tạo ra kết quả không chính xác, và hai là giao diện được trình bày cho người dùng hoặc lập trình viên được dự kiến có thể tương thích ngược với các phiên bản trước đó.

2. Chúng ta có nên kỳ vọng rằng sẽ có sự thay đổi về tính năng giữa bản phát hành 2.6.14 và bản phát hành 2.6.15 không?

   Không. Số thứ ba trong quy tắc phân phiên bản ngữ nghĩa đã chỉ ra rằng nó là một bản vá được tạo ra để sửa lỗi hoặc thực hiện một số tác vụ nhỏ khác như tái định dạng. Một thay đổi về tính năng sẽ dẫn đến một bản phát hành phụ hoặc chính.

3. Chúng ta có nên kỳ vọng rằng sẽ có sự thay đổi về tính năng giữa bản phát hành 2.6.0beta và bản phát hành 2.6.0 không?

   Không. Phiên bản beta là một phiên bản thử nghiệm có chứa tất cả các tính năng sẽ có trong bản phát hành cuối cùng.

4. Tại sao chúng ta lại có thể biết rằng bản phát hành 1.0 sẽ không tương thích ngược với bản phát hành 0.9?

   Số phiên bản 0.9 đã cảnh báo rõ ràng với các người dùng tiềm năng rằng phần mềm vẫn đang được thiết kế và rất có thể sẽ có giao diện khác khi được hoàn thiện ổn định thành phiên bản 1.0.

5. Nếu phát hiện ra một lỗi bảo mật trong một phiên bản sau một phiên đóng băng tính năng và trước thời điểm phát hành, bạn có thể sửa lỗi đó được không?

   Chắc chắn là có thể. Khoảng thời gian giữa một phiên đóng băng tính năng và lúc phát hành được coi là khoảng thời gian để phát hiện và sửa lỗi (bao gồm cả lỗi bảo mật).

Đáp án Bài tập Mở rộng

1. Giả sử bạn muốn tiếp tục sử dụng một phiên bản phần mềm mã nguồn mở sau EOL vì nó có một tính năng mà bạn cần. Bạn có thể làm gì để giữ cho nó vẫn có thể tiếp tục sử dụng được?

   Sau EOL, các nhà phát triển dự án không có còn bất cứ cam kết nào liên quan đến vấn đề sửa lỗi (bao gồm cả các lỗi bảo mật). Do đó, bạn nên theo dõi trình theo dõi lỗi và danh sách gửi thư của dự án một cách thường xuyên để biết được lỗi nào vừa xuất hiện. Vì mã đã có sẵn nên bạn có thể và nên sửa lỗi có trong phiên bản của mình. Về lý thuyết, bạn thậm chí có thể kết hợp các tính năng mới vào phiên bản của mình. Điều này thực sự sẽ biến phiên bản của bạn thành một nhánh của bản gốc.

2. Một số tiêu chí cho phép một bản sửa lỗi hoặc một yêu cầu tính năng được chọn trong số những yêu cầu khác là gì?

   Đối với các bản sửa lỗi, tiêu chí sẽ bao gồm mức độ nghiêm trọng (được gán cho lỗi sau khi lỗi đó được đưa vào trình theo dõi lỗi) và số lượng người dùng bị ảnh hưởng. Đối với một tính năng, tiêu chí sẽ bao gồm số lượng người dùng mong muốn tính năng đó, mức độ khó khi mã hóa và tác động tiềm tàng của tính năng đó đối với các phần khác của chương trình.

055.3 Quản lý Cộng đồng

Tham khảo các mục tiêu LPI

Open Source Essentials version 1.0, Exam 050, Objective 055.3

Khối lượng

2

Các lĩnh vực kiến thức chính

- Hiểu rõ về các vai trò trong một dự án nguồn mở
- Hiểu rõ về các nhiệm vụ chung trong một dự án mã nguồn mở
- Hiểu rõ về các loại đóng góp khác nhau cho các dự án mã nguồn mở
- Hiểu rõ về các nhóm đóng góp khác nhau cho các dự án nguồn mở
- Hiểu rõ về vai trò của các tổ chức trong việc duy trì các dự án mã nguồn mở
- Hiểu rõ về việc chuyển giao quyền từ cá nhân sang tổ chức duy trì một dự án
- Hiểu rõ về các quy tắc và chính sách trong các dự án mã nguồn mở
- Hiểu rõ về việc ghi công và minh bạch trong các đóng góp
- Hiểu rõ về các khía cạnh của sự đa dạng, công bằng, hoà nhập và không phân biệt đối xử

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Phát triển phần mềm
- Tài liệu
- Thiết kế và sản phẩm nghệ thuật
- Hỗ trợ người dùng
- Nhà phát triển

- Quản lý phát hành
- Người dùng
- Trưởng dự án và nhà độc tài tốt bụng
- Cá nhân và tập đoàn
- Người làm vì đam mê và chuyên gia
- Thành viên cốt cán và người đóng góp không thường xuyên
- Đóng góp mã và tài liệu
- Báo cáo lỗi
- Nhánh
- Quỹ và nhà tài trợ
- Thỏa thuận đóng góp
- Chứng chỉ Nguồn gốc của Nhà phát triển (DCO)
- Hướng dẫn viết mã
- Quy tắc ứng xử

Linux Professional Institute

Bài 1

Chứng chỉ:	Open Source Essentials
Phiên bản:	1.0
Chủ đề:	055 Quản lý Dự án
Mục tiêu:	055.3 Quản lý Cộng đồng
Bài học:	1 trên 1

Giới thiệu

Một trong những lý do chủ chốt của việc tạo ra dự án mã nguồn mở là để thu hút sự đóng góp từ nhiều cá nhân khác nhau. Mã nguồn mở sẽ giúp nhà phát triển dễ dàng đáp ứng các nhu cầu và sở thích khác nhau của người dùng dự án cũng như hưởng lợi từ các kỹ năng đa dạng của họ. Do đó, một cộng đồng đa dạng chính là một yếu tố cốt lõi đối với sự thành công của dự án. Cộng đồng là nơi các lực lượng sáng tạo quy tụ lại để giúp cho dự án của chúng ta thành công.

Bài học này sẽ giải thích về các thành phần cơ bản trong cộng đồng phần mềm mã nguồn mở và tự do cũng như cách mọi người trong cộng đồng làm việc với nhau. Tất nhiên, vì mỗi cộng đồng đều sẽ bao gồm nhiều cá nhân khác nhau và mỗi dự án cũng đều có các mục tiêu và yêu cầu khác nhau nên mỗi một cộng đồng đều là độc nhất.

Các Vai trò trong một Dự án Mã nguồn mở

Đầu ra chính của hầu hết các dự án mã nguồn mở đều sẽ là phần mềm; vì vậy, các dự án này ít nhất cần phải có các lập trình viên, nhà thiết kế hoặc kiến trúc sư phần mềm, kiểm thử viên, người quản lý phát hành và các chuyên gia khác trong phát triển mã. Tài liệu thường cũng là một phần của các dự án này; vì vậy, cộng đồng của dự án cũng sẽ cần phải có các tác giả, biên tập viên

và người đánh giá.

Một số dự án có thể sẽ không tạo ra mã; thay vào đó, chúng lại taọ ra các "sản phẩm có thể chuyển giao được" khác như sách, dự án nghệ thuật hoặc âm nhạc hay báo cáo chính sách. Wikipedia là một ví dụ nổi tiếng về sự hợp tác trong một dự án mã nguồn mở tập trung vào tài liệu văn bản và phương tiện truyền thông. Các dự án này cần có các chuyên gia về thiết kế, sản xuất và phân phối các sản phẩm có thể chuyển giao được.

Các cộng đồng cũng sẽ được hưởng lợi từ nhiều kỹ năng phổ quát khác như tiếp thị và truyền bá, quản lý, tư vấn pháp lý, hỗ trợ về nghệ thuật để tạo ra logo hoặc sơ đồ trong tài liệu cũng như các kỹ năng duy trì trang web của dự án hay các công cụ truyền thông khác. Các dự án mã nguồn mở có thể sẽ tổ chức các cuộc họp mặt thực tế hay thậm chí là các hội nghị nơi họ sẽ cần tới các chuyên viên tổ chức sự kiện để biến những hội nghị đó thành hiện thực.

Các phần tiếp theo sẽ cung cấp cho chúng ta một cái nhìn chung về các loại kỹ năng mà một cộng đồng tìm kiếm. Những vai trò chung này có thể được chia nhỏ hơn thành nhiều nhiệm vụ tập trung khác nhau. Một ví dụ đơn giản về các nhiệm vụ như vậy là khi một tác giả biên tập lại tác phẩm của các tác giả khác.

Đối với các lập trình viên, phạm vi kinh nghiệm là một yếu tố quan trọng. Một dự án lành mạnh sẽ luôn cần tới một số lập trình viên *kỳ cựu* (hoặc *lập trình viên_cao cấp) hiểu rõ về mã và tiêu chuẩn mã hóa của dự án cũng như những lập trình viên _non trẻ* có khả năng cung cấp các ý tưởng mới và trợ giúp các tác vụ đơn giản như sửa lỗi. Các lập trình viên non trẻ luôn được kỳ vọng sẽ phát triển lên thành thế hệ kỳ cựu tiếp theo.

Việc đảm bảo chất lượng của mã cần tới một sự kiểm soát nghiêm ngặt về những gì được đưa vào kho lưu trữ trung tâm chia sẻ với người dùng. Do đó, một số lập trình viên cao cấp sẽ được chỉ định trạng thái *người được uỷ thác* (committer). Họ sẽ chịu trách nhiệm kiểm tra các đóng góp về chất lượng, tính hữu dụng và sự tuân thủ các tiêu chuẩn về mã hóa. Họ có đặc quyền chấp nhận mã được đề xuất vào kho lưu trữ. Các nhà phát triển đóng góp khác sẽ gửi mã cho người được uỷ thác để họ xét duyệt.

Rất nhiều người được uỷ thác và những lập trình viên kỳ cựu khác cũng sẽ *đào tạo* các nhà phát triển đóng góp mới để hướng dẫn họ về các tiêu chuẩn và thông lệ thực hành cần thiết để mã của họ được chấp nhận.

Một số người được uỷ thác không biết cách trân trọng món quà mà các nhà phát triển đóng góp mang lại. Nếu đóng góp không đạt về chất lượng hoặc tiêu chuẩn mã, người được uỷ thác hoàn toàn có thể từ chối đóng góp đó. Tuy nhiên, việc từ chối đóng góp có thể sẽ khiến các nhà phát triển đóng góp tiềm năng nản lòng và có thể khiến cộng đồng mất đi một cơ hội mang tính giáo dục có giá trị. Hãy nhớ rằng một người được uỷ thác tốt cũng sẽ đóng vai trò như một nhà cố vấn. Vì vậy, họ sẽ đưa ra gợi ý cho các nhà phát triển đóng góp để giúp mã đạt mức tiêu chuẩn cũng

như để đồng hành với họ theo thời gian. Nếu đóng góp thực sự không thể sử dụng được, ít nhất nhà phát triển đóng góp cũng nên được cảm ơn và khuyến khích đóng góp trên các phương diện khác cho dự án. Điều quan trọng nhất là giúp tất cả mọi người khám phá và tinh chỉnh các kỹ năng của chính họ.

Khi một công ty bắt đầu một dự án mã nguồn mở, đôi khi công ty sẽ chỉ định người được uỷ thác từ chính các nhân viên của mình để đảm bảo rằng công ty có thể kiểm soát được hướng đi và chất lượng của dự án. Tuy nhiên, các công ty cũng thường cho phép những người không phải nhân viên nhưng thể hiện được kỹ năng và sự tận tâm trong việc trở thành người được uỷ thác.

Trưởng dự án (project lead) sẽ đưa ra các quyết định quan trọng (như những tính năng mới nào sẽ được hỗ trợ). Những trưởng dự án này thường cũng sẽ phải đưa ra các quyết định không liên quan đến mã hóa như cách quảng bá dự án, giải quyết các bất đồng lớn và nhận tài trợ. Các trưởng dự án và người được uỷ thác sẽ làm việc chặt chẽ với *các nhà quản lý phát hành* để quyết định xem khi nào cơ sở mã được coi là ổn định và sẵn sàng được chia sẻ với người dùng.

Một số dự án chỉ có một trưởng dự án duy nhất thường được gọi là *nhà độc tài tốt bụng* (benevolent dictator). Đây thường là người đã bắt đầu dự án (Linus Torvalds là một ví dụ điển hình trong trường hợp của Linux) và cũng là người có thẩm quyền hay thậm chí là có sức hút lớn. Tuy nhiên, hầu hết các dự án đều chọn thành lập một ủy ban nhỏ các trưởng dự án thay vì một nhà độc tài tốt bụng duy nhất. Ngay cả dự án Linux cũng đã chuyển sang cách tiếp cận này.

Các dự án cũng có thể yêu cầu các nhà phát triển đóng góp cụ thể cung cấp hỗ trợ người dùng trên diễn đàn của dự án. Các dự án lành mạnh nên có nhiều người dùng hiểu biết hỗ trợ lẫn nhau.

Chúng ta sẽ kết thúc phần này bằng một vai trò rất quan trọng: *người quản lý cộng đồng* (community manager). Đây là người chịu trách nhiệm duy trì giao tiếp cởi mở và mang tính xây dựng để đảm bảo cộng đồng tiến triển theo đúng mục tiêu của nó. Người quản lý cộng đồng phải biết cách đưa các nhà phát triển đóng góp lên cùng chiến tuyến và khuyến khích họ, giải quyết các tranh cãi, bảo vệ các thành viên cộng đồng khỏi bị lạm dụng và thực hiện các nhiệm vụ khác để duy trì sức khỏe của cộng đồng.

Các Nhiệm vụ phổ biến trong Dự án Mã nguồn Mở

Trên nhiều phương diện, các dự án mã nguồn mở sẽ bao gồm các nhiệm vụ tương tự như các dự án truyền thống được thực hiện trong một tổ chức duy nhất (ví dụ như việc mọi mã đều yêu cầu kiểm thử). Tuy nhiên, theo một số cách, các dự án mã nguồn mở cũng sẽ khác với các dự án độc quyền nơi chỉ một nhóm người hạn chế mới được phép thay đổi mã và mã nguồn thường bị ẩn khỏi công chúng.

Ví dụ: các công ty thường sẽ chỉ định nhân viên đảm bảo chất lượng (Quality Assurance - QA) đã

qua đào tạo để kiểm thử mã. Tuy nhiên, nhiều dự án mã nguồn mở sẽ chỉ yêu cầu người dùng thử nghiệm mỗi bản phát hành (các dự án độc quyền cũng sẽ yêu cầu người dùng của họ tham gia vào quá trình kiểm thử bên cạnh QA).

Một ví dụ khác về sự khác biệt: một dự án độc quyền thường sẽ giao cho các nhà phát triển được trả lương những nhiệm vụ cụ thể và cho họ biết cần dành bao nhiêu thời gian mỗi tuần cho các nhiệm vụ đó. Một số dự án mã nguồn mở lại được hưởng lợi từ những nhà phát triển đóng góp được trả lương bởi người sử dụng lao động của họ, hoặc đôi khi thậm chí được chính dự án tuyển dụng, và sẽ được giao nhiệm vụ theo cùng cách với các nhà phát triển độc quyền. Trong hầu hết các dự án lành mạnh, rất nhiều đóng góp lại đến từ các tình nguyện viên chỉ thực hiện các nhiệm vụ khi thời gian của họ cho phép. Vì dự án không dựa vào các tình nguyện viên để hoàn thành đúng tiến độ nên bản phát hành mã nguồn mở có thể bị trì hoãn cho đến khi các nhà phát triển cảm thấy các tính năng đã sẵn sàng hoặc có thể được phát hành vào những thời điểm cố định với bất kỳ tính năng nào đã sẵn sàng.

Giao tiếp rất quan trọng đối với cả các dự án độc quyền lẫn các dự án mã nguồn mở nhưng lại thường được tiến hành theo các cách khác nhau. Các dự án độc quyền thường tập hợp một nhóm trong văn phòng và tổ chức các cuộc họp ngắn tuân thủ lịch trình thường xuyên theo một phương pháp phát triển như SCRUM. Vì các dự án mã nguồn mở được phân bổ theo địa lý nên các phương pháp như vậy rất hiếm. Thay vào đó, giao tiếp sẽ diễn ra trực tuyến và không đồng bộ.

Tóm lại, các dự án phần mềm mã nguồn mở và tự do thường đạt được các mục tiêu tương tự như các dự án độc quyền nhưng theo những cách khác.

Báo cáo lỗi là một phần quan trọng của quá trình phát triển phần mềm với một tập hợp các vai trò riêng do chính nó tạo ra. Sẽ phải có người theo dõi cơ sở dữ liệu lỗi và chỉ định mức độ ưu tiên. Nếu một lỗi được tính là nghiêm trọng (và đặc biệt là nếu lỗi có thể làm suy yếu tính bảo mật của phần mềm), nó nên chỉ định cho một chuyên viên bảo trì có năng lực để sửa chữa.

Trưởng dự án và quản lý cộng đồng nên xem xét sự tham gia của các thành viên vào dự án và quyết định xem có thiếu sót gì không. Liệu có phải có quá ít người sẵn sàng kiểm thử mã không? Liệu có bị thiếu tài liệu không? Có phải có quá nhiều thành viên kỳ cựu hoặc thành viên dự án cấp cao rời đi mà không được thay thế hay không? Trưởng dự án và quản lý cộng đồng có thể tập trung vào việc tuyển dụng để lấp đầy các vị trí cần thiết.

Trưởng dự án và quản lý cộng đồng của các dự án lớn thường cũng sẽ thu thập các số liệu để tìm hiểu những thông tin quan trọng (như liệu các lỗi quan trọng có được khắc phục kịp thời hay không).

Các phương thức đóng góp trong Mã nguồn Mở

Như đã được giải thích trong phần trước, những người đưa mã, tài liệu hoặc các mục khác vào kho lưu trữ trung tâm được gọi là *các nhà phát triển đóng góp*. Các thành viên dự án chấp thuận các đóng góp và nhận chúng vào kho lưu trữ được gọi là *người được uỷ thác*. Một số thành viên sẽ đảm nhận các vai trò chung khác trong phát triển phần mềm.

Mặc dù thuật ngữ *nhà phát triển đóng góp* thường được dành riêng cho người phát triển mã hoặc một phần nào khác của dự án chính thức nhưng bất kỳ ai tham gia vào thành công của dự án cũng đều đã đóng góp theo một cách nào đó và nên được cảm ơn vì lý do này. Những đóng góp ít chính thức hơn này bao gồm việc báo cáo lỗi, trả lời câu hỏi trên diễn đàn, quyên góp hoặc gây quỹ và quảng bá dự án cho những người bên ngoài dự án.

Giống như các dự án độc quyền, các dự án mã nguồn mở cũng sẽ hỏi người dùng về mong muốn của họ trên phương diện tính năng, cải thiện hiệu suất hoặc các thay đổi khác đối với dự án. Những người dùng này đều mang tới những đóng góp quan trọng bằng cách nêu lên ý kiến của họ.

Các kiểu Nhà phát triển Đóng góp trong Mã nguồn Mở

Nhiều cộng đồng có thể nhận được sự đóng góp không chỉ từ các cá nhân mà còn từ các tập đoàn trả lương cho nhân viên để họ đóng góp vào một dự án mà tập đoàn cho là quan trọng đối với doanh nghiệp của mình.

Đôi khi việc có các nhà phát triển đóng góp được trả lương ngay bên cạnh những tình nguyện viên có thể sẽ tạo ra căng thẳng. Các tình nguyện viên có thể sẽ cảm thấy rằng công việc của họ đang bị lợi dụng hoặc lo sợ rằng những người đóng góp được trả lương đang cố gắng bẻ cong hướng đi của dự án để hướng tới lợi ích của người sử dụng lao động. Quản lý cộng đồng và trưởng dự án phải đảm bảo rằng tất cả những đóng góp được chấp nhận đều mang lại lợi ích cho toàn bộ dự án và người dùng của dự án. Những người tình nguyện cũng phải nhận được động lực để đóng góp vì lý do cá nhân của họ, cho dù là xuất phát từ lòng trung thành với cộng đồng hay để đáp ứng nhu cầu của bản thân.

Một số người thường xuyên đóng góp cho một dự án và đảm nhiệm một số trách nhiệm dài hạn; những người này được gọi là các *thành viên cốt cán*. Các cộng đồng lành mạnh cũng có những nhà phát triển đóng góp thỉnh thoảng báo cáo lỗi hoặc đăng lời khuyên trên diễn đàn nhưng không được yêu cầu chịu trách nhiệm cho dự án.

Tương tự như vậy, một số nhà phát triển đóng góp sẽ là chuyên gia trong lĩnh vực của họ — bất kể có công ty trả tiền cho họ để làm việc trong dự án hay không — trong khi những cá nhân khác lại là dân nghiệp dư thường được gọi là *những người làm vì đam mê*. Chúng ta không cần phải là một chuyên gia để đảm nhận một vai trò quan trọng; chúng ta thậm chí có thể trở thành một thành

viên cốt cán hoặc một trưởng dự án.

Vai trò của các Tổ chức trong các Dự án Mã nguồn Mở

Mặc dù nhiều dự án mã nguồn mở được tạo ra bởi những cá nhân nhiệt thành hoặc các nhóm tình nguyện viên nhỏ nhưng họ sẽ thường tìm kiếm sự hỗ trợ của các tổ chức khi các dự án trở nên lớn hơn. Sự hỗ trợ của các tổ chức có thể ở dưới nhiều hình thức. Nhìn chung, các tổ chức thường thực hiện những đóng góp này khi một dự án mã nguồn mở có thể đáp ứng nhu cầu kinh doanh của họ với chi phí thấp và đáng tin cậy hơn so với việc tái phát minh các tính năng tương tự trong mã độc quyền. Nhiều tổ chức cũng rất coi trọng các đảm bảo do giấy phép mã nguồn mở hoặc tự do mang lại.

Một số người theo chủ nghĩa lý tưởng không tin vào các tập đoàn hoặc tổ chức lớn và sẽ muốn giữ các dự án mã nguồn mở hoàn toàn không bị ảnh hưởng bởi họ. Trong những năm qua, kiểu suy nghĩ tương đối dễ lý giải này đã trở nên ít phổ biến hơn. Hầu hết những người ủng hộ mã nguồn mở đều tin rằng các tập đoàn và các tổ chức khác có thể cung cấp nhiều nền tảng quan trọng cho các dự án mã nguồn mở.

Nhiều dự án mã nguồn mở được bắt đầu trong một tổ chức và thậm chí có thể được dựa trên mã độc quyền mà công ty quyết định mở ra. Để kiếm tiền, công ty có thể quyết định giữ quyền kiểm soát chặt chẽ đối với dự án và chia dự án thành các phần mở và độc quyền: đây được gọi là mô hình *lõi mở* (open core). Có nhiều người sáng lập dự án bắt đầu dự án dưới dạng mã nguồn mở nhưng dần từ đó thành lập ra một công ty; công ty này có thể hoặc không sử dụng mô hình lõi mở.

Các dự án khác thường sẽ cố gắng đảm bảo rằng không có công ty nào kiểm soát hướng đi của họ. Việc duy trì sự độc lập thường đòi hỏi phải có sự tham gia của nhiều tổ chức ủng hộ để không ai có đủ quyền lực để đưa ra các quyết định độc đoán.

Các tổ chức sẽ hỗ trợ mã nguồn mở như thế nào? Như đã đề cập tới trước đây, nhiều tổ chức sẽ đưa các nhân viên được trả lương vào các dự án. Ngoài ra, họ có thể thuê những cá nhân có năng suất cao đã đóng góp cho dự án với tư cách là tình nguyện viên và cũng đã có sự phát triển về chuyên môn trong dự án. Đây là một con đường rất có giá trị giúp cho sinh viên và những nhà phát triển đóng góp tình nguyện khác thăng tiến trong sự nghiệp của mình.

Những công ty cần các phần mở rộng không thu hút người dùng khác không nên gây sức ép để buộc dự án mã nguồn mở phải thêm các tính năng bổ sung; họ nên trả tiền cho nhân viên của mình để viết các tính năng đó mà không phải đóng góp chúng lại cho dự án.

Một ví dụ thú vị về sự căng thẳng có thể xảy ra do nhu cầu của công ty được thể hiện qua hệ điều hành Android nổi tiếng; hệ điều hành này dựa trên Linux và được Google phát triển để điều khiển các thiết bị di động của mình. Một số thay đổi do Google thực hiện đối với Linux sẽ được đóng góp

ngược trở lại cho cộng đồng Linux, trong khi những thay đổi khác sẽ chỉ xuất hiện trong Android. Các nhà phát triển Linux đôi khi cũng sẽ từ chối những thay đổi do Google mang lại — giống như việc mọi dự án đều sẽ chỉ chọn những đóng góp thích hợp.

Có nhiều lý do khiến một công ty hoặc một nhóm nhà phát triển tạo ra một phiên bản riêng cho mã. Lý tưởng nhất là họ có thể đáp ứng nhu cầu của chính mình bằng cách thêm một thư viện tùy chọn hoặc một chuỗi mã có thể được loại trừ trong quá trình biên dịch. Tuy nhiên, đôi khi một nhóm sẽ cảm nhận được một nhu cầu lớn không tương thích với định hướng mà các nhà lãnh đạo dự án đã chọn. Khi một phiên bản riêng biệt của dự án được tạo ra, nó sẽ được gọi là *nhánh* (fork).

Việc phân nhánh từng được coi là đại diện cho sự thất bại trong hợp tác; nhưng ngày nay, nó đã được chấp nhận nhiều hơn. Thông thường, những người tạo ra nhánh sẽ thiết lập một kho lưu trữ mới và bắt đầu một dự án mới. Một số người sẽ làm việc trên cả dự án gốc lẫn dự án nhánh.

(Hãy lưu ý rằng GitHub sử dụng từ “nhánh” cho một hiện tượng rất khác: nhân bản hoặc tạo bản sao của mã dự án để làm việc trên đó một cách riêng biệt.)

Bên cạnh mã, nhiều công ty còn đóng góp cả về mặt tài chính. Họ có thể tài trợ trên các phương diện như tiếp thị hay hội nghị. Họ cũng có thể tham gia vào hội đồng quản trị và đưa ra lời khuyên chuyên môn về định hướng và chiến lược của dự án.

Một ví dụ về tầm quan trọng của những “hỗ trợ mềm” như vậy về máy chủ web Apache cũ. Dự án đã có bước tiến lớn ngay từ đầu khi được IBM quan tâm. Các luật sư của IBM đã chỉ cho dự án cách tạo ra các biện pháp bảo vệ pháp lý và một cuộc họp do IBM chi trả đã giúp ban lãnh đạo Apache tạo ra một tổ chức vững mạnh.

Để duy trì tính độc lập, nhiều dự án mã nguồn mở đã hình thành một quỹ phi lợi nhuận hoặc tham gia một quỹ đã có sẵn. Các ví dụ nổi tiếng về các quỹ hướng dẫn các dự án mã nguồn mở bao gồm Quỹ Linux (Linux Foundation), Quỹ Apache (Apache Foundation) và Quỹ Eclipse (Eclipse Foundation).

Sự hỗ trợ của một tổ chức rất có giá trị vì nó có thể cung cấp các dịch vụ hậu cần mà hầu hết các nhà phát triển phần mềm không muốn giải quyết: hỗ trợ pháp lý như đăng ký nhãn hiệu, bồi thường và cấp phép; hỗ trợ huy động vốn; cơ sở hạ tầng (như cơ sở dữ liệu lỗi và trang web), v.v.

Chuyển giao Quyền

Cũng giống như sách, nhạc và các nỗ lực sáng tạo khác, phần mềm cũng liên quan đến mối quan hệ phức tạp giữa các nhà phát triển, các tổ chức phân phối đóng góp của họ và công chúng nói chung. Về cơ bản, những người đóng góp phải thực hiện một số bước để hợp thức hóa quyền của một dự án mã nguồn mở để có thể sử dụng và phân phối mã của mình.

Vì vậy, nhiều tổ chức mã nguồn mở sẽ yêu cầu những người đóng góp ký một thỏa thuận cấp phép cho người đóng góp (Contributor License Agreement - CLA). Đôi khi, người đóng góp sẽ cung cấp luôn mã cho dự án. Dự án sẽ sở hữu mã và mọi quyền đối với mã đó giống như một công ty độc quyền sở hữu mã mà họ trả tiền cho nhân viên của mình để phát triển.

Các CLA khác sẽ để lại một số quyền trong tay của những cá nhân đóng góp. Những người đóng góp có thể sẽ thích sự linh hoạt này vì họ có thể tiếp tục đóng góp cùng một mã cho một dự án khác hoặc xây dựng doanh nghiệp của riêng họ dựa trên đó.

Để cân đối các đóng góp khác nhau từ những bên khác nhau, giấy phép được cấp cho mã là một yếu tố rất quan trọng. Nhân Linux là một trong những dự án để lại quyền sở hữu trong tay những người đóng góp; do đó, đến nay đã có hàng ngàn người sở hữu quyền đối với mã Linux. Tuy vật, tất cả những người đóng góp cho mã lõi đều đưa nó vào Giấy phép Công cộng Chung GNU (phiên bản 2). Do đó, Linux có thể được tất cả mọi người sử dụng, thay đổi và tái phân phối.

Việc sử dụng một giấy phép duy nhất cho tất cả mọi mã trong một dự án là cách đơn giản nhất để đảm bảo rằng không có điều gì cản trở việc phân phối và sử dụng mã. Tuy nhiên, đôi khi một dự án sẽ cho phép các phần khác nhau của mã được đóng góp theo các giấy phép khác nhau, thường là vì dự án muốn tận dụng một số mã đã tồn tại trước đó đã được phát hành theo một giấy phép khác. Các chuyên gia cấp phép nên đảm bảo rằng các giấy phép trong dự án phải tương thích với nhau.

Quy tắc và Chính sách

Nhiều cộng đồng trực tuyến nổi tiếng với việc ngôn luận không giới hạn (một cách nghĩ khá vô tư rằng "thế nào cũng được") dẫn đến nhiều tình huống lạm dụng lời nói và cãi vã. Ngày nay, hầu hết các cộng đồng mã nguồn mở đều đang đấu tranh với khuynh hướng này (vốn rất dễ được tiếp nhận). Ngược lại, các cộng đồng hiện đại đều muốn mọi người có tính xây dựng, lịch sự, tôn trọng và không phân biệt bất kể giới tính, nhóm dân tộc, kiểu tính cách, v.v.

Kỳ vọng của các thành viên nhóm thường được nêu rõ trong *bản quy tắc ứng xử* giải thích cách tương tác với những thành viên khác trong dự án, cả trực tuyến lẫn trực tiếp. Để có hiệu quả, bản quy tắc ứng xử này phải được thực thi bởi người quản lý cộng đồng và các lãnh đạo của dự án.

Đôi khi, một cá nhân công khai chế giễu hoặc hạ thấp một thành viên khác của cộng đồng sẽ bị trục xuất tạm thời hoặc vĩnh viễn. Trong những trường hợp khác, quản lý cộng đồng hoặc một đồng nghiệp chỉ cần công khai tuyên bố rằng hành vi đó đã vi phạm quy tắc ứng xử và có thể cần trao đổi với người vi phạm bên ngoài diễn đàn. Thông thường, người vi phạm trước đó đã cảm thấy thất vọng, thiếu sự chú ý hoặc kiệt sức, và các thành viên trong cộng đồng có thể giúp người vi phạm tìm ra một biện pháp phù hợp hơn để thể hiện bản thân.

Ngoài hành vi xã hội, cộng đồng cũng sẽ thiết lập các tiêu chuẩn về chất lượng. Tiêu chuẩn chính thức nhất bao gồm *hướng dẫn cấu trúc mã* (để cố gắng đảm bảo rằng tất cả mã có thể trông giống nhau). Các hướng dẫn có thể nhắc nhở các nhà phát triển về các thông lệ thực hành tốt như sử dụng các dấu ngoặc bao bọc các khối mã hoặc có thể chỉ định các chi tiết như cách đặt tên biến và loại thụt lề cần sử dụng.

Các cộng đồng mã nguồn mở cũng có các quy tắc xung quanh việc phát hành mã hoặc các sản phẩm khác. Một số dự án sẽ thiết lập thời gian cố định cho các bản phát hành (ví dụ: Ubuntu đã hứa hẹn một phiên bản hỗ trợ dài hạn (LTS) mới sau mỗi hai năm cùng với các bản phát hành tạm thời thường xuyên hơn). Các dự án khác có thể duy trì một danh sách các tính năng mới và các bản sửa lỗi mà họ muốn trong bản phát hành sắp tới cũng như phê duyệt bản phát hành khi mọi thứ trong danh sách đã được thực hiện. Không giống như hầu hết các doanh nghiệp, các cộng đồng mã nguồn mở thường không thiết lập thời hạn cho những người tham gia vì không ai có thể yêu cầu quá nhiều từ những tình nguyện viên.

Ghi công và Minh bạch

Ngoài thỏa thuận cấp phép cho người đóng góp đã được đề cập tới ở trên, một số dự án sẽ yêu cầu người đóng góp ký *giấy chứng nhận xuất xứ của nhà phát triển* (Developer Certificate of Origin - DCO). Trong DCO, người đóng góp sẽ cam kết rằng họ có quyền hợp pháp để tặng mã.

Ngoài thỏa thuận cấp phép cho những nhà phát triển đóng góp đã đề cập tới ở trên, một số dự án sẽ yêu cầu họ ký *giấy chứng nhận xuất xứ của nhà phát triển* (DCO). Trong DCO, nhà phát triển đóng góp sẽ hứa rằng họ có quyền hợp pháp để tặng mã.

DCO được cho là sẽ đảm bảo rằng nhà phát triển đóng góp thực sự đã viết mã hoặc có được mã một cách hợp pháp. Dự án mã nguồn mở phụ thuộc vào sự trung thực của người đóng góp và những thông tin họ điền vào chứng chỉ.

Tính Đa dạng, Công bằng, Hòa nhập và Không phân biệt đối xử

Các nhà khoa học xã hội đã khẳng định rằng các dự án và công ty được hưởng lợi từ việc có nhiều thành viên với các giới tính, chủng tộc, hoàn cảnh kinh tế, quốc tịch và khả năng khác nhau. Mọi tổ chức đều mang một xu hướng tự nhiên của con người là gắn kết với những cá nhân khác giống với họ. Vì vậy, một cộng đồng coi trọng sự đa dạng và công bằng phải có ý thức đào tạo các thành viên của mình trở nên cởi mở hơn với những người khác biệt so với họ. Phong trào thực hiện lý tưởng này được gọi là sự đa dạng, công bằng và hòa nhập (Diversity, Equity, Inclusion - DEI).

Quy tắc ứng xử là điểm khởi đầu cho DEI. Quy tắc này phải hoan nghênh những người thuộc các giới tính, chủng tộc, v.v. khác nhau. Mọi hành vi vi phạm quy tắc ứng xử đều phải được xử lý kịp thời với một thái độ phản đối rõ ràng. Điều này là do một số nhóm thiểu số đã phải chịu đựng sự

kỳ thị và những bình luận tiêu cực trong suốt cuộc đời của họ, và chỉ một tương tác tồi tệ trên diễn đàn của một cá nhân nào đó cũng có thể khiến họ quyết định rằng họ không còn lý do gì để tham gia vào cộng đồng nữa. Phản ứng nhanh chóng đối với các hành vi gây hấn có thể trấn an họ, để họ biết được rằng cộng đồng đang ủng hộ họ.

Nhưng DEI còn đi xa hơn thế nữa. Cộng đồng nên tiếp cận các nhóm chưa được đại diện nhiều. Ví dụ: nếu đang phát triển một ứng dụng tìm kiếm việc làm, chúng ta nên đảm bảo rằng nó sẽ hiển thị cả việc làm cho những người thu nhập thấp lẫn những người thuộc tầng lớp trung lưu và thượng lưu. Chúng ta cũng nên quảng cáo ứng dụng trong cộng đồng thu nhập thấp và tuyển dụng thành viên từ những cộng đồng đó để thử nghiệm.

Một cộng đồng có thể hưởng lợi từ việc tìm kiếm các tổ chức đào tạo những người từ các cộng đồng thiểu số và từ đó có thể tuyển dụng các thành viên mới cho nhóm của mình. Nhiều tổ chức như vậy đã được thành lập tại các khu vực địa phương cũng như các khu vực không được biết đến rộng rãi trên phạm vi toàn quốc hoặc quốc tế.

Việc hiểu được nhu cầu của các cộng đồng thiểu số chính là chìa khóa. Liệu những người khiếm thị có thể truy cập vào trang web của chúng ta không? Chúng ta có dịch tài liệu của mình sang các ngôn ngữ mà cộng đồng mục tiêu quen thuộc không? Có lẽ chúng ta cần tạo ra các diễn đàn cụ thể sử dụng các ngôn ngữ khác nữa.

Hầu hết các giao tiếp trong cộng đồng mã nguồn mở đều là các giao tiếp không đồng bộ và trực tuyến. Tuy nhiên, nếu có các cuộc họp hoặc phiên trò chuyện đồng bộ, hãy xem xét về vị trí địa lý của mọi thành viên và tìm cách để giúp tất cả mọi người đều có mặt. Ví dụ: khi mọi người đến từ nhiều quốc gia và khu vực đang cùng giao tiếp bằng tiếng Anh, hãy cố gắng giữ cho từ vựng và ngữ pháp thật đơn giản.

Cuối cùng, nếu có thành viên là người thuộc nhóm thiểu số trong nhóm, hãy đảm bảo chúng ta đang lắng nghe họ. Một triệu chứng dễ nhận thấy của sự phân biệt đối xử là không coi trọng những gì các thành viên thuộc nhóm thiểu số nói hoặc là đánh giá thấp tầm quan trọng của chúng.

Mặt khác, đừng tạo gánh nặng cho các thành viên thuộc nhóm thiểu số bằng việc khiến họ phải giải thích về nhu cầu từ cộng đồng của họ — mọi người đều nên được giao nhiệm vụ tìm hiểu về chúng. Các thành viên thuộc nhóm thiểu số có thể thực hiện hoạt động tiếp cận có giá trị cho dự án, nhưng chúng ta không nên gây áp lực để buộc họ phải làm điều này. Mỗi người đều nên có các cơ hội bình đẳng để tham gia theo cách họ muốn mà không cần phải là một nhóm thiểu số tượng trưng.

Bài tập Hướng dẫn

1. Tại sao tất cả những nhà phát triển đóng góp không cung cấp mã của họ cho dự án và từ bỏ mọi quyền đối với mã đó?

2. Nếu ai đó muốn đóng góp cho dự án nhưng không thể lập trình, họ có thể đóng góp bằng cách nào?

3. Tại sao nhiều dự án mã nguồn mở lại tham gia vào một tổ chức?

Bài tập Mở rộng

1. Bạn là người được uỷ thác của dự án. Một người nào đó đã gửi tới đoạn mã được lấy từ một dự án khác (nhưng người đóng góp có quyền đối với mã đó). Mã có định dạng sao khác hoàn toàn so với phần còn lại của mã của bạn. Bạn sẽ làm gì?

2. Bạn đã tạo ra một sản phẩm phần mềm độc quyền và muốn đóng góp một phần của nó cho một dự án mã nguồn mở. Bạn có thể tiếp tục cung cấp sản phẩm độc quyền của mình trong trường hợp nào?

3. Có hai người trong danh sách gửi thư bắt đầu tranh luận về một tính năng trong mã của bạn. Cuộc tranh luận dần trở nên gay gắt hơn cho đến khi một người gọi người kia là kẻ ngốc. Bạn có thể xử lý tình huống này như thế nào?

Tóm tắt

Trong bài học này, chúng ta đã tìm hiểu về việc trở thành một phần của cộng đồng và cách cộng đồng duy trì năng suất. Chúng ta cũng đã học về các hình thức đóng góp, kiểu người đóng góp và các vai trò khác nhau. Chúng ta cũng đã học được cách cộng đồng kiểm soát quyền sử dụng các khoản đóng góp. Chúng ta đã học về các quy tắc trong cộng đồng và cách chúng bảo vệ mọi người thuộc các chủng tộc và giới tính khác nhau khỏi việc bị lăng mạ bằng lời nói.

Đáp án Bài tập Hướng dẫn

1. Tại sao tất cả những nhà phát triển đóng góp không cung cấp mã của họ cho dự án và từ bỏ mọi quyền đối với mã đó?

   Người đóng góp có thể sẽ muốn đóng góp cùng một mã cho một dự án khác hoặc phát hành sản phẩm dựa trên mã đó. Vì vậy, họ có thể sẽ muốn giữ lại một số quyền.

2. Nếu ai đó muốn đóng góp cho dự án nhưng không thể lập trình, họ có thể đóng góp bằng cách nào?

   Có rất nhiều vai trò dành cho người đóng góp ngoài việc viết mã. Một trong số đó bao gồm việc lập tài liệu, kiểm thử và báo cáo lỗi, quản lý cộng đồng, tham gia vào các biểu mẫu, quảng bá dự án và thực hiện các công tác nghệ thuật.

3. Tại sao nhiều dự án mã nguồn mở lại tham gia vào một tổ chức?

   Một tổ chức sẽ xử lý rất nhiều các nhiệm vụ pháp lý, tài chính và các nhiệm vụ khác mà cộng đồng của dự án có thể sẽ gặp khó khăn khi thực hiện.

Đáp án Bài tập Mở rộng

1. Bạn là người được uỷ thác của dự án. Một người nào đó đã gửi tới đoạn mã được lấy từ một dự án khác (nhưng người đóng góp có quyền đối với mã đó). Mã có định dạng sao khác hoàn toàn so với phần còn lại của mã của bạn. Bạn sẽ làm gì?

   Tiêu chuẩn mã hóa của dự án phải giải thích được một cách rõ ràng định của dạng mã. Hãy cảm ơn người đóng góp, hướng dẫn họ về các tiêu chuẩn và nhờ họ định dạng lại mã. Đôi khi chúng ta cũng sẽ có các công cụ tự động để định dạng lại mã theo ý muốn của mình. Nếu người đóng góp không có thời gian để định dạng lại, hãy tìm một thành viên cấp dưới trong nhóm có khả năng thực hiện công việc này.

2. Bạn đã tạo ra một sản phẩm phần mềm độc quyền và muốn đóng góp một phần của nó cho một dự án mã nguồn mở. Bạn có thể tiếp tục cung cấp sản phẩm độc quyền của mình trong trường hợp nào?

   Câu trả lời phụ thuộc vào thỏa thuận cấp phép của nhà phát triển đóng góp. Nếu CLA yêu cầu bạn giao mã lại cho dự án, cấp cho họ mọi quyền, bạn có thể sẽ không thể tiếp tục cung cấp sản phẩm độc quyền của mình. Tuy nhiên, nếu họ cho phép bạn giữ lại quyền đối với mã, bạn có thể phát hành mã theo bất kỳ giấy phép nào mà bạn mong muốn.

3. Có hai người trong danh sách gửi thư bắt đầu tranh luận về một tính năng trong mã của bạn. Cuộc tranh luận dần trở nên gay gắt hơn cho đến khi một người gọi người kia là kẻ ngốc. Bạn có thể xử lý tình huống này như thế nào?

   Bất kỳ thành viên nào nhận thấy có sự căng thẳng hay những bình luận lăng mạ cần phải phản ứng nhanh nhất có thể. Quản lý cộng đồng là người chịu trách nhiệm cuối cùng trong việc khắc phục thiệt hại. Người đã can thiệp nên đăng một bình luận thông báo với toàn bộ danh sách về hành vi vi phạm quy tắc ứng xử của dự án (với hy vọng rằng hành động này sẽ loại trừ được hành vi đó). Người can thiệp cũng có thể liên hệ riêng với từng bên để đảm bảo rằng họ hài lòng với cách giải quyết mâu thuẫn và hiểu về cách thảo luận các bất đồng trên tinh thần xây dựng.

Linux Professional Institute

Chủ đề 056: Cộng tác và Giao tiếp

056.1 Công cụ Phát triển

Tham khảo các mục tiêu LPI

Open Source Essentials version 1.0, Exam 050, Objective 056.1

Khối lượng

2

Các lĩnh vực kiến thức chính

- Hiểu rõ về các công cụ phát triển phần mềm phổ biến
- Hiểu rõ về các môi trường triển khai phổ biến
- Hiểu rõ về các loại thử nghiệm phần mềm phổ biến
- Hiểu rõ các khái niệm về Tích hợp liên tục và Phân phối liên tục (CI/CD)

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Môi trường Phát triển Tích hợp (IDE)
- Xơ
- Trình Biên dịch
- Trình gỡ lỗi
- Kỹ thuật đảo ngược
- Tái cấu trúc
- Thử nghiệm đơn vị
- Thử nghiệm tích hợp
- Thử nghiệm chấp nhận
- Thử nghiệm hiệu suất

- Thử nghiệm khói
- Thử nghiệm hồi quy
- Hệ thống sản xuất, dàn dựng và phát triển
- Hệ thống phát triển cục bộ
- Hệ thống phát triển từ xa
- Đường ống CI/CD

Linux Professional Institute

Bài 1

Chứng chỉ:	Open Source Essentials
Phiên bản:	1.0
Chủ đề:	056 Cộng tác và Giao tiếp
Mục tiêu:	056.1 Công cụ Phát triển
Bài học:	1 trên 1

Giới thiệu

Có tới hàng ngàn công cụ phát triển phần mềm cho cả mã nguồn mở lẫn các sản phẩm độc quyền. Đương nhiên rồi! Các lập trình viên rất thích phát triển các công cụ phục vụ cho bản thân và các đồng nghiệp của họ; vì thế nên rất tự nhiên khi họ dành nhiều thời gian để cố gắng tìm ra một công cụ có thể hoạt động một cách tốt hơn, loại bỏ một số phiền toái khỏi quy trình làm việc hoặc giúp việc triển khai trở nên dễ dàng hơn.

Bài học này sẽ tập trung vào *quy trình* phát triển và giải thích cách các loại công cụ phát triển khác nhau được đưa vào quy trình này. Sẽ có rất ít các công cụ cụ thể được nêu tên vì bất kỳ công cụ nào trong số chúng cũng có thể bị coi là lỗi thời hoặc không còn thông dụng và có thể được thay thế bằng một công cụ được yêu thích mới vào thời điểm bạn đọc bài học này.

Mục tiêu Phát triển

Các công cụ lập trình đôi khi sẽ phải giải quyết nhiều mục tiêu xung đột với nhau. Sau đây là một số mục tiêu phát triển mẫu:

- Tạo ra các chương trình có hiệu suất cao và chính xác.

- Tạo ra các chương trình chạy nhanh.
- Tạo ra các chương trình có khả năng mở rộng tốt.
- Tạo ra các chương trình có khả năng thích ứng cao với nhiều kiểu người dùng, nhiều thiết bị (như máy tính xách tay, điện thoại di động và máy tính bảng) và cácmôi trường khác nhau.
- Tăng tốc quá trình phát triển.
- Tăng tốc việc triển khai các tính năng mới phát triển hoặc sửa lỗi.
- Giảm bớt các tác vụ tẻ nhạt cũng như số lượng lỗi chính tả lập trình lọt qua giai đoạn thử nghiệm của chương trình.
- Hỗ trợ mã và các thiết bị cũ mà tổ chức gặp khó khăn khi thay thế.
- Làm việc cùng với các công cụ quen thuộc khác.
- Cho phép khôi phục dễ dàng trong trường hợp có lỗi hoặc thay đổi kế hoạch.

Các quy trình được mô tả trong bài học này cũng như các công cụ phát triển chính là kết quả của những mục tiêu này (bên cạnh nhiều mục tiêu khác nữa).

Quy trình Phát triển Chung

Phần này sẽ so sánh hai mô hình chung có tầm quan trọng trong lịch sử: *mô hình thác nước* (đã được giới thiệu trong bài học trước) và *tích hợp liên tục/phân phối liên tục* (CI/CD).

Mô hình Thác nước

Mặc dù mô hình thác nước vẫn được sử dụng rộng rãi — đặc biệt là bởi các tổ chức lớn — nhưng nó đã không còn được nhiều nhà phát triển ưa chuộng nữa. Mô hình này phổ biến từ những năm 1950 đến những năm 1970. Nhiều tính chất của mô hình này ngày nay vẫn có thể được tìm thấy rộng rãi (chẳng hạn như các định nghĩa chính thức về yêu cầu và các chương trình kiểm thử ngay trước khi phát hành — tức quy trình *đảm bảo chất lượng*).

Ngay cả vào thời điểm thuật ngữ “thác nước” được đặt ra cho mô hình này, nó đã bị mất uy tín rộng rãi. Các vấn đề gồm có:

- Việc thay đổi các yêu cầu trở nên khó khăn một khi quá trình phát triển bắt đầu.
- Các yêu cầu và thiết kế thường bị hiểu sai vì văn bản thuần ngôn ngữ thông thường có thể sẽ khá mơ hồ. Vấn đề này có thể dẫn đến việc các sản phẩm không đáp ứng được các yêu cầu dự kiến và mất nhiều tháng để khắc phục.
- Quy trình này diễn ra tương đối chậm. Một bản phát hành mới có thể chỉ được thực hiện mỗi năm một lần hoặc thậm chí là ít hơn.

- Các lỗi xuất phát từ tương tác giữa các mô-đun khác nhau rất khó phát hiện cho đến giai đoạn cuối của quy trình và điều này có thể dẫn đến càng nhiều sự chậm trễ hơn cho dự án.
- Quy trình này tạo ra rào cản giữa các bộ phận khác nhau của tổ chức; điều này không tốt cho chất lượng sản phẩm cũng như tinh thần đồng đội của tổ chức.

Nguyên tắc Tích hợp Liên tục/Phân phối Liên tục (CI/CD)

Mô hình thác nước đã gặp phải thách thức vào những năm 1970 bởi một loạt các phong trào khởi nguồn cho mô hình được ca tụng nhất ngày nay (nếu không muốn nói là được sử dụng rộng rãi nhất): CI/CD (Continuous Integration/Continuous Delivery). Các giai đoạn của quá trình áp dụng các hoạt động CI/CD bao gồm *Tuyên ngôn Agile* được phát hành vào năm 2001, SCRUM và DevOps. Mô hình mới này được xây dựng dựa trên các nguyên tắc giao tiếp chặt chẽ giữa các thành viên trong nhóm phát triển, sự tham gia của người dùng cuối hoặc khách hàng, quy trình triển khai nhanh chóng các bản sửa lỗi và tính năng mới cũng như lịch trình kiểm thử liên tục và nghiêm ngặt để duy trì chất lượng trong một môi trường thay đổi nhanh chóng — đôi khi thậm chí là hỗn loạn.

CI/CD tự động hóa càng nhiều bước càng tốt trong các giai đoạn từ viết mã đến triển khai chương trình đã được hoàn thiện cho người dùng cuối. CI/CD yêu cầu phải định nghĩa chuẩn từng bước và thay thế một hành động của con người (chẳng hạn như cài đặt phần mềm thủ công) bằng một quy trình do một chương trình chạy. Do đó, CI/CD là một phần của phong trào "mọi thứ dưới dạng mã" và "cơ sở hạ tầng dưới dạng mã".

Hơn nữa, một khi nhóm đã đưa các quy trình của mình vào mã, các quy trình này có thể được sửa đổi, nâng cấp và ghi lại theo trình tự thời gian giống như mã. Bất kỳ thứ gì do nhóm phát triển viết (cho dù là chương trình, quy trình tự động hay tài liệu) đều sẽ được lưu trữ trong hệ thống kiểm soát phiên bản. Điều này sẽ được thảo luận trong bài học sắp tới.

Khi một số quy trình được tự động hóa, chúng có thể chạy theo trình tự. Do đó, các nhóm phát triển thường nói về *đường ống (pipeline) CI/CD* — tức là một bước thành công trong đường ống sẽ tự động kích hoạt một hoặc nhiều quy trình tiếp theo. Đường ống phải được giám sát theo phương thức tự động để bất kỳ lỗi nào trong quá trình chạy đều sẽ khiến đường ống dừng lại và thông báo lỗi tới các thành viên trong nhóm phát triển.

Mặc dù các phần sau đây sẽ thảo luận về CI và CD một cách riêng biệt nhưng chúng thường được kết hợp với nhau và ranh giới giữa chúng sẽ không thật sự rõ ràng.

Tích hợp Liên tục (CI)

Tích hợp liên tục đề cập đến việc kết hợp nhanh các thay đổi nhỏ vào mã. Kho lưu trữ trung tâm được sử dụng để tạo sản phẩm cho người dùng cuối được gọi là *kho lưu trữ cốt lõi* (hoặc *core repo*).

Mỗi lập trình viên sẽ tạo ra một không gian làm việc cá nhân (thường được gọi là *hộp cát* - sandbox) trên máy tính của họ và họ sẽ tải (pull) các tệp mình cần sửa hoặc nâng cấp từ kho lưu trữ cốt lõi.

Lập trình viên có thể sử dụng máy tính xách tay hoặc máy tính để bàn cá nhân (được gọi là *hệ thống phát triển cục bộ*) để tải xuống các phần có liên quan của kho lưu trữ cốt lõi rồi tải lên các thay đổi sau khi thử nghiệm cục bộ. Ngoài ra, lập trình viên có thể tận dụng xu hướng phổ biến được gọi là "điện toán đám mây" và làm việc trên hệ thống chia sẻ do tổ chức của họ chạy và quản lý — tức một *hệ thống phát triển từ xa*.

Người lập trình thường kiểm tra lỗi bằng một *trình gỡ lỗi* (debugger) — một chương trình riêng biệt chạy công việc của lập trình viên trong một môi trường được kiểm soát. Chúng ta sẽ thảo luận về trình gỡ lỗi và các công cụ phát hiện lỗi khác ở các phần sau.

Để phát hiện lỗi sớm nhất có thể trong quá trình phát triển, lập trình viên cũng sẽ chạy các thử nghiệm trên mã trước khi tải nó lên kho lưu trữ cốt lõi. Chúng được gọi là *thử nghiệm đơn vị* vì mỗi thử nghiệm sẽ tập trung vào một phần tử nhỏ của mã (ví dụ như liệu liệu một hàm có tăng bộ đếm như dự định hay không).

Giai đoạn cuối cùng trong CI là kiểm tra các thay đổi mà lập trình viên đưa vào kho lưu trữ cốt lõi. Ở giai đoạn này, các công cụ sẽ chạy các *thử nghiệm tích hợp* để đảm bảo lập trình viên không làm hỏng bất kỳ thứ gì trong toàn bộ dự án.

Bất kể tự động hóa đã phát triển đến mức nào, một số thành viên chuyên gia của nhóm vẫn nên can thiệp tại thời điểm tích hợp để đảm bảo sự thay đổi sẽ phù hợp với mong muốn của cả nhóm. Các thử nghiệm tự động có thể xác định rằng không có phần bị lỗi và thậm chí là liệu thay đổi có tạo ra hiệu ứng cần thiết trong chương trình hay không. Tuy nhiên, các thử nghiệm này không thể kiểm tra tất cả các nguyên tắc được nhóm coi là quan trọng.

Do đó, trước khi tiến hành triển khai, thành viên trong nhóm nên kiểm tra về vấn đề bảo mật, tuân thủ các tiêu chuẩn mã hóa, tài liệu phù hợp và các nguyên tắc khác (không có gì đáng ngạc nhiên khi cũng có các công cụ để thực hiện một vài trong số các nhiệm vụ này; chúng ta sẽ tìm hiểu về chúng trong các phần sau của bài học này).

Có rất nhiều thử nghiệm được tiến hành trong CI và chúng phải diễn ra một cách nhanh chóng và đáng tin cậy để hỗ trợ tốc độ phát triển hiện đại. Do đó, các công cụ hiện đại để tích hợp mã và chạy thử nghiệm phải được tự động hóa. Một lập trình viên phải có khả năng chạy toàn bộ bộ thử nghiệm đơn vị thông qua một lệnh hoặc một cú nhấp chuột duy nhất.

Nhìn chung, một thử nghiệm tích hợp một phần hoặc toàn phần sẽ chạy bất cứ khi nào lập trình viên tải mã lên kho lưu trữ cốt lõi. Bất kỳ lỗi nào được phát hiện thông qua các thử nghiệm đều có thể nhanh chóng được báo cáo tới lập trình viên.

Phân phối Liên tục (CD)

Như đã được giải thích trong phần trước, CI bao gồm các quy trình tự động tạo ra một phiên bản mới của chương trình trong kho cốt lõi. Khái niệm *Phân phối liên tục* đề cập đến bất kỳ điều gì xảy ra sau giai đoạn đó để đưa phiên bản mới ra bên ngoài và để người dùng cuối có thể hưởng lợi từ nó. D trong CD đôi khi được mở rộng thành “phát triển” (development) hoặc “triển khai” (deployment) bên cạnh định nghĩa “phân phối” (delivery).

Nhiệm vụ chính của CD là kiểm tra mã một cách kỹ lưỡng và tải mã đó lên máy tính của người dùng hoặc một kho lưu trữ ứng dụng.

Giai đoạn CD sẽ chạy một loạt các thử nghiệm để đảm bảo tối đa rằng sản phẩm có thể hoạt động tốt. Một bộ thử nghiệm tích hợp lớn hơn có thể được chạy cùng với một số thử nghiệm khác để xác định tác động của chúng đến người dùng, hiệu suất cũng như bảo mật. Phần sau của bài học này sẽ mô tả về một số thử nghiệm như vậy.

Thử nghiệm nên được thực hiện trên các hệ thống khác với các hệ thống sản xuất phục vụ người dùng. Một lỗi nghiêm trọng không chỉ có thể làm sập hệ thống sản xuất mà còn có thể làm hỏng dữ liệu người dùng. Hơn nữa, các thử nghiệm sẽ cạnh tranh lẫn nhau về thời gian hệ thống và làm giảm hiệu suất cho người dùng.

Vì vậy, để thử nghiệm, tốt nhất là thiết lập một môi trường điện toán hoàn chỉnh tương tự như môi trường sản xuất của dự án với các phần cứng và phần mềm tương tự. Môi trường trung gian nơi được sử dụng để chạy thử nghiệm trước khi triển khai thường được gọi là môi trường *dàn dựng* (staging environment).

Tất nhiên, việc tạo môi trường thử nghiệm có cùng quy mô với môi trường sản xuất là không thực tế; tuy nhiên, các yếu tố thiết yếu của môi trường sản xuất (như cơ sở dữ liệu) vẫn phải được đưa vào.

Một yêu cầu của tự động hóa CD là phân biệt giữa môi trường thử nghiệm và môi trường sản xuất. Tất cả các cài đặt và quy trình của mã đều phải được điều chỉnh theo môi trường mục tiêu.

CD cung cấp tiềm năng phát triển nhanh tốt nhất khi mã là một dịch vụ chạy trên hệ thống riêng của tổ chức. Ví dụ: nếu là một sàn bán lẻ cung cấp một trang web tương tác, chúng ta sẽ có quyền truy cập vào tất cả các máy chủ web của mình. Nếu muốn, chúng ta cũng có có thể cập nhật chúng nhiều lần trong ngày và nhanh chóng khôi phục các thay đổi nếu chúng gây ra sự cố cho người dùng.

Các Công cụ Phát triển Phần mềm phổ biến

Phần này sẽ trình bày về các loại công cụ phổ biến được các nhóm lập trình sử dụng ở ba cấp độ:

tạo mã, thử nghiệm và triển khai.

Trình Biên dịch

Một công cụ lập trình cơ bản là *trình biên dịch* (complier) được sử dụng để chuyển đổi các ngôn ngữ lập trình cấp cao và phức tạp thành các lệnh chạy trên bộ xử lý của máy tính.

Các máy tính sẽ chạy *mã máy* gồm các chuỗi bit (một và không) mà bộ xử lý chuyển thành lệnh và dữ liệu: tải giá trị từ bộ nhớ vào thanh ghi, thêm giá trị của hai thanh ghi, v.v. Bộ xử lý được phân biệt bởi các định dạng và tập lệnh (set of instructions) khác nhau mà chúng có; vì vậy, chúng cũng có mã máy khác nhau. Các nhà cung cấp luôn cố gắng phát minh ra các bộ xử lý mới sử dụng cùng một mã máy để cho phép khách hàng chạy các chương trình cũ của họ trên các bộ xử lý mới. Khi các nhà cung cấp làm như vậy, chúng ta cần nhắc đến *họ* (families) của bộ xử lý.

Trong những năm đầu của lập trình, giai đoạn tiếp theo sẽ là *hợp ngữ* (assembly language) được sử dụng để cung cấp các thuật ngữ mà con người có thể đọc được (chẳng hạn như ADD) cho mỗi lệnh. Các lập trình viên sẽ viết bằng hợp ngữ và gửi mã của họ đến một công cụ gọi là *trình dịch hợp ngữ* (assembler) để dịch hợp ngữ thành mã máy. Mã máy cũng có cách gọi khác là ngôn ngữ máy.

Sau đó, các ngôn ngữ cấp cao hơn sẽ được tạo ra. Ngày nay, chúng ta có thể tạo các luồng điều khiển phức tạp mà thậm chí không cần chỉ định các chi tiết cho chúng; chúng ta chỉ cần chỉ ra kết quả mà mình mong muốn. Các chương trình này cần có một trình biên dịch để chuyển mã nguồn thành mã máy. Trình biên dịch có thể thực hiện các phép biến đổi và tối ưu hóa một cách khá thông minh.

Có khá nhiều các ngôn ngữ có sẵn nhiều trình biên dịch khả dụng. Chúng ta có thể tìm thấy rất nhiều trình biên dịch cho các ngôn ngữ phổ biến như C và Java. Trong cộng đồng mã nguồn mở và tự do, hầu hết mọi người đều sử dụng GCC (GNU Compiler Collection) hoặc trình biên dịch LLVM cho C và C++.

Ban đầu, trình biên dịch sẽ biên dịch từng tệp mã nguồn một để tạo ra một *tệp đối tượng* trung gian, sau đó kết hợp tất cả các tệp đối tượng thành một chương trình bằng cách gọi một công cụ khác có tên là *trình liên kết* (linker). Trình biên dịch hiện đại có thể biên dịch nhiều tệp cùng một lúc để thực hiện tối ưu hóa vượt qua ranh giới các tệp.

Trình biên dịch được sử dụng để biên dịch thành hợp ngữ; điều này có thể sẽ hữu ích vì mỗi loại bộ xử lý máy tính sẽ hỗ trợ một mã máy khác nhau, nhưng chúng cũng có thể hỗ trợ cùng một hợp ngữ. Hợp ngữ cũng có rất nhiều biến thể. Bước cuối cùng trong biên dịch và liên kết là tạo ra mã máy. Cũng giống như liên kết, trình biên dịch hiện đại sẽ biết cách lắp ráp mã thành mã máy.

Có một giai đoạn khác trong nhiều ngôn ngữ hiện đại là mã trung gian, thường được gọi là *mã byte*

(byte code). Mã này đã được biên dịch thành một định dạng nhị phân đủ cao cấp để có thể di động. Ví dụ: ngôn ngữ Java được biên dịch thành mã byte để chương trình có thể được tải lên nhiều loại bộ xử lý khác nhau.

Trong quá trình tạo mã byte từ mã nguồn, trình biên dịch đã thực hiện phần lớn công việc. Sau đó, mỗi máy tính sẽ lưu trữ phiên bản chương trình riêng của mình được gọi là *máy ảo* để hoàn tất quá trình chuyển đổi từ mã byte thành một tập lệnh mà máy ảo có thể thực thi. Đôi khi mã byte cũng được biên dịch thành mã máy. Việc chuyển từ mã byte sang một tập lệnh thuận tiện hơn cho người dùng cuối so với việc chuyển từ ngôn ngữ lập trình cấp cao sang mã máy. Đây là một lợi thế lớn cho Java khi nó được phát minh vì các nhà thiết kế muốn nó chạy trong một trình cắm máy ảo cho các trình duyệt web nơi bộ xử lý và môi trường hoạt động của người dùng có thể nói là khá đa dạng.

Các nhà thiết kế Java đã quảng bá lợi ích của mã byte thông qua câu khẩu hiệu tiếp thị: “Biên dịch một lần, chạy mã mọi nơi”. Một số lợi thế khác của mã byte dần xuất hiện sau đó. Các ngôn ngữ mới có thể được tạo ra để cung cấp một trải nghiệm rất khác cho các lập trình viên (hy vọng làm cho công việc lập trình trở nên dễ dàng hơn và tạo ra các mã dễ bảo trì hơn) trong khi tạo ra cùng một mã byte mà máy ảo Java hỗ trợ. Các hàm từ các ngôn ngữ này có thể dễ dàng kết hợp vào các chương trình Java hiện có.

Có một số ngôn ngữ (chẳng hạn như Python) mà chúng ta không cần phải biên dịch mã nguồn: chúng ta chỉ cần nhập các câu lệnh vào một công cụ xử lý được gọi là *trình thông dịch* (interpreter) và công cụ này sẽ chuyển đổi mã trực tiếp thành mã máy và thực thi mã đó. Trình thông dịch luôn chậm hơn so với trình biên dịch; tuy nhiên, hiện nay, một số trình thông dịch đã đủ hiệu quả để không gây ra nhiều ảnh hưởng đến hiệu suất. Tuy nhiên, một số thư viện phổ biến trong ngôn ngữ Python sẽ có bao gồm cả những hàm mà các nhà phát triển có thể truy cập được bằng ngôn ngữ thông dịch nhưng được lập trình bằng ngôn ngữ C để tăng tốc độ thực thi.

Nhiều ngôn ngữ được thông dịch cũng cung cấp cả trình biên dịch. Chúng sẽ tạo ra mã byte hoặc mã máy; điều này sau đó sẽ giúp chúng thực thi nhanh hơn trình thông dịch.

Có nhiều công cụ dựng để giúp lập trình viên quản lý các tệp và hàm. Một chương trình phức tạp có thể chứa tới hàng trăm tệp và lập trình viên sẽ cần phải biên dịch các kết hợp tệp khác nhau bằng các tùy chọn biên dịch khác nhau tại các thời điểm khác nhau (ví dụ như để hỗ trợ gỡ lỗi). Một công cụ xây dựng sẽ cho phép lập trình viên lưu trữ các tùy chọn và kết hợp tệp khác nhau và dễ dàng chọn loại bản dựng mong muốn. Maven và Gradle là các công cụ xây dựng phổ biến cho Java và các ngôn ngữ liên quan.

Công cụ tạo Mã

Lập trình viên không cần phải bắt đầu viết mã từ những ký tự đầu tiên. Gần đây, nhiều dịch vụ đã

xuất hiện để tạo mã dựa trên các mô tả thuần văn bản của người dùng về những gì họ cần. Đây là một dạng AI tạo sinh (generative AI). Cũng giống như các dịch vụ tương tự khác, một số người cũng phàn nàn rằng nó sử dụng công sức của các lập trình viên trước đây mà không trả một khoản phí nào cho họ và tạo ra các mã nguồn yếu hơn (cho đến nay). Bên cạnh những tranh cãi về đạo đức, nhiều lập trình viên cho biết việc tạo mã tự động đã cải thiện đáng kể năng suất của họ.

Một dạng tạo mã đã có trong nhiều ngôn ngữ lập trình trong nhiều năm là *tái cấu trúc* (refactoring). Tái cấu trúc sẽ kiểm tra cả một chương trình lớn và có thể dễ dàng phát triển thành một mớ hỗn độn các tệp và hàm theo thời gian. Quy trình tái cấu trúc sẽ luân chuyển các hàm để tạo ra một cấu trúc logic hơn cho chương trình nhằm theo đuổi khả năng bảo trì được cải thiện và giảm tỷ lệ trùng lặp mã nguồn.

Một số lập trình viên sẽ được giao nhiệm vụ tái tạo chức năng của mã khác. Có thể họ sẽ phải viết một chương trình mới để thay thế một chương trình cũ bằng mã nguồn bị thiếu phải ngừng hoạt động. Hoặc họ cũng có thể bắt chước một chương trình của đối thủ cạnh tranh. Loại nghiên cứu này được gọi là *kỹ thuật đảo ngược* (reverse engineering). Một công cụ hữu ích cho mục đích này chính là *trình dịch ngược* (disassembler); nó sẽ chuyển mã máy thành hợp ngữ. Ngoài ra, chúng ta còn có trình dịch ngược cho mã byte.

Trình gỡ lỗi

Hầu hết các lập trình viên đều sẽ dành nhiều thời gian để gỡ lỗi hơn là viết mã. Con người khó có thể luôn luôn suy nghĩ logic một cách toàn diện; do đó, chúng ta luôn có khả năng quên mất một số chi tiết mà máy tính yêu cầu để chạy chương trình theo cách mà chúng ta muốn. Vì thế mà mã của chúng ta thường sẽ thất bại trong những lần thử nghiệm đầu tiên. Chúng ta có thể hưởng lợi rất nhiều từ *trình gỡ lỗi* (debugger) khi đối mặt với vấn đề này.

Trình gỡ lỗi sẽ hỗ trợ các công tác nghiên cứu chuyên sâu để có thể cắt giảm đi nhiều giờ làm việc của quá trình tìm lỗi.

Một lập trình viên có thể yêu cầu chương trình dừng tại một số vị trí quan trọng trong chương trình (một *điểm nghỉ*, còn được gọi là breakpoint) — chẳng hạn như phần đầu của một hàm hoặc một vòng lặp. Trình gỡ lỗi có thể hiển thị giá trị của các biến và thậm chí là các thanh ghi máy tính tại điểm hiện tại trong chương trình. Lập trình viên cũng có thể chạy từng câu lệnh một và xem kết quả (*chạy bước đơn* - single stepping). Nếu lập trình viên muốn xem khi nào một biến thay đổi và cách nó thay đổi trong quá trình chạy, họ có thể đặt một *điểm quan sát* (watchpoint).

Trình gỡ lỗi nổi bật nhất trong thế giới mã nguồn mở và tự do — đặc biệt là đối với C và C++ — là trình gỡ lỗi GNU; nó hoạt động cùng với trình biên dịch GNU đã đề cập tới ở trên. Các ngôn ngữ khác cũng sẽ có các trình gỡ lỗi chuyên dụng của chúng.

Công cụ Phân tích

Mặc dù việc gỡ lỗi thường có thể phát hiện ra lỗi một cách nhanh chóng nhưng tốt hơn hết là chúng ta nên loại bỏ các lỗi chính tả và các vấn đề cơ bản khác trong những giai đoạn đầu quá trình lập trình. Có nhiều công cụ phân tích rất tinh xảo để kiểm tra một chương trình. Công cụ *Phân tích tĩnh* sẽ kiểm tra mã của chương trình. Công cụ *Phân tích động* sẽ chạy chương trình và kiểm tra các vấn đề trong quá trình thực thi.

Một trong những dạng phân tích tĩnh sớm nhất được gọi là *trình phân tích xơ* (linter). Nó có thể phát hiện ra các vấn đề như khi chúng ta gán giá trị của một biến dấu phẩy động cho một số nguyên. Điều này có thể có hoặc không tạo ra vấn đề và có thể hoặc không thể bị trình biên dịch phát hiện. Trong quá trình sản xuất, nó có thể dẫn đến các kết quả không chính xác.

Ngày nay, hầu hết các trình biên dịch đều có thể thực hiện công việc của một trình phân tích xơ. Một số trình biên dịch (chẳng hạn như trình biên dịch cho ngôn ngữ Rust) nổi tiếng vì tính nghiêm ngặt trong việc từ chối mã được viết một cách yếu kém.

Nhiều loại công cụ phân tích khác sẽ chạy biệt lập với trình biên dịch. Ví dụ: các công cụ phân tích bảo mật có thể tìm ra các vấn đề khiến một chương trình có khả năng cao sẽ bị hack. Một trong các lỗi phổ biến của các nhà phát triển có thể kể đến là khi mã của họ gọi một hàm và không kiểm tra xem hàm đó có trả về lỗi hay không.

Môi trường Phát triển Tích hợp (IDE)

Có rất nhiều lập trình viên sử dụng trình soạn thảo văn bản để nhập và chỉnh sửa mã của họ. Trình soạn thảo văn bản khác với trình xử lý văn bản vốn có nhiều định dạng (như in nghiêng và in đậm, dấu đầu dòng và danh sách được đánh số, v.v.). Trình xử lý văn bản sẽ tạo các ra văn bản không có trang trí; đây là điều mà các ngôn ngữ lập trình yêu cầu.

Ngoài ra, chúng ta còn có các công cụ tinh vi dành riêng cho việc giúp các lập trình viên phát triển các chương trình của họ. Các công cụ này sẽ cảnh báo về ngôn ngữ lập trình đang sử dụng. Ví dụ: nếu chúng ta bắt đầu nhập tên của một biến hoặc hàm, công cụ có thể gợi ý một ví dụ hoàn chỉnh. Các công cụ này có thể kiểm tra lỗi trong khi chúng ta viết mã, định dạng mã theo một cách nhất quán và gọn gàng, chạy các công cụ phân tích và trình gỡ lỗi, biên dịch mã, xử lý các phiên tải vào hệ thống kiểm soát phiên bản và xử lý các tác vụ khác. Do đó, chúng được gọi là *môi trường phát triển tích hợp* (Integrated Development Environments - IDE).

Eclipse là một IDE mã nguồn mở rất phổ biến.

Các Phương pháp Thử nghiệm Phần mềm phổ biến

Thử nghiệm là một phần quan trọng trong phát triển phần mềm. Các lập trình viên sẽ chạy các thử nghiệm đơn vị khi họ phát triển mã. Các loại thử nghiệm khác thường sẽ chạy khi một lập trình viên tải mã trở lại kho lưu trữ cốt lõi hoặc trong quá trình triển khai.

Thử nghiệm Đơn vị

Chúng ta đã thấy rằng một lập trình viên phải hết sức tìm kiếm hết sức cẩn thận các lỗi trước khi gửi mã để tích hợp vào kho lưu trữ cốt lõi. Việc thử nghiệm đơn vị rất quan trọng trong việc phát hiện lỗi.

Việc viết các thử nghiệm này vừa là một môn nghệ thuật, vừa là một môn khoa học; khối lượng mã thử nghiệm có thể sẽ vượt qua cả khối lượng của mã sản xuất. Thậm chí còn có một mô hình phát triển được gọi là *phát triển theo hướng thử nghiệm* (Test-Driven Development - TDD); trong đó, các lập trình viên sẽ viết các thử nghiệm trước khi viết mã mà họ muốn thử nghiệm. Những người ủng hộ TDD cho rằng nó sẽ lấp đầy các lỗ hổng trong quy trình thử nghiệm và giúp đảm bảo rằng mã sẽ thực hiện những gì mà lập trình viên muốn nó thực hiện.

Việc kiểm tra các sai sót trong quá trình thực thi chương trình cũng quan trọng không kém so với việc kiểm tra các tình huống hoạt động đúng cách. Nếu người dùng hoặc một phần khác của chương trình gửi dữ liệu đầu vào không hợp lệ cho một hàm, điều quan trọng là hàm phải phát hiện được ra vấn đề và báo cáo một thông báo lỗi phù hợp.

JUnit là một công cụ mã nguồn mở phổ biến được dùng để chạy các thử nghiệm đơn vị trên các chương trình Java.

Thử nghiệm Tích hợp, Hồi quy và Khói

Trong khi các thử nghiệm đơn vị tập trung vào các hành động riêng lẻ của các hàm cụ thể, nhóm phát triển cũng nên chạy các thử nghiệm ở cấp độ cao hơn để đảm bảo tổng thể sản phẩm hoạt động một cách bình thường. Các thử nghiệm thường sẽ mô phỏng hành vi của người dùng. Ví dụ: trong ứng dụng quản lý nhà hàng, các thử nghiệm có thể kiểm tra xem người dùng có nhận được món mà họ đã đặt hay không.

Khi một thay đổi trong một chương trình làm hỏng một số hàm hoạt động bình thường trước đó, lỗi này sẽ được gọi là *hồi quy* (regression) và các thử nghiệm sẽ được gọi là *thử nghiệm hồi quy*.

Khi một nhóm phát triển chuẩn bị phát hành một sản phẩm, giai đoạn thử nghiệm đầu tiên thường sẽ rất ngắn. Nhóm phát triển sẽ kiểm tra các hoạt động quan trọng nhất do một chương trình thực hiện và sẽ dừng thử nghiệm nếu có bất kỳ lỗi nào phát sinh, từ đó có thể tiết kiệm được

thời gian. Loại thử nghiệm này được gọi là *thử nghiệm khói* (smoke test) vì một ứng dụng dễ gặp trục trặc như vậy rất giống như một thiết bị kém chất lượng dễ bị bắt lửa.

Một số sản phẩm sẽ liên quan tới tương tác của người dùng; các ứng dụng web và di động là những ví dụ phổ biến nhất. Do đó, nhiều công cụ đã được tạo ra để mô phỏng tương tác của người dùng. Thử nghiệm sẽ chạy tự động và kích hoạt mã (mã này sẽ chạy nếu người dùng nhấn nút). Selenium là một công cụ phổ biến trong danh mục này.

Thử nghiệm Chấp nhận

Các chương trình có giao diện người dùng cần một mức thử nghiệm bổ sung vượt qua mục tiêu chứng minh rằng chúng phản ứng đúng với một số đầu vào nhất định. Các chương trình cũng phải có một bố cục hợp lý trên màn hình. *Thử nghiệm chấp nhận* sẽ kiểm tra tác động của chương trình lên người dùng. Các nhóm đảm bảo chất lượng thường sẽ chạy các thử nghiệm này sau khi thử nghiệm tích hợp và hồi quy đã chứng minh rằng chương trình đã chính thức chính xác và đáp ứng được kỳ vọng của người dùng.

Thử nghiệm Bảo mật

Bảo mật là một khía cạnh vô cùng quan trọng và ngay cả một lỗi bảo mật nhỏ cũng có thể khiến một tổ chức phải chịu những thiệt hại rất lớn. Chúng ta đã thấy rằng các lập trình viên có thể chạy các công cụ phân tích để kiểm tra tính bảo mật của mã. Trong giai đoạn kiểm tra đảm bảo chất lượng, các thử nghiệm cũng có thể xác định xem chương trình có lỗ hổng hay không. Các thử nghiệm sẽ chạy chương trình với các đầu vào độc hại và đảm bảo rằng chương trình sẽ từ chối đầu vào mà không bị lỗi, thực hiện các hành động không mong muốn hoặc tiết lộ các thông tin nhạy cảm.

Một loại thử nghiệm đã được chứng minh là có giá trị trong một số tình huống là *thử nghiệm mờ* (fuzz testing). Khuôn khổ thử nghiệm này sẽ chỉ tạo ra các chuỗi rác ngẫu nhiên và gửi chúng đi làm đầu vào cho chương trình. Điều này nghe có vẻ như sẽ chỉ lãng phí thời gian nhưng nó lại thường phát hiện ra các lỗ hổng mà các thử nghiệm thông thường không phát hiện được.

Thử nghiệm Hiệu suất

Sau khi chương trình được đánh giá là hoạt động một cách chính xác bằng các thử nghiệm khác, các nhóm phát triển nên xác định xem chương trình có chạy đủ nhanh hay không. *Thử nghiệm hiệu suất* yêu cầu một môi trường tương tự như môi trường mà người dùng sẽ tương tác với chương trình. Ví dụ: nếu người dùng gửi yêu cầu từ xa qua mạng thì thử nghiệm hiệu suất cũng nên được thực hiện qua một mạng từ xa.

Một số thư viện lập trình sẽ được thử nghiệm thông qua các *mức chuẩn mực* (benchmarks): các

thử nghiệm tiêu chuẩn được sử dụng để so sánh các thư viện khác nhau hoặc các phiên bản khác nhau của cùng một thư viện.

Môi trường Triển khai Chung

Một công cụ CI/CD sẽ cho phép lập trình viên xây dựng các đường ống vô cùng phức tạp. Giống như các chương trình máy tính, một đường ống có thể chứa nhiều thử nghiệm và nhánh. Bằng cách phân nhánh, chúng ta có thể thực hiện một tập hợp các hoạt động này trong môi trường thử nghiệm và một tập hợp khác trong môi trường sản xuất. Chúng ta có thể sử dụng công cụ này để tự động cài đặt cơ sở dữ liệu chính xác hoặc các phần mềm khác cần thiết cho các chương trình khác nhau.

CD sẽ chạy chồng lên DevOps. Trong các môi trường đám mây, các công cụ CD và DevOps sẽ tạo ra các hệ thống máy tính ảo theo thủ tục tự động với tất cả các thành phần cần thiết để các chương trình có thể chạy. Các công cụ tự động (đôi khi được gọi là các công cụ *phối hợp* - orchestration tools) sẽ kiểm tra các lỗi của hệ thống ảo và tự động khởi động lại chúng.

Công việc chính của một công cụ CD là khởi chạy và thực hiện từng bước thông qua từng đường ống. Công cụ này sẽ kiểm tra các kết quả của từng giai đoạn của đường ống và lựa chọn xem phải tiếp tục hay dừng lại. Công cụ này cũng sẽ cho phép chúng ta lập lịch và ghi lại các hoạt động của nó.

Thông thường, chúng ta sẽ phải chạy một tác vụ nhiều lần với các thay đổi nhỏ nhặt. Ví dụ: các nhóm phát triển sẽ phân biệt giữa việc triển khai vào môi trường thử nghiệm và việc triển khai vào môi trường sản xuất. Do đó, các công cụ CD sẽ cung cấp các tham số mà chúng ta có thể điền vào với các giá trị khác nhau khi chạy đường ống.

Đôi khi một tiến trình sẽ yêu cầu các lệnh được nhập theo cách truyền thống vào cửa sổ dòng lệnh. Do đó, một công cụ CD sẽ cung cấp các cơ chế để chạy các lệnh tùy ý. Thông thường, nó sẽ cung cấp các móc nối như `preprocess` để chạy các lệnh trước một giai đoạn của đường ống và `postprocess` để chạy các lệnh sau một giai đoạn của đường ống.

Jenkins có lẽ là công cụ mã nguồn mở phổ biến nhất đối với trường hợp phối hợp được mô tả trong phần này.

Bài tập Hướng dẫn

1. Một số cách để kiểm tra tính bảo mật của một chương trình là gì?

2. Tại sao bạn lại phải viết nhiều thử nghiệm đơn vị cho một hàm chương trình duy nhất?

Bài tập Mở rộng

1. Nhóm phát triển của bạn đã thừa hưởng một ứng dụng cũ chạy chậm và khó để thêm tính năng. Có cách nào có thể cải thiện ứng dụng mà không phải vứt bỏ và viết lại một ứng dụng mới từ đầu hay không?

2. Trong các dự án lớn, thường sẽ xảy ra trường hợp Nhóm A muốn một tính năng được triển khai trong một phần của hệ thống do Nhóm B duy trì, nhưng Nhóm B lại không coi tính năng đó là ưu tiên. Làm thế nào để Nhóm A có thể mã hóa tính năng đó như một phần của dự án của Nhóm B?

Tóm tắt

Bài học này đã nêu ra các loại công cụ được sử dụng trong suốt quá trình phát triển: trình biên dịch và các công cụ tạo mã khác, trình phân tích, các thử nghiệm và các công cụ CI/CD tự động hóa tích hợp và phân phối. Có nhiều tùy chọn cho từng hoạt động này và một công cụ phổ biến hiện nay có thể dễ dàng bị thay thế sau một năm. Việc hiểu rõ cách tất cả các công cụ này kết hợp với nhau trong quá trình phát triển sẽ giúp chúng ta xác định được rõ những yếu tố cần thiết đối với nhu cầu của mình.

Đáp án Bài tập Hướng dẫn

1. Một số cách để kiểm tra tính bảo mật của một chương trình là gì?

   Đầu tiên, các chuyên gia — là chính con người — có thể kiểm tra mã.

   Có nhiều công cụ phân tích tĩnh và động hiện có có thể phát hiện các hoạt động lập trình kém khiến chương trình có nguy cơ bị đe dọa về bảo mật.

   Các công cụ khác sẽ gửi dữ liệu đầu vào độc hại đến các chương trình đang chạy và kiểm tra phản ứng của chúng.

2. Tại sao bạn lại phải viết nhiều thử nghiệm đơn vị cho một hàm chương trình duy nhất?

   Một hàm chương trình thường phải chạy trên nhiều loại đầu vào khác nhau và mỗi loại đều xứng đáng có một thử nghiệm riêng. Ví dụ: hàm có thể xử lý giá trị đầu vào bằng 0 theo một cách đặc biệt. Bạn cũng cần dự đoán đầu vào không hợp lệ và viết các thử nghiệm để cho thấy hàm có thể xử lý đầu vào đó một cách phù hợp.

Đáp án Bài tập Mở rộng

1. Nhóm phát triển của bạn đã thừa hưởng một ứng dụng cũ chạy chậm và khó để thêm tính năng. Có cách nào có thể cải thiện ứng dụng mà không phải vứt bỏ và viết lại một ứng dụng mới từ đầu hay không?

   Đầu tiên, hãy đảm bảo rằng dự án đang được kiểm soát phiên bản nếu nó chưa có sẵn ở đây.

   Sau khi thêm một tính năng mới, hãy chạy các thử nghiệm hồi quy để xác định xem chương trình gặp lỗi ở đâu và chỉ định các lập trình viên để tìm ra hàm nào chịu trách nhiệm cho lỗi đó. Các hàm này có thể được thay thế một cách có chọn lọc.

   Thử nghiệm hiệu suất có thể xác định các hàm cụ thể đang chạy chậm để từ đó, bạn có thể tập trung nỗ lực vào việc sửa hoặc thay thế các mã kém hiệu quả nhất.

   Nếu mã được viết bằng một ngôn ngữ không còn phổ biến nữa, hãy cân nhắc về việc thêm các tính năng được viết bằng một ngôn ngữ mà nhóm ưa thích. Hãy đảm bảo rằng các hàm trong ngôn ngữ mới có thể được tích hợp với các hàm cũ.

2. Trong các dự án lớn, thường sẽ xảy ra trường hợp Nhóm A muốn một tính năng được triển khai trong một phần của hệ thống do Nhóm B duy trì, nhưng Nhóm B lại không coi tính năng đó là ưu tiên. Làm thế nào để Nhóm A có thể mã hóa tính năng đó như một phần của dự án của Nhóm B?

   + Nhóm B có thể cho phép Nhóm A tạo một nhánh mới, mã hóa tính năng và gửi nhánh đó cho Nhóm B để hợp nhất vào dự án của mình. Tuy nhiên, Nhóm A không thể được trao quyền để làm bất cứ điều gì mà họ muốn. Nhóm B phải cung cấp tài liệu và trợ giúp để Nhóm A tuân thủ các tiêu chuẩn của dự án. Một thành viên của Nhóm B cũng phải xem xét nội dung gửi của Nhóm A và chạy tất cả các hình thức thử nghiệm tích hợp thông thường.

   + Hình thức cộng tác này đôi khi được gọi là InnerSource vì nó giống với mã nguồn mở nhưng sẽ chỉ diễn ra trong một tổ chức duy nhất.

056.2 Quản lý Mã nguồn

Tham khảo các mục tiêu LPI

Open Source Essentials version 1.0, Exam 050, Objective 056.2

Khối lượng

3

Các lĩnh vực kiến thức chính

- Hiểu rõ về kho lưu trữ mã nguồn (công khai và riêng tư)
- Hiểu rõ về các nguyên tắc quản lý mã nguồn và tổ chức kho lưu trữ
- Nắm được kiến thức về các hệ thống SCM phổ biến (Git, Subversion, CVS)
- Nắm được kiến thức về các thuật ngữ Hệ thống Kiểm soát Phiên bản (VCS), Hệ thống Kiểm soát Sửa đổi và Hệ thống Quản lý Mã nguồn (SCM)

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Kho lưu trữ mã nguồn
- Cam kết, Nhánh và Thẻ
- Tính năng, nhánh phát triển và nhánh phát hành
- Kho lưu trữ phụ
- Hợp nhất mã

Linux Professional Institute

Bài 1

Chứng chỉ:	Open Source Essentials
Phiên bản:	1.0
Chủ đề:	056 Cộng tác và Giao tiếp
Mục tiêu:	056.2 Quản lý Mã nguồn
Bài học:	1 trên 1

Giới thiệu

Bất kỳ ai đã từng hiệu đính một tài liệu văn bản trong phạm vi nhóm đều hiểu loại công việc cộng tác như vậy có thể phát sinh những vấn đề gì: Phiên bản nào là phiên bản hiện tại? Phiên bản này được lưu ở đâu? Có ai đó hiện đang chỉnh sửa văn bản này không? Ai đã đưa ra những bình luận hoặc thay đổi trên văn bản — khi nào và tại sao? Kết quả của những vấn đề này thường là một cơ số các phiên bản khác nhau của tài liệu và trong trường hợp tệ nhất sẽ là một tập hợp các phiên bản mà không ai thực sự nắm bắt rõ được.

Hãy tưởng tượng một dự án phần mềm với hàng trăm tệp nơi các nhà phát triển từ khắp nơi trên thế giới đang làm việc bằng cách phát triển các tính năng mới, sửa lỗi, tách các phần và phát triển chúng một cách biệt lập, v.v. Sẽ không thể quản lý được một quy trình phát triển như vậy nếu không có các công cụ phù hợp.

Phần mềm đặc biệt dành cho việc *quản lý mã nguồn* (Source Code Management - SCM) — hay còn được gọi là *hệ thống kiểm soát phiên bản* (Version Control Systems - VCS) hoặc *hệ thống kiểm soát sửa đổi* (Revision Control Systems - RCS) — cung cấp một giải pháp để khắc phục hoặc loại bỏ các vấn đề vừa được nêu ở trên.

Trong thế giới phát triển phần mềm, SCM như một trụ cột cơ bản bảo vệ tính toàn vẹn của cơ sở mã. Hãy hình dung nó như một người bảo vệ cần cù và tỉ mỉ theo dõi mọi điều chỉnh và thay đổi được thực hiện đối với mã nguồn theo thời gian.

Hệ thống Quản lý Mã nguồn và Kho lưu trữ

Hệ thống quản lý mã nguồn là trái tim của một dự án phần mềm. Mặc dù các công việc quan trọng cũng diễn ra trên các diễn đàn thảo luận và những nơi khác nhưng hệ thống SCM sẽ đại diện cho toàn bộ lịch sử của dự án cũng như trạng thái hiện tại của nó như một thực thể sống.

Kho lưu trữ (repository) mã nguồn là một loại xưởng kỹ thuật số cho một dự án. Giống như một xưởng vật lý lưu trữ tất cả các công cụ và vật phẩm cần thiết để sản xuất một phôi thành phẩm, một kho lưu trữ mã nguồn cũng sẽ lưu trữ tất cả các tệp, tài liệu và mã liên quan đến một dự án phần mềm. Nó cung cấp một môi trường có cấu trúc để tổ chức và quản lý toàn bộ tài sản của dự án.

Thông tin thường được lưu trữ dưới dạng một cây thư mục. Các hệ thống SCM thường sử dụng mô hình máy khách-máy chủ nơi bất kỳ người dùng nào cũng có thể *tải xuống* (pull) dữ liệu từ kho lưu trữ cũng như *đẩy* (push) dữ liệu lên đó. Hệ thống sẽ theo dõi xem ai đã thực hiện những thay đổi nào và thực hiện vào thời điểm nào nhằm đảm bảo tính minh bạch và trách nhiệm giải trình trong nhóm phát triển.

Hãy tưởng tượng rằng chúng ta đang làm việc trong một dự án với nhiều nhà phát triển và phát hiện ra một lỗi trong mã. Với SCM, chúng ta có thể dễ dàng kiểm tra các thay đổi giữa các phiên bản để xác định chính xác sự thay đổi cụ thể nào đã gây ra lỗi.

Vì hệ thống sẽ ghi nhớ từng phiên bản của các tệp khi chúng thay đổi nên người dùng có thể truy cập vào bất kỳ phiên bản nào trong số các phiên bản này và có thể quay lại các phiên bản trước đó (trong trường hợp xảy ra các thay đổi không chính xác). Vì vậy, kho lưu trữ cũng là một loại lưu trữ cấp quyền truy cập vào mọi thay đổi từng được thực hiện đối với dự án và trạng thái của dự án tại bất kỳ thời điểm nào trong lịch sử của dự án.

Hệ thống SCM có thể tiết kiệm không gian bằng cách theo dõi các thay đổi đối với tệp thay vì lưu trữ toàn bộ tệp mỗi khi có thay đổi. Phương pháp lưu trữ hiệu quả này sẽ đảm bảo được rằng mọi phiên bản trong lịch sử dự án đều có thể truy cập được mà không tiêu tốn quá nhiều tài nguyên.

Có nhiều công cụ (chẳng hạn như công cụ tích hợp liên tục/phân phối liên tục (CI/CD) và thử nghiệm) xoay quanh hệ thống SCM. Nhiều người cũng xây dựng danh tiếng của mình thông qua hệ thống bằng cách làm cho những đóng góp của họ có thể được mọi người nhìn thấy. Do đó, việc quản lý mã nguồn không chỉ là theo dõi các thay đổi mà còn là bảo toàn tính toàn vẹn của cơ sở mã, thúc đẩy sự hợp tác giữa các nhà phát triển và đảm bảo rằng các dự án có thể được phát triển

trong một bối cảnh năng động và luôn thay đổi.

Các hệ thống SCM phổ biến bao gồm Git, Subversion và CVS. Giống như nhiều chức năng phần mềm khác, SCM hiện nay thường được cung cấp bởi các nhà cung cấp chuyên biệt. Nói cách khác, đây chính là Phần mềm dưới dạng Dịch vụ (SaaS) hoặc dịch vụ "đám mây": những người tham gia sẽ lưu phần mềm trong hệ thống cục bộ của họ để quản lý các thay đổi cá nhân nhưng sẽ tải các thay đổi của họ lên kho lưu trữ trung tâm để được hưởng lợi từ các tính năng đám mây thông thường như thời gian hoạt động 24/7, sao lưu và truy cập an toàn.

Hàng triệu nhà phát triển hiện đang sử dụng các dịch vụ đám mây, đặc biệt là GitHub. GitLab là một giải pháp thay thế dựa trên mã nguồn mở của GitHub. Cả GitHub và GitLab đều cho phép mọi người làm việc trong kho lưu trữ đám mây của nhà cung cấp hoặc thiết lập một phiên bản cục bộ của hệ thống SCM. Một lợi thế khi làm việc trên các hệ thống đó là danh tiếng mà người ta có thể đạt được thông qua các hệ thống "đánh giá sao" của họ cho phép người dùng đánh giá công việc của nhau. Các dịch vụ đám mây cũng bổ sung thêm các điểm hấp dẫn như trình theo dõi yêu cầu thay đổi, hệ thống đánh giá, wiki và các diễn đàn thảo luận.

Kho lưu trữ có thể chứa cả dự án cá nhân và doanh nghiệp — nghĩa là chúng không nhất thiết chỉ dành riêng cho mục đích sử dụng của doanh nghiệp. Bất kỳ nhà phát triển nào muốn bắt đầu một dự án phát triển hoặc làm việc trên một dự án mã nguồn mở — sao chép và sửa đổi theo nhu cầu của họ — đều có thể làm như vậy. Trong tất cả các trường hợp này, điều quan trọng là chúng ta phải kiểm soát chặt chẽ những người có thể truy cập vào kho lưu trữ.

Việc hiểu rõ các thuật ngữ xoay quanh vấn đề kiểm soát phiên bản và quản lý mã nguồn là một điều cần thiết để điều hướng việc sử dụng nó. Trong các phần sau, chúng ta sẽ xem xét kỹ hơn một số khái niệm và thuật ngữ này.

Cam kết, Thẻ và Nhánh

Nhà phát triển sẽ tạo một *cam kết* (commit) mỗi lần một thay đổi được đẩy lên kho lưu trữ. Các cam kết sẽ đại diện cho ảnh chụp tức thời của các thay đổi được thực hiện đối với cơ sở mã tại một thời điểm cụ thể. Mỗi cam kết đều sẽ bao gồm các siêu dữ liệu như tên tác giả, dấu thời gian và thông báo mô tả giải thích các thay đổi. Các cam kết sẽ giúp nhà phát triển theo dõi được sự phát triển của cơ sở mã và nắm rõ lịch sử của các thay đổi cụ thể.

Hãy xem xét một nhóm các nhà phát triển đang làm việc trên một ứng dụng web. Mỗi lần họ thực hiện các thay đổi đối với cơ sở mã, họ sẽ tạo một cam kết để ghi lại những thay đổi đó. Ví dụ: một người nào đó thêm một tính năng mới vào ứng dụng có thể tạo một cam kết với thông báo như "Đã thêm tính năng xác thực người dùng" (bằng tiếng Anh). Cam kết này sẽ ghi lại trạng thái của cơ sở mã sau khi tính năng được triển khai.

Khi nhà phát triển sửa một lỗi, thông báo cam kết thường sẽ nhắc đến số (thứ tự) của lỗi trong hệ thống theo dõi lỗi của dự án.

Thẻ (tag) là các tham chiếu được đặt tên cho các cam kết cụ thể. Chúng thường được sử dụng để đánh dấu các điểm quan trọng trong lịch sử dự án (chẳng hạn như các bản phát hành hay các cột mốc). Thẻ cung cấp cho chúng ta một phương thức gắn nhãn và tham chiếu đến các phiên bản quan trọng của cơ sở mã để giúp việc quản lý và điều hướng lịch sử của dự án trở nên dễ dàng hơn.

Nhánh (branch) có thể phát huy tác dụng khi các nhà phát triển cần làm việc trên nhiều tác vụ khác nhau cùng một lúc. Nhánh là các dòng phát triển độc lập tách biệt khỏi cơ sở mã chính. Chúng cho phép các nhà phát triển làm việc trên các tính năng hoặc các bản sửa lỗi một cách riêng biệt mà không ảnh hưởng đến cơ sở mã chính cho đến khi chúng đã sẵn sàng để tích hợp. Nhánh có thể giúp chúng ta tổ chức các công việc phát triển và tạo điều kiện cho sự hợp tác giữa các thành viên trong nhóm.

Ví dụ: nếu một nhà phát triển đang làm việc để thêm một tính năng mới vào ứng dụng trong khi một người khác đang sửa lỗi, mỗi người có thể tạo các nhánh riêng biệt để cô lập các thay đổi của mình. Khi công việc của họ hoàn tất, họ có thể *gộp* các nhánh của mình trở lại cơ sở mã chính (Branches).

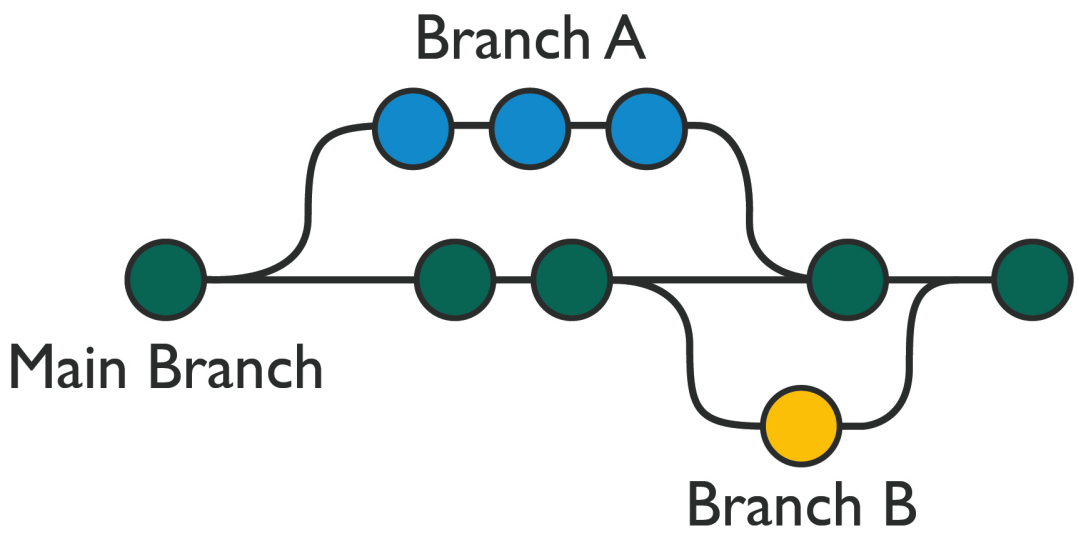

Figure 16. Branches

Khi nhiều người làm việc trên cùng một tệp hoặc đang đưa tệp vào một nhánh khác, đôi khi sẽ xảy ra trường hợp hai người đã thực hiện thay đổi trên cùng một dòng của tệp. Khi người thứ hai cố gắng đưa thay đổi vào, hệ thống sẽ cảnh báo về *xung đột*.

Một số VCS sẽ không cho phép nhà phát triển đưa tệp vào nếu tệp đó có xung đột với phiên bản

hiện tại trong kho lưu trữ. Nhà phát triển sẽ phải kiểm tra phiên bản hiện tại và tìm cách giải quyết xung đột, sau đó mới có thể đưa phiên bản mới của họ vào.

Các hệ thống dựa trên đám mây cho phép *gộp* (merge requests) hoặc *tải yêu cầu* (pull requests). Những yêu cầu này được tạo bởi một nhà phát triển đã làm việc trên một hệ thống hoặc một nhánh cục bộ và tin rằng những thay đổi của họ phù hợp để đưa vào một nhánh khác. Các nhà phát triển nhánh mục tiêu sẽ quyết định xem có chấp nhận yêu cầu hay không.

Kho lưu trữ phụ

Chúng ta sẽ thường gặp phải trường hợp cần tới mã của một dự án độc lập khác (ví dụ như trình phát phương tiện) để phát triển một dự án phần mềm (ví dụ như một trang web phức tạp). Thay vì sao chép một phần hoặc toàn bộ mã của trình phát phương tiện vào dự án, nhiều VCS có cung cấp tính năng *kho lưu trữ phụ* (subrepositories) — hay còn được gọi là *mô-đun con* (submodules).

Trong ví dụ của chúng ta, kho lưu trữ của trình phát phương tiện đã được tích hợp vào kho lưu trữ của trang web dưới dạng một kho lưu trữ phụ. Sau đó, nó xuất hiện dưới dạng một thư mục riêng biệt trong cây thư mục. Điều này có nghĩa là cơ sở mã của trình phát phương tiện hoàn toàn có sẵn nhưng vẫn là độc lập. Nếu cần, thậm chí nó cũng có thể được cập nhật từ kho lưu trữ gốc của trình phát phương tiện.

Khả năng này đã tự chứng minh giá trị của nó trong việc xử lý các dự án phức tạp với nhiều phần phụ thuộc hoặc tích hợp các thư viện và bộ khung của bên thứ ba vào một cơ sở mã. Các mô-đun con sẽ tăng cường tổ chức dự án và tạo điều kiện cho việc cộng tác bằng cách cho phép các nhà phát triển làm việc với các cơ sở mã được kết nối với nhau một cách hiệu quả.

Ứng dụng chung của một Hệ thống Quản lý Kiểm soát Nguồn

Mỗi một cá nhân tham gia dự án sẽ bắt đầu bằng cách tạo một hồ sơ — thường được liên kết với địa chỉ email của riêng cá nhân đó. Một hệ thống dựa trên đám mây sẽ quản lý hồ sơ thông qua các tài khoản giống như các trang mạng xã hội và các tổ chức khác.

Các nhà phát triển mới sẽ trải qua một trình tự như sau khi sử dụng một hệ thống SCM:

1. Cài đặt phần mềm SCM nếu hệ điều hành của người dùng chưa có phần mềm này.
2. Đưa toàn bộ dự án vào hệ thống cục bộ một lần hay còn được gọi là *nhân bản* (cloning).
3. Từ bước này trở đi sẽ là làm việc cục bộ (trong một hoặc nhiều nhánh nếu cần) và đẩy các thay đổi vào kho lưu trữ.
4. Tải phiên bản mới nhất từ kho lưu trữ trước phiên làm việc tiếp theo.

Người quản lý dự án sẽ quyết định việc tin tưởng và cấp quyền truy cập vào kho lưu trữ cho những ai. Các nhà phát triển cao cấp có trách nhiệm quan trọng trong việc quyết định xem các thay đổi do những người đóng góp khác gửi tới đã sẵn sàng để được đưa vào nhánh chính của kho lưu trữ hay chưa.

Trên các hệ thống dựa trên đám mây, một chủ sở hữu dự án có thể kiểm soát khả năng truy cập vào kho lưu trữ bằng cách đặt khả năng hiển thị của kho lưu trữ thành công khai hoặc riêng tư. Kho lưu trữ công khai sẽ cấp quyền đọc cho bất kỳ ai trên internet. Tuy nhiên, kho lưu trữ riêng tư sẽ chỉ giới hạn quyền truy cập cho chủ sở hữu, những cá nhân mà chủ sở hữu đã chia sẻ quyền truy cập và các thành viên cụ thể của tổ chức trong trường hợp kho lưu trữ thuộc về tổ chức.

Hãy tưởng tượng có một nhóm các nhà phát triển làm việc trên một trang web thương mại điện tử. Họ quyết định thêm một tính năng mới để cho phép khách hàng theo dõi các đơn hàng của mình. Để triển khai tính năng này, họ đã tạo một *nhánh tính năng* có tên như `order-tracking` nơi họ có thể làm việc trên các thay đổi mã cần thiết mà không ảnh hưởng đến cơ sở mã chính. Sau khi tính năng được hoàn tất và đã qua thử nghiệm, họ sẽ hợp nhất nhánh này vào *nhánh phát triển* chính để tích hợp và thử nghiệm thêm.

Nhánh phát triển chính sẽ đóng vai trò là trung tâm nơi tất cả các tính năng mới được tập hợp lại để thử nghiệm. Ví dụ: nếu nhiều nhà phát triển đang cùng lúc làm việc trên các tính năng khác nhau, họ có thể hợp nhất các nhánh tính năng của mình vào nhánh phát triển để đảm bảo cho mọi thứ có thể hoạt động trơn tru. Quy trình tích hợp này sẽ giúp xác định và giải quyết mọi xung đột hoặc các vấn đề về tương thích ngay từ đầu.

Khi đến thời điểm phát hành phiên bản mới của trang web thương mại điện tử, nhóm phát triển sẽ tạo một *nhánh phát hành* (chẳng hạn như `v2.0`) từ nhánh phát triển. Họ sẽ tập trung vào việc ổn định cơ sở mã, sửa mọi lỗi xuất hiện ở phút chót và tiến hành thử nghiệm kỹ lưỡng để đảm bảo việc phát hành có thể được suôn sẻ. Khi bản phát hành đã sẵn sàng, mã từ nhánh phát hành sẽ được triển khai vào sản xuất và chu kỳ sẽ bắt đầu lại.

Hệ thống Kiểm soát Phiên bản Chung

Một trong số các hệ thống kiểm soát phiên bản nổi tiếng nhất là Git, Subversion (còn gọi là SVN) và CVS. Tất cả các hệ thống này đều là mã nguồn mở.

Git là một hệ thống kiểm soát phiên bản phân phối được sử dụng rộng rãi trong phát triển phần mềm và các lĩnh vực khác. Khi sử dụng Git, mỗi nhà phát triển sẽ có một bản sao cơ sở mã đầy đủ trên máy tính của họ.

Cách tiếp cận phi tập trung này cho phép các nhà phát triển làm việc ngoại tuyến và cộng tác một cách liền mạch mà không cần dựa vào một máy chủ trung tâm. Điều này có nghĩa là các nhà phát

triển có thể làm việc độc lập trên các tính năng hoặc các bản sửa lỗi khác nhau và hợp nhất các thay đổi lại với nhau một cách liền mạch. Ngay cả khi máy chủ trung tâm đang ngoại tuyến, các nhà phát triển vẫn có thể tiếp tục làm việc và chia sẻ các bản cập nhật với nhau.

Hãy cùng xem xét sự phát triển của hạt nhân Linux ban đầu dựa trên hệ thống kiểm soát phiên bản tập trung có tên là BitKeeper. Khi BitKeeper không còn miễn phí, Linus Torvalds và cộng đồng Linux đã phát triển Git như một giải pháp thay thế phân phối. Quyết định này cho phép chúng ta phát triển phi tuyến tính và xử lý hiệu quả các dự án lớn như nhân Linux. Sự thành công của Git đối với nhân Linux — một dự án cực kỳ phức tạp với hàng nghìn nhà phát triển và vô số nhánh — đã cho thấy sức mạnh và khả năng mở rộng của Git.

Hầu hết các phần mềm phát triển hiện nay đều sử dụng Git. Các dịch vụ SaaS cực kỳ phổ biến cũng đều được xây dựng dựa trên Git.

Git xử lý xung đột bằng cách cung cấp cho nhà phát triển một tệp có cả hai phiên bản của dòng đã thay đổi và đánh dấu rõ ràng phiên bản tệp nơi các dòng đó bắt nguồn. Nhà phát triển sẽ phải quyết định cách giải quyết xung đột và đưa ra một phiên bản nhất quán của tệp.

Subversion (SVN) có lẽ là hệ thống SCM phổ biến nhất trước khi Git được phát minh. Không giống như Git, Subversion được tập trung hóa: lịch sử phiên bản sẽ nằm trên một máy chủ trung tâm. Các nhà phát triển sẽ kết nối với máy chủ này để thực hiện các thay đổi nhằm đảm bảo rằng mọi người đều sẽ làm việc với phiên bản mới nhất của cơ sở mã.

Giả sử chúng ta là thành viên của một nhóm phát triển làm việc trên một dự án sử dụng Subversion. Mỗi lần cần thực hiện thay đổi đối với cơ sở mã, chúng ta sẽ kết nối với máy chủ SVN trung tâm để kiểm tra bản sao đang hoạt động của mã. Điều này sẽ đảm bảo được rằng chúng ta đang làm việc với phiên bản mới nhất của dự án. Sau khi thực hiện thay đổi, chúng ta sẽ tạo cam kết và đẩy chúng trở lại máy chủ và cập nhật các sửa đổi tới kho lưu trữ trung tâm. Quy trình làm việc tập trung sẽ giúp duy trì tính nhất quán và đảm bảo rằng mọi người đều đang làm việc hướng tới cùng một mục tiêu.

Trước Subversion, *CVS* là một hệ thống kiểm soát phiên bản tập trung rất phổ biến. Một số vấn đề về thiết kế của CVS đã dẫn đến việc Subversion được thiết kế và phát triển như một giải pháp thay thế.

Bài tập Hướng dẫn

1. Hãy kể tên ba tính năng cốt lõi của hệ thống SCM.

2. Hãy mô tả khái niệm gắn thẻ trong hệ thống SCM và giải thích lý do tại sao nó lại quan trọng đối với việc quản lý các bản phát hành phần mềm.

3. Sự khác biệt giữa nhánh và kho lưu trữ phụ trong hệ thống SCM là gì?

Bài tập Mở rộng

1. Hãy so sánh Git và Subversion (SVN) trên phương diện kiến trúc và quy trình làm việc của chúng.

2. “Chỉ mục chính” (index) hoặc “vùng dựng” (staging area) trong Git là gì?

3. Hãy phác hoạ chiến lược phân nhánh dựa trên trục (trunk) trong Git.

4. Hệ thống SCM nào sau đây là mã nguồn mở?

Git	
Mercurial	
Subversion	
GitHub	
Bitbucket	
GitLab	

Tóm tắt

Bài học này đã giải thích vai trò trung tâm của hệ thống quản lý mã nguồn trong phát triển phần mềm hiện đại. Chúng ta đã học về các thuật ngữ cơ bản cũng như cách sử dụng hệ thống bao gồm kho lưu trữ, nhánh, thẻ và tính năng hợp nhất.

Đáp án Bài tập Hướng dẫn

1. Hãy kể tên ba tính năng cốt lõi của hệ thống SCM.
   - Ghi nhật ký thay đổi đối với mã nguồn
   - Quản lý các truy cập (đồng thời) vào mã nguồn của các nhà phát triển
   - Khả năng khôi phục bất kỳ trạng thái phát triển nào của các tệp hoặc toàn bộ dự án
2. Hãy mô tả khái niệm gắn thẻ trong hệ thống SCM và giải thích lý do tại sao nó lại quan trọng đối với việc quản lý các bản phát hành phần mềm.

   Gắn thẻ là hoạt động gán nhãn mô tả hoặc tên cho các cam kết cụ thể trong cơ sở mã; nó đóng vai trò là điểm đánh dấu cho các điểm quan trọng trong lịch sử của dự án (chẳng hạn như các bản phát hành hoặc các cột mốc). Các thẻ này sẽ cung cấp một phương thức thuận tiện để tham chiếu đến các phiên bản cụ thể của cơ sở mã và theo dõi sự phát triển của dự án theo thời gian.

3. Sự khác biệt giữa nhánh và kho lưu trữ phụ trong hệ thống SCM là gì?

   Nhánh là một dòng phát triển song song trong một dự án (chẳng hạn như để sửa lỗi hoặc phát triển các tính năng mới) thường được hợp nhất trở lại nhánh phát triển chính ngay khi nhiệm vụ được hoàn thành. Kho lưu trữ phụ hoặc mô-đun con là một dự án độc lập có kho lưu trữ được tích hợp vào một dự án để truy cập vào cơ sở mã của dự án đó. Kho lưu trữ phụ sẽ xuất hiện dưới dạng một thư mục trong cây thư mục của dự án và luôn giữ trạng thái độc lập.

Đáp án Bài tập Mở rộng

1. Hãy so sánh Git và Subversion (SVN) trên phương diện kiến trúc và quy trình làm việc của chúng.

   Git là một hệ thống kiểm soát phiên bản phân tán (DVCS) cho phép mỗi nhà phát triển có một bản sao hoàn chỉnh của cơ sở mã và làm việc ngay cả khi ngoại tuyến trên mã nguồn. Git đã trở nên phổ biến rộng rãi trong số các nhà phát triển do tốc độ, tính linh hoạt và khả năng phân nhánh và hợp nhất mạnh mẽ của nó. SVN là VCS tập trung với lịch sử phiên bản nằm trên một máy chủ trung tâm. Nó vẫn phổ biến trong một số môi trường doanh nghiệp do bản chất tập trung và bộ tính năng hoàn thiện của nó.

2. “Chỉ mục chính” (index) hoặc “vùng dựng” (staging area) trong Git là gì?

   Chỉ mục hoặc vùng dựng là một lớp trung gian giữa bản sao làm việc cục bộ của dự án và phiên bản hiện tại trên máy chủ. Đây là tệp lưu trữ tất cả thông tin cho lần cam kết tiếp theo của người dùng.

3. Hãy phác hoạ chiến lược phân nhánh dựa trên trunk trong Git.

   Phát triển dựa trên trục là chiến lược nhấn mạnh việc tích hợp thường xuyên các thay đổi vào cơ sở mã chính (trục). Các nhà phát triển sẽ làm việc trên các nhánh tính năng tồn tại trong thời gian ngắn và hợp nhất chúng vào trunk nhiều lần trong ngày để đảm bảo việc tích hợp liên tục và phản hồi nhanh chóng.

4. Hệ thống SCM nào sau đây là mã nguồn mở?

Git	X
Mercurial	X
Subversion	X
GitHub	
Bitbucket	
GitLab	X

056.3 Công cụ Giao tiếp và Cộng tác

Tham khảo các mục tiêu LPI

Open Source Essentials version 1.0, Exam 050, Objective 056.3

Khối lượng

2

Các lĩnh vực kiến thức chính

- Hiểu rõ về các công cụ chung để giao tiếp
- Hiểu rõ về các phương pháp chung để nắm bắt và bảo mật kiến thức
- Hiểu rõ về các công cụ chung để quản lý và xuất bản thông tin
- Hiểu rõ về các loại tài liệu chung
- Hiểu rõ các tính năng cộng tác chung của các nền tảng quản lý mã nguồn
- Hiểu rõ các khái niệm về ứng dụng và nền tảng được quản lý độc lập, liên kết và tập trung

Sau đây là danh sách một phần các tệp, thuật ngữ và tiện ích được sử dụng

- Các chương trình nhắn tin tức thời
- Nền tảng trò chuyện
- Danh sách gửi thư
- Bản tin
- Trình theo dõi sự cố và trình theo dõi lỗi
- Báo cáo lỗi
- Yêu cầu hợp nhất và yêu cầu kéo
- Hệ thống hỗ trợ và gửi phiếu yêu cầu

- Wiki
- Hệ thống Quản lý Tài liệu (DMS)
- Trang web tài liệu
- Trang web sản phẩm
- Hệ thống Quản lý Nội dung (CMS)
- Tài liệu kiến trúc
- Tài liệu người dùng
- Tài liệu quản trị viên
- Tài liệu nhà phát triển

Linux Professional Institute

Bài 1

Chứng chỉ:	Open Source Essentials
Phiên bản:	1.0
Chủ đề:	056 Cộng tác và Giao tiếp
Mục tiêu:	056.3 Công cụ Giao tiếp và Cộng tác
Bài học:	1 trên 1

Giới thiệu

Rất nhiều dự án mã nguồn mở đều nhận được sự đóng góp từ nhiều cá nhân tích cực trên toàn thế giới. Sự hợp tác sẽ chủ yếu diễn ra trong "không gian ảo" và những người đóng góp thường phân bổ trên khắp các quốc gia, châu lục và múi giờ. Thêm vào đó, họ cũng sẽ nói các ngôn ngữ khác nhau. Điều này có nghĩa là chúng ta có thể ở bất kỳ đâu trên thế giới và đóng góp cho một dự án mã nguồn mở cho dù quốc gia hay ngôn ngữ của chúng ta có là gì đi chăng nữa!

Sự đa dạng này khiến việc đóng góp cho một dự án mã nguồn mở trở nên cực kỳ có giá trị vì chúng ta có thể từ đó mở rộng phạm vi làm việc và học được rất nhiều điều điều; tuy nhiên, nó cũng đồng thời có thể khiến cho việc giao tiếp và phối hợp trở nên tương đối khó khăn. Giao tiếp hiệu quả và năng suất chính là chìa khóa cho sự thành công và tính bền vững của bất kỳ một dự án mã nguồn mở nào. Để đảm bảo việc giao tiếp hiệu quả, các dự án mã nguồn mở đã tạo ra các cấu trúc và sử dụng nhiều công cụ khác nhau nhằm tăng cường hiệu quả hợp tác cũng như giúp việc đóng góp trở nên dễ dàng hơn.

Một thách thức khác nữa là các dự án mã nguồn mở — thường do các tình nguyện viên đứng sau — phải đối mặt với một số biến động về sự tham gia của các cá nhân. Cuộc sống và sở thích của mọi người đều sẽ thay đổi, và một số người có thể sẽ không còn hứng thú hoặc đơn giản chỉ là

không có thời gian. Chúng ta có thể đã đóng góp cho một dự án mã nguồn mở trong nhiều năm, nhưng khi chúng ta thay đổi công việc, kết hôn hoặc sinh con, rất có thể chúng ta sẽ không còn đủ thời gian để làm các công việc tình nguyện nữa.

Do đó, ngoài việc cung cấp phương tiện giao tiếp hiệu quả thông qua các công cụ phù hợp, các dự án mã nguồn mở cũng sẽ cần phải đảm bảo việc bảo tồn và chia sẻ kiến thức để tránh tình trạng "bình mới, rượu cũ". Việc bảo tồn thông tin cũng giúp những người đóng góp có thể học hỏi từ những sai lầm trong quá khứ của những người đi trước và tránh mắc phải những lỗi tương tự.

Bài học này sẽ trình bày về các công cụ chung hỗ trợ cho việc cộng tác trong một dự án mã nguồn mở cũng như giới thiệu về cách giao tiếp trong cộng đồng quốc tế. Chúng ta sẽ biết được rằng có một phương pháp rất dễ dàng để mọi người đều có thể thực hiện những đóng góp đầu tiên của mình!

Một ví dụ về giao tiếp và chia sẻ thông tin đáng được nhắc tới là *LibreOffice*. LibreOffice là một bộ công cụ năng suất văn phòng mã nguồn mở có sẵn tới hơn một trăm ngôn ngữ. Người dùng cuối của nó có thể là từ những người dùng gia đình thông thường đến các văn phòng chính phủ và tập đoàn lớn. Tương tự, nó cũng có một số lượng người đóng góp rất đa dạng — không chỉ là các nhà phát triển mà còn cả những người dân bản địa hóa, tác giả tài liệu, người tiếp thị, kỹ sư đảm bảo chất lượng, quản trị viên cơ sở hạ tầng, nhà thiết kế đồ họa và UX và hơn thế nữa.

Nói cách khác: đối với LibreOffice cũng như nhiều dự án mã nguồn mở khác, chúng ta có thể đưa các kỹ năng và thiên phú của bản thân vào các lĩnh vực mà chúng ta cảm thấy phù hợp. Chúng ta không nhất thiết phải là người được đào tạo về kỹ thuật hoặc là một nhà phát triển — chúng ta cũng có thể đưa vào sự sáng tạo, tài năng nghệ thuật hoặc kỹ năng ngôn ngữ của mình. Dự án này cũng đã ra mắt một trang web hiển thị các lĩnh vực đóng góp khác nhau: https://whatcanidoforlibreoffice.org. Vì những lý do này, LibreOffice là một ví dụ minh họa rất hay về nhiều khía cạnh của công tác cộng đồng.

Các Phương thức Giao tiếp

Trước khi đi sâu vào chi tiết về các công cụ giao tiếp, chúng ta cần phải hiểu cách thức giao tiếp nói chung vì những cân nhắc này sẽ ảnh hưởng đến việc lựa chọn công cụ.

Các dự án mã nguồn mở có hai phương thức chính để giao tiếp. Với giao tiếp *đồng bộ*, mọi người sẽ giao tiếp cùng một lúc như trong các cuộc trò chuyện trực tiếp, các cuộc họp video hoặc các cuộc gọi. Với giao tiếp *không đồng bộ*, mọi người sẽ giao tiếp vào những thời điểm khác nhau thông qua email và SMS hoặc thư bưu chính hay fax.

Tuy nhiên, sự phân biệt này không phải lúc nào cũng dễ thực hiện. Ví dụ: một ứng dụng nhắn tin trên điện thoại di động như WhatsApp, Telegram, Signal hoặc Element về mặt kỹ thuật là một

phương thức giao tiếp không đồng bộ. Tuy nhiên, nếu cả hai bên đối tác đều đang trực tuyến cùng một lúc và trả lời ngay lập tức, điều này có nghĩa là họ đang tham gia vào một cuộc trò chuyện trực tiếp — tức giao tiếp đồng bộ.

Ví dụ này cho thấy một phần của giao tiếp cũng phụ thuộc vào cách mọi người sử dụng công cụ cũng như kỳ vọng của họ. Mỗi nhiệm vụ có thể sẽ cần tới một bộ công cụ giao tiếp khác nhau. Các phần sau sẽ giải thích chi tiết về những vấn đề này.

Giao tiếp Đồng bộ

Giao tiếp đồng bộ là một cách giao tiếp rất hiệu quả nhưng đồng thời cũng đòi hỏi rất cao. Nó đưa mọi người lại gần với nhau trong cùng một thời điểm và một không gian vật lý hoặc ảo.

Một cuộc họp trực tiếp là một phương pháp lý tưởng để giao tiếp một cách tương tác thay vì gửi những email dài dòng có nguy cơ gây hiểu lầm. Giả sử chúng ta đang muốn tìm hiểu thêm về một dự án mã nguồn mở và làm quen với mọi người trong cộng đồng. Email và trang web có thể giúp giới thiệu và khiến cho rào cản gia nhập giảm bớt; tuy nhiên, một ấn tượng ban đầu thực sự tốt sẽ được hình thành khi chúng ta thực sự có thể nói chuyện trực tiếp với ai đó — chính những tương tác trực tiếp này thường sẽ khiến mọi người thích thú hơn về một dự án mã nguồn mở và làm họ nảy sinh muốn đóng góp cho dự án.

Ngoài việc giúp mọi người hiểu nhau hơn, các cuộc họp đồng bộ cũng rất hữu dụng khi chúng ta cần thảo luận mọi việc thông qua tương tác (ví dụ như trong trường hợp xảy ra xung đột hoặc vấn đề, hoặc khi chúng ta phải chia sẻ một thông điệp tiêu cực).

Nhược điểm của nó là các dự án quốc tế sẽ phải đối mặt với một thách thức mà công nghệ không thể vượt qua được: sự khác biệt về múi giờ. Nếu dự án có những người đóng góp sống ở các châu lục khác nhau, sẽ rất khó để tìm được một khung giờ họp phù hợp với tất cả mọi người. Một người nào đó ở Úc có thể vừa mới thức dậy, trong khi một người khác ở châu Âu lại sắp kết thúc một ngày làm việc. Một thách thức nữa là một số người chỉ có thể làm việc vào ban đêm hoặc cuối tuần, trong khi những người khác lại thích làm việc trong giờ hành chính.

Ngoài ra, không phải ai cũng có thể nói tiếng Anh trôi chảy, và hiện vẫn chưa có công cụ dịch thuật trực tiếp nào có thể được truy cập một cách rộng rãi. Điều này cũng tạo ra thêm một rào cản khác nữa.

Mặc dù vậy, các cuộc họp video là một trong những công cụ giao tiếp thường xuyên nhất trong các dự án mã nguồn mở. Dự án LibreOffice — được dùng làm ví dụ cho bài học này — có các cuộc họp trực tuyến thường xuyên cho cộng đồng của mình (ví dụ như các nhà phát triển, tiếp thị, cơ sở hạ tầng, đảm bảo chất lượng, trải nghiệm người dùng và thiết kế).

Giao tiếp không Đồng bộ

Giao tiếp không đồng bộ cho phép chúng ta tham gia thảo luận vào thời điểm và tốc độ thuận tiện đối với bản thân. Ví dụ được biết đến nhiều nhất có lẽ chính là email: chúng ta có thể trả lời một email bất cứ khi nào chúng ta muốn dù là sau một phút, một tiếng hay thậm chí là một ngày.

Đối với giao tiếp không đồng bộ, nội dung thường ở dưới dạng chữ viết — điều này mở ra tính năng dịch máy. Điều này có thể giúp chúng ta đọc và hiểu các tin nhắn được viết bằng các ngôn ngữ khác với ngôn ngữ mẹ đẻ của mình. Chúng ta thậm chí cũng có thể dịch ngược lại câu trả lời của chính mình.

Ngoài ra, nội dung khi được viết ra có thể sẽ giúp chúng ta dễ dàng ghi nhớ kiến thức và tạo tài liệu hơn. Giả sử chúng ta muốn viết một tài liệu hỗ trợ về cách sử dụng một tính năng cụ thể của phần mềm: nếu chúng ta giải thích cho người dùng qua điện thoại, việc chuyển lời nói của mình thành một trang tài liệu phù hợp sẽ khó hơn nhiều so với khi chúng ta giải thích quy trình bằng một văn bản.

Giao tiếp nội bộ và Giao tiếp bên ngoài

Một lưu ý cuối cùng nhưng cũng không kém phần quan trọng: giao tiếp cũng phụ thuộc vào người tiếp nhận giáo tiếp — tức là cả nội bộ lẫn bên ngoài. Cách viết một ghi chú kỹ thuật nội bộ dành cho các quản trị viên hệ thống sẽ khác với cách viết một thông cáo báo chí được gửi đến hàng trăm nhà báo. Tuy nhiên, hãy nhớ rằng, do bản chất của một dự án mã nguồn mở, các giao tiếp thường không được định sẵn là dành riêng cho một đối tượng công chúng vẫn sẽ có thể được hiển thị một cách công khai — ví dụ như trong kho lưu trữ danh sách gửi thư (trong trường hợp của LibreOffice sẽ là https://listarchives.documentfoundation.org/).

Công cụ Giao tiếp

Với kiến thức về những khía cạnh khác nhau về giao tiếp này, trong các phần sau, chúng ta sẽ cùng tìm hiểu về một số công cụ giao tiếp thường được sử dụng trong một dự án mã nguồn mở.

Email, Danh sách gửi thư và Bản tin

Email chính là một trong những công cụ "kinh điển" đầu tiên mà chúng ta sẽ gặp khi tham gia vào một dự án mã nguồn mở. Có nhiều dự án chạy các *danh sách gửi thư* — về cơ bản là các danh sách phân phối dành cho email: chỉ với một email, chúng ta có thể tiếp cận tới hàng trăm hoặc thậm chí là hàng nghìn người đăng ký theo dõi quan tâm đến các chủ đề cụ thể.

Danh sách gửi thư là một trong những công cụ lâu đời nhất được biết đến trong bất kỳ một dự án mã nguồn mở nào và được sử dụng để phối hợp nội bộ dự án cũng như để tương tác với người

dùng. Nếu có câu hỏi về phần mềm hoặc muốn báo cáo lỗi trong chương trình, khả năng cao là sẽ có một danh sách gửi thư cho phép chúng ta làm điều này. Ví dụ: cộng đồng LibreOffice cung cấp nhiều danh sách gửi thư quốc tế và địa phương cho nhiều chủ đề khác nhau (https://www.libreoffice.org/get-help/mailing-lists/) từ hỗ trợ người dùng đến các thảo luận và phối hợp cơ sở hạ tầng của nhà phát triển (Trang web danh sách gửi thư của LibreOffice).

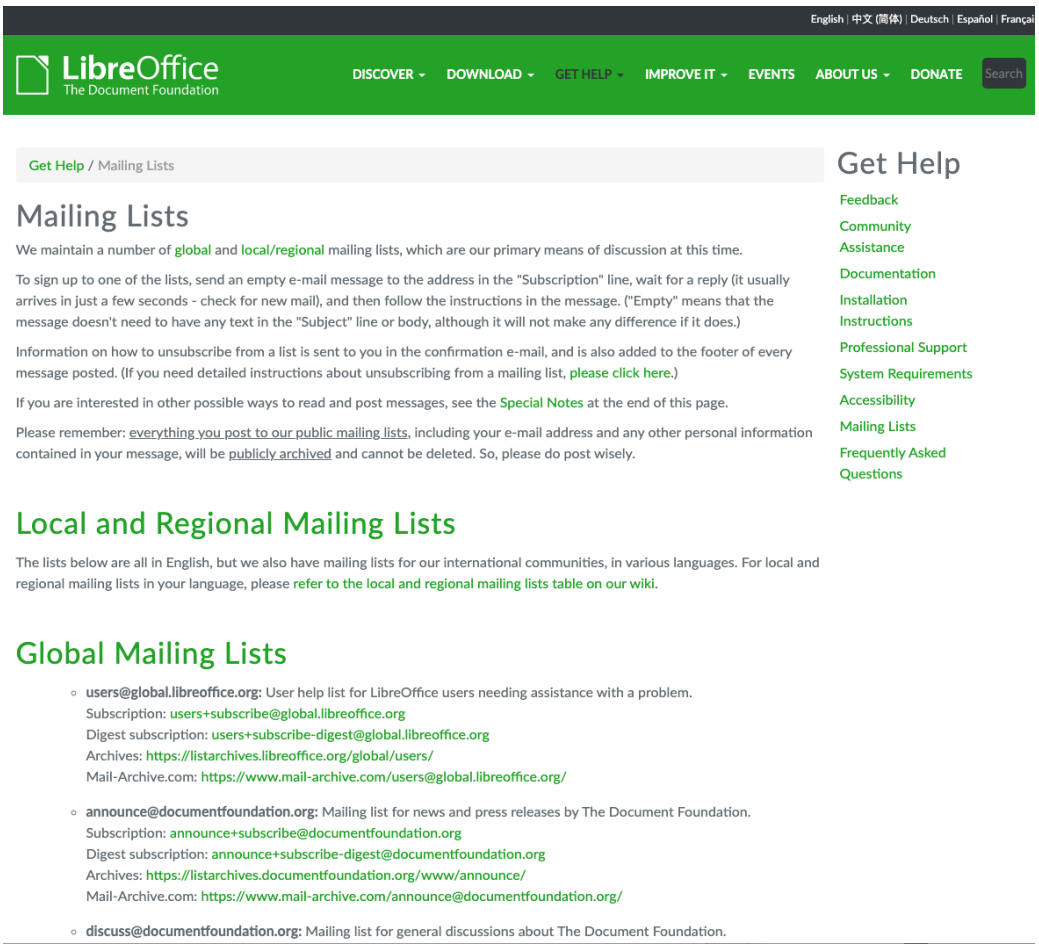

Figure 17. Trang web danh sách gửi thư của LibreOffice

Mỗi lá thư được gửi đi thường cũng sẽ được lưu trữ trong *kho lưu trữ danh sách gửi thư* công khai nhằm mục đích tham khảo trong tương lai. Một khi đã được gửi, thư sẽ không thể dễ dàng bị xóa đi. Có một câu nói rất phổ biến là “Internet không bao giờ "forget" (quên)”. Do đó, chúng ta nên cẩn thận với những gì mình viết vì khả năng cao là chúng ta sẽ không thể rút lại chúng. Một số người để địa chỉ hoặc số điện thoại cá nhân trong chữ ký hoặc gửi các tài liệu mật dưới dạng tệp đính kèm — đây là những ví dụ về các hành vi mà chúng ta nên tránh.

Một nhược điểm của danh sách gửi thư chính là việc quản lý chúng trong các chương trình thư không phải lúc nào cũng đơn giản. Chúng ta sẽ cần tới thứ được gọi là *bộ lọc* dựa trên các thành phần thư cụ thể (ví dụ như tiền tố trong chủ đề thư). Những thao tác cần tới sự tinh tế trong việc quản lý một số lượng lớn email có thể sẽ là một rào cản đối với những người dùng thiếu kinh

nghiệm — đặc biệt là nếu thư của chúng ta là thư gửi một lần. Do đó, ngày càng có nhiều dự án chuyển hướng sang diễn đàn thảo luận mà chúng ta sẽ sớm cùng tìm hiểu sau đây.

Bản tin chính là một hình thức đặc biệt của danh sách gửi thư. Nếu muốn cập nhật những diễn biến mới nhất của dự án và nhận thông báo về các bản phát hành phần mềm mới, chúng ta có thể đăng ký nhận bản tin và nhận email về những sự kiện quan trọng.

Diễn đàn Thảo luận

Ngoài việc phải quản lý danh sách gửi thư trong ứng dụng email, một nhược điểm khác của email chính là ngày càng có nhiều người — đặc biệt là thế hệ trẻ --không còn thích giao tiếp qua email nữa. Vì nhiều lý do khác nhau, ngày càng có nhiều dự án mã nguồn mở chuyển hướng việc giao tiếp của họ sang *diễn đàn thảo luận*. Ý tưởng này nhìn chung cũng khá giống với email: mỗi diễn đàn sẽ có nhiều danh mục khác nhau với các đề tài cụ thể để mọi người cùng tham gia thảo luận — các danh mục này được gọi là các *chủ đề*. Tương tự như email, trên diễn đàn, chúng ta có thể liên hệ với dự án mã nguồn mở và phối hợp các hoạt động, đưa ra các đề xuất về hướng đi của dự án cũng như báo cáo lỗi với tư cách là người dùng cuối.

Mọi thứ được đăng lên một diễn đàn thường sẽ được hiển thị cho toàn bộ công chúng nhìn thấy giống như trong danh sách gửi thư; nhưng điểm khác biệt là ở chỗ, tùy thuộc vào cấu hình của diễn đàn, nội dung cũng có thể sẽ được chỉnh sửa hoặc xóa đi sau đó. Khả năng sử dụng của diễn đàn — đặc biệt là đối với người dùng thiếu kinh nghiệm — thường tốt hơn so với danh sách gửi thư. Dự án LibreOffice đã bắt đầu chuyển đổi một số danh sách gửi thư của mình thành các diễn đàn (https://community.documentfoundation.org/) và kể từ đó đã nhận thấy sự gia tăng trong việc tham gia thảo luận.

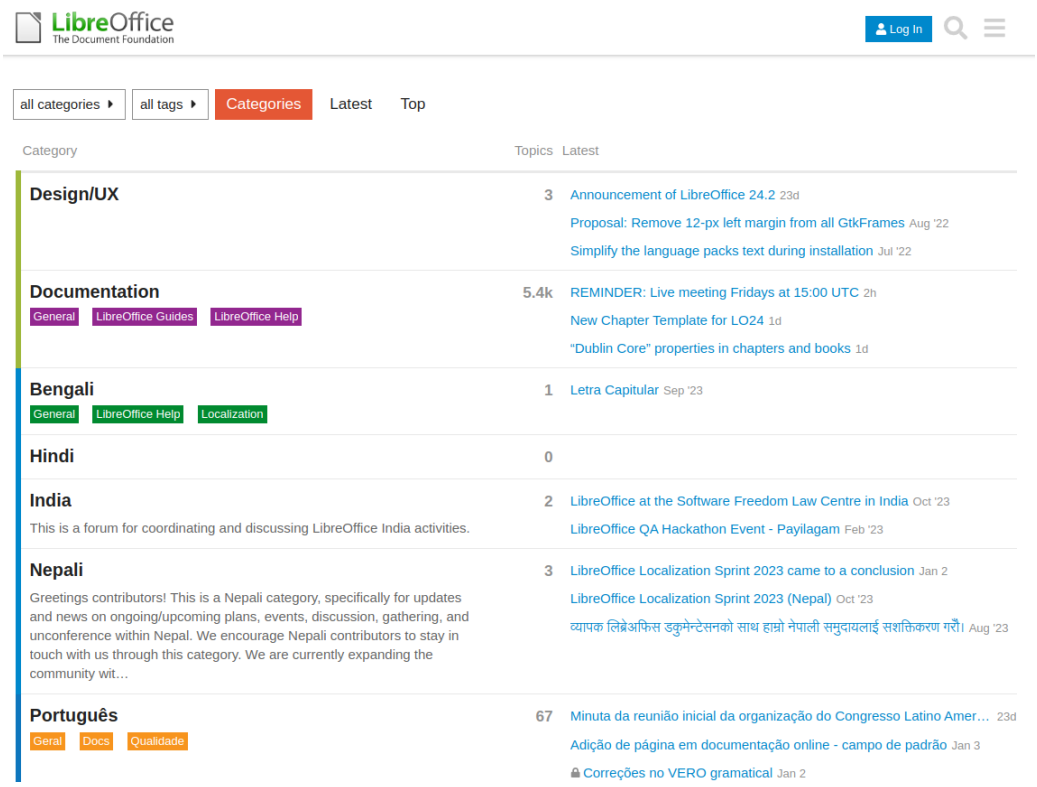

Figure 18. Trang web diễn đàn LibreOffice

Tin nhắn tức thời và Nền tảng Trò chuyện

Một cách khác để liên lạc với cộng đồng mã nguồn mở chính là thông qua tin nhắn tức thời và các nền tảng trò chuyện. Với sự gia tăng của các công cụ phổ biến như WhatsApp, Telegram, Signal và Matrix, hầu như mọi người đều đã cài đặt một trong những ứng dụng phổ biến này trên thiết bị của họ. Điều này khiến cho rào cản gia nhập giảm đi rất nhiều. Tin nhắn tức thời cũng phổ biến hơn nhiều đối với thế hệ trẻ so với email hoặc diễn đàn. Do đó, không có gì ngạc nhiên khi nhiều dự án mã nguồn mở hiện nay đều áp dụng chúng.

Người tham gia trò chuyện sẽ nhập tin nhắn được gửi đến tất cả những người tham gia khác tương tự như email. Tùy thuộc vào nền tảng trò chuyện, tin nhắn có thể trở nên phong phú bằng các định dạng, đồ họa và tệp đính kèm khác nhau.

Về mặt cấu trúc, các ứng dụng tin nhắn cũng được tổ chức tương tự như các diễn đàn hoặc email. Có một số *nhóm* hoặc *kênh* có sẵn để chúng ta có thể tham gia thảo luận về các chủ đề mà mình quan tâm. Tin nhắn thường có thể được chỉnh sửa hoặc xóa và thường cũng có các kênh chỉ sử dụng để thông báo có chức năng tương tự như các bản tin email.

Một nhược điểm của các ứng dụng nhắn tin tức thời là mọi người thường cài đặt chúng trên điện thoại của họ để có thể nhận được thông báo về mọi tin nhắn được gửi. Điều này có thể nhanh chóng gây ra tình trạng thông tin hoặc "mệt mỏi vì thông báo". Tuy nhiên, với một cấu hình phù

hợp, chúng ta có thể kiểm soát được những thông báo này.

Một nhược điểm khác nữa là nhiều ứng dụng nhắn tin đều là sản phẩm độc quyền và nằm trong tay một nhà cung cấp. Điều này làm cho việc lưu giữ thông tin lâu dài trở nên phức tạp hơn vì dữ liệu không thể được truy cập một cách miễn phí.

Giao tiếp độc lập, liên kết và tập trung

Các dự án mã nguồn mở hoạt động trong môi trường mở dựa trên các tiêu chuẩn mở và các công cụ mở. Do đó, điều quan trọng là chúng ta phải hiểu được cách các công cụ khác nhau được thiết kế về phương diện khả năng tương tác. Chúng ta có thể chia các lựa chọn ra thành ba loại chính.

Một nền tảng *độc lập* sẽ chạy biệt lập cho riêng một cộng đồng. Các diễn đàn hoặc wiki là một ví dụ và chúng thường không được kết nối với các tiến trình của các dự án khác.

Các hệ thống *phi tập trung* hoặc *liên kết* chạy riêng cho từng cộng đồng nhưng lại có thể kết nối với nhau. Email chính là một ví dụ vì máy chủ email cục bộ có thể gửi email đến bất kỳ một máy chủ email nào khác trên thế giới. Các ví dụ khác là Nextcloud và ownCloud có thể "liên kết" chia sẻ tệp với các máy chủ khác hay dịch vụ nhắn tin Element nơi chúng ta có thể giao tiếp với người dùng của các máy chủ khác. Các nguyên tắc tương tự cũng áp dụng cho mạng xã hội Mastodon.

Cả các nền tảng độc lập và phân tán đều có một lợi thế lớn: dự án mã nguồn mở vẫn sẽ kiểm soát toàn bộ nội dung và các chức năng. Mọi kiến thức được lưu trữ trong hệ thống như vậy vẫn là tài sản của cộng đồng mã nguồn mở và sẽ không bị bên thứ ba quản lý.

Mặt khác, một hệ thống *tập trung* sẽ được điều hành bởi một nhà cung cấp và không tương tác với bên thứ ba. Ví dụ điển hình chính là các mạng xã hội như Facebook hoặc Instagram hay các ứng dụng nhắn tin như WhatsApp hoặc Telegram. Tất cả các nội dung sẽ được lưu trữ trên máy chủ của nhà cung cấp bên ngoài và sẽ tuân theo các điều khoản và điều kiện của họ.

Nếu đang hoạt động trong một cộng đồng mã nguồn mở, chúng ta có thể liên lạc với nhau bằng cả ba tùy chọn này. Các hệ thống tập trung rất hữu dụng trong việc tiếp cận mọi người vì chúng rất phổ biến và có một lượng người dùng lớn. Tuy nhiên, đối với công việc thực tế trong dự án, một hệ thống liên kết hoặc độc lập do cộng đồng kiểm soát sẽ là lựa chọn tốt nhất.

Công cụ cộng tác

Sự khác biệt giữa các công cụ giao tiếp và công cụ cộng tác không phải lúc nào cũng dễ thấy. Đối với mục đích của bài học này, mục tiêu chính của các công cụ giao tiếp là cho phép những người tham gia khác nhau giao tiếp với nhau; còn mục tiêu chính của các công cụ cộng tác là giúp mọi người làm việc cùng nhau.

Các công cụ cộng tác phù hợp sẽ cho phép chúng ta lưu trữ tệp, cộng tác trong thời gian thực trên các tài liệu, theo dõi các phiên bản tài liệu và thay đổi giữa các bản phát hành phần mềm, v.v. Trong khi đó, email hoặc các diễn đàn có thể đóng vai trò là nơi lưu trữ nhằm mục đích chung, các công cụ chuyên dụng sẽ giúp kiến thức có thể trở nên dễ truy cập hơn và việc cộng tác trở nên hiệu quả hơn.

Nói cách khác, đây là những công cụ chuyên dụng dành cho các nhiệm vụ cụ thể. Nếu muốn đóng góp cho một dự án mã nguồn mở, chúng ta sẽ sớm được gặp chúng.

Wikis

Một trong những công cụ cộng tác lâu đời và phổ biến nhất chính là *wiki*. Trở nên phổ biến đặc biệt bởi Wikipedia, một wiki sẽ cho phép nhiều người dùng làm việc cùng nhau trên một trang web được biên soạn từ nhiều tài liệu hoặc "bài viết". Chúng có thể được nhóm thành nhiều danh mục khác nhau, được lọc theo ngôn ngữ và chứa nhiều định dạng, bảng tính và hình ảnh.

Wiki thường được sử dụng như một cơ sở kiến thức nơi ai cũng có thể đóng góp. Nếu muốn đóng góp nội dung cho một dự án mã nguồn mở, việc tham gia vào wiki của họ là một trong những cách đơn giản nhất. Chúng ta có thể lấy nội dung hiện có và dịch sang ngôn ngữ mẹ đẻ của mình, chỉnh sửa và cập nhật các bài viết hiện có hoặc tạo nội dung mới. Trong wiki của dự án LibreOffice (https://wiki.documentfoundation.org), chúng ta có thể tìm thấy các tài liệu tiếp thị, biên bản họp hội đồng quản trị, hướng dẫn cài đặt và lập kế hoạch hội nghị - tất cả đều bằng nhiều ngôn ngữ khác nhau (LibreOffice Wiki).

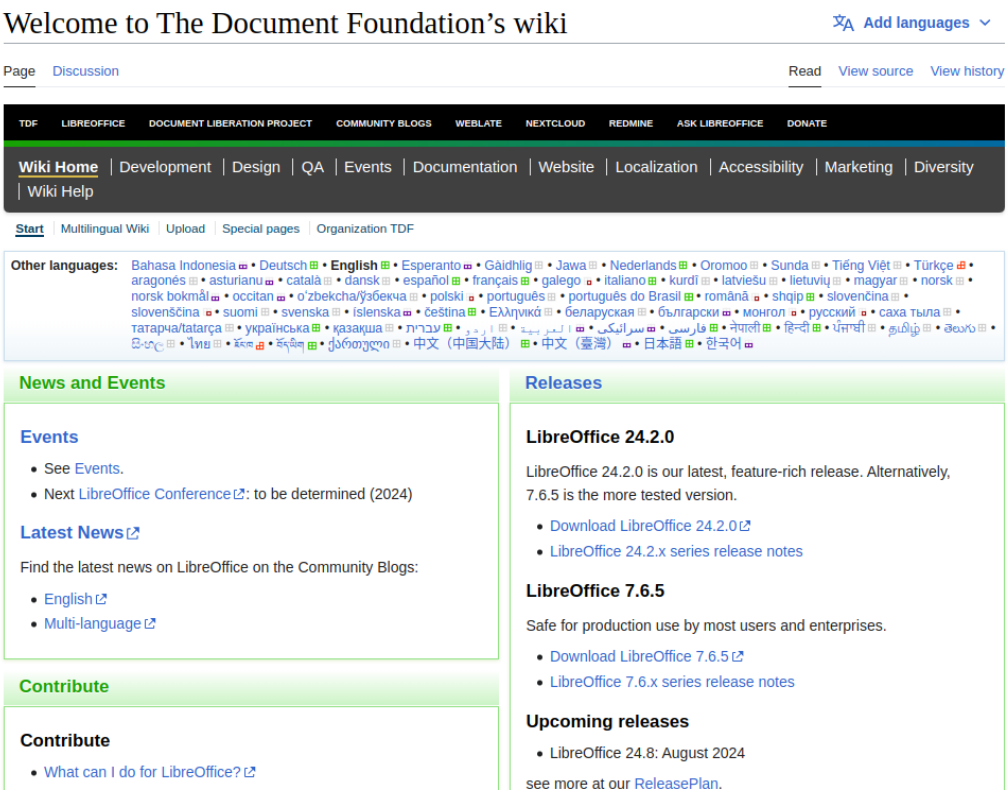

Figure 19. LibreOffice Wiki

Nhiều dự án mã nguồn mở cũng lưu trữ tài liệu và hệ thống trợ giúp tích hợp của chương trình trong wiki. Ngoài ra, các phiên bản cũ của một trang — được gọi là *các bản sửa đổi* — cũng sẽ được lưu trữ cho mục đích tham khảo trong tương lai.

Mặc dù wiki cũng có thể lưu trữ các tệp dự án nhưng chúng ta còn có nhiều công cụ tốt hơn dành cho mục đích này. Chúng ta sẽ cùng tìm hiểu về các công cụ đó trong bài học này.

Trình theo dõi Lỗi và Sự cố

Một công cụ thường được sử dụng khác trong một dự án nguồn mã mở là *trình theo dõi Lỗi* (bug trackers) hay còn được gọi là *trình theo dõi Sự cố* (issue trackers). Nếu phát hiện ra một vấn đề trong phần mềm hoặc muốn đề xuất một tính năng mới, chúng ta có thể chỉ nghĩ đơn giản là gửi đi một email về vấn đề đó. Tuy nhiên, với một công cụ chuyên dụng như trình theo dõi Lỗi, chúng ta có thể cung cấp tất cả các thông tin và các bước cần thiết để tái tạo một vấn đề theo một cách có cấu trúc để giúp các nhà phát triển tái tạo vấn đề một cách dễ dàng hơn. Một báo cáo có cấu trúc như vậy được gọi là một *báo cáo lỗi* (bug report).

Ngoài ra, thông tin sẽ không bị mất và dự án sẽ không bỏ quên vấn đề; nó có thể giao vấn đề cho đúng người và xem xem có bao nhiêu lỗi đang được xử lý hoặc đang trong quá trình sửa chữa.

Dự án LibreOffice cung cấp một công cụ theo dõi lỗi mà mọi người đều có thể đóng góp (https://bugs.documentfoundation.org).

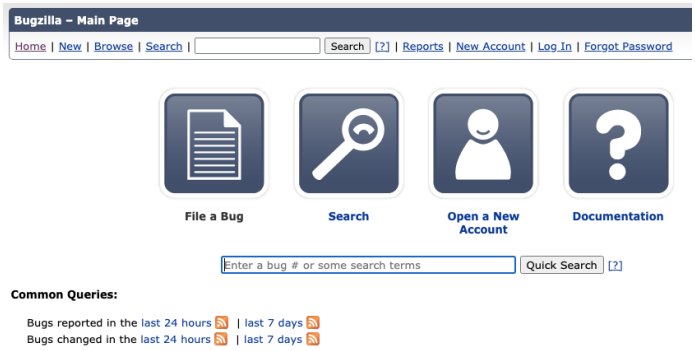

Figure 20. LibreOffice bug tracker

Bộ phận Trợ giúp và Hệ thống Gửi Phiếu Yêu cầu

Một công cụ tương tự chính là *bộ phân trợ giúp* (helpdesk) hoặc *hệ thống gửi phiếu yêu cầu* (ticketing system). Trọng tâm của nó không tập trung vào việc báo cáo các vấn đề về phần mềm mà là giúp người dùng giải quyết mọi loại vấn đề và yêu cầu (chẳng hạn như đối với trang web của dự án).

Bộ phận trợ giúp cổ điển là một đường dây nóng dành cho dịch vụ khách hàng. Khách hàng sẽ gọi điện và báo cáo sự cố và sau đó sự cố này sẽ được chuyển thành một chiếc *vé*. Quy trình làm việc của một hệ thống bộ phận trợ giúp thường xoay quanh các ưu tiên, mức độ leo thang và thời gian phản hồi.

Không phải tất cả các cộng đồng mã nguồn mở đều cung cấp một hệ thống như vậy, nhưng rất nhiều công ty thương mại đều có dịch vụ này. Ví dụ như đối với dự án LibreOffice, có một hệ thống gửi phiếu yêu cầu cho cơ sở hạ tầng để báo cáo các vấn đề về máy chủ và dịch vụ web.

Hệ thống Quản lý Nội dung (CMS)

Một công cụ quan trọng khác trong việc cộng tác là *hệ thống quản lý nội dung* (CMS). Đúng như tên gọi, nó giúp chúng ta quản lý nội dung, đặc biệt là đối với các trang web. Nếu bạn có nhu cầu đóng góp vào nội dung hoặc thiết kế trang web của một dự án, hãy làm quen với CMS của họ.

Tương tự như wiki, hệ thống CMS sẽ giúp cấu trúc nội dung theo danh mục và ngôn ngữ. Chúng thường cung cấp một trình soạn thảo WYSIWYG (“what you see is what you get” - những gì bạn thấy là những gì bạn sẽ nhận được) và nhúng mọi thứ vào một mẫu phù hợp: tiêu đề, tiêu đề phụ, bố cục trang và mục menu sẽ được thực hiện tự động; do đó, chúng ta có thể tập trung hoàn toàn vào nội dung. Ngoài ra, trong trường hợp phải thiết kế lại trang, nội dung sẽ không biến mất mà sẽ được điều chỉnh theo mẫu mới.

Hệ thống Quản lý Tài liệu (DMS)

Chúng ta không được nhầm lẫn giữa hệ thống quản lý tài liệu với hệ thống quản lý nội dung. Trong khi một CMS được thiết kế để hiển thị nội dung theo mẫu được xác định trước (chẳng hạn như để trình bày trang web) thì DMS lại được sử dụng để quản lý các tài liệu như hợp đồng, hóa đơn, biên lai, tin nhắn email và tất cả các loại thư từ khác.

Một điểm khác biệt nữa là CMS thường được sử dụng để trình bày công khai, trong khi DMS chủ yếu được sử dụng để quản lý các tài liệu nội bộ.

Với tư cách là người đóng góp cho một dự án mã nguồn mở, chúng ta sẽ ít có khả năng "chạm trán" với hệ thống quản lý tài liệu vì hệ thống này thường dành riêng cho các vai trò cụ thể như kế toán hoặc pháp lý.

Quản lý Mã nguồn (SCM)

Nếu là một nhà phát triển trong một dự án mã nguồn mở, một trong những công cụ chính mà chúng ta sẽ sớm gặp phải là nền tảng quản lý mã nguồn. Tương tự như wiki dành cho các tác giả tài liệu, SCM được các nhà phát triển phần mềm sử dụng để cùng nhau làm việc trên mã.

Chúng ta có các SCM "cổ" như CVS và Subversion; tuy nhiên, hiện nay, Git lại chủ yếu được sử dụng bởi cộng đồng LibreOffice (https://git.libreoffice.org/). Các công cụ này có sẵn cả trên dòng lệnh lẫn giao diện đồ họa để chúng ta có thể tương tác một cách dễ dàng hơn, đặc biệt là đối với người mới bắt đầu.

Các nền tảng quản lý mã nguồn sẽ theo dõi các phiên bản khác nhau của từng tệp, xử lý các thay đổi và đưa các chỉnh sửa vào mã, ghi lại thông tin người đã thực hiện thay đổi nào và lý tưởng nhất là cung cấp một lịch sử đầy đủ về quá trình phát triển của phần mềm.

Các nhà phát triển có thể “kiểm tra” một trạng thái cụ thể của phần mềm, làm việc cục bộ trên đó — có thể là sửa lỗi hoặc triển khai một tính năng mới — sau đó yêu cầu thay đổi này được thêm vào dòng phát triển chính của phần mềm thông qua tính năng *yêu cầu hợp nhất* (merge request) hay còn được gọi là một *yêu cầu kéo* (pull request). Việc chấp nhận yêu cầu hợp nhất hoặc yêu cầu kéo sẽ “hợp nhất” các thay đổi do một tác giả thực hiện vào với mã chính.

Chúng ta có các nền tảng tập trung (GitHub và GitLab) tích hợp SCM với wiki, trình theo dõi sự cố và các công cụ cộng tác khác.

Nền tảng quản lý mã nguồn không chỉ giới hạn ở mã chương trình. Một ví dụ cho việc này chính là bản thân bài học này cũng được thực hiện thông qua sự hợp tác trong kho lưu trữ Git!

Tài liệu

Chìa khóa cho sự thành công của mọi dự án mã nguồn mở chính là một tài liệu hướng dẫn phù hợp, lý tưởng nhất là được viết bằng nhiều ngôn ngữ. Dự án LibreOffice có cung cấp sách, tài liệu hướng dẫn và các thẻ tham khảo mới nhất (https://documentation.libreoffice.org) cho phần mềm của mình cũng như các trang trợ giúp riêng về các chức năng cụ thể (https://help.libreoffice.org).

Nhìn chung, có rất nhiều loại tài liệu khác nhau mà chúng ta sẽ thảo luận tới ở các phần sau.

Tài liệu Người dùng

Tài liệu hướng dẫn dành cho người dùng cuối là loại tài liệu được biết đến nhiều nhất; chúng sẽ giải thích cách sử dụng phần mềm. Nếu chưa biết về một tính năng hoặc chức năng nào đó trong chương trình, tài liệu hướng dẫn — có thể là từ dưới hình thức các trang trợ giúp riêng lẻ cho đến một cuốn sách hoàn chỉnh — chính là nơi đầu tiên chúng ta cần tham khảo.

Trang web tài liệu - nơi giải thích cách sử dụng phần mềm - thường là một trong những trang web được truy cập nhiều nhất bên cạnh trang web sản phẩm nơi sẽ cung cấp cho chúng ta một cái nhìn tổng quan về phần mềm và cộng đồng sử dụng phần mềm.

Tài liệu Quản trị viên

Để sử dụng trong các môi trường lớn hơn (như trong một công ty), tài liệu quản trị viên sẽ cung cấp tất cả những thông tin có liên quan để triển khai phần mềm trên quy mô lớn hơn. Những thông tin này bao gồm việc kết nối với cơ sở dữ liệu người dùng và lưu trữ tệp, quản lý cấu hình tập trung và xử lý cập nhật.

Tài liệu dành cho Nhà phát triển và Kiến trúc

Một loại tài liệu khác được hướng đến các nhà phát triển và được gọi là tài liệu dành cho nhà phát triển hoặc tài liệu kiến trúc. Nếu là một nhà phát triển và muốn đóng góp vào mã của một chương trình, tài liệu này sẽ cho chúng ta biết về kiến trúc phần mềm, các tiêu chuẩn mã hóa và các công cụ và quy trình làm việc được sử dụng để làm việc trên phần mềm.

Dự án LibreOffice đã xuất bản một hướng dẫn dành cho nhà phát triển trên wiki của mình để giúp các nhà phát triển quan tâm tham gia vào cộng đồng (https://wiki.documentfoundation.org/Documentation/DevGuide).

Bài tập Hướng dẫn

1. Tại sao các dự án mã nguồn mở nói riêng phải đặc biệt quan tâm tới các công cụ thích hợp dành cho việc giao tiếp và cộng tác?

2. Hãy nêu một ví dụ cho hai loại hình giao tiếp đồng bộ và không đồng bộ.

3. Nhược điểm của ứng dụng nhắn tin là gì và làm sao để tránh được nhược điểm này?

4. Hãy kể tên hai chức năng của một wiki.

5. Sự khác biệt giữa trình theo dõi lỗi và bộ phận trợ giúp là gì?

6. Sự khác biệt giữa một hệ thống quản lý nội dung và một hệ thống quản lý tài liệu là gì?

7. Ưu điểm của một hệ thống độc lập hoặc liên kết so với một hệ thống tập trung là gì?

Bài tập Mở rộng

1. Một trong những điểm khác biệt chính giữa một câu lạc bộ thể thao địa phương và một dự án mã nguồn mở quốc tế là gì?

2. Tại sao việc đóng góp cho một dự án mã nguồn mở lại có thể đặc biệt có giá trị?

3. Dự án Ubuntu sử dụng phần mềm theo dõi lỗi nào?

4. Tên của trang web dành cho danh sách gửi thư của hạt nhân Linux là gì?

Tóm tắt

Trong bài học này, chúng ta đã tìm hiểu về nhiều công cụ khác nhau được sử dụng để giao tiếp và cộng tác trong một dự án mã nguồn mở. Chúng ta đã học về sự khác biệt giữa giao tiếp đồng bộ và không đồng bộ và giữa các công cụ phi tập trung, tập trung và độc lập. Chúng ta cũng đã tìm hiểu về lý do tại sao các công cụ cụ thể lại hữu ích cho các nhiệm vụ cụ thể để làm cho việc đóng góp vào một dự án mã nguồn mở trở nên càng thú vị và có giá trị.

Đáp án Bài tập Hướng dẫn

1. Tại sao các dự án mã nguồn mở nói riêng phải đặc biệt quan tâm tới các công cụ thích hợp dành cho việc giao tiếp và cộng tác?

   Một mặt, việc làm việc cùng nhau trong một nhóm phân tán trên toàn thế giới có khả năng sẽ gặp phải rất nhiều thách thức; những thách thức này có thể được giải quyết với các công cụ phù hợp. Mặt khác, các tình nguyện viên có thể sẽ không ở lại mãi; vì vậy, việc giữ lại và chia sẻ kiến thức là một khía cạnh quan trọng khác trong việc sử dụng các công cụ giao tiếp và cộng tác. Việc khiến cho các đóng góp trở nên dễ dàng sẽ giúp duy trì tính bền vững của một dự án.

2. Hãy nêu một ví dụ cho hai loại hình giao tiếp đồng bộ và không đồng bộ.

   Giao tiếp đồng bộ có thể là một cuộc trò chuyện trực tiếp, một cuộc điện thoại hoặc cuộc gọi video. Email, tin nhắn SMS, thư bưu chính và fax chính là các ví dụ về giao tiếp không đồng bộ.

3. Nhược điểm của ứng dụng nhắn tin là gì và làm sao để tránh được nhược điểm này?

   Khi cài đặt chúng trên điện thoại, chúng ta có thể nhận được quá nhiều thông báo, mỗi tin nhắn mới sẽ là một thông báo. Một cấu hình phù hợp có thể sẽ giúp tránh được điều này. Một nhược điểm khác chính là việc nhiều ứng dụng nhắn tin được điều hành bởi các nhà cung cấp độc quyền.

4. Hãy kể tên hai chức năng của một wiki.

   Biên tập và dịch bài viết dưới hình thức hợp tác.

5. Sự khác biệt giữa trình theo dõi lỗi và bộ phận trợ giúp là gì?

   Trình theo dõi lỗi là một công cụ phần mềm chuyên dụng để báo cáo lỗi hoặc yêu cầu các tính năng trong phần mềm. Bộ phận hỗ trợ tập trung vào việc hỗ trợ các yêu cầu và quản lý mọi loại vấn đề và yêu cầu (ví dụ như trên trang web).

6. Sự khác biệt giữa một hệ thống quản lý nội dung và một hệ thống quản lý tài liệu là gì?

   CMS được sử dụng để trình bày nội dung theo một cách hoặc một mẫu cụ thể, chủ yếu là sử dụng cho các mục đích công khai/ công cộng. DMS được sử dụng để lưu trữ các thư từ hiện có, chủ yếu được sử dụng nội bộ.

7. Ưu điểm của một hệ thống độc lập hoặc liên kết so với một hệ thống tập trung là gì?

   Là một hệ thống tập trung nằm dưới sự kiểm soát của một nhà cung cấp bên ngoài. Mọi nội dung sẽ được lưu trữ trên máy chủ của nhà cung cấp bên ngoài và phải tuân theo các điều

khoản và điều kiện của họ.

Đáp án Bài tập Mở rộng

1. Một trong những điểm khác biệt chính giữa một câu lạc bộ thể thao địa phương và một dự án mã nguồn mở quốc tế là gì?

   Các dự án mã nguồn mở không bị ràng buộc với một ngôn ngữ hoặc một địa điểm cụ thể. Những người đóng góp có thể sống ở các quốc gia và châu lục khác nhau, có ngôn ngữ bản địa khác nhau và thậm chí sống ở các múi giờ khác nhau. Hầu hết các hoạt động được thực hiện trong một dự án nguồn mở đều diễn ra trực tuyến chứ không trực tiếp.

2. Tại sao việc đóng góp cho một dự án mã nguồn mở lại có thể đặc biệt có giá trị?

   Nhiều dự án mã nguồn mở có những nhóm người phát triển rất đa dạng. Bằng cách cộng tác với họ, chúng ta có thể học hỏi từ họ, khám phá những điều mới mẻ và mở rộng phạm vi của chính mình.

3. Dự án Ubuntu sử dụng phần mềm theo dõi lỗi nào?

   Launchpad.

4. Tên của trang web dành cho danh sách gửi thư của hạt nhân Linux là gì?

   `https://lkml.org/`.

Ấn bản

PDF được tạo vào: 2025-01-08

Tài liệu Học tập LPI là một sáng kiến của Linux Professional Institute (https://lpi.org). Tài liệu Học tập và các Bản dịch của chúng có thể được tìm thấy tại https://learning.lpi.org.

Đối với các câu hỏi và nhận xét về ấn bản này cũng như về toàn bộ dự án, hãy gửi email tới: learning@lpi.org.