TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

Chủ đề : The Broken Window Theory & The Boy Scout Rule

Môn học: Thiết kế phần mềm

Lớp: 22_3

Sinh viên thực hiện: 22120185 – Võ Văn Lĩnh
22120201 – Huỳnh Mẫn
22120208 – Hoàng Hồ Nhật Minh
22120247 – Nguyễn Hữu Khánh Nhân

Hồ Chí Minh, ngày 27 tháng 03 năm 2025

MỤC LỤC

I. Lý thuyết Cửa sổ Vỡ:

1. Định nghĩa và nguồn gốc:

Lý thuyết Cửa sổ Vỡ, một khái niệm có nguồn gốc từ lĩnh vực tội phạm học, được James Q. Wilson và George L. Kelling đưa ra vào những năm 1980.

Thí nghiệm nổi tiếng của nhà tâm lý học Philip Zimbardo minh họa rõ ràng nguyên lý này. Ông đã để hai chiếc xe ô tô không có người trông coi ở hai khu vực khác nhau: một khu giàu có và một khu nghèo. Ở khu nghèo, chiếc xe nhanh chóng bị phá hoại. Ngược lại, ở khu giàu có, chiếc xe vẫn nguyên vẹn cho đến khi chính Zimbardo đập vỡ một cửa sổ. Sau đó, chiếc xe cũng bị phá hoại. Thí nghiệm này cho thấy rằng nguyên nhân không nằm ở yếu tố kinh tế mà ở tín hiệu về sự bỏ bê mà chiếc cửa sổ vỡ truyền đi.

Theo lý thuyết, một cửa sổ vỡ không chỉ đơn thuần là một mảnh kính bị hỏng. Nó là một dấu hiệu hữu hình của sự bỏ bê, một thông điệp ngầm rằng "trật tự không còn quan trọng ở đây, sự quan tâm không còn ý nghĩa ở đây". Thông điệp này có thể lan tỏa, khuyến khích các hành vi vô văn hóa và thậm chí là phạm pháp khác.

Áp dụng ẩn dụ Cửa sổ Vỡ vào Phát triển Phần mềm

Trong bối cảnh phát triển phần mềm, khái niệm "entropy phần mềm" hay "sự suy thoái phần mềm" thường được sử dụng để mô tả xu hướng chất lượng phần mềm giảm dần theo thời gian nếu không được duy trì tích cực. Tương tự như một tòa nhà với cửa sổ vỡ, các vấn đề nhỏ trong mã nguồn hoặc thiết kế phần mềm có thể được coi là những "cửa sổ vỡ" trong bối cảnh này. Các vấn đề này có thể bao gồm:

- Các quyết định thiết kế tồi.
- Phong cách mã hóa kém và không nhất quán.
- Các cảnh báo trình biên dịch chưa được giải quyết.
- Quy ước đặt tên không nhất quán.
- Thiếu chú thích hoặc tài liệu lỗi thời.
- Các giải pháp tạm bợ được triển khai để đáp ứng thời hạn.
- Các kiểm thử không ổn định hoặc dễ bị lỗi.
- "Magic numbers" và mã trùng lặp.
- Các chú thích "TODO" chưa được xử lý.
- Các quyết định kiến trúc kém.

Ví dụ về tiêu đề HTTP Referer bị viết sai chính tả là một "cửa sổ vỡ" tồn tại lâu dài trong thế giới phần mềm, gây ra nhiều vấn đề lặp đi lặp lại cho vô số người. Điều này cho thấy ngay cả những lỗi nhỏ, dai dẳng cũng có thể gây ra những hậu quả lan rộng và kéo dài. Nếu những vấn đề nhỏ này không được giải quyết, chúng sẽ phát đi tín hiệu về sự thiếu quan tâm và có thể dẫn đến sự suy giảm dần dần về chất lượng mã nguồn tổng thể và tinh thần làm việc của nhóm.

2. Hậu quả của các vấn đề bị bỏ qua:

Việc bỏ qua các vấn đề nhỏ trong mã nguồn có thể tạo ra một vòng phản hồi tiêu cực. Sự bỏ bê là một yếu tố thúc đẩy mạnh mẽ sự suy thoái của phần mềm. Khi các "cửa sổ vỡ" tích tụ, chúng có thể bình thường hóa các thực hành kém. Các nhà phát triển có thể nghĩ rằng, "Nếu điều này được chấp nhận, thì đoạn mã hơi lộn xộn của tôi có lẽ cũng không sao." Điều này tạo ra một hiệu ứng lan tỏa, trong đó các tiêu chuẩn chất lượng dần bị hạ thấp. Việc cho phép các "cửa sổ vỡ" tích tụ có thể dẫn đến nhiều hậu quả tiêu cực:

- **Nợ kỹ thuật gia tăng:** Các vấn đề nhỏ không được giải quyết sẽ tích tụ thành nợ kỹ thuật, khiến mã nguồn ngày càng khó hiểu và cập nhật.
- **Tốc độ phát triển chậm hơn:** Khi mã nguồn trở nên phức tạp và lộn xộn, việc thêm các tính năng mới hoặc sửa lỗi sẽ mất nhiều thời gian hơn.
- **Tỷ lệ lỗi cao hơn:** Mã nguồn kém chất lượng dễ chứa lỗi hơn và việc tìm kiếm và sửa lỗi cũng trở nên khó khăn hơn.
- **Khó khăn trong việc hiểu và bảo trì mã nguồn:** Một codebase đầy "cửa sổ vỡ" sẽ gây khó khăn cho các nhà phát triển trong việc hiểu logic, dẫn đến tốn thời gian và dễ gây ra lỗi khi bảo trì hoặc mở rộng.
- **Giảm tinh thần làm việc của nhóm và nguy cơ kỹ sư rời bỏ:** Làm việc với một codebase lộn xộn và khó bảo trì có thể gây ra sự thất vọng và căng thẳng cho các nhà phát triển, làm tăng nguy cơ họ rời bỏ dự án.
- **Chi phí và thời gian cho các thay đổi và tái cấu trúc trong tương lai tăng lên:** Khi nợ kỹ thuật tích tụ, việc thực hiện các thay đổi lớn hoặc tái cấu trúc mã nguồn sẽ trở nên tốn kém và rủi ro hơn.
- **Nguy cơ phát sinh lỗi mới khi sửa lỗi hiện có:** Trong một codebase phức tạp, việc sửa một lỗi có thể vô tình tạo ra các lỗi mới.
- **Phần mềm có nguy cơ trở nên không thể bảo trì, dẫn đến thất bại của dự án:** Nếu tình trạng bỏ bê kéo dài, codebase có thể trở nên quá lộn xộn và khó hiểu đến mức không thể tiếp tục phát triển hoặc bảo trì, cuối cùng dẫn đến sự thất bại của dự án.

Nếu một dự án có các kiểm thử không ổn định hoặc một thiết kế chắp vá, các nhà phát triển có nhiều khả năng sẽ thêm vào các kiểm thử không ổn định hoặc các đoạn mã chắp vá khác. Các lỗi hiện có tạo ra một tiền lệ và hạ thấp tiêu chuẩn cho các đóng góp mới. Do đó, việc giải quyết các vấn đề kịp thời là rất quan trọng để ngăn chặn sự suy thoái hơn nữa.

II. Quy tắc Hướng đạo sinh:

1. Định nghĩa và nguồn gốc:

Quy tắc Hướng đạo sinh là một nguyên tắc quan trọng trong phát triển phần mềm, tập trung vào việc duy trì một codebase sạch và lành mạnh. Nguyên tắc này bắt nguồn từ quy tắc của Tổ chức Hướng đạo Mỹ, đó là luôn rời khỏi khu cắm trại sạch sẽ hơn so với khi đến.

Robert C. Martin ("Uncle Bob") đã phổ biến quy tắc này trong giới phát triển phần mềm, đặc biệt là trong cuốn sách nổi tiếng "Clean Code" . Nguyên tắc cốt lõi của Quy tắc Hướng đạo sinh trong phát triển phần mềm là: "Luôn để lại mã nguồn tốt hơn so với khi bạn tìm thấy nó".

2. Nguyên tắc luôn để lại mã nguồn tốt hơn so với khi bạn tìm thấy nó:

Nguyên tắc này không yêu cầu các nhà phát triển phải thực hiện các cuộc đại tu mã nguồn lớn với mỗi thay đổi. Thay vào đó, nó khuyến khích thực hiện các cải tiến nhỏ, gia tăng bất cứ khi nào có cơ hội. Điều này có thể bao gồm nhiều hành động nhỏ :

- Đổi tên các biến hoặc phương thức được đặt tên kém.
- Sửa lỗi chính tả và các vấn đề về định dạng.
- Xóa mã không cần thiết hoặc đã được chú thích.
- Thêm chú thích để giải thích logic phức tạp.
- Chia các hàm phức tạp thành các hàm nhỏ hơn, dễ đọc hơn.
- Cải thiện cấu trúc và khả năng đọc của mã.
- Loại bỏ mã trùng lặp (tuân thủ nguyên tắc DRY - Don't Repeat Yourself).
- Thêm các kiểm thử cho các trường hợp sử dụng còn thiếu.
- Cập nhật tài liệu hoặc chú thích đã lỗi thời.
- Giải quyết các cảnh báo của trình biên dịch.

Điều quan trọng cần lưu ý là việc cải thiện mã nên tập trung vào phần mã mà nhà phát triển đang làm việc, thay vì cố gắng dọn dẹp toàn bộ codebase cùng một lúc. Hơn nữa, việc để lại một mớ hỗn độn trong mã nên được coi là một hành động không thể chấp nhận được trong một đội ngũ phát triển.

3. Lợi ích của việc Tuân thủ Quy tắc Hướng đạo sinh:

Việc áp dụng nhất quán Quy tắc Hướng đạo sinh mang lại nhiều lợi ích đáng kể cho dự án phần mềm:

- **Giảm nợ kỹ thuật:** Bằng cách giải quyết các vấn đề nhỏ ngay lập tức, nợ kỹ thuật sẽ không có cơ hội tích tụ theo thời gian.
- **Cải tiến liên tục:** Thay vì chờ đợi các đợt tái cấu trúc lớn, codebase sẽ được cải thiện dần dần qua từng thay đổi nhỏ.
- **Cải thiện sự hợp tác của nhóm:** Mã nguồn sạch và dễ hiểu giúp các thành viên trong nhóm làm việc cùng nhau hiệu quả hơn.
- **Tiết kiệm thời gian:** Mã sạch dễ dàng gỡ lỗi và bảo trì hơn, giúp tiết kiệm thời gian trong dài hạn.
- **Mang lại giá trị cho người dùng:** Các cải tiến nhỏ và nhất quán dẫn đến một hệ thống ổn định và đáng tin cậy hơn.
- **Xây dựng thói quen tốt:** Việc dọn dẹp sau mỗi lần làm việc giúp hình thành một văn hóa trách nhiệm và quan tâm đến chất lượng mã.
- **Duy trì mã nguồn dễ quản lý:** Khi dự án phát triển, việc liên tục cải thiện mã giúp codebase không trở nên quá phức tạp.

- **Dễ dàng mở rộng và thêm tính năng mới:** Một codebase sạch sẽ và có cấu trúc tốt sẽ tạo điều kiện thuận lợi cho việc thêm các tính năng mới.
- **Giảm nguy cơ kỹ sư rời bỏ:** Làm việc trong một môi trường với mã nguồn chất lượng cao sẽ giúp tăng sự hài lòng và giảm nguy cơ kỹ sư bỏ đi do chán nản.
- **Nâng cao khả năng đọc và bảo trì mã cho các nhà phát triển trong tương lai:** Điều này bao gồm cả chính bạn khi quay lại mã sau một thời gian.

Những nỗ lực nhỏ và nhất quán này sẽ tích lũy theo thời gian, dẫn đến một codebase khỏe mạnh hơn đáng kể.

4. Các tình huống thực tế:

Quy tắc Hướng đạo sinh có thể khó khăn, chúng ta có thể hình dung các ví dụ về các dự án mà việc áp dụng nhất quán nguyên tắc này đã mang lại kết quả tích cực:

- Một dự án tồn tại lâu dài, nơi các nhà phát triển thường xuyên tái cấu trúc các phần mã nhỏ, cải thiện quy ước đặt tên và thêm chú thích, dẫn đến một codebase tương đối sạch sẽ và dễ bảo trì trong nhiều năm.
- Một nhóm đã tạo thói quen giải quyết tất cả các cảnh báo của trình biên dịch và các vấn đề phân tích mã với mỗi commit, dẫn đến việc giảm số lượng lỗi và cải thiện tính ổn định của mã.
- Một dự án mà các code review tập trung cụ thể vào việc xác định và giải quyết các khu vực nhỏ cần cải thiện, nuôi dưỡng một văn hóa liên tục nâng cao chất lượng mã.
- Một nhóm liên tục cập nhật tài liệu cùng với các thay đổi mã, đảm bảo rằng codebase luôn được ghi chép đầy đủ và dễ dàng cho các thành viên mới tham gia.

Những tác động tích cực này bao gồm giảm nợ kỹ thuật, cải thiện hiệu quả của nhóm và chất lượng phần mềm cao hơn.

III. Mối tương quan:

Quy tắc Hướng đạo sinh Giải quyết Lý thuyết Cửa sổ Vỡ trong Phần mềm như thế nào ?

Ngăn chặn "Cửa sổ Vỡ" Thông qua Cải tiến Liên tục

Tính chủ động của Quy tắc Hướng đạo sinh trực tiếp chống lại những tác động tiêu cực được mô tả bởi Lý thuyết Cửa sổ Vỡ. Quy tắc Hướng đạo sinh là một chiến lược triển khai thiết thực để ngăn chặn sự suy thoái của phần mềm. Bằng cách khuyến khích các nhà phát triển sửa chữa các vấn đề nhỏ khi họ gặp phải, Quy tắc Hướng đạo sinh ngăn chặn sự xuất hiện của "cửa sổ vỡ" hoặc đảm bảo chúng không bị bỏ mặc mà không được sửa chữa. Việc liên tục dọn dẹp mã nguồn giúp ngăn chặn sự tích tụ của các vấn đề nhỏ, những vấn đề có thể báo hiệu sự bỏ bê và dẫn đến sự suy thoái hơn nữa. Việc bảo trì thường xuyên đóng vai trò như một hình thức "giám sát những thứ nhỏ nhặt" trong phần mềm, tương tự như việc giải quyết các hành vi phạm tội nhỏ để ngăn chặn các hành vi lớn hơn trong lý thuyết tội phạm học.

Duy trì Văn hóa Chất lượng và Chú trọng Chi tiết

Việc tuân thủ Quy tắc Hướng đạo sinh nuôi dưỡng một văn hóa nhóm coi trọng chất lượng mã và sự chú trọng đến chi tiết. Khi mọi người trong nhóm đều được kỳ vọng sẽ để lại mã nguồn tốt hơn so với khi họ tìm thấy nó, điều này tạo ra một ý thức trách nhiệm chung đối với chất lượng mã. Bằng cách giải quyết ngay cả những sự không nhất quán và lỗi nhỏ nhất, nhóm gửi đi một thông điệp rằng chất lượng là quan trọng, từ đó ngăn chặn việc đưa vào các "cửa sổ vỡ" khác. Sự chú trọng nhất quán đến chi tiết tạo ra một tiền lệ tích cực cho toàn bộ nhóm, củng cố ý tưởng rằng môi trường (codebase) truyền tải các kỳ vọng. Một codebase sạch sẽ và được bảo trì tốt khuyến khích các nhà phát triển duy trì các tiêu chuẩn đó.

Hiệu ứng Tổng hợp đối với Sức khỏe Dự án Dài hạn

Việc kết hợp áp dụng cả hai nguyên tắc này dẫn đến một dự án phần mềm bền vững và dễ bảo trì hơn. Lý thuyết Cửa sổ Vỡ làm nổi bật những rủi ro của việc bỏ bê, trong khi Quy tắc Hướng đạo sinh cung cấp một giải pháp thông qua cải tiến chủ động. Hai nguyên tắc này bổ sung cho nhau. Việc hiểu rõ nguy cơ suy thoái thúc đẩy việc áp dụng nhất quán quy tắc cải thiện. Bằng cách ngăn chặn sự tích tụ của nợ kỹ thuật và nuôi dưỡng một văn hóa chất lượng, nhóm có thể đảm bảo sức khỏe và khả năng phát triển lâu dài của phần mềm. Một codebase khỏe mạnh sẽ dễ dàng thích ứng với các yêu cầu và công nghệ thay đổi, đảm bảo phần mềm vẫn có giá trị theo thời gian.

IV. Các phương pháp và công cụ hỗ trợ việc tuân thủ Quy tác Hướng đạo sinh:

Vai trò của Code Review trong Việc Thực thi các Tiêu chuẩn Chất lượng

Code review là một cơ chế quan trọng để đảm bảo việc tuân thủ Quy tắc Hướng đạo sinh. Trong quá trình review, người đánh giá không chỉ nên tập trung vào chức năng của mã mà còn phải chú ý đến việc xác định các khu vực có thể cải thiện về độ rõ ràng, phong cách và khả năng bảo trì. Cần lưu ý cụ thể các khía cạnh như quy ước đặt tên, định dạng mã, khả năng đơn giản hóa và sự hiện diện của "code smells" (những dấu hiệu cho thấy có vấn đề tiềm ẩn trong mã).

Tái cấu trúc Mã (Refactoring) như một Thực hành Cải tiến Liên tục

Tái cấu trúc mã là một thực hành thiết yếu để "luôn để lại mã nguồn tốt hơn so với khi bạn tìm thấy nó". Đây nên là một quá trình liên tục, gia tăng thay vì một nỗ lực lớn và không thường xuyên. Các ví dụ về các tác vụ tái cấu trúc nhỏ phù hợp với Quy tắc Hướng đạo sinh bao gồm việc tách các phương thức, đổi tên biến và đơn giản hóa logic điều kiện.

Tận dụng các Công cụ Phân tích Tĩnh và Định dạng Mã

Các công cụ tự động có thể hỗ trợ rất nhiều trong việc xác định và giải quyết các "cửa sổ vỡ" tiềm ẩn và đảm bảo tính nhất quán của mã. Việc sử dụng các công cụ phân tích tĩnh (linters) như ESLint, SonarQube, Pylint có thể giúp thực thi các tiêu chuẩn mã hóa và xác định các vấn đề tiềm ẩn. Các công cụ này cũng có giá trị trong việc phát hiện các "code smells", lỗi và các lỗ hổng bảo mật. Ngoài ra, các

công cụ định dạng mã (formatters) như Prettier, Black có thể tự động thực thi phong cách mã nhất quán, giảm thiểu các cuộc tranh luận về định dạng và đảm bảo khả năng đọc của mã.

Nuôi dưỡng Văn hóa Nhóm Trách nhiệm và Sở hữu

Việc triển khai thành công Quy tắc Hướng đạo sinh đòi hỏi một văn hóa nhóm mà mọi nhà phát triển đều cảm thấy có trách nhiệm đối với chất lượng của codebase. Các chiến lược để nuôi dưỡng văn hóa này bao gồm:

- Giáo dục nhóm về các nguyên tắc và lợi ích.
- Lãnh đạo bằng gương và thể hiện cam kết đối với chất lượng mã.
- Ghi nhận và khen thưởng những nỗ lực cải thiện codebase.
- Tạo một môi trường an toàn và hỗ trợ để đưa ra và nhận phản hồi trong quá trình code review.
- Biến chất lượng mã trở thành một khía cạnh không thể thương lượng trong quy trình phát triển.

NGUỒN THAM KHẢO

1. Broken windows theory - Wikipedia
   https://en.wikipedia.org/wiki/Broken_windows_theory
2. Broken Windows Theory | Psychology Today
   https://www.psychologytoday.com/us/basics/broken-windows-theory
3. Broken windows theory, what is it? | People ACCIONA
   https://people.acciona.com/trends-and-inspiration/broken-windows-theory/
4. Broken Windows, Informal Social Control, and Crime: Assessing Causality in Empirical Studies - PubMed Central
   https://pmc.ncbi.nlm.nih.gov/articles/PMC8059646/
5. The Broken Windows Theory: The Key to Tackling Software Entropy - ArcherPoint
   https://archerpoint.com/the-broken-windows-theory-the-key-to-tackling-software-entropy/
6. The Broken Window in Software Projects - KungFuDev
   https://www.kungfudev.com/blog/2024/05/18/broken-windows
7. Broken Windows Theory in Software Development: Why Details Matter | HackerNoon
   https://hackernoon.com/broken-windows-theory-in-software-development-why-details-matter
8. Broken windows theory: why code quality and simplistic design are non-negotiable | by Mat Ryer | Medium
   https://medium.com/@matryer/broken-windows-theory-why-code-quality-and-simplistic-design-are-non-negotiable-e37f8ce23dab
9. Broken Windows in Software Engineering - San Tuon
   https://www.santuon.com/broken-windows-in-code/
10. How the Boy Scout Rule Solves Common Code Maintenance Issues (2024) - Snappify
    https://snappify.com/blog/boy-scout-rule
11. Boy Scout Rule | DevIQ
    https://deviq.com/principles/boy-scout-rule/
12. "Boy Scout Rule" Definition - Innolution
    https://innolution.com/resources/glossary/boy-scout-rule