THỬ NGHIỆM ĐÁNH GIÁ ÁP DỤNG MỘT SỐ KỸ THUẬT KIỂM THỬ ĐỂ NÂNG CAO ĐỘ TIN CẬY CHO ỨNG DỤNG DI ĐỘNG TRONG MÔI TRƯỜNG PHÁT TRIỂN LINH HOẠT

Nguyễn Thanh Hùng[1], *Nguyễn Đức Mận*[2], *Huỳnh Quyết Thắng*[1]

Tóm tắt

Hệ sinh thái ứng dụng di động đã phát triển rất nhanh với hàng triệu ứng dụng và hàng trăm nghìn nhà phát triển trong những năm gần đây. Trong xu hướng cạnh tranh, để sản phẩm ứng dụng di động được tin dùng, các nhà phát triển cần có các kỹ thuật, phương pháp và công cụ để (i) nâng cao hiệu quả việc kiểm thử trong quá trình phát triển và xác định những thất bại tiềm ẩn trước khi ứng dụng được phát hành, (ii) đánh giá độ tin cậy phần mềm từ các dữ liệu thực nghiệm của các pha trong quá trình phát triển sản phẩm phần mềm, dựa vào các kỹ thuật đánh giá nhằm tính toán giá trị độ đo độ tin cậy phần mềm và (iii) hiệu suất khi đối mặt với các điều kiện khác nhau khi sử dụng thực tế. Trong nghiên cứu này, chúng tôi sử dụng mô hình tăng trưởng độ tin cậy NHPP (Non-Homogeneous Poisson Process) để đánh giá, xác định độ tin cậy của các ứng dụng di động thông qua việc áp dụng các kỹ thuật, phương pháp kiểm thử và kiểm thử tự động đã được chúng tôi công bố ở các nghiên cứu trước như (1) kỹ thuật tối ưu mã nguồn, (2) kiểm thử hướng ngữ cảnh One2explore, (3) ứng dụng Heuristics và Machine learning trong kiểm thử, (4) sinh kiểm thử tự động từ user stories và acceptance criteria. Kết quả nghiên cứu và thực nghiệm đánh giá trên 2 dự án thực tế, và thử nghiệm trên một số ứng dụng Android từ kho mã nguồn FOSS cho kết quả tích cực có thể giúp cho các nhà phát triển ứng dụng Android cải tiến chất lượng, nâng cao độ tin cậy và hiệu năng khi phát triển các ứng dụng di động trong môi trường phát triển linh hoạt và cạnh tranh hiện nay.

Từ khóa

Kỹ thuật kiểm thử; kiểm thử ứng dụng di động; độ tin cậy ứng dụng di động; phát triển phần mềm linh hoạt.

1. Giới thiệu

Phần mềm đã được sử dụng trong mọi lĩnh vực trong cuộc sống hiện đại của chúng ta. Tuy nhiên, phần mềm có thể có lỗi và thất bại của nó dẫn đến các hậu quả, có thể là các tai nạn thảm khốc, gây thiệt hại lớn về kinh tế và con người [1]. Vì vậy, chất lượng phần mềm trở thành một vấn đề quan trọng và các thuộc tính chất lượng phần

[1]Đại học Bách khoa Hà Nội, [2]Đại học Duy Tân.

mềm luôn là những vấn đề được tập trung nghiên cứu trong nhiều năm qua [1]. Độ tin cậy là thuộc tính quan trọng nhất của phần mềm bởi vì nó liên quan đến hoạt động và tính chính xác của sản phẩm. Độ tin cậy của phần mềm là xác suất mà hệ thống phần mềm sẽ hoạt động mà không có thất bại trong một môi trường nhất định và trong một khoảng thời gian nhất định [35]. Kỹ nghệ độ tin cậy của phần mềm SRE (Software Realiability Engineering) đã được nghiên cứu và phát triển để giải quyết các vấn đề về độ tin cậy. Các kỹ thuật dự đoán, đánh giá độ tin cậy của phần mềm thường dựa chủ yếu vào các mô hình toán học, trong đó kỹ thuật thống kê đã đóng một vai trò quan trọng. Hàng trăm mô hình độ tin cậy đã và đang được nghiên cứu và phát triển trong những thập kỷ qua [35]. Các mô hình này xác định các biện pháp thích hợp cho độ tin cậy và mục đích chính của chúng là ước tính và dự đoán độ tin cậy của phần mềm dựa trên dữ liệu thất bại được thu thập trong quá trình phát triển, thử nghiệm và sau khi phát hành. Các thước đo độ tin cậy bao gồm thời gian trung bình thất bại (MTTF), thời gian trung bình giữa các thất bại (MTBF), cường độ hỏng hóc, thời gian thử nghiệm bổ sung cần thiết để đạt được mục tiêu độ tin cậy và do đó, là công cụ trợ giúp cho người quản lý phần mềm đưa ra quyết định kiểm thử tiếp hay phát hành sản phẩm [35]. Sự khác biệt về độ tin cậy của các ứng dụng trên máy tính để bàn và điện thoại thông minh xuất phát từ những lý do chính [2], [4] như sau: sự khác biệt về phần cứng; sự khác biệt về hệ điều hành (OS); khác biệt về tính chất và kích cỡ của các ứng dụng được thực hiện trong máy tính để bàn và điện thoại thông minh; sự khác biệt trong môi trường hoạt động (ở đâu và khi nào thiết bị được sử dụng) và hồ sơ sử dụng (cách thiết bị được sử dụng) trong cả hai trường hợp và cuối cùng là sự khác biệt trong chức năng hiển thị. Việc đo lường, đánh giá độ tin cậy của ứng dụng phần mềm, của hệ thống phần mềm bằng nhiều phương pháp, kỹ thuật khác nhau, phổ biến nhất là các mô hình tăng trưởng độ tin cậy phần mềm (SRGMs), cụ thể như: NHPP, Musa, Nelson, ... Hiện tại có hai hướng tiếp cận chính trong việc đo lường và xác định độ tin cậy phần mềm:

- (i) Dự đoán độ tin cậy phần mềm: từ các thông số của hệ thống hoặc dự án phát triển sản phẩm phần mềm, dựa vào các kỹ thuật dự đoán nhằm ước tính giá trị độ đo độ tin cậy phần mềm.
- (ii) Đánh giá độ tin cậy phần mềm: từ các dữ liệu thực nghiệm của các pha trong quá trình phát triển sản phẩm phần mềm, dựa vào các kỹ thuật đánh giá nhằm tính toán giá trị độ đo độ tin cậy phần mềm.

Giá trị độ đo độ tin cậy phần mềm là một thông số quan trọng được sử dụng trong nhiều pha khác nhau của quá trình phát triển sản phẩm phần mềm: lập trình, gỡ lỗi, phát hành và bảo trì. Việc sử dụng thông số này giúp gia tăng chất lượng cũng như hỗ trợ các thao tác ra quyết định trong các pha đó. Phương pháp phát triển linh hoạt (Agile) là phương pháp hướng đến cách tiếp cận phi truyền thống, làm nổi bật sự hợp tác của khách hàng, các thành viên trong nhóm có động lực cao, cung cấp phần mềm chất lượng cao, khả năng thích ứng với các thay đổi và duy trì sự đơn giản trong phát triển [16], [43]. Cách tiếp cận linh hoạt tuân theo triết lý phát hành sớm, phát hành thường xuyên, trong đó nhấn mạnh tầm quan trọng của việc phát hành sớm và thường xuyên của hệ thống. Các bản phát hành sớm của phần mềm cung cấp một sản phẩm

cốt lõi với một bộ các chức năng hạn chế và mỗi bản phát hành tiếp theo sẽ tăng thêm các chức năng mới, sửa chữa các lỗi hiện có hoặc điều chỉnh các công nghệ mới. Vì mỗi bản phát hành đều thêm mã mới vào hệ thống, nó có khả năng đưa ra các lỗi mới. Mặc dù một bản phát hành mới có nghĩa là để cải thiện hệ thống, luôn có khả năng bị thoái hóa do sự bổ sung của các lỗi mới. Về cơ bản, mọi bản phát hành đều thêm một số nội dung lỗi vào hệ thống hiện có. Trong giai đoạn các bản phát hành thường xuyên được thực hiện sẽ làm tăng tỷ lệ thất bại chung, do đó làm giảm độ tin cậy của hệ thống. Các nghiên cứu [4], [13], [14], [16]–[18], [43], [46] sẽ là cơ sở đề đề xuất việc áp dụng một số kỹ thuật kiểm thử vào trong quy trình phát triển Agile Scrum. Trong phạm vi của nghiên cứu này, chúng tôi thực hiện việc đánh giá độ tin cậy của ứng dụng di động bằng mô hình tăng trưởng độ tin cậy phần mềm thông qua việc áp dụng các kỹ thuật tối ưu hóa và tái cấu trúc mã nguồn, kỹ thuật PMD và Android Lint, kỹ thuật sinh ca kiểm thử tự động dựa trên User story và điều kiện chấp nhận, kỹ thuật kiểm thử hướng ngữ cảnh, kỹ thuật ứng dụng Heuristics và học máy (ML) đã được nghiên cứu và công bố [21]–[26], [49]. Chúng tôi đề xuất quy trình, cách thực hiện và vận dụng hiệu quả các kỹ thuật này để nâng cao độ tin cậy cho ứng dụng di động và thử nghiệm, đánh giá kết quả áp dụng quy trình trong một số dự án thực tế. Các nội dung tiếp theo của bài báo sẽ trình bày tổng quan về các khái niệm liên quan ở phần 2, các nghiên cứu liên quan được trình bày ở phần 3. Phần 4 trình bày quy trình và các kỹ thuật kiểm thử nhằm nâng cao độ tin cậy cho ứng dụng di động. Phần 5 thảo luận các kết quả thực nghiệm. Kết luận và hướng phát triển được trình bày ở phần 6.

2. Các khái niệm và các nghiên cứu liên quan

2.1. Độ tin cậy phần mềm

Theo tiêu chuẩn chất lượng quốc tế về chất lượng phần mềm ISO/IEC 25010 (ISO/IEC 25000:2014) [5], [32], độ tin cậy là một trong những đặc tính chất lượng chính. Độ tin cậy có những ứng dụng nhất định trong các pha khác nhau của vòng đời phần mềm: thiết kế, lập trình, kiểm thử và triển khai. Theo tác giả Chengjie [40]: Độ tin cậy là xác suất phần mềm hoạt động không lỗi ở điều kiện cho trước trong một khoảng thời gian xác định. Trong khi đó Lyu [35] định nghĩa: Độ tin cậy là xác suất một phần mềm không có lỗi hoạt động trong một khoảng thời gian xác định ở điều kiện xác định. Tiêu chuẩn IEEE 610.12-1990 [5] định nghĩa độ tin cậy là "Khả năng của một hệ thống hoặc một bộ phận để thực hiện các chức năng cần thiết của nó theo các điều kiện đã nêu trong một khoảng thời gian nhất định". Độ tin cậy của phần mềm bao gồm ba hoạt động: (i) Phòng tránh lỗi; (ii) Phát hiện lỗi và gỡ bỏ; (iii) Các phép đo để tối đa hóa độ tin cậy, cụ thể là các biện pháp hỗ trợ hai hoạt động (i) và (ii). Đã có nhiều nghiên cứu về đo độ tin cậy bằng cách sử dụng thời gian trung bình giữa thất bại và thời gian trung bình để thất bại. Mô hình hóa thành công đã được thực hiện để dự đoán tỷ lệ lỗi và độ tin cậy [5]. Các hoạt động này giải quyết các khía cạnh đầu tiên và thứ ba về độ tin cậy, xác định và loại bỏ các lỗi để phần mềm hoạt động như mong đợi với độ tin cậy được xác định.

2.2. Biểu diễn toán học cho độ tin cậy

Về phương diện toán học, hàm tin cậy của hệ thống R(t) là xác suất mà hệ thống sẽ hoạt động thành công trong khoảng thời gian từ 0 đến thời điểm t: *R(t) = P (T > t), t > 0* với T là một biến ngẫu nhiên biểu diễn thời gian thất bại, hay chính là thời gian từ lúc hoạt động đến lúc bị lỗi. Xác suất thất bại là: *F (t) = 1- R(t) = P(T <= t)* , đây cũng chính là hàm phân bố của biến ngẫu nhiên T. Nếu biến ngẫu nhiên T có hàm mật độ xác suất là *f(t)*, khi đó độ tin cậy của hệ thống sẽ là [5]:

$$R(t) = \int_t^\infty f(x)dx \tag{1}$$

Thời gian trung bình giữa thất bại MTBF (Mean Time Between Failure) hoặc thời gian trung bình để thất bại MTTF (Mean Time To Failure) là thời gian trung bình mà hệ thống sẽ tiếp tục gặp thất bại. Nói cách khác đó chính là kỳ vọng của thời gian hệ thống hoạt động bình thường trước khi bị thất bại. Công thức toán học tính MTBF là [5]:

$$MTTF = \int_0^\infty t * f(t)dt = \int_0^\infty R(t)dt \tag{2}$$

Nếu thời gian thời gian trung bình giữa thất bại của hệ thống là một biến ngẫu nhiên tuân theo phân bố mũ với tham số là λ với hàm phân bố là $F(t) = 1 - e^{-\lambda t}$, khi đó:

$$MTTF = \int_0^\infty R(t)dt = \int_0^\infty e^{-\lambda t}dt = \frac{1}{\lambda} \tag{3}$$

Hàm mật độ thất bại (Failure density function): là hàm có vai trò rất quan trọng trong phân tích độ tin cậy của phần mềm. Định nghĩa về mặt toán học của hàm mật độ thất bại được cho bởi công thức (4).

$$\lambda(t) = \lim_{\Delta t \to \infty} \frac{R(t) - R(t + \Delta t)}{\Delta t R(t)} = \frac{f(t)}{R(t)} \tag{4}$$

Đại lượng $\lambda(t)dt$ biểu diễn xác suất một phần mềm sẽ gặp thất bại trong khoảng thời gian từ t đến t + dt . Hàm mật độ thất bại có vai trò quan trọng ở chỗ nó chỉ ra mật độ lão hóa của phần mềm. Ví dụ, hai thiết kế phần mềm khác nhau sẽ có cùng một độ tin cậy giống nhau ở cùng một thời điểm nào đó, nhưng mật độ tiến đến thất bại của chúng lại có thể rất khác nhau. Trong trường hợp đặc biệt, nếu thời gian trước thất bại tuân theo phân bố mũ, với tham số là λ thì hàm mật độ thất bại của nó sẽ là [5]:

$$\lambda(t) = \frac{f(t)}{R(t)} = \frac{\lambda \cdot e^{-\lambda t}}{e^{-\lambda t}} = \lambda \tag{5}$$

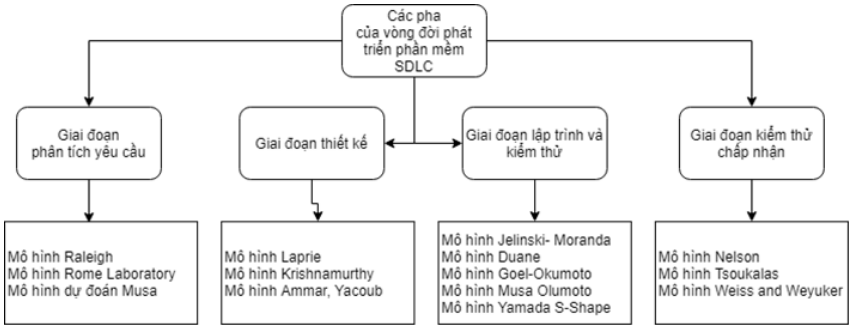
Điều đó có nghĩa là hàm mật độ thất bại của phần mềm là một hằng số, nói cách khác phần mềm không có hiện tượng lão hóa. Do độ tin cậy là một thuộc tính về chất lượng xây dựng phần mềm nên độ tin cậy cao phụ thuộc vào việc áp dụng các thuộc tính chất lượng ở từng giai đoạn của vòng đời phát triển với sự nhấn mạnh vào việc ngăn ngừa lỗi, đặc biệt là trong giai đoạn đầu của chu kỳ phát triển phần mềm [5].

2.3. Mô hình tăng trưởng độ tin cậy phần mềm

Mô hình tăng trưởng độ tin cậy phần mềm (Software Reliabilty Growth Models - SRGMs) là một trong những kỹ thuật cơ bản để đánh giá độ tin cậy phần mềm [6]–[8]. Để ước tính cũng như dự đoán độ tin cậy của các hệ thống phần mềm, dữ liệu lỗi cần phải được đo bằng các phương tiện khác nhau trong quá trình phát triển cũng như các giai đoạn vận hành sản phẩm. Phần mềm được mong đợi là đáng tin cậy hơn có thể được mô phỏng thông qua việc sử dụng mô hình tăng trưởng độ tin cậy phần mềm. SRGMs có thể ước tính số lỗi ban đầu, độ tin cậy của phần mềm, cường độ lỗi, khoảng thời gian trung bình giữa các lỗi, ... Lý tưởng nhất là các mô hình này cung cấp phương tiện mô tả quá trình phát triển và cho phép các chuyên gia về độ tin cậy phần mềm đưa ra dự đoán về độ tin cậy mong đợi trong tương lai của phần mềm đang được phát triển.

2.4. Các nghiên cứu liên quan đến mô hình độ tin cậy phần mềm

Một số mô hình phân tích đã được đề xuất để giải quyết vấn đề đo lường độ tin cậy phần mềm [7], [8]. Các phương pháp tiếp cận này dựa chủ yếu vào lịch sử thất bại của phần mềm và có thể được phân loại theo tính chất của quá trình thất bại được nghiên cứu như: Mô hình thời gian giữa các thất bại (Times between Failures Models), Mô hình đếm số thất bại (Failure Count Models), Mô hình gieo lỗi (Fault Seeding Models) và Mô hình dựa trên miền đầu vào (Input Domain Based Models). Phân loại mô hình độ tin cậy phần mềm theo các pha của vòng đời phát triển phần mềm được chỉ ra ở Hình 1, trong đó liệt kê các mô hình tin cậy phần mềm áp dụng ở các giai đoạn phát triển phần mềm [5].



Hình 1. Phân loại mô hình tin cậy phần mềm

Có rất nhiều mô hình độ tin cậy được rất nhiều nhà nghiên cứu đưa ra với hàng trăm mô hình khác nhau. Hiện tại, không mô hình nào trong số các mô hình đã công bố có thể được gọi là hoàn hảo hoặc tốt hơn mô hình khác. Jelinski và Moranda [39] đã

đề xuất mô hình độ tin cậy phần mềm đầu tiên và hàng trăm mô hình đã được giới thiệu cho đến nay [44]. Trong các SRGM hình chữ S của Yamada và cộng sự đưa ra cường độ thất bại thay đổi theo thời gian; Mô hình gỡ lỗi không hoàn hảo của Goel và Okumoto nhấn mạnh thực tế là không phải tất cả các lỗi trong hệ thống đều được loại bỏ khi chúng được phát hiện. Tỷ lệ phát hiện lỗi được mô tả bởi một hằng số [45]. Nhiều SRGM dựa trên NHPP đã được đề xuất trước đó. Yamada và cộng sự đã đề xuất một SRGM theo cấp số nhân xem xét hai loại lỗi. SRGM được đề xuất bởi Phạm và Zhang [45] giả định nhiều đặc điểm thất bại. Tác giả Nguyễn Hùng Cường [47], [48] đã thử nghiệm các phương pháp toán học để tính toán và sắp xếp các bộ chỉ số đo dựa trên các dữ liệu thực tế. Trong [46], tác giả đã sử dụng các dữ liệu thống kê trong 10 năm (2006-2015) để so sánh các nghiên cứu trong phát triển phần mềm và đề xuất, trong mọi bản phát hành, các lỗi hiện có được loại bỏ và các tính năng có giá trị được gửi đến khách hàng [46]. Sự tương tác giữa triển khai mới và hiện tại thường làm tăng nội dung lỗi của hệ thống, do đó làm giảm độ tin cậy của hệ thống. Trong thực tế, khi phần mềm được phát hành, nó chứa một số lỗi ẩn được báo cáo triển khai và được khắc phục trong các bản phát hành trong tương lai. Theo các nghiên cứu [4], [5], [11], [19], [20], [35], [50], ba mô hình Exponential model NHPP, Musa-Basic và mô hình Musa-Okumoto là những mô hình hữu ích và thành công nhất trong miền ứng dụng máy tính, được xem là các mô hình áp dụng chính xác nhất đối với nhiều dự án phần mềm [19]. Kiểm thử phần mềm và độ tin cậy phần mềm theo truyền thống thuộc hai mảng riêng biệt. Tuy nhiên hiện nay, có một mối liên kết chặt chẽ giữa kiểm thử phần mềm và độ tin cậy của phần mềm. Thuộc tính độ tin cậy không thể đo lường trực tiếp và do đó phải được lấy từ các phép đo khác như dữ liệu lỗi được thu thập trong quá trình kiểm thử. Kiểm thử phần mềm là một phương pháp hiệu quả để ước lượng độ tin cậy hiện tại, dự đoán độ tin cậy trong tương lai và cũng để cải thiện nó. Thuộc tính độ tin cậy của phần mềm chỉ có ý nghĩa nếu nó liên quan đến một người dùng cụ thể của hệ thống. Người dùng khác nhau trải nghiệm độ tin cậy khác nhau, bởi vì họ sử dụng hệ thống theo những cách khác nhau. Mặt khác, độ tin cậy phần mềm có thể được sử dụng để đo lường mức độ tiến bộ đã đạt được trong kiểm thử mức hệ thống. Trong giai đoạn lập trình và kiểm thử, mô hình NHPP, Musa, Nelson được vận dụng để đánh giá độ tin cậy phần mềm [5], [47], [48] (Hình 1). Trong phạm vi của nghiên cứu này chúng tôi chỉ tập trung nghiên cứu và vận dụng mô hình NHPP để đánh giá độ tin cậy của ứng dụng di động dựa trên việc áp dụng các kỹ thuật kiểm thử đã nghiên cứu ở các công bố trước đó trong các bài báo [21]–[25], [49].

3. Mô hình tăng trưởng độ tin cậy phần mềm NHPP áp dụng cho ứng dụng di động

Độ tin cậy trở nên rất quan trọng đối với sự phát triển của điện thoại thông minh, việc dự đoán và đảm bảo độ tin cậy của các ứng dụng sẽ trở thành vấn đề then chốt hiện nay [2], [15], [18]. Quy trình phát triển ứng dụng di động hiện nay chủ yếu là các quy trình linh hoạt như XP, Scrum, RAD [16], [17] và được phát triển bởi các nhóm nhỏ đảm nhận từ giai đoạn thiết kế đến thử nghiệm và phát hành. Phương pháp phát

triển Agile ít sai sót do yếu tố của con người so với các dự án lớn hay phương pháp phát triển khác.

Các mô hình tăng trưởng độ tin cậy phần mềm dựa trên quá trình Poisson không đồng nhất (NHPP) thường được phân thành hai nhóm. Nhóm đầu tiên chứa các mô hình, sử dụng thời gian thực hiện của máy hoặc thời gian lịch [20] làm đơn vị thời gian phát hiện / loại bỏ lỗi. Các mô hình như vậy được gọi là các mô hình thời gian liên tục. Nhóm thứ hai chứa các mô hình, sử dụng số lần kiểm tra / trường hợp làm đơn vị thời gian phát hiện lỗi. Các mô hình như vậy được gọi là các mô hình thời gian rời rạc [19], vì đơn vị thời gian phát hiện lỗi phần mềm có thể đếm được. Mô hình Goel Okumoto còn được gọi là mô hình NHPP theo cấp số nhân dựa trên các giả định sau đây [19], [50]:

- Số lần thất bại tích lũy theo thời gian t tuân theo quy trình Poisson.
- Số lỗi được phát hiện trong mỗi khoảng thời gian là độc lập đối với mọi tập hợp hữu hạn của các khoảng thời gian.
- Không có mã mới được thêm vào phần mềm trong quá trình thử nghiệm.
- Mỗi đơn vị thời gian thực hiện trong quá trình kiểm tra có khả năng tìm thấy một lỗi như nhau nếu cùng một mã được thực thi cùng một lúc
- Số lần phát hiện lỗi bất kỳ lúc nào tỷ lệ thuận với số lỗi hiện tại trong một thành phần của ứng dụng.
- Lỗi được loại bỏ ngay lập tức ngay khi được phát hiện, không có lỗi mới nào được đưa vào trong lúc loại bỏ lỗi.

Phương trình vi phân sau đây bao gồm các giả định trên. Trong đó $m(t)$ dự kiến số lần thất bại thành phần phần mềm theo thời gian t, a là hàm tổng số lượng lỗi, tức là tổng số lỗi được kỳ vọng của lỗi phát hiện ban đầu và lỗi được phát hiện theo thời gian t và b là tỷ lệ phát hiện lỗi trên mỗi lỗi tại thời điểm t.

$$\frac{\partial m(t)}{\partial t} = b[a - m(t)] \tag{6}$$

Lời giải cho giá trị trung bình của phương trình (6) được cho bởi công thức:

$$^{\wedge}m(t) = a\left(1 - e^{-bt_n}\right) \tag{7}$$

Hàm tính cường độ lỗi được cho bởi công thức:

$$\lambda(t) = abe^{-bt_n} \tag{8}$$

Các tham số ước lượng: Tham số a và b khác nhau cũng phản ánh các giả định khác nhau về các quy trình kiểm thử phần mềm. Trong phần này, chúng ta lấy ra một mô hình NHPP mới cho một hàm phụ thuộc tương quan giữa tham số a và b bởi một tham số chung từ một lớp mô hình tổng quát. Phương pháp phổ biến nhất để ước tính các tham số là phương pháp ước lượng hợp lý cực đại (MLE - Maximum Likelihood

Estimation). Phương pháp ước lượng MLE của một bộ sưu tập rộng về độ tin cậy của phần mềm. Để ước lượng a và b cho một mẫu n đơn vị, đầu tiên có được hàm số khả năng: lấy logarit tự nhiên ở cả hai bên. Phương trình ước lượng a và b được cho trong phương trình (9).

$$\boldsymbol{a} = \frac{\mathrm{y_n}}{(1 - e^{-bt_n})} \tag{9}$$

Với $\mathrm{y_n}$ là giá trị thực sự của lỗi thứ n^{th} quan sát được tại thời gian t. Tham số a có thể được ước lượng bằng phương pháp MLE dựa trên số lần thất bại trong một khoảng thời gian cụ thể. Giả sử khoảng thời gian quan sát $0, t_k$ được chia thành các khoảng phụ $(0, t_i], (t_1, t_2]. \ldots\ldots\ldots (t_{k-1}, t_k]$, phương trình 10 được sử dụng để xác định giá trị của b.

$$\frac{\mathrm{y_n t_n e^{-bt_n}}}{1 - \mathrm{e^{-bt_n}}} = \sum_{k=1}^{n} \left(\frac{(\mathrm{y_k} - y_{k-1}) \left(\mathrm{t_k e^{-bt_k}} - \mathrm{t_{k-1} e^{-bt} k} - 1 \right)}{\left(\mathrm{e^{-bt_{k-1}} - e^{-bt_k}} \right)} \right) \tag{10}$$

Số lỗi trên mỗi khoảng con được ghi là $n_i (i = 1, 2, 3, .., k)$ đối với số lần thất bại trong $(t_{i-1}, t_i]$. Các tham số a và b được ước tính bằng cách sử dụng Phương pháp Newton Raphson lặp đi lặp lại, được đưa ra trong Phương trình 11, 12 và 13.

$$b = b0 - \frac{f(b0)}{fr(b0)} \tag{11}$$

$$f(b) = \frac{\mathrm{y_n n^{et_n}}}{1 - \mathrm{e^{bt_n}}} - \sum_{k-1}^{n} \left(\frac{(\mathrm{y_k} - y_{k-1}) \left(t_k e^{-bt_k} - t_{k-1} e^{-bt_{k-1}} \right)}{\left(e^{-bt_{k-1}} - e^{-bt_k} \right)} \right) = 0 \tag{12}$$

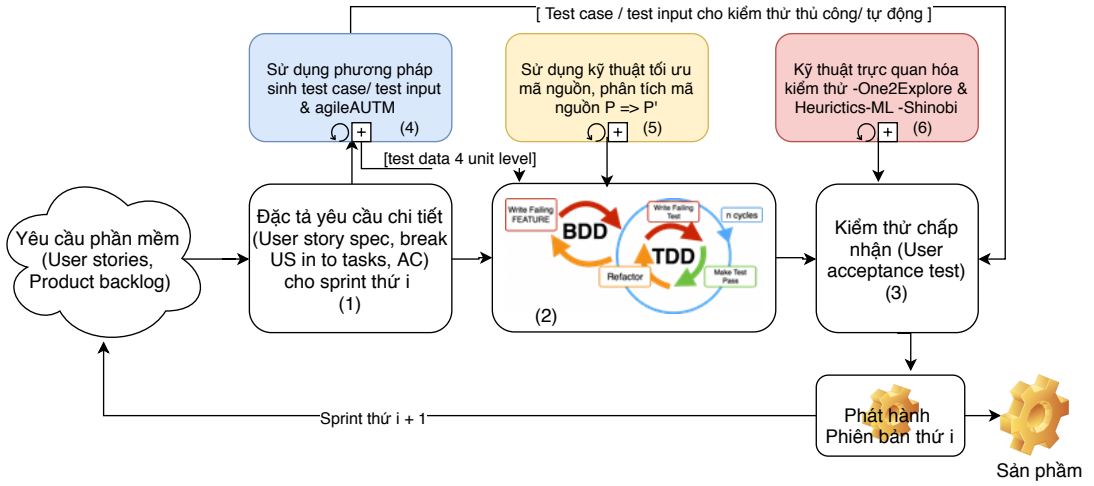
$$f'(b) = \sum_{k-1}^{n} \left(\frac{(\mathrm{y_k} - \mathrm{y_{k-1}}) \left(\mathrm{t_k} - \mathrm{t_{k-1}} \right)^{2_{\mathrm{c}}^{-\mathrm{b(k+t_{k-1})}}}}{\left(e^{-bt_{k-1}} - e^{-bt_k} \right)^2} \right) \tag{13}$$

4. Đề xuất áp dụng các kỹ thuật tối ưu mã nguồn và kiểm thử để nâng cao độ tin cậy cho ứng dụng di động

4.1. Quy trình phát triển ứng dụng di động theo cách tiếp cận Agile kết hợp các kỹ thuật kiểm thử

Phương pháp linh hoạt Agile Scrum là một trong những phương pháp hiệu quả nhất để phát triển phần mềm ứng dụng, đảm bảo công việc phối hợp của các chuyên gia và liên lạc liên tục giữa các thành viên trong nhóm và khách hàng. Cách tiếp cận này sẽ thúc đẩy lập kế hoạch thích ứng, phát triển tiến hóa, giao hàng sớm và cải tiến liên tục. Sự lặp lại và linh hoạt của phương pháp có thể được sử dụng trong các dự án phức tạp, nơi các yêu cầu của khách hàng thay đổi thường xuyên [13], [14], [16], [17]. Trong

nghiên cứu này chúng tôi đề xuất áp dụng các kỹ thuật kiểm thử bao gồm: phân tích mã nguồn và kiểm thử hộp trắng, tối ưu hóa mã nguồn, sinh dữ liệu kiểm thử vào trong từng giai đoạn phát triển của quy trình Agile Scrum.



Hình 2. Quy trình phát triển Scrum có đề xuất ứng dụng các kỹ thuật kiểm thử và tối ưu hóa mã nguồn

Quy trình Scrum cơ bản sẽ bao gồm các giai đoạn lấy yêu cầu, đề xuất yêu cầu bởi chủ sản phẩm (Product Owner -PO) bằng việc đưa ra các câu chuyện người dùng (User stories (US), Product backlog) [17]. Từ Product Backlog, PO và đội phát triển sẽ chọn lựa thực hiện những US nào được ưu tiên thực hiện trước để hình thành nên một sprint phát triển đầu tiên (i=1), mỗi sprint có thời gian từ 1 đến 4 tuần. Tại giai đoạn này (thành phần (1) trong Hình 2 ở trên), PO cùng với đội phát triển sẽ đặc tả chi tiết các câu chuyện người dùng (US), đưa ra các điều kiện chấp nhận (Acceptance criteria – AC) cho từng US, phân rã các US thành các công việc cụ thể để giao việc cho các thành viên của đội. Sau khi có đặc tả chi tiết sẽ sang giai đoạn lập trình và kiểm thử đơn vị (thành phần (2) ở Hình 2). Ở giai đoạn này, đội phát triển sẽ dùng phương pháp TDD (Test-Driven Development), BDD (Behavior Driven Development) để thực hiện "Viết và thực thi kiểm thử trước – Lập trình – Thực thi kiểm thử và tái cấu trúc mã nguồn". Kết thúc từng tính năng (feature/ mô đun) sẽ được chuyển sang giai đoạn kiểm thử chấp nhận (thành phần (3) ở Hình 2). Giai đoạn này người kiểm thử (và PO) sẽ thực hiện kiểm thử mức người dùng để đảm bảo yêu cầu phát hành phiên bản thứ nhất (hoặc thứ i, i=1,n). Kết thúc phát hành phiên bản thứ i, đội phát triển cùng PO tiếp tục chọn và phát triển các US cho phiên bản thứ i+1.

Đề xuất áp dụng các kỹ thuật như sau trong mỗi giai đoạn:

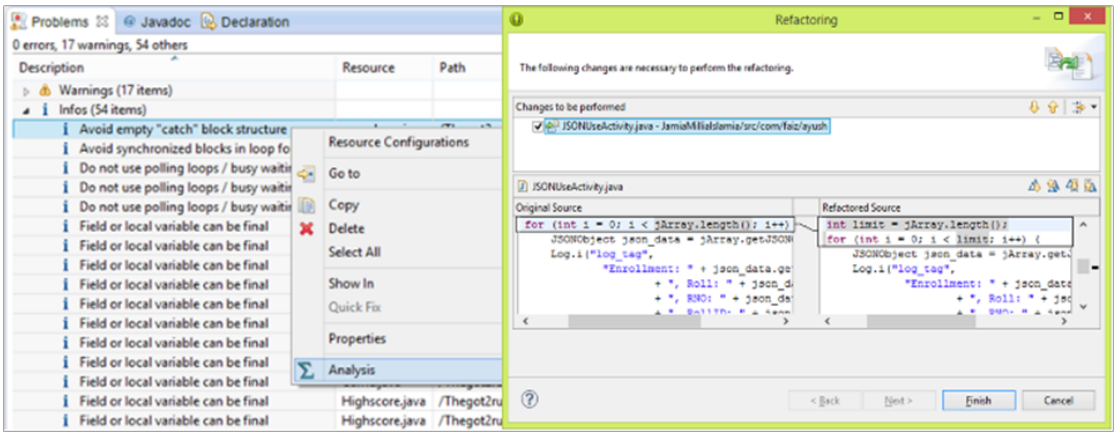
- Ở giai đoạn (1) chúng tôi đề xuất áp dụng kỹ thuật sinh ca kiểm thử (test case) và dữ liệu kiểm thử (test input) được suy diễn từ US và AC thông qua sử dụng phương pháp đặc tả hình thức. Kỹ thuật này (trong thành phần (4) của Hình 2) được mô tả ở phần 4.2 c), kết quả đầu ra của kỹ thuật này có thể được sử dụng ở giai đoạn (2) hoặc (3).

- Ở giai đoạn (2), kỹ thuật tối ưu mã nguồn bằng PMD và Android Lint, Kỹ thuật phân tích và kiểm thử mã nguồn Java được đề xuất áp dụng. Mô tả của kỹ thuật được trình bày trong phần 4.2.1) và 4.2.2), đây là thành phần (5) của Hình 2.
- Ở giai đoạn (3), kỹ thuật trực quan hóa kiểm thử hướng ngữ cảnh và kỹ thuật áp dụng Heurictics – học máy trong kiểm thử ứng dụng mobile web được đề xuất áp dụng (thành phần 6), hai kỹ thuật này sẽ được thực hiện ở các nghiên cứu tiếp theo trong tương lai.

Như vậy, trong Hình 2, các kỹ thuật được đề xuất ở (4), (5), (6) nhằm mục đích cải tiến, nâng cao hiệu năng, tăng chất lượng và độ tin cậy của ứng dụng di động, cho từng giai đoạn (1), (2), (3) theo quy trình Scrum cơ bản.

4.2. Các kỹ thuật kiểm thử đề xuất áp dụng

4.2.1. Kỹ thuật phân tích và tối ưu mã nguồn sử dụng PMD - Android lint: Mô hình độ tin cậy liên quan mật thiết đến quá trình phát hiện ra các lỗi của phần mềm. Quá trình này bao gồm các thời điểm phát hiện ra lỗi trong toàn bộ vòng đời phần mềm. Kỹ thuật tối ưu mã nguồn sử dụng PMD và Android lint, sử dụng cây cú pháp trừu tượng và đánh giá ảnh hưởng trong việc giảm thiểu các lỗi của hệ thống qua đó nâng cao độ tin cậy của ứng dụng di động, kỹ thuật này đã được nghiên cứu và công bố tại [21], [22]. Tối ưu mã nguồn trong Java: Khi kích thước của phần mềm tăng nhanh,



Hình 3. Màn hình của công cụ phân tích, tối ưu hóa và tái cấu trúc mã nguồn

các nhà nghiên cứu tập trung vào việc tối ưu mã nguồn để tiết kiệm tài nguyên. A. V. Aho [36] quan tâm đến việc tối ưu mã nguồn như là một vấn đề con của bài toán trình biên dịch. Ngày nay, một số trình dịch có thể tối ưu hóa mã nguồn ở trong một ngữ cảnh nào đó và được gọi là "mức trình dịch" (compile level). Tuy nhiên, việc tối ưu mã nguồn nên được thực hiện thủ công bởi các lập trình viên vì kỹ thuật này quá phức tạp để thực hiện tự động. Lập trình an toàn liên quan đến các vấn đề về lỗi: phát hiện, chịu lỗi, sao lưu, ... các biến cần được định nghĩa quyền truy cập, các đối tượng cần được thiết lập "uncloneable". Từ bài toán tối ưu hóa mã nguồn để nâng cao độ tin cậy cho ứng dụng di động được trình bày ở [21], chúng tôi xây dựng tập luật để đưa