Chapter 1: Bản chất của kiểm thử Tự động sai lầm là sai sót trong các công thức_interest cho khách hàng. Độ nguyên là sai sót trong mã □□. Nguyên nhân gốc là sản phẩm chủ sở hữu không có kiến thức về cách tính lãi, và kết quả là phàn nàn của khách hàng. Nguyên nhân gốc có thể được addressed bằng cách cung cấp thêm đào tạo về các công thức lãi cho chủ sở hữu sản phẩm, và có thể có thêm đánh giá của câu chuyện người dùng bởi các chuyên gia tính lãi. Nếu được thực hiện, thì sai sót trong các công thức lãi do câu chuyện người dùng không rõ ràng sẽ là một chuyện của quá khứ. Khảo sát nguyên nhân là được bài chi tiết hơn trong hai giấy chứng nhận ISTQB khác: Cấp bậc Chuyên gia Quản lý kiểm thử, và Cấp bậc Chuyên gia Tăng cường quan trắc kiểm thử. 1.3: Mười điều nguyên tắc kiểm thử Đối tượng học của Mười điều nguyên tắc kiểm thử (K2) Trong phần này, chúng ta sẽ xem xét 7 nguyên tắc cơ bản của kiểm thử đã được quan sát trong hơn 40 năm qua. Những nguyên tắc này, dù không phải lúc nào cũng được hiểu hoặc chú ý, đều xuất hiện trên hầu hết các dự án. Biết cách phát hiện những nguyên tắc này và cách lợi dụng chúng sẽ làm cho bạn thành một người kiểm thử tốt hơn. ngoài các mô tả của từng nguyên tắc sau đây, các bạn cũng có thể tham khảo Bảng 1.1 để có.reference nhanh của các nguyên tắc và nội dung như được viết trong Bản chất. Nguyên tắc 1: Kiểm thử cho thấy sự hiện diện của lỗi, không phải là sự vắng mặt của lỗi Như đã đề cập trong phần trước, mục tiêu thường của nhiều nỗ lực kiểm thử là tìm lỗi. Nhiều tổ chức kiểm thử mà tác giả đã làm việc với lại rất hiệu quả trong việc làm это. Một trong những khách hàng của chúng tôi lại có hiệu suất rất cao, cho thấy, trung bình, 99,5% của toàn bộ lỗi trong phần mềm có thể tìm thấy. Hơn nữa, những lỗi không được phát hiện lại ít quan trọng và không xảy ra thường xuyên trong sản xuất. Trong một số trường hợp,.test team của chúng tôi lại đã tìm thấy 100% của toàn bộ lỗi có thể quan tâm đến khách hàng, vì không có các lỗi không được báo cáo trước sau khi phát hành. Tuy nhiên, mức độ hiệu quả này không phổ biến. Dù vậy, không đội kiểm thử, kỹ thuật kiểm thử hoặc chiến lược kiểm thử nào có thể đảm bảo đạt được 100% tỷ lệ phát hiện lỗi (DDP) - hoặc ngay cả 95%, được coi là vượt trội.sequently, nó là quan trọng để hiểu rằng, dù kiểm thử có thể cho thấy sự hiện diện của lỗi, thì lại không có thể chứng minh là không có lỗi nào bị bỏ qua. Đương nhiên, khi kiểm thử tiếp tục, chúng ta giảm thiểu khả năng có các lỗi bị bỏ qua, nhưng cuối cùng, dạng hình Zeno's paradox lại xảy ra: mỗi lần chạy kiểm thử sau đó may cắt giảm rủi ro của một lỗi bị bỏ qua bằng một nửa, nhưng chỉ cần một số lượng vô hạn lần kiểm thử mới cắt giảm rủi ro xuống mức 0. Tuy nhiên, kiểm thử viên không nên tuyệt vọng hoặc cho phép niềm tin trong hoàn hảo. Dù kiểm thử không thể chứng minh rằng phần mềm hoạt động, nó có thể giảm thiểu rủi ro của chất lượng sản phẩm xuống mức được chấp nhận, như đã nhắc trước. Trong mọi nỗ lực đáng giá, có một mức độ rủi ro nào đó. Các dự án phần mềm - và kiểm thử phần mềm - đều là những nỗ lực đáng giá.
Chương 3: Mười điều nguyên tắc kiểm thử Nguyên tắc 1: Kiểm thử cho thấy sự hiện diện của lỗi, không phải là sự vắng mặt của lỗi Kiểm thử có thể cho thấy sự hiện diện của lỗi, nhưng không có thể chứng minh là không có lỗi nào bị bỏ qua. Kiểm thử giảm thiểu khả năng có các lỗi bị bỏ qua, nhưng ngay cả khi không tìm thấy lỗi nào, kiểm thử cũng không phải là chứng minh của sự đúng đắn. Nguyên tắc 2: Đánh giá thâm dụng là không khả thi Đánh giá toàn bộ (tất cả các kết hợp của đầu vào và điều kiện ban đầu) là không khả thi, trừ các trường hợp đơn giản nhất. Thay vì cố gắng đánh giá toàn bộ, nên sử dụngRisk analysis, các kỹ thuật kiểm thử và ưu tiên để focus vào công việc kiểm thử. Nguyên tắc 3: Đánh giá sớm tiết kiệm thời gian và tiền Để tìm lỗi sớm, cả các hoạt động kiểm thử tĩnh và hoạt động kiểm thử động đều nên được thực hiện càng sớm càng tốt trong chu kỳ phát triển phần mềm. Đánh giá sớm trong chu kỳ phát triển phần mềm giúp giảm thiểu hoặc xoá bỏ các thay đổi đắt tiền (xem Chương 3, Phần 3.1). Nguyên tắc 4: Lỗi tập trung lại Một số ít mô-đun thường chứa hầu hết các lỗi được phát hiện trong quá trình kiểm thử trước khi phát hành, hoặc họ chịu trách nhiệm về hầu hết các sự cố hoạt động. Dự đoán cluster lỗi và cluster lỗi được quan sát trong quá trình kiểm thử hoặc trong hoạt động, là một đầu vào quan trọng trong analysis rủi ro được sử

dụng để focus vào công việc kiểm thử (như được nhắc đến trong Nguyên tắc 2). Nguyên tắc 5: Cẩn thận với hiện tượng phá mọt Nếu các bài kiểm thử là được lặp lại nhiều lần,ließlich các bài kiểm thử này không còn tìm thấy các lỗi mới nữa. Để phát hiện các lỗi mới, các bài kiểm thử và dữ liệu kiểm thử hiện tại cần phải được thay đổi và các bài kiểm thử mới cần phải được viết ra. (Các bài kiểm thử không còn hiệu quả nữa, giống như thuốc trừ côn trùng không còn hiệu quả sau một thời gian). Trong một số trường hợp, như trong kiểm thử lặp lại tự động, hiện tượng phá mọt có kết quả có lợi, đó là số lượng các lỗi giảm sút. Nguyên tắc 6: Kiểm thử là tỉ lệ với ngữ cảnh Kiểm thử được thực hiện khác nhau trong các ngữ cảnh khác nhau. Ví dụ, phần mềm quan trọng về an toàn được kiểm thử khác với ứng dụng mobile thương mại điện tử. Ví dụ khác, kiểm thử trong dự án Agile được thực hiện khác với kiểm thử trong dự án chu kỳ phát triển tuyến tính (xem Chương 2, Phần 2.1). Nguyên tắc 7: Đạo đức không có lỗi là sai Một số tổ chức mong đợi rằng kiểm thử viên có thể chạy hết tất cả các bài kiểm thử và tìm hết tất cả các lỗi, nhưng Nguyên tắc 2 và Nguyên tắc 1, lần lượt, cho chúng ta biết rằng điều này là không khả thi. Ngoài ra, là sai khi ngờ rằng chỉ tìm thấy và sửa chữa một số lượng lớn các lỗi sẽ đảm bảo sự thành công của một hệ thống. Ví dụ, kiểm thử kỹ lưỡng tất cả các yêu cầu đã quy định và sửa chữa tất cả các lỗi tìm thấy có thể vẫn sản xuất một hệ thống khó dùng, không đáp ứng nhu cầu và kì vọng của người dùng hay kém hơn so với các hệ thống khác. BẢNG 1.1: Nguyên tắc kiểm thử
Chương 1: Bản chất của kiểm thử Nguyên tắc 2: Đánh giá thâm dụng là không khả thi Nguyên tắc này liên quan khá gần với nguyên tắc trước đó. Với bất kỳ hệ thống nào (từ small software đến system cỡ lớn), số lượng các testcase khả thi là vô hạn hoặc gần vô hạn đến mức gần như không thể đếm được. Vô hạn là một khái niệm khó khăn cho não bộ con người để hiểu hay chấp nhận, nên chúng ta hãy sử dụng ví dụ. Một trong những khách hàng của chúng tôi đã tính toán số lượng các kết hợp giá trị dữ liệu nội bộ trong hệ điều hành Unix là lớn hơn số lượng các phân tử trong vũ trụ bốn bậc. Họ thậm chí đã tính toán rằng, ngay cả với những bài kiểm thử tự động nhanh nhất, để thử tất cả các quan hệ trạng thái nội bộ sẽ cần thời gian dài hơn chu kỳ tuổi thọ của vũ trụ. ngay cả vậy, sẽ không phải là một kiểm thử đầy đủ của hệ điều hành; nó chỉ sẽ καλύ những kết hợp giá trị dữ liệu nội bộ. Chúng ta lại phải đối mặt với một đám mây lớn của các bài kiểm thử; chúng ta phải chọn một phần nào đó. Một cách để chọn bài kiểm thử là đi điệu láng trong đám mây các bài kiểm thử, chọn ngẫu nhiên cho đến khi hết thời gian. mặc dù có một chỗ nào đó cho kiểm thử tự động random, riêng biệt thì đó là một chiến lược kém. Chúng ta sẽ bàn luận về các chiến lược kiểm thử trong Chương 5, nhưng cho đến lúc này, hãy nhìn vào hai chiến lược sau. Một chiến lược để chọn bài kiểm thử là kiểm thử dựa trên rủi ro. Trong kiểm thử dựa trên rủi ro, chúng ta có một đội ngũ dự án và người dùng tập trung thảo luận phân tích rủi ro. Trong phân tích này, người dùng và người dùng giao tiếp đã xác định rủi ro về chất lượng của hệ thống, và đánh giá mức độ rủi ro (thường Sử dụng xác suất và ảnh hưởng) liên quan đến từng rủi ro. Chúng ta tập trung công việc kiểm thử dựa trên mức độ rủi ro, sử dụng mức độ rủi ro để xác định số lượng bài kiểm thử phù hợp cho từng rủi ro, và cũng để.sequence việc kiểm thử. Chiến lược khác để chọn bài kiểm thử là kiểm thử dựa trên yêu cầu. Trong kiểm thử dựa trên yêu cầu, người kiểm thử analý yêu cầu specification (được gọi là user stories trong dự án Agile) để xác định các điều kiện kiểm thử. Những điều kiện kiểm thử này thừa hưởng mức độ ưu tiên của yêu cầu hoặc user story nó được duy trì. Chúng ta tập trung công việc kiểm thử dựa trên mức độ ưu tiên để xác định số lượng bài kiểm thử phù hợp cho từng điểm, và cũng để.sequence việc kiểm thử. Nguyên tắc 3: Đánh giá sớm tiết kiệm thời gian và tiền Nguyên tắc này cho chúng ta biết rằng chúng ta nên bắt đầu kiểm thử càng sớm càng tốt để tìm càng nhiều lỗi càng nhiều. Ngoài ra, bởi vì chi phí tìm thấy và xóa bỏ lỗi tăng theo thời gian mà lỗi đó vẫn còn trong hệ thống, kiểm thử sớm cũng có nghĩa là chúng ta có thể giảm thiểu chi phí xóa bỏ các lỗi. Chúng ta có thể kết luận gì khi chúng ta kết hợp ba nguyên tắc này lại? Đầu tiên, chúng ta không thể tìm tất cả các lỗi, chỉ có thể tìm một phân tròn của chúng. Nguyên tắc thứ hai cho

chúng ta biết rằng chúng ta không thể chạy hết tất cả các bài kiểm thử. nguyên tắc thứ ba cho chúng ta biết rằng chúng ta nên bắt đầu kiểm thử sớm. Đừng hỏi, chúng ta có thể kết luận gì nếu chúng ta kết hợp ba nguyên tắc này lại?
Chương 3: Mười điều nguyên tắc kiểm thử Nguyên tắc 4: Lỗi tập trung lại Nguyên tắc này liên quan đến một vấn đề chúng ta đã thảo luận trước đó, rằng dựa vào chiến lược kiểm thử ngẫu nhiên trong đám mây các bài kiểm thử là kém hiệu quả. Lỗi không được phân bố ngẫu nhiên và đều đặn trên phần mềm đang được kiểm thử. Thay vào đó, lỗi tend to be found in clusters, with 20% (or fewer) of the total defects typically found in 80% of the code. Đây là một thông tin quan trọng khi chúng ta xây dựng kế hoạch kiểm thử. Do đó, chúng ta không thể dựa vào kiểm thử ngẫu nhiên để tìm các lỗi, mà phải bằng cách tập trung vào các vùng có thể có các lỗi nhất. FIGURE 1.3: Lãng phí thời gian của việc xóa bỏ lỗi sớm Yêu cầu Lỗi tìm thấy Thiết kế Mã/Mô-đun kiểm thử Kiểm thử tích hợp Lãng phí thời gian của việc xóa bỏ lỗi sớm Kiểm thử hệ thống Nguyên tắc 5: Cẩn thận với các kết hợp không mong đợi Nguyên tắc này cho chúng ta biết rằng chúng ta phải cẩn thận với các kết hợp không mong đợi trong các bài kiểm thử. Dưới đây là một ví dụ:... (đoạn văn tiếp tục)
Chương 1: Bản chất của kiểm thử Nguyên tắc 5: Cẩn thận với hiện tượng phá mọt Nguyên tắc này do Boris Beizer đề xuất [Beizer 1990]. Ông đã quan sát thấy rằng, giống như một loại thuốc diệt côn trùng được rải rác trên một mảnh đất sẽ giết chết những con vật càng ít càng ít mỗi lần được sử dụng, thì những bài kiểm thử giống đó cũng sẽ dần dần dừng lại tìm kiếm các lỗi mới khi được chạy lại trên một hệ thống đang được phát triển hay được bảo trì. Nếu các bài kiểm thử không cung cấp diện tích phủ sóng thích hợp, thì sự chậm dần trong việc tìm kiếm các lỗi mới sẽ dẫn đến một mức độ tin tưởng sai lầm và quá tự tin trong nhóm dự án. Tuy nhiên, không khí sẽ được thổi ra khỏi bóng khi hệ thống được phát hành cho khách hàng và người dùng. Sử dụng các chiến lược kiểm thử phù hợp là bước đầu tiên hướng đến đạt được diện tích phủ sóng thích hợp. Tuy nhiên, không có chiến lược nào là hoàn hảo. Bạn nên định kỳ xem lại kết quả kiểm thử trong suốt quá trình dự án và sửa đổi các bài kiểm thử dựa trên những phát hiện của bạn. Trong một số trường hợp, bạn phải viết lại các bài kiểm thử mới và khác nhau để kiểm định các phần khác nhau của phần mềm hay hệ thống. Những bài kiểm thử mới này có thể dẫn đến phát hiện các cụm lỗi chưa được biết trước, đó là một lý do không nên chờ đến cuối cuộc kiểm thử để xem lại kết quả kiểm thử và đánh giá về diện tích phủ sóng thích hợp. Hypothesis paradox là quan trọng khi thực hiện các hoạt động kiểm thử đa cấp đã được thảo luận trước đó về nguyên tắc của kiểm thử sớm. Đơn giản chỉ là lặp lại các bài kiểm thử của cùng các điều kiện không dẫn đến kết quả tốt trong việc phát hiện các lỗi. Tuy nhiên, khi sử dụng đúng cách, mỗi loại và cấp độ kiểm thử đều có các điểm mạnh và yếu điểm trong việc phát hiện các lỗi, và chúng ta có thể xây dựng một chuỗi các hoạt động kiểm thử hiệu quả để tìm kiếm các lỗi. Sau chuỗi các hoạt động kiểm thử này, chúng ta có thể tin tưởng rằng diện tích phủ sóng được đạt và mức độ rủi ro được chấp nhận.