Software Testing – Review

Review for the Final Exam

Review for the Final Exam

- [2] Software Engineering
- [3] Software Development Process
- [4] Black Box Testing Techniques
- [6] Which Box Testing Techniques
- [8] Evaluation of Testing and Reliability
- [9] Programming Tips and Techniques
- [12] Software Quality Management & Software Metrics
- [13] Bug Prediction and Test Plan

[2] Software Engineering (1/6)

- **Hệ thống máy tính**
  - Software và Hardware
  - Software: Vai trò là làm chủ Hardware (Nghĩa là tận dụng tối đa Hardware)
- **Kỹ thuật phần mềm (Software Engineering)**
  - Các lĩnh vực / ngành học để phát triển hiệu quả phần mềm chất lượng cao
  - Ước lượng man-month/hour/day cũng là 1 nhánh
  - The Mythical Man-Month: Thêm người vào 1 dự án đang trễ chỉ làm cho nó càng trễ thêm

[2] Software Engineering (2/6)

- **Độ tin cậy phần mềm (Software Reliability)**
  - Lỗi phần mềm gây ra lỗi nghiêm trọng trong đời sống
  - → Phần mềm cần phải được kiểm thử (testing) ở nhiều tình huống thực tế khác nhau
- **Đặc trưng phần mềm (Software Features)**
  - Khó nắm bắt tình hình thực tế
  - Tập trung vào quá trình phát triển
  - Thời gian vận hành và bảo trì lâu dài
  - Ít tái sử dụng

[2] Software Engineering (3/6)

- **Software Reuse (Tái sử dụng phần mềm)**
  - Blackbox reuse: Frameworks, Libraries
  - Whitebox reuse: Copy paste và chỉnh sửa lại
- **Good Software (Phần mềm tốt)**
  - User / Client:
    - Đáp ứng các yêu cầu đặc tả
    - Dễ sử dụng, thân thiện, nhanh, nhẹ, …
  - Dev:
    - Chi phí thấp và thời gian ngắn trong khi vẫn đáp ứng yêu cầu
    - Khả năng bảo trì dễ dàng

[2] Software Engineering (4/6)

Phân loại phần mềm

- Basic software
  - OS, Compiler/Interpreter, Service Programs (Utilities), …
- Application software
  - Phần mềm bảng tính, trình chiếu, …
- Middleware
  - DBMS, Web server, Application server …
- Embedded software
  - Phần mềm cho Thiết bị điện, ô tô, thang máy
  - Phần mềm cho Thiết bị sức khỏe, viễn thông, cảm biến, …

[2] Software Engineering (5/6)

Vấn đề

- Nhu cầu về các hệ thống ngày càng tăng, phần mềm ngày càng trở nên lớn và phức tạp hơn
- Chi phí phát triển và bảo trì phần mềm tăng lên theo thời gian

Giải pháp

- Thiết lập các lý thuyết và kỹ thuật liên quan đến phát triển phần mềm
- Hệ thống hóa thành các "kỹ thuật, tiêu chuẩn" chung → **Software Engineering**

[2] Software Engineering (6/6)

Khó khăn và hướng giải quyết

- Khó khăn trong phân tích yêu cầu
  → Tạo các Artifacts một cách rõ ràng và chi tiết, …
- Khó khăn trong tái sử dụng mã nguồn
  → Nắm bắt Design Patterns, System Architecture, …
- Khó khăn trong quản lý dự án
  → Theo dõi tiến độ, nhân sự, trao đổi giữa các bên liên quan, các quy trình phát triển tiên tiến (Agile), …
- Khó khăn trong việc ước tính thời gian và chi phí
  → Sử dụng các mô hình, phương pháp thống kê để ước lượng và dự đoán, …

[3] Software Development Process (1/8)

- **Quy trình phát triển phần mềm**
  - Development plan
  - Requirements analysis
  - Design
  - Implementation
  - Testing
  - Operation/Maintenance

[3] Software Development Process (2/8)

- **Kế hoạch phát triển (Development Plan)**
  - Lên kế hoạch (Setup)
    - Loại phần mềm
    - Thời hạn giao cho khách hàng
  - Ước lượng (Estimation)
    - Man-hours / chi phí
    - Khả năng mở rộng
  - Chuẩn bị (Preparation)
    - Nguồn nhân lực
    - Ngân sách
    - Môi trường phát triển

[3] Software Development Process (3/8)

- **Phân tích yêu cầu (Requirements Analysis)**
  - Yêu cầu nghiệp vụ
    - Yêu cầu chức năng
    - Yêu cầu phi chức năng
  - Mô tả yêu cầu
    - Mô tả chính xác, không mơ hồ
    - Xem xét tính khả thi: chức năng, chi phí, thời hạn
  - Artifact/Deliverable: Đặc tả cho các yêu cầu
    - Sơ đồ Use-case, Đặc tả Use-case, …
    - Class / Activity / Sequence Diagram, …
    - UI, Dev/Deploy Environment, …

[3] Software Development Process (4/8)

Artifact

- **Thiết kế (Design)**
  - Kiến trúc hệ thống, Cấu trúc chương trình
    - Đặc tả phầm mềm: Dependencies, Libraries, Frameworks, Services, …
    - Đặc tả phần cứng: Máy khách, máy chủ, môi trường mạng, …
  - Các chức năng, UI, cách vận hành và trao đổi của từng thành phần
  - Cấu trúc dữ liệu và giải thuật, …

[3] Software Development Process (5/8)

Artifact

- **Hiện thực (Implementation)**
  - Source code
  - Đặc tả chương trình
    - Các scripts thực thi, tệp cấu hình, …
    - Tài liệu kỹ thuật: Mô tả hđộng nội bộ của phần mềm (flowchart, kiến trúc, thuật toán, cấu trúc dữ liệu và UI), mô tả về API, …

[3] Software Development Process (6/8)

Artifact

- **Kiểm thử (Testing)**
  - Đặc tả kiểm thử (Test Specifications)
    - Test Plan (Cách tiếp cận, mục tiêu, phạm vi và lịch trình tổng thể của các hoạt động kiểm thử)
    - Test Scripts (Cho kiểm thử tự động)
    - Test Data (dữ liệu mẫu, các trường hợp/điều kiện biên và các kịch bản thực tế)
  - Báo cáo kiểm thử (Test Reports)
    - Các Test cases và Kết quả (Báo cáo lỗi, phạm vi/độ phủ của việc kiểm thử)
    - Các báo cáo về Hiệu suất, Bảo mật, sự chấp nhận của user

[3] Software Development Process (7/8)

Artifact

- **Vận hành & bảo trì (Operation & Maintenance)**
  - Tài liệu hướng dẫn
    - Cách cài đặt, định cấu hình, cách sử dụng các chức năng
    - Hướng dẫn khắc phục sự cố
    - Bảo trì và hỗ trợ
  - Báo cáo lỗi, Báo cáo phản hồi, …

[3] Software Development Process (8/8)

- **Development Process Model**
  - Thác nước (Waterfall)
    - Tiến hành tuần tự từ trên xuống
    - Dễ nắm bắt tiến độ nhưng giai đoạn sau cần chờ giai đoạn trước đó hoàn thành
  - Tạo mẫu (Prototyping)
    - Dựa trên các nguyên mẫu (prototype)
      → Để xác nhận và đánh giá thông số kỹ thuật và thiết kế
  - Xoắn ốc (Spiral)
    - Có sự kiểm tra và lặp lại ở các bước (theo chu kỳ)

[4] Black Box Testing Techniques (1/8)

- **Kiểm thử (Testing)**
  - Tìm lỗi
  - Test case < Test suite < Test domain
  - Kiểm thử hộp đen (Blackbox Testing)
    - Dựa trên các đặc tả
  - Kiểm thử hộp trắng (Whitebox Testing)
    - Dựa vào cấu trúc của chương trình
  - Kiểm thử ngẫu nhiên (Random Testing)
    - Tạo ra các test cases ngẫu nhiên và kiểm thử

Lớp tương đương: Chương trình sẽ xử lý chúng theo cùng một cách thức

[4] Black Box Testing Techniques (2/8)

- **Kỹ thuật Blackbox Testing 1 – Phân vùng tương đương (Equivalence Partitioning)**
  - Chia input thành các lớp tương đương bao gồm hợp lệ (Valid Input) và không hợp lệ (Invalid Input)
  - **B1**. Đặt lớp tương đương hợp lệ và lớp tương đương không hợp lệ cho từng điều kiện đầu vào.
  - **B2**. Tạo tất cả các kết hợp của **các lớp tương đương hợp lệ**
  - **B3**. Tạo tất cả các kết hợp **chỉ chứa một** lớp tương đương không hợp lệ

[4] Black Box Testing Techniques (3/8)

- **Kỹ thuật Blackbox Testing 1 – Phân vùng tương đương (Equivalence Partitioning)**
  - **Ví dụ:**
    - Input 1 có 3 lớp valid – 2 lớp invalid
    - Input 2 có 2 lớp valid – 1 lớp invalid
    - → Số test case valid: 3x2 = 6
    - → Số test case invalid: 2x2 + 1x3 = 7
    - **➔ Tổng: 13 (Đây là số lượng cơ bản của phương pháp này, còn nhiều biến thể khác sẽ có số lượng test cases khác)**
  - Thực hiện kiểm thử theo từng cột

[4] Black Box Testing Techniques (4/8)

- **Kỹ thuật Blackbox Testing 2 – Phân tích giá trị ranh giới / biên (BVA)**
  - Kiểm thử dựa trên các **giá trị biên/ranh giới** của các lớp tương đương hợp lệ và không hợp lệ
  - **B1**. Xác định miền giá trị cho từng điều kiện đầu vào
  - **B2**. Với từng input, xem xét 5 giá trị: ***[min, min+, nominal, max-, max]*** (nominal là giá trị trung bình)
  - **B3.** Cố định các giá trị nominal của n-1 biến và thực hiện kết hợp với từng giá trị của biến còn lại
  - Với Standard BVA, số test cases là **4n + 1**

[4] Black Box Testing Techniques (5/8)

- **Kỹ thuật Blackbox Testing 2 – Phân tích giá trị ranh giới / biên (BVA)**
  - Ví dụ: Giả sử có 3 biến đầu vào là X, Y, Z
    - **X: 0 → 100.**
    - **Y: 20 → 60.**
    - **Z: 80 → 100.**
  - Sử dụng Standard BVA để tạo các test cases

[4] Black Box Testing Techniques (6/8)

- **Kỹ thuật Blackbox Testing 2 – Phân tích giá trị ranh giới / biên (BVA)**
  - Ví dụ: 3 biến đầu vào là X, Y, Z đã cho
  - B1: Tìm các giá trị cho mỗi biến

	X	Y	Z
Min	0	20	80
Min+	1	21	81
Nominal	50	40	90
Max-	99	59	99
Max	100	60	100

[4] Black Box Testing Techniques (7/8)

- **Kỹ thuật Blackbox Testing 2 – Phân tích giá trị ranh giới / biên (BVA)**
  - Ví dụ: 3 biến đầu vào là X, Y, Z đã cho
  - B2: Cố định nominal của X, Y và tạo kết hợp với mỗi giá trị của Z
  - B3: Thực hiện tương tự cho các cặp còn lại: Y, Z và X, Z

[4] Black Box Testing Techniques (8/8)

- **Kỹ thuật Blackbox Testing 2 – Phân tích giá trị ranh giới / biên (BVA)**
  - Ví dụ: 3 biến đầu vào là X, Y, Z đã cho

Kết quả
(Lưu ý: Đây là Standard BVA, còn nhiều biến thể khác sẽ có số lượng test cases khác)

Test case #	X	Y	Z
1	50	40	80
2	50	40	81
3	50	40	90
4	50	40	99
5	50	40	100
6	0	40	90
7	1	40	90
8	99	40	90
9	100	40	90
10	50	20	90
11	50	21	90
12	50	59	90
13	50	60	90

[6] White Box Testing Techniques (1/3)

- **Tập trung vào cấu trúc của chương trình**
- **C0 (Instruction Coverage)**
  - Số dòng lệnh được thực thi ít nhất 1 lần ÷ Tổng số dòng lệnh
- **C1 (Branch Coverage)**
  - Số nhánh True, False được thực thi ít nhất 1 lần của các nhánh điều kiện (if, for, while) ÷ Tổng số nhánh True, False của các nhánh điều kiện
- **C2 (Condition Coverage)**
  - Số các kết hợp True / False **CÓ THỂ CÓ** giữa các nhánh điều kiện được thực thi ít nhất 1 lần ÷ Tổng số các kết hợp True / False **CÓ THỂ CÓ** của các nhánh điều kiện

[6] White Box Testing Techniques (2/3)

Mối quan hệ

- Tuân theo 1 chiều:
  C2 100% → C1 100%, C1 100% → C0 100%
  (Các trường hợp khác đều không đúng)
- Độ phủ đạt 100% (bất kể C0, C1, C2 hoặc cả 3) **KHÔNG** có nghĩa là chương trình không có lỗi
- Việc viết test case theo điều kiện nào trong C0, C1, C2 cũng sẽ dẫn đến độ chính xác và số lượng test case khác nhau
  → Cần xác định và thống nhất việc kiểm thử

[6] White Box Testing Techniques (3/3)

- **Random Testing**
  - Tạo tự động một lượng lớn các test cases
  - Tìm ra các kịch bản không ngờ đến
  - Không thể kiểm thử theo đặc tả
  - Tỷ lệ bao phủ thấp
- **Blackbox Testing**
  - Được dùng từ Unit Testing đến System Testing
- **Whitebox Testing**
  - Chủ yếu trong Unit Testing và Small Integration Testing

[8] Evaluation of Testing and Reliability (1/5)

- **Đánh giá hoạt động kiểm thử**
  - Dựa trên tỷ lệ phủ (Coverage rate)
  - Dựa trên tỷ lệ xử lý lỗi (Bug resolution rate)
    - Số bugs đã xử lý ÷ Tổng số bugs
    - Không biết chính xác Tổng số bugs → Ước lượng
  - Dựa trên vận hành (Operability)
    - MTBF (Mean Time Between Failures): Thời gian trung bình giữa các lần xảy ra lỗi

[8] Evaluation of Testing and Reliability (2/5)

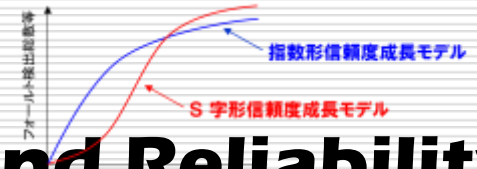
- **Phương pháp ước lượng Tổng số bug 1 – Capture / Recapture**
  - Chuẩn bị n bug nhân tạo
  - Giả sử kiểm thử tìm thấy x bug nhân tạo và y bug thật
  - Tổng số bug = $\frac{n\,y}{x}$
  - → Các lỗi nhân tạo thiếu thực tế và có chủ định trước

[8] Evaluation of Testing and Reliability (3/5)

- **Phương pháp ước lượng Tổng số bug 2 – Two-step Editing**
  - Số lỗi được tìm thấy bởi nhóm A = **a**
  - Số lỗi được tìm thấy bởi nhóm B = **b**
  - Số lỗi được tìm thấy trong cả hai nhóm = **c**
  - Tổng số bug = $\frac{ab}{c}$
  - Số bug còn lại = $\frac{ab}{c} - a - b + c$
  - → Tốn nhiều thời gian và chi phí

[8] Evaluation of Testing and Reliability (4/5)

- **Phương pháp ước lượng Tổng số bug 3 – Past Statistical Data (Density)**
  - Mật độ lỗi trước đó **x** [bugs/KLOC]
  - Quy mô phát triển **y** [KLOC]
  - Tổng số bug = x*y
- **Đường cong tăng trưởng (Growth Curve)**
  - Một đường cong mô hình cách dữ liệu mục tiêu thay đổi theo thời gian
  - Sử dụng giá trị tích lũy để bắt kịp tốc độ tăng trưởng của dữ liệu mục tiêu (số bugs)



[8] Evaluation of Testing and Reliability (5/5)

Software Reliability Growth Model (SRGM)

- 1 mô hình điển hình về mối quan hệ giữa thời gian kiểm thử và tổng số bug được phát hiện
  - Exponential SRGM: $a(1 - e^{-bt})$
  - Delayed S-shaped SRGM: $a\{1 - (1 + bt)e^{-bt}\}$
  - …
- Mục tiêu: Ước tính các tham số của đường cong để khớp với dữ liệu thực tế nhất có thể
- → Từ đó có thể ước lượng tổng số bug trong tương lai

[9] Programming Tips and Techniques (1/5)

- **Programming**
  - Là 1 bước trong quy trình phát triển phần mềm
  - Mục tiêu:
    - Đáp ứng các đặc tả yêu cầu
    - Dễ hiểu
    - Dễ bảo trì
  - Phương pháp:
    - Code conventions
    - Các quy định chung của nhóm, dự án, công ty, …

[9] Programming Tips and Techniques (2/5)

- **Dễ đọc và hiểu**
  - Thụt lề, Khoảng trắng, Dòng mới
  - Đặt tên hàm, tên biến mang ngữ nghĩa
- **Ngăn ngừa lỗi**
  - Đóng mở ngoặc đầy đủ
  - Không đặt tên biến trùng nhau
- **Cấu trúc đơn giản**
  - Tránh dùng goto
  - Tránh quá nhiều điều kiện rẽ nhánh

[9] Programming Tips and Techniques (3/5)

- **Độc lập với môi trường**
  - Sử dụng các cú pháp, hàm dùng chung
- **Comment thích hợp**
  - Tránh “Thiếu Why Thừa What”
- **Dễ dàng thay đổi**
  - Tránh Hardcode
  - Sử dụng các Macros

[9] Programming Tips and Techniques (4/5)

- **XP (eXtreme Programming)**
  - Dùng trong quy mô nhỏ (Thời gian ngắn và nhân lực ít)
- **XP Practice 1 – Test-first**
  - Viết các test cases trước rồi mới viết chương trình
  - Dùng trong Test-Driven Development
- **XP Practice 2 – Refactoring**
  - Tinh chỉnh code, tránh duplicate code, tách code, …
  - Tăng cường tính dễ đọc dễ hiểu
  - Cải thiện hiệu suất và khả năng bảo trì chương trình

[9] Programming Tips and Techniques (5/5)

XP Practice 3 – Pair Programming

- 1 người viết, 1 người quan sát/đánh giá/góp ý/trao đổi → Đề xuất cải tiến và sửa lỗi trong quá trình thực hiện, sau đó hoán đổi vị trí cho nhau
- Chia sẻ kiến thức/kinh nghiệm, phát hiện lỗi sớm

XP Practice 4 – Coding Standards

- Áp dụng các quy tắc về Code Conventions của ngôn ngữ sử dụng
- Một nhóm phải có các phương pháp viết mã chung, sử dụng cùng định dạng và phong cách viết mã

[12] Software Quality Management and Software Metrics (1/6)

- **Đảm bảo chất lượng phần mềm**
  - **Kiểm tra**
    - Kiểm thử đầy đủ trước khi release
    - Fix các lỗi phát hiện được
  - **Giám sát và cải tiến**
    - Ghi lại, phân tích, đánh giá và cải thiện các công việc khác nhau trong quá trình phát triển
  - **Bảo trì**
    - Đảm bảo phần mềm hoạt động một cách phù hợp
  - **Kiểm thử hồi quy (Regression Testing)**

[12] Software Quality Management and Software Metrics (2/6)

- **Bảo trì thích ứng**
  - Đáp ứng sự thay đổi của môi trường (OS, libraries, framework, …)
- **Bảo trì khắc phục**
  - Sửa các lỗi xảy ra trong lúc sử dụng phần mềm
- **Bảo trì khẩn cấp**
  - Sửa các lỗi bất chợp, tạm thời
- **Bảo trì tăng cường**
  - Cải thiện tính năng, yêu cầu mới
- **Bảo trì hoàn hảo**
  - Tinh chỉnh mã, cải thiện hiệu suất
- **Bảo trì phòng ngừa**
  - Rà soát và sửa các lỗi trước khi phát sinh

[12] Software Quality Management and Software Metrics (3/6)

- **TQC (Total Quality Control)**
  - Kiểm soát chất lượng với sự tham gia của tất cả các bên liên quan, bao gồm cả các nhà quản lý, lãnh đạo, …
- **Tập trung vào quản lý**
  - Ưu tiên: Quality, Cost, Delivery
- **Quản lý ngược dòng**
  - Xác định và giải quyết nguồn gốc ngược dòng vấn đề
- **Phòng ngừa tái diễn**
  - Học hỏi từ những vấn đề đã xảy ra trong quá khứ

[12] Software Quality Management and Software Metrics (4/6)

- **Nắm bắt vấn đề từ dữ liệu**
  - Số hóa những gì đang thực sự xảy ra và xử lý các vấn đề bằng cách đánh giá chúng một cách logic, khách quan và thống kê
- **Tiêu chuẩn hóa**
  - Manuals, Rules, Patterns
- **PCDA cycle**
  - Plan → Do → Check → Act

[12] Software Quality Management and Software Metrics (5/6)

- **Nguyên lý Pareto**
  - Quy tắc 80 : 20
  - 80% số lỗi của phần mềm đến từ 20% modules
  - → Biết được nơi cần cải thiện để đạt hiệu quả cao nhất
- **Chỉ số phần mềm (Software Metrics)**
  - Đo lường các đặc tính chất lượng phần mềm
  - LOC (Lines of Code) / KLOC
  - Cyclomatic Number (McCabe)
  - CALL, …

[12] Software Quality Management and Software Metrics (6/6)

- **Phân tích dữ liệu (Data Analysis)**
  - Histogram: Biểu đồ thể hiện tần suất của dữ liệu
  - Kernel Destiny Estimation:
    - Ước tính hàm mật độ xác suất của dữ liệu
    - Phân phối lệch phải (**Right-Skewed**) là phổ biến nhất
- **Mean và Median**
  - Mean: Giá trị trung bình của một tập dữ liệu
  - Median: Trung vị là giá trị ở trung tâm khi tập dữ liệu được sắp xếp từ nhỏ đến lớn (Nếu có 2 vị trí ở trung tâm thì cộng lại chia trung bình)

[13] Bug Prediction and Test Plan (1/4)

Fault-Prone Module Analysis

- Ý nghĩa: Tìm các đặc trưng của những module dễ bị lỗi theo số liệu
- Tập trung vào phân bố của lỗi
- Sử dụng thống kê để dự đoán xu hướng tổng thể

Phương pháp

- Chọn 1 loại dữ liệu (LOC, CC, CALL, …)
- Giả định nếu nó lớn hơn một giá trị nhất định → lỗi
- Chia dữ liệu thành 2 phần: Train và Test

[13] Bug Prediction and Test Plan (2/4)

- **Recall**
  - Số dự đoán đúng được dự đoán là lỗi / Tổng số lỗi thực tế
- **Precision**
  - Số dự đoán đúng được dự đoán là lỗi / Tổng số dự đoán
- **F-score**
  - Precision và Recall càng cao càng tốt. Tuy nhiên trong thực tế nếu ta điều chỉnh model để tăng Recall quá mức có thể dẫn đến Precision giảm và ngược lại.
  - Cần tìm 1 đại lượng trung hòa / cân bằng 2 giá trị này
    ➔ **F-score**: Trung bình điều hòa

[13] Bug Prediction and Test Plan (3/4)

Test Prioritization

- Kiểm thử dựa trên các giá trị chỉ số giảm dần
- Mục tiêu: Cần kiểm thử bao nhiêu phần trăm để có thể tìm thấy 80% tổng số lỗi? → Số càng thấp càng tốt

Dự đoán với nhiều số liệu

- Kết hợp LOC, CC, CALL, … trong cùng 1 mô hình
- Mỗi đại lượng sẽ có các trọng số khác nhau
- Các phép biến đổi như Logarit, Sigmoid, … là để đưa miền giá trị của hàm dự đoán y về [0,1]

[13] Bug Prediction and Test Plan (4/4)

- **Vấn đề**
  - Mô hình dự đoán sẽ không tốt nếu dữ liệu bị mất cân bằng (Overwhelming) → Dẫn đến Overfitting
- **Cách giải quyết:**
  - 1. Thay đổi trọng số của các đại lượng trong mô hình
  - 2. Thay đổi lượng dữ liệu để cân bằng lại
    - Undersampling: Giảm số lượng lớp đa số
    - Oversampling: Tăng số lượng lớp thiểu số

    → Thuật toán SMOTE là 1 trong số đó

Hỏi & Đáp

Cảm ơn!

Chúc các bạn thi tốt!