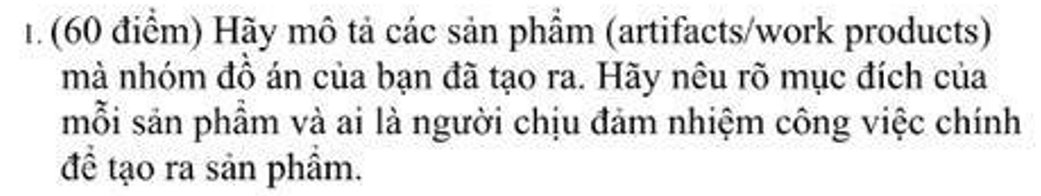
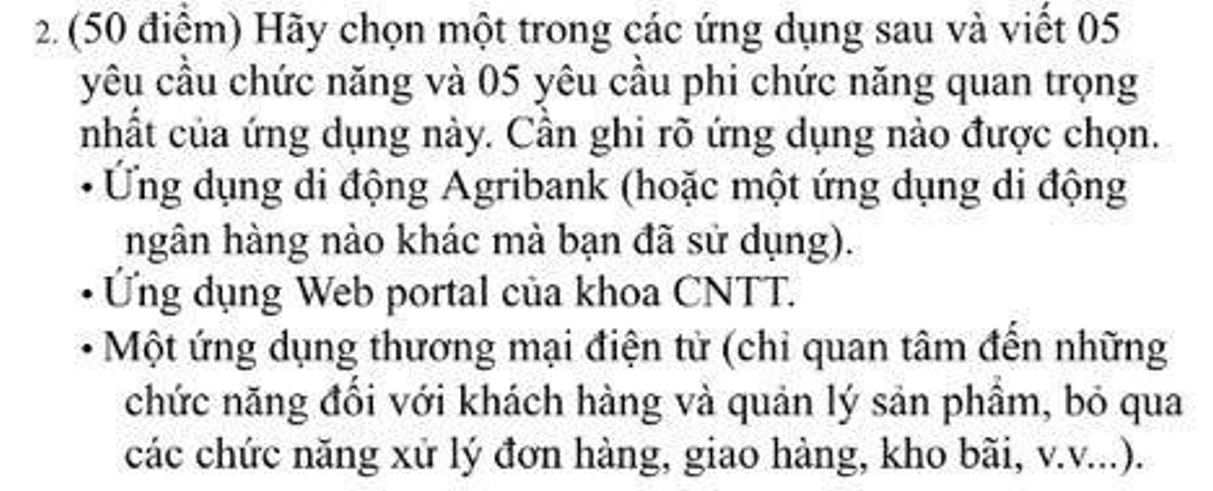
**NMCNPM**



* Sản phẩm của nhóm đồ án: Hệ thống online video conference như Zoom, Google meet.
* Mục đích: Cho phép tạo và tham gia cuộc họp trực tuyến, đặc biệt cần thiết trong làn sóng “Work from home” như hiện nay. Ngoài ra phần mềm cũng có những tính năng như bật webcam, chia sẻ màn hình, biểu tượng giơ tay, …
* Phân công công việc:
  + Em phụ trách về phần Backend, quản lý Database, viết báo cáo – các template. Ngoài ra cũng thực hiện phần Testing chung với cả nhóm.
  + Bạn Tiến: Phụ trách phần Frontend, thiết kế giao diện người dùng.
  + Bạn Bình và Trung thực hiện phần Backend, tổ chức các API, code các tính năng cần thiết của phần mềm.
  + Bạn An thực hiện xây dựng các requirement cho ứng dụng, quản lý chính việc testing.



* Ứng dụng Angribank:
  + Yêu cầu chức năng:
    - Chức năng tạo tài khoản, đăng nhập tài khoản.
    - Chức năng nạp / rút tiền.
    - Chức năng chuyển khoản.
    - Chức năng gửi tiết kiệm.
    - Chức năng báo lỗi giao dịch, trợ giúp khi gặp sự cố.
  + Yêu cầu phi chức năng:
    - Ứng dụng có hiệu năng cao, tốc độ chạy nhanh.
    - Ứng dụng có độ bảo mật cao, có các biện pháp bảo vệ cho tài khoản người dùng.
    - Giao dịch diễn ra nhanh chóng, an toàn, bảo mật.
    - Giao diện thân thiện với người dùng.
    - Có khả năng tương thích trên nhiều nền tảng, liên kết được với các ví điện tử.
    - Chi phí thực hiện và bảo trì rẻ.
* Ứng dụng Web portal:
  + Yêu cầu chức năng:
    - Chức năng tạo tài khoản, đăng nhập tài khoản.
    - Cập nhật thông tin cá nhân, chứng chỉ liên quan.
    - Đăng kí học phần.
    - Xem lịch thi, xem điểm thi, xem học phí.
    - Đánh giá môn học.
  + Yêu cầu phi chức năng:
    - Ứng dụng có hiệu năng cao, tốc độ chạy nhanh.
    - Bảo mật thông tin tốt.
    - Không bị nghẽn khi có lượng lớn truy cập vào cùng thời điểm.
    - Giao diện thân thiện với người dùng.
    - Có khả năng tích hợp với các trang web khác của trường, các khóa học online, …
    - Chi phí thực hiện và bảo trì rẻ.
* (30 points) Explain why Waterfall model can be more effective than Scrum if the project’s requirements are stable during the project.

Nếu yêu cầu về project ổn định, không thay đổi quá nhiều thì Waterfall có thể hoạt động tốt hơn Scrum. Mô hình Waterfall theo thứ tự tuần tự, công đoạn này hoàn thành mới bắt đầu công đoạn tiếp. Với các dự án ổn định, không có quá nhiều sự thay đổi thì Waterfall thực hiện đơn giản, ổn định hơn. Nó dễ hiểu, dễ triển khai, dễ đo lường các yếu tố như năng suất, … Với lại nếu ổn định,

* (30 points) Explain why Scrum is more effective than RUP if the project’s requirements change rapidly during the project?

Rational Unified Process (RUP)

Nếu yêu cầu project thay đổi nhiều, nhanh chóng thì Scrum tốt hơn. Do RUP cần xác định các scope, major milestones cũng như ngày cụ thể (Scrum thì dùng backlog chứ không phải scope). Ngoài ra RUP có 4 phases lớn trong vòng đời project còn Scrum thì dồn lại thành 1 lần lặp.

While both Scrum and Rational Unified Process (RUP) follow the Agile framework, RUP involves more formal definition of scope, major milestones, and specific dates (Scrum uses a project backlog instead of scope). In addition, RUP involves four major phases of the project lifecycle (inception, elaboration, construction, and transition), whereas Scrum dictates that the whole “traditional lifecycle” fits into one iteration.

* (30 points) What is the difference between high-level design and low-level design? Give one example output of each.

Với HLD, **phần cứng được mô hình hóa bằng chính ngôn ngữ dùng để phát triển phần mềm** như C/C++ hay SystemC. Điều này **giúp phần mềm và phần cứng có thể được tích hợp sớm để thử nghiệm và đánh giá hiệu quả và tính khả thi**. Sau khi hệ thống được tối ưu như yêu cầu, phần cứng mới bắt đầu được thiết kế trùng khớp với mô hình đã phát triển. Với cách này, phần cứng được đảm bảo tương thích phần mềm và đáp ứng tốt cho hệ thống mong muốn. Bên cạnh đó, phần mềm có thể tiếp tục được thử nghiệm và hoàn chỉnh song song với việc phát triển phần cứng.

Low level design giống như 1 phiên bản cụ thể hóa của high level design. Low-level design (LLD) là một quá trình trong quy trình phát triển phần mềm, trong đó **người lập trình xác định các chi tiết cụ thể về cách thức hệ thống sẽ thực hiện các tác vụ được yêu cầu**. Đây là một bước quan trọng trong quy trình phát triển phần mềm, vì nó cung cấp một khung cảnh rõ ràng về cách thức hệ thống sẽ hoạt động và cách thức nó sẽ đáp ứng các yêu cầu của người dùng.

Trong low-level design, người lập trình sẽ **xác định các module cần thiết để thực hiện các tác vụ được yêu cầu, các liên kết giữa các module và các giao diện giữa các module**. Người lập trình cũng sẽ xác định các thuộc tính của các module và các tham số cần thiết để thực hiện các tác vụ. Low-level design cũng có thể bao gồm việc xác định các thuật toán hoặc các giải thuật sẽ được sử dụng trong các module để thực hiện các tác vụ.

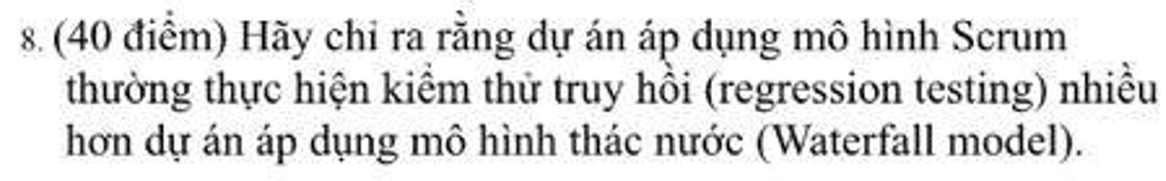
**The Low-level design is created after the High-Level Design**.

**(LLD tạo sau HLD)**

Trong high-level design, người lập trình sẽ xác định các tác vụ chính mà hệ thống sẽ thực hiện và cách thức chúng sẽ được thực hiện bởi hệ thống.

Sau khi high-level design đã hoàn thành, người lập trình sẽ tiến hành low-level design để xác định chi tiết hơn cách thức hệ thống sẽ thực hiện các tác vụ được yêu cầu. Low-level design sẽ xác định các module cần thiết để thực hiện các tác vụ, các liên kết giữa các module và các giao diện giữa các module. Nó cũng sẽ xác định các thuộc tính của các module và các tham số cần thiết để thực hiện các tác vụ.





Mục đích Regression test khi thay đổi cái mới vẫn đảm bảo chạy đc cái cũ, đảm bảo ko phát sinh lỗi trong cái cũ.

Tại sao nó time consuming:

- test all

- khi thay đổi là phải test lại hết

Regression test có 3 cách:

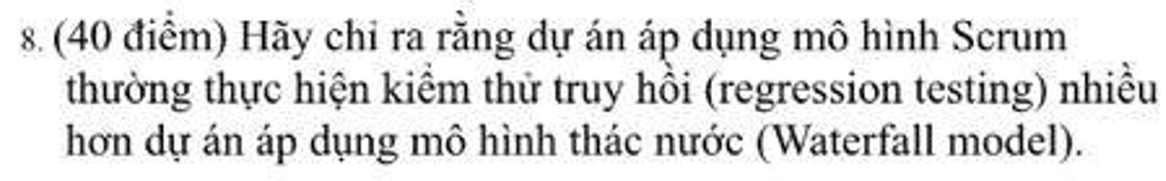
- retest all (test lại hết) Kiểm tra lại tất cả: Điều này giúp kiểm tra lại tính toàn vẹn của phần mềm từ trên xuống dưới.

- Lựa chọn kiểm tra hồi quy: Phương pháp này cho phép nhóm chọn một lựa chọn đại diện cho các thử nghiệm sẽ xấp xỉ một thử nghiệm đầy đủ. Điều này đòi hỏi ít thời gian hoặc chi phí hơn nhiều so với bộ thử nghiệm đầy đủ

- Ưu tiên cho trường hợp thử nghiệm: Với các trường hợp thử nghiệm hạn chế, hãy thử ưu tiên các thử nghiệm có thể ảnh hưởng đến cả bản dựng hiện tại và tương lai của phần mềm.

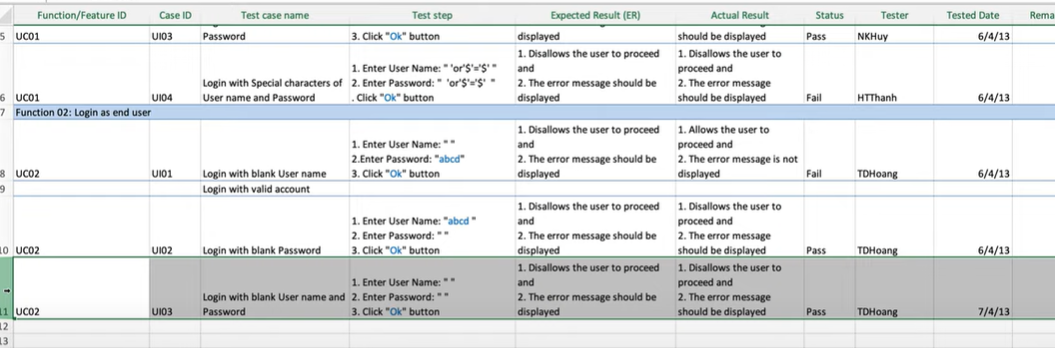
**Thực hành tốt nhất trong thử nghiệm hồi quy**

1. Duy trì lịch biểu: Chọn lịch kiểm tra mà bạn có thể duy trì trong suốt vòng đời phát triển phần mềm. Điều này tránh các trường hợp kiểm tra được đặt trên ổ ghi phía sau.
2. Sử dụng công cụ quản lý kiểm tra: Theo dõi chính xác tất cả các kiểm tra được thực hiện một cách thường xuyên và có hồ sơ lịch sử về hiệu suất của chúng theo thời gian. Làm điều này bằng cách sử dụng một công cụ quản lý kiểm tra đơn giản của một số loại. Điều này có thể dẫn đến các quyết định hiệu quả hơn về những trường hợp thử nghiệm phải được thực hiện. Nó có thể giúp xác định chính xác những cải tiến và thay đổi trong các trường hợp thử nghiệm. Nó cũng có thể thúc đẩy phân tích rõ ràng hơn về kết quả kiểm tra hồi quy.
3. Đánh giá mức độ ưu tiên của thử nghiệm: Thử nghiệm hồi quy khó khăn hơn khi phạm vi ứng dụng có phạm vi lớn và có các mức tăng hoặc vá liên tục cho hệ thống.



|  |  |
| --- | --- |
| In agile testing, regression testing is carried out after every iteration. | In waterfall testing, regression testing is carried out only in the end. |

**5 test case của login**



**Các nguyên nhân khiến cho các phương pháp phát triển phần mềm truyền thống dễ dẫn đến thất bại hơn so với các phương pháp phát triển nhanh bao gồm:**

1. Các yêu cầu không rõ ràng: Trong các phương pháp phát triển truyền thống, rất khó để khách hàng và người phát triển phần mềm hiểu rõ những yêu cầu của dự án. Khi yêu cầu không rõ ràng, có thể dẫn đến việc phát triển sai đối tượng hoặc không đáp ứng được những yêu cầu của khách hàng, dẫn đến thất bại của dự án.
2. Không có sự linh hoạt trong quy trình: Trong các phương pháp phát triển truyền thống, quy trình phát triển được xác định trước và không có sự linh hoạt để thay đổi khi có sự thay đổi trong yêu cầu hoặc môi trường. Điều này có thể dẫn đến việc không thể đáp
3. Không có sự tham gia động của khách hàng: Trong các phương pháp phát triển truyền thống, khách hàng thường không được tham gia quá nhiều trong quá trình phát triển. Điều này có thể dẫn đến việc không đáp ứng được những yêu cầu của khách hàng, và khi đó dự án có thể dễ dẫn đến thất bại.
4. Thời gian phát triển dài: Trong các phương pháp phát triển truyền thống, thời gian phát triển dự án thường rất dài vì phải tuân thủ một quy tr
5. Thị truwofnmg biến động
6. Phần mềm không đáp ứng được yêu cầu thay đổi: Do các yêu cầu không rõ ràng và không có sự linh hoạt trong quy trình, các phương pháp phát triển truyền thống thường không thể đáp ứng được sự thay đổi trong yêu cầu của khách hàng. Khi phần mềm không thể đáp ứng được yêu cầu thay đổi, dự án có thể dẫn đến thất bại.

***Top 20 nguyên nhân dẫn đến lỗi phần mềm***

***https://viblo.asia/p/tai-sao-pham-mem-lai-co-loi-Eb85oJYkl2G?fbclid=IwAR1tRvB\_MKQdc9KFu4MGvGCgHNUVm9SvkuKiTmNqEsJtUM9WcfAmtd2LODc***

*1. Hiểu lầm trong giao tiếp hoặc không có giao tiếp*

*Thành công của ứng dụng phần mềm nào cũng đều dựa vào sự giao tiếp giữa các bên liên quan, đội ngũ phát triển và đội ngũ kiểm thử. Những yêu cầu không rõ ràng và những hiểu lầm về các yêu cầu là hai lý do chính dẫn đến khiếm khuyết trong phần mềm. Ngoài ra, những khiếm khuyết phát sinh trong giai đoạn phát triển còn là do đội ngũ phát triển không được thông báo chính xác về các yêu cầu.*

*2. Sự phức tạp của phần mềm*

*Sự phức tạp của các ứng dụng phần mềm hiện hành có thể trở nên rất khó hiểu đối với bất kỳ ai không có kinh nghiệm trong lĩnh vực phát triển phần mềm hiện đại. Các giao diện loại Windows, cấu trúc máy khách-máy chủ và các ứng dụng phân tán, truyền thông dữ liệu, cơ sở dữ liệu quan hệ khổng lồ, và kích thước của các ứng dụng đã góp phần làm tăng thêm sự phức tạp của phần mềm / hệ thống. Và việc áp dụng những kỹ thuật hướng đối tượng có thể phức tạp hóa, thay vì đơn giản hóa, một dự án, trừ khi dự án đó được thiết kế tốt.*

*3. Lỗi lập trình*

*Các lập trình viên cũng là con người, nên việc họ mắc sai sót là điều bình thường. Không phải thành viên nào của đội phát triển cũng là chuyên gia về miền. Những lập trình viên thiếu kinh nghiệm hay không có kiến thức thích hợp về miền có thể mắc phải những lỗi rất đơn giản trong khi lập trình. Việc thiếu thực hành các thao tác lập trình đơn giản, kiểm thử đơn vị hay gỡ lỗi là một số nguyên nhân thường thấy dẫn đến các lỗi phát sinh trong quá trình phát triển.*

*4. Thay đổi yêu cầu*

*Khách hàng có thể không hiểu được ảnh hưởng mà những thay đổi đó gây ra, hoặc là họ hiểu nhưng vẫn yêu cầu thay đổi - thiết kế lại, thay đổi nhân sự, ảnh hưởng đến những dự án khác, những đầu việc đã hoàn thành có thể cần làm lại hoặc bỏ đi, những yêu cầu về phần cứng có thể bị ảnh hưởng, v.v...Khi thay đổi xảy ra, dù lớn hay nhỏ thì những mắt xích giữa các phần của dự án cũng rất có khả năng sẽ tương tác với nhau và gây ra vấn đề; và quá trình giám sát những thay đổi sẽ rất phức tạp và dễ dẫn đến lỗi. Qua đó, tinh thần làm việc của đội ngũ nhân sự cũng có thể bị ảnh hưởng.  
Trong một vài môi trường kinh doanh hay thay đổi thì yêu cầu thay đổi liên tục có thể là một thực tế của cuộc sống. Trong trường hợp này, đội ngũ quản lý phải hiểu rõ được những nguy cơ sẽ xảy ra, và đội ngũ QA và kỹ sư kiểm thử phải thích ứng và lên kế hoạch cho việc kiểm thử rộng rãi và liên tục để đảm bảo là các lỗi không thể tránh khỏi sẽ không vượt khỏi tầm kiểm soát.*

*5. Áp lực thời gian*

*Việc lên lịch cho các dự án phần mềm là khó nhất, và thường đòi hỏi con người phải phỏng đoán rất nhiều. Khi hạn chót gần kề và khủng hoảng xuất hiện thì sai lầm là điều rất khó tránh khỏi. Lịch trình không thực tế, mặc dù không phổ biến, nhưng lại là mối quan tâm chính trong các dự án quy mô nhỏ / công ty và dễ dẫn đến lỗi phần mềm. Nếu không có đủ thời gian để thiết kế, coding và kiểm thử một cách hoàn chỉnh thì chắc chắn phần mềm sẽ có lỗi.*

*6. Người tự đại hoặc quá tự tin*

*Là người thường nói những câu như: ‘Không vấn đề’ ‘dễ như ăn bánh mà’ ‘chỉ cần vài tiếng là tôi có thể hoàn thành’ ‘nâng cấp đoạn code cũ đó cũng dễ ấy mà’ thay vì những câu như: ‘chuyện đó sẽ khiến tất cả phức tạp lên rất nhiều và có thể dẫn đến một loạt lỗi sai’ ‘chúng tôi không chắc là có thể làm được không, nhưng chúng tôi sẽ thử xem’ ‘chừng nào chưa xem xét kỹ thì tôi chưa thể ước lượng xem việc này sẽ mất bao lâu’  
‘chúng tôi không thể tìm hiểu được là đoạn "code lẩu" (spaghetti code) đó ban đầu có tác dụng gì’ Nếu có quá nhiều câu “không vấn đề” không thực tế như vậy thì kết quả sẽ là một loạt các lỗi phần mềm.*

*7. Code được tài liệu hóa kém*

*Rất khó để duy trì và sửa đổi code được viết hoặc được ghi lại với chất lượng kém; và kết quả sẽ là lỗi phần mềm. Trong các tổ chức, phía quản lý thường không thể hiện sự khuyến khích để các lập trình tài liệu hóa code của họ hoặc viết những đoạn code rõ ràng, dễ hiểu. Thực tế còn chứng minh điều ngược lại: họ giành được nhiều điểm hơn khi viết code nhanh hơn, và nếu không có ai hiểu thì đã có cớ để bảo vệ việc làm của mình (‘nếu viết ra đã khó thì hiểu được còn khó hơn’) Bất kỳ lập trình viên nào mới làm việc với những đoạn code này cũng sẽ lúng túng bởi sự phức tạp của dự án và những đoạn code được tài liệu hóa kém. Và thường thì việc tạo ra những thay đổi nhỏ trong những code được tài liệu hóa kém đó sẽ mất thời gian hơn nhiều vì trước khi thay đổi bất cứ đoạn code nào, lập trình viên cũng sẽ phải học hỏi rất nhiều.*

*8. Các công cụ phát triển phần mềm*

*Các công cụ trực quan, các thư viện lớp, trình biên dịch, công cụ kịch bản, v.v...thường cũng có lỗi riêng hoặc được tài liệu hóa rất kém, dẫn đến việc sản sinh thêm lỗi. Các lập trình viên phần mềm thường dùng những công cụ phần mềm thay đổi liên tục, nhưng việc giữ bắt kịp với các phiên bản khác nhau và khả năng tương thích của chúng là một vấn đề rất lớn đang hiện hữu.*

*9. Tập lệnh tự động hóa lỗi thời*

*Việc viết tập lệnh tự động hóa mất rất nhiều thời gian, nhất là đối với những kịch bản phức tạp. Nếu đội ngũ tự động đã ghi lại hoặc viết một tập lệnh kiểm thử nào nhưng lại quên cập nhật nó trong một khoảng thời gian thì nó sẽ trở nên lỗi thời. Nếu việc kiểm thử tự động hóa không xác nhận kết quả đúng thì nó sẽ không thể nắm bắt được các khiếm khuyết.*

*10. Thiếu kiểm thử viên có kỹ năng*

*Các kiểm thử viên có kỹ năng và kiến thức miền chắc là nhân tố cực kỳ quan trọng đối với thành công của bất kỳ dự án nào; nhưng không phải công ty nào cũng có một đội ngũ kiểm thử viên có kinh nghiệm 100%. Kiến thức về miền và khả năng tìm ra lỗi của kiểm thử viên đóng vai trò rất lớn trong việc tạo nên một phần mềm chất lượng cao. Việc thiếu một trong hai nhân tố này đều có thể dẫn đến một phần mềm nhiều lỗi. Dưới đây là một vài nguyên nhân dẫn đến lỗi phần mềm nữa. Các nguyên nhân này chủ yếu được áp dụng cho chu kỳ kiểm thử phần mềm:*

1. *Không thiết lập thử nghiệm thích hợp (môi trường thử nghiệm) để kiểm tra tất cả các yêu cầu.*
2. *Viết code hoặc kiểm thử các trường hợp khi chưa hiểu rõ tất cả các yêu cầu.*
3. *Thiết kế sai dẫn đến việc trong tất cả các giai đoạn của chu kỳ phát triển phần mềm đều có vấn đề.*
4. *Thường xuyên phát hành các bản vá lỗi phần mềm trong khi chưa hoàn thành việc kiểm tra vòng đời của phần mềm.*
5. *Không cung cấp những kiến thức hay tài nguyên liên quan đến những kỹ năng cần dùng trong việc phát triển và kiểm thử phần mềm một cách phù hợp.*
6. *Dành ít thời gian hoặc bỏ qua hoàn toàn việc kiểm tra hồi quy.*
7. *Không tự động hóa các trường hợp kiểm thử lặp lại và luôn phụ thuộc vào kiểm thử viên trong việc kiểm tra thủ công.*
8. *Không ưu tiên thực hiện kiểm tra.*
9. *Không theo dõi quá trình phát triển và kiểm thử một cách liền mạch. Những thay đổi vào phút cuối rất có thể sẽ dẫn đến lỗi.*
10. *Những giả thiết sai trong giai đoạn mã hoá và thử nghiệm.*