Chương 10

Phân tích và giải thích kết quả kiểm thử

10.1 Một số thuật ngữ

Lúc bắt đầu kiểm thử, các testcase đều được ghi nhận là chưa được kiểm thử (unattempted).

Nếu kết quả kiểm thử thỏa mãn đầy đủ kết quả kỳ vọng, testcase sẽ chuyển về trạng thái đã kiểm thử và thành công (attempted and successful).

Nếu chỉ 1 phần kết quả kiểm thử phù hợp với kết quả kỳ vọng, testcase sẽ chuyển về trạng thái đã kiểm thử nhưng chưa thành công (attempted but unsuccessful).

1 trong các công việc chính của quản lý kiểm thử là theo dõi trạng thái của từng testcase vì trạng thái của các testcase là 1 chỉ thị rõ ràng về tiến độ kiểm thử : nếu còn 90% testcase chưa được kiểm thử, ta nói quá trình kiểm thử chỉ mới bắt đầu, nếu chỉ còn 10% testcase chưa được kiểm thử, ta nói quá trình kiểm thử sắp kết thúc.

Ước lượng ban đầu về lịch kiểm thử : Số testcase được kiểm thử trong 1 khoảng thời gian và số người kiểm thử sẽ cho người quản lý biết tiến độ kiểm thử tốt hay quá chậm. Thí dụ 15 testcase được kiểm thử trong 2 tuần đầu (10 ngày làm việc), ta tính được tốc độ kiểm thử là 1.5 testcase/ngày. Nếu kế hoạch cần kiểm thử 100 testcase, ta phải tốn 100/1.5 = 67 ngày (tức 14 tuần).

Nhưng khi kiểm thử một số testcase, ta có thể phát hiện lỗi, do đó ta phải tốn thời gian sửa lỗi và kiểm thử lại testcase đó lần 2, 3, ... Do đó thời gian kiểm thử sẽ lâu hơn nhiều so với ước lượng ban đầu về lịch kiểm thử.

Một số testcase sẽ về trạng thái "được kiểm thử nhưng không thành công" vì kết quả thu được không đúng theo kết quả kỳ vọng hay vì máy bị dừng trước khi hoàn thành kiểm thử testcase đó.

Thách thức cho người quản lý là lập thứ tự ưu tiên cho việc sửa lỗi và kiểm thử lại các testcase này :

- Ứng với testcase làm máy bị dừng khi chưa cho kết quả thì nên ưu tiên cho việc sữa lỗi nó và kiểm thử lại ngay.
- Còn các testcase kiểm thử làm TPPM chạy được nhưng cho kết quả không giống với kỳ vọng thì sẽ lập thứ tự ưu tiên cho việc sửa lỗi. Các tester thường dùng 4 mức 1 tới 4 để đánh giá mức độ cần sửa : mức 1 là tầm trọng nhất và mức 4 là nhẹ nhất và có thể bỏ qua.

Lịch kiểm thử và kết quả tuần đầu

Test case execution schedule					
Date	Test case ID	Date atttempted	Outcome	Severity	Date corrected
Week 1	FT-001	5 Jul	Successful		
	FT-002	6 Jul	Successful		
	FT-003	6 Jul	Successful		
	FT-004	6 Jul	Unsuccessful	1	
	FT-005	7 Jul	Successful		
	FT-006	7 Jul	Successful		
	FT-007	7 Jul	Unsuccessful	2	
	FT-008	7 Jul	Unsuccessful	1	
	FT-009	7 Jul	Unsuccessful	1	
	FT-010	8 Jul	Successful		
Week 2	FT-011				
	FT-012				
	FT-013				
	FT-014				
...	...				
Week 14	FT-095				
	FT-096				
	FT-097				
	FT-098				
	FT-099				
	FT-100				

Kết quả phân tích tuần kiểm thử đầu tiên

Test case execution progress—week 1				
100	Total test cases to attempt			
10	Test cases attempted to date			
10%	Percent attempted to date			
6	Test cases attempted—successful			
60%	Percent test cases successful			
4	Test case attempted—unsuccessful			
40%	Percent test case unsuccessful			
	Severity 1s	3	75%	
	Severity 2s	1	25%	
	Severity 3s	0	0%	
	Severity 4s	0	0%	

10.2 Khám phá lỗi dựa vào các lỗi riêng lẻ tìm được

Người kiểm thử phát hiện từng lỗi theo thời gian. Mỗi khi lỗi được phát hiện, nó có thể được sửa và kiểm thử lại. Vùng code cần kiểm thử có thể chạy được cho toàn bộ testcase.

Nhưng thường gặp hơn là khi kiểm thử lại lỗi vừa sửa, ta phát hiện lỗi khác và phải sửa nó. Điều này có thể lặp lại nhiều lần cho đến khi kiểm thử xong testcase tương ứng.

Cũng có thể ta phải kiểm thử nhiều testcase khác nhau phục vụ cho những mục tiêu khác nhau trên cùng vùng code xác định, và mỗi testcase sẽ giúp phát hiện các lỗi khác nhau.

Tóm lại, việc kiểm thử thành công 1 testcase riêng lẻ chưa đảm bảo vùng code được kiểm thử tương ứng không còn lỗi. Việc kiểm thử/ phát hiện lỗi/sửa lỗi/kiểm thử lại theo kiểu tăng tiến là cách chính yếu để người kiểm thử giúp đở người phát triển viết đúng phần mềm thỏa mãn các yêu cầu đặt ra.

Cách thức và mức độ theo dõi các lỗi tìm được, sửa chữa và kiểm thử lại sẽ quyết định độ hiệu quả của việc kiểm thử. Ta có thể xây dựng bảng theo dõi lỗi bằng worksheet Excel hay bằng 1 tiện ích cao cấp hơn.

Lịch kiểm thử & bảng theo dõi việc xử lý lỗi

Test case execution schedule

Date	Test case ID	Date attempted	Outcome	Severity	Date corrected
Week 1	FT 001	5 Jul	Successful		
	FT 002	6 Jul	Successful		
	FT 003	6 Jul	Successful		
	FT 004	6 Jul	Unsuccessful	1	
	FT 005	7 Jul	Successful		
	FT 006	7 Jul	Successful		
	FT 007	7 Jul	Unsuccessful	2	
	FT 008	7 Jul	Unsuccessful	1	
	FT 009	7 Jul	Unsuccessful	1	
	FT 010	8 Jul	Successful		
Week 2	FT 011				
	FT 012				

Defect tracking log

Defect ID	Date discovered	Test case ID	Severity	Date assigned for correction	Date corrected
SD-0001	6 Jul	FT-004	1		
SD-0002	7 Jul	FT-007	2		
SD-0003	7 Jul	FT-007	2		
SD-0004	7 Jul	FT-007	3		
SD-0005	7 Jul	FT-007	2		
SD-0006	7 Jul	FT-008	1		
SD-0007	7 Jul	FT-009	1		

Mỗi lỗi được gán 1 chỉ số đánh giá mức 1 độ tầm trọng. Mức độ tầm trọng của lỗi phụ thuộc vào loại lỗi và thời điểm lỗi xảy ra ở đâu trong chu kỳ phát triển phần mềm.

Thường ta dùng 3 loại lỗi sau đây :

- loại 1 : lỗi ngăn cản việc kiểm thử.
- loại 2 : lỗi ngăn cản việc phát triển phần mềm.
- loại 3 : lỗi ngăn cản việc triển khai/phân phối phần mềm.

Càng về cuối chu kỳ phát triển phần mềm thì loại lỗi 3 nên có mức độ tầm trọng ngày càng cao.

10.3 Khám phá lỗi dựa vào backlog

Chúng ta đã giới thiệu 2 thước đo hoạt động kiểm thử :

- thước đo 1 : số lượng và tỉ lệ testcase đã thử/testcase trong kế hoạch cho ta biết ta đang ở giai đoạn nào của quá trình kiểm thử.

- thước đo 2 : số lượng và tỉ lệ testcase bị lỗi/testcase đã kiểm thử cho ta biết mức độ tìm lỗi của người kiểm thử.

Bây giờ ta sẽ giới thiệu thước đo thứ 3 : danh sách các lỗi chưa được sửa (backlog).

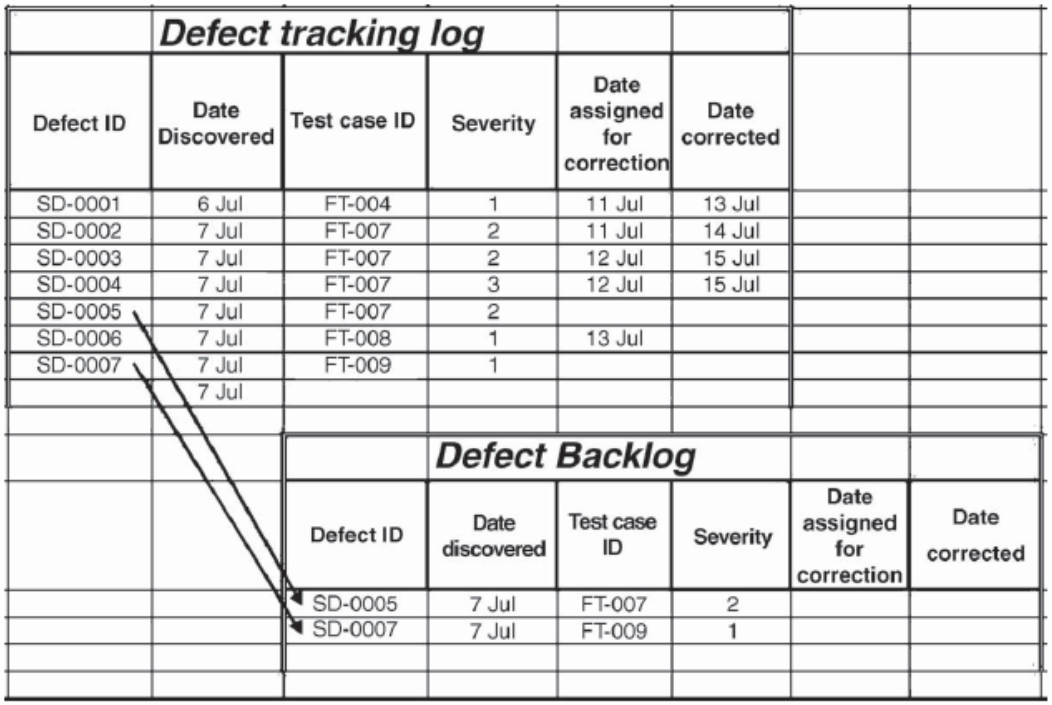
Định nghĩa Danh sách lỗi chưa sửa (Backlog)

Nếu việc kiểm thử đã phát hiện 300 lỗi và người phát triển đã sửa được 100 lỗi thì danh sách các lỗi chưa sửa sẽ chứa 200 lỗi.

Thách thức cho đội phát triển là xác định xem có đủ thời gian, tài nguyên lập trình và kiểm thử để kiểm thử các lỗi còn lại sao cho backlog không còn chứa lỗi nào. Câu trả lời thường là "không thể".

Thách thức kế tiếp là xem lại backlog để lập thứ tự mức độ tầm trọng của lỗi, dựa vào đó các lỗi càng nặng thì nên được sửa đầu tiên. Hy vọng rằng khi thời gian phát triển và kiểm thử đã hết thì backlog sẽ được tối thiểu hóa và chỉ chứa các lỗi không ảnh hưởng đến chức năng nghiệp vụ của chương trình.

Các lỗi trong backlog sẽ được sửa và phân phối trong service pack hay trong version kế tiếp của phần mềm.



Defect tracking log

Defect ID	Date Discovered	Test case ID	Severity	Date assigned for correction	Date corrected
SD-0001	6 Jul	FT-004	1	11 Jul	13 Jul
SD-0002	7 Jul	FT-007	2	11 Jul	14 Jul
SD-0003	7 Jul	FT-007	2	12 Jul	15 Jul
SD-0004	7 Jul	FT-007	3	12 Jul	15 Jul
SD-0005	7 Jul	FT-007	2		
SD-0006	7 Jul	FT-008	1	13 Jul	
SD-0007	7 Jul	FT-009	1		
	7 Jul				

Defect Backlog

Defect ID	Date discovered	Test case ID	Severity	Date assigned for correction	Date corrected
SD-0005	7 Jul	FT-007	2		
SD-0007	7 Jul	FT-009	1		

Một thách thức khác cho đội phân tích backlog là backlog được cập nhật theo chu kỳ (thí dụ hàng tuần), các lỗi mới phát hiện (do việc sửa lỗi trước đây tạo ra) có thể được đánh giá là tầm trọng và được ưu tiên để ở đầu backlog để cần sửa gấp.

Ở điểm nối này có thể xảy ra tranh cải nảy lửa về chất lượng phần mềm vì chưa có chuẩn công nghiệp nào về vấn đề này.

Trong phạm vi của môn học, chúng ta đánh giá chất lượng phần mềm dựa vào những gì phần mềm thực hiện được so với những đặc tả đề ra ban đầu. Cụ thể, nếu đội phát triển phần mềm đã viết được phần mềm thỏa mãn các đặc tả chức năng và đội kiểm thử đã xác thực được điều này thì ta nói phần mềm có chất lượng tốt.

10.4 Khám phá lỗi dựa vào chùm lỗi

Mã earmark của lỗi và công dụng

Bây giờ ta xem xét sự dung hoà của giá/lợi ích mang lại khi thêm 1 field thông tin mới vào từng record theo dõi lỗi trong bảng theo dõi lỗi. Ta gọi field này là mã earmark : mã nhận dạng vùng code chứa lỗi.

Chi phí quản lý field earmark không cao :

- Các thành viên đội phát triển phải thống nhất qui tắc tạo và đọc mã này. Thường trong đặc tả phần mềm, ta dùng 1 mã để nhận dạng 1 module, 1 class, 1 hàm chức năng, mã này cũng có thể dùng làm mã earmark cho lỗi tìm được.
- Các thành viên đội phát triển phải điền giá trị đúng vào field này cho từng lỗi trong bảng theo dõi lỗi mà tester tạo ra (sau khi họ sửa lỗi này).

	Defect tracking log					
Defect ID	**Date discovered**	**Test case ID**	**Severity**	**Date assigned for correction**	**Date corrected**	**code earmark**
SD-0001	6 Jul	FT-004	1	11 Jul	13 Jul	GL106
SD-0002	7 Jul	FT-007	2	11 Jul	14 Jul	AP234
SD-0003	7 Jul	FT-007	2	12 Jul	15 Jul	AP234
SD-0004	7 Jul	FT-007	3	12 Jul	15 Jul	AP236
SD-0005	7 Jul	FT-007	2			
SD-0006	7 Jul	FT-008	1	13 Jul		
SD-0007	7 Jul	FT-009	1			

Phương pháp phân tích nguyên nhân gốc (root cause analysis) là 1 phương pháp toán học trong lĩnh vực thống kê được trình bày khá phổ biến trong các sách về toán thống kê. Người ta đã dùng khá nhiều phương pháp này để phân tích danh sách lỗi phát hiện của dự án phần mềm.

Ý tưởng cơ bản của phương pháp này trong phân tích lỗi phát hiện là giúp tìm kiếm và xác định các chùm lỗi lớn nhất (buggiest) :

- tính số lượng lỗi/tỉ lệ lỗi theo từng mã earcode.
- lập thứ tự từ cao xuống thấp.

Phương pháp này rất hữu dụng trong khoảng thời gian từ 1/3 tới 1/2 thời gian trong kế hoạch kiểm thử vì lúc này phần mềm có độ ổn định rất thấp, nên cần nhiều sửa chữa và cập nhật.

Kết quả phân tích nguyên nhân gốc của 1 dự án phần mềm trong giai đoạn đầu của công đoạn hiện thực, lúc này đội phát triển đã sửa được 2000 lỗi.

Defect tracking log

Root cause analysys of defects

Corrected during preliminary construction

May 1 thru Aug 15

2000 total defects corrected during preliminary construction

Code earmarks	Defects corrected per code earmark	Percent defects corrected per code earmark
AP234	450	22.5%
AP745	310	15.5%
GL106	150	7.5%
AR218	75	3.8%
PY512	10	0.5%
BK459	8	0.4%
DP113	8	0.4%
All others	989	less than 0.4%
Total	2000	

Phân tích số liệu của bảng trong slide trước, ta thấy :

- cần thiết phải xem lại các đoạn code và bảng thiết kế liên quan gây ra 3 nhóm lỗi AP234, AP745 và GL106 (vì chúng chiếm tới 45.5% lỗi) trước khi tiến hành bước hiện thực kế tiếp. Lưu ý thêm là 2 nhóm lỗi AP234, AP745 có mối quan hệ rất khắng khích vì chúng thuộc cùng 1 module chức năng.
- còn các lỗi thuộc nhóm AR218 có tỉ lệ tương đối thấp nên có thể được xem xét hay không tùy thuộc vào đội kiểm thử đã tìm hiểu và nắm vững về vùng code gây lỗi chưa.
- Có thể bỏ qua việc xem lại vùng code chứa các lỗi thuộc các nhóm còn lại vì chúng chiếm tỉ trọng quá nhỏ.

Sở dĩ xuất hiện các nhóm lỗi chứa nhiều lỗi là vì :

- thiết kế không chính xác với chức năng hoặc thiết kế chưa đầy đủ.
- viết code chưa đủ.
- debug code chưa đủ.
- dùng chuẩn lập trình không tốt.
- người lập trình chưa thông thạo và nắm vững khả năng của ngôn ngữ lập trình.
- người lập trình chưa nắm vững giải thuật phức tạp được dùng để giải quyết chức năng.

Việc xem lại vùng code chứa nhiều lỗi cũng tốn thời gian (thí dụ 3 tuần) và làm cho bước kế tiếp bị delay, tuy nhiên cái lợi thì lớn hơn nhiều :

- trong việc xem lại vùng code/bảng thiết kế, ta có thể tìm ra được 1 số nguyên nhân gốc gây ra đồng thời nhiều lỗi, chỉ cần sửa và cập nhật các nguyên nhân gốc này thì rất nhiều lỗi đã được sửa.
- và khi kiểm thử lại, số lỗi thuộc các vùng code liên quan sẽ giảm thiểu rất lớn. Thí dụ ta chỉ còn phát hiện 50 lỗi (so với 1500 lỗi của lần kiểm thử trước). các lỗi tìm được này có xu hướng không phải là lỗi nặng và có thể được đưa vào danh sách lỗi chưa sửa nên không cần sửa gắp chúng, có thể để cho bước phát triển sau đó thực hiện.

Ta có thể lặp lại việc phân tích nguyên nhân gốc của 1 dự án phần mềm trong các giai đoạn sau đó, thí dụ tại giai đoạn cuối của phát triển phần mềm, đội phát triển đã sửa được 1500 lỗi.

Defect tracking log

Root cause analysys of defects

Corrected during Final construction in progress

Sept 15 – Nov 15

1500 total defects corrected during Final construction in progress

Code earmarks	Defects corrected per code earmark	Percent defects corrected per code earmark
AR218	30	2.0%
PY512	25	1.7%
BK459	15	1.0%
DP113	8	0.5%
AP234	5	0.3%
GL106	4	0.3%
All others	1413	less than 0.3%
Total	1500	

Bây giờ ta thấy :

- 3 nhóm lỗi quan trọng nhất chỉ chiếm tỉ lệ rất nhỏ (4.7%), điều này nói lên phần mềm đã khá ổn định.
- các nhóm lỗi quan trọng của bước trước là AP234 và GL106 vẫn còn trong danh sách, nhưng với tỉ lệ rất nhỏ. Điều này cho thấy tính hiệu quả của việc dùng mã earmark và việc xem xét lại vùng code tương ứng.
- do các nhóm lỗi quan trọng nhất chỉ chiếm tỉ lệ rất nhỏ nên việc phân tích nguyên nhân gốc ở bước này là không có tác dụng lớn như lúc trước nữa. Mặc dù vậy, thông tin earmark vẫn hữu dụng cho việc quản lý danh sách lỗi chưa sửa (backlog).

Defect backlog		
Root cause analysys of defects		
Not corrected as of Dec 15		
1 Total defect—severity 1	Included in analysis	
24 Total defects—severity 2	Included in analysis	
475 Total defects—severity 3 and 4	Not included in analysis	
500 Total defects not corrected as of Dec 15		
Code earmark guesses	**Defects awaiting correction per code earmark**	**Percent defects awaiting correction per code earmark**
AR477 (includes severity 1)	13	52.0%
GL105	5	20.0%
PY632	3	12.0%
GL498	2	8.0%
GL551	1	4.0%
GL600	1	4.0%
Total	25	

10.5 Dùng mẫu về phát hiện lỗi của project trước

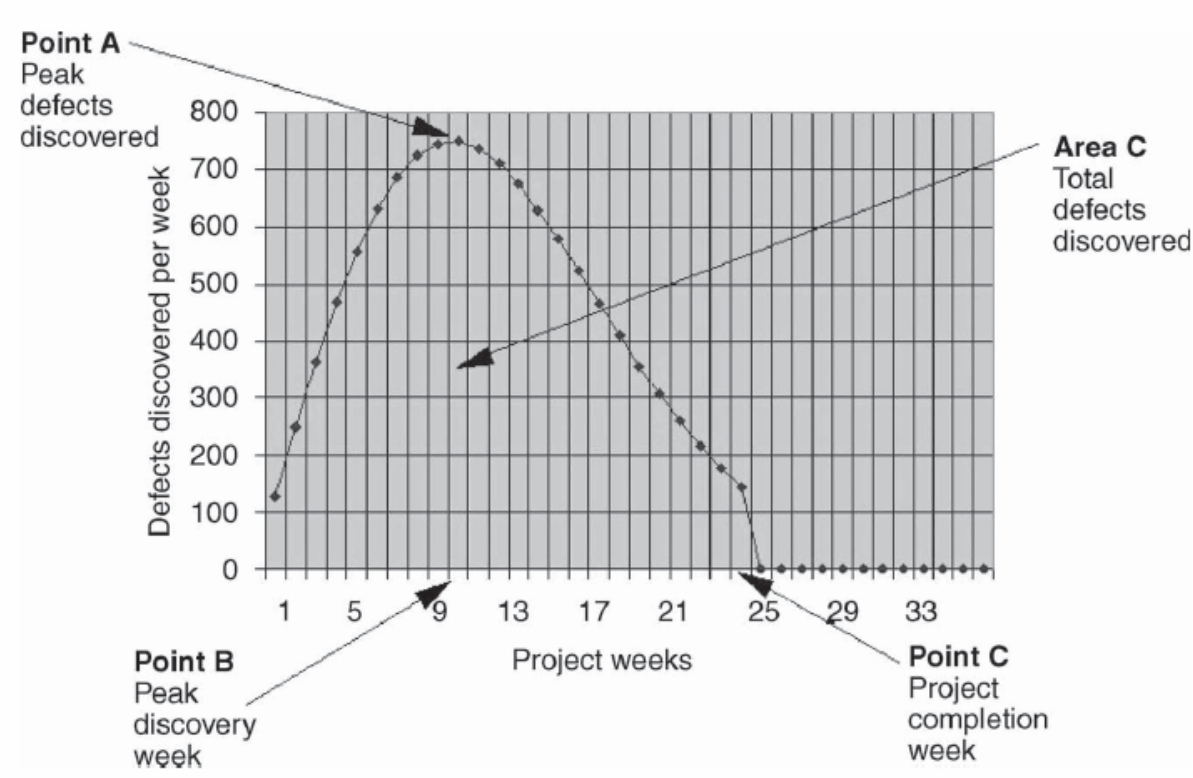
Việc tiên đoán số lỗi mà đội kiểm thử sẽ phát hiện được trong dự án hiện hành là rất quan trọng, nó giúp ta đưa ra kế hoạch, chuẩn bị tài nguyên kiểm thử hiệu quả. Nếu không dựa trên thông tin nào, hiện nay chưa có phương pháp nào giúp ta tiên đoán chính xác được.

Tuy nhiên, có 1 số phương pháp dựa vào yếu tố lịch sử sẽ giúp ta tiên đoán với độ chính xác nằm trong phạm vi lệch 10%.

Phương pháp tiên đoán dựa vào yếu tố lịch sử đặc biệt thích hợp cho các công ty phần mềm lâu đời, họ đã phát triển thành công nhiều phần mềm với nhiều kích cở khác nhau theo thời gian.

Càng có nhiều thông tin trong record miêu tả lỗi trong project trước càng cho ta tiên đoán với độ chính xác cao hơn. Chúng ta hãy bắt đầu bởi 2 thông tin cơ bản trong record lỗi : mã lỗi và ngày phát hiện.

Đường biểu diễn lỗi phát hiện trong project 200KLOC, được kiểm thử trong 24 tuần.



Đường cong miêu tả việc phát hiện lỗi của project trước cho chúng ta những thông tin chung cho hầu hết các project phần mềm :

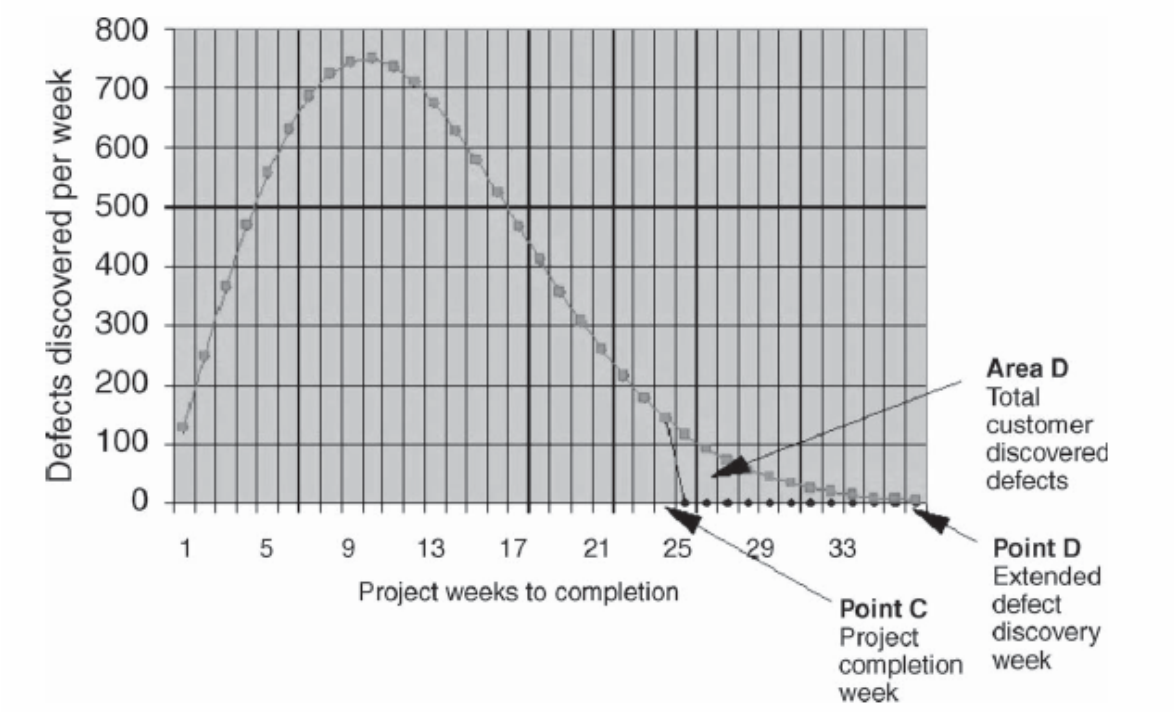
- lúc đầu số lỗi phát hiện được thường rất nhiều và lên nhanh tới đỉnh ở thời điểm khoảng 1/3 thời gian phát triển phần mềm (giai đoạn đầu này độ ổn định của phần mềm còn rất thấp).
- sau đó số lỗi được phát hiện giảm dần khi phần mềm ngày càng hoàn chỉnh và ổn định.
- điểm A miêu tả số lỗi được phát hiện nhiều nhất (là 749 lỗi).
- điểm B miêu tả tuần phát hiện lỗi nhiều nhất (là tuần 10).
- điểm C miêu tả tuần kết thúc việc phát triển phần mềm (tuần 24).

- vùng C dưới đường cong (diện tích của vùng) miêu tả tổng số lỗi tìm được trong dự án phần mềm (11,497).

Có nhiều nghiên cứu nói rằng : nếu ta tìm được lỗi càng nhiều và càng sớm thì lỗi được phát hiện bởi khách hàng sau này càng ít. Lưu ý rằng chi phí sửa lỗi do khách hàng phát hiện được là rất lớn.

Nghiên cứu này còn nói là nếu ta nội suy đường biểu diễn lỗi phát hiện về phía phải trục x đến khi tiếp xúc trục x thì vùng nới rộng này (vùng D) sẽ sắp xỉ bằng số lỗi mà khách hàng sẽ phát hiện được sau đó.

Biểu đồ nới rộng được vẽ ở slide kế cho ta thấy điểm D tiếp xúc với trục x ở tuần 36, số lỗi ở vùng D là 487 lỗi.



10.6 Sử dụng biểu đồ phân loại các nhóm lỗi

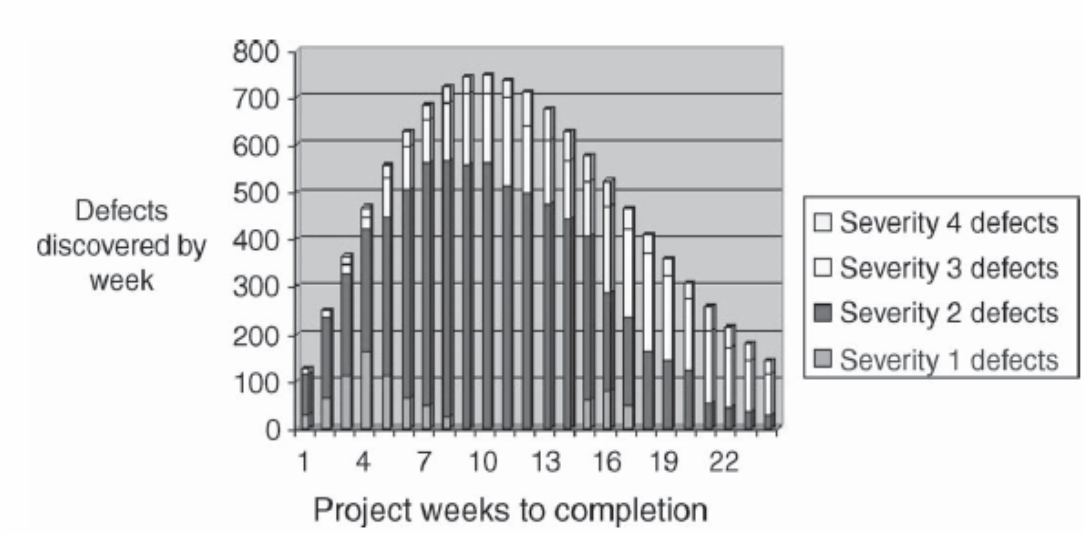
Nếu trong record thông tin về lỗi được phát hiện trong project trước có thêm field miêu tả mức độ tầm trọng của lỗi (1-4), ta có thể xây dựng và sử dụng biểu đồ phân loại các nhóm lỗi như sau :

- Từng chu kỳ (tuần), tính số lượng lỗi thuộc từng mức nặng/nhẹ phát hiện được (ta dùng 4 mức).

- Xếp tầng số lỗi thuộc từng mức nặng/nhẹ trong tuần và vẽ thành 1 cột cho tuần đó.

Dựa vào kết quả của biểu đồ ta rút ra được các kết luận :

- Các lỗi có mức độ càng nặng thường xảy ra và được phát hiện rất sớm vì lúc này project chưa có độ ổn định cao.
- Mức độ lỗi phát hiện giảm dần về sau khi project ngày càng ổn định hơn.



10.7 Độ tương quan về lỗi do khách hàng phát hiện

Trong khoảng thời gian từ lúc project trước đã hoàn thành đến lúc project sau được phát triển, ta có thêm nhiều thông tin khác. Một trong các thông tin quan trọng là số lượng lỗi và tính chất lỗi được phát hiện và phản hồi về bộ phận HelpDesk bởi khách hàng.

Ta dùng độ tương quan về lỗi do khách hàng phát hiện như sau :

- Tỉ lệ lỗi tiên đoán/lỗi được khách hàng phát hiện thực tế
- 487/70 = 6.9

Nhờ độ tương quan này mà ta có thể tiên đoán được số lượng lỗi mà khách hàng phát hiện được khi sử dụng project phần mềm sắp phát triển. Dựa vào số lượng lỗi này mà ta qui hoạch tương đối chính xác số lượng và thời lượng làm việc của các nhân viên trong

2 bộ phận hỗ trợ (Support) và giúp đỡ (HelpDesk) người dùng ⇒ dự toán chính xác chi phí sửa lỗi.

Bảng theo dõi các lỗi được phản ánh bởi khách hàng do bộ phận giúp đở khách hàng ghi nhận và bộ phận phát triển phân loại nhóm lỗi.

HelpDesk tracking log

Root cause analysys of defects

Corrected after prior project completed before next project started

2 Total defect—severity 1	Included in analysis
13 Total defect—severity 2	Included in analysis
55 Total defect—severity 3 & 4	Not included in analysis
-----	
70 Total defects discovered by customers	

Code earmarks	Defects corrected per code earmark	Percent defects corrected per code earmark
AR477 (includes 2 Severity 1s)	7	46.7%
GL431	3	20.0%
DP268	2	13.3%
AP365	1	6.7%
BK663	1	6.7%
PY315	1	6.7%
Total	15	

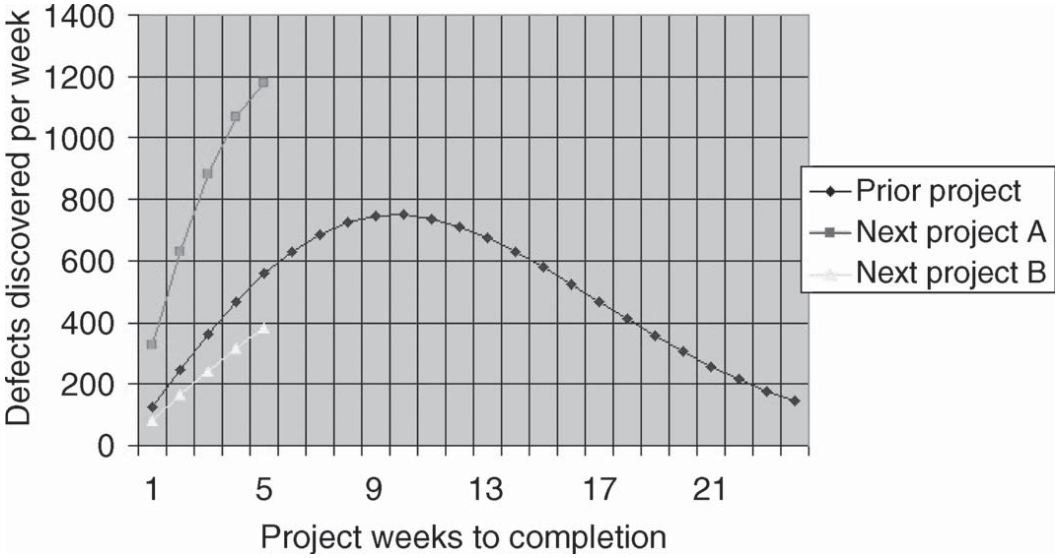
Bắt đầu project mới

Lúc này ta đã có :

- số lượng hàng lệnh và số lỗi của project trước (được thể hiện trong đường cong miêu tả số lỗi phát hiện được).
- mức độ tầm trọng của các lỗi và cách phân phối chúng theo thời gian (được thể hiện trong biểu đồ phân phối mức độ lỗi của project trước).

So sánh với số lượng hàng lệnh của project đang phát triển, ta có thể tính số lượng lỗi cho project mới này, từ đó kế hoạch hóa việc kiểm thử, chuẩn bị tài nguyên phù hợp cho việc kiểm thử.

Trong từng chu kỳ kiểm thử, ta tổng kết và vẽ số lượng lỗi phát hiện trực tiếp lên đường biểu diễn của project trước để dễ dàng so sánh và nội suy thông tin khi cần.



Do các project có qui mô và tính chất khác nhau, số lượng lỗi phát hiện trong tuần đầu sẽ khác nhau.

Thí dụ Project hiện hành là A có số lỗi phát hiện được nhiều hơn project cũ và tốc độ phát hiện lỗi trong các tuần đầu cao hơn nhiều so với project cũ. Đây là xu thế mà ta mong muốn. Điều này có thể đặt ra câu hỏi : tại sao ta kiểm thử hiệu quả hơn trước ? Câu trả lời có thể là do ta dùng nhiều tools kiểm thử tự động hơn trước.

Thí dụ Project hiện hành là B có số lỗi phát hiện được ít hơn project cũ và tốc độ phát hiện lỗi cũng chậm hơn so với project cũ. Đây là xu thế mà ta không mong muốn. Điều này có thể đặt ra câu hỏi : tại sao ta kiểm thử kém hiệu quả hơn trước ? Câu trả lời có thể là do ta bỏ đi hay làm thiếu 1 số hoạt động mà project trước đã làm. Sau khi xác định được nguyên nhân rõ ràng, ta sẽ khắc phục để việc kiểm thử sẽ hiệu quả hơn cho các tuần còn lại.

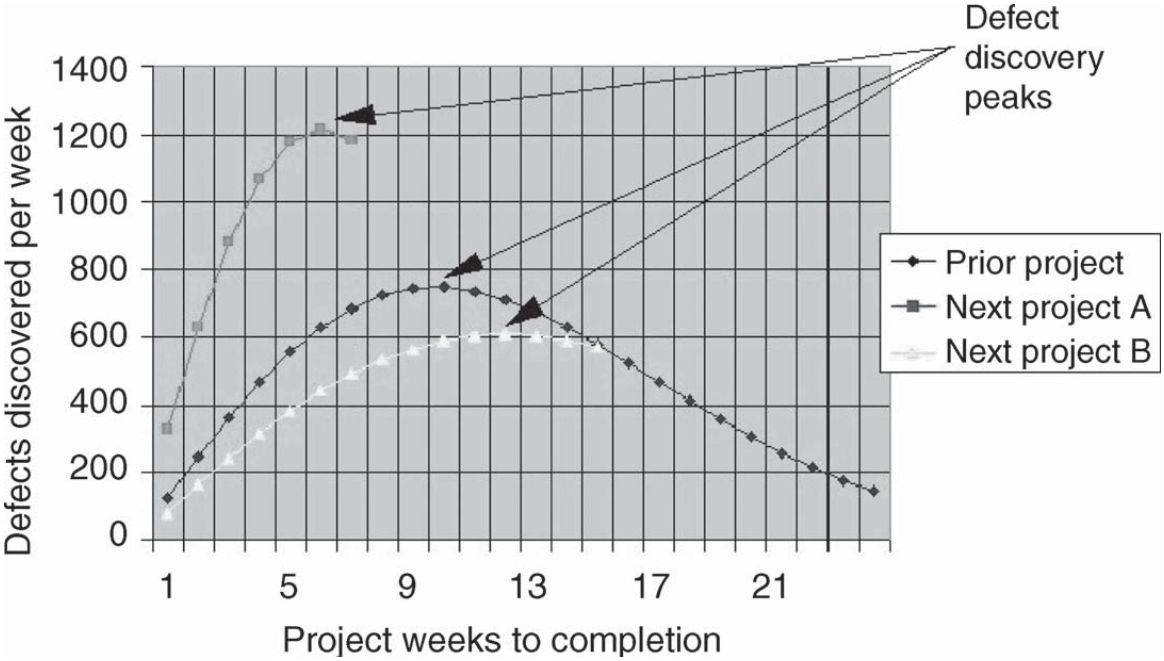
Khi project mới tiếp tục

Tiếp tục kiểm soát và vẽ đường cong biểu diễn lỗi phát hiện cho tới cột mốc kế tiếp trong quá trình phát triển phần mềm. Tới 1 lúc nào đó, thường là khoảng 1/3 thời gian phát triển phần mềm,

tốc độ phát hiện lỗi (số lỗi phát hiện/trên tuần) sẽ đạt ngưỡng và bắt đầu giảm xuống.

Điểm uốn này là điểm neo quan trọng nhất cho hầu hết mọi hoạt động phân tích kết quả về lỗi :

- Project A có điểm ngưởng ở tuần 6 với số lượng 1213 lỗi ⇒ Project A được kiểm thử hiệu quả hơn project cũ.
- Project B có điểm ngưỡng ở tuần 12 với số lượng 607 lỗi ⇒ Project B được kiểm thử kém hiệu quả hơn project cũ và project A.



Khi project mới kết thúc

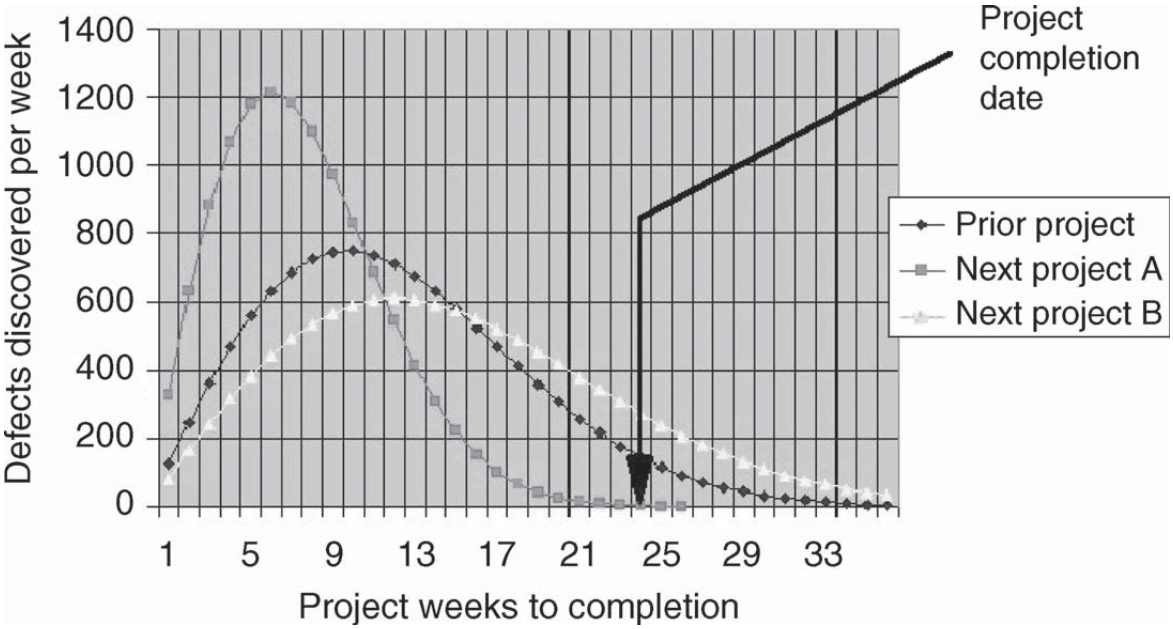
Tiếp tục kiểm thử và vẽ kết quả theo thời gian. Mỗi lần vẽ 1 điểm mới vào đồ thị, ta phân tích điểm này có bất thường gì không ? Nếu có hãy đặt câu hỏi tạo sao và từ đó có biện pháp khắc phục.

Khi project mới kết thúc (giả sử theo kế hoạch là tuần 24 như trong biểu đồ), ta nội suy và nới rộng đồ thị sang phải cho đến khi tiếp xúc trục x. Một số tiên đoán và ước lượng :

- Project A chứa 14 lỗi được tiên đoán là khách hàng sẽ tìm được nên sẽ có 2 lỗi mà khách hàng thực sự tìm ra → chi

phí sửa là 2*14000 = 28000$ (giảm được 952000$ so với project trước).

- Project B chứa 1377 lỗi được tiên đoán là khách hàng sẽ tìm được nên sẽ có 196 lỗi mà khách hàng thực sự tìm ra → chi phí sửa là 196*14000 = 2744000$ (tăng hơn 1746000$ so với project trước).

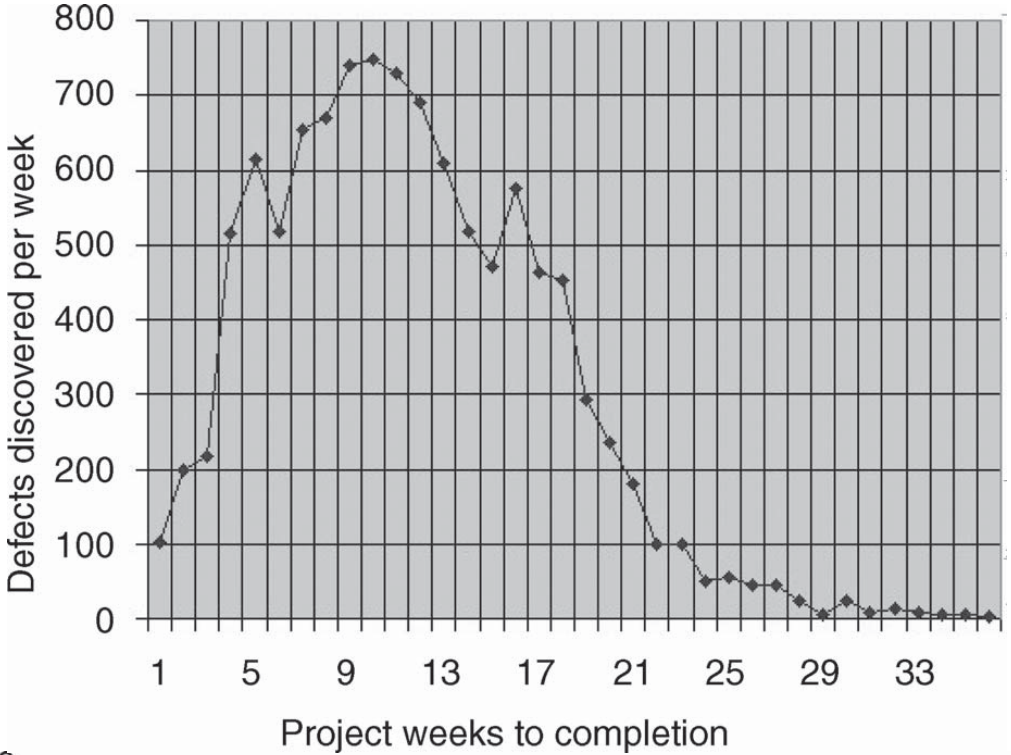


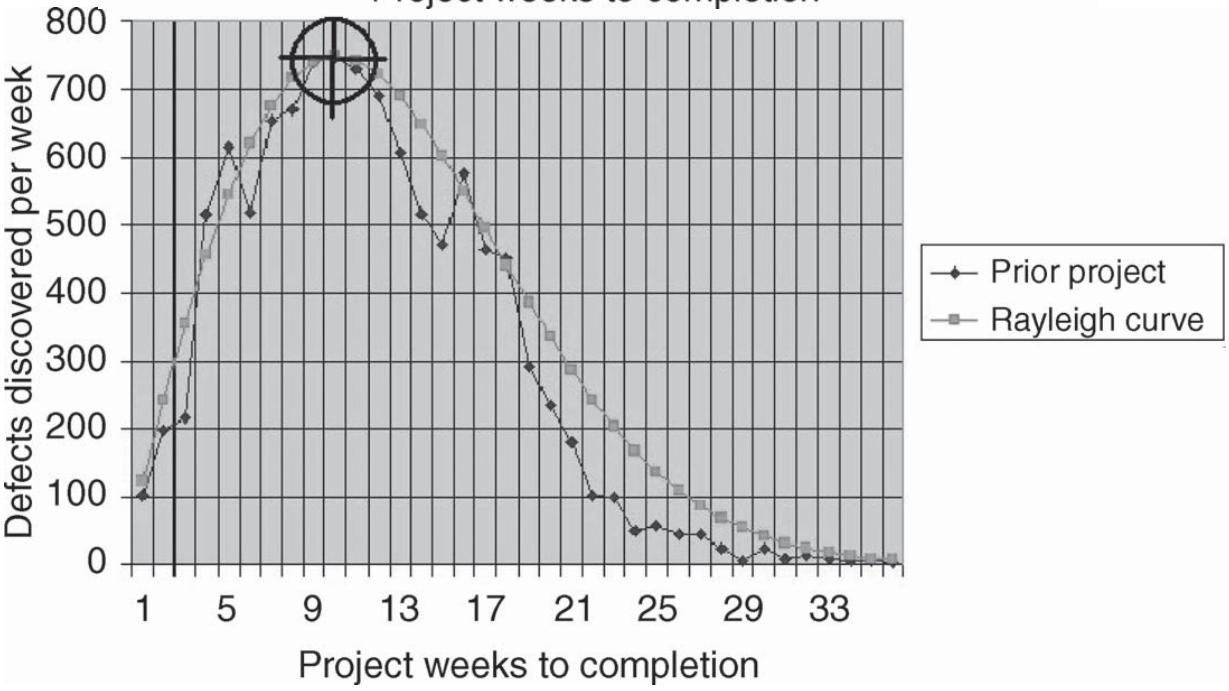
10.8 Đường cong Rayleight - Gunsight cho mẫu phát hiện lỗi

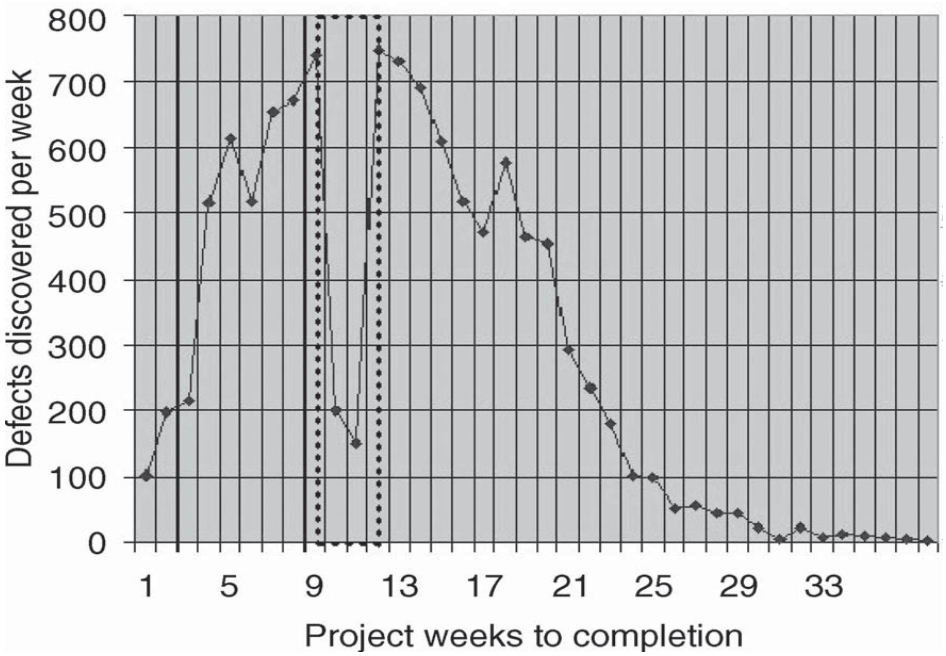
Làm sao biết được tốc độ phát hiện lỗi trong các project công nghiệp của các hãng lớn hiện nay ? Nhờ đó so sánh, đánh giá hoạt động kiểm thử trong project của mình ?

Rất tiếc, ta khó lòng có được các thông tin xác thực này vì các công ty phần mềm có khuynh hướng dấu nhẹm, không công bố (vì nó ảnh hưởng lớn đến hình ảnh công ty).

Hiện nay nguồn cung cấp chủ yếu là ứng dụng đường cong Rayleight. Đây là 1 công thức toán học cho phép ta tái tạo đường cong chuẩn dựa vào điểm ngưỡng (gunsight). Đường cong chuẩn này sai lệch so với giá trị thật của nhiều dự án phần mềm trong 5 năm gần đây dưới 10%.


800
700
600
500
400
300
200
100
0
Defects discovered per week
1
5
9
13
17
21
25
29
33
Project weeks to completion


800
700
600
500
400
300
200
100
0
Defects discovered per week
1
5
9
13
17
21
25
29
33
Project weeks to completion
Prior project
Rayleigh curve



10.9 Các thước đo khác về sự theo dõi lỗi

Các thước đo của hoạt động phát triển phần mềm :

- mã lỗi phát triển duy nhất.
- ngày phát hiện lỗi phát triển.
- mức độ trầm trọng của lỗi phát triển.
- ngày sửa lỗi phát triển.
- mã earcode của lỗi phát triển.

Các thước đo của hoạt động HelpDesk :

- mã lỗi khách hàng duy nhất.
- ngày phát hiện lỗi khách hàng.
- mức độ trầm trọng của lỗi khách hàng.
- ngày sửa lỗi khách hàng.
- mã earcode của lỗi khách hàng.

Thước đo khác : ODC (Orthogonal Defect Classifi cation) của hãng IBM.

10.10 Kết chương

Chương này đã giới thiệu 1 số thuật ngữ được dùng trong hoạt động quản lý quá trình kiểm thử phần mềm.

Chúng ta cũng đã giới thiệu 1 số kỹ thuật phổ dụng để hỗ trợ việc phát hiện lỗi như phát hiện lỗi mới dựa vào lỗi đã phát hiện được, phát hiện lỗi mới dựa vào backlog, phát hiện lỗi mới dựa vào chùm lỗi...

Chúng ta cũng đã giới thiệu 1 số kỹ thuật để quản lý hoạt động kiểm thử phần mềm như dùng đường cong phát hiện lỗi của các project trước, dùng biểu đồ phân loại các nhóm lỗi của các project trước, dùng đường cong Rayleight...