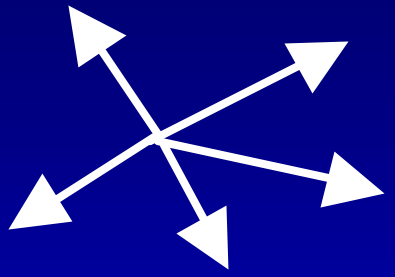
Các kinh nghiệm quí của Công nghệ phần mềm

Mục đích:

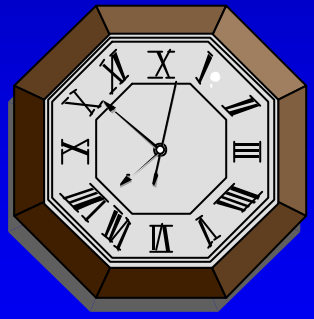
- Khám phá các triệu chứng và các nguyên nhân cốt lõi của các vấn đề trong phát triển phần mềm
- Trình bày Rationals **6 kinh nghiệm tốt** cho quá trình phát triển phần mềm
- Xem xét cách dùng các kinh nghiệm này để giải quyết các vấn đề trong phát triển phần mềm

Phân tích tình hình của CNPM

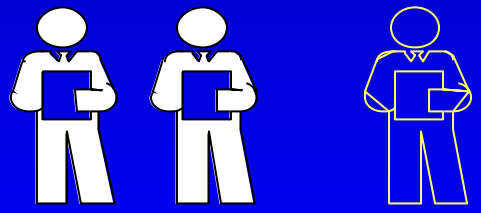
Kinh tế thế giớI ngày càng phụ thuộc hơn vào CNPM



Các ứng dụng mơ rộng về kích thước, độ phức tạp, và phân bố



Thương trường đòi hỏi nâng cao năng suất & chất lượng và giảm thời gian



Không đủ nhân lực có trình độ

Phát triển phần mềm là công việc tập thể

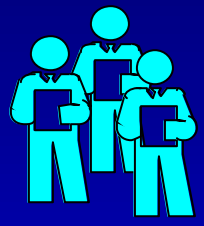
Các thách thức

Các nhóm đông hơn

Sự chuyên môn hóa

Phân tán

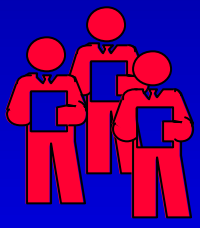
Công nghệ thay đổi quá nhanh



Analyst

Performance Engineer

Project Manager

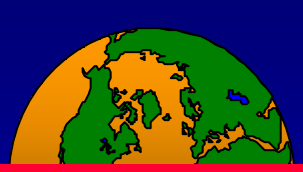


Developer

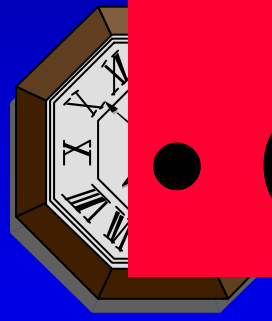
Tester

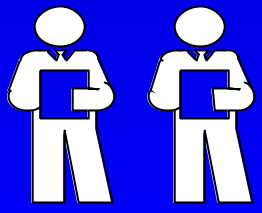
Release Engineer

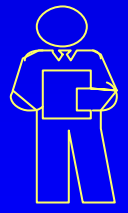
Chúng ta đã làm việc ra sao ?



- Nhiều thành công
- Quá nhiều thất bại







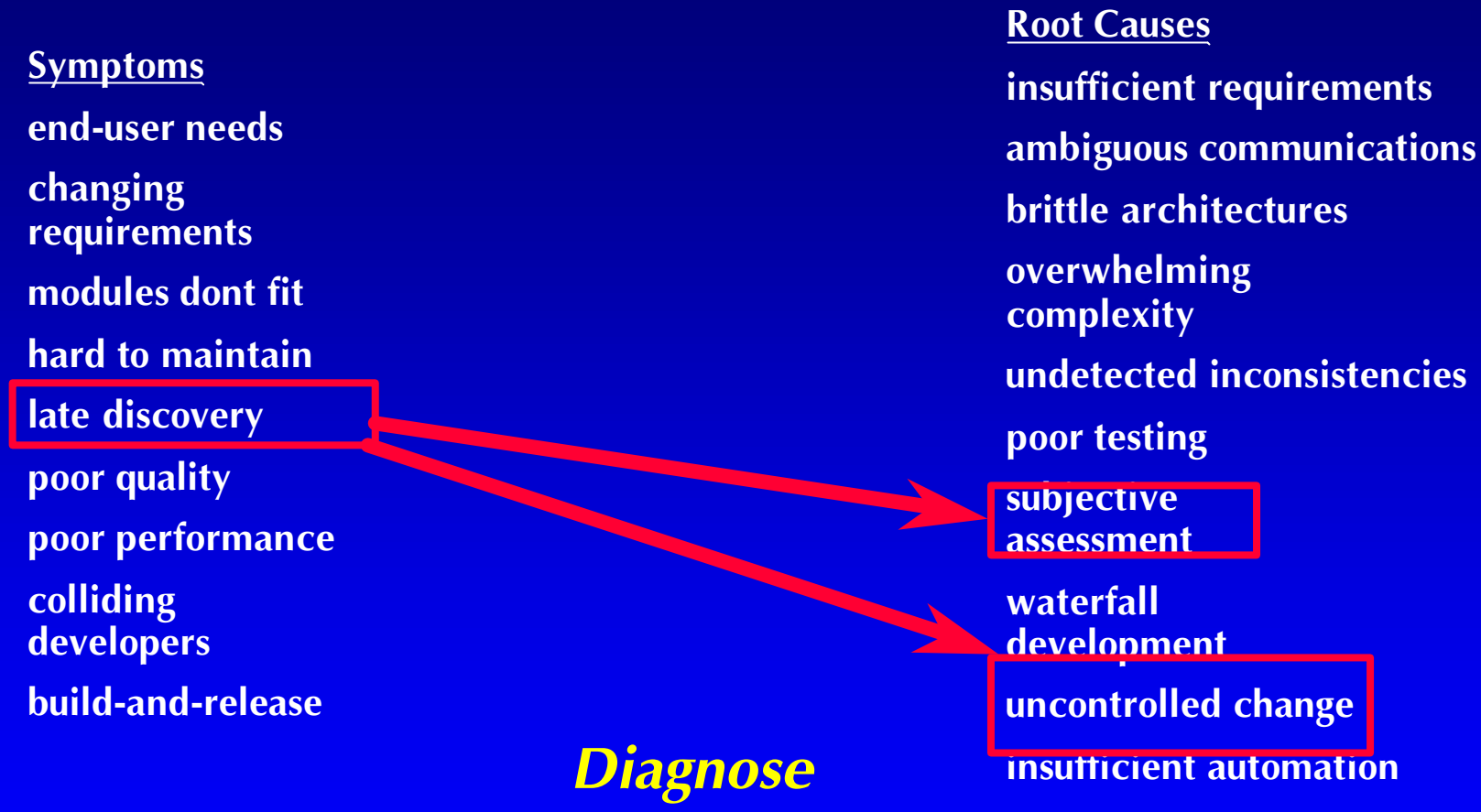
Tester

Release Engineer

Các triệu chứng của các vấn đề trong PTPM

- Hiểu không đúng những gì người dùng cần
- Không thể thích ứng với các thay đổi về y/c đ/v hệ thống
- Các Module không khớp với nhau
- Phần mềm khó bảo trì và nâng cấp, mở rộng
- Phát hiện trễ các lỗ hổng của dự án
- Chất lượng phần mềm kém
- Hiệu năng của phần mềm thấp
- Các thành viên trong nhóm không biết được ai đã thay đổi cái gì, khi nào, ở đâu, tai sao phải thay đổi
- Quá trình build-and-release không đáng tin cậy

Chữa trị triệu chứng không giải quyết vấn đề



Các nguyên nhân chính của các v/đ trong PTPM

- Sự quản lý y/c người dùng không đầy đủ
- Trao đổi thông tin mơ hồ và không đầy đủ
- Kiến trúc không vững chắc
- Độ phức tạp vượt quá tầm kiểm soát
- Có những mâu thuẫn không phát hiện được giữa y/c, thiết kế, và cài đặt
- Kiểm chứng không đầy đủ
- Sự lượng giá chủ quan về tình trạng của dự án
- Sự trễ nải trong việc giảm rủi ro do mô hình thác nước
- Sự lan truyền không thể kiểm soát của các thay đổi
- Thiếu các công cụ tự động hóa

Các kinh nghiệm giúp giải quyết các vấn đề

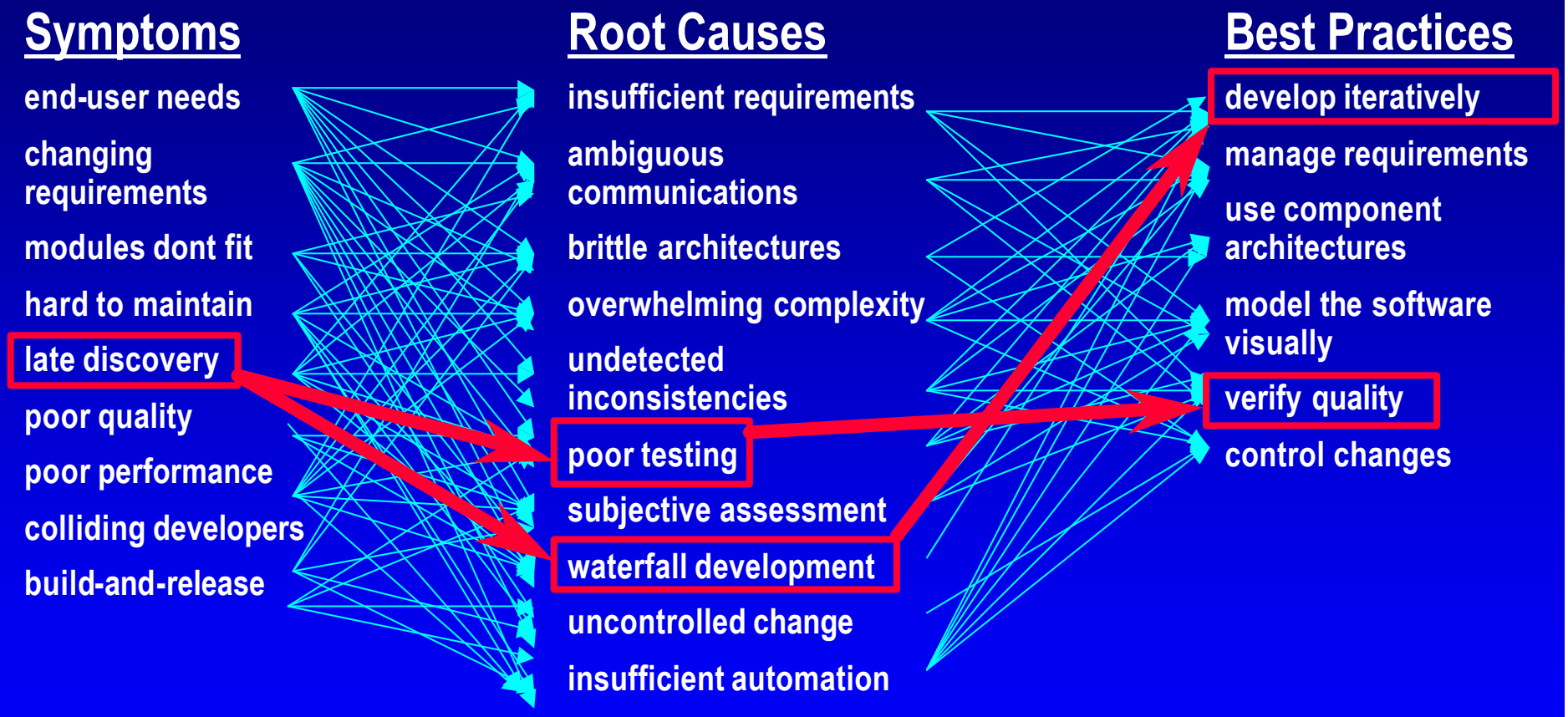
Nguyên nhân cốt lõi

- Các y/c không đầy đủ
- Trao đổi thông tin mơ hồ
- Kiến trúc kém bền vững
- Độ phức tạp quá cao
- Các lượng giá chủ quan
- Các mẫu thuẫn chưa thấy
- Kiểm chứng nghèo nàn
- Q/tr phát triển thác nước
- Sự thay đổi không k/soát
- Thiếu sự tự động hóa

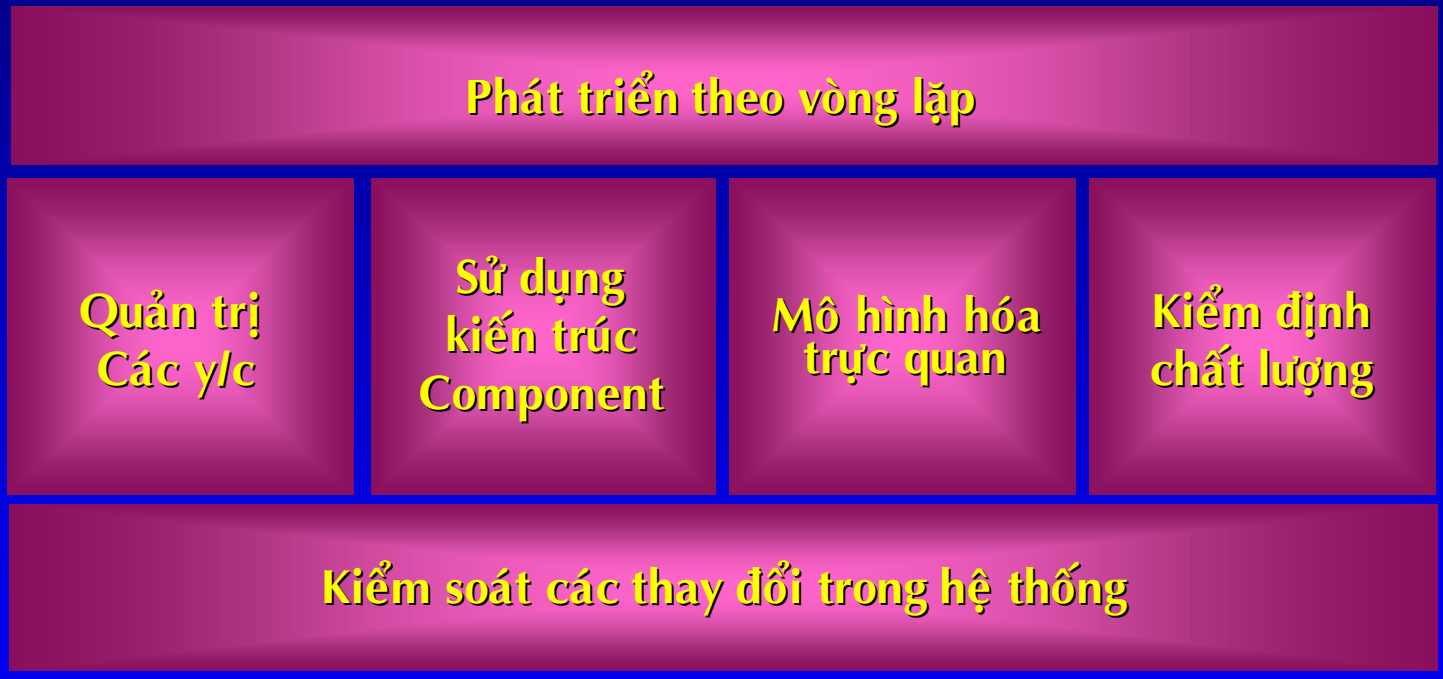
Các kinh nghiệm tốt

- Phát triển theo vòng lặp
- Quản trị các y/c
- Sử dụng KT component
- Mô hình hóa trực quan
- Kiểm định chất lượng
- Kiểm soát các thay đổi

G/q các nguyên nhân giúp giảm các triệu chứng



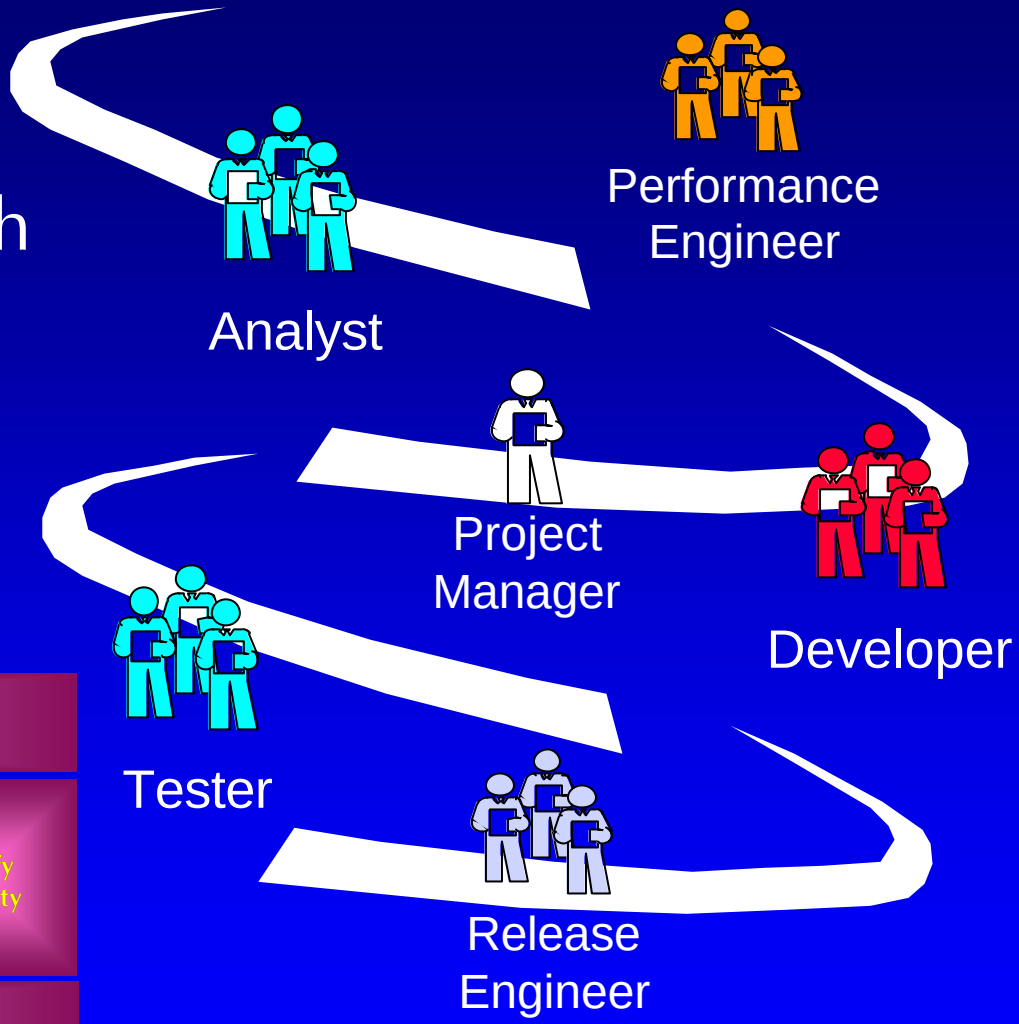
Các kinh nghiệm quí của CNPM

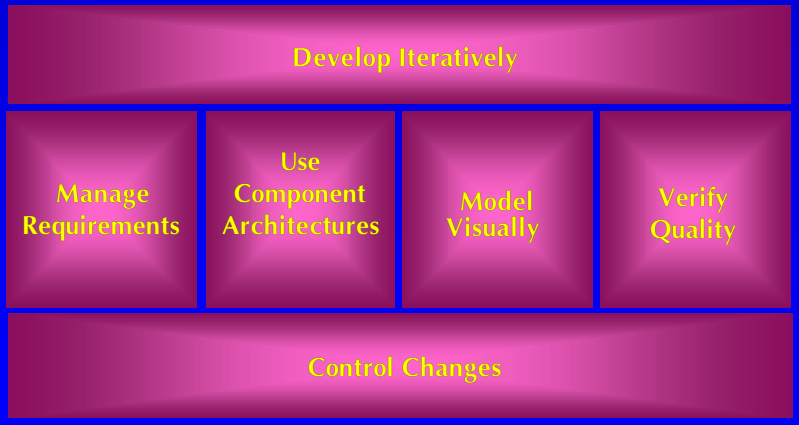


Các kinh nghiệm tạo ra các nhóm lv hiệu năng cao

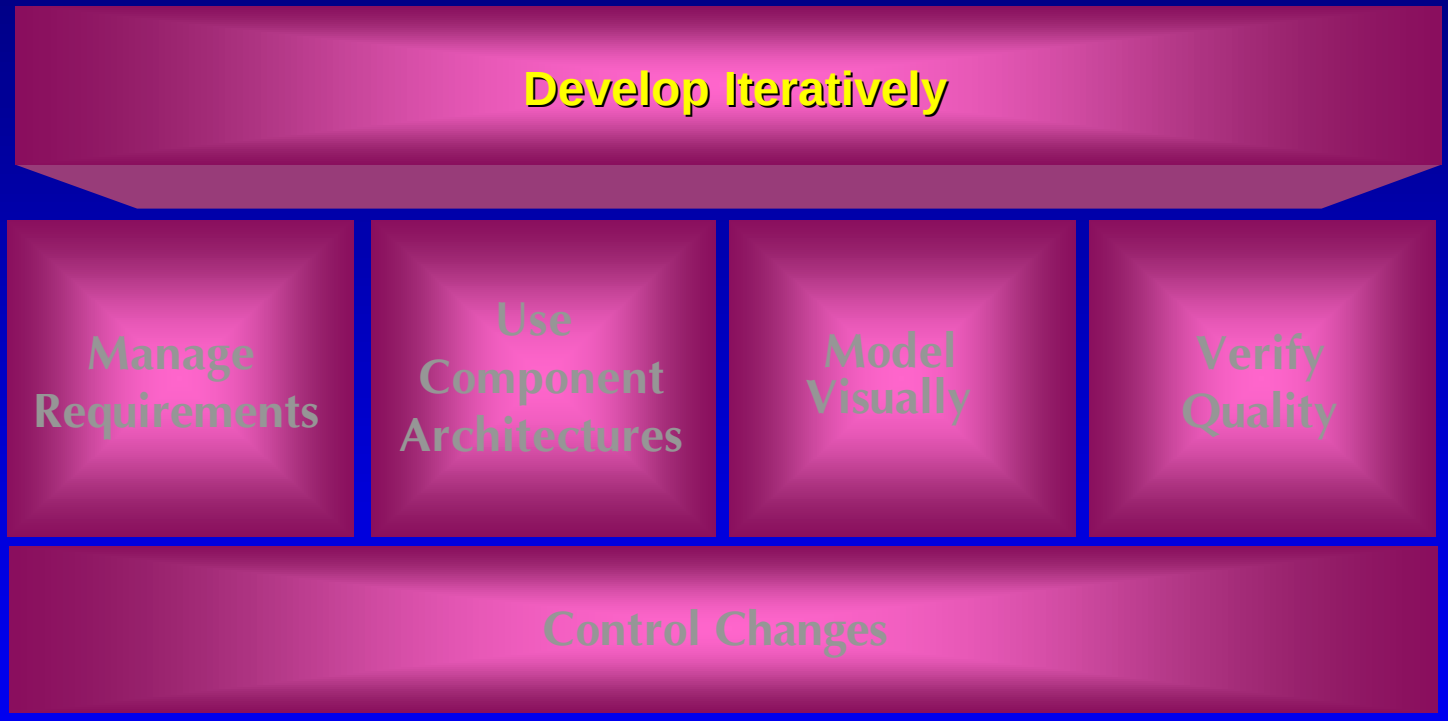
Kết quả

Nhiều dự án thành công hơn



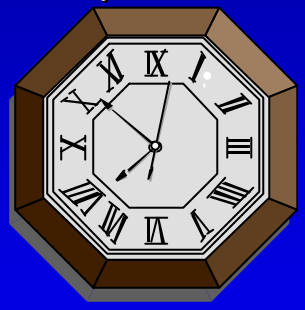


Kinh nghiệm 1: PTPM theo vòng lặp



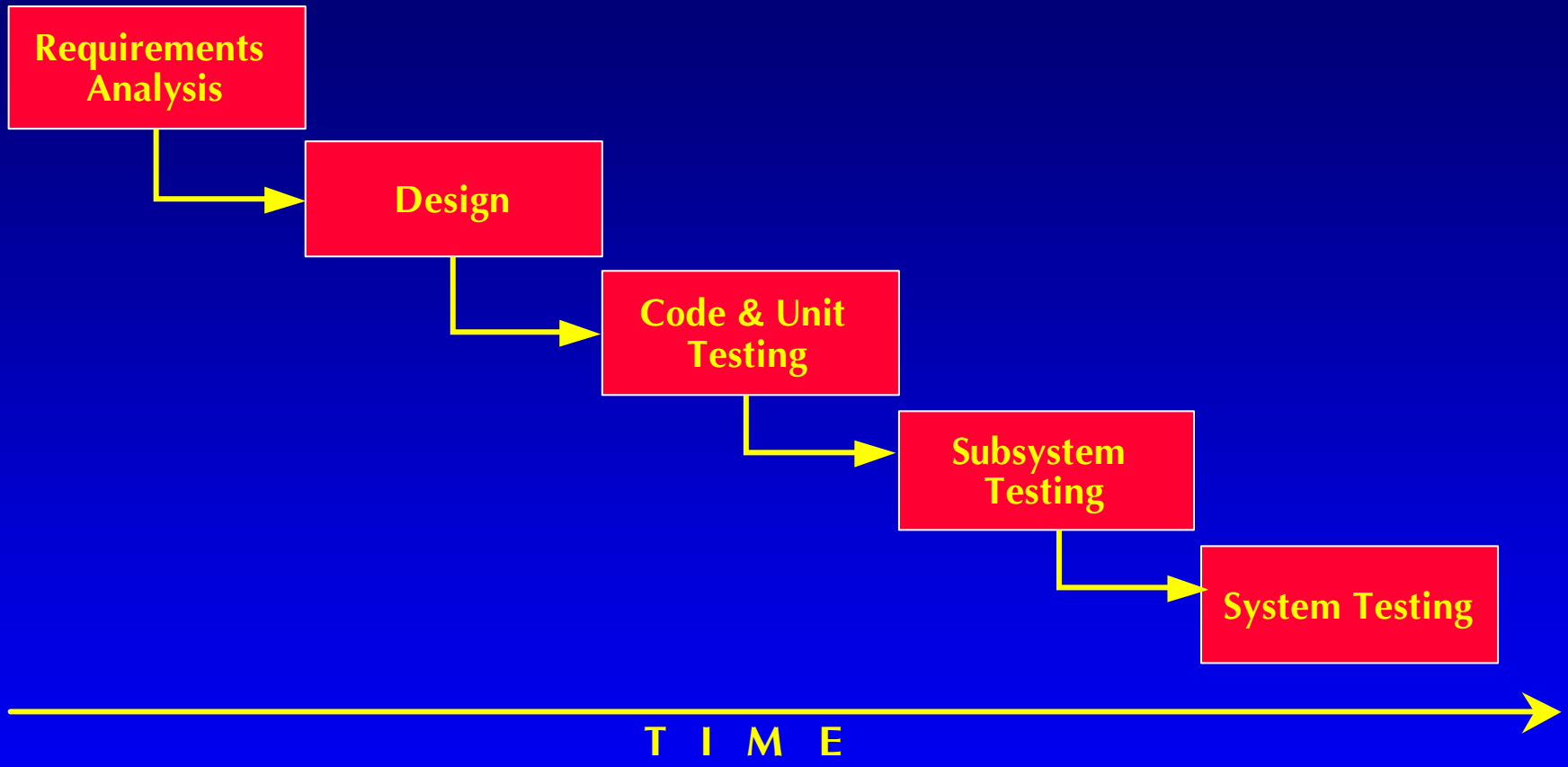
Kinh nghiệm 1: PTPM theo vòng lặp

- Một thiết kế ban đầu có thể không hoàn chỉnh so với các yêu cầu chính
- Việc phát hiện trễ các thiếu sót trong bản thiết kế sẽ làm tăng giá thành, tốn thời gian và thậm chí làm hủy bỏ dự án

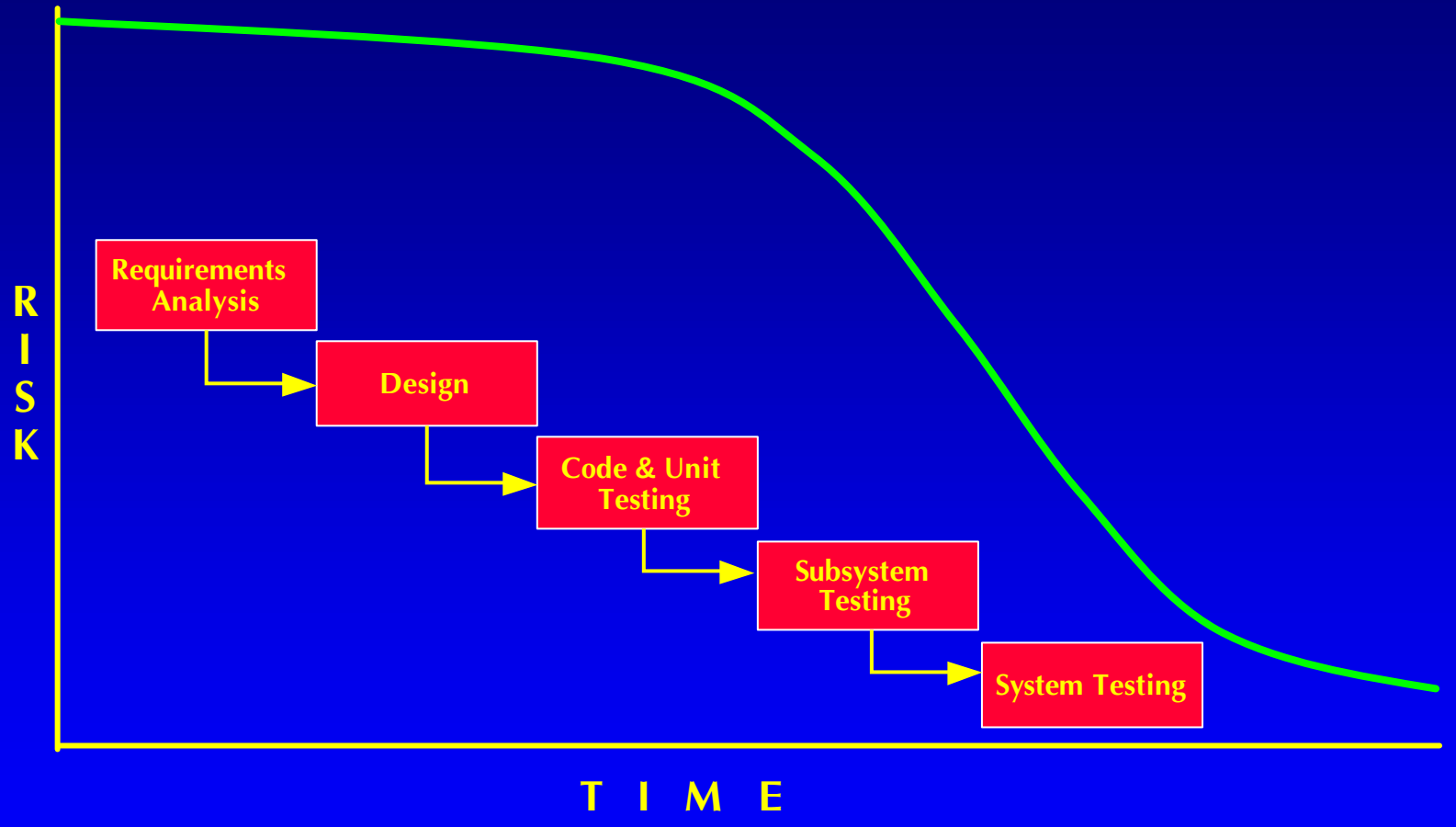


Thời gian và tiền bạc chi ra để cài đặt một thiết kế sai là không thể bù đắp

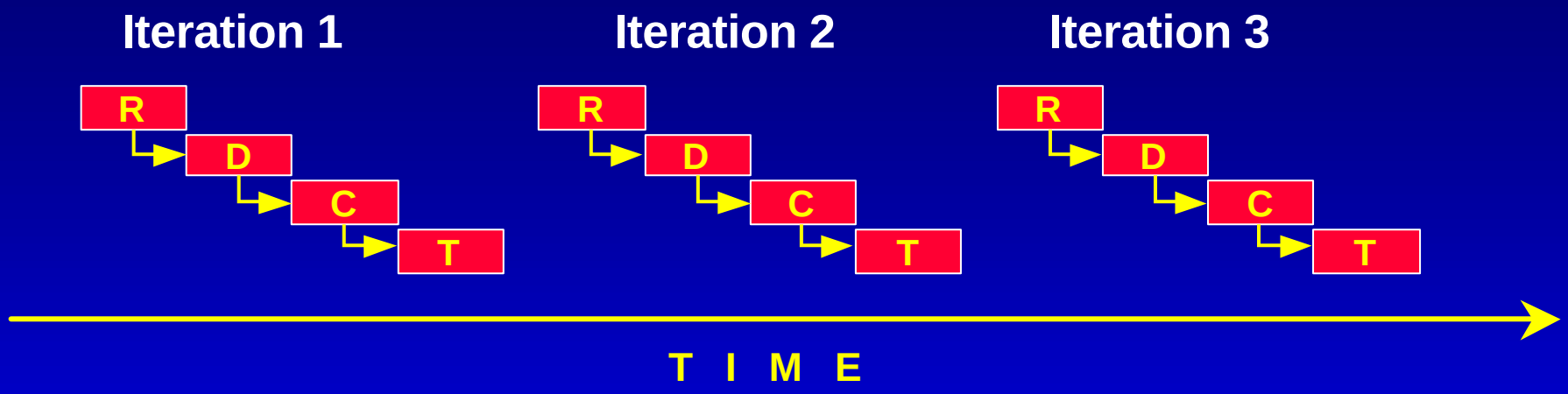
Qui trình thác nước truyền thống



Qui trình thác nước có nhiều rủi ro

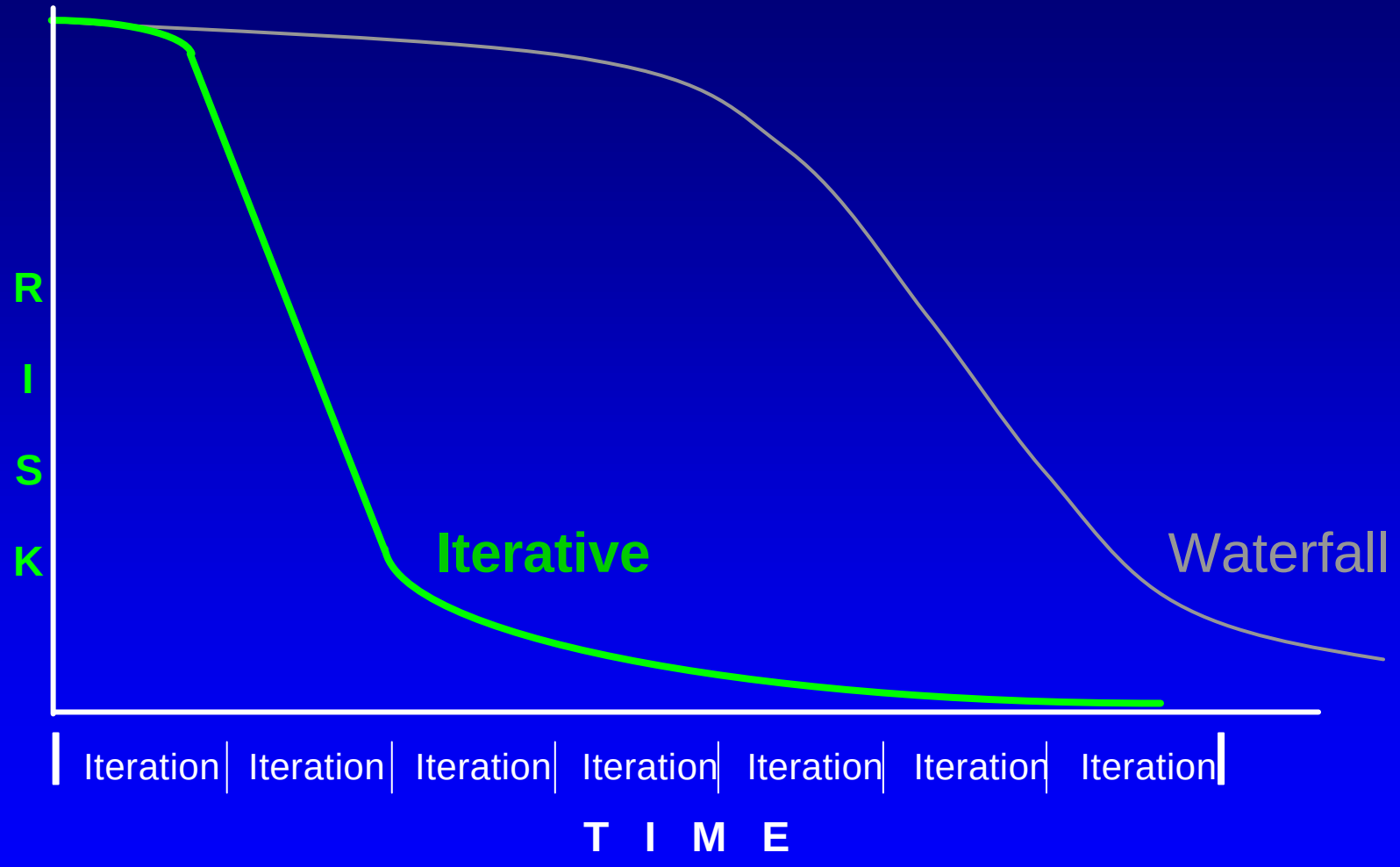


Ứ/d QT thác nước theo vòng lặp



- Các vòng lặp đầu dành cho các v/đ nhiều rủi ro
- Mỗi vòng lặp sinh ra một phiên bản với một sự bổ sung cho hệ thống
- Mỗi VL bao gồm cả việc tích hợp và kiểm chứng

Qui trình lặp đẩy nhanh việc giảm rủi ro



Các đặc tính của qui trình lặp

- Các rủi ro chính được giải quyết trước khi có các phát triển lớn
- Các vòng lặp đầu tiên cho phép nhận feedback
- Việc kiểm chứng và tích hợp diễn ra liên tục
- Các cột mốc cục bộ sẽ định ra các tiêu điểm ngắn hạn
- Sự tiến triển được đo bằng bản cài đặt
- Các cài đặt bộ phận có thể triển khai riêng

Áp dụng các kinh nghiệm trong chu kỳ sống PM

Process Workflows

Business Modeling

Requirements

Analysis & Design

Implementation

Test

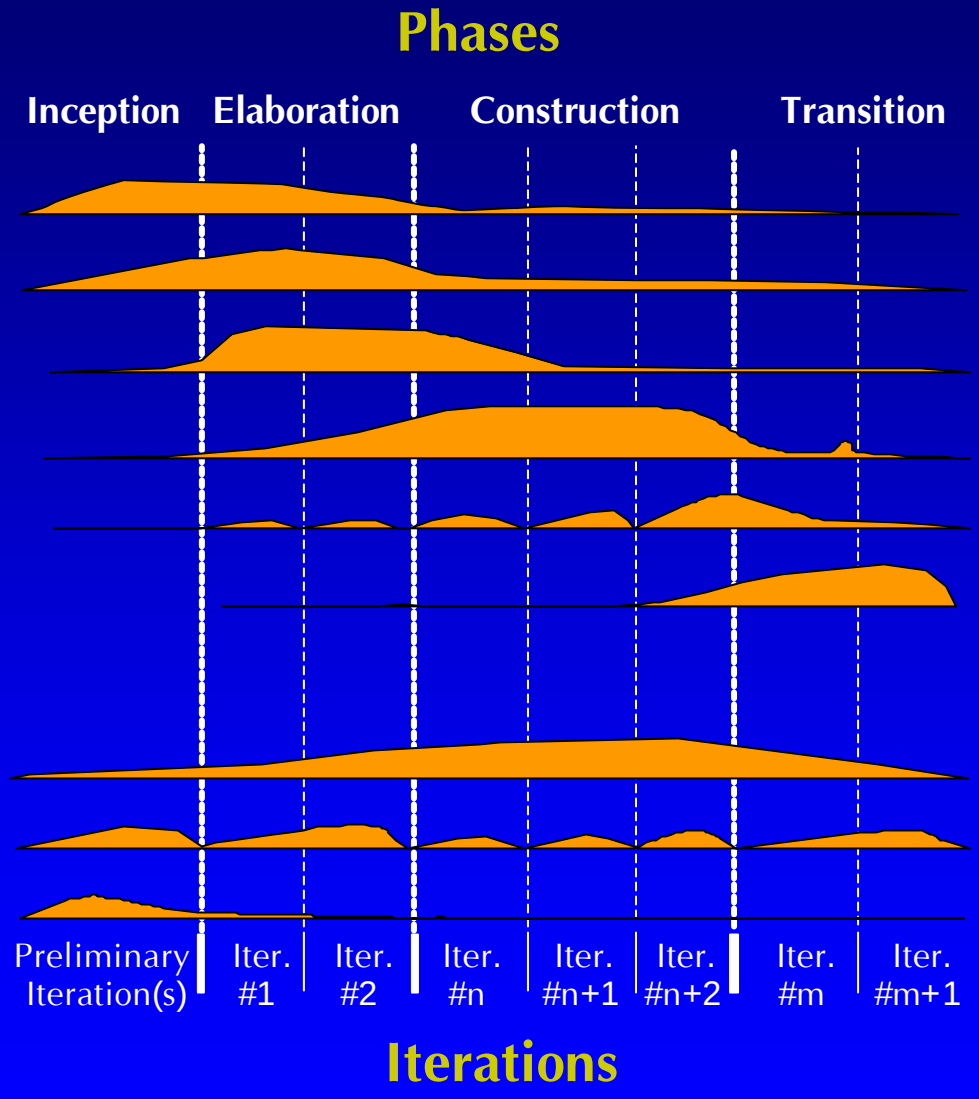
Deployment

Supporting Workflows

Configuration & Change Mgmt

Project Management

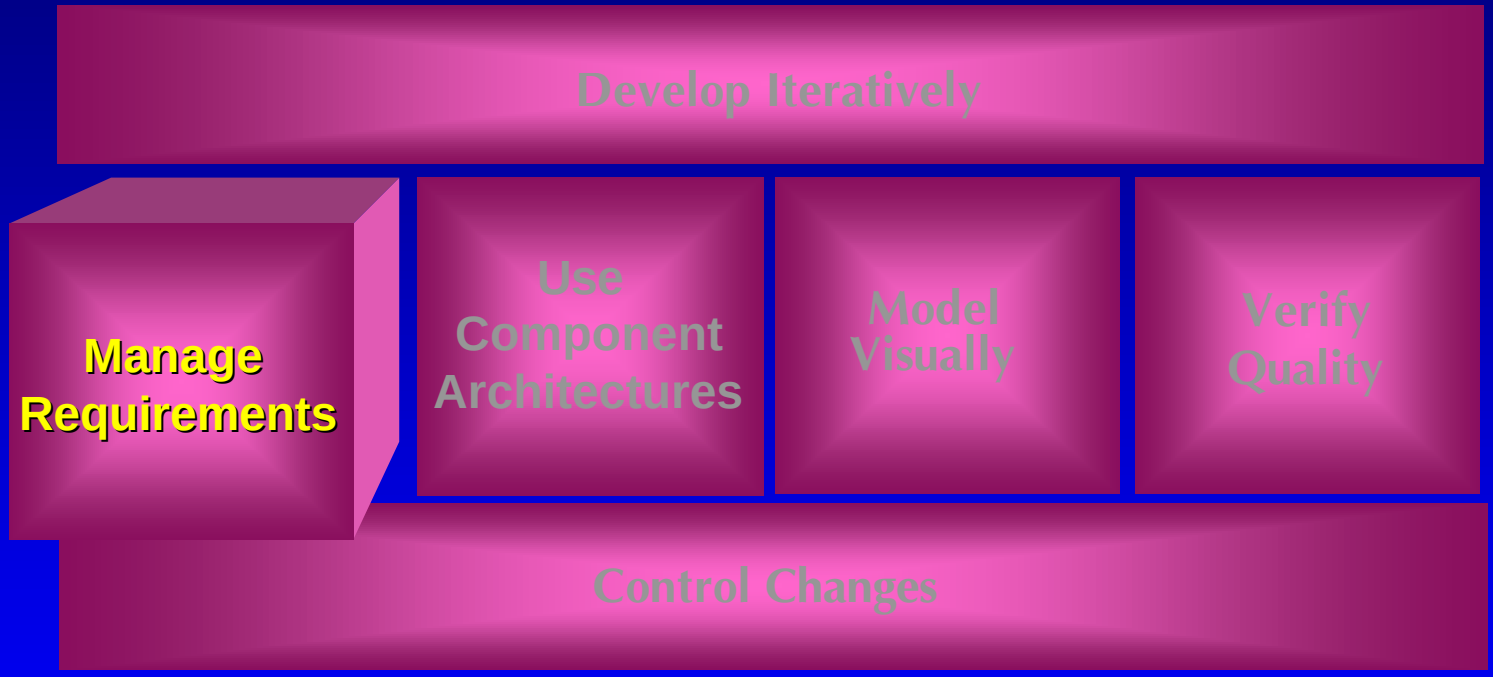
Environment



Qui trình lặp giải quyết các vấn đề

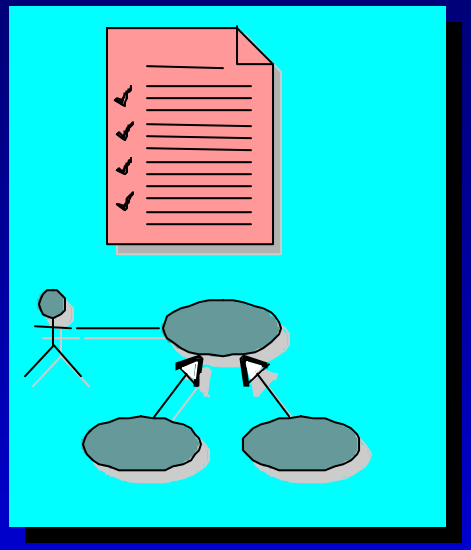
Nguyeân nhaân coát loõi	Caùch giaûi quyeát
Không đủ các yêu cầu đ/v hệ thống	Nhận và khuyến khích các feedback từ người dùng
Trao đổi TT mơ hồ	Các hiểu lầm nghiêm trọng được làm rõ sớm
Kiến trúc kém bền vững	
Độ phức tạp quá cao	Tập trung phát triển các khái niệm chứa nhiều rủi ro trước
Đánh giá chủ quan	Đánh giá khách quan thông qua test
Các mâu thuẫn không được phát hiện	Mâu thuẫn đc phát hiện sớm
Kiểm chứng kém	Bắt đầu test sớm
QT thác nước	
Các thay đổi không ks	Các rủi ro được xác định và giải quyết sớm
Thiếu ccụ tự động	

Kinh nghiệm 2: Quản lý yêu cầu đ/v hệ thống



Kinh nghiệm 2: Quản lý yêu cầu đ/v hệ thống

- Suy dẫn, tổ chức, và tạo sưu liệu về các yêu cầu chức năng và các ràng buộc
- Lượng giá các thay đổi và xác định ảnh hưởng của chúng
- Theo dấu và tao sưu liệu về các thỏa hiệp & các quyết định

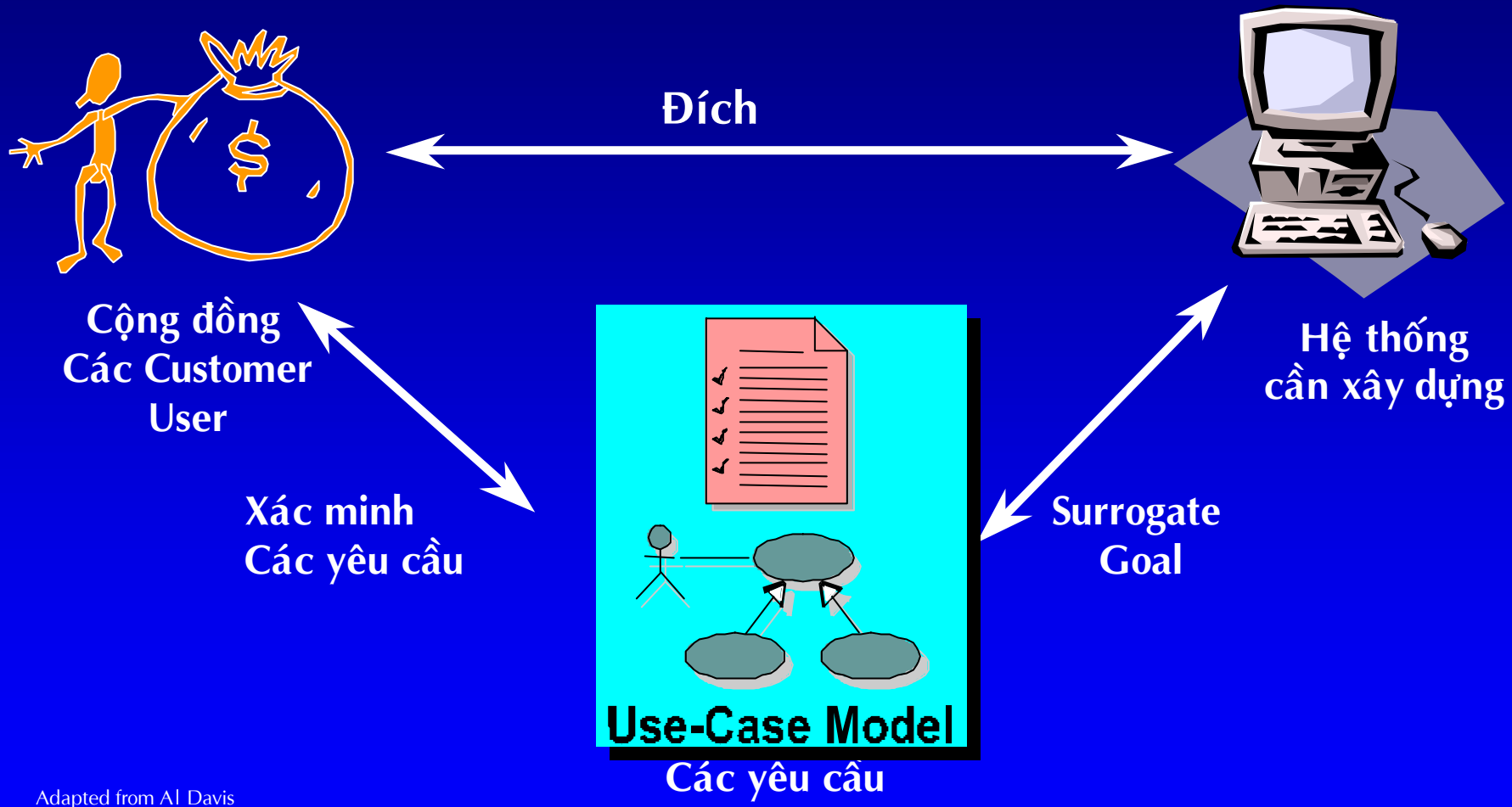


Yêu cầu đối với hệ thống luôn động --
Phải lường trước khả năng chúng bị thay đổi trong quá trình PTPM

Định nghĩa: Y/c đ/v HT và sự quản lý chúng

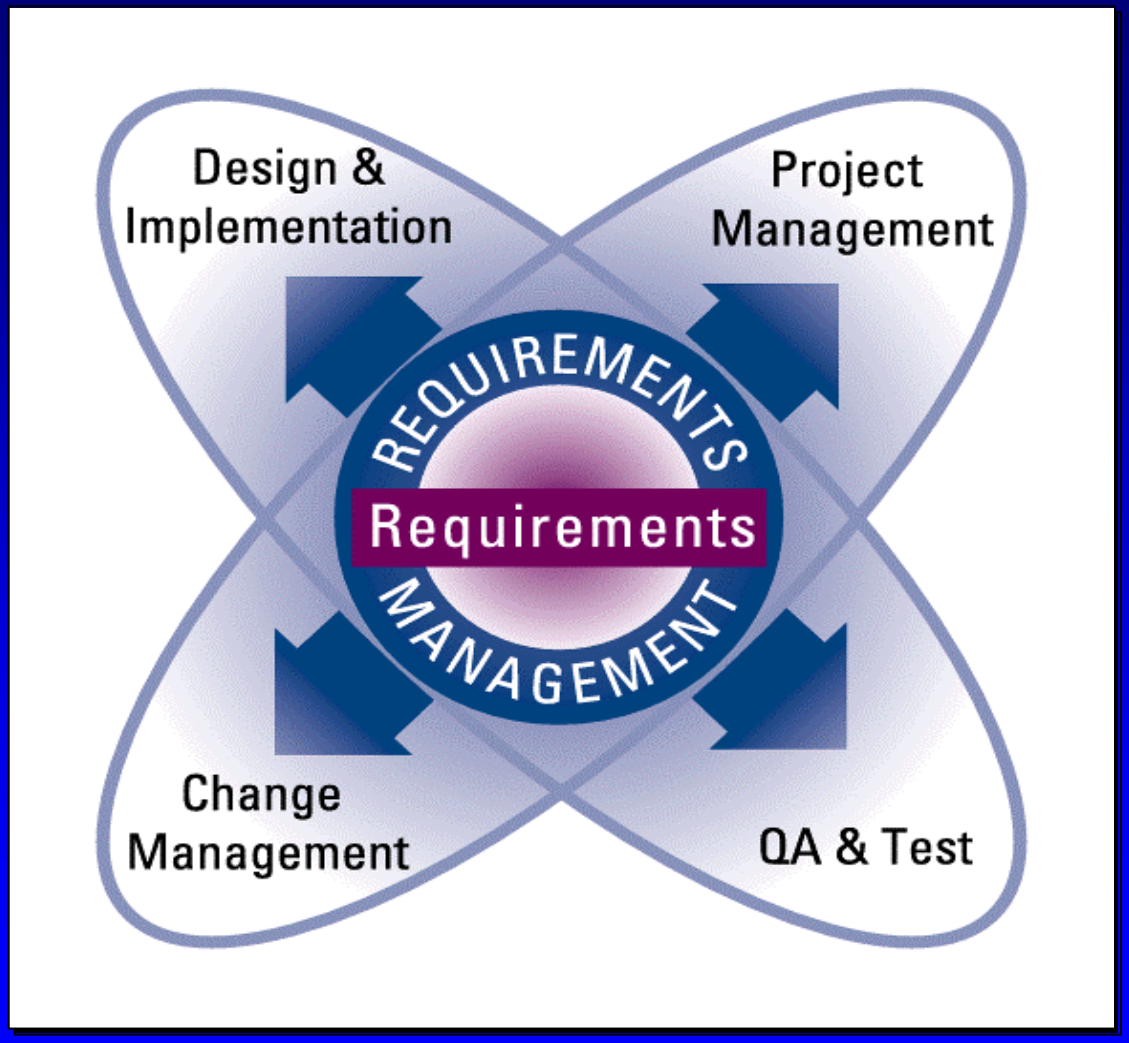
- Một **yêu cầu** là một điều kiện hoặc khả năng mà hệ thống phải tuân theo/có
- **Quản lý y/c** là một tiếp cận có hệ thống để
  - Suy dẫn, tổ chức, và tạo sưu liệu về các yêu cầu chức năng đ/v hệ thống, và
  - Thiết lập và duy trì sự thỏa thuận giữa customer/user và project team liên quan đến các thay đổi về yêu cầu đ/v hệ thống

Thỏa thuận về những gì mà HT phải làm



Adapted from Al Davis

Y/c ảnh hưởng đến nhiều thành phần khác



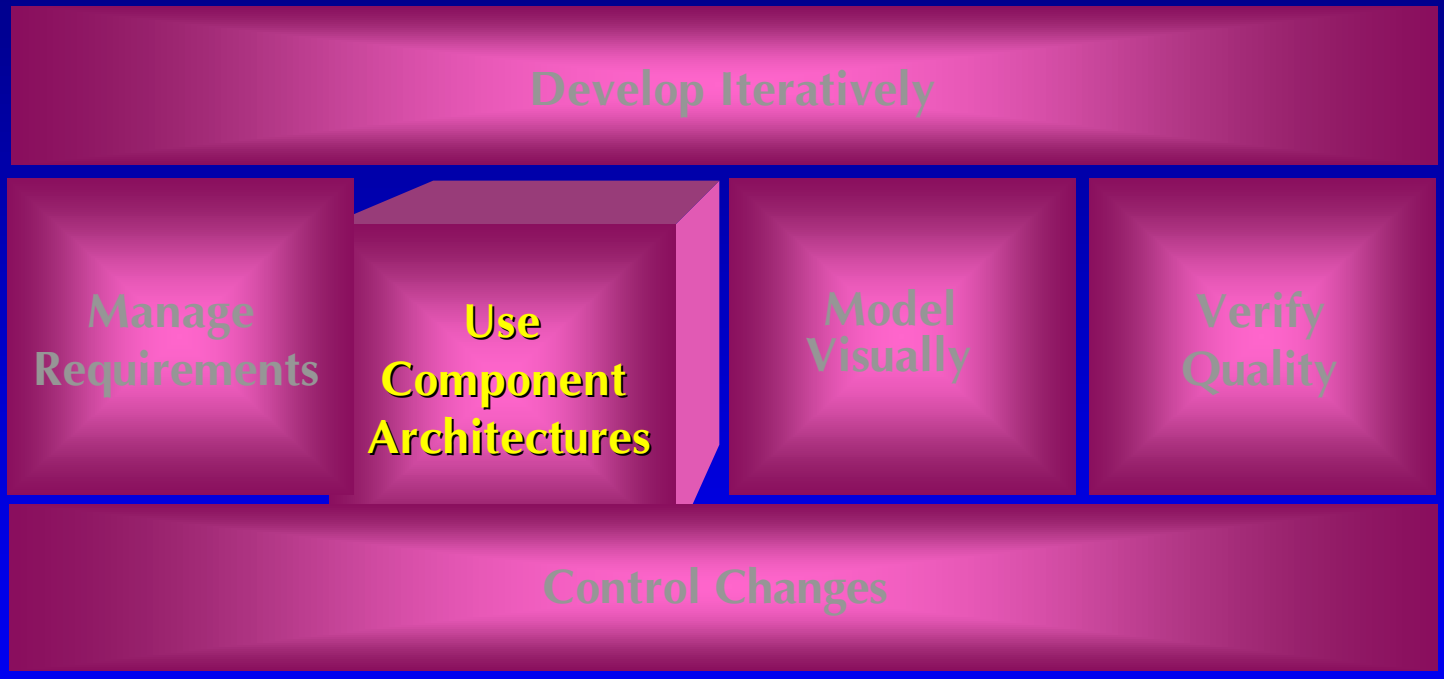
Làm thế nào để bắt được lỗi về y/c sớm ?

- Phân tích vấn đề và suy dẫn ra các nhu cầu của người dùng một cách có hiệu quả
- Đạt được thỏa thuận với customer/user về các yêu cầu đối với hệ thống
- Mô hình hóa sự tương tác giữa user và system
- Thiết lập một đường ranh giới (baseline) và qui trình kiểm soát thay đổi (change control process)
- Duy trì khả năng theo vết tiến và lùi các yêu cầu đ/v hệ thống
- Sử dụng một qui trình lặp

Các vấn đề giải quyết nhờ quản lý y/c đ/v HT

Nguyên nhân cốt lõi	Cách giải quyết
Thiếu các y/c đ/v HT	Xây dựng trong quản lý Y/C một tiếp cận kỷ luật
Trao đổi TT mơ hồ	Trao đổi thông tin dựa trên các y/c đã xác định
Kiến trúc kém bền vững	
Độ phức tạp quá cao	Đặt độ ưu tiên, lọc và theo dõi các yêu cầu
Đánh giá chủ quan	Đánh giá khách quan các chức năng và hiệu năng
Các mâu thuẫn không được phát hiện	Các mâu thuẫn để phát hiện
Kiểm chứng kém	
QT thác nước	
Các thay đổi không ks	RM tool cung cấp một kho chứa các y/c, thuộc tính và đồ hình, sẽ được kết nối tự động với sưu liệu
Thiếu ccụ tự động	

Kinh nghiệm 3: Dùng kiến trúc Component-Based



Kiến trúc phần mềm xác định:

- **Kiến trúc phần mềm** chứa đựng các quyết định quan trọng về tổ chức của hệ thống phần mềm
  - Sự lựa chọn các phần tử cầu trúc và interface của chúng để cấu thành một hệ thống
  - Hành vi được mô tả như sự cộng tác giữa các phần tử này
  - Sự tổng hợp của các phẩn tử cấu trúc và hành vi này thành các subsystem lớn hơn
  - Kiểu kiến trúc định hướng cho tổ chức này, cho các phần tử cấu trúc và interface của chúng, các công tác, và sự tổng hợp giữa chúng

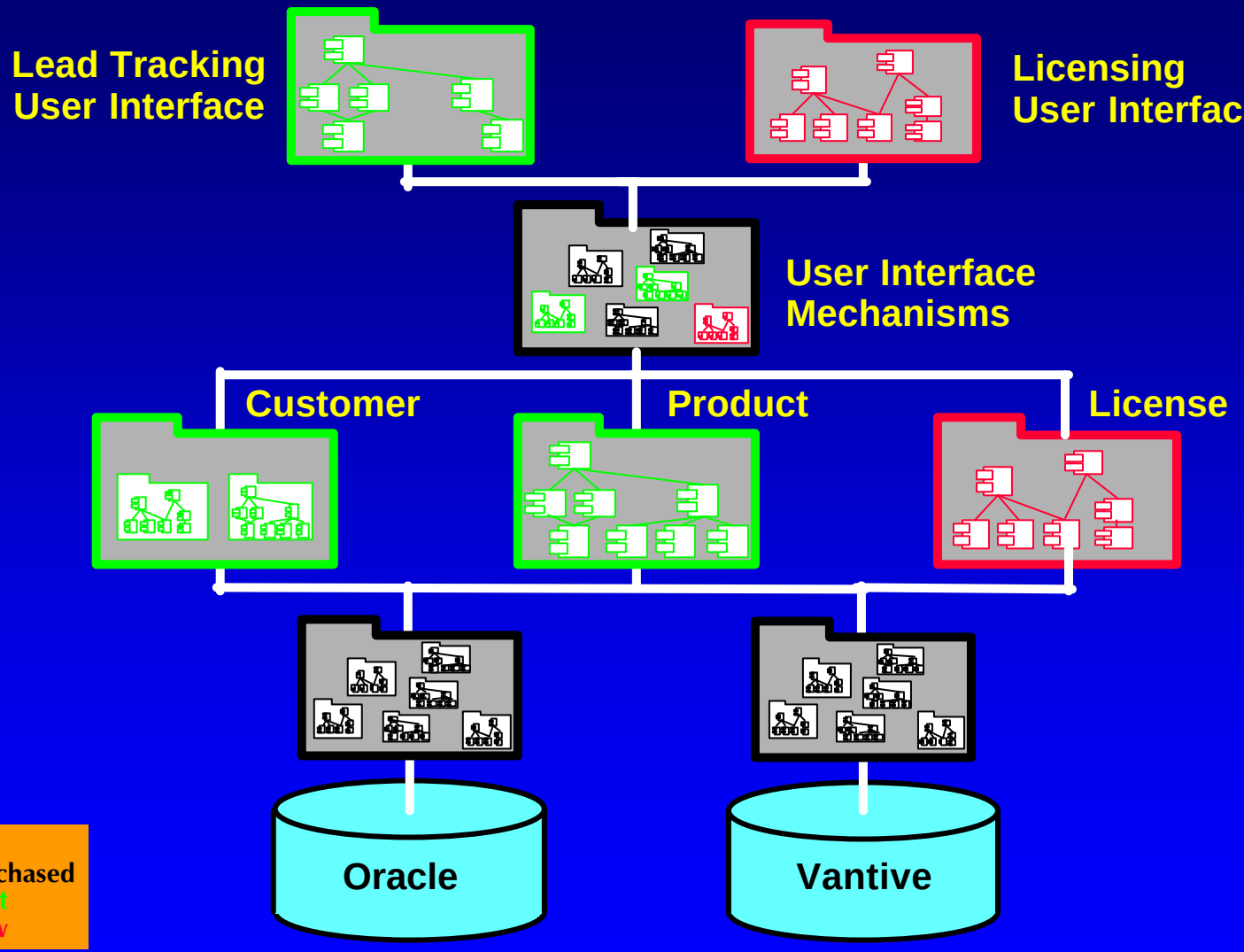
Các ảnh hưởng của kiến trúc

- Kiến trúc phần mềm liên quan đến cấu trúc, hành vi và ngữ cảnh (context):
  - Cách dùng (Usage)
  - Chức năng (Functionality)
  - Hiệu năng (Performance)
  - Tính co dãn (Resilience)
  - Khả năng tái sử dụng (Reuse)
  - Tính dễ hiểu (Comprehensibility)
  - Các ràng buộc về kinh tế và kỹ thuật và các dung hòa
  - Tính thẩm mỹ (Aesthetics)

Resilient, Component-Based Architectures

- Các kiến trúc tốt thỏa mãn các y/c đ/v chúng, là **tính đàn hồi,** và **component-based**
- Một kiến trúc **đàn hồi** cho phép
  - Tăng cường khả năng dễ bảo trì và dễ mở rộng
  - Khả năng tái sử dụng với lợi ích kinh tế cao
  - Phân chia công việc rõ ràng trong đội ngũ PTPM
  - Gói gọn các phụ thuộc phần cứng & hệ thống
- Một kiến trúc **component-based** cho phép
  - Tái sử dụng hoặc tùy chỉnh các component sẵn có
  - Chọn lựa giữa hàng ngàn component thương mại trên thị trường
  - Tiến hóa không ngừng phần mềm đang tồn tại

Ví dụ: Component-Based Architecture



Kiến trúc Component giải quyết các vấn đề

Các nguyên nhân cốt lõi

- Thiếu y/c đ/v hệ thống
- Trao đổi TT mơ hồ
- **Kiến trúc kém bền**
- **Quá phức tạp**
- Đánh giá chủ quan
- Các mâu thuẫn chưa xác định
- Test kém
- Qui trình thác nước
- **Các thay đổi không thể kiểm soát**
- **Thiếu ccụ tự động**

Cách giải quyết

Các Component dễ tạo ra các kiến trúc đàn hồi

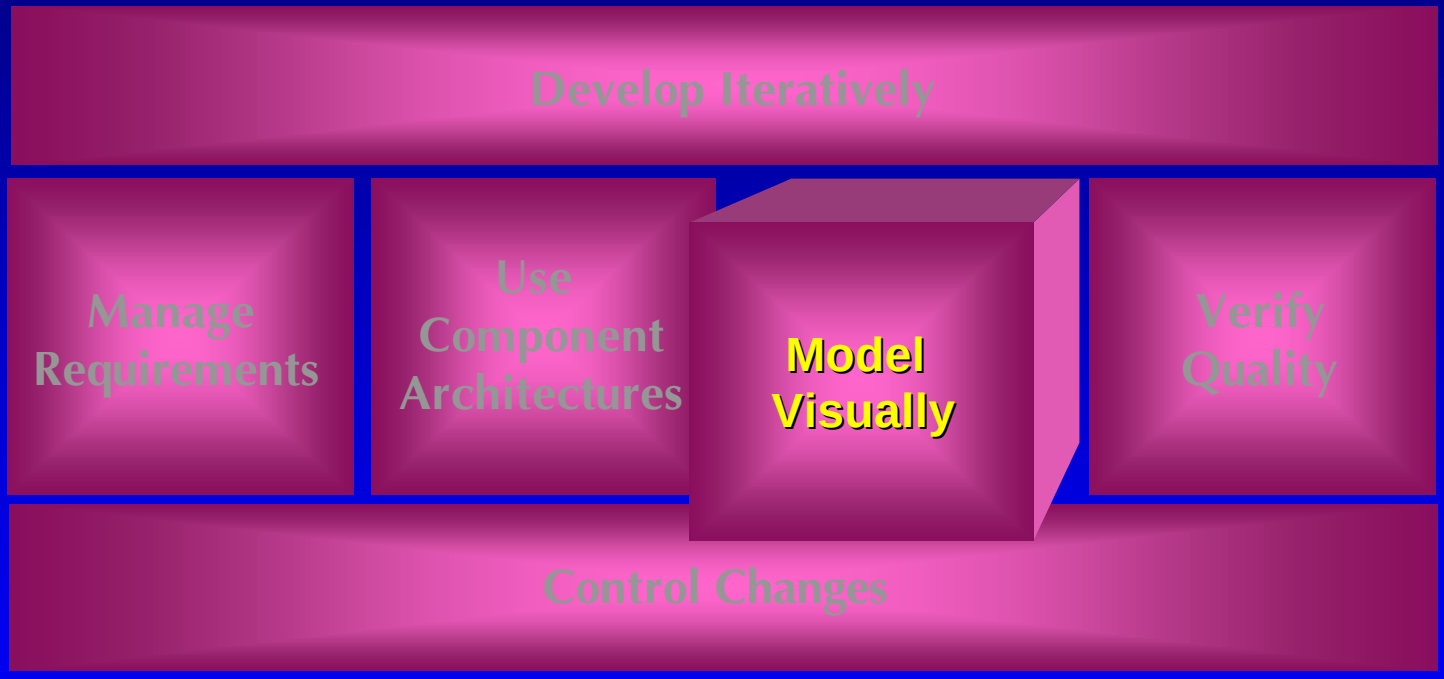
Tái sử dụng các com. và framework Thương mại trở nên dễ dàng

Tính đơn thể cho phép phân tách các điều lo lắng

Component cung cấp nền tảng tự nhiên cho quản lý cấu hình

Các ccụ mô hình hóa trực quan hỗ trợ thiết kế tự động component-based

Kinh nghiệm 4: Mô hình hóa trực quan phần mềm



Kinh nghiệm 4: Mô hình hóa trực quan phần mềm

- Nắm bắt cấu trúc và hành vi của các thành phần kiến trúc
- Thể hiện cách mà các phần tử hệ thống khớp với nhau
- Che dấu hoặc phơi bày chi tiết theo nhu cầu công việc
- Duy trì tinhd nhất quán giữa thiết kế và cài đặt
- Tăng cường trao đổi thông tin rõ ràng

Moâ hình hoùa tröïc quan taêng khaû naêng quaûn lyù ñoä phöùc taïp cuûa phaàn meàm

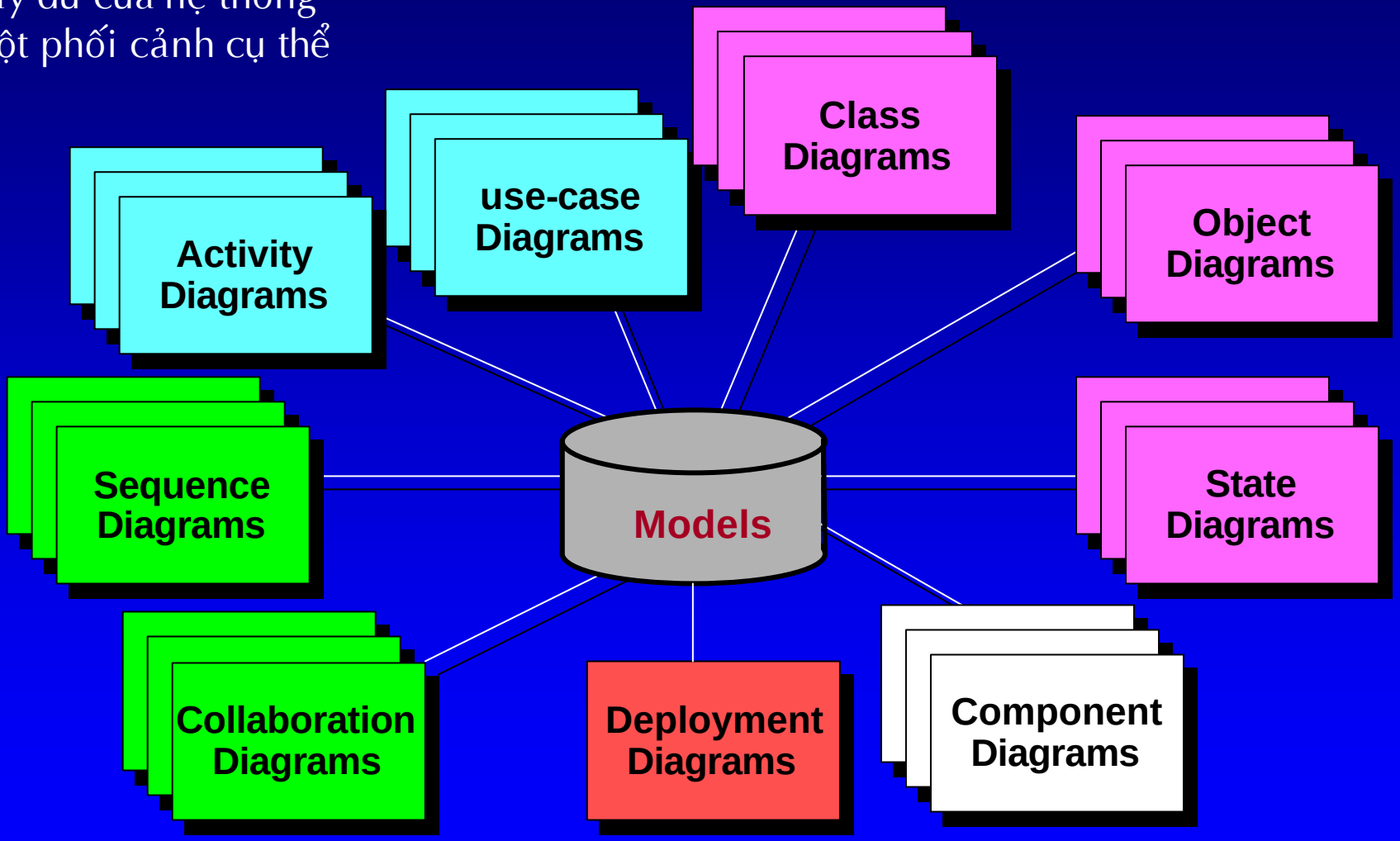
UML là gì ?

- Unified Modeling Language (UML) là ngôn ngữ
  - đặc tả
  - trực quan hóa
  - xây dựng
  - làm sưu liệu

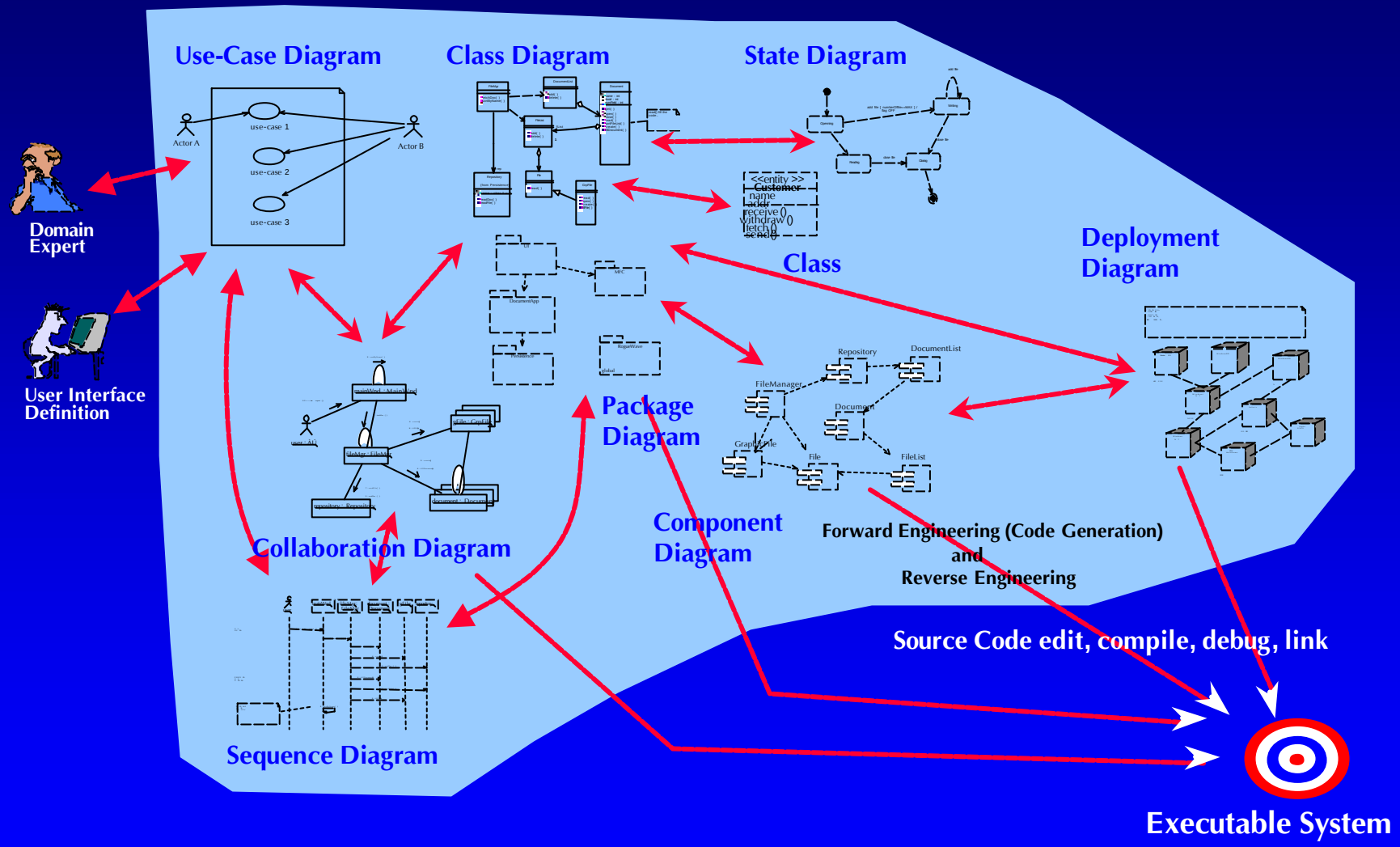
  các artifact của một hệ thống phần mềm

Các lược đồ là các khung nhìn của mô hình

Một *mô hình* là một mô tả đầy đủ của hệ thống từ một phối cảnh cụ thể



Mô hình hóa trực quan dừng các lược đồ UML



Mô hình hóa trực quan và phát triển theo vòng lặp

Thay ñoåi baûn thieát keá ?

Mô hình hóa trực quan và phát triển theo vòng lặp

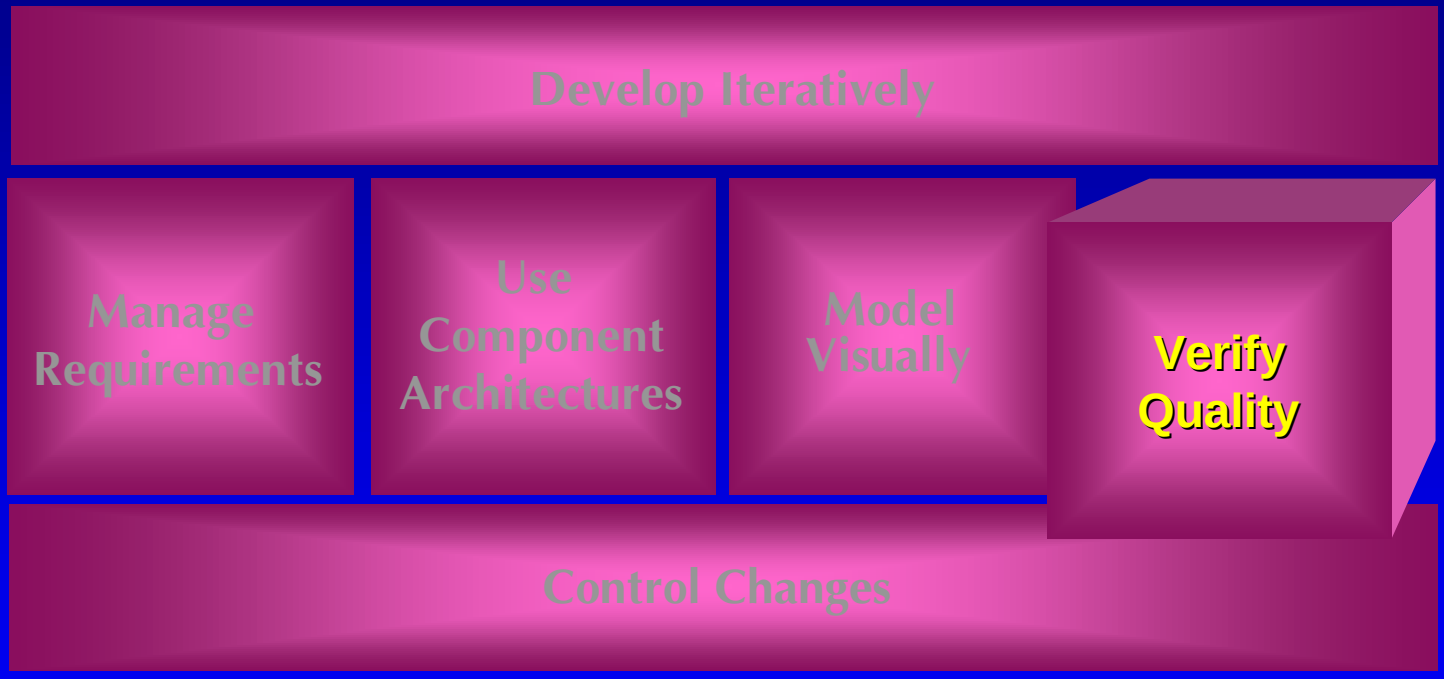
Caùi gì thay ñoåi? Nhöõng thay ñoåi naøy ñöôïc pheùp khoâng?

Giải quyết vấn đề nhờ mô hình hóa trực quan

Caùc nguyeân nhaân coát loõLôøi giaûi

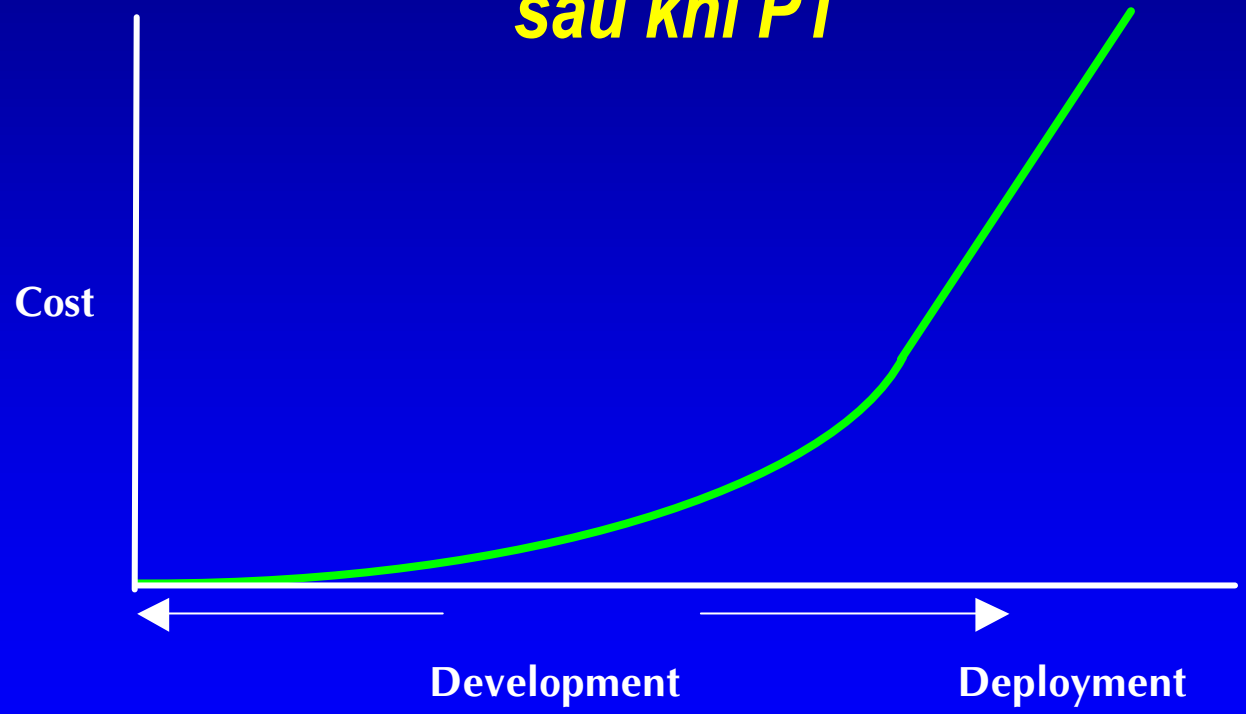
- Thiếu y/c đ/v HT → Các use-case và scenario đặc tả hành vi rõ ràng
- Truyền tin mơ hồ → Các mô hình nắm bắt tường minh các thiết kế
- Kiến trúc kém bền → Các kiến trúc không đơn thể hay cứng nhắc bị phơi bày
- Quá phức tạp → Các chi tiết không cần thiết được che dấu khi cần
- Đánh giá chủ quan
- Các mâu thuẫn chưa xác định → Các thiết kế tường minh chỉ ra các mâu thuẫn dễ dàng
- Test kém → Chất lượng của ứng dụng đi kèm với bản thiết kế tốt
- Qui trình thác nước
- Thay đổi không thể KS
- Thiếu ccụ tự động → Các ccụ trực quan hỗ trợ cho mô hình hóa bằng UML

Kinh nghiệm 5: Kiểm định chất lượng phần mềm

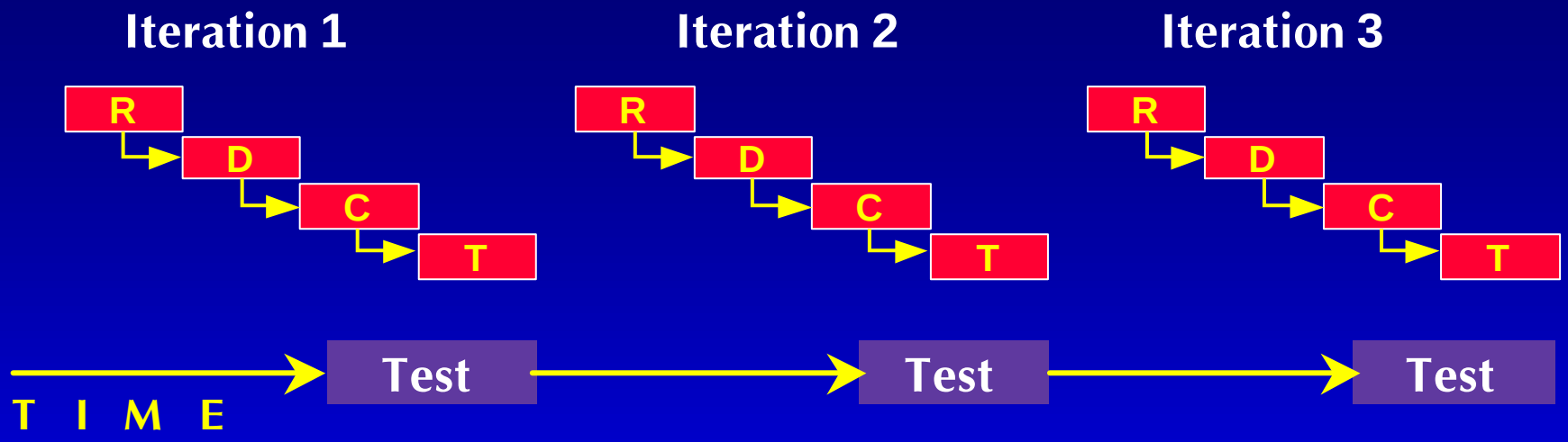


Kinh nghiệm 5: Kiểm định chất lượng phần mềm

Chi phí tìm kieám vaø söûa chöõa caùc vaán ñeà cuûa phaàn meàm seõ taêng haøng 100, haøng 1000 laàn sau khi PT



PT theo vòng lặp cho phép test liên tục



Test Life Cycle

Plan
Design
Implement
Execute
Evaluate

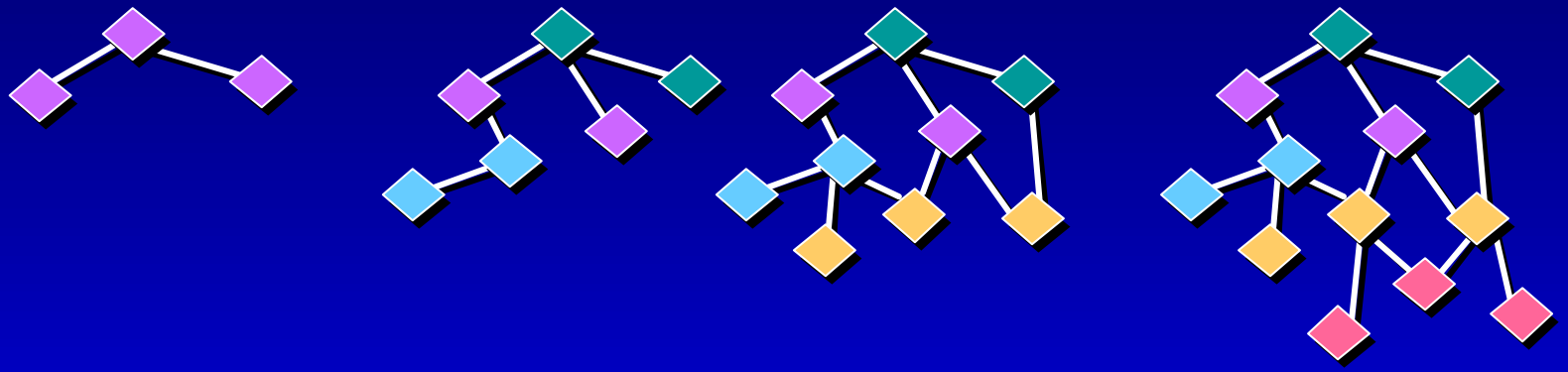
Plan
Design
Implement
Execute
Evaluate

Plan
Design
Implement
Execute
Evaluate

Test trong một môi trường PT theo vòng lặp

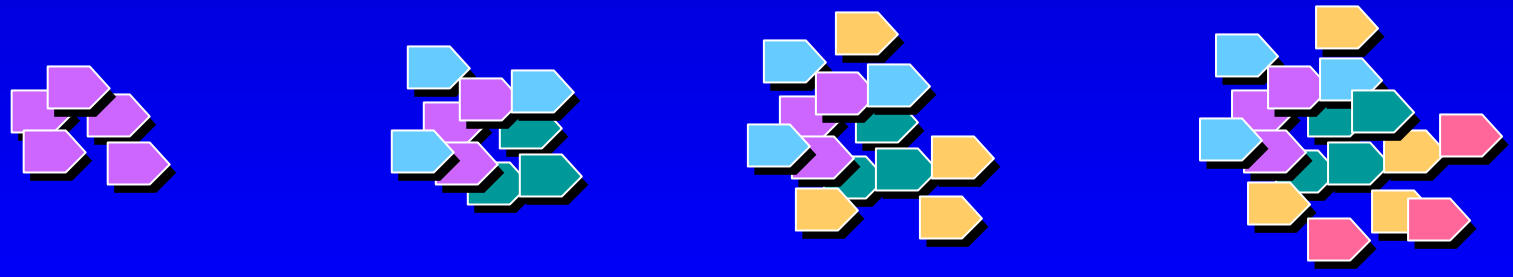
Requirements

Iteration 1 Iteration 2 Iteration 3 Iteration 4

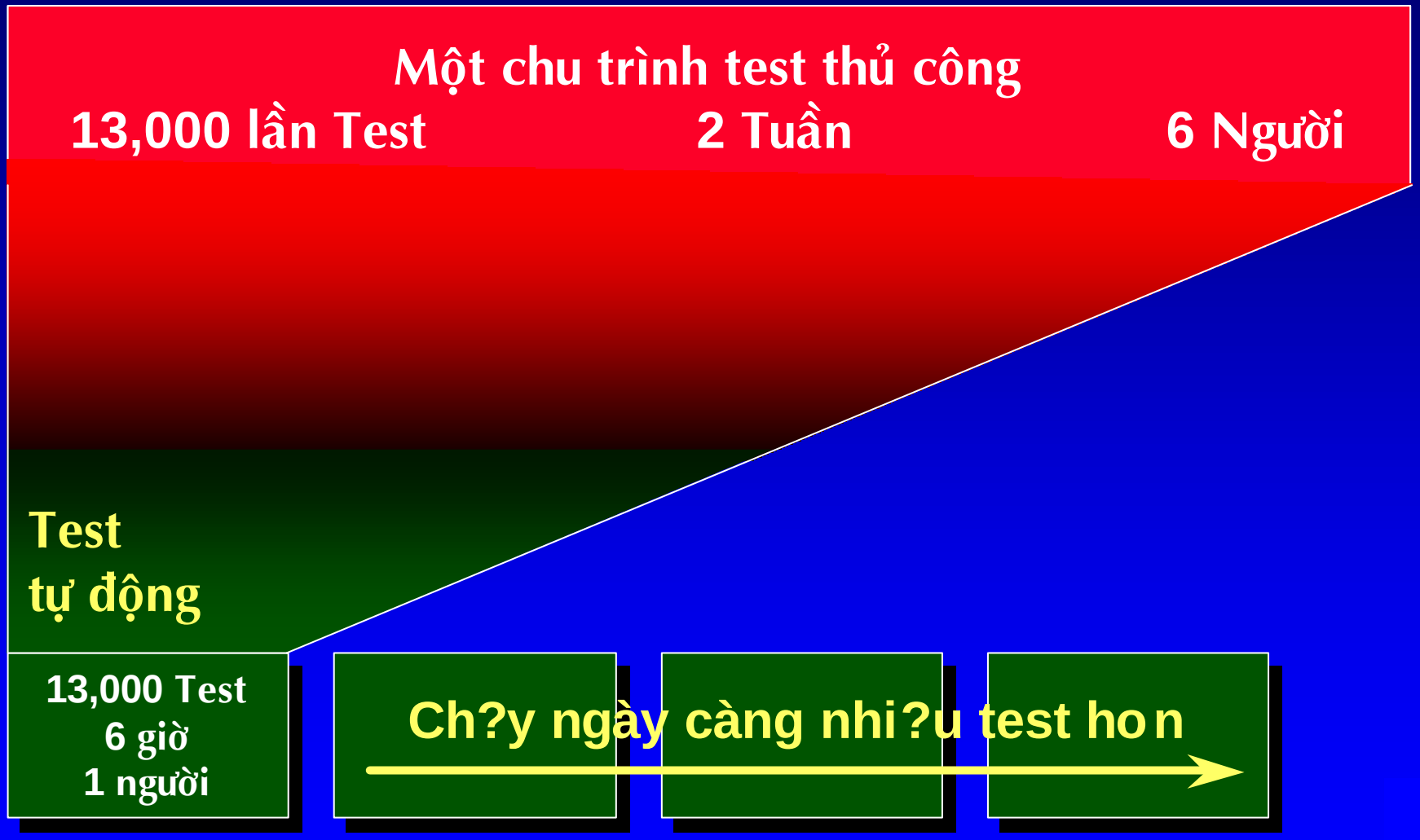


Automated Tests

Test Suite 1 Test Suite 2 Test Suite 3 Test Suite 4



Tự động hóa giảm thời gian và công sức test



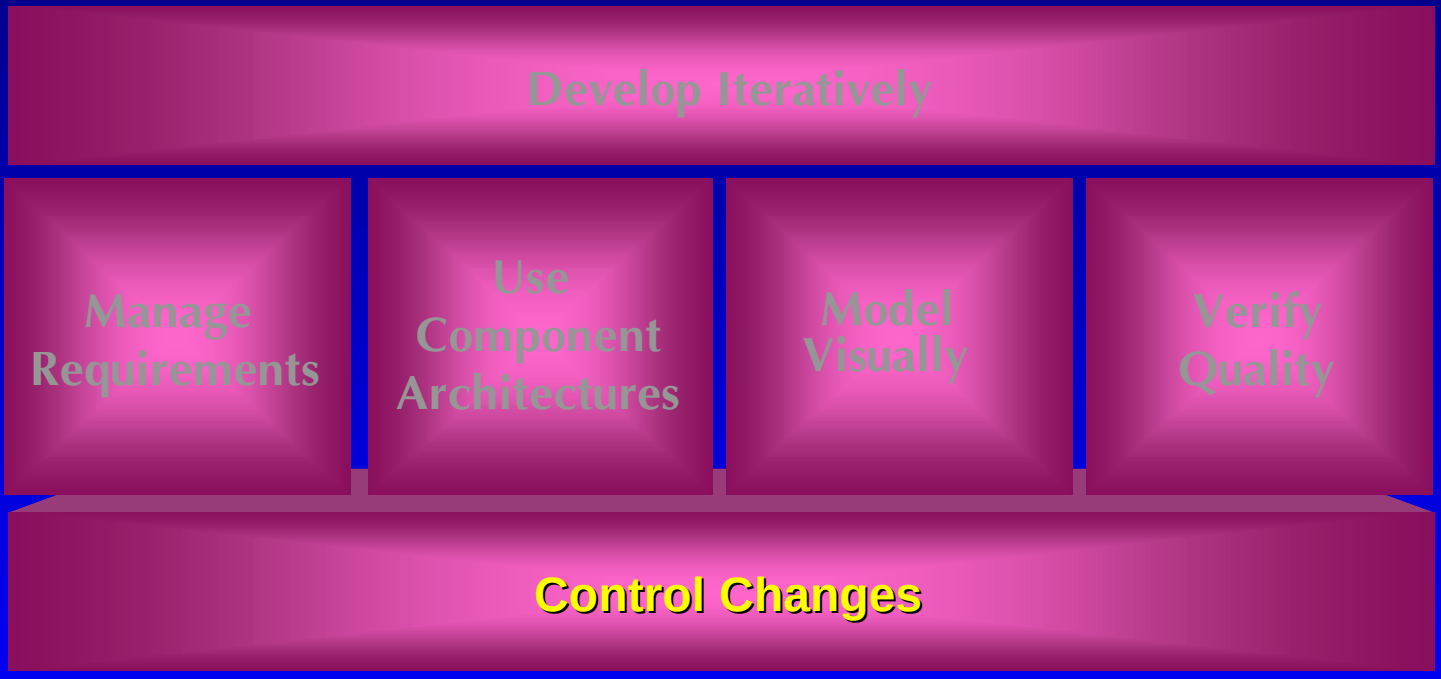
Các khía cạnh của chất lượng phần mềm

Kiểu	Tại sao?	Thế nào?
Chức năng	Ư/d của tôi có làm những gì được yêu cầu?	Tạo cácTest case cho mỗi scenario đã cài đặt
Độ tin cậy	Ư/d của tôi có làm mất bộ nhớ?	Các công cụ phân tích và các thiết bị coding
Hiệu năng ứng dụng	Ư/d của tôi có hồi đáp hợp lệ?	Kiểm tra hiệu năng của mỗi use-case/scenario đã cài đặt
Hiệu năng của hệ thống	Ư/d của tôi có hoạt động dưới công suất thiết kế?	Kiểm tra hiệu năng của tất cả use-case ở mức độ tin cậy và trường hợp xấu nhất

Các vấn đề được giải quyết nhờ kiểm định CL

Nguyeân nhaân coát loõi	Caùch giaûi quyeát
Thiếu y/c đ/v HT	Testing đánh giá khách quan về trạng thái dự án
Truyền tin mơ hồ	
Kiến trúc kém bền	
Quá phức tạp	
Đánh giá chủ quan	**Đánh giá khách quan triệt tiêu các mâu thuẫn sớm**
Các mâu thuẫn chưa được xác định	**Testing và kiểm định tập trung vào vùng high risk**
Test kém	**Tìm thấy thiếu sót sớm và chi phí sửa chữa thấp**
Qui trình thác nước	
Thay đổi không thể KS	
Thiếu ccụ tự động	**Các ccụ test tự động giúp test độ tin cậy, chức năng và hiệu năng**

Kinh nghiệm 6: Kiểm soát thay đổi trong PM

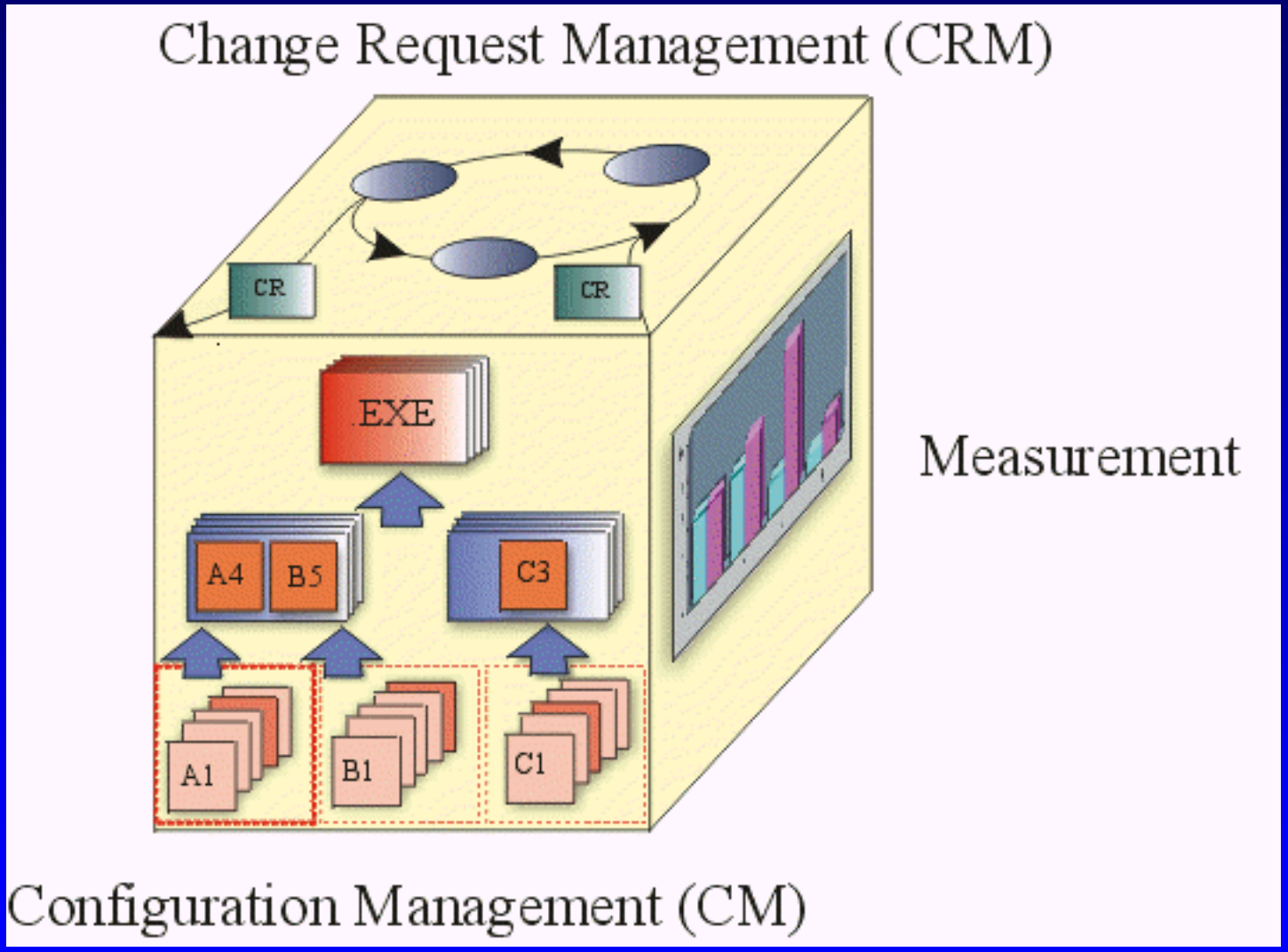


Kinh nghiệm 6: Kiểm soát thay đổi trong PM

- Nhiều developer
- Nhiều team
- Nhiều vị trí
- Nhiều vòng lập
- Nhiều release
- Nhiều project
- Nhiều platform

Thiếu sự kiểm soát tường minh, đầy đủ
Phát triển song song dễ biến thành hỗn độn

Ba khía cạnh chính của CM System



Các khái niệm của Configuration & Change M.

- Phân rã kiến trúc thành các subsystem và gán trách nhiệm thực hiện các subsystem cho mỗi nhóm
- Thiết lập vùng làm việc an toàn cho mỗi developer
  - Cho phép cô lập với các thay đổi tạo bởi vùng làm việc khác
  - Kiểm soát tất cả software artifact - models, code, docs,
- Thiết lập một vùng làm việc tích hợp
- Thiết lập một cơ chế khả thi kiểm soát các thay đổi
- Nắm bắt thay đổi xuất hiện nào xuất hiện trong release nào
- Đưa ra một đường ranh giới hạn chỗ hoàn tất của mỗi vòng lặp

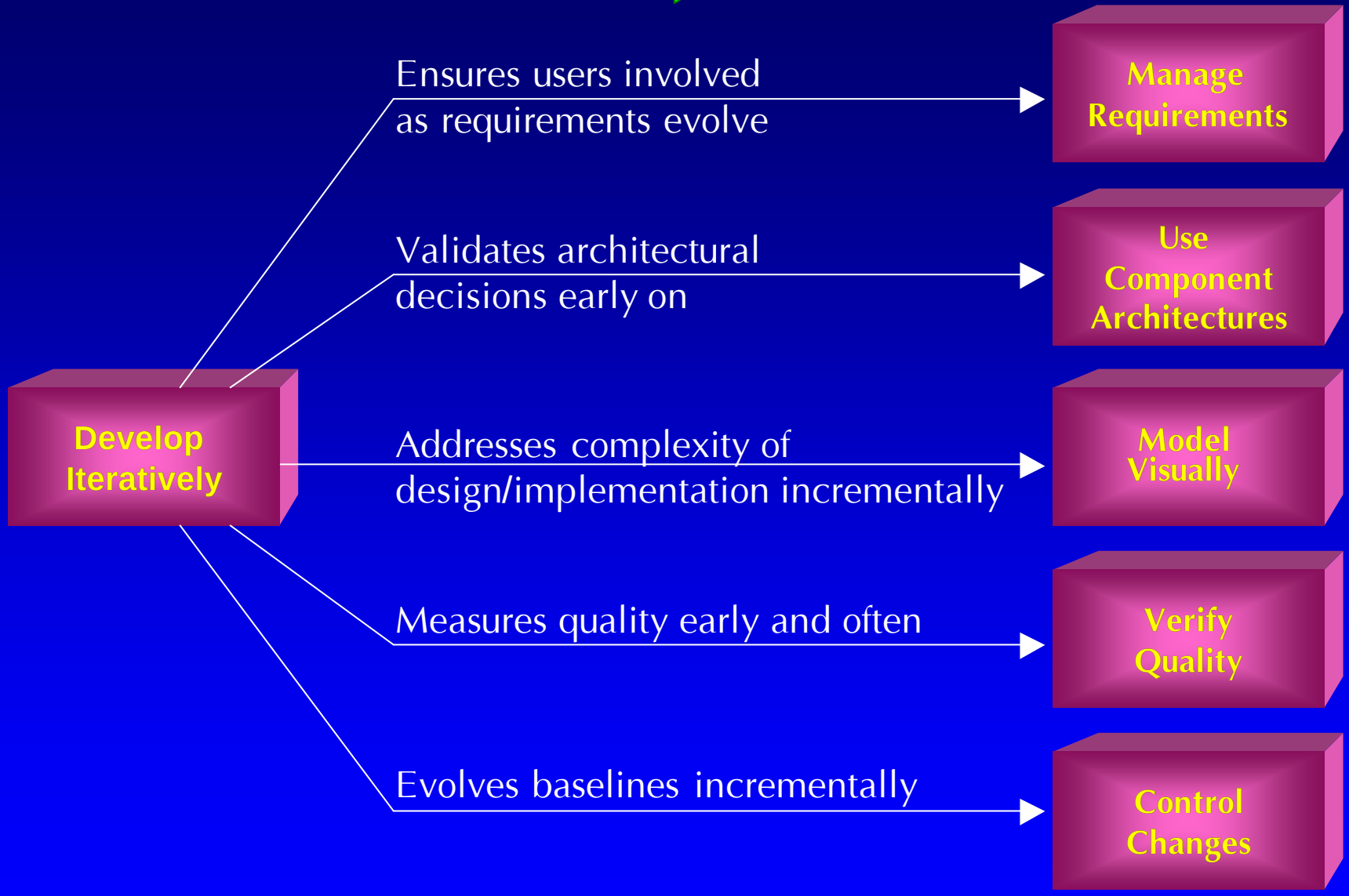
Change Control hỗ trợ tất cả Best Practices khác

- Phát triển theo qui trình lặp
- Quản lý Y/c
- Dùng kiến trúc component
- Mô hình hóa trực quan
- Kiểm định chất lượng
- Dự án chỉ tiến triển khi các thay đổi được kiểm soát
- Để loại bỏ sự dãn phạm vị, đánh giá ảnh hưởng của mọi thay đổi dự kiến trước khi chấp nhận
- Các Component phải đáng tin cậy, i.e., tìm thấy phiên bản đúng đắn của tất cả các phần hợp thành
- Để bảo đảm sự hội tụ, phải tăng dần kiểm soát các model khi các thiết kế ổn định
- Test chỉ có ý nghĩa nếu các version các phần tử đang test được biết rõ và các phần tử được bỏa vệ trước các thay đổi

Các vần đề được giải quyết nhờ Control Change

Nguyên nhân cốt lõi	Cách giải quyết
Thiếu y/c đ/v HT	Requirements change workflow được xác định và lặp lại đi lặp lại
Truyền tin mơ hồ	Các Change request làm cho thông tin trao đổi rõ ràng
Kiến trúc kém bền	
Quá phức tạp	Vùng làm việc biệt lập giảm các trở ngại do làm việc song song
Đánh giá chủ quan	Thống kê về mức độ thay đổi là độ đo tốt cho các đánh giá khách quan về trạng thái của dự án
Mâu thuẫn chưa được xác định	Vùng làm việc chứa tất cả các artifact để tạo sự nhất quán
Test kém	
Qui trình thác nước	
Thay đổi không thể kiểm soát	Kiểm soát được sự lan truyền các thay đổi
Thiếu ccụ tự động	Các thay đổi được duy trì trong một hệ thống mạnh mẽ, có khả năng tùy chỉnh

Các kinh nghiệm hỗ trợ lẫn nhau



Tổng kết

- Kết quả là phần mềm trở nên
  - Đúng thời hạn
  - Bảo đảm ngân sách
  - Thỏa mãn nhu cầu user