COMP 3711

OOA/OOD

Review For Final Exam

Lecture # Week of	Class Lecture – Outcome / Material Covered	Reference/ Reading	Ass' mt	Labs / Quizzes
W1 Sep 1-5	No classes			No labs
W2 Sep 8-12	Review course outline. Refresher on OOAD, UML Iterative Development – UP, Agile and Case Studies	Chapter 1, 22 Chapter 2-3		Lab 1 (UML Diagramming tool)
W3 Sep 15-19	Inception - Stories and Use Cases Relating Use Cases Elaboration – Iterative and Evolutionary	Chapter 4-6 Chapter 30 Chapter 8		Lab 2 (Use Case diagram) Quiz 1
W4 Sep 22-26	Domain Model - Conceptual Classes, adding associations and attributes Class Hierarchies, Generalization, Composition, Association	Chapter 9, 31.1 Chapter 31.2-31.17		Lab 3 (Domain Class diagram)
W5 Sep 29-Oct 3	Detail Refinement - Operation Contracts SSD – Systems Sequence Diagrams	Chapter 11 Chapter 10, 32		Lab 3 continue Quiz 2
W6 Oct 6-10	Interaction Diagrams Design Class – add methods, dependency	Chapter 14, 15 Chapter 16	A1 due Oct 10	Lab 4 (System Sequence diagram) Quiz 3
W7 Oct 13-17	*No class lecture on Oct 13 (Thanksgiving)* GRASP and Patterns, Separation of assigning responsibilities	Chapter 17		*No lab on Oct 13* Lab 5 (Design Class)
W8 Oct 20-24	**Review** **Midterm Exam (Oct 21)**			Lab 5 (Design Class)
W9 Oct 27-31	Use Case Realization, assigning GRASP patterns to object design and More Grasp Determining Visibility/Mapping designs to code	Chapter 18, 25 Chapter 19-20		Finish Lab 5 and do Assignment 2
W10 Nov 3-7	Test driven development and refactoring Moving on iterations	Chapter 21 Chapter 23-24, 27	A2 due Nov 7	Lab 6 (Implement code) Quiz 4
W11 Nov 10-14	Machine Modeling UML Deployment, Component Diagrams *No class lecture on Nov 11 (Remembrance)*	Chapter 29		Lab 7 (Tester Tutorial)
W12 Nov 17-21	QA overview Test Model and Test Plan			Lab 8 (Functional Tester Tutorial) Quiz 5
W13 Nov 24-28	Test Cases and Test Scripts Test Types and Execution			Lab 9 (Apply Test Script)
W14 Dec 1-5	Test Automation Comparison, Pre and Post Processing Review		A3 due Dec 1	Quiz 6
W15 Dec 8-12	**Final Exam (date to be scheduled)**			

COMP 3711
2008 Fall
Schedule
Version 3

Final Exam

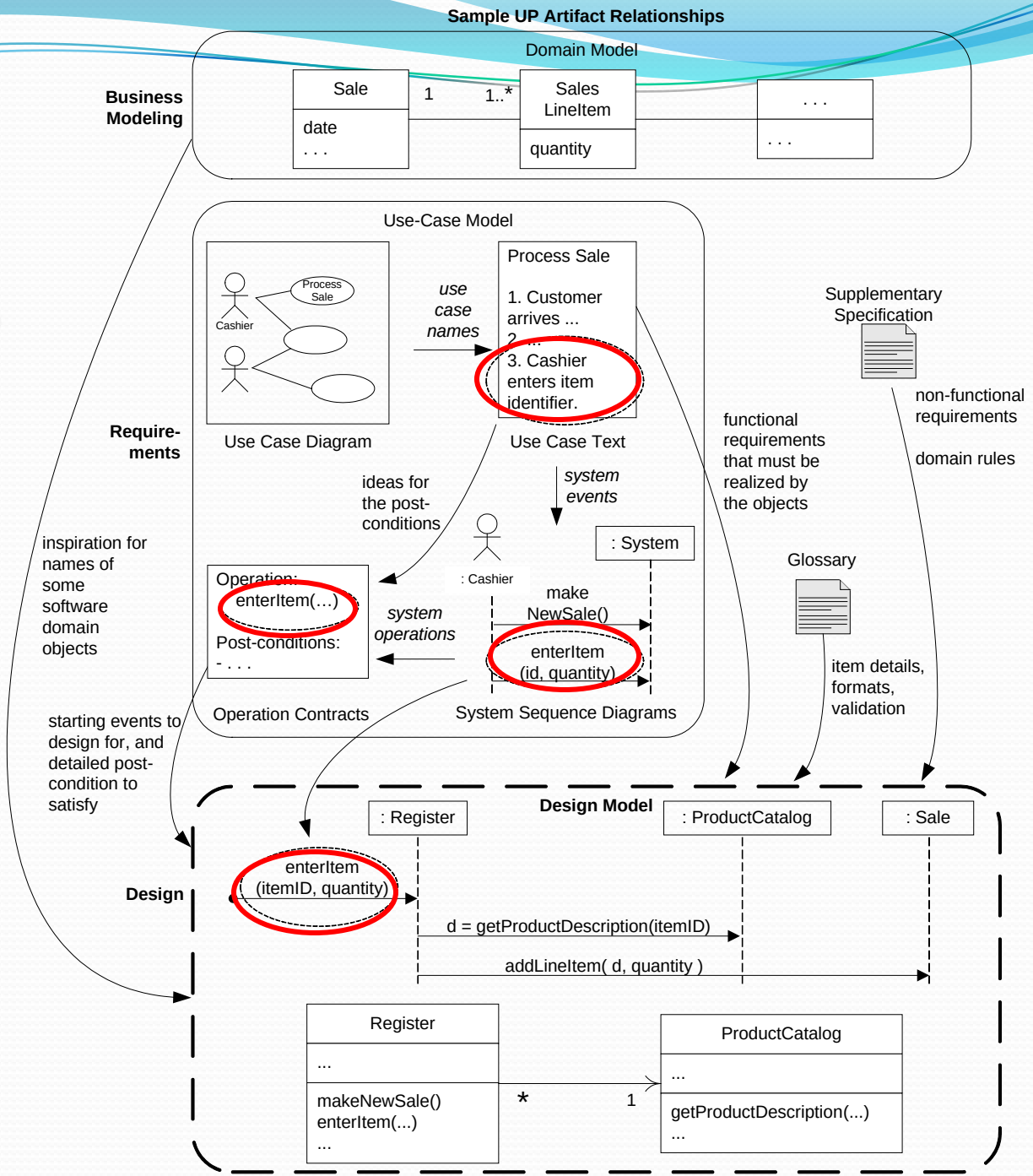
- *Based on the knowledge and midterm feedback you've gained in the first half of the course before the midterm exam, the final exam will focus on the second half of the course starting from the lectures material and tutorials from week 9 (see the red highlight on previous slide).*

Week 9 Use Case Realization

- *Move from Domain model to Design model by applying appropriate patterns such as GRASP, GoF, and Visibility). You should be familiar with GRASP and Visibility (which requires you to understand inheritance). GoF is covered in Week 10.*

- *Use Case Realization creates new Design Model artifacts such as Sequence Diagram, Collaboration Diagram and DCD.*

OOD Design Artifacts



Larman fig. 18.1

Use Case Realization In Design Model

- Use Case Realization can be designed from:
  - Use case description / use case diagram
  - Operation contracts (e.g. work through post-condition state changes and design message interactions to satisfy requirements)
  - Domain Model (e.g. iterative design that permits inclusion of new conceptual classes that were missed)
  - User (domain experts)

Use Case Realization-Starting Points

- Use Case artifacts suggest the system operations that are shown in SSDs

- System operations are the starting messages entering the Controllers for the domain layer interaction diagrams

- Domain layer interaction diagrams shows how objects interact to fulfill the required tasks

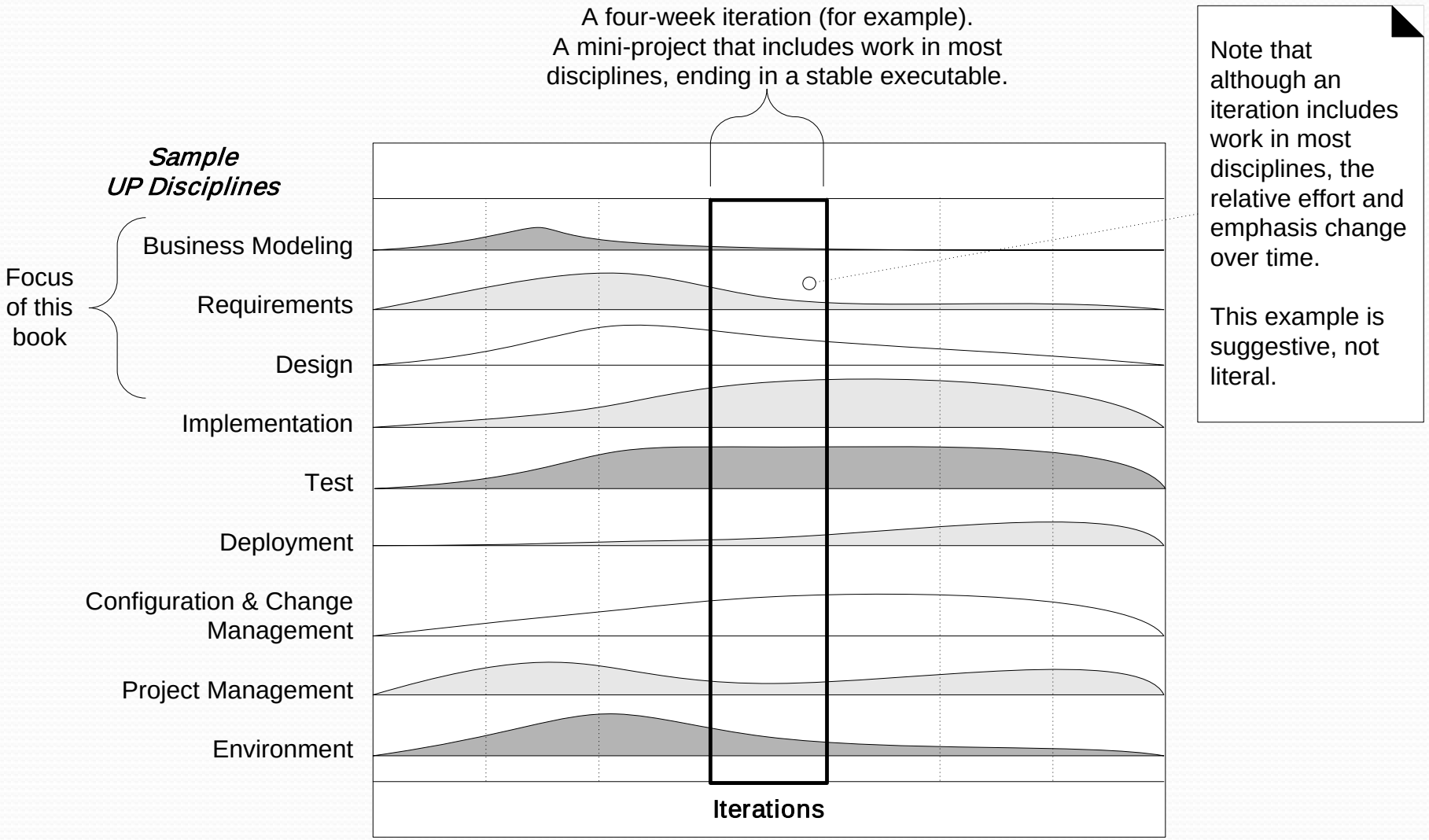
Week 10 Iterative Development

- *One of the best practices in UP and Agile is the practice of iterative development*
- *We examine 5 out of 23 GoF (Gang-Of-Four are Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides) patterns in this week.*
- *Based on the principle of TDD (Test Driven Design), we look at the 2 core practices of Agile (XP): continuous testing and refactoring.*

UP – Reinforces 6 Best Practices

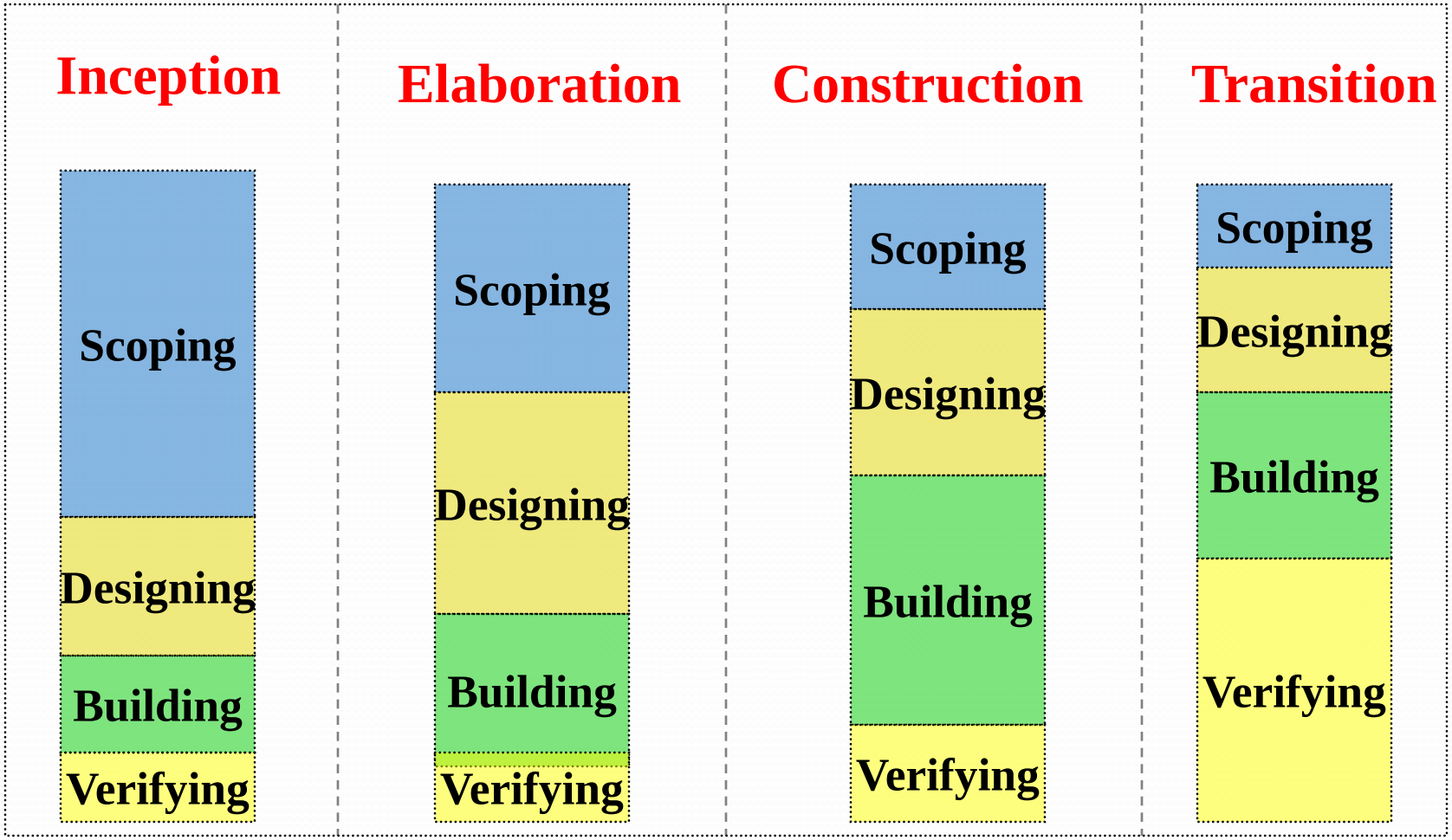
- Develop iteratively
- Define and manage system requirements
- Use component architectures
- Create visual models
- Verify quality
- Control changes

Iterative UP

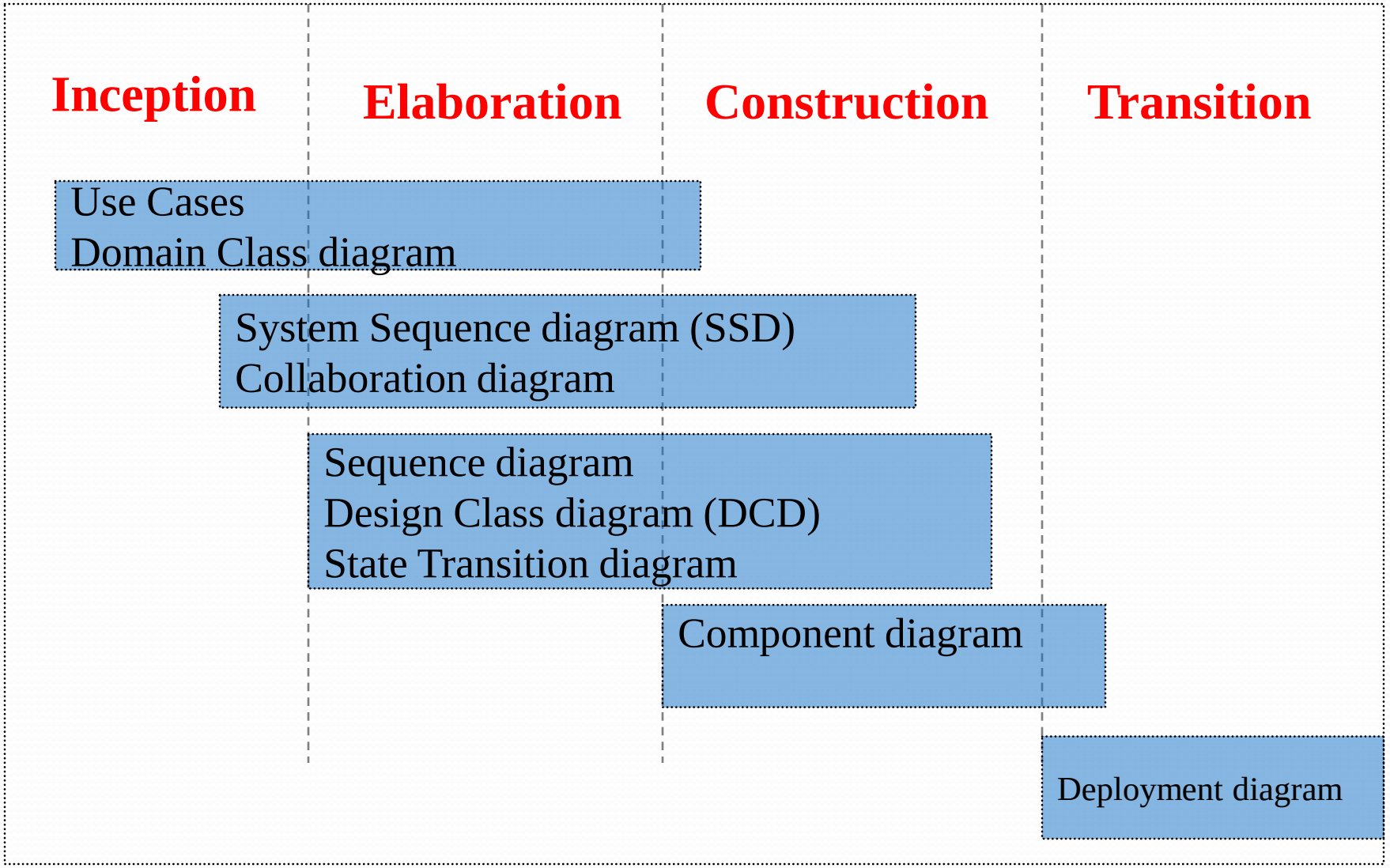


Larman Fig 2.7

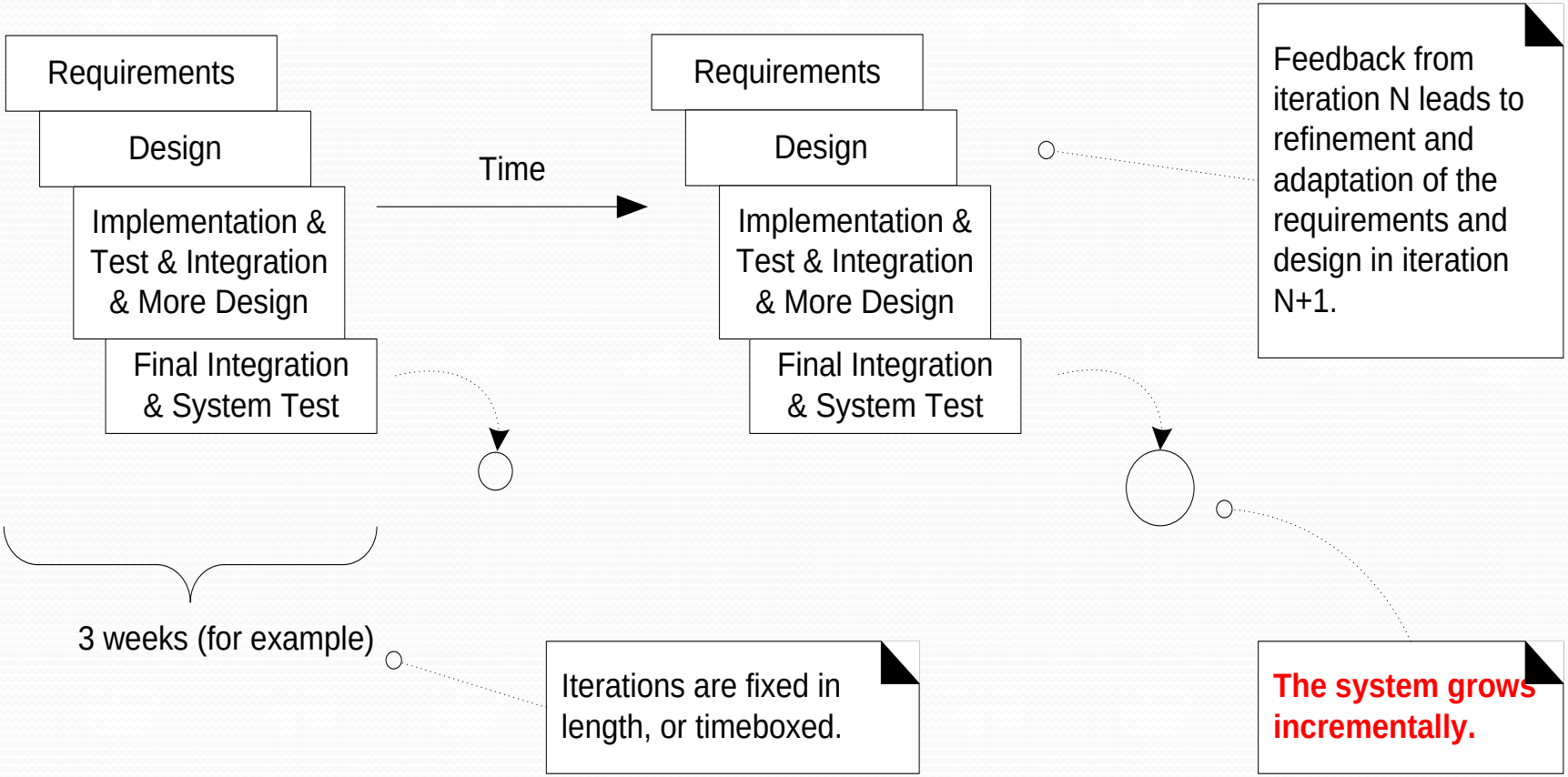
Iterative UP



UML Diagrams In UP Iterations



Iterative Development in UP



Larman fig 2.1

GoF Design Patterns Overview

"Design Patterns", mostly coded in C++ and Smalltalk , was introduced in 1994 by the Gang-of-Four, covering 23 patterns with 15 commonly used.

Creational	Structural	Behavioral
• Abstract • Factory • Builder • Prototype • Singleton	• Adapter • Bridge • Composite • Decorator • Facade • Flyweight • Proxy	• Chain of Responsibility • Command • Interpreter • Iterator • Mediator • Momento • Observer • State • Strategy • Template Method • Visitor

The GoF Patterns

- Creational
  - Deal with class instantiation
- Structural
  - Deal with Class and Object composition
- Strategy
  - Deal with communication between objects

A Few Selected GoF Patterns For NextGen POS Example

- Adapter
- Factory
- Singleton
- Strategy
- Composite

Test Driven Development (TDD)

- Unit Test refers to a test of individual parts of an application, as oppose to Application Test
- The unit test code is written before the class to be tested (not an after-thought exercise)
- Developer writes unit testing code for all production code
- The unit test is written first, imagining the code to be tested is written and the challenge is to write code that will pass the test

XP – Continuous Testing

Test a little → Code a little

1. Planning	*7. Paired Programming*
2. Small Releases	*8. Collective Ownership*
3. System Metaphor	*9. Continuous Integration*
4. Simple Design	*10. 40-Hour Week*
5. Continuous Testing	*11. On-site Customer*
6. Refactoring	*12. Coding Standards*

5. Continuous Testing

- Unit tests (test single class or cluster of classes) - written by developer
- Acceptance testing (overall system is functioning) - written by users
- Paired programming allows better test plans
- Simple design allows frequent automated testing

XP – Refactoring

1. Planning
2. Small Releases
3. System Metaphor
4. Simple Design
5. Continuous Testing
6. Refactoring
7. Paired Programming
8. Collective Ownership
9. Continuous Integration
10. 40-Hour Week
11. On-site Customer
12. Coding Standards

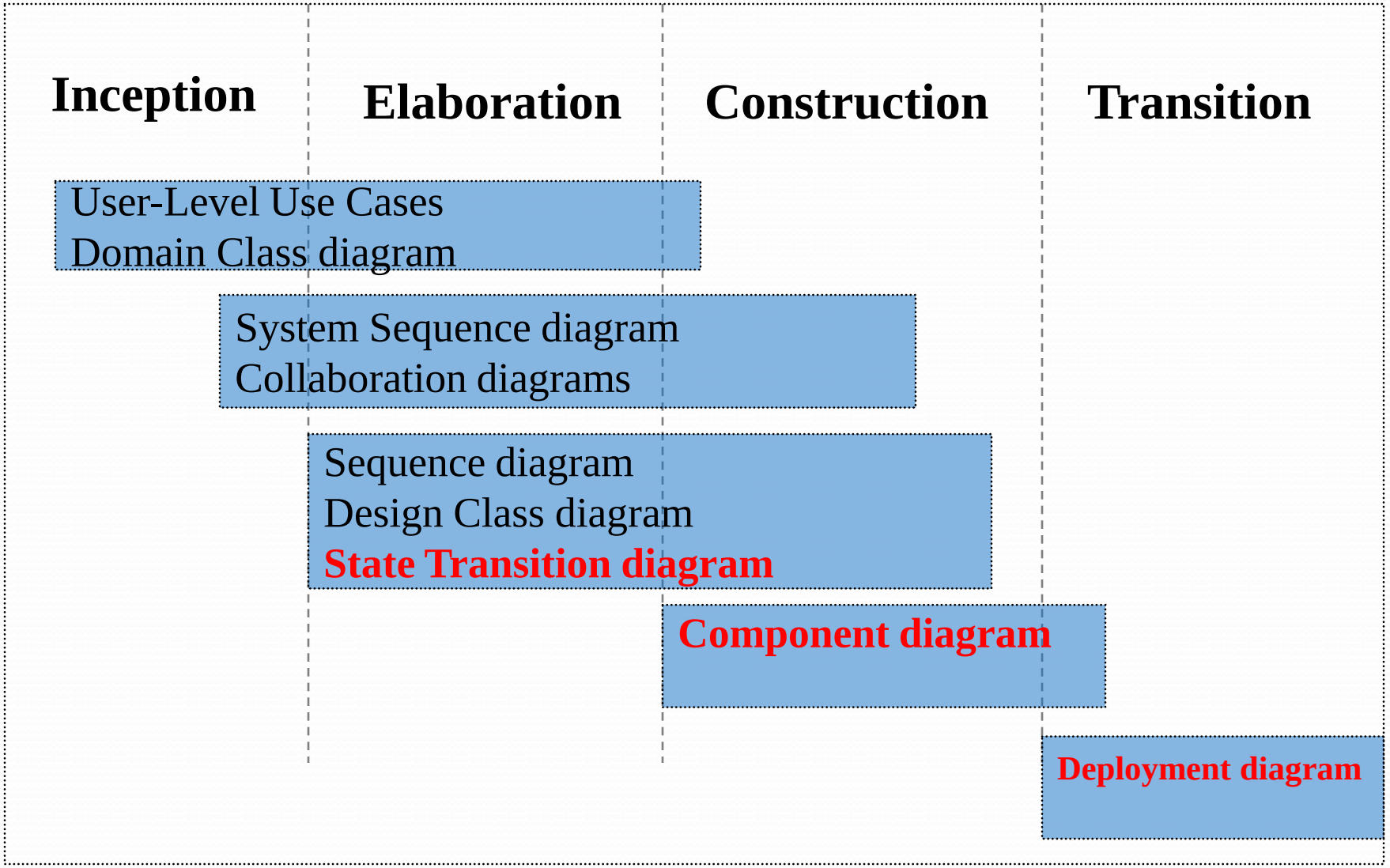
6. Refactoring

- Restructure and simplify the system without changing its behavior
- Refactor out duplications (based on needs by systems and code)
- Simple Design, Continuous Integration, Collection Ownereship and Paired Programming foster an environment that provides confidence to refactor

Week 11 More UML Diagrams

- *We finished up with a more comprehensive understanding of UP and UML by examining the State-Transition diagram, Component diagram and Deployment diagram. But in doing so, we need to draw relevance to some Architecture issues / artifacts.*

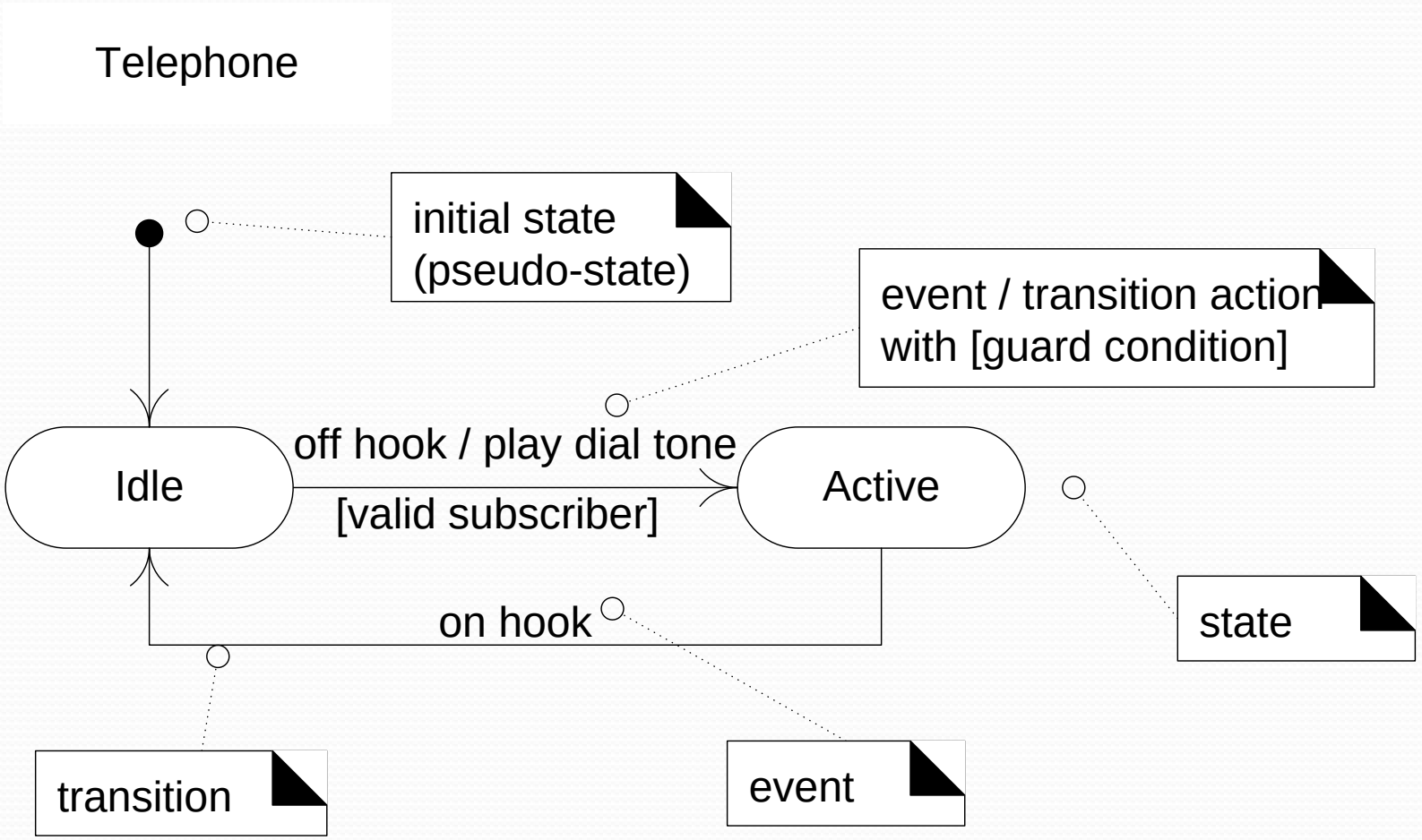
UML Diagrams Over UP



UML State Machine Diagram

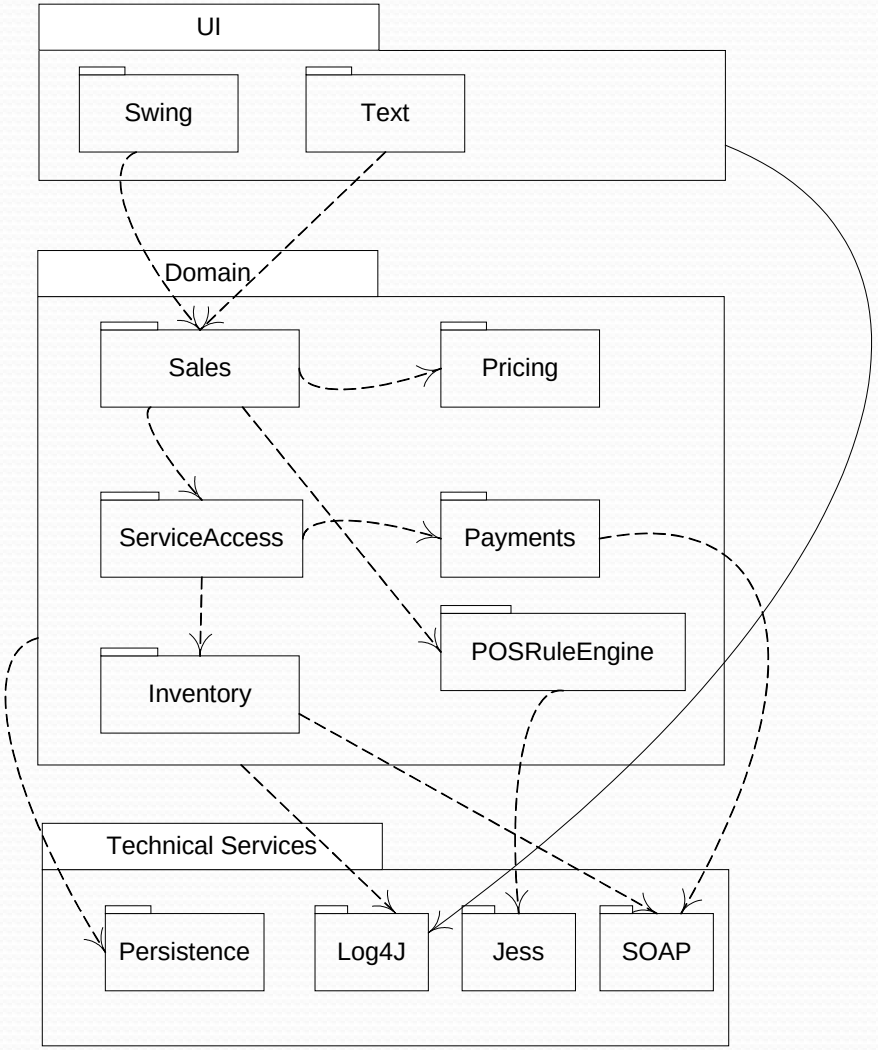
- State Machine Diagram = State Transition Diagram = Statechart
- Shows transition behaviour of an object in reaction to an event through out the object lifecycle
- An “event” is a significant or noteworthy occurrence
- A “state” is condition of an object at a moment in time between events, 1 start state, multiple or 0 stop states
- A “transition” is a relationship between two states as the object moves from prior state to subsequent state

State Machine Diagram Notations



Larman 29.1, 29.2

NextGen Logical Architecture Diagram



Showing key note-worthy elements for “big ideas” and ignoring the Application layer

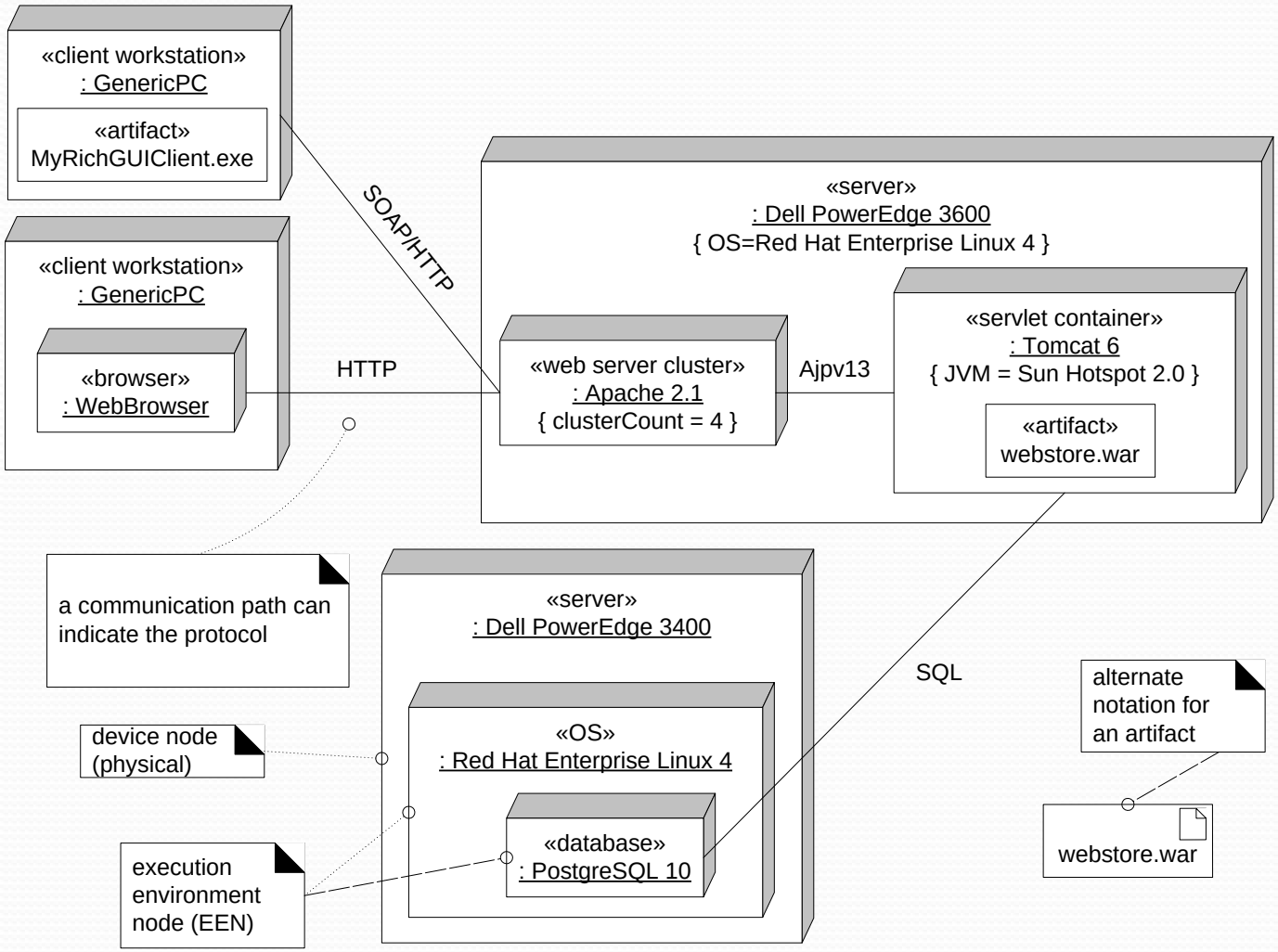
Larman 34.3

UML – Component Diagram

- *Quote from UML spec[OMG-03b] "A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. A component defines its behaviour in terms of provided and required interfaces. As such, a component serves as a type, whose conformances is defined by these provided and required interfaces"*

- Represent design-level perspective and not concrete software perspective

Example Of Deployment Diagram



Larman 38.1 A Deployment Diagram is often used to communicate the physical and deployment architecture

Week 12-14 Software Testing

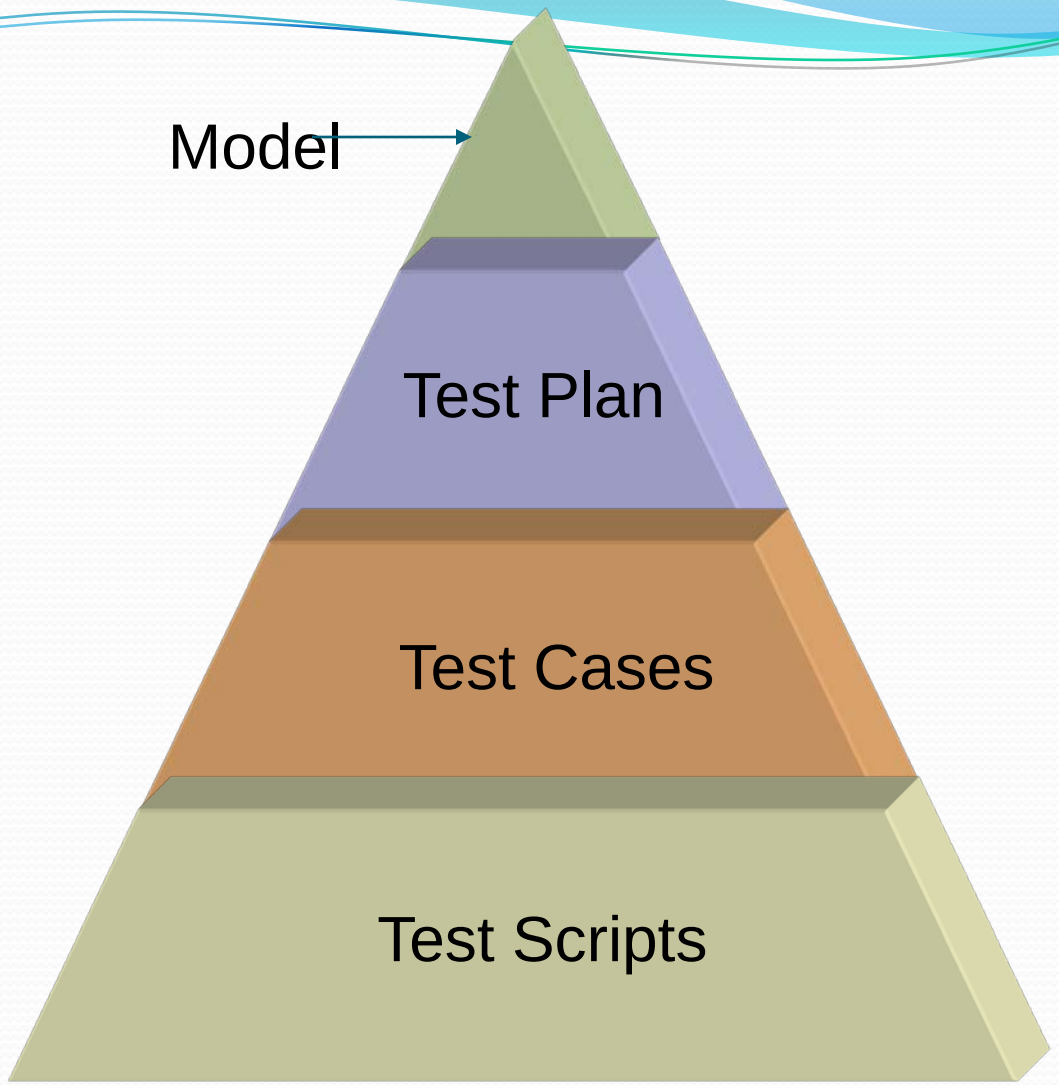
- *We finish our course by looking at the various kinds of software testing, from unit test typically done by the programmer / developer to acceptance(functional tests) done by the users.*

- *A good understanding of the discipline of testing will improve the quality of the software being developed, based on the TDD principle found in UP and Agile. It is necessary for your iterative development.*

Week 12-14 Software Testing

- *Week 12-13, we look at the Test Model, Test Plan, Test Cases, Test Scripts (including the use of verification points and equivalent classifications).*

- *Week 14, we look at measures and maintenance applied to testing.*


Model
Test Plan
Test Cases
Test Scripts

Test Plan: Document

- Describes:
  - scope, approach, resources, schedule of intended testing activities
- Identifies:
  - test items, features to be tested, testing tasks and sequence, who will do each task, any risks requiring contingency planning, description of the test environment

Test Case

- A commonly used term for a specific test
- This is usually the ***smallest unit of testing***
- Consists of information such as:
  - Requirements tested, test steps, verification steps, prerequisites, outputs, test environment state
  - Set of inputs, execution preconditions, and expected outcomes
- Developed for a particular objective:
  - e.g. exercise a particular program path
  - e.g. verify compliance with a specific requirement

Test Case Development

- Test the things that are most important first.
  - Often used ***scenarios***
  - ***Equivalence classes*** and ***components*** involved in these
- Can test the scenario with different equivalent classes (one test case for each).
- Could include several equivalent classes in a scenario but make sure your test case doesn't become too complicated.

Test Scripts

- This is the real thing
- A detailed ***step-by-step series of instructions*** for operating the program/application being tested
- ***Automated*** or ***manual*** scripts
- Often include expected results (we will include)
- Test scripts have traditionally been sets of instructions for the testers
- Very important that testers follow scripts so that tests are ***repeatable***
- For one or more test cases

Detailed manual scripts

- **Verification point**
  - a point at which we want to verify the output; requires expected output

Script Documentation

- A little bit of documentation will go a long way
- No substitute for well designed, well organized code
- Automatically generated code may require more documentation
  1. Purpose
  2. Inputs and outputs
  3. Anything tricky or unusual in the implementation

Scripting Techniques

1. Linear
2. Structured
3. Shared
4. Data-driven
5. Keyword-driven

Prescriptive

Descriptive

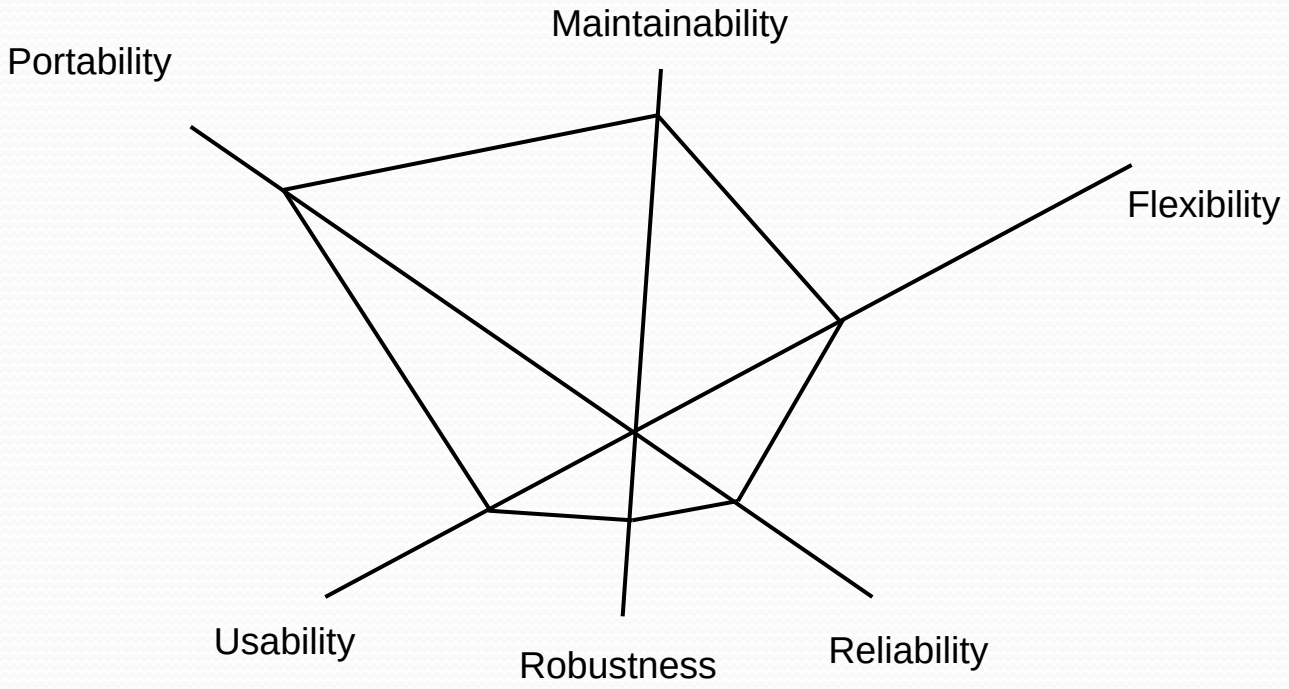
Equivalence Classes

- If two tests produce the same result, they're equivalent
- A group of test forms an equivalence class if:
  - They all test the same thing
  - If one test finds a bug, the others probably will too
  - If one test misses a bug, the others probably won't

Example: Table Listing Equivalence Classes

Input or Output Event	Valid Equivalence Classes	Invalid Equivalence Classes
Enter a number	Numbers between 1 and 99	0 > 99 An expression that yields an invalid number, such as 5-5, which yields 0 Negative numbers Letters and other non-numeric characters
Enter the first letter of a name	First character is a capital letter First character is a lower case letter	First character is not a letter
Draw a line	From 1 dot-width to 4 inches long	No line Line longer than 4 inches Not a line (curve)

Various Measurements

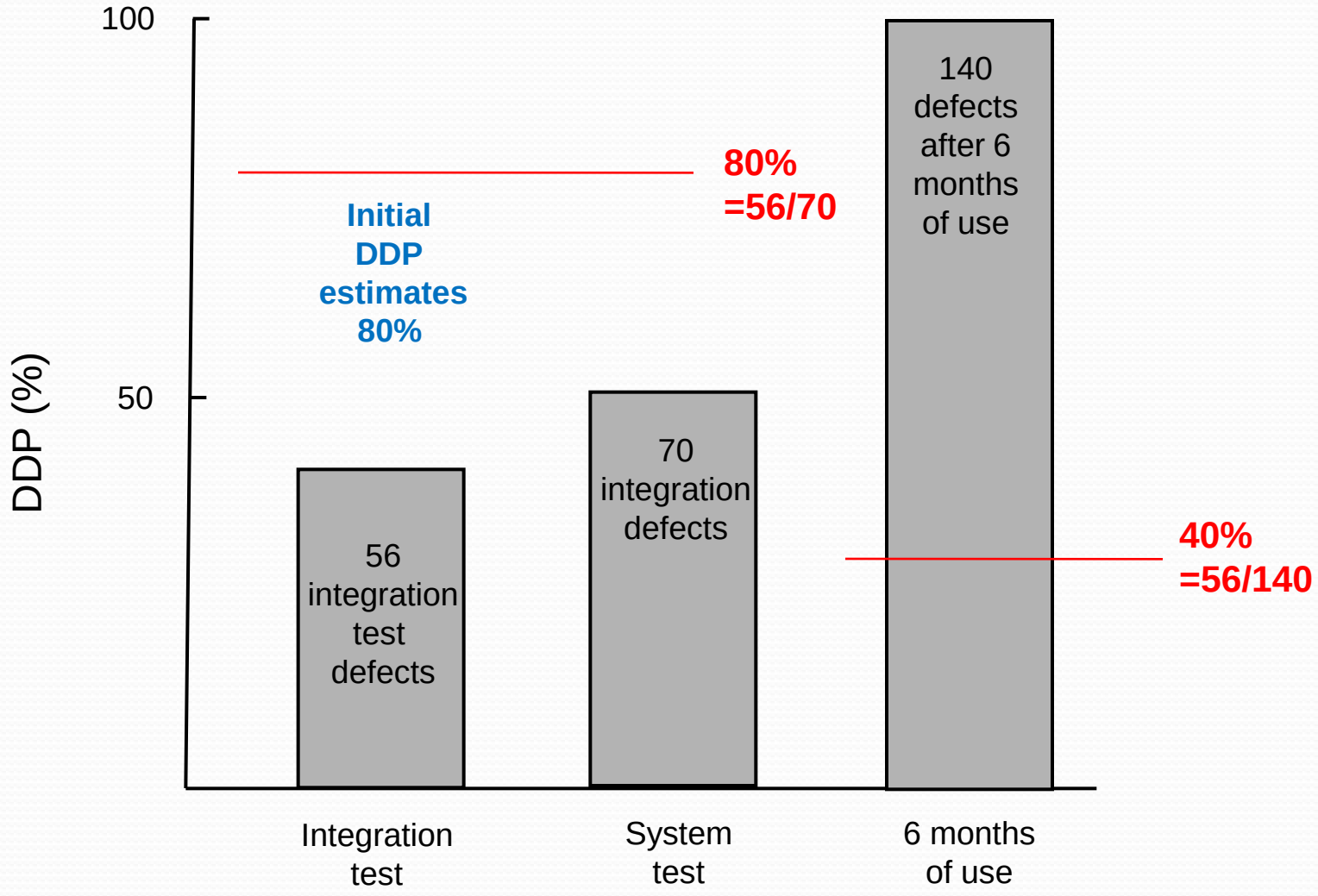


Example: Measuring test effectiveness

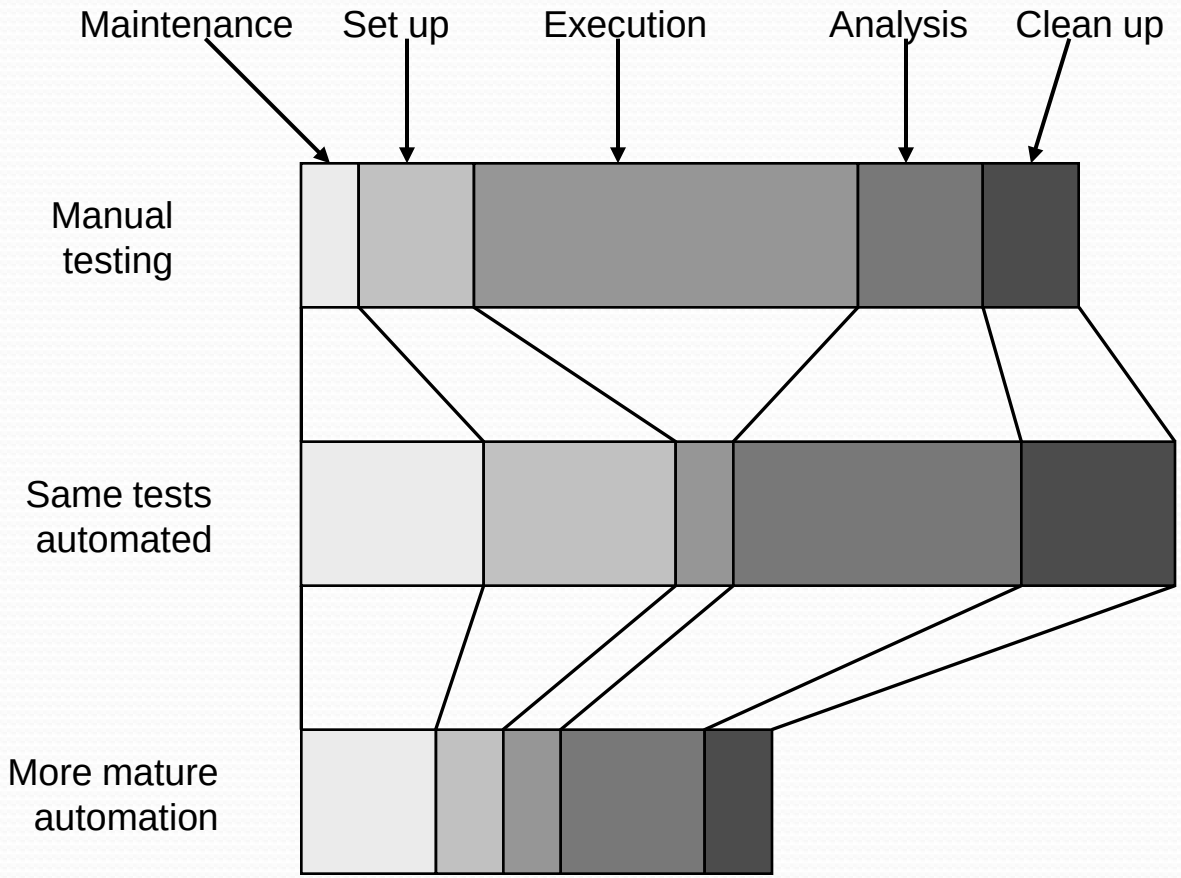
$$DDP = \frac{defectsfoundbytesting}{totalknowndefects}$$

- DDP = Defect Dectection Percentage
- Totalknowndefects=number of defects found by this test + number of defects found afterwards.
- Measurement of how effective test process is in finding bugs
- DDP index will decline as more bugs are found in service (i.e. more effective testing captures those defects that had escaped earlier detection)

Example: DDP At Different Stages

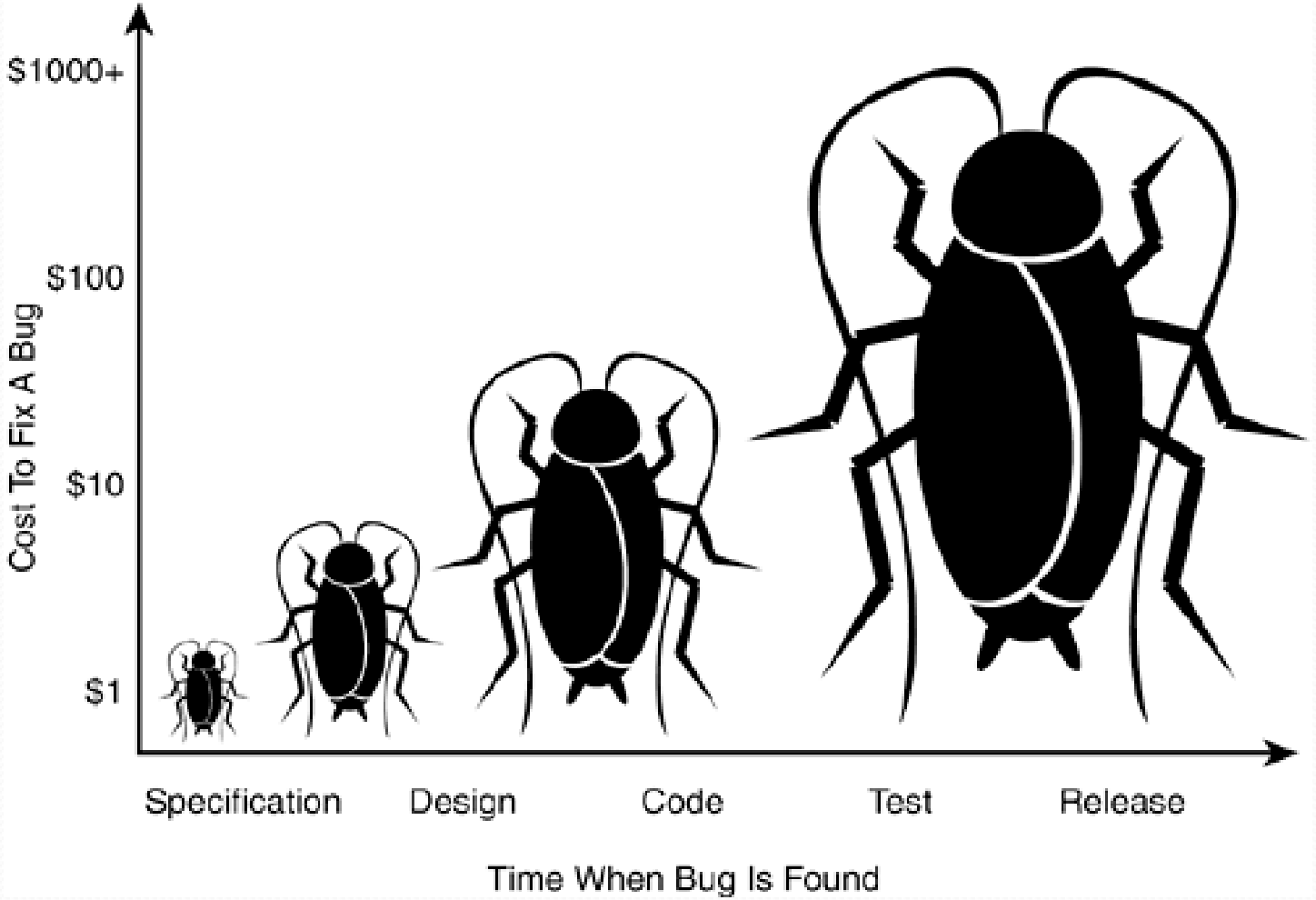


Efficiency - Test Automation



Relationship between test activities in manual testing, early automation and more mature automation

Cost of fixing bugs



Preprocessing and Post Processing

1. Preprocessing for an automated suite of tests puts the environment into the state needed to run the test case

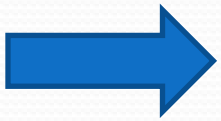
2. Postprocessing puts the environment in some known state after the test suite has been run

F.I.R.S.T.

- Clean tests have the following charateristics:
  - Fast
    - Tests should run quickly
  - Independent
    - Tests should not depended on another test
  - Repeatable
    - Tests should be repeatable in any environment (e.g. QA, production, client)
  - Self-Validating
    - Tests should have boolean output (pass/fail)
  - Timely
    - Tests should be written before coding

Ask These Two Big Questions

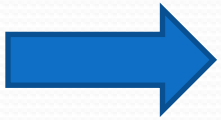
- Are we building the software product "right"?



Verification

Specifications conformance

- Are we building the "right" software product?



Validation

Requirements compliance