COMP 3711

(OOA and OOD)

Software Testing 1
Introduction

Testing Computer Software (2ed) Kaner/Falk/Nguyen (Chapters 1-4)
Lessons Learned in Software Testing, Kaner/Bach/Pettichord (Chapters 1-3)

Why look at software testing now?

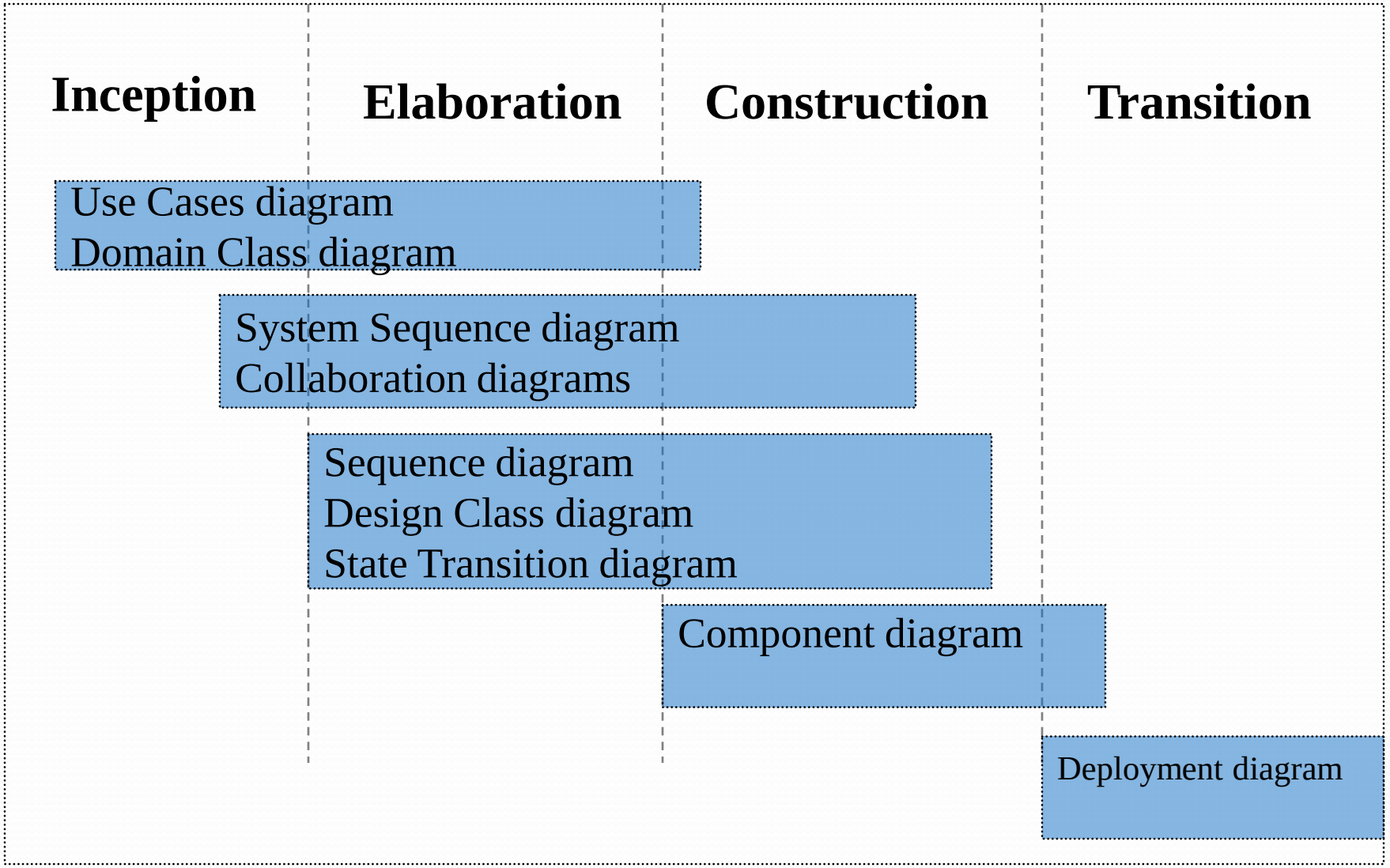
- This course focuses on OOA / OOD, why do we look at testing?

QUALITY

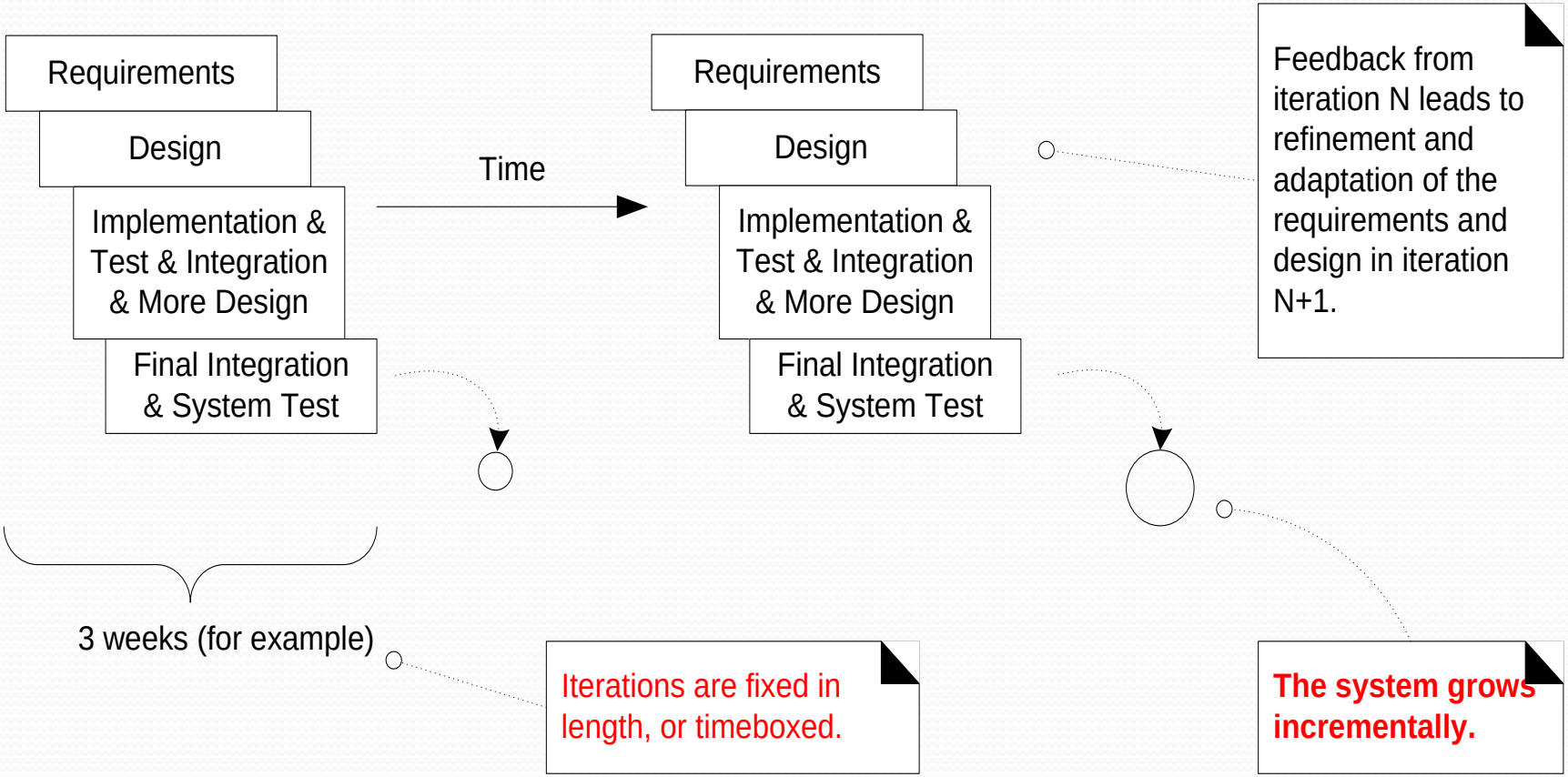
- UP and Agile offer iterative OO development with XP practices such as *continuous testing* (with writing unit-test before coding) , *refactoring* and *continuous integration*

- UML notation and UML-IDE tool support OO iterative development that produces incremental small releases

UML Diagrams Over UP

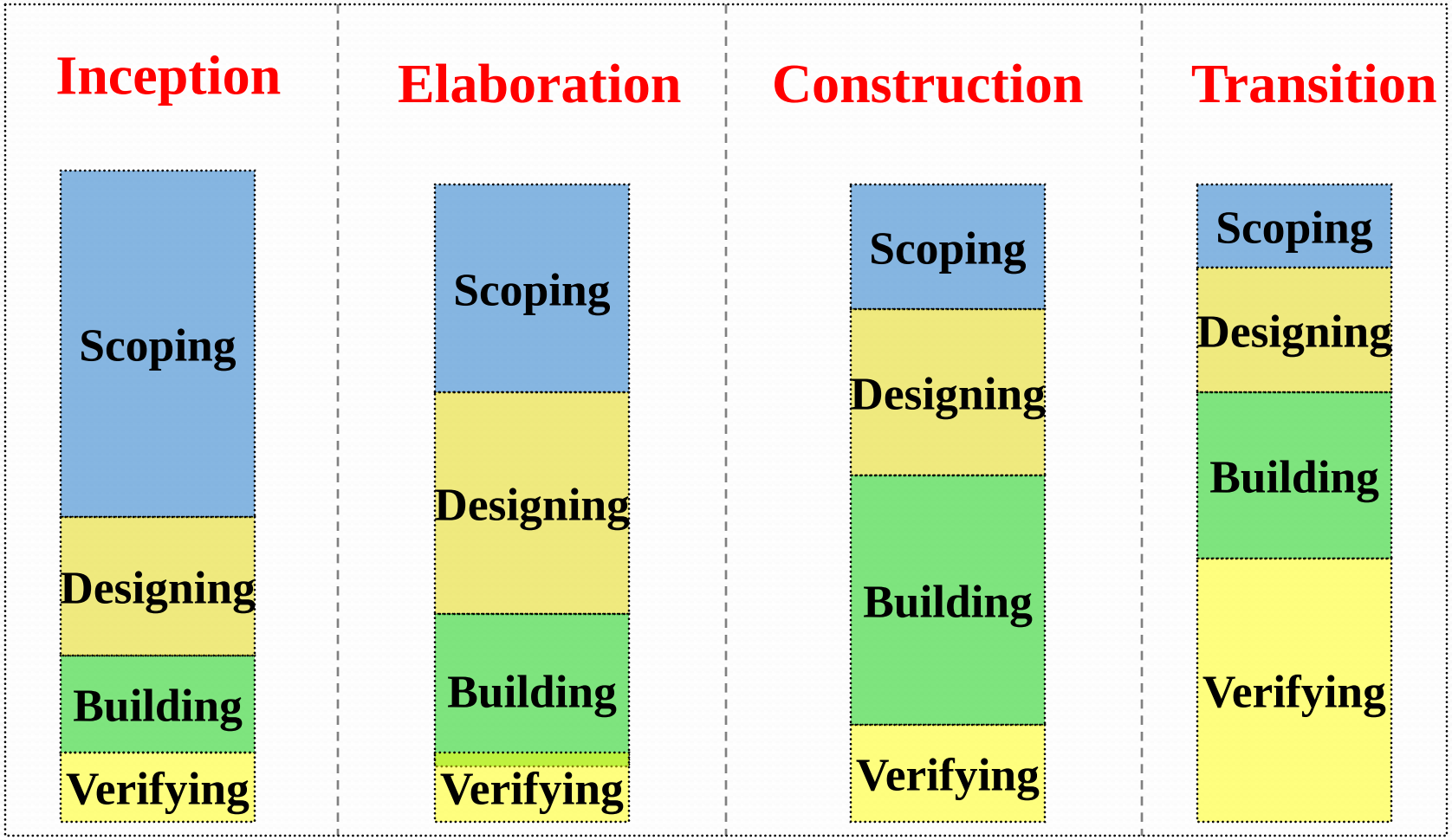


Iterative Development in UP



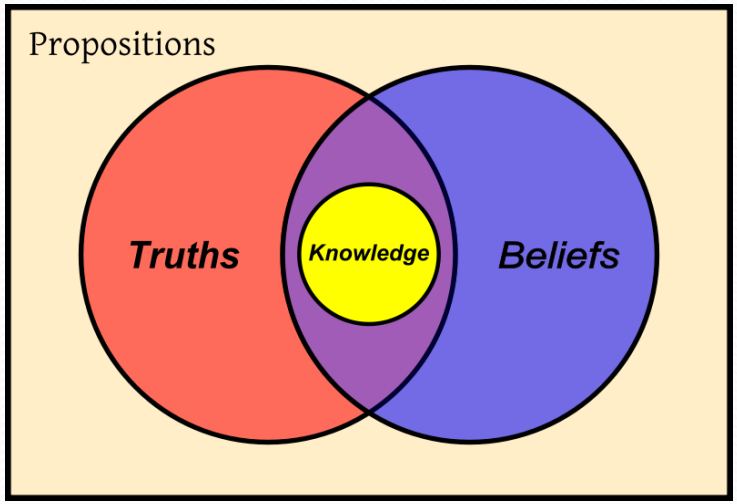
Larman fig 2.1

Iterative UP



Epistemology

- Applied epistemology
  - from Greek *επιστήμη* - *episteme*, "knowledge" + *λόγος*, "logos"



- *Study of*
  - *What is knowledge?*
  - *How is knowledge acquired?*
  - *What do people know?*
  - *How do we know what we know?*

Apply Epistemology To Software Testing

- *How do you know the software is good?*
- *How would you know if it wasn't good enough?*
- *How do you know you've tested enough?*
- *Critical examination of belief … Socrates*

Apply Epistemology To Software Testing

- *How to gather and assess evidence?*
- *How to make valid inferences?*
- *How to use different forms of logic?*
- *What it means to have a justified belief?*

Apply Epistemology To Software Testing

- *Testing is in your head*
  - *Ability to make design choices*
  - *Ability to interpret what you observe*
  - *Ability to make exploratory inference (from one idea leading to another…)*
  - *Ability to tell a compelling story about it*

Tester

- *A good tester thinks*
  - *technically*
    - *model technology, causes/effects*
  - *creatively*
    - *generate ideas and see possibilities*
  - *critically*
    - *evaluate ideas and make exploratory inferences*
  - *practically*
    - *put ideas into practice*

Testing

- *All testing is based on models (starting from a mental picture)*
- *A flawed model results in flawed tests*
- *Good tester is*
  - *skillful in the art of modeling*
  - *exploring and understanding the model*
    - *Forward thinking*
    - *Backward thinking*
    - *Lateral thinking*

Some Big Questions

- *What would failure look like?*
- *Why should this feature fail?*
- *What risk factor likely to have adversely affected this feature to fail?*

Abductive Inference (Hypothetical Induction)

- *Reasoning to the best explanation*
  - *Gather some data … reliable data*
  - *Construct explanations on the gathered data*
  - *Seek more data that help corroborate or refute each explanation*
  - *Choose the most coherent explanation that accounts for all the important data*

Is it possible to prove?

- *Is it possible to prove the software works 100%?*
- *Is it possible to find all the bugs?*
- *Unless you run every possible test*
- *Confidence is based on the results from the tests performed*

Why Complete Testing is Impossible?

- Very large number of possible inputs
- Very large number of possible outputs
- Very large number of paths through the software
- The specification or goal is seldom completely clear
  - How to test a program when you don't know exactly what it is supposed to do

Example

- How many test cases to completely test the following?

```
myfunction(int a,int b,int c)
```

- How many test cases to completely test the following?

$$2^{32} * 2^{32} * 2^{32} = 7.92 * 10^{28}$$

Takes more than 2 billion years at a speed of 1 billion tests/second

The Testing Problem

- It can be proven mathematically that it ***impossible*** to prove that a software program is correct

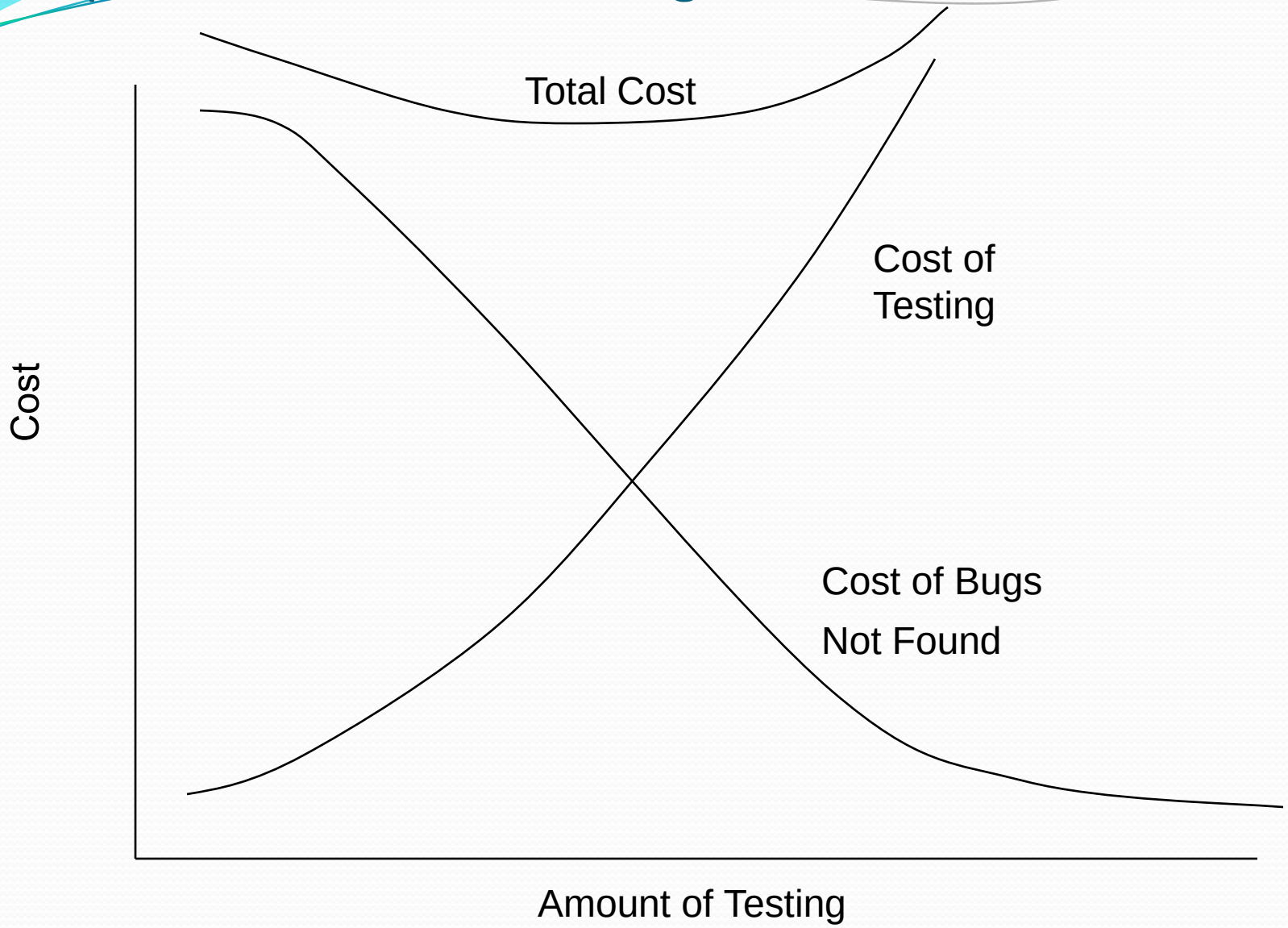
- It is not possible to test all possible states of any non-trivial program

- The financial consequences of shipping software that does not perform as advertised are catastrophic

Software Testing and Risk

- Goal is to minimize the risks of serious bugs with the minimal amount of testing
- Balance the cost of testing against the cost of undetected bugs
- Difficult part is estimating the cost of the bugs
- There will be an optimal point at which the total cost is minimized

Optimal Amount of Testing



No Negative Proof

- Testing cannot prove that bugs do not exist in code
  - Ideally a set of successfully completed tests would strongly predict a probability distribution of bug numbers and severity
  - Hard to explain this concept to people that:
    - Don't understand math
    - Want certainty
    - Feel that the testers aren't fully competent and won't commit themselves

Use heuristics to generate ideas for tests

- *Heuristic – "Greek word, serving to discover" … a rule of thumb*

- *A rule-of-thumb example:*
  - *Test at the boundaries*
  - *Test every error message*
  - *Test configurations different to development environment*
  - *Run tests that are annoying to set up*
  - *Avoid redundant tests*

Testing Techniques

- *All testing techniques focus attention on one or few of the five common dimensions:*
  - ***People** (Testers)*
    - *What will do the testing?*
  - ***Coverage***
    - *What aspects of the program are to be tested?*
  - ***Potential problems***
    - *What types of problems to focus on?*
  - ***Activities***
    - *What tasks will be performed?*
  - ***Evaluation***
    - *How to tell if test succeeded or failed?*

1. Testing Techniques focus on *People*

- *User Testing*
- *Alpha Testing*
- *Beta Testing*
- *Bug bashes*
- *Subject mater expert testing*
- *Paired testing*
- *Use software before selling or releasing*

2. Testing Techniques focus on *Coverage*

- *Function Testing*
- *Functions Integration Testing*
- *Specification-based Testing*
- *Requirements-based Testing*
- *Menu Tour (walk through all menus, dialogs)*
- *Domain Testing (choose variables for function)*
- *Equivalence Class Analysis (test sparingly for equivalence classes)*
- *Best Representative Class Testing*

2. Testing Techniques focus on *Coverage* … *conti*

- *Boundary Values Testing*
- *Input Field Test Catalog / Matrices*
- *Testing different ways to enter a value*
- *Logic Testing*
- *State-based (transition) Testing*
- *Path Testing*
- *Statement and Branch Coverage Testing*
- *Configuration Coverage Testing*
- *Combination (Variables) Testing*

3. Testing Techniques focus on *Problems*

- *Risk-based Testing*
- *Input Constraints*
- *Output Constraints*
- *Computation Constraints*
- *Storage / Memory (or Data) Constraints*

4. Testing Techniques focus on *Activity*

- *Regression Testing*
  - *Bug-fixed regression*
  - *Old bugs regression (e.g. old bug fixes are unfixed by new modification)*
  - *Side-effect regression*
    - *Smoke Testing (a type of Side-effect regression)*
- *Scripted Testing (step-by-step)*

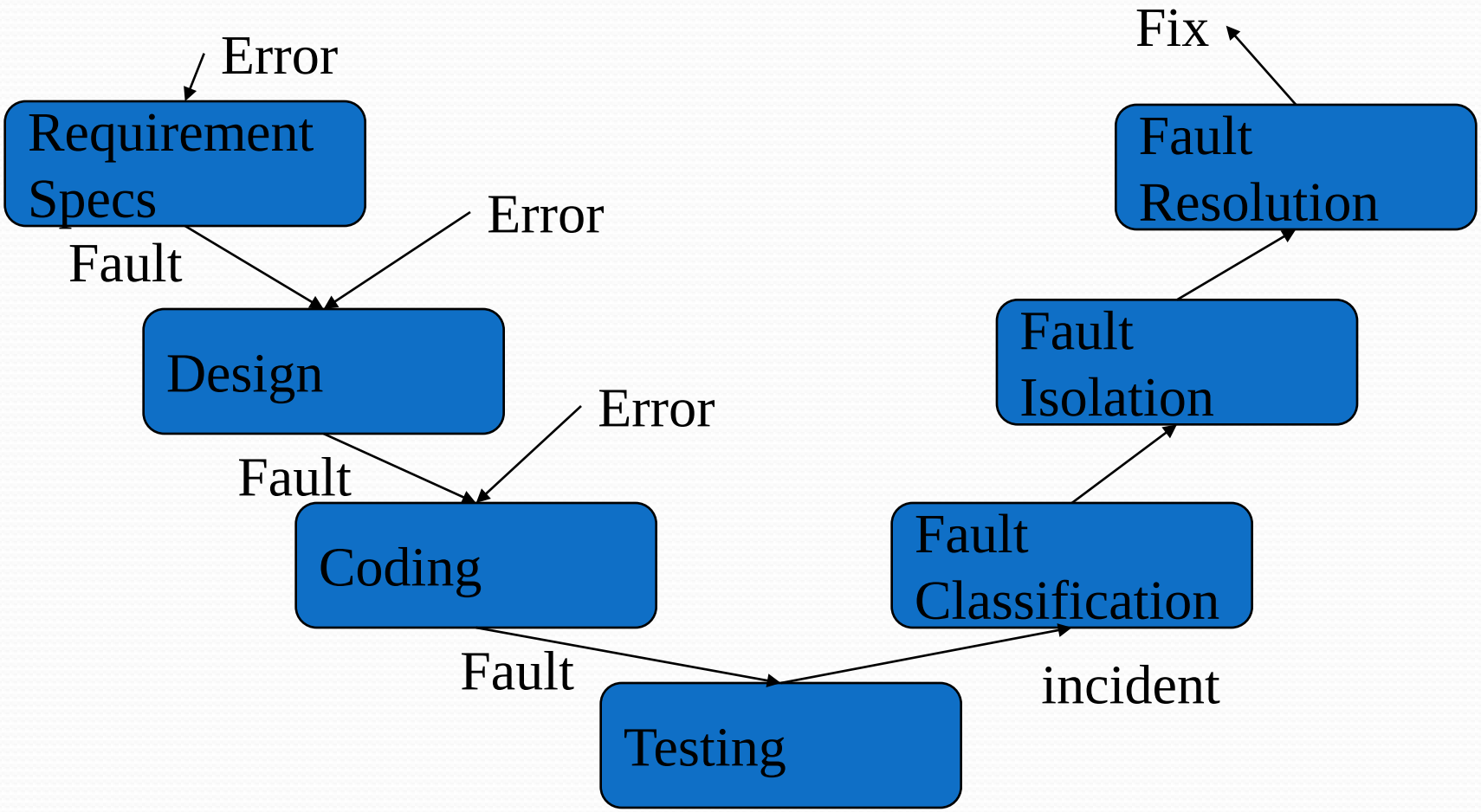
4. Testing Techniques focus on *Activity* … *conti..*

- *Exploratory Testing*
  - *Guerilla Testing (a form of Exploratory)*
- *Scenario Testing (e.g. Use Case Scenario)*
- *Installation Testing*
- *Load Testing*
- *Long Sequence (duration) Testing*
- *Performance Testing*

5. Testing Techniques focus on *Evaluation*

- *Self-verifying data*
- *Compare with past results*
- *Compare with specification*
- *Heuristic consistency*
  - *Consistent with history, image, comparable products, claims, users' expectations, product, purpose.*

Test Life Cycle



Specifications and Testing

- Software testing, cannot be meaningful unless there is a specification for the software.

- can range from a single document to a complex hierarchy of documents.

Object Software Specifications

- Use Cases are roughly equivalent to the Functional Specifications
  - Describe **what** the software should

- The Design Model is similar to the Design Specifications
  - Describes **how** the functionality is to be implemented

Functional vs Structural Testing

- Functional Testing
  - Does the software do what it is supposed to do?
  - Black box testing generally by testers
  - Tied to functional specification
- Structural Testing
  - Does the software work the way it is supposed to work?
  - White or glass box testing often by developers
  - Tied to design specification
- Each can find errors not detectable by the other

Timing of Testing

- Earlier the better
  - Cost of fixing bugs rises as time passes
- Before any code is written
  - Can work through specification
    - Test ideas not code, may not be done by official test team
  - Can start organizing test tools and planning the testing process
- Start testing code as it's written (Agile practice)

Programmer Testing

- Incremental testing
  - Test pieces as they are built
  - Test with drivers and stubs
    - Requires extra coding
  - Test pieces as they're assembled together with other pieces
    - Early integration testing
  - Much easier to discover, reproduce and fix bugs

Big Bang Testing Dilemma

- Integrate and test the whole works the night before shipping
  - Hard to know which bug is causing a failure
  - Many bugs are masked
  - Difficult to understand failures caused by combinations of bugs
  - Potential bad feelings between programmers
  - A way of deferring the day of reckoning

Testing and Evolution

- Cutting corners on testing in the early stages is the surest way to destroy a project
- Core program is not properly tested early
- Nothing shows on the schedule
  - Everything appears to be fine
- Huge amount of untested code at the end
- Important not to press ahead with new code at the expense of bug fixes

Tester's Dilemma

- Should a particular marginal behaviour be reported as a bug?
  - No
    - Reporting the same bug more than once wastes time
    - Really trivial problems should not be reported
  - Yes
    - An unreported bug will likely never be fixed, increasing the chances that a customer will find the bug
- This is another area where tester judgment is critical

The Status Problem

- During development of a large system, it is often difficult to know where the system stands at any given point in time
- Most of the people involved with development have a vested interest in promoting as optimistic a view as possible
- Quality assurance has gradually assumed the role of accurately determining the true status of a project - what actually works

Testing and the Release Game

- By the end of the project testing is on the critical path
- In many organizations QA "officially" has the final say on whether or not the product is fit to be shipped – "Go-NoGo" decision
- If the project is significantly late, there will be tremendous pressure to ship
- All sorts of tricks will be used to get QA to sign off

List Of Common Software Errors

- User interface
- Error handling – incorrect, missed
- Boundary – overflow, underflow
- Calculation errors – wrong algorithm
- Control flow
- Race conditions
- Load conditions
- Hardware error handling
- Version control errors
- Documentation

Common Tests - Examples

- Exploratory testing – guided by experience
- White Box testing – guided by the software
- Black Box testing – guided by specifications
- Unit testing – start with the programmer
- Integration testing – progressive iteration
- System testing – guided by the QA
- Acceptance testing – guided by the user

Some Testing Types

1. Unit Test
2. Smoke Test
3. Acceptance (Functional) Test
4. Regression Test
5. Performance Test
6. Thread Test
7. Multiple Builds Test

1. Unit Test

- As developers program units
- Find as many problems as early in the process as possible
  - major motherhood issue
- Starts with developers/programmers
  - Without programmers dedicated to the production of a quality product, testing isn't going to work
  - Requires management commitment as well
  - Programmers should not depend on testers to find bugs

Unit Test… continue

- Both managers and testers should encourage programmers to produce quality code
- Quality of output must be at least as important as volume of output
- Tool support is required
- Things like Quantify and Purify
- Make quality code part of the culture

2. Smoke Test

- A quick-and-dirty test
-
- major functions of a piece of software work

- Originated in the hardware testing practice of turning on a new piece of hardware for the first time and considering it a success if it does not catch on fire.

3. Acceptance testing

- Sometimes referred to as “functional test” based on user stories
- Run on each new release
- May be published to customers
- Designed to weed out serious problems
- May or may not continue to test program that fails this test
- Customer milestone could be dependant on these

4. Regression Testing

- Done every time the program changes
- Tests that are repeated, generally over and over again
- Extreme case would be:
  - Fix one small bug,
  - Rerun every test that's ever been run on the entire application
- Trick is in deciding how much testing should be done after a change

4. Regression Testing

- After a bug fix, it's mandatory to repeat the test that uncovered the bug
- It's important to test around the bug to see if
  - The actual underlying problem has been solved
  - No new problems have been introduced
- Include the most difficult tests:
  - Previous test that found bugs
  - Test for customer reported bugs
  - Boundary tests

Regression Testing … continue

- It may be important to check the rest of the module involved or maybe even the entire system
- Tend to develop a standard suite of tests that are run on every new release
- Subsets of the main suite that are run on bug fixes
- Regression test suites evolve over time

5. Performance Testing

- To uncover potential performance bottlenecks:
  - Large test data sets are used
  - High volume of transactions
  - Load the system

- Very often not required
- Don’t really know if performance testing is required until a certain amount has been done

6. Thread Testing

- Is based on certain scenarios of operation of the application – mainly events or other forms of input.

- Since much development is based on event handling, this is a sensible way of testing an application.

6. Thread Testing

- In object oriented design, *scenarios* are used to look at how a system will be used so that it can be written accordingly.
- In this way, it is assured that objects involved in the scenario provide acceptable responses to the appropriate data tests.
- Because of the large number of possible scenarios in modern applications, one of the key parts of thread testing is identification of the threads, and designing appropriate tests for each one.

7:Testing Multiple Builds

- Build is a version of the program being tested
- Can have a new build every day
- Test manager or Software Configuration Management can define and manage builds
- Significantly more effort than dealing with one build

7:Testing Multiple Builds

- Organized into a build tree
- Must focus test efforts on appropriate builds
- Not possible to fully test every one
- May be useful for backtracking on really pathological bugs
- Keeps developers working while bugs are fixed

Need for a Model

- Testing is a complex and confusing process
- Huge domain of possible tests that can be done
- Object of model is to illuminate the relative importance of various tasks that could be performed at some particular time in the development cycle