cs304
Software Engineering

TAN, Shin Hwei
陈馨慧
Southern University of Science and Technology

Slides adapted from cs427 (UIUC) and cs409 (SUSTech)

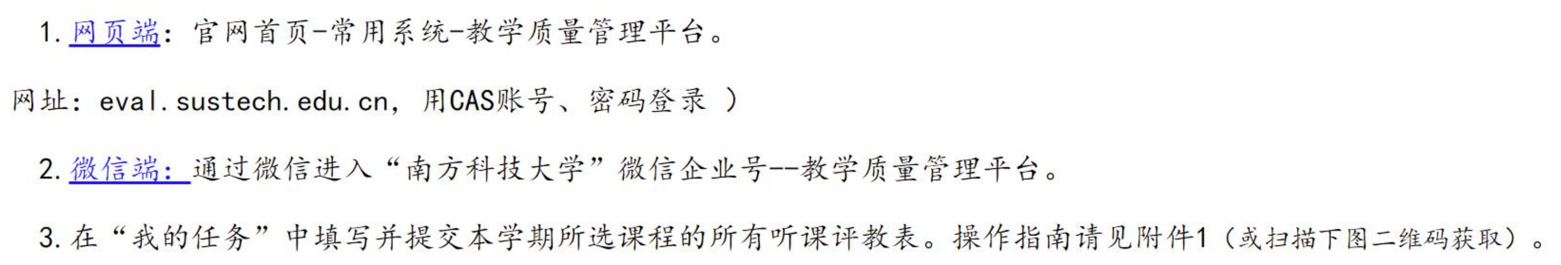
Schedule & Reminder

- Schedule for the remaining weeks:
  - 27 May 2022 All lab exercises due
  - 29 May 2022 Final Presentation
  - Week of 30 May 2022
    - Lecture: Lecture, Best Project Voting & Final Exam Review
    - Lab: Final Presentation
  - 6 June 2022: Bonus deadline for the Code Review Experiment
  - 17 June 2022: Final Exam
- Project Final Presentation Uploaded:
  - due on 29 May 2022, 11.59pm
- All lab exercises due on 27 May 2022,11.59pm
  - coverage lab: https://classroom.github.com/a/rtj7QxND
  - junit lab(Pair programming): https://classroom.github.com/a/0EgnbwO5
  - metrics lab: https://classroom.github.com/a/uD2YOLI2
  - pit-mutation: https://classroom.github.com/a/q9vuleVv
  - reverse engineering lab: https://classroom.github.com/a/uiEoYMrU
  - ui-ci: https://classroom.github.com/a/izTR-pU1
  - security: https://classroom.github.com/a/oORiKfAP

Best Group Voting

- 【腾讯文档】GitHub Projects2022
- https://docs.qq.com/sheet/DWHNWa2NLZWxPQW5j

Teaching Evaluation



1. 网页端：官网首页-常用系统-教学质量管理平台。

网址：eval.sustech.edu.cn，用CAS账号、密码登录 ）

2. 微信端：通过微信进入“南方科技大学”微信企业号--教学质量管理平台。

3. 在“我的任务”中填写并提交本学期所选课程的所有听课评教表。操作指南请见附件1（或扫描下图二维码获取）。

- Go to: http://eval.sustech.edu.cn/ or scan QR code

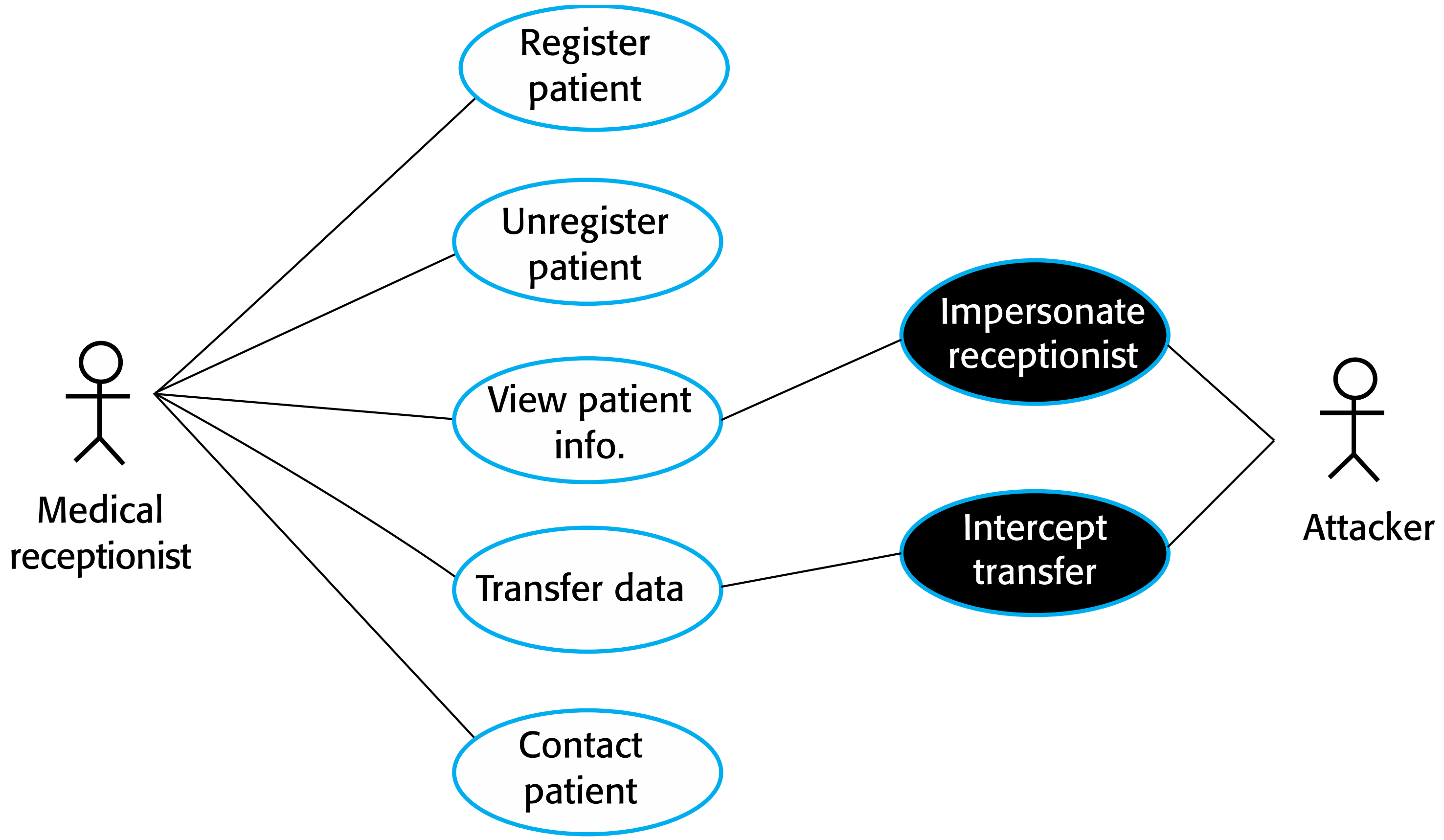
Bonus Opportunity: Code Review Experiment

- Two steps:
  1. We collect your PR link in this QR code
  2. We sent you instruction on how to perform Code Review Experiment

Misuse cases

- Misuse cases are instances of threats to a system
- Interception threats
  - Attacker gains access to an asset
- Interruption threats
  - Attacker makes part of a system unavailable
- Modification threats
  - A system asset is tampered with
- Fabrication threats
  - False information is added to a system

Misuse cases



Mentcare use case – Transfer data

Mentcare system: Transfer data	
Actors	Medical receptionist, Patient records system (PRS)
Description	A receptionist may transfer data from the Mentcare system to a general patient record database that is maintained by a health authority. The information transferred may either be updated personal information (address, phone number, etc.) or a summary of the patient's diagnosis and treatment.
Data	Patient's personal information, treatment summary.
Stimulus	User command issued by medical receptionist.
Response	Confirmation that PRS has been updated.
Comments	The receptionist must have appropriate security permissions to access the patient information and the PRS.

Mentcare misuse case: Intercept transfer

Mentcare system: Intercept transfer (Misuse case)	
Actors	Medical receptionist, Patient records system (PRS), Attacker
Description	A receptionist transfers data from his or her PC to the Mentcare system on the server. An attacker intercepts the data transfer and takes a copy of that data.
Data (assets)	Patient's personal information, treatment summary
Attacks	A network monitor is added to the system and packets from the receptionist to the server are intercepted. A spoof server is set up between the receptionist and the database server so that receptionist believes they are interacting with the real system.

Misuse case: Intercept transfer

Mentcare system: Intercept transfer (Misuse case)	
Mitigations	All networking equipment must be maintained in a locked room. Engineers accessing the equipment must be accredited. All data transfers between the client and server must be encrypted. Certificate-based client-server communication must be used
Requirements	All communications between the client and the server must use the Secure Socket Layer (SSL). The https protocol uses certificate based authentication and encryption.

Secure systems design

Secure systems design

- Security should be designed into a system – it is very difficult to make an insecure system secure after it has been designed or implemented
- Architectural design
  - how do architectural design decisions affect the security of a system?
- Good practice
  - what is accepted good practice when designing secure systems?

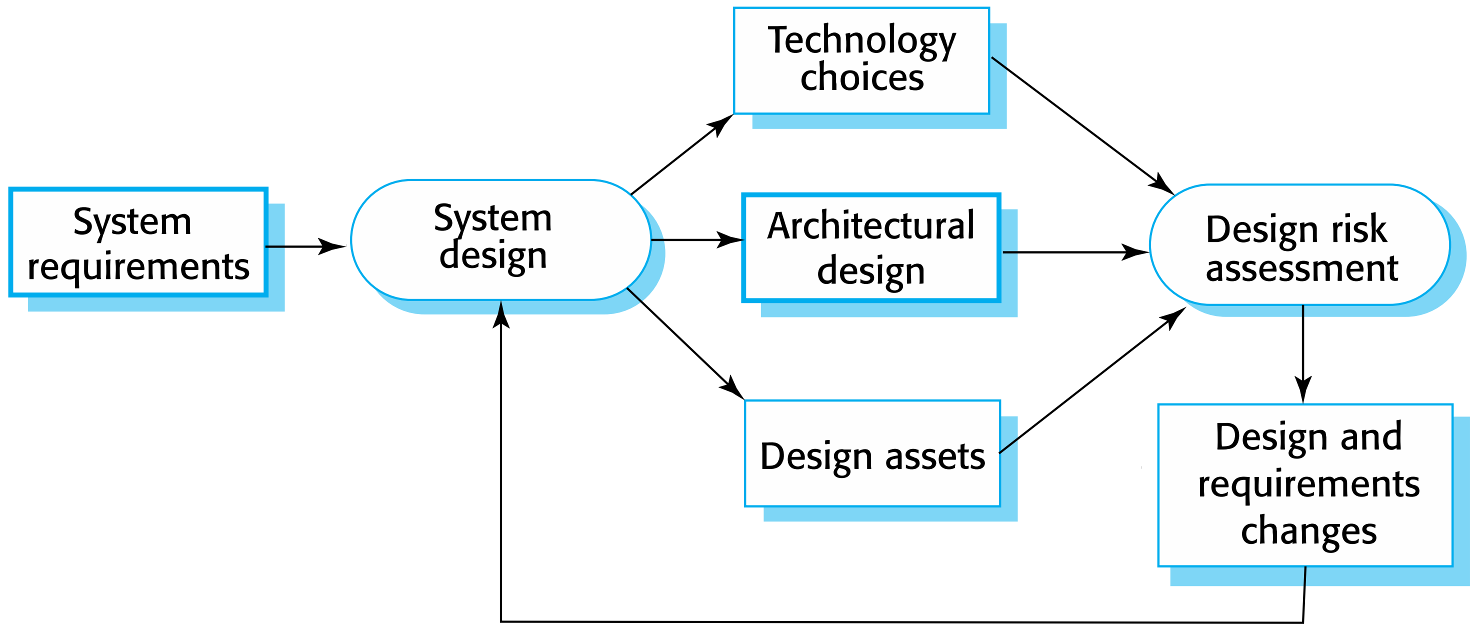
Design compromises

- Adding security features to a system to enhance its security affects other attributes of the system
- Performance
  - Additional security checks slow down a system so its response time or throughput may be affected
- Usability
  - Security measures may require users to remember information or require additional interactions to complete a transaction. This makes the system less usable and can frustrate system users.

Design risk assessment

- Risk assessment while the system is being developed and after it has been deployed
- More information is available - system platform, middleware and the system architecture and data organisation.
- Vulnerabilities that arise from design choices may therefore be identified.

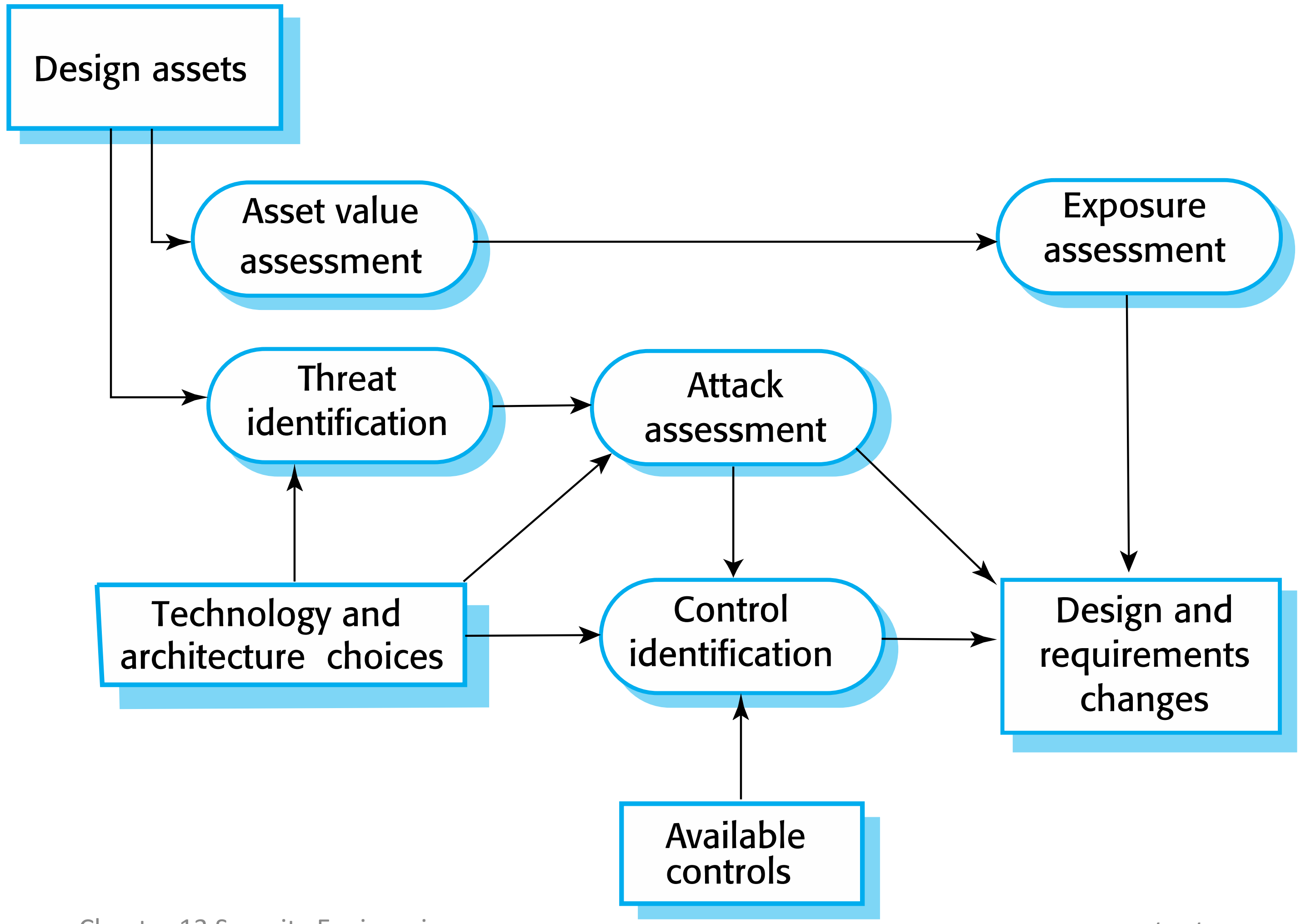
Design and risk assessment



Protection requirements

- Protection requirements may be generated when knowledge of information representation and system distribution
- Separating patient and treatment information limits the amount of information (personal patient data) that needs to be protected
- Maintaining copies of records on a local client protects against denial of service attacks on the server
  - But these may need to be encrypted

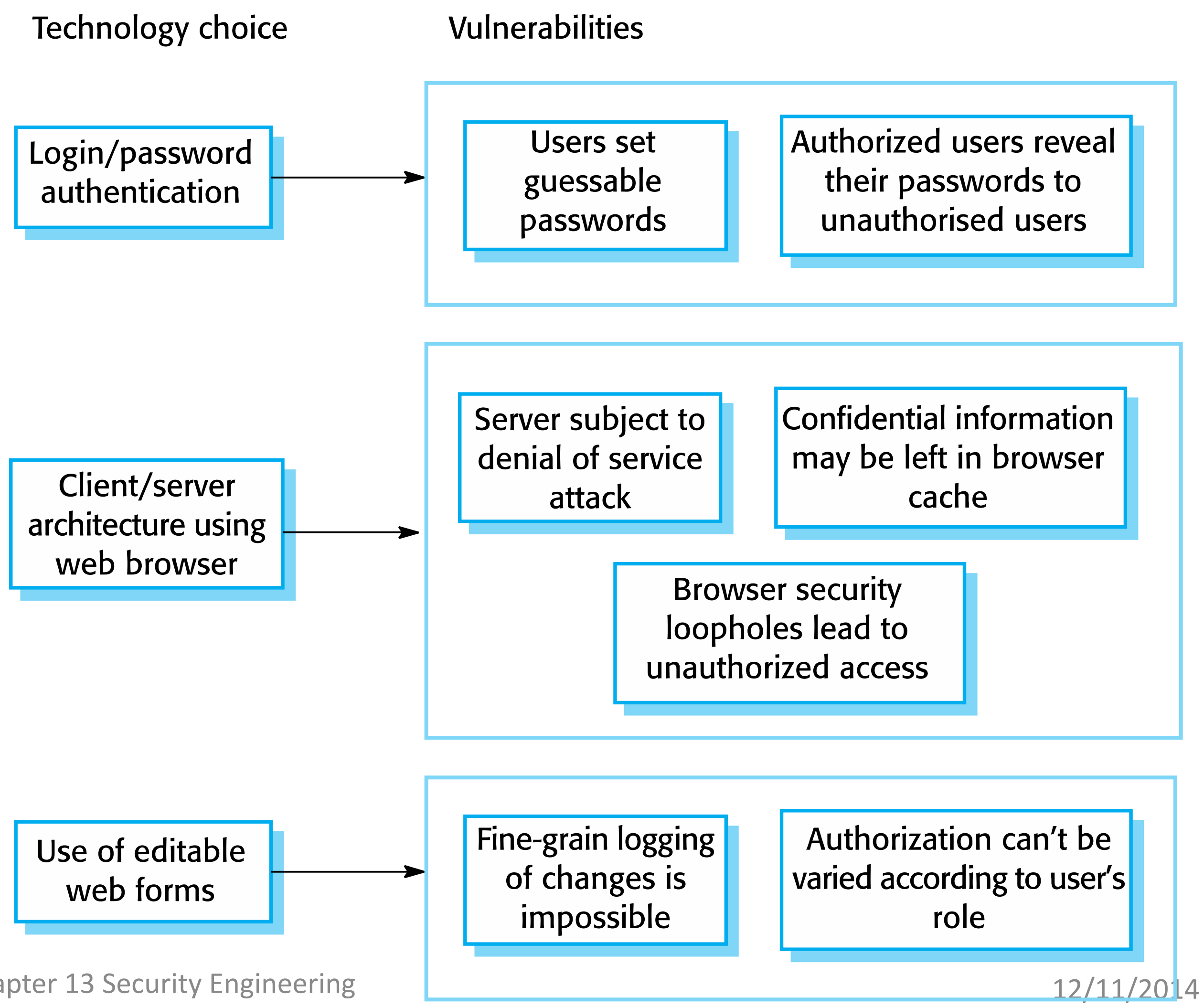
Design risk assessment



Design decisions from use of COTS

- System users authenticated using a name/password combination.
- The system architecture is client-server with clients accessing the system through a standard web browser.
- Information is presented as an editable web form.

Vulnerabilities associated with technology choices


Technology choice
Vulnerabilities
Login/password authentication
Users set guessable passwords
Authorized users reveal their passwords to unauthorised users
Client/server architecture using web browser
Server subject to denial of service attack
Confidential information may be left in browser cache
Browser security loopholes lead to unauthorized access
Use of editable web forms
Fine-grain logging of changes is impossible
Authorization can't be varied according to user's role

Security requirements

- A password checker shall be made available and shall be run daily. Weak passwords shall be reported to system administrators.
- Access to the system shall only be allowed by approved client computers.
- All client computers shall have a single, approved web browser installed by system administrators.

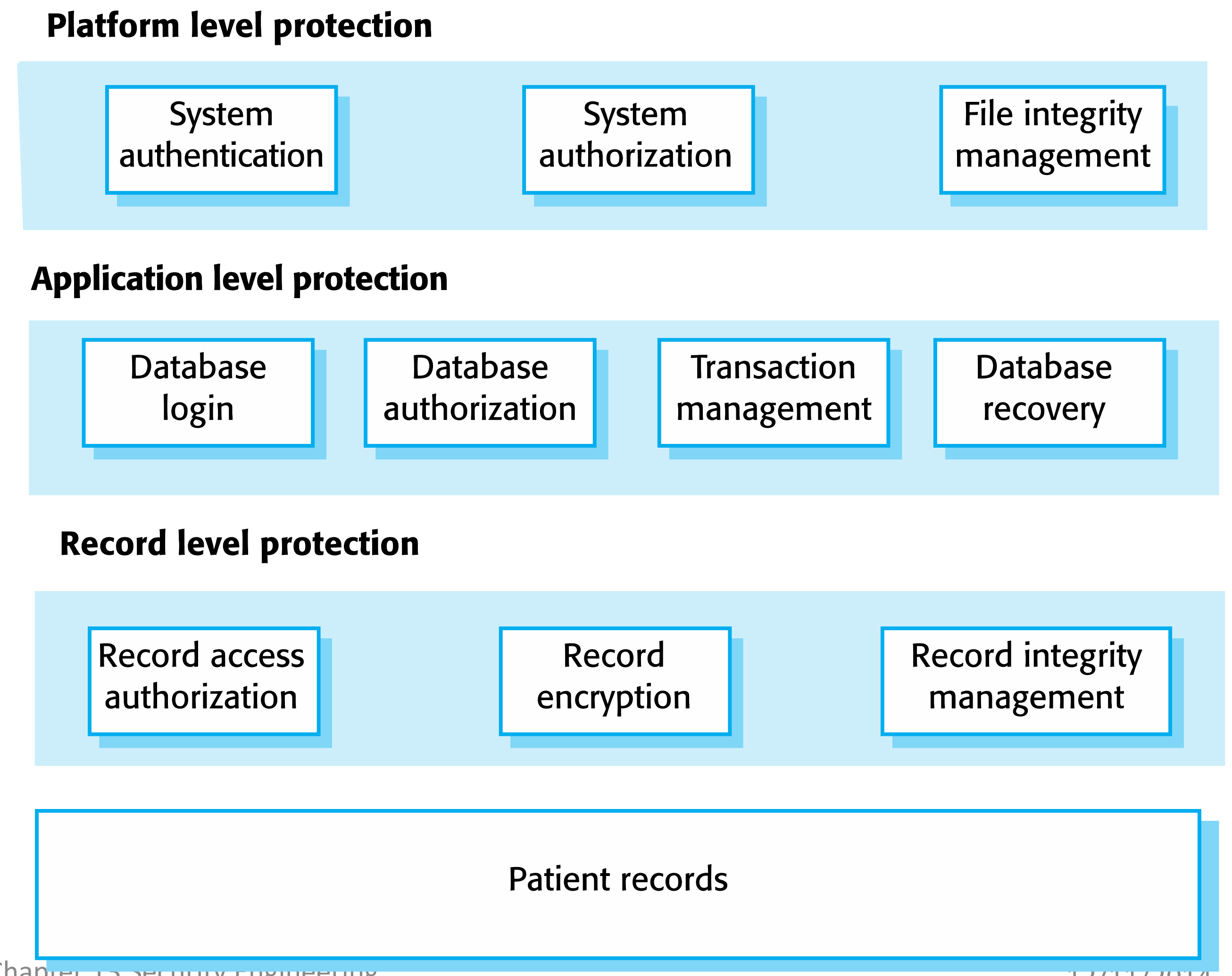
Architectural design

- Two fundamental issues have to be considered when designing an architecture for security.
  - Protection
    - How should the system be organised so that critical assets can be protected against external attack?
  - Distribution
    - How should system assets be distributed so that the effects of a successful attack are minimized?
- These are potentially conflicting
  - If assets are distributed, then they are more expensive to protect. If assets are protected, then usability and performance requirements may be compromised.

Protection

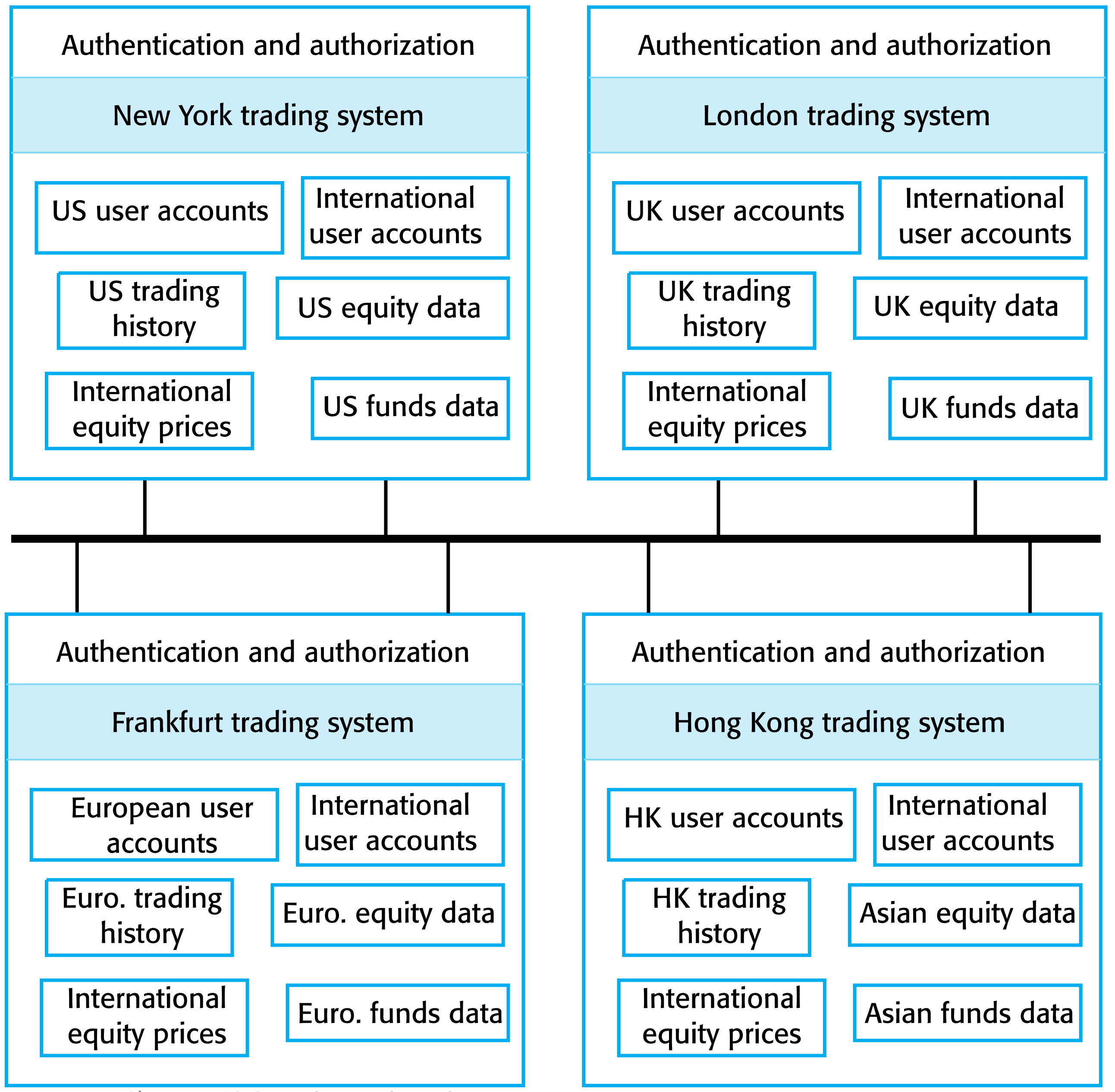
- Platform-level protection
  - Top-level controls on the platform on which a system runs.
- Application-level protection
  - Specific protection mechanisms built into the application itself e.g. additional password protection.
- Record-level protection
  - Protection that is invoked when access to specific information is requested
- These lead to a layered protection architecture

A layered protection architecture



Distribution

- Distributing assets means that attacks on one system do not necessarily lead to complete loss of system service
- Each platform has separate protection features and may be different from other platforms so that they do not share a common vulnerability
- Distribution is particularly important if the risk of denial of service attacks is high



Distributed assets in an equity trading system

Design guidelines for security engineering

- Design guidelines encapsulate good practice in secure systems design
- Design guidelines serve two purposes:
  - They raise awareness of security issues in a software engineering team. Security is considered when design decisions are made.
  - They can be used as the basis of a review checklist that is applied during the system validation process.
- Design guidelines here are applicable during software specification and design

Design guidelines for secure systems engineering

Security guidelines	
Base security decisions on an explicit security policy	
Avoid a single point of failure	
Fail securely	
Balance security and usability	
Log user actions	
Use redundancy and diversity to reduce risk	
Specify the format of all system inputs	
Compartmentalize your assets	
Design for deployment	
Design for recoverability	

Design guidelines 1-3

- Base decisions on an explicit security policy
  - Define a security policy for the organization that sets out the fundamental security requirements that should apply to all organizational systems.
- Avoid a single point of failure
  - Ensure that a security failure can only result when there is more than one failure in security procedures. For example, have password and question-based authentication.
- Fail securely
  - When systems fail, for whatever reason, ensure that sensitive information cannot be accessed by unauthorized users even although normal security procedures are unavailable.

Design guidelines 4-6

- Balance security and usability
  - Try to avoid security procedures that make the system difficult to use. Sometimes you have to accept weaker security to make the system more usable.
- Log user actions
  - Maintain a log of user actions that can be analyzed to discover who did what. If users know about such a log, they are less likely to behave in an irresponsible way.
- Use redundancy and diversity to reduce risk
  - Keep multiple copies of data and use diverse infrastructure so that an infrastructure vulnerability cannot be the single point of failure.

Design guidelines 7-10

- Specify the format of all system inputs
  - If input formats are known then you can check that all inputs are within range so that unexpected inputs don’t cause problems.
- Compartmentalize your assets
  - Organize the system so that assets are in separate areas and users only have access to the information that they need rather than all system information.
- Design for deployment
  - Design the system to avoid deployment problems
- Design for recoverability
  - Design the system to simplify recoverability after a successful attack.

Aspects of secure systems programming

- Vulnerabilities are often language-specific.
  - Array bound checking is automatic in languages like Java so this is not a vulnerability that can be exploited in Java programs.
  - However, millions of programs are written in C and C++ as these allow for the development of more efficient software so simply avoiding the use of these languages is not a realistic option.
- Security vulnerabilities are closely related to program reliability.
  - Programs without array bound checking can crash so actions taken to improve program reliability can also improve system security.

Dependable programming guidelines

Dependable programming guidelines

1. **Limit the visibility of information in a program**
2. **Check all inputs for validity**
3. **Provide a handler for all exceptions**
4. **Minimize the use of error-prone constructs**
5. **Provide restart capabilities**
6. **Check array bounds**
7. **Include timeouts when calling external components**
8. **Name all constants that represent real-world values**

Key points

- Security engineering is concerned with how to develop systems that can resist malicious attacks
- Security threats can be threats to confidentiality, integrity or availability of a system or its data
- Security risk management is concerned with assessing possible losses from attacks and deriving security requirements to minimise losses
- To specify security requirements, you should identify the assets that are to be protected and define how security techniques and technology should be used to protect these assets.

Key points

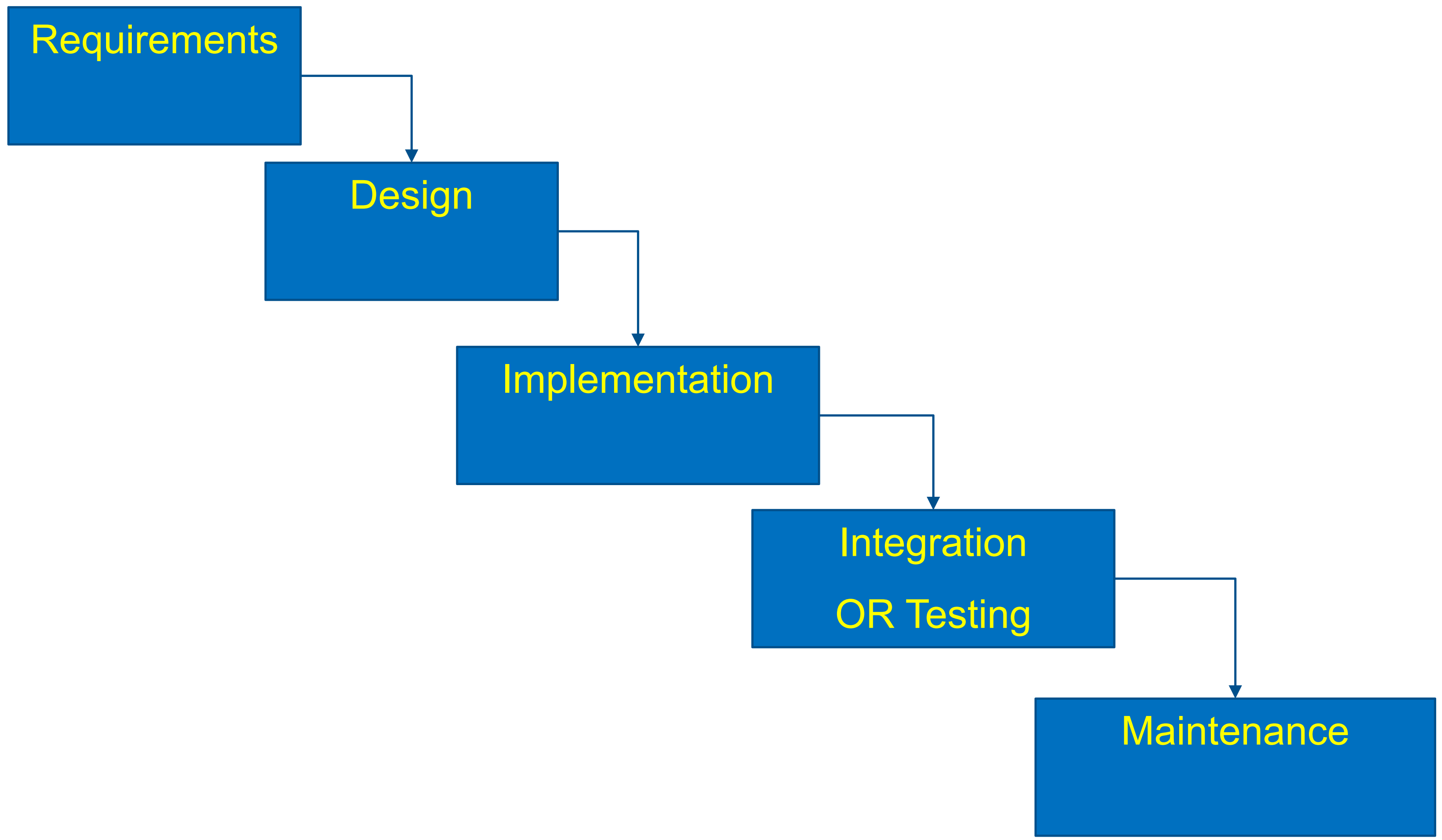
- Key issues when designing a secure systems architecture include organizing the system structure to protect key assets and distributing the system assets to minimize the losses from a successful attack.
- Security design guidelines sensitize system designers to security issues that they may not have considered. They provide a basis for creating security review checklists.
- Security validation is difficult because security requirements state what should not happen in a system, rather than what should. Furthermore, system attackers are intelligent and may have more time to probe for weaknesses than is available for security testing.

Final Exam Review

Summary of SCM

- Four aspects
  - Change control
  - Version control
  - Building
  - Releasing
- Supported by tools
- Requires expertise and oversight
- More important on large projects

waterfall model



eXtreme Programming XP

- Different from the rigid waterfall process
  - Replace it with a collaborative and iterative design process
- Main ideas
  - Don’t write much documentation
    - Working code and tests are the main written product
  - Implement features one by one
  - Release code frequently
  - Work closely with the customer
  - Communicate a lot with team members

XP: Some key practices

- Planning game for requirements
- Test-driven development for design and testing
- Refactoring for design
- Pair programming for development
- Continuous integration for integration

Terminology: Mistake, Fault/Bug, Failure, Error

Programmer makes a mistake

Running the test inputs …

Fault (defect, bug) appears in the program

Fault remains undetected during testing

Program failure occurs during execution (program behaves unexpectedly)

Error: difference between *computed, observed, or measured value or condition* and *true, specified, or theoretically correct value or condition*

Example Fault, Error, Failure

Fault: Should start searching at 0, not 1

```
public static int numZero (int [ ] arr)
{  // Effects: If arr is null throw NullPo          ion
   // else return the number of occurrence          rr
   int count = 0;
   for (int i = 1; i < arr.length; i++)
   {
      if (arr [ i ] ==
      {
         count++;
      }
   }
   return count;
}
```

Test 1
[2, 7, 0]
Expected: 1
Actual: 1

Error exists but no failure
Because expected=actual

Test 2
[0, 2, 7]
Expected: 1
Actual: 0

Error causes failure
Because error propagates to the output

Which tests is correct?

Example 2 is correct!

- Correct method signature should be assertEquals(expected,actual)

Example 1

```
@Test
public void sizeTest() {
 MyStack s = new MyStack ();
 assertEquals (s.size (),0);
}
```

Example 2

```
@Test
public void emptyTest() {
  MyStack s = new MyStack ();
  assertEquals (0, s.size ());
}
```

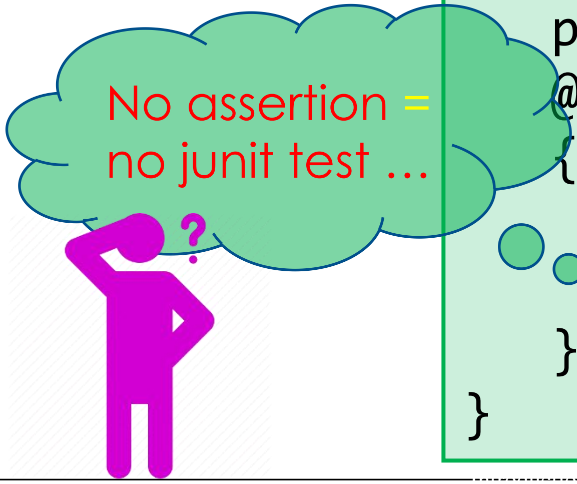
Example JUnit Test

```
public class Calc
{
public long add (int a, int b)
    {
        return a + b;
    }
}
```

Expected result

The test

```
import org.junit.Test;
import static org.junit.Assert.*;
public class calcTest
{
   private Calc calc;
   @Test public void testAdd()
   {
       calc = new Calc ();
       assertEquals((long) 5, calc.add(2,3));
   }
}
```



Testing static versus non-static methods

Non-static Method

```
@Test
public void nonStaticTest() {
 MyStack s = new MyStack ();
 assertEquals (s.size (),0);
}
```

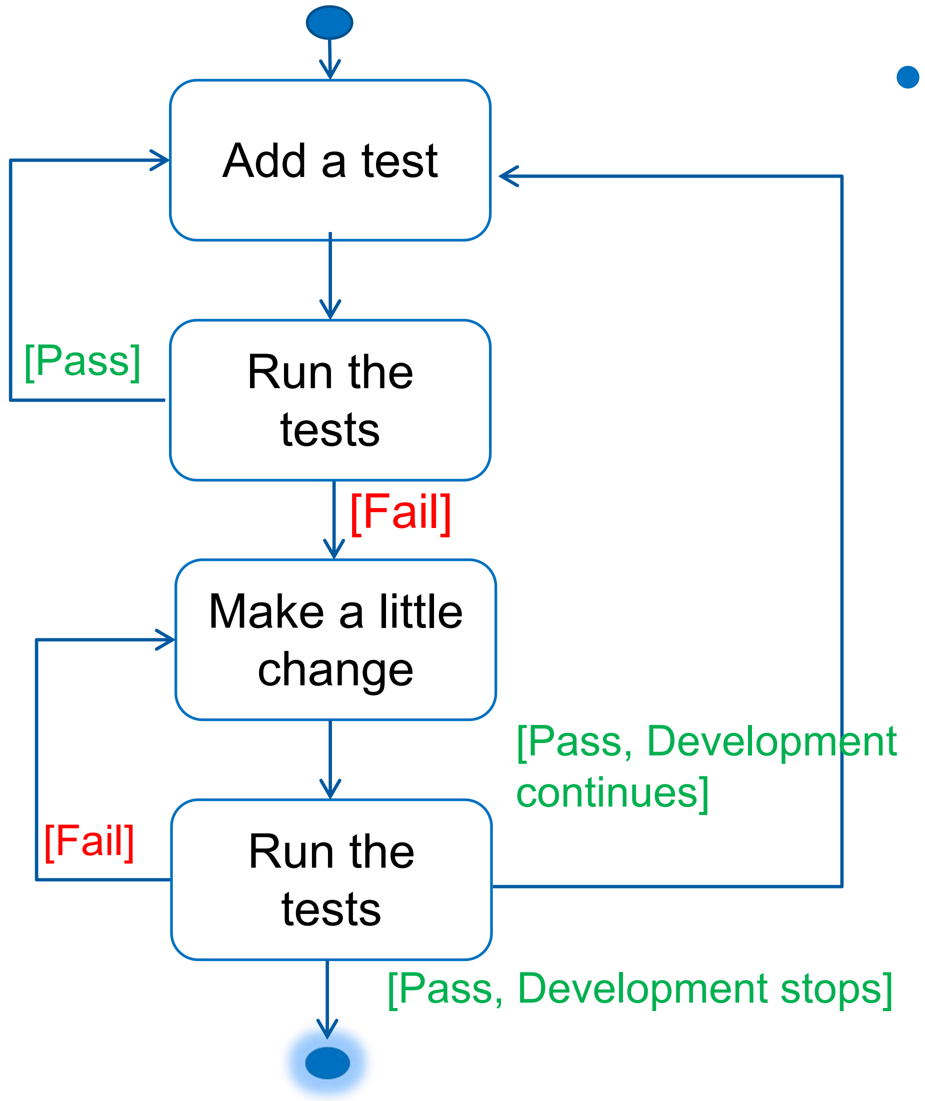
Needs to create a new object

Static Method

```
@Test
public void staticTest() {
assertEquals(Character.Unicode
Block.BASIC_LATIN,
Character.UnicodeBlock.of((cha
r)0x0));
}
```

Directly call the method *of* with the class name

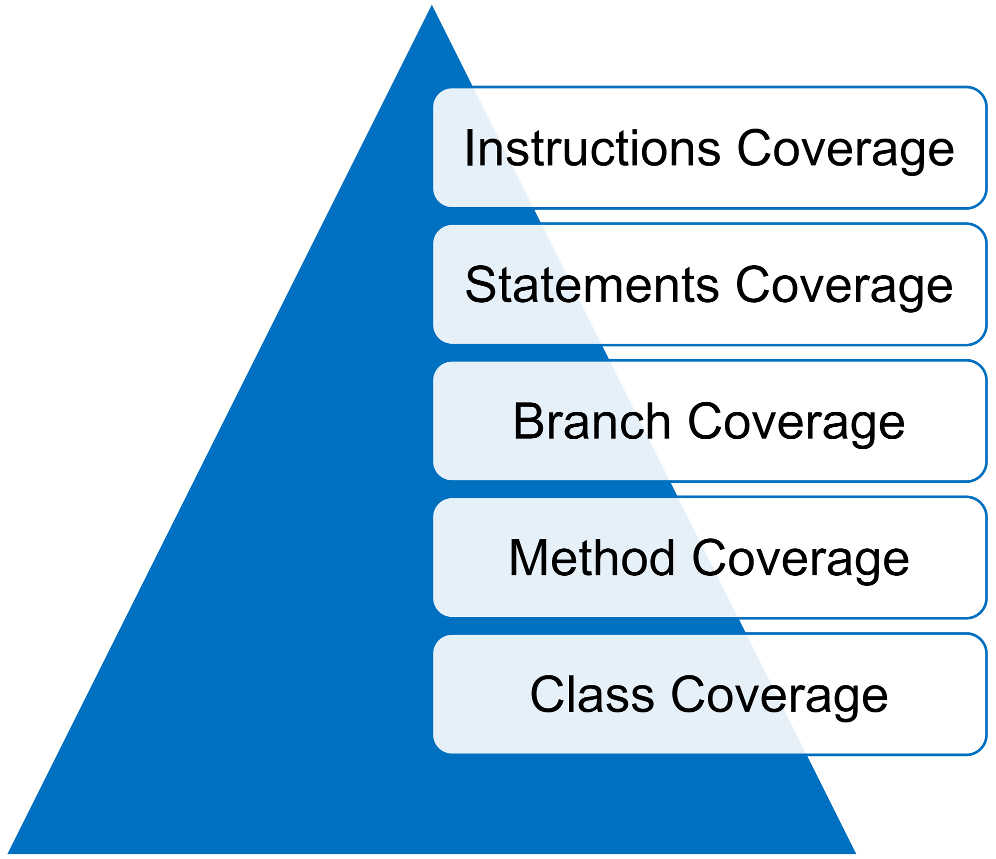
Steps in Test Driven Development (TDD)



- The iterative process
  - Quickly add a test.
  - Run all tests and see the new one fail.
  - Make a little change to code.
  - Run all tests and see them all succeed.
  - Refactor to remove duplication.

Coverage Criteria

- To measure what percentage of code has been exercised by a test suite, one or more coverage criteria are used



Is there any tool that helps you increase coverage fast by generating JUnit tests automatically?

- Yes, there are several popular open-source test generations
  - Randoop
  - Evosuite

Write Javadoc comment for the method below

```
public static String printNum(int i)
        {
                String num="";
                if(i%3==0)
                        num+="Fizz";
                if(i%5==0)
                        num+="Buzz";
                if(num.length()==0)
                        num=Integer.toString(i);
                System.out.println(num);
                return num;
        }
```

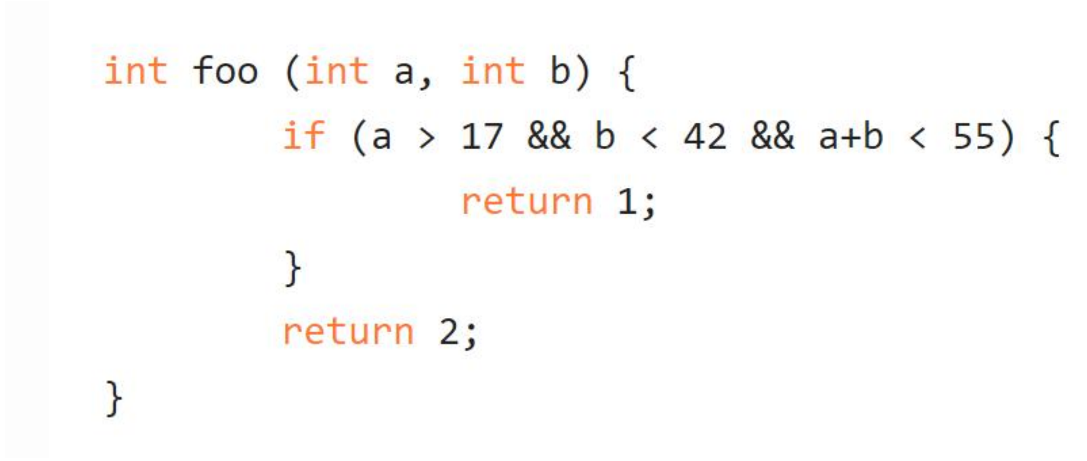
Cyclomatic Complexity

- A measure of logical complexity
- Tells how many tests are needed to execute every statement of program
- **Calculated by developing a Control Flow Graph of the code that measures the number of linearly-independent paths through a program module.**

=Number of branches (e.g, `if`, `while`, `for`) + 1

Different tools tells you different values of Cyclomatic Complexity

- Original paper is not clear about how to derive the control flow graph
  - different implementations gives different values for the same code.
  - For example, the following code is reported with complexity 2 by the Eclipse Metrics Plugin, with 4 by GMetrics, and with complexity 5 by SonarQube:

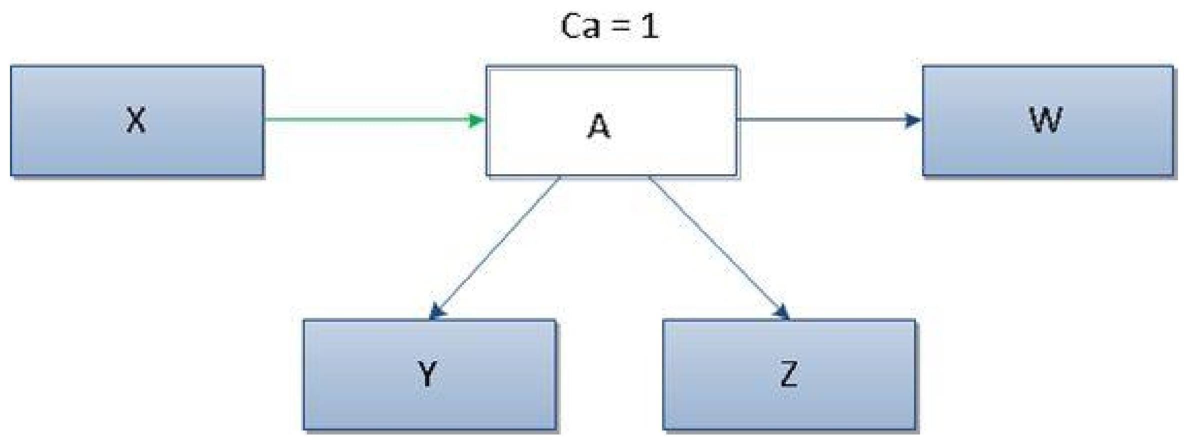


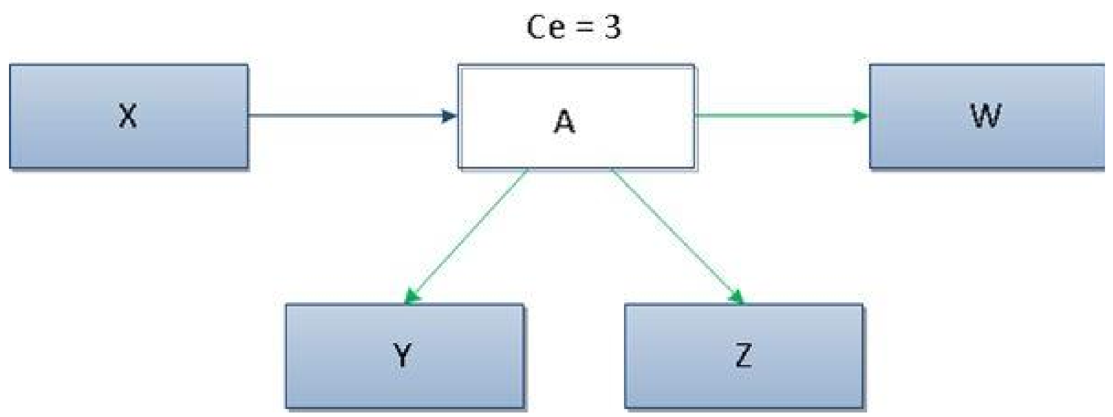
```
int foo (int a, int b) {
        if (a > 17 && b < 42 && a+b < 55) {
                return 1;
        }
        return 2;
}
```

Martin’s Coupling Metric

- Ca : Afferent coupling: measure incoming dependencies
- Ce : Efferent coupling: measure outgoing dependecies

Instability = Ce / (Ca + Ce)

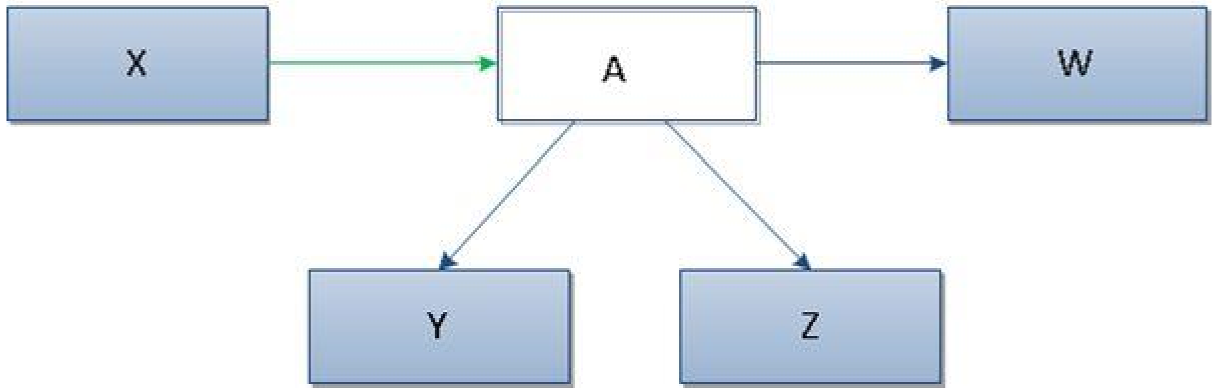




Instability

$$I = \frac{Ce}{Ce+Ca}$$

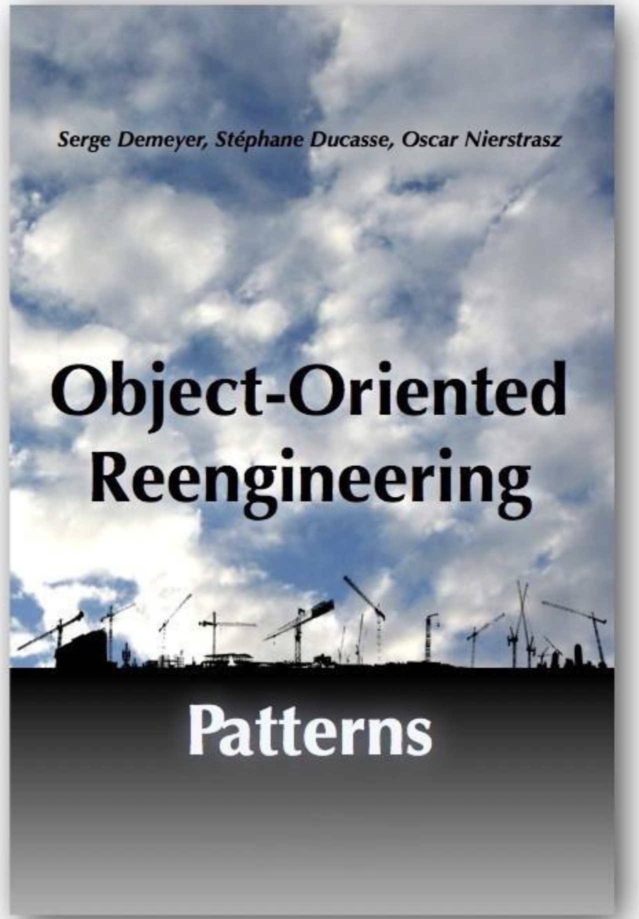
What is the instability of A?



What is reverse engineering?

- Discovering design of an artifact
  - From lower level to higher level; for example:
    - Given binary, discover source code
    - Given code, discover specification & design rationale
- Layman: trying to understand how the system works
- When are you done?
  - Learn enough to:
    - Change it, or
    - Replace it, or
    - Write a book about it

OORP book

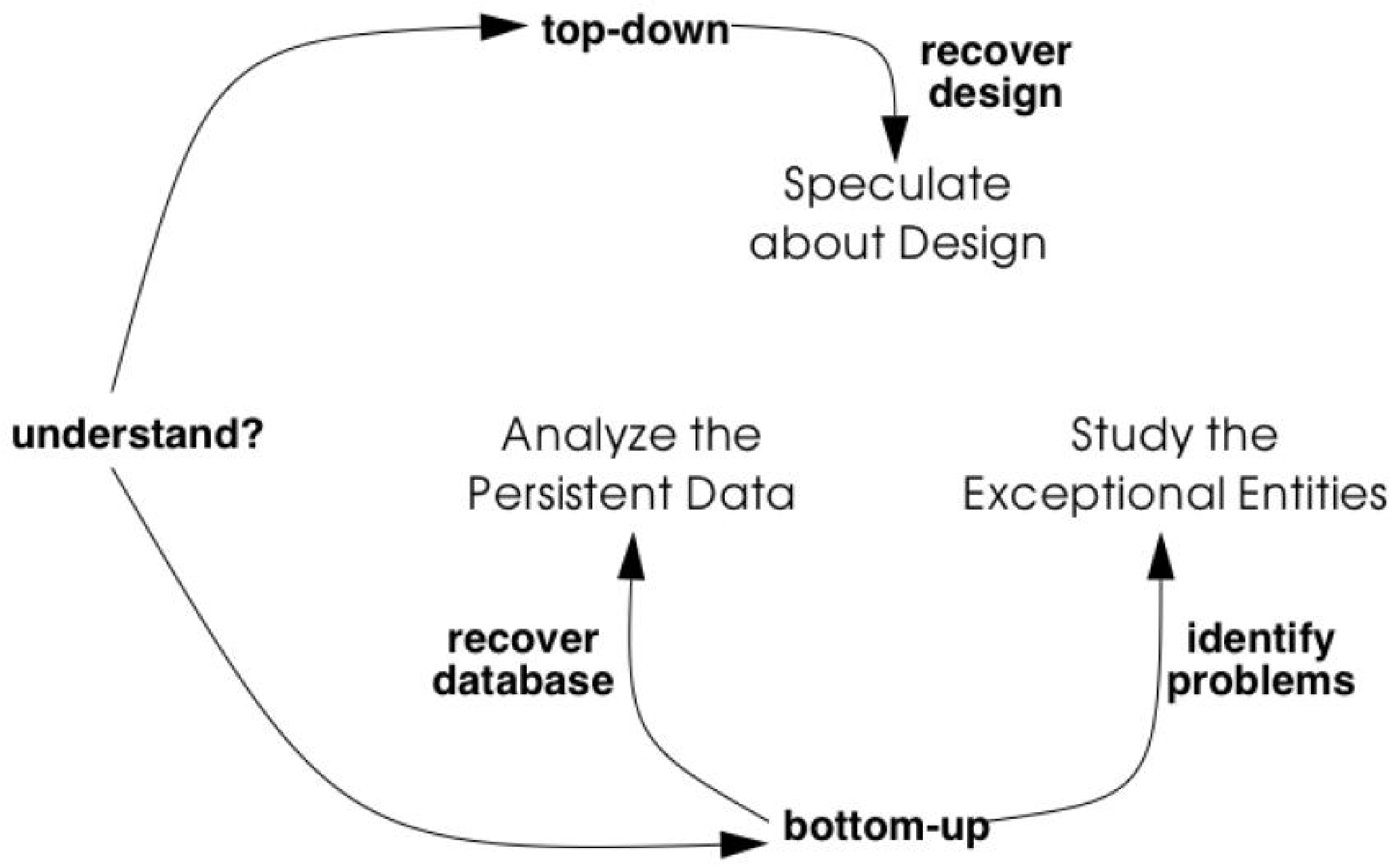

Serge Demeyer, Stéphane Ducasse, Oscar Nierstrasz
Object-Oriented Reengineering
Patterns

- Patterns
  - Expert solutions to common problems
  - Document best practices
  - Should not apply blindly
  - Described using names, context, forces
- You may have heard of design patterns

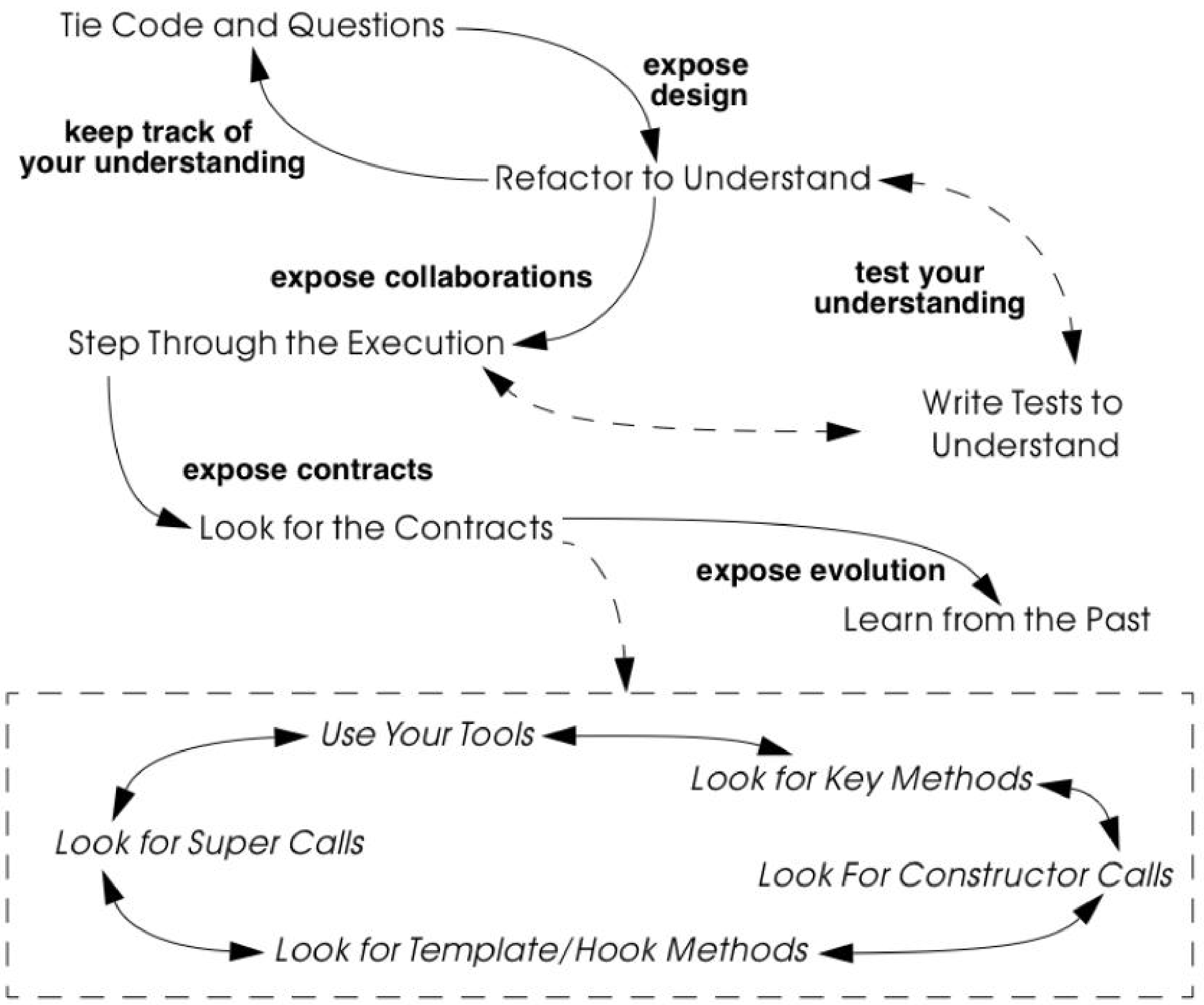
Reverse engineering patterns
Read All Code in 1 Hour

- Intent: assess a software system via a brief but intensive code review
- Problem: system is large/varied, unfamiliar
- Solution: read code & document findings (important entities, “code smells”, tests)
- Hints: what to look for?
  - Coding styles/idioms, tests (system and unit)
  - Abstract classes or classes high in hierarchy
  - Large entities, comments

Patterns for Initial understanding



Patterns For Detailed model capture



What is testing?

- Dynamic Analysis
- Static Analysis

Tools for Static Analysis

- Checkstyle
- PMD
- FindBugs

10 rules of Good UI Design

1. Make Everything the User Needs Readily Accessible
2. Be Consistent
3. Be Clear
4. Give Feedback
5. Use Recognition, Not Recall
6. Choose How People Will Interact First
7. Follow Design Standards
8. Elemental Hierarchy Matters
9. Keep Things Simple
10. Keep Your Users Free & In Control

https://www.elegantthemes.com/blog/resources/10-rules-of-good-ui-design-to-follow-on-every-web-design-project

Which comment is better?

A. Finds the first blank in the string.
B. Find the first blank in the string.
C. This method finds the first blank in the string.
D. Method findBlank(String s) finds the first blank in the string.

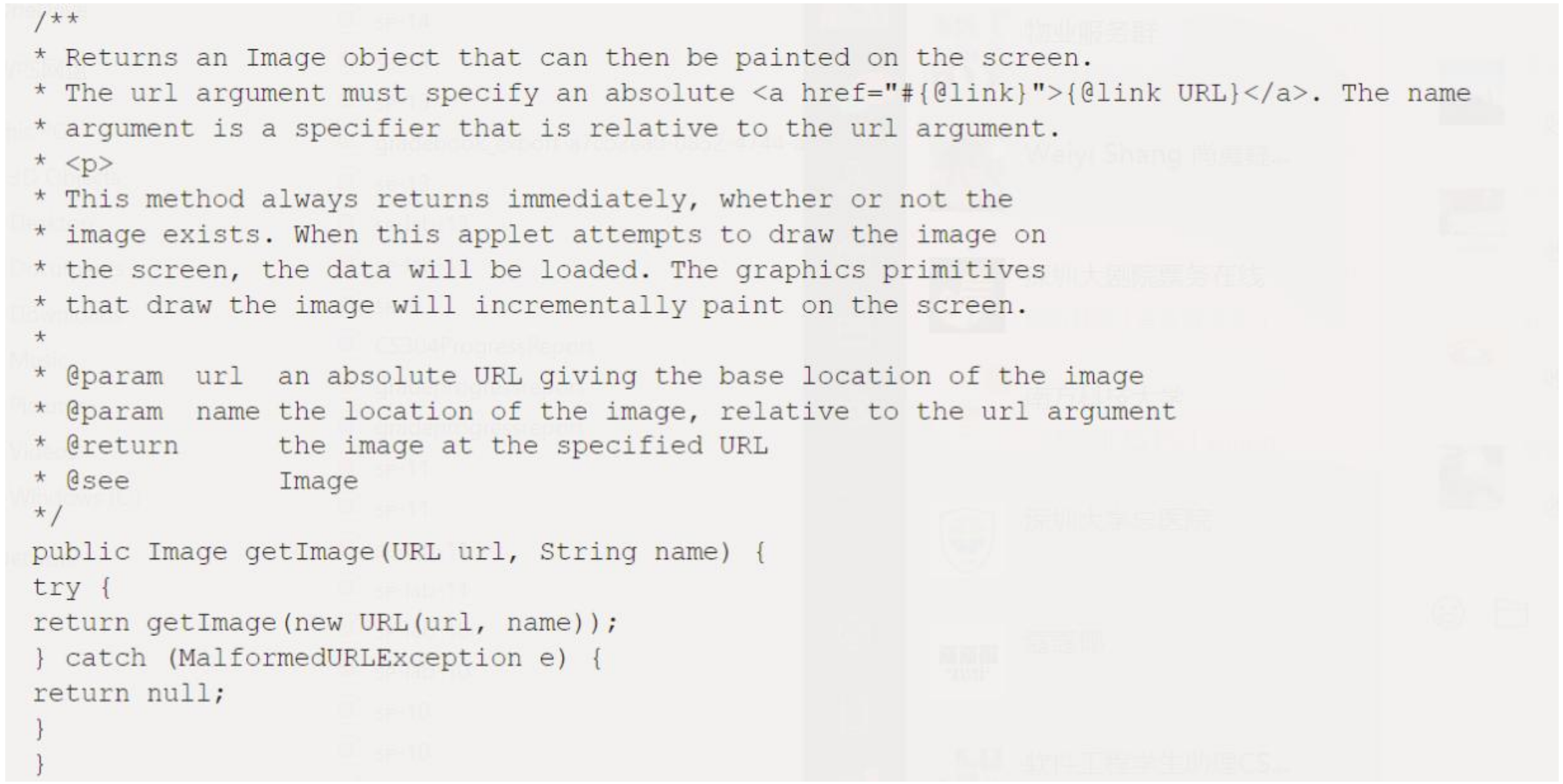
```
 int findBlock(String s){

…
}
```

Rules for writing summaries

- The *first sentence* should summarize the purpose of the element
- For methods, omit the subject and write in the third-person narrative form
  - Good: Finds the first blank in the string.
  - Not as good: Find the first blank in the string.
  - Bad: This method finds the first blank in the string.
  - Worse: Method findBlank(String s) finds the first blank in the string.
- Use the word this rather than “the” when referring to instances of the current class (for example, Multiplies this fraction…)
- Do not add parentheses to a method or constructor name unless you want to specify a particular signature
- Keep comments up to date!

Format of JavaDoc



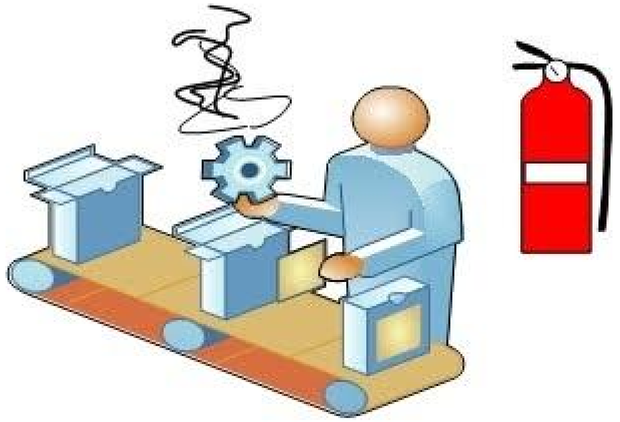
```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute <a href="#{@link}">{@link URL}</a>. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the base location of the image
 * @param  name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
public Image getImage(URL url, String name) {
try {
return getImage(new URL(url, name));
} catch (MalformedURLException e) {
return null;
}
}
```

What is DevOps?

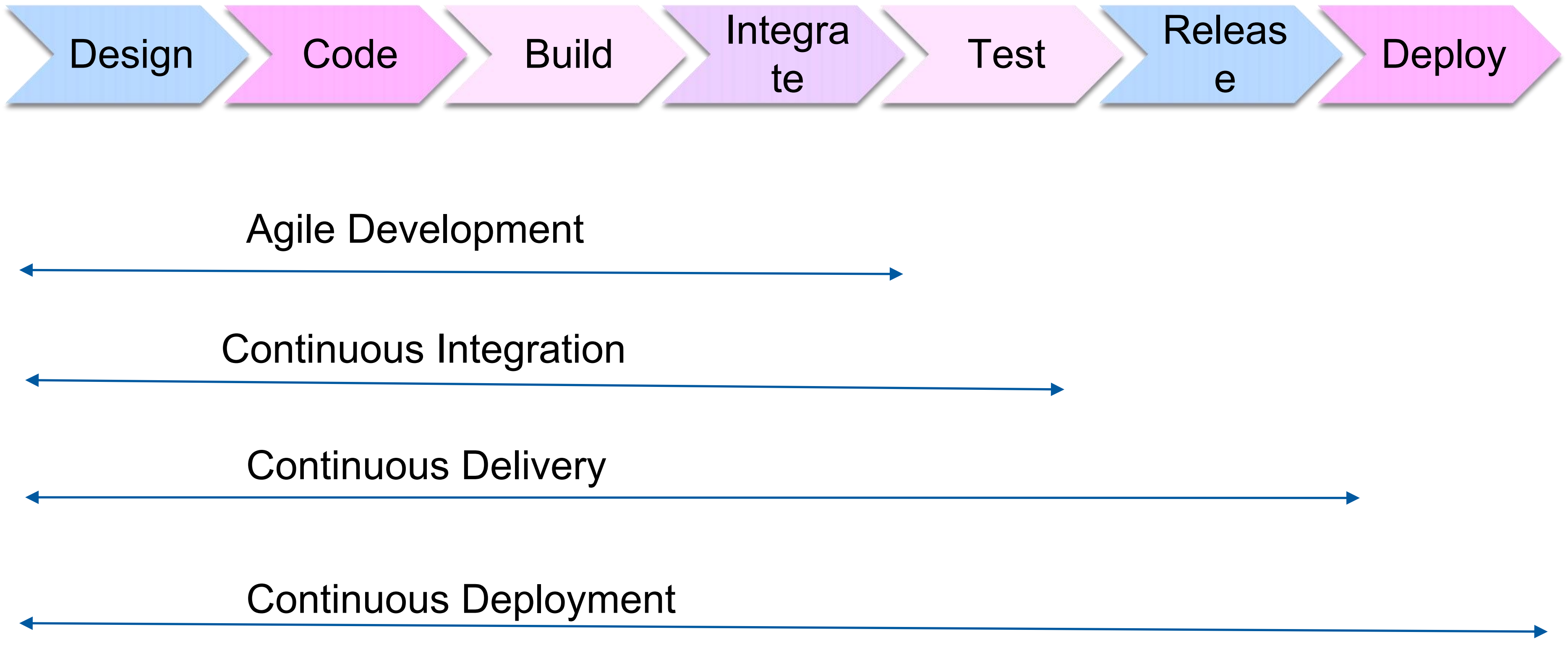
A. Design + IT Operations
B. Design + Optimization
C. Development + IT Operations
D. Development + Optimization

What is Smoke Test?

- A quick set of tests run on the daily build.
  - Cover most important functionalities of the software but NOT exhaustive
  - Check whether code catches fire or “smoke” (breaks)
  - Expose integration problems earlier



Continuous Integration & Continuous Deployment


Design
Code
Build
Integra te
Test
Releas e
Deploy
Agile Development
Continuous Integration
Continuous Delivery
Continuous Deployment