cs304
Software Engineering

TAN, Shin Hwei

陈馨慧

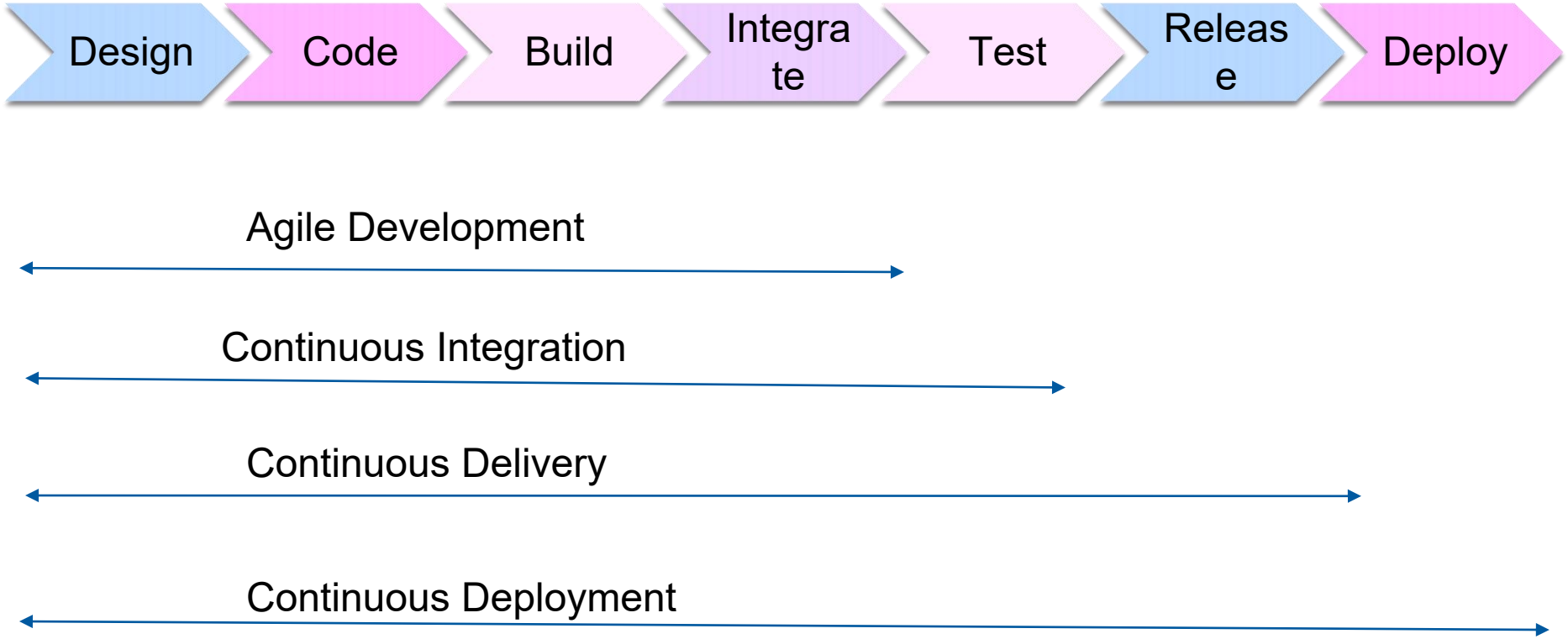
Southern University of Science and Technology

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**

Recap: Continuous Integration & Continuous Deployment



Deployment strategies

Strategy 1: Zero-downtime deployment

1. Deploy version 1 of your service
2. Migrate your database to a new version
3. Deploy version 2 of your service in parallel to the version 1
4. If version 2 works fine, bring down version 1
5. Deployment Complete!

Strategy 2: Blue-green deployment

1. Maintain two copies of your production environment (“blue” and “green”)
2. Route all traffic to the blue environment by mapping production URLs to it
3. Deploy and test any changes to the application in the green environment
4. “Flip the switch”: Map URLs onto green & unmap them from blue.

Zero-downtime & Blue-green Deployment

- Advantage:
  - No outage/shut down
  - User can still use the application without downtime
- Disadvantages:
  - Needs to maintain 2 copies
  - Double the efforts required to support multiple copies
  - Migration of database may not be backward compatible

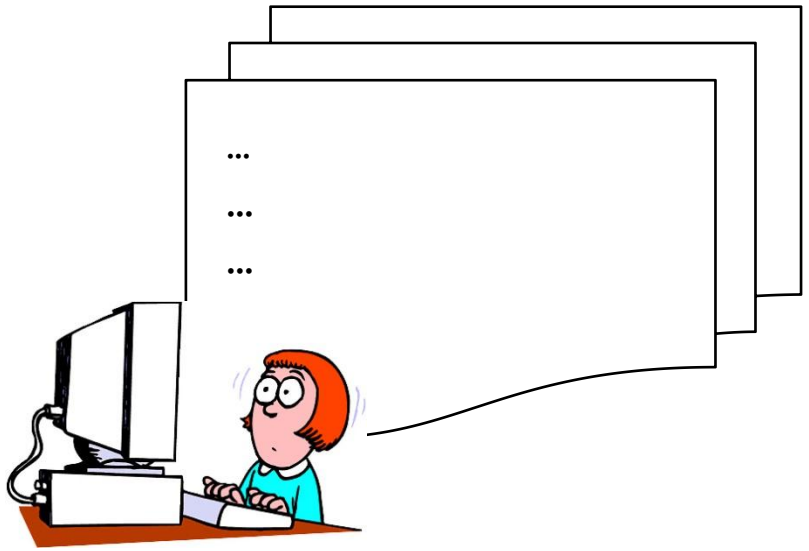
Safer Strategy: Shut down→Migrate → Deploy

Question:

What is difference between **Continuous Integration** and **Continuous Delivery** ?

Continuous Delivery requires **automated testing** before release.

Regression Testing



Test 1	✗	✓
Test 2	✓	✓
Test 3	✓	✗

Regression!

```
+...
while (out > line)
   out;
- ...
```

Test 1	✓
Test 2	✓
Test 3	✓

Regression Fixed!

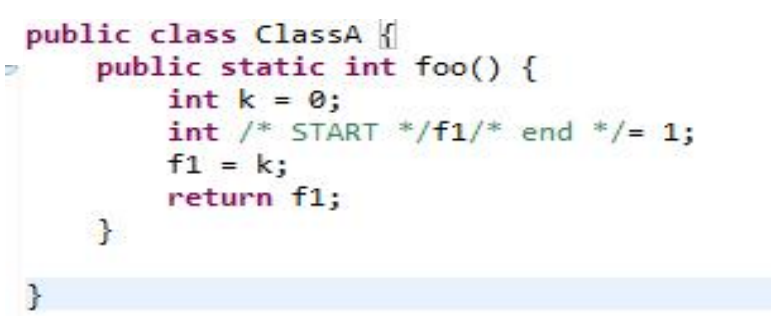
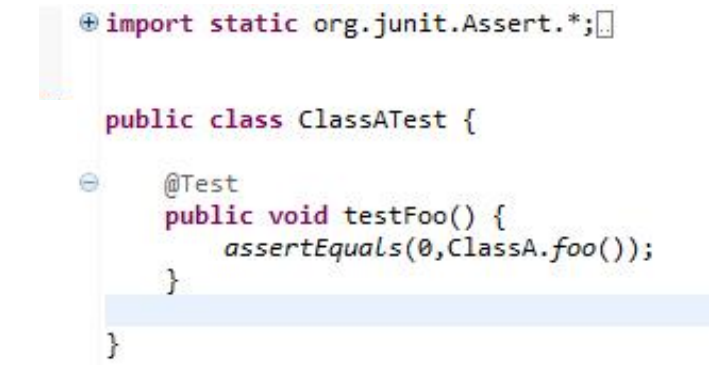
What is Regression?

- Software undergoes changes
- But changes can both
  - improve software, adding feature and fixing bugs
  - break software, introducing new bugs
- We call such “breaking changes” regressions

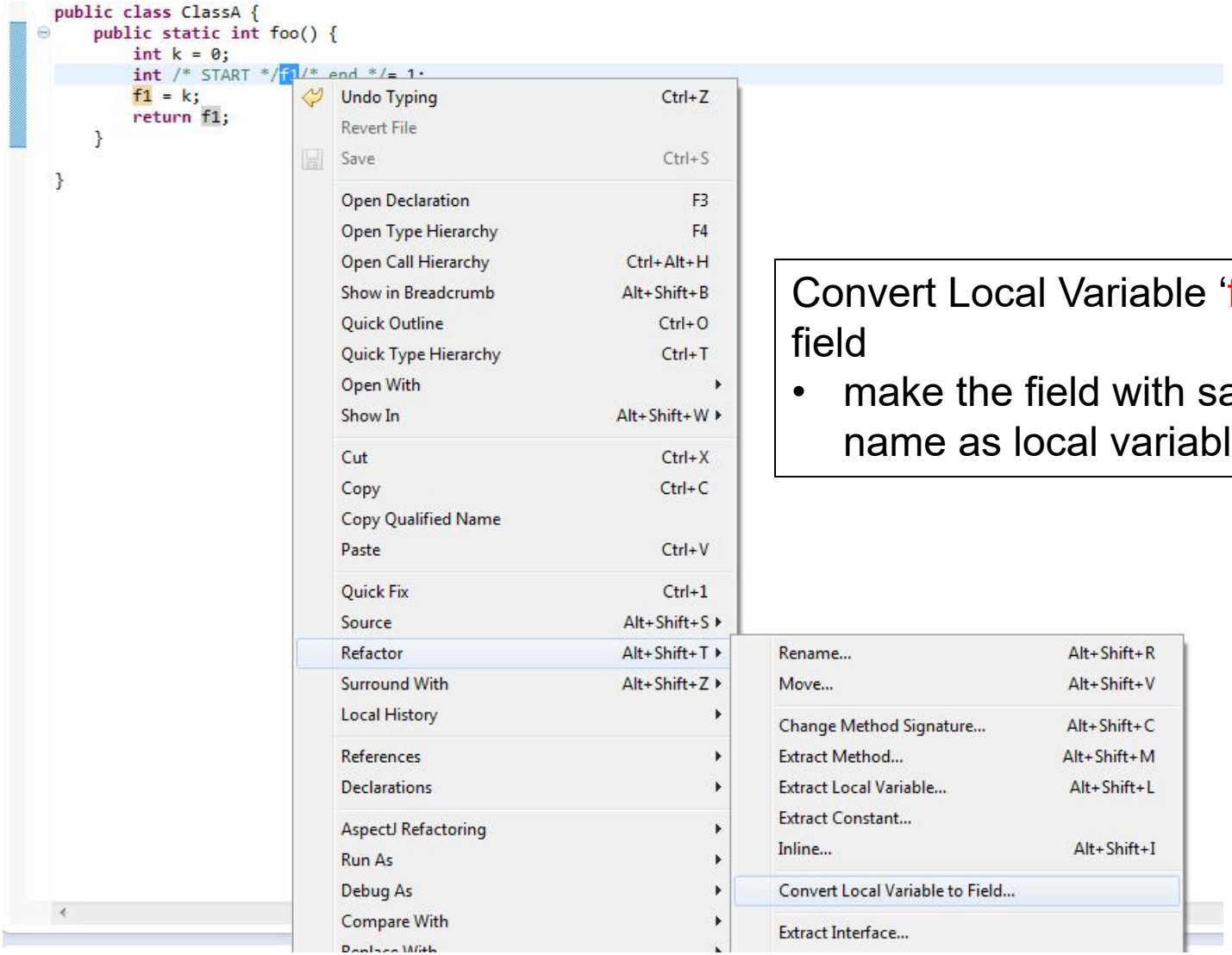
What is Regression testing?

- Testing that are performed to ensure that changes made does not break existing functionality
- It means re-running test cases from existing test suites to ensure that software changes do not introduce new faults

Example:
How can regression test helps in Refactoring?

Class		• foo() returns 0 before refactoring
Test		• Existing test serves as regression test • Run regression test before refactoring
Test Result		• All test passes!

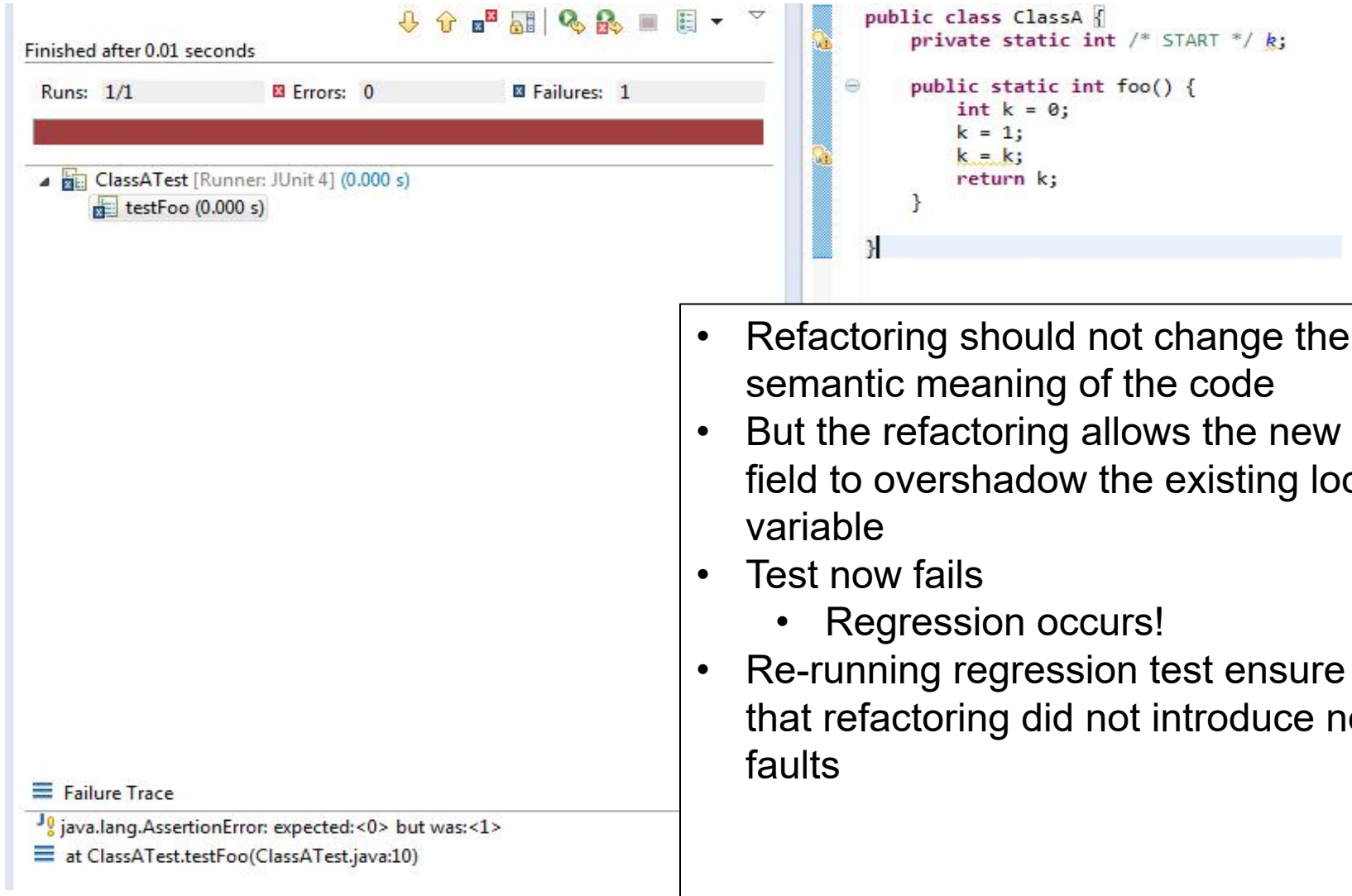
Perform Refactoring: Convert Local Variable to Field



Convert Local Variable ‘f1’ to field

- make the field with same name as local variable ‘k’

After refactoring



- Refactoring should not change the semantic meaning of the code
- But the refactoring allows the new field to overshadow the existing local variable
- Test now fails
  - Regression occurs!
- Re-running regression test ensure that refactoring did not introduce new faults

How to fix regression bug?

- Do a diff against the program version that worked
  - Use version control system to see the changes between version
    - E.g., svn diff, git diff

- Find which changes are related to the tests failure
  - Use debugger to locate the failure inducing changes
  - E.g., Eclipse Debugger

- Run tests again to see that the fix passes all tests
- Add new tests to reflect the changes

If you are interested in automated approach, read:
http://www.shinhwei.com/relifix.pdf

The State of Continuous Integration Testing @Google

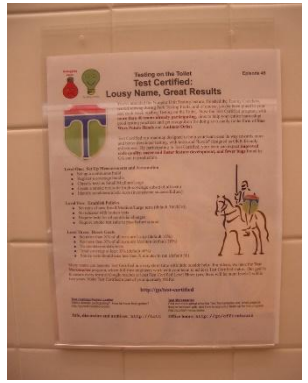
Adapted from https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45880.pdf

Testing Scale at Google

- 4.2 million individual tests running continuously
  - Testing runs before and after code submission
- 150 million test executions / day (averaging 35 runs / test / day)
- Distributed using internal version of bazel.io to a large compute farm
- Almost all testing is automated - no time for Quality Assurance
- 13,000+ individual project teams - all submitting to one branch
- Drives continuous delivery for Google
- 99% of all test executions pass

Testing Culture @Google

- ~10 Years of testing culture promoting hand-curated automated testing
  - Testing on the toilet and Google testing blog started in 2007
  - GTAC conference since 2006 to share best practices across the industry
  - Part of our new hire orientation program
- SETI role
  - Usually 1-2 SETI engineers / 8-10 person team
  - Develop test infrastructure to enable testing
- Engineers are expected to write automated tests for their submissions
- Limited experimentation with model-based / automated testing
  - Fuzzing, UI waltkthroughs, Mutation testing, etc.
  - Not a large fraction of overall testing



Example: Testing in the toilet

Can you tell if this test is correct?

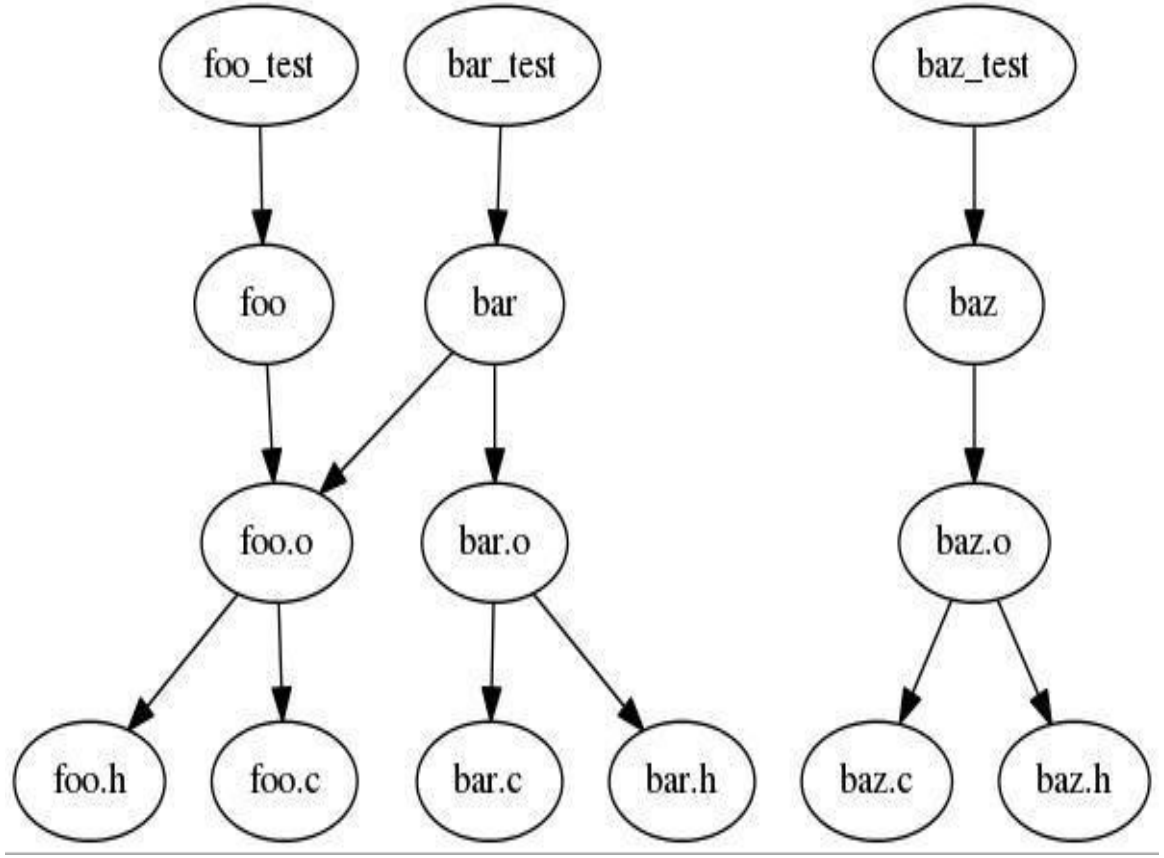
```
208: @Test public void testIncrement_existingKey() {
209:   assertEquals(9, tally.get("key1"));
210: }
```

It's impossible to know without seeing how the tally object is set up:

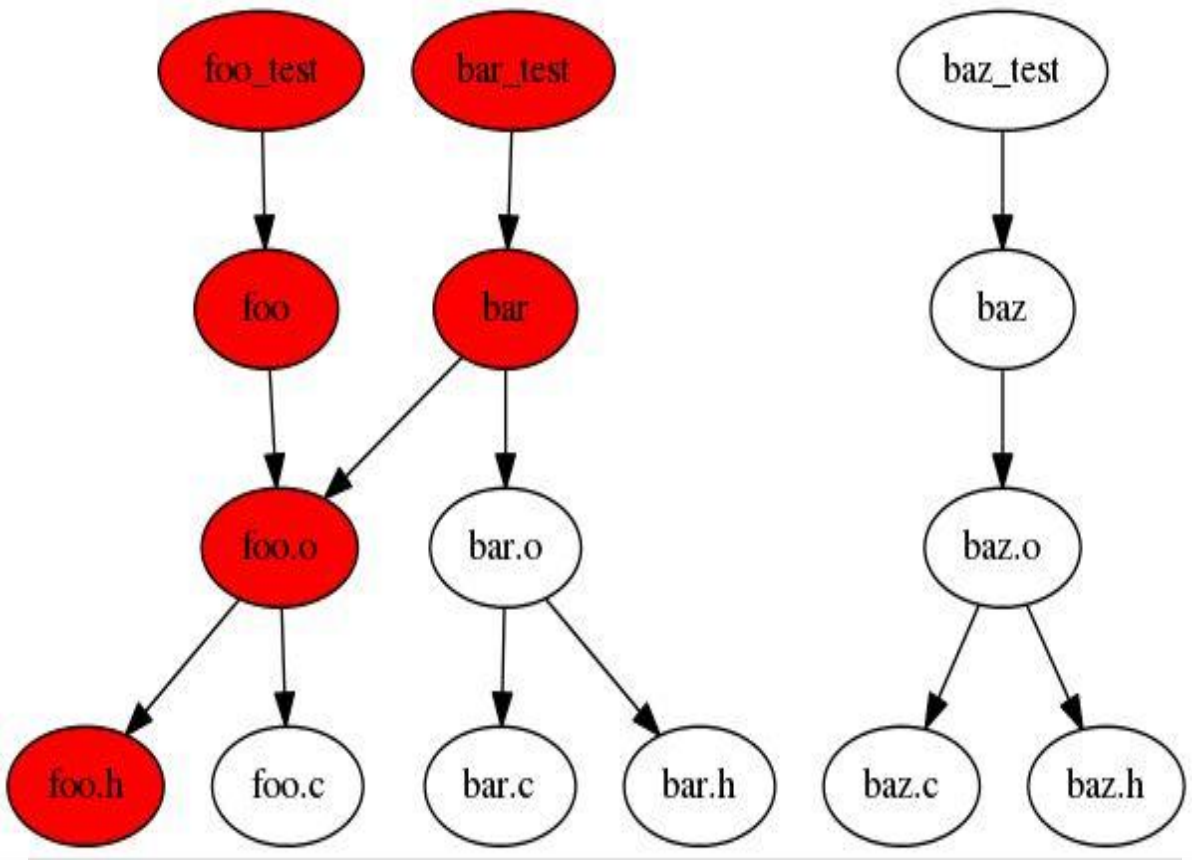
```
1:   private final Tally tally = new Tally();
2:   @Before public void setUp() {
3:     tally.increment("key1", 8);
4:     tally.increment("key2", 100);
5:     tally.increment("key1", 0);
6:     tally.increment("key1", 1);
7:   }
// 200 lines away
208: @Test public void testIncrement_existingKey() {
209:   assertEquals(9, tally.get("key1"));
210: }
```

Keep Cause and Effect Clear!

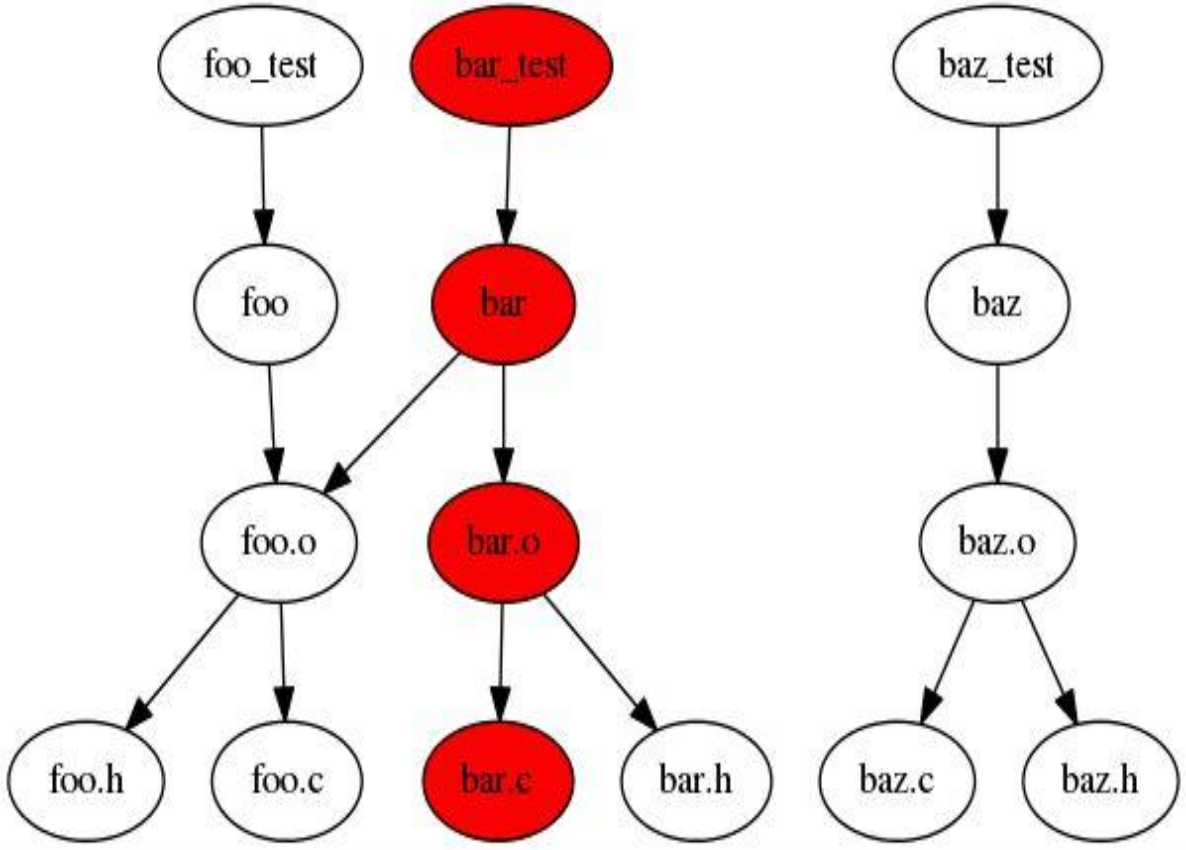
Regression Test Selection (RTS)



Regression Test Selection (RTS)



Regression Test Selection (RTS)



Presubmit Testing

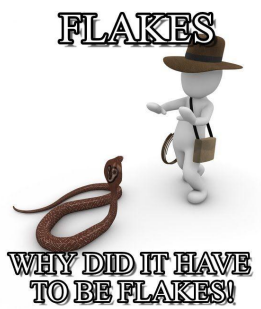
- Uses fine-grained dependencies
- Uses same pool of compute resources
- Avoids breaking the build
- Captures contents of a change and tests in isolation
  - Tests against HEAD
- Integrates with
  - submission tool - submit iff testing is green
  - Code Review Tool - results are posted to the review

Example Presubmit Display

Postsubmit testing

- Continuously runs 4.2M tests as changes are submitted
  - A test is affected iff a file being changed is present in the transitive closure of the test dependencies. (Regression Test Selection)
  - Each test runs in 2 distinct flag combinations (on average)
  - Build and run tests concurrently on distributed backend.
  - Runs as often as capacity allows
- Records the pass / fail result for each test in a database
  - Each run is uniquely identified by the test + flags + change
  - We have 2 years of results for all tests
  - And accurate information about what was changed

Analysis of Test Results at Google



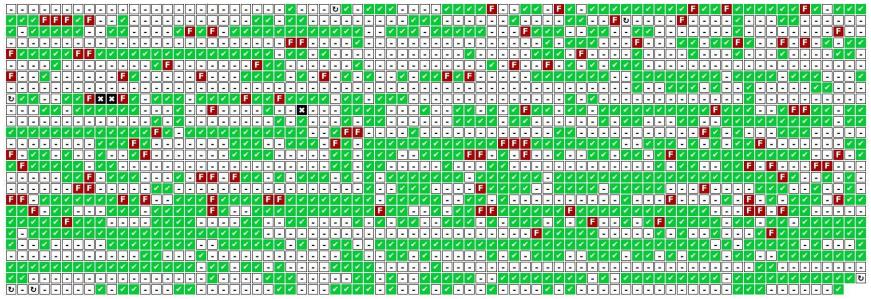
- Analysis of a large sample of tests (1 month) showed:
  - 84% of transitions from Pass -> Fail are from "flaky" tests
  - Only 1.23% of tests ever found a breakage
  - Frequently changed files more likely to cause a breakage
  - 3 or more developers changing a file is more likely to cause a breakage
  - Changes "closer" in the dependency graph more likely to cause a breakage
  - Certain people / automation more likely to cause breakages (oops!)
  - Certain languages more likely to cause breakages (sorry)

Problems of CI in Google: Flaky Tests

- Flaky Tests
  - Test which could fail or pass for the same code
- Sources of test flakiness:
  - Concurrency
  - Environment / setup problems
  - Non-deterministic or undefined behaviors

Flaky Tests

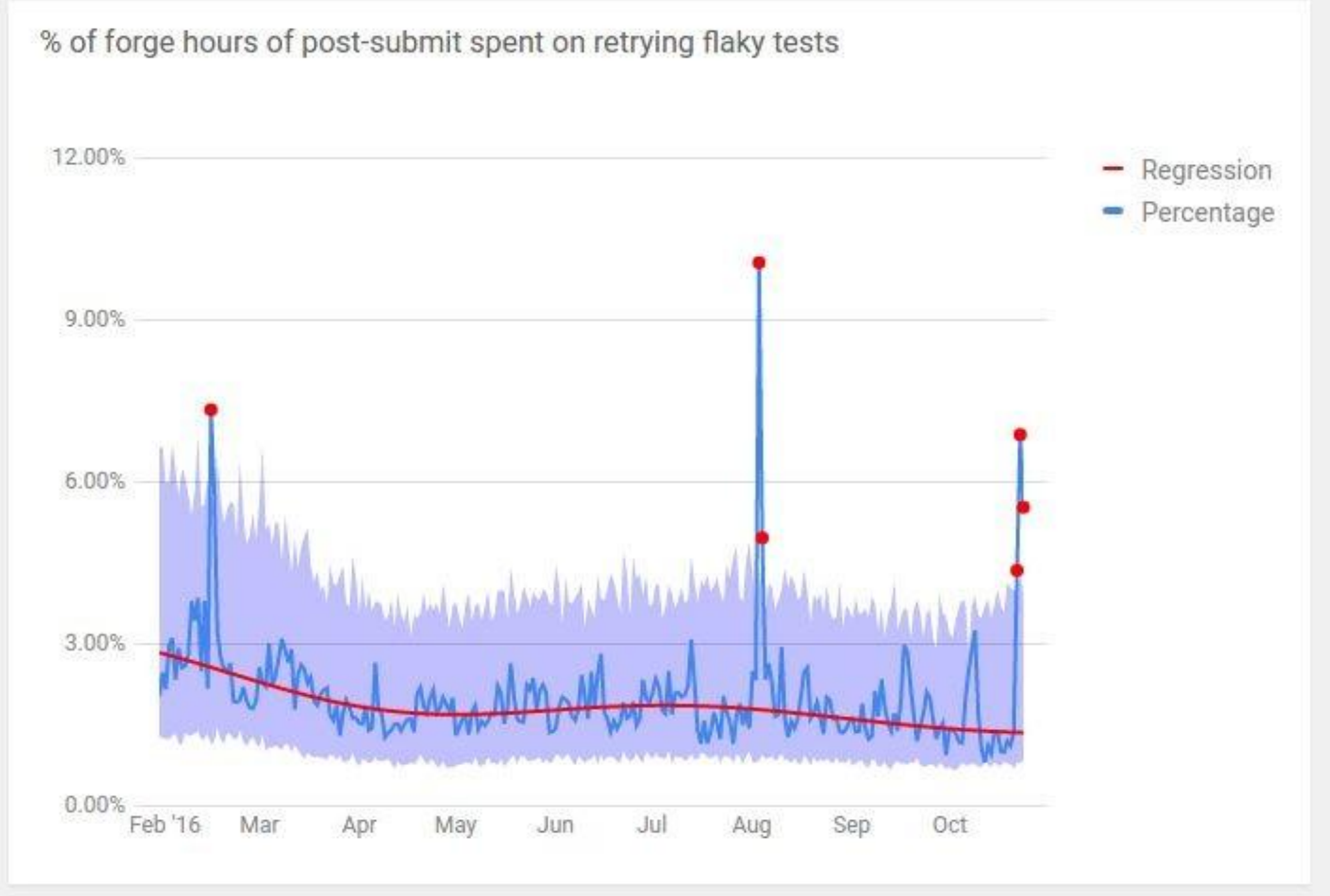
- Test Flakiness is a huge problem
- Flakiness is a test that is observed to both Pass and Fail with the same code
- Almost 16% of our 4.2M tests have some level of flakiness
- Flaky failures frequently block and delay releases
- Developers ignore flaky tests when submitting - sometimes incorrectly
- We spend between 2 and 16% of our compute resources re-running flaky tests



Flaky test impact on project health

- Many tests need to be aggregated to qualify a project
- Probability of flake aggregates as well
- Flakes
  - Consume developer time investigating
  - Delay project releases
  - Waste compute resources re-running to confirm flakes

Percentage of resources spent re-running flakes



Sources of Flakiness

Factors that causes flakiness

- Async Wait
- Concurrency
- Test Order Dependency
- Resource Leak
- Network
- Time
- IO
- Randomness
- Floating Point Operations
- Unordered Collections

See: https://pdfs.semanticscholar.org/02da/46889ee3c6bc44bfa0fc45071195781b99ce.pdf

Example of flaky tests


```
1 @Test
2 public void testRsReportsWrongServerName() throws Exception {
3 MiniHBaseCluster cluster = TEST_UTIL.getHBaseCluster();
4 MiniHBaseClusterRegionServer firstServer =
5 (MiniHBaseClusterRegionServer)cluster.getRegionServer(0);
6 HServerInfo hsi = firstServer.getServerInfo();
7 firstServer.setHServerInfo(...);
8
9 // Sleep while the region server pings back
10 Thread.sleep(2000);
11 assertTrue(firstServer.isOnline());
12 assertEquals(2,cluster.getLiveRegionServerThreads().size());
13 ... // similarly for secondServer
14 }
```

- Test fails if the server does not respond fast enough, e.g., because of thread scheduling or network delay

Flakes are Inevitable

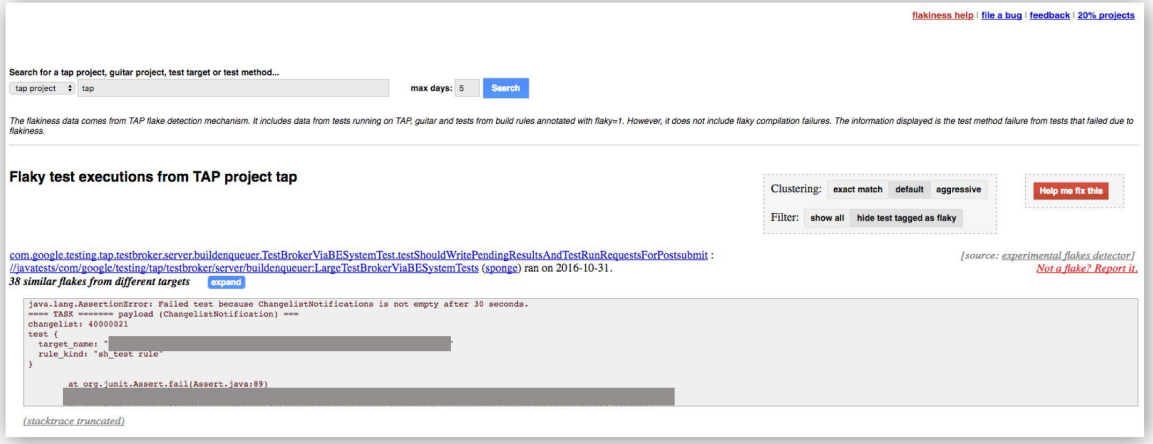
- Continual rate of 1.5% of test executions reporting a "flaky" result
- Despite large effort to identify and remove flakiness
  - Targeted "fixits"
  - Continual pressure on flakes
- Observed insertion rate is about the same as fix rate

Conclusion: Testing systems must be able to deal with a certain level of flakiness. Preferably minimizing the cost to developers

Flaky Test Infrastructure

- We re-run test failure transitions (10x) to verify flakiness
  - If we observe a pass the test was flaky
  - Keep a database and web UI for "known" flaky tests

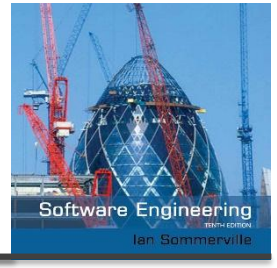
le



Continuous Integration at Google Scale

- 5000+ projects under active development
- 17k submissions per day (1 every 5 seconds)
- 20+ sustained code changes per minute
- 50% of code changes monthly
- 100+ million test cases run per day

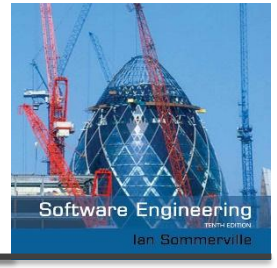
Taken From: https://www.eclipsecon.org/2013/node/1251.html



Chapter 13 – Security Engineering

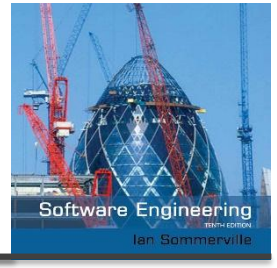
Slides Adapted from Software Engineering Book by Ian Sommerville

Topics covered



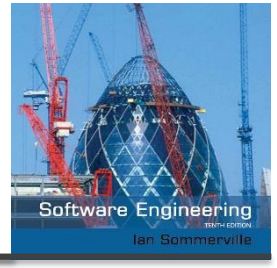
- ✧ Security and dependability
- ✧ Security and organizations
- ✧ Security requirements
- ✧ Secure systems design
- ✧ Security testing and assurance

Security engineering



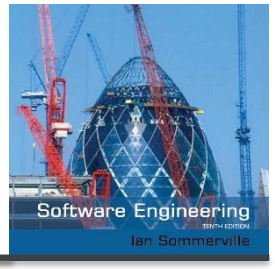
- Tools, techniques and methods to support the development and maintenance of systems that can resist malicious attacks that are intended to damage a computer-based system or its data.
- A sub-field of the broader field of computer security.

Security dimensions



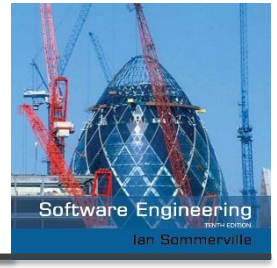
- *Confidentiality*
  - Information in a system may be disclosed or made accessible to people or programs that are not authorized to have access to that information.
- *Integrity*
  - Information in a system may be damaged or corrupted making it unusual or unreliable.
- *Availability*
  - Access to a system or its data that is normally available may not be possible.

Security levels



- Infrastructure security, which is concerned with maintaining the security of all systems and networks that provide an infrastructure and a set of shared services to the organization.
- Application security, which is concerned with the security of individual application systems or related groups of systems.
- Operational security, which is concerned with the secure operation and use of the organization's systems.

System layers where security may be compromised



Application

Reusable components and libraries

Middleware

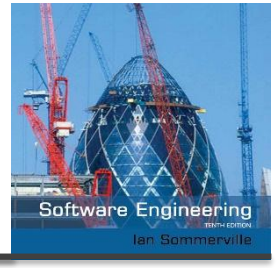
Database management

Generic, shared applications (browsers, e--mail, etc)

Operating System

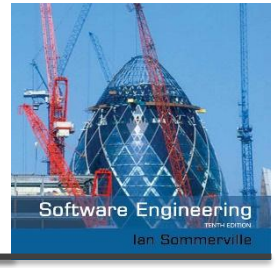
Network Computer hardware

Application/infrastructure security



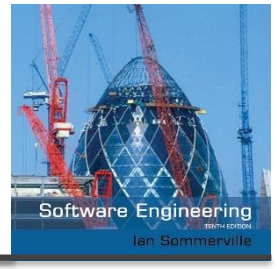
- Application security is a software engineering problem where the system is designed to resist attacks.
- Infrastructure security is a systems management problem where the infrastructure is configured to resist attacks.
- The focus of this chapter is application security rather than infrastructure security.

System security management

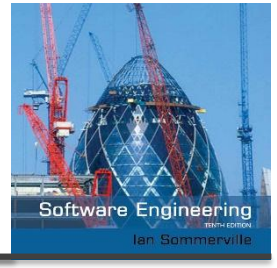


- ✧ User and permission management
  - Adding and removing users from the system and setting up appropriate permissions for users
- ✧ Software deployment and maintenance
  - Installing application software and middleware and configuring these systems so that vulnerabilities are avoided.
- ✧ Attack monitoring, detection and recovery
  - Monitoring the system for unauthorized access, design strategies for resisting attacks and develop backup and recovery strategies.

Operational security

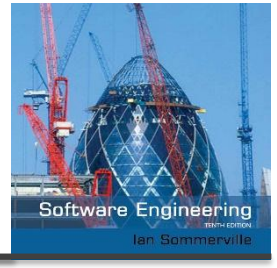


- Primarily a human and social issue
- Concerned with ensuring the people do not take actions that may compromise system security
  - E.g. Tell others passwords, leave computers logged on
- Users sometimes take insecure actions to make it easier for them to do their jobs
- There is therefore a trade-off between system security and system effectiveness.



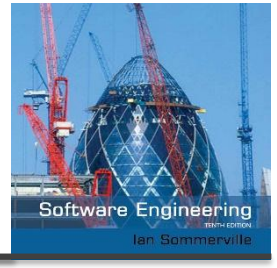
Security and dependability

Security



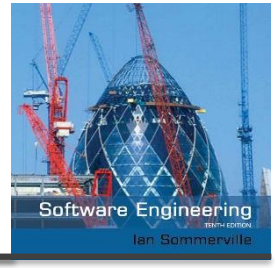
- The security of a system is a system property that reflects the system's ability to protect itself from accidental or deliberate external attack.
- Security is essential as most systems are networked so that external access to the system through the Internet is possible.
- Security is an essential pre-requisite for availability, reliability and safety.

Fundamental security



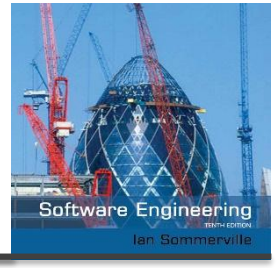
- If a system is a networked system and is insecure then statements about its reliability and its safety are unreliable.
- These statements depend on the executing system and the developed system being the same. However, intrusion can change the executing system and/or its data.
- Therefore, the reliability and safety assurance is no longer valid.

Security terminology



Term	Definition
Asset	Something of value which has to be protected. The asset may be the software system itself or data used by that system.
Attack	An exploitation of a system's vulnerability. Generally, this is from outside the system and is a deliberate attempt to cause some damage.
Control	A protective measure that reduces a system's vulnerability. Encryption is an example of a control that reduces a vulnerability of a weak access control system
Exposure	Possible loss or harm to a computing system. This can be loss or damage to data, or can be a loss of time and effort if recovery is necessary after a security breach.
Threat	Circumstances that have potential to cause loss or harm. You can think of these as a system vulnerability that is subjected to an attack.
Vulnerability	A weakness in a computer-based system that may be exploited to cause loss or harm.

Examples of security terminology (Mentcare)



Term	Example
Asset	The records of each patient that is receiving or has received treatment.
Exposure	Potential financial loss from future patients who do not seek treatment because they do not trust the clinic to maintain their data. Financial loss from legal action by the sports star. Loss of reputation.
Vulnerability	A weak password system which makes it easy for users to set guessable passwords. User ids that are the same as names.
Attack	An impersonation of an authorized user.
Threat	An unauthorized user will gain access to the system by guessing the credentials (login name and password) of an authorized user.
Control	A password checking system that disallows user passwords that are proper names or words that are normally included in a dictionary.

Question: Fill in the following form for the SUSTech Sakai system

Term	Example
Asset	
Exposure	
Vulnerability	
Attack	
Threat	
Control	

Threat types

- Interception threats that allow an attacker to gain access to an asset.
  - A possible threat to the Mentcare system might be a situation where an attacker gains access to the records of an individual patient.
- Interruption threats that allow an attacker to make part of the system unavailable.
  - A possible threat might be a denial of service attack on a system database server so that database connections become impossible.

Threat types

- Modification threats that allow an attacker to tamper with a system asset.
  - In the Mentcare system, a modification threat would be where an attacker alters or destroys a patient record.
- Fabrication threats that allow an attacker to insert false information into a system.
  - This is perhaps not a credible threat in the Mentcare system but would be a threat in a banking system, where false transactions might be added to the system that transfer money to the perpetrator's bank account.

Security assurance

- Vulnerability avoidance
  - The system is designed so that vulnerabilities do not occur. For example, if there is no external network connection then external attack is impossible
- Attack detection and elimination
  - The system is designed so that attacks on vulnerabilities are detected and neutralised before they result in an exposure. For example, virus checkers find and remove viruses before they infect a system
- Exposure limitation and recovery
  - The system is designed so that the adverse consequences of a successful attack are minimised. For example, a backup policy allows damaged information to be restored

Security and dependability

- *Security and reliability*
  - If a system is attacked and the system or its data are corrupted as a consequence of that attack, then this may induce system failures that compromise the reliability of the system.
- *Security and availability*
  - A common attack on a web-based system is a denial of service attack, where a web server is flooded with service requests from a range of different sources. The aim of this attack is to make the system unavailable.

Security and dependability

- *Security and safety*
  - An attack that corrupts the system or its data means that assumptions about safety may not hold. Safety checks rely on analysing the source code of safety critical software and assume the executing code is a completely accurate translation of that source code. If this is not the case, safety-related failures may be induced and the safety case made for the software is invalid.
- *Security and resilience*
  - Resilience is a system characteristic that reflects its ability to resist and recover from damaging events. The most probable damaging event on networked software systems is a cyberattack of some kind so most of the work now done in resilience is aimed at deterring, detecting and recovering from such attacks.

Security risk assessment and management

- Risk assessment and management is concerned with assessing the possible losses that might ensue from attacks on the system and balancing these losses against the costs of security procedures that may reduce these losses.
- Risk management should be driven by an organisational security policy.
- Risk management involves
  - Preliminary risk assessment
  - Life cycle risk assessment
  - Operational risk assessment

Preliminary risk assessment

- The aim of this initial risk assessment is to identify generic risks that are applicable to the system and to decide if an adequate level of security can be achieved at a reasonable cost.
- The risk assessment should focus on the identification and analysis of high-level risks to the system.
- The outcomes of the risk assessment process are used to help identify security requirements.

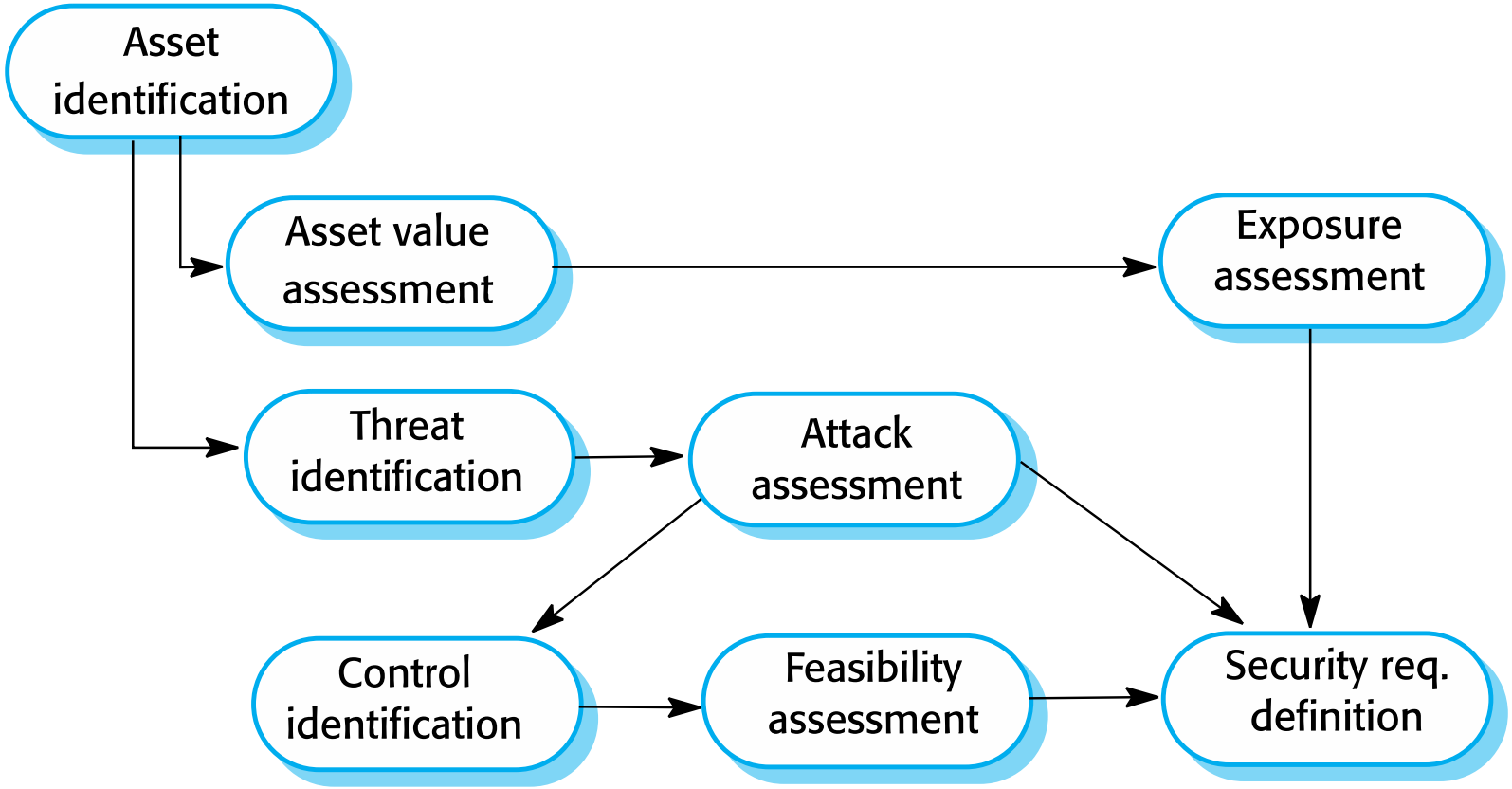
Design risk assessment

- This risk assessment takes place during the system development life cycle and is informed by the technical system design and implementation decisions.
- The results of the assessment may lead to changes to the security requirements and the addition of new requirements.
- Known and potential vulnerabilities are identified, and this knowledge is used to inform decision making about the system functionality and how it is to be implemented, tested, and deployed.

Operational risk assessment

- This risk assessment process focuses on the use of the system and the possible risks that can arise from human behavior.
- Operational risk assessment should continue after a system has been installed to take account of how the system is used.
- Organizational changes may mean that the system is used in different ways from those originally planned. These changes lead to new security requirements that have to be implemented as the system evolves.

The preliminary risk assessment process for security requirements



Security risk assessment

- Asset identification
  - Identify the key system assets (or services) that have to be protected.
- Asset value assessment
  - Estimate the value of the identified assets.
- Exposure assessment
  - Assess the potential losses associated with each asset.
- Threat identification
  - Identify the most probable threats to the system assets

Security risk assessment

- Attack assessment
  - Decompose threats into possible attacks on the system and the ways that these may occur.
- Control identification
  - Propose the controls that may be put in place to protect an asset.
- Feasibility assessment
  - Assess the technical feasibility and cost of the controls.
- Security requirements definition
  - Define system security requirements. These can be infrastructure or application system requirements.

Asset analysis in a preliminary risk assessment report for the Mentcare system

Asset	Value	Exposure
The information system	High. Required to support all clinical consultations. Potentially safety-critical.	High. Financial loss as clinics may have to be canceled. Costs of restoring system. Possible patient harm if treatment cannot be prescribed.
The patient database	High. Required to support all clinical consultations. Potentially safety-critical.	High. Financial loss as clinics may have to be canceled. Costs of restoring system. Possible patient harm if treatment cannot be prescribed.
An individual patient record	Normally low although may be high for specific high-profile patients.	Low direct losses but possible loss of reputation.

Threat and control analysis in a preliminary risk assessment report

Threat	Probability	Control	Feasibility
An unauthorized user gains access as system manager and makes system unavailable	Low	Only allow system management from specific locations that are physically secure.	Low cost of implementation but care must be taken with key distribution and to ensure that keys are available in the event of an emergency.
An unauthorized user gains access as system user and accesses confidential information	High	Require all users to authenticate themselves using a biometric mechanism. Log all changes to patient information to track system usage.	Technically feasible but high-cost solution. Possible user resistance. Simple and transparent to implement and also supports recovery.

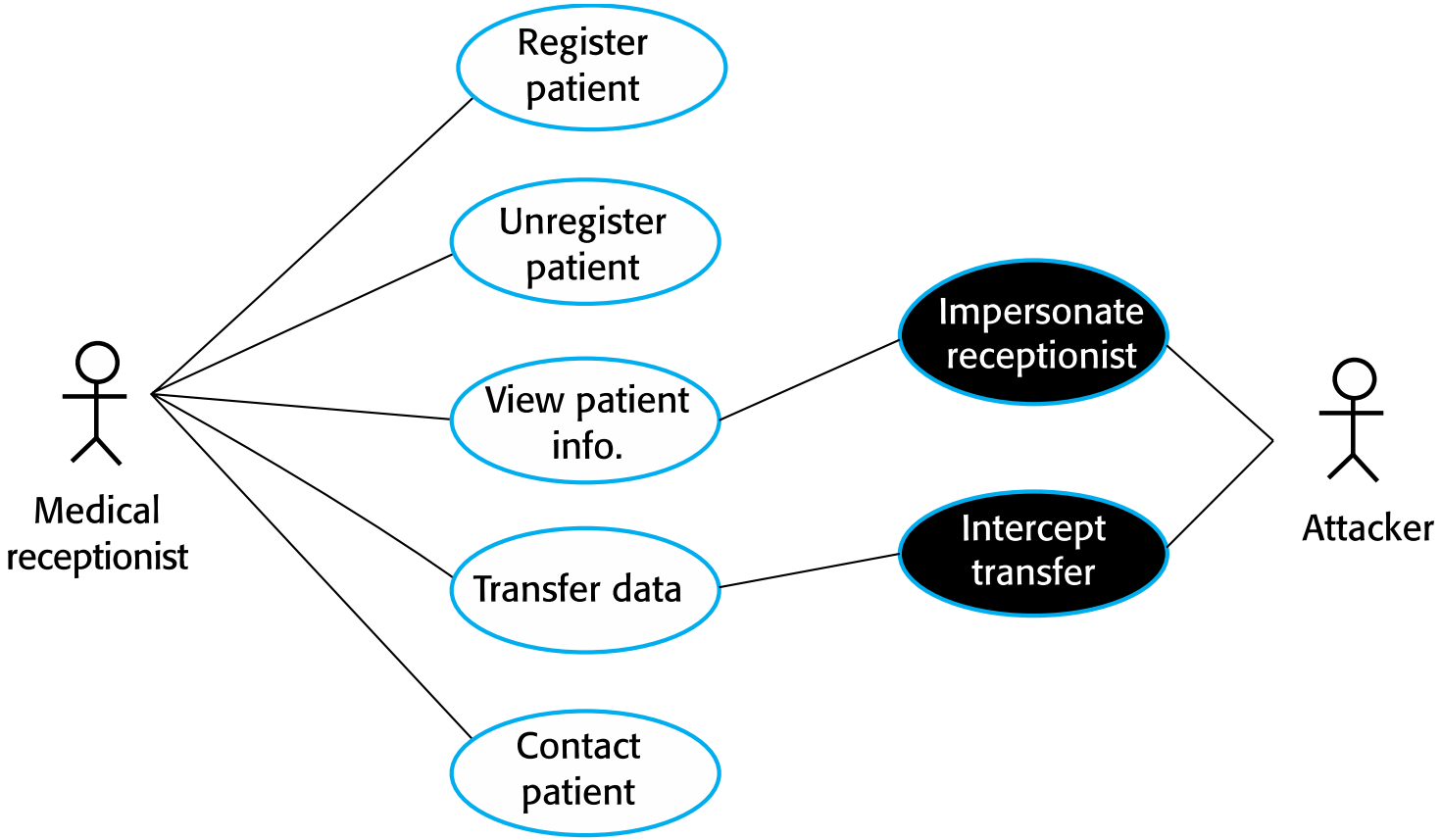
Security requirements for the Mentcare system

- Patient information shall be downloaded at the start of a clinic session to a secure area on the system client that is used by clinical staff.
- All patient information on the system client shall be encrypted.
- Patient information shall be uploaded to the database after a clinic session has finished and deleted from the client computer.
- A log on a separate computer from the database server must be maintained of all changes made to the system database.

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?

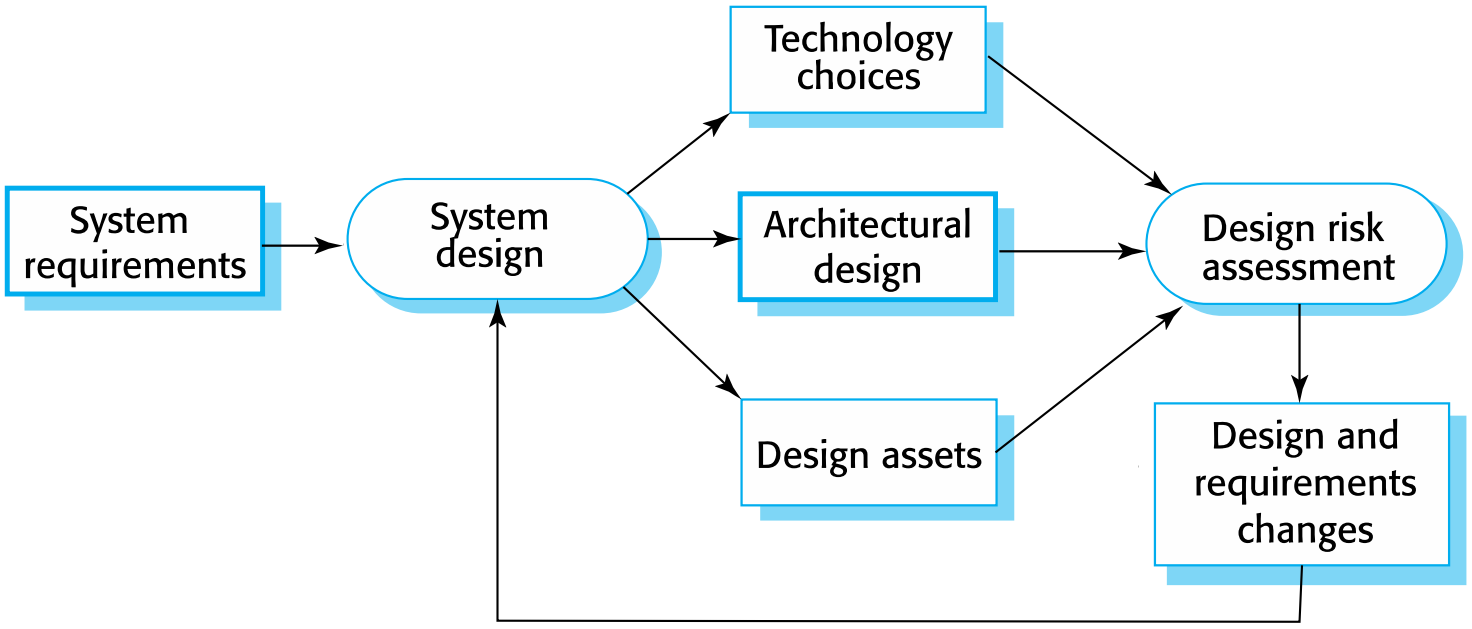
Design compromises

- 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.

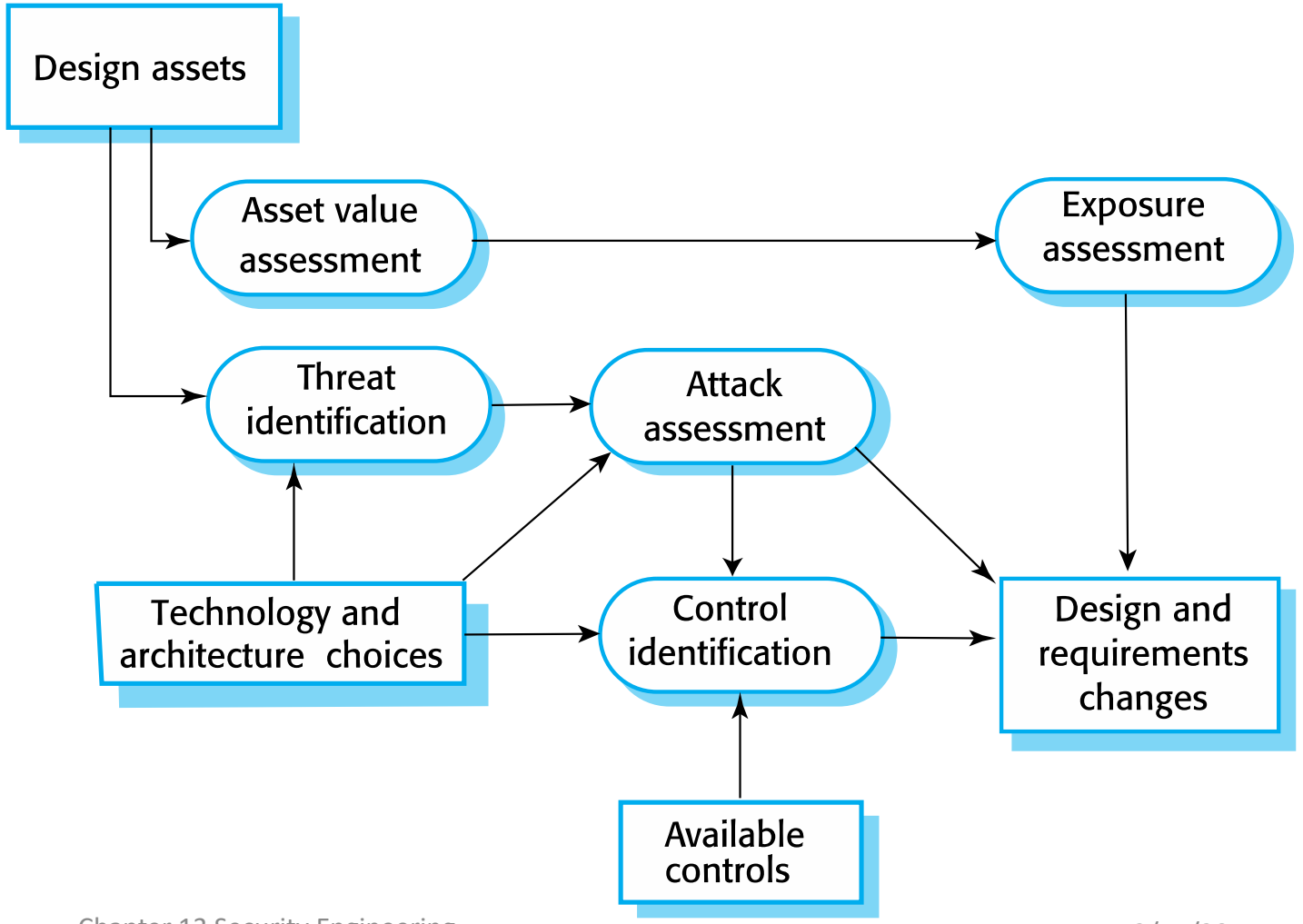
Design and risk assessment



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

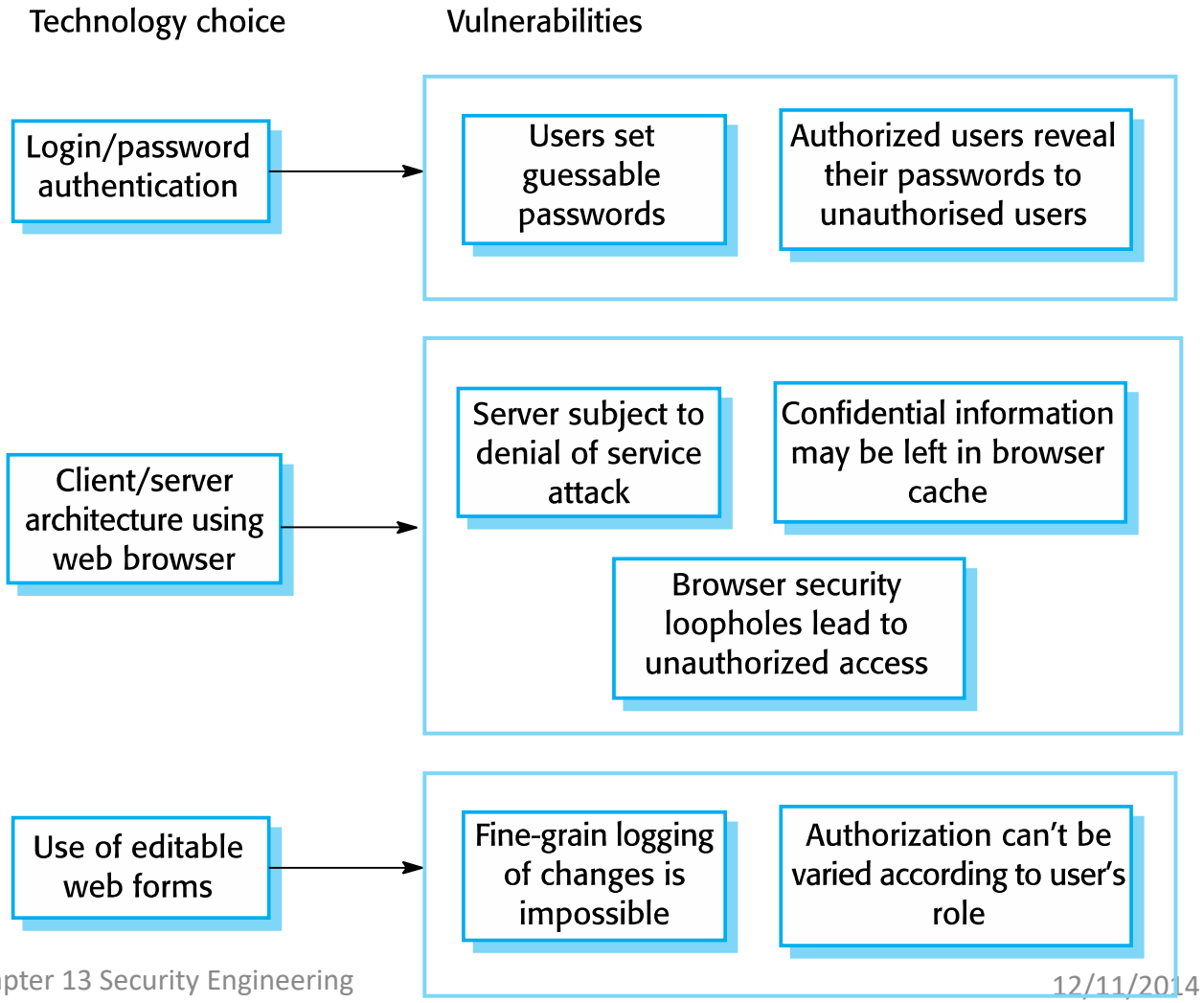
Design risk assessment



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



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.

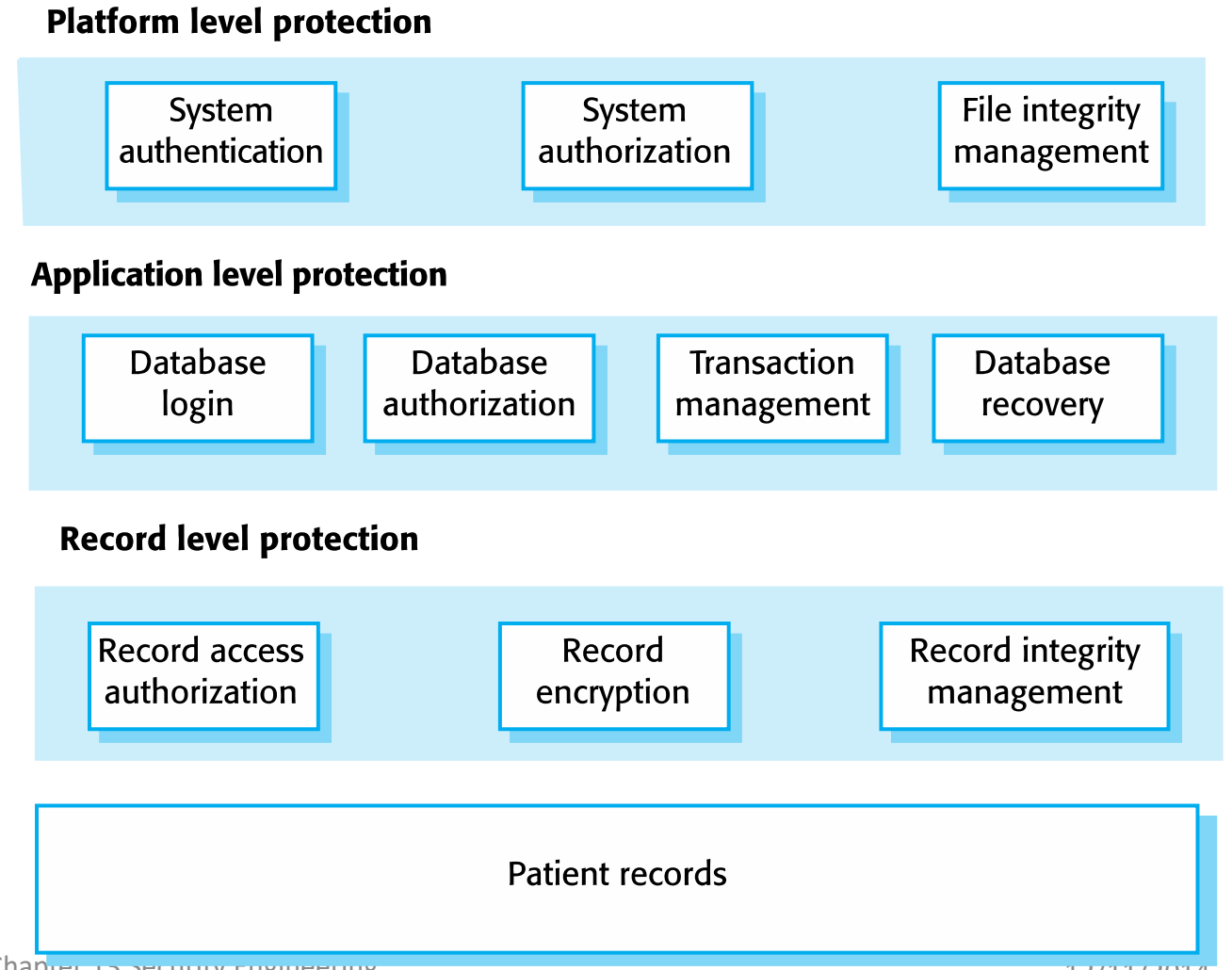
Architectural design

- 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

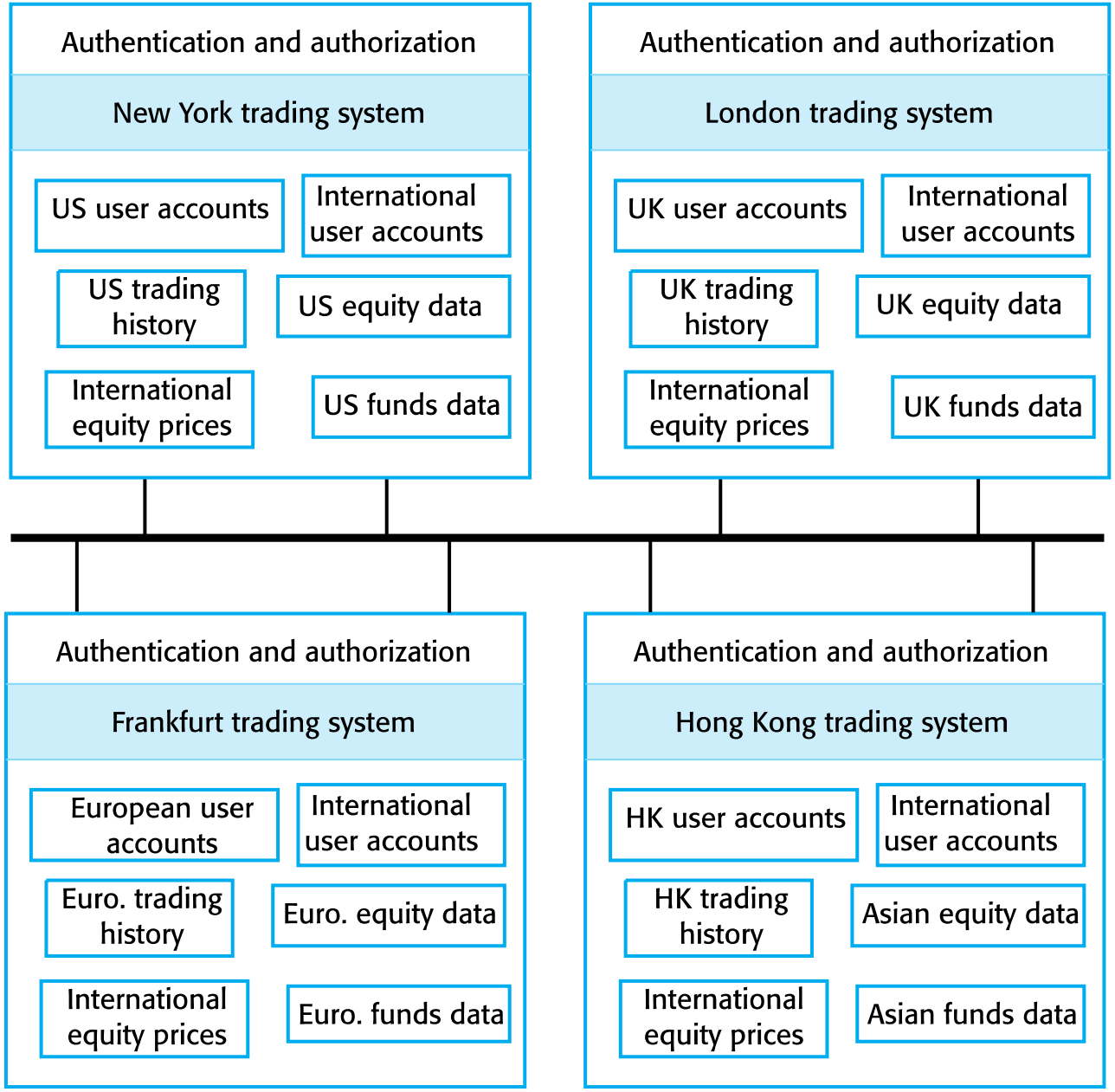
- Platform-level protection
  - Top-level controls on the platform on which a system runs.
- Application-level protection
  - Specific protection mechanisms built into the application itself e.g. additional password protection.
- Record-level protection
  - Protection that is invoked when access to specific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

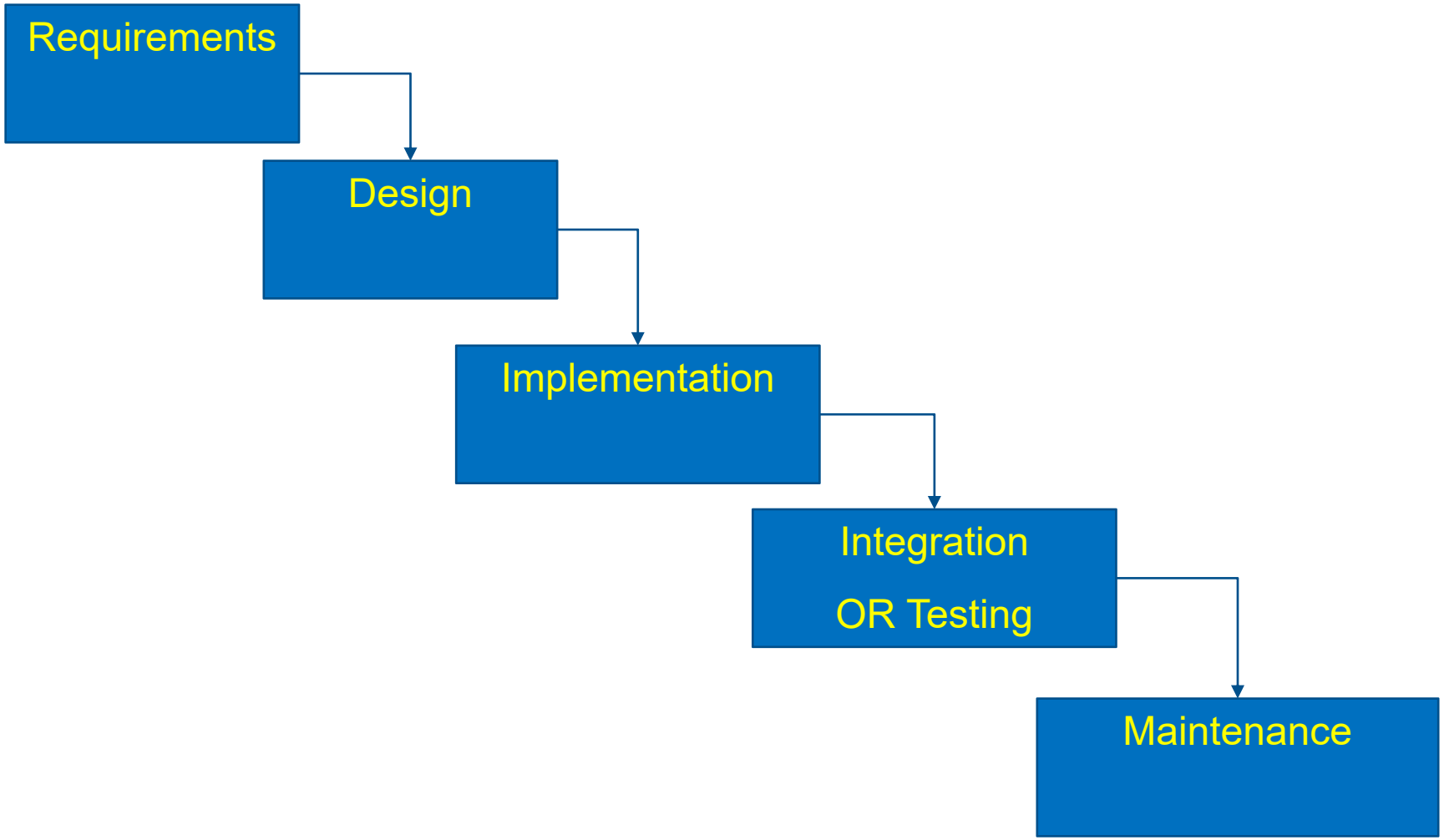
- Key issues when designing a secure systems architecture include organizing the system structure to protect key assets and distributing the system assets to minimize the losses from a successful attack.
- Security design guidelines sensitize system designers to security issues that they may not have 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

Fault (defect, bug) appears in the program

Fault remains undetected during testing

Program failure occurs during execution (program behaves unexpectedly)

Running the test inputs …

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 ());
}
```

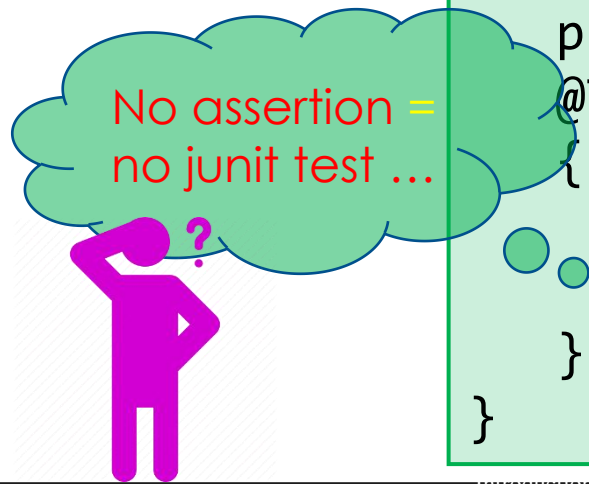
Example JUnit Test

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

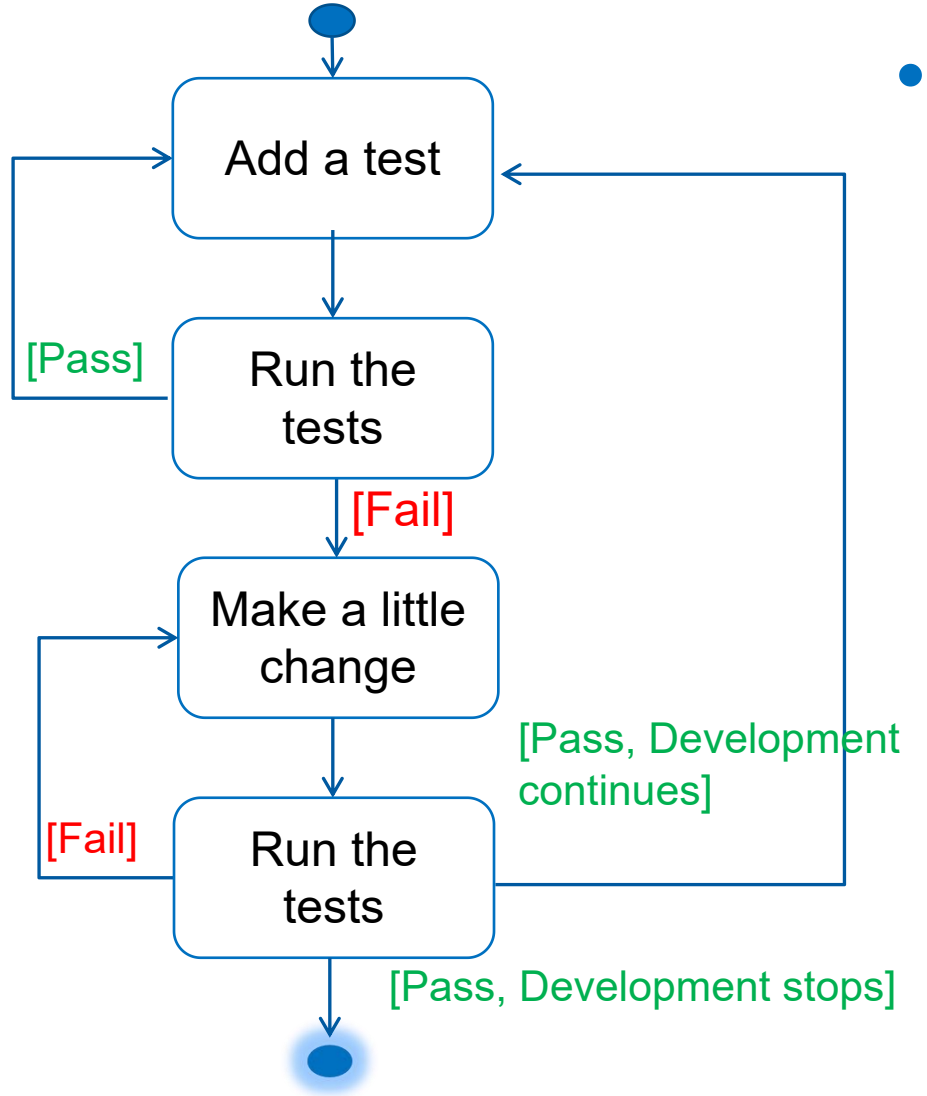
```
import org.junit.Test;
import static org.junit.Assert.*;
public class calcTest
{
    private Calc calc;
    @Test public void testAdd()
    {
        calc = new Calc ();
        assertEquals((long) 5, calc.add(2,3));
    }
}
```

Expected result

The test



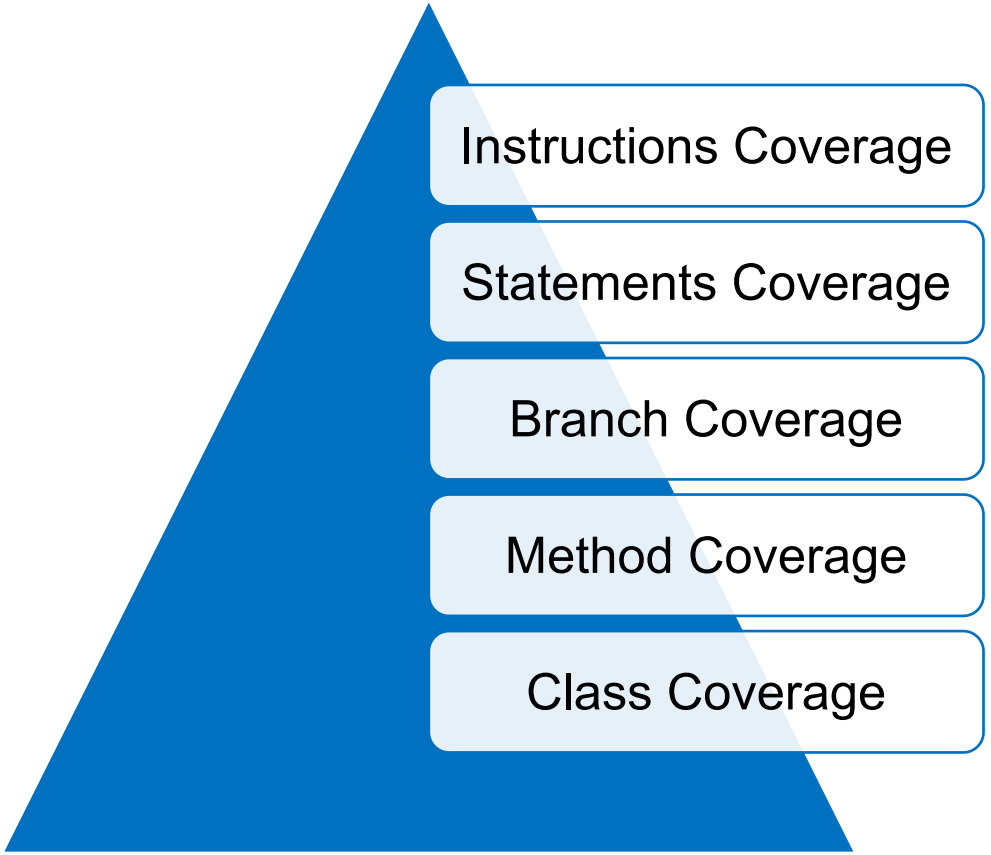
Steps in Test Driven Development (TDD)



- 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

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

Different tools tells you different values of Cyclomatic Complexity

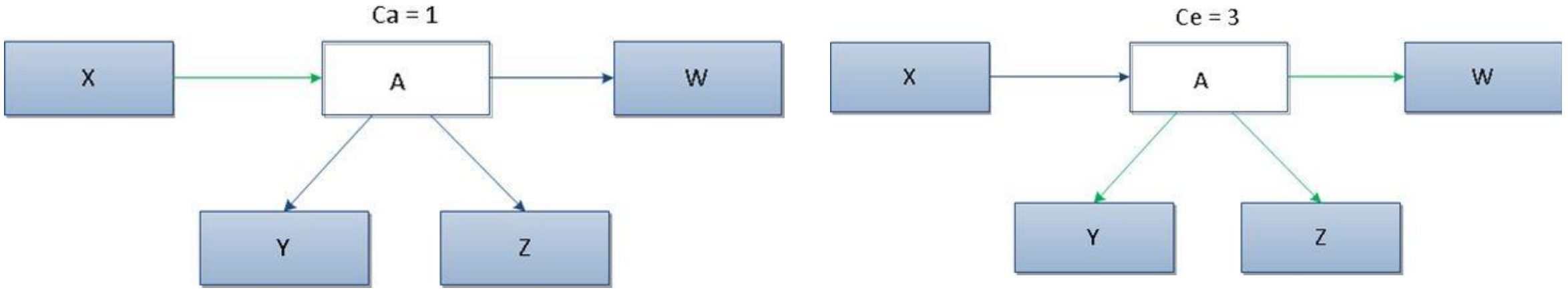
- Original paper is not clear about how to derive the control flow graph
  - different implementations gives different values for the same code.
  - For example, the following code is reported with complexity 2 by the Eclipse Metrics Plugin, with 4 by GMetrics, and with complexity 5 by SonarQube:

```
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

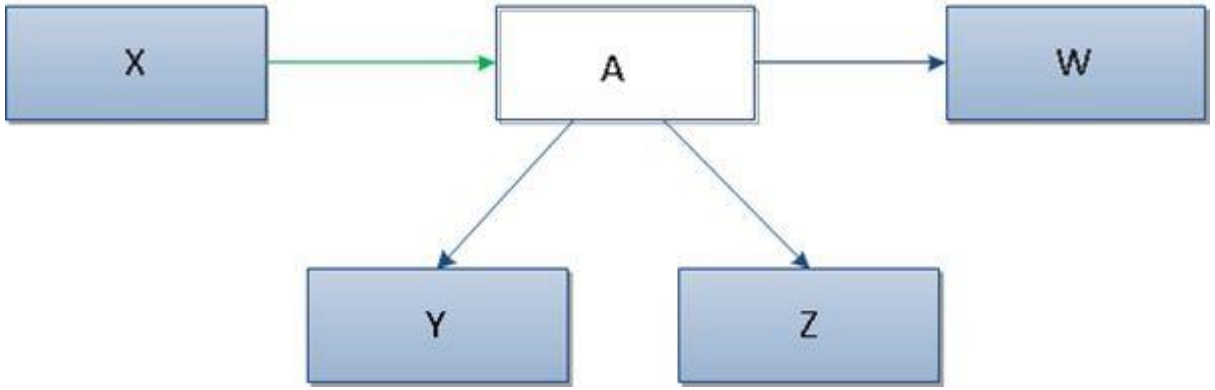
Instability = Ce / (Ca + Ce)



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

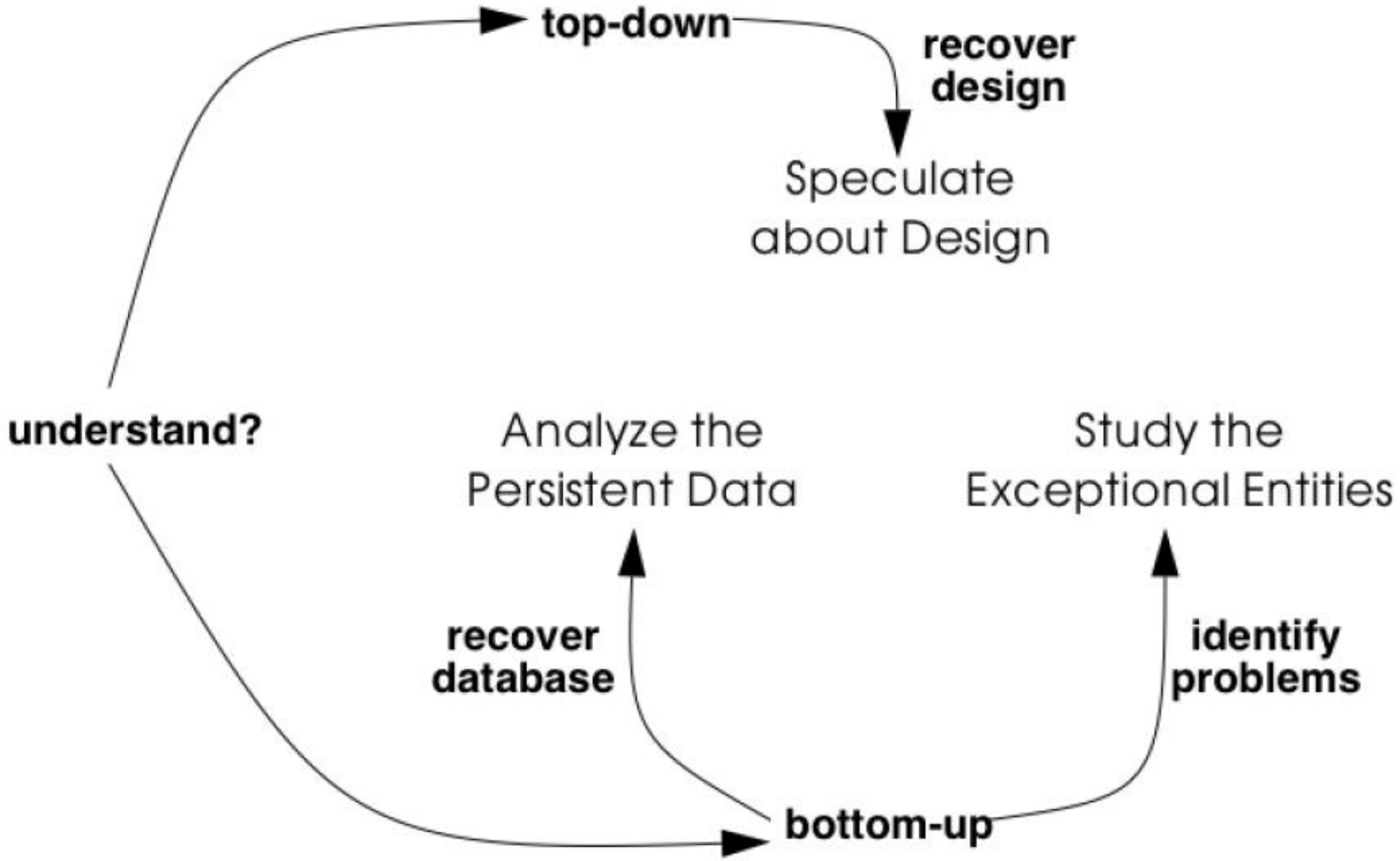
OORP book

- 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

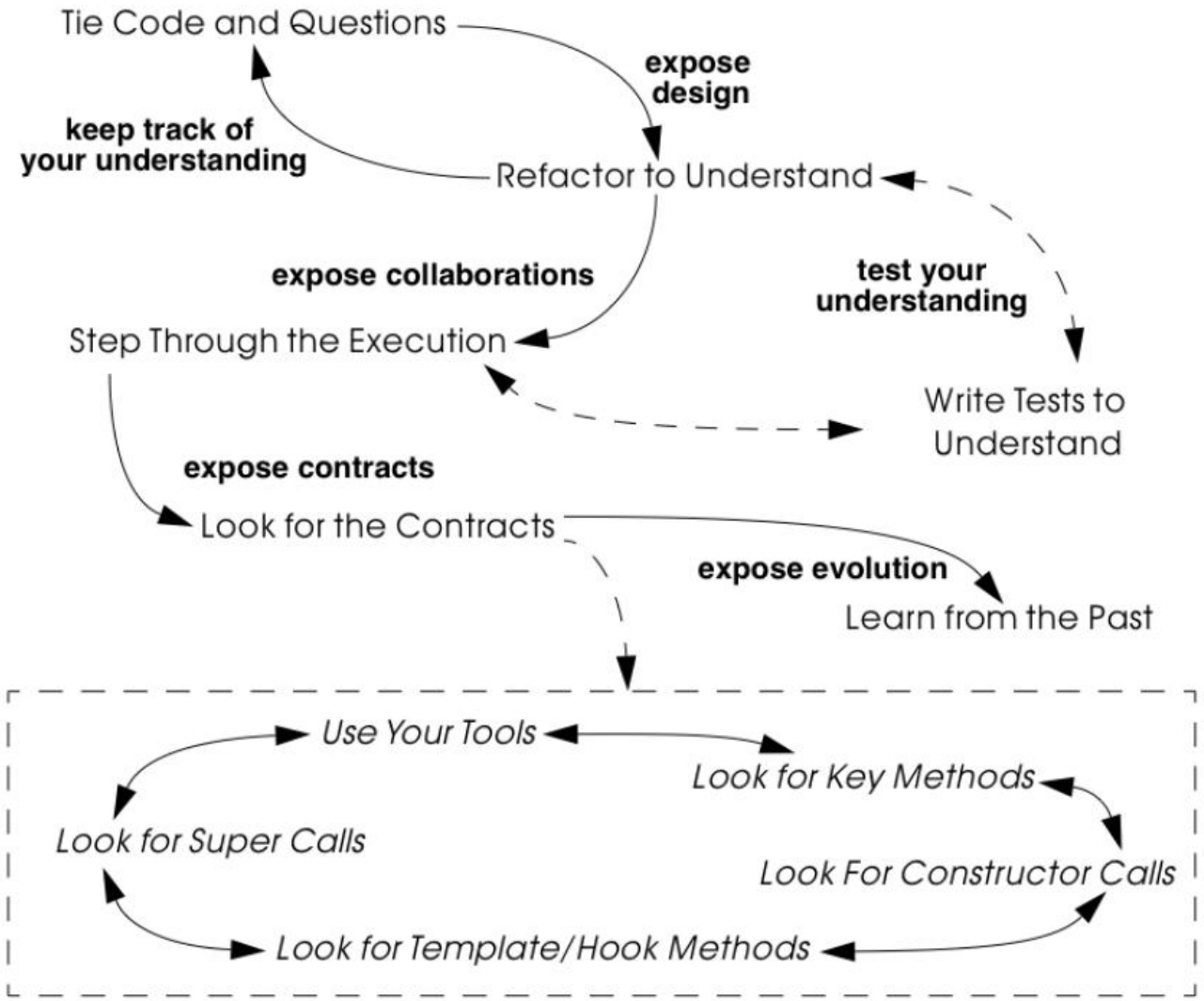
Reverse engineering patterns
Read All Code in 1 Hour

- 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

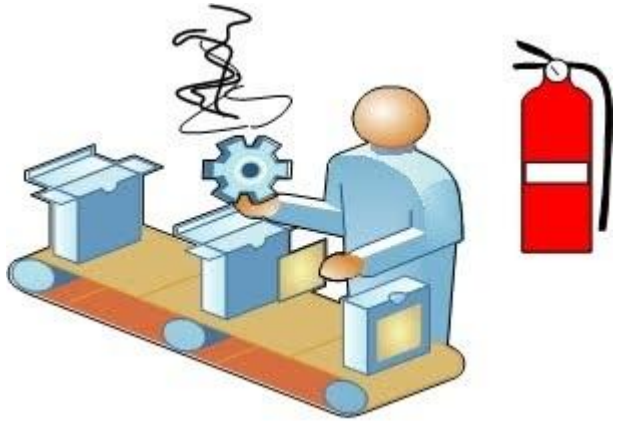
- The *first sentence* should summarize the purpose of the element
- For methods, omit the subject and write in the third-person narrative form
  - Good: Finds the first blank in the string.
  - Not as good: Find the first blank in the string.
  - Bad: This method finds the first blank in the string.
  - 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!

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