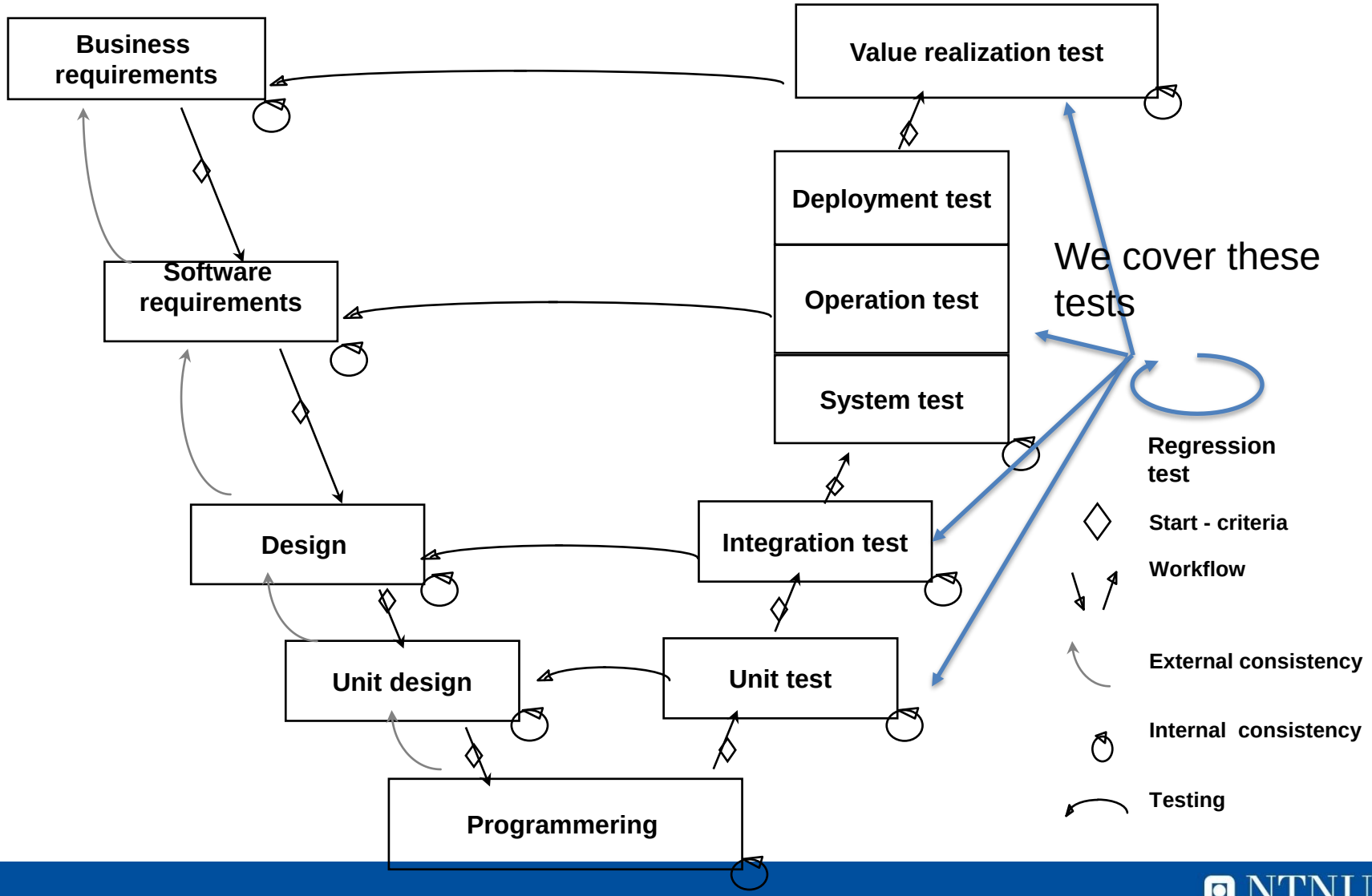
Integration, system, acceptance, and regression testing

V-model


Business requirements
Value realization test
Deployment test
Software requirements
Operation test
System test
We cover these tests
Regression test
Design
Integration test
Start - criteria
Workflow
Unit design
Unit test
External consistency
Internal consistency
Testing
Programmering

Integration testing

- Focus on testing interfaces between components
- Not as well understood as unit testing and system testing
- Usually poorly done

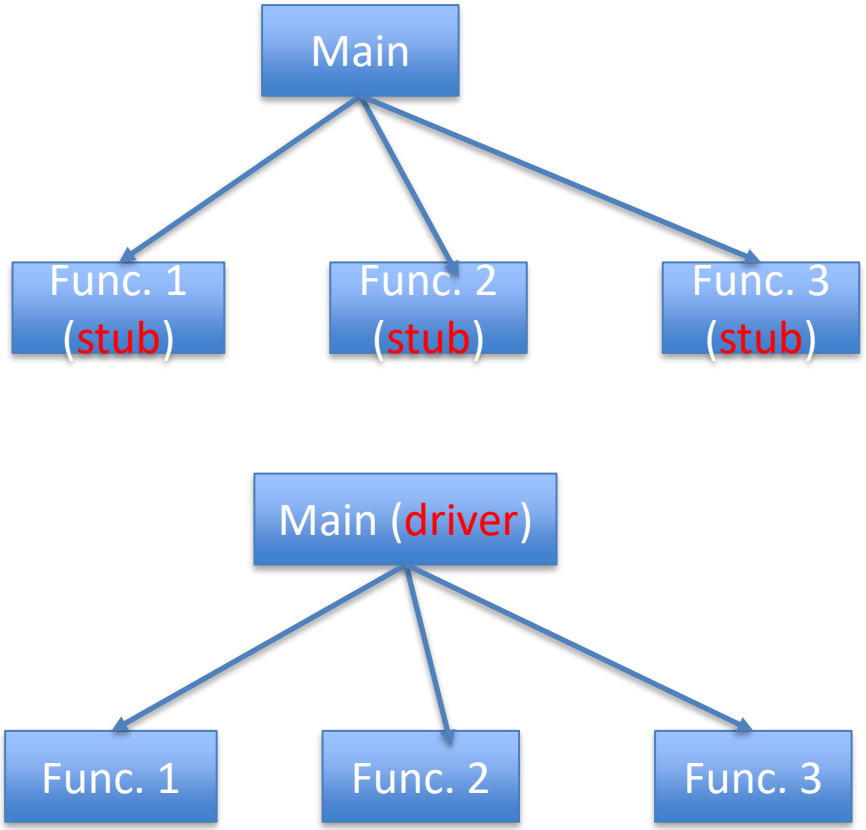
- Strategies
  - Decomposition-based
  - Call graph-based/interface matrix
  - Path-based

Decomposition-based integration

- Based on functional decomposition tree
- Need to know function dependencies between components
- Based on the order of integration testing, can be classified into four strategies
  - Top-down
  - Bottom-up
  - Sandwich
  - Big bang

Decomposition-based integration (cont')

- Top-down
  - Begin with main
  - Use "stubs" to simulate called functions
  - Replace stubs with real functions one by one
- Bottom-up
  - Begin with leaves
  - Use "drivers" to emulate functions call the leaves
  - Replace "drivers" with real function later
- Sandwich
- Big bang



Pros and cons of decomposition-based integration

- Pros
  - Incremental and intuitive
  - Easy fault isolation

- Cons
  - Need “stub” or “driver”

Call graph-based/interface matrix-based integration

- Based on the call graph/interface matrix of components
- Use actual components rather than “stubs” or “drivers”
- Two strategies
  - Pairwise integration
  - Neighborhoods integration

Pair-wise integration

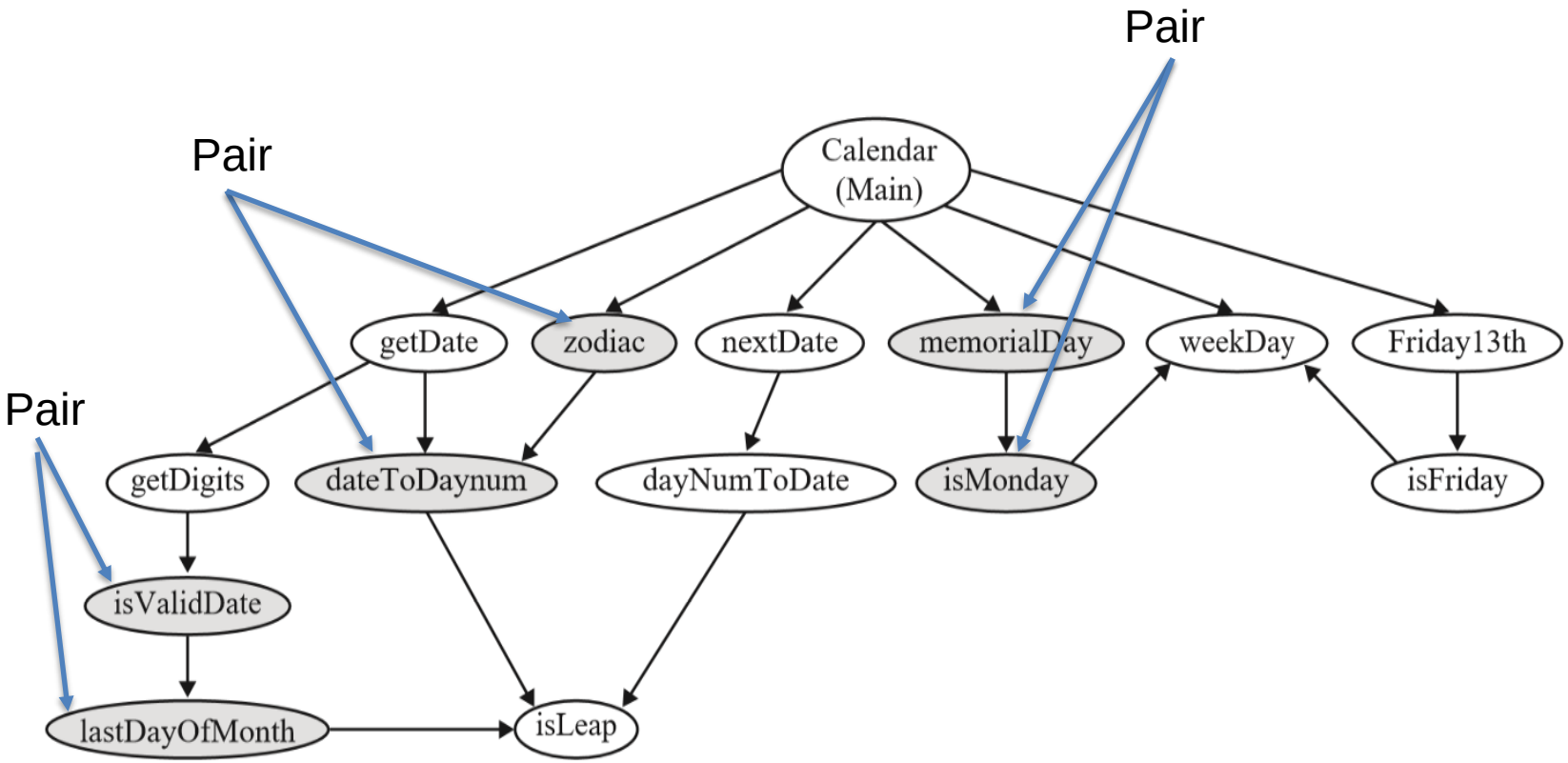


Figure 13.9 Three pairs for pairwise integration.

*

* Software Testing, A craftsman's approach, 4th Edition

Interface matrix

	Sys 1	Sys 2	Sys 3	Sys 4
Sys 1	X	X		
Sys 2	X	X		
Sys 3			X	X
Sys 4			X	X

Neighborhoods integration

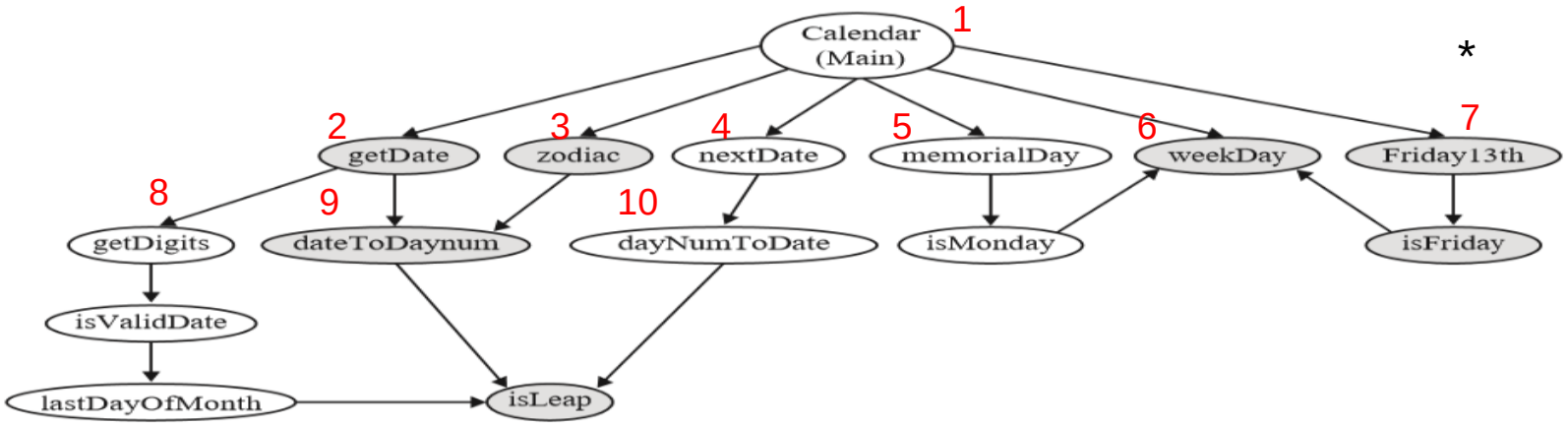


Figure 13.10 Three neighborhoods (of radius 1) for neighborhood integration.

Table 13.1 Neighborhoods of Radius 1 in Calendar Call Graph

Neighborhoods in Calendar Program Call Graph			
Node	*Unit Name*	*Predecessors*	*Successors*
1	Calendar (Main)	(None)	2, 3, 4, 5, 6, 7
2	getDate	1	8, 9
3	zodiac	1	9

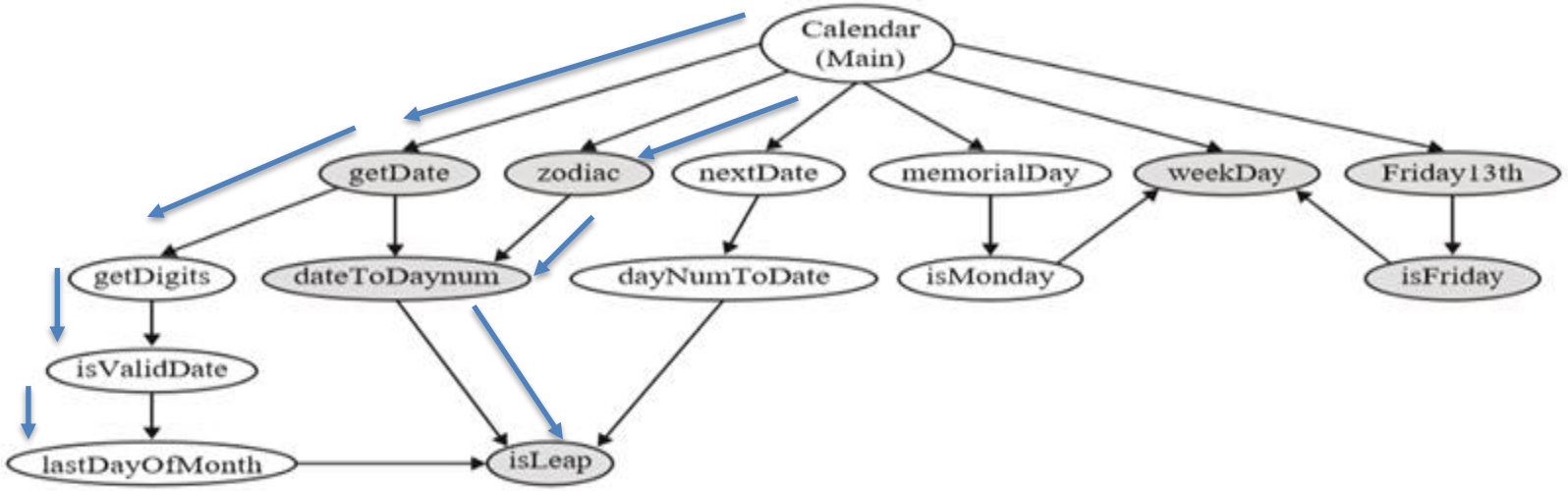
* Software Testing, A craftsman's approach, 4th Edition

Pros and cons of call graph-based integration

- Pros
  - Do not need “stub” or “driver”
- Cons
  - Limited to local integration

Path-based integration

- Not focusing on testing interfaces among separately developed and tested unit
- Rather focus on interactions among these units



Path-based integration (cont')

- Like enlarged unit testing
  - The node is not a statement
  - The node is a component/test unit
  - The edge is message transferred between components

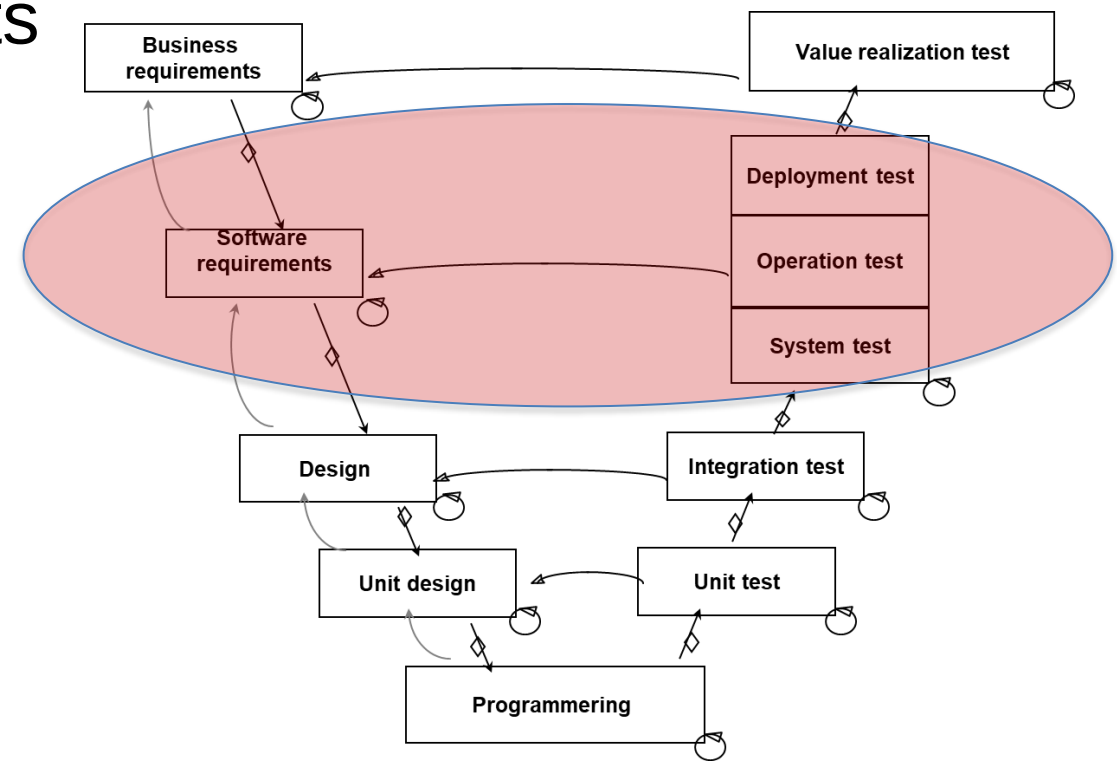
Pros and cons of path-based integration

- Pros
  - Test more global and complex integrations
  - Closely coupled with actual system behavior

- Cons
  - Difficult fault isolation
  - Extra effort is need to identify message path

System tests categories

- Type of system tests
  - Functionality
  - Reliability
  - Usability
  - Performance
  - Robustness
  - Scalability
  - Stress
  - Load and stability
  - …

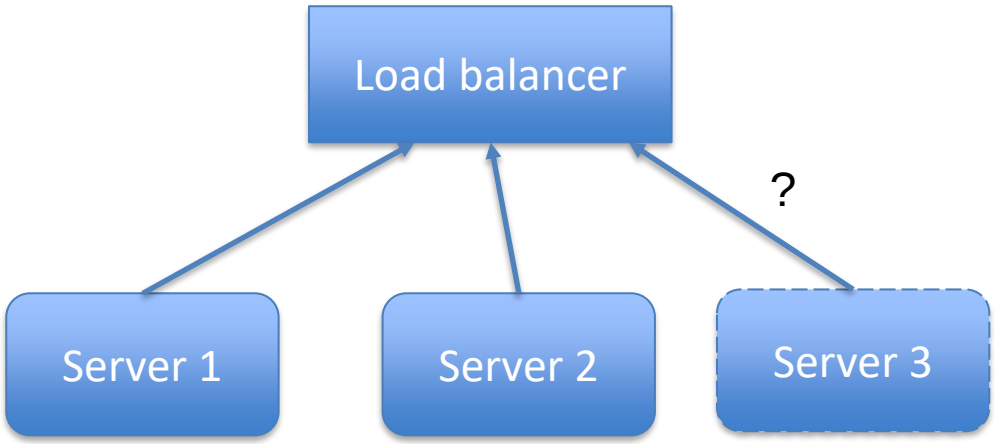


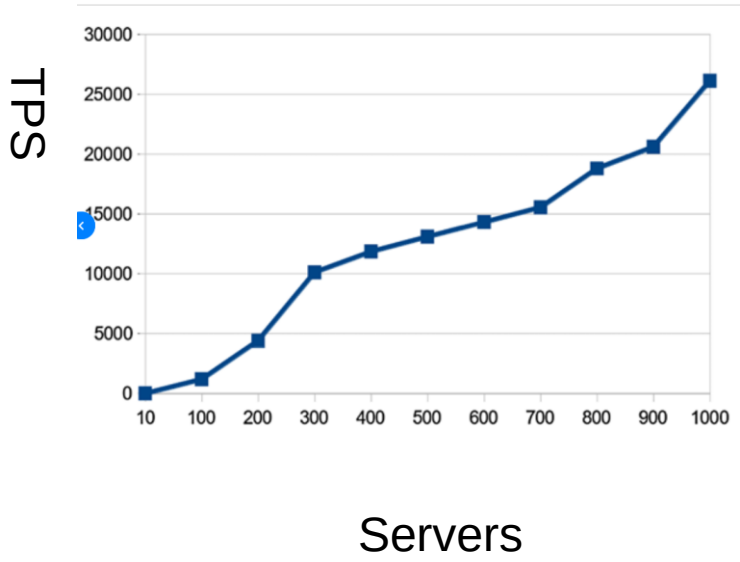
Robustness testing

- How sensitive is a system to erroneous inputs or changes in the operational environment?
  - Stupid/uncommon inputs (like testing special values in boundary testing)
  - Failures from other systems
  - Degraded node
  - Etc.

Scalability testing

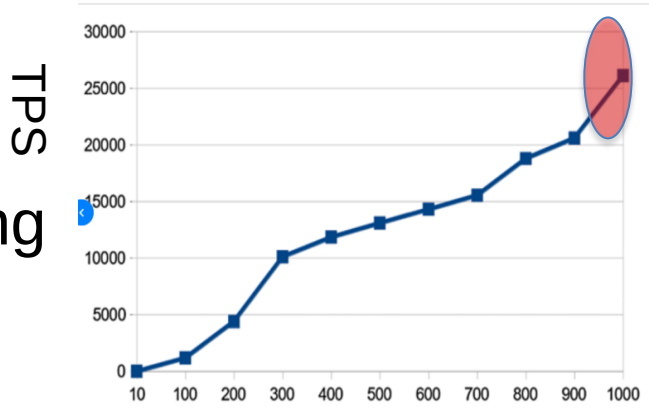
- To identify how well the system can scale, i.e., the magnitude of demand that can be placed on the system while continuing to meet performance requirements





Stress testing

- It can ensure the system can perform acceptably under the worse-case condition
- The system is deliberately stressed by pushing it to and beyond its specific limits for a while
- For testing
  - Buffer allocation and memory carving
  - Network bandwidth



CULTURE

Olympics ticket system crashes

Domestic ticket sales in China are halted after 8 million hits and 200,000 orders per second overwhelm the booking system.

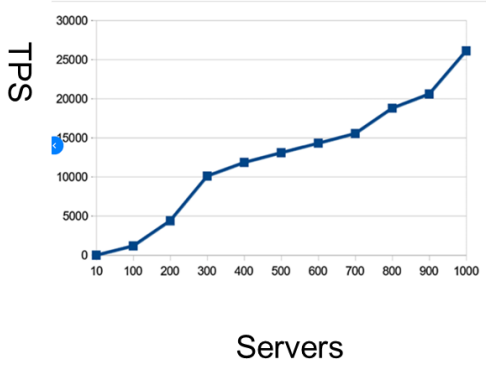
BY SUZANNE TINDAL | NOVEMBER 1, 2007 6:18 AM PDT

Load and stability testing

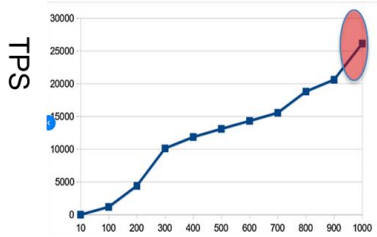
- Important for providing 24x7 services
- System and applications that run for months are likely to
  - Slow down
  - Encounter functionality problems
  - Silently failover
  - Crash altogether
- Ensure the system remains stable for a long period of time under full load

Scalability vs. Stress vs. Load and statbility testing

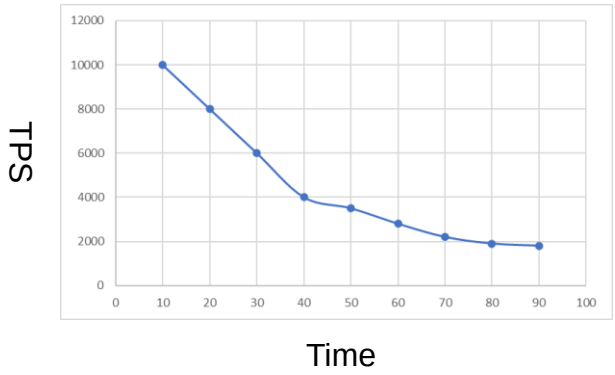
- Scalability



- Stress



- Load and stability

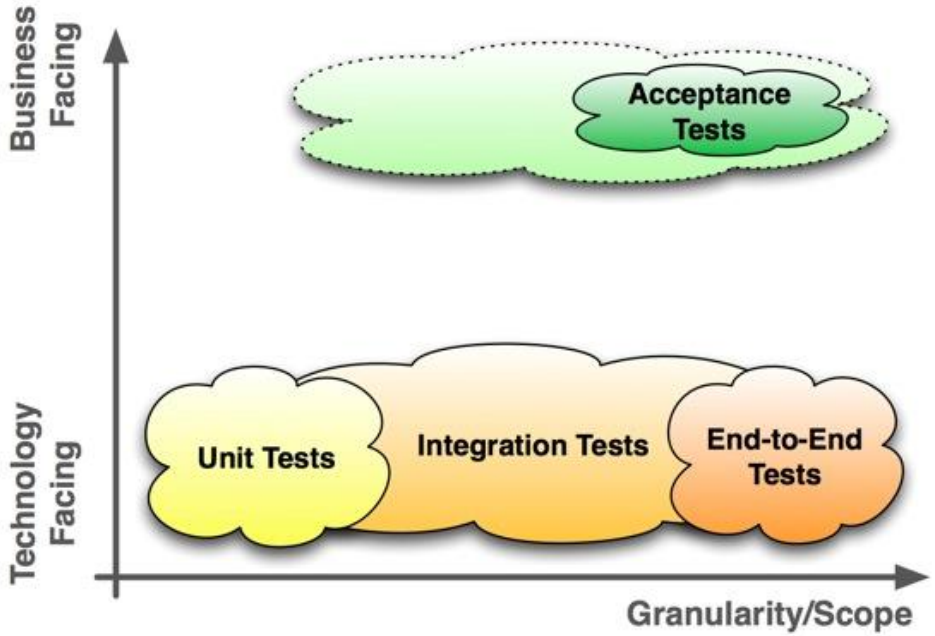


Using Apache JMeter to run load and performance testing

https://youtu.be/yNnbW2n9s8E

Acceptance testing

- ISTQB : **(user) acceptance testing**: Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.



Typical forms of acceptance testing

User acceptance testing

- Functions are correct?
- Test basis
  - User/business requirements
  - System requirements
  - Use cases
  - Business processes
  - Risk analysis reports

Operational acceptance test

- Read to operate?
  - Backup facilities
  - Procedures for disaster recovery
  - Training or manual for end-users
  - Maintenance procedures and manual
  - Security procedures

Contract and regulation acceptance testing

- Test against contract
- Test against regulations
  - Governmental regulation
  - Legal standards
  - Safety standards

Alpha and beta testing

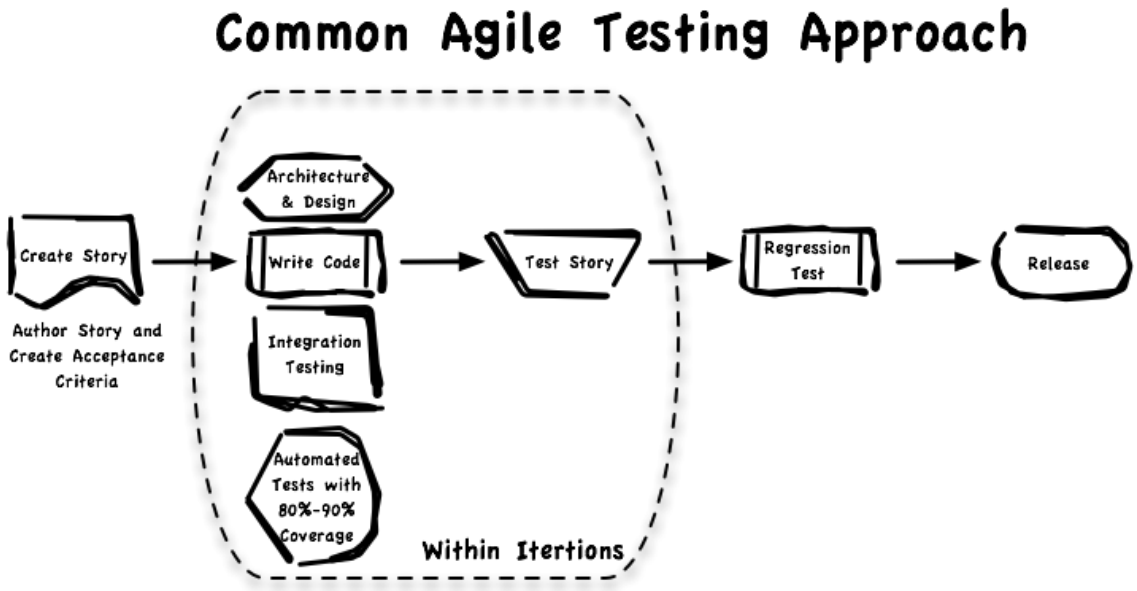
- Alpha
  - At developers' sites
  - By internal staff
  - Before it is released to external customers

- Beta (field testing)
  - At customers' sites
  - Before the system is released to other customers

Acceptance testing in agile

- The terms "functional test", "acceptance test" and "customer test" are used more or less interchangeably.

- Referring to user stories



Outline

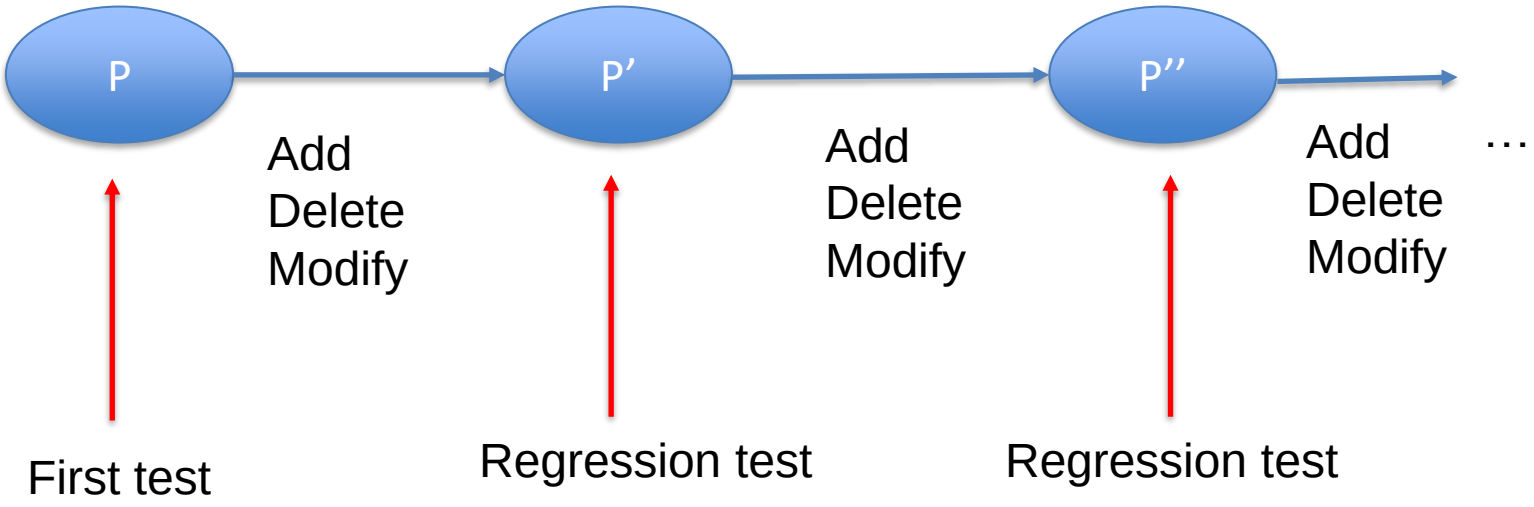
- Testing approaches
  - Integration tests
  - System tests
  - Acceptance tests
- ➢ Regression tests

Regression testing

- Basic regression testing concepts
- Regression test case selection
- Regression minimization
- Regression test case prioritization

What is regression testing?

- Constitute the vast majority of testing effort in many software development projects



Two types of regression testing

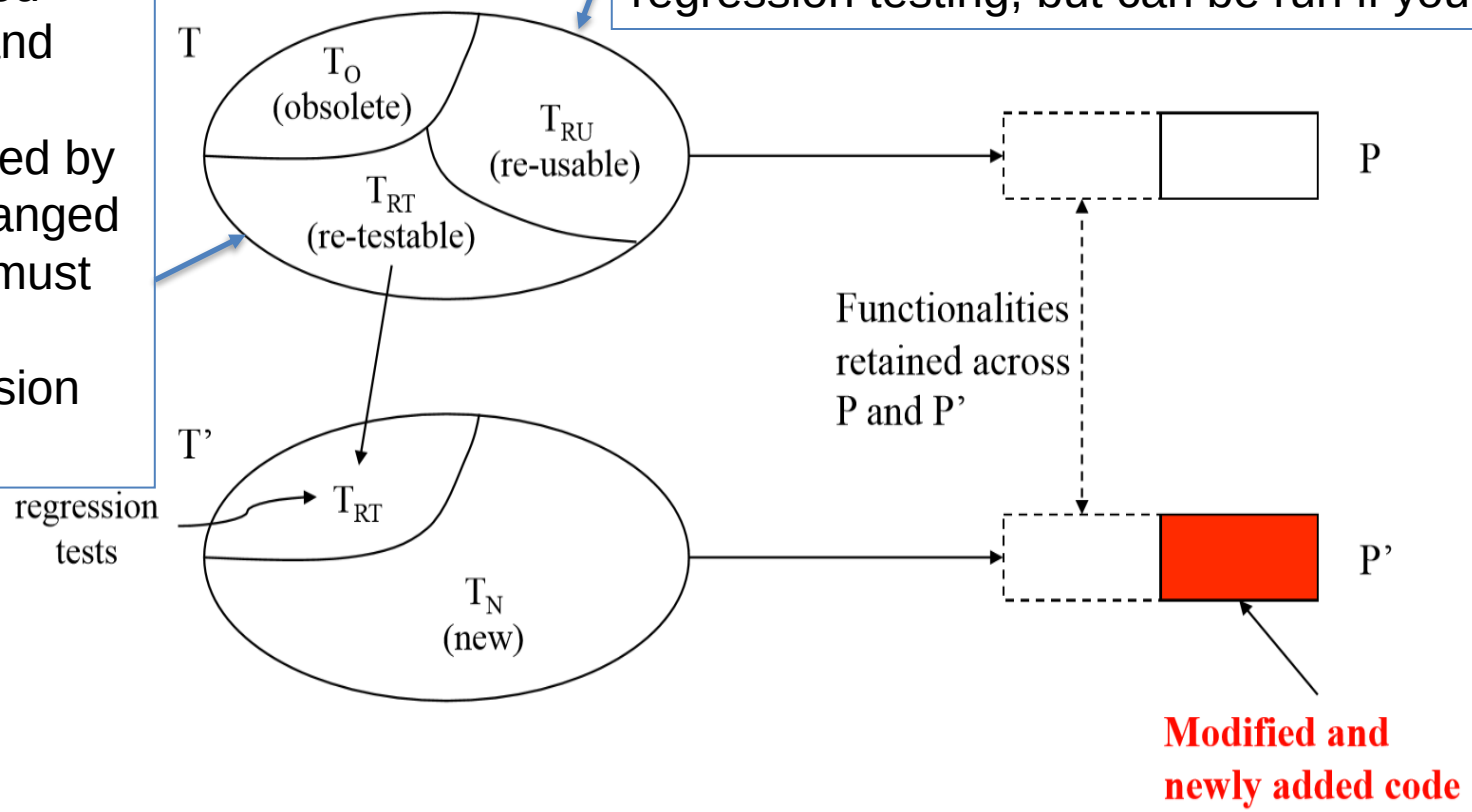
- Corrective regression testing
  - No requirements change
  - Modified code behaves correctly
  - Unchanged code continues to behave correctly

- Progressive regression testing
  - Requirements change
  - Newly added or modified code
  - Unchanged code continues to behave correctly

Overview of first & regression test cases*

Test changed code and code impacted by the changed code, must run in regression testing

Test code that is unchanged and not impacted by the changed code, unnecessary to run in regression testing, but can be run if you like



* https://www.uio.no/studier/emner/matnat/ifi/INF4290/v11/undervisningsmateriale/INF4290-RegTest.pdf

Regression testing processes

- Test revalidation
  - Are test cases obsolete?
- (Regression) test selection
  - Verify changed and impacted code (corrective)
  - Verify new structure and requirements (progressive)
- Test minimization
  - Remove redundant test cases
- Test prioritization
  - Rank test cases and run them according to available resources

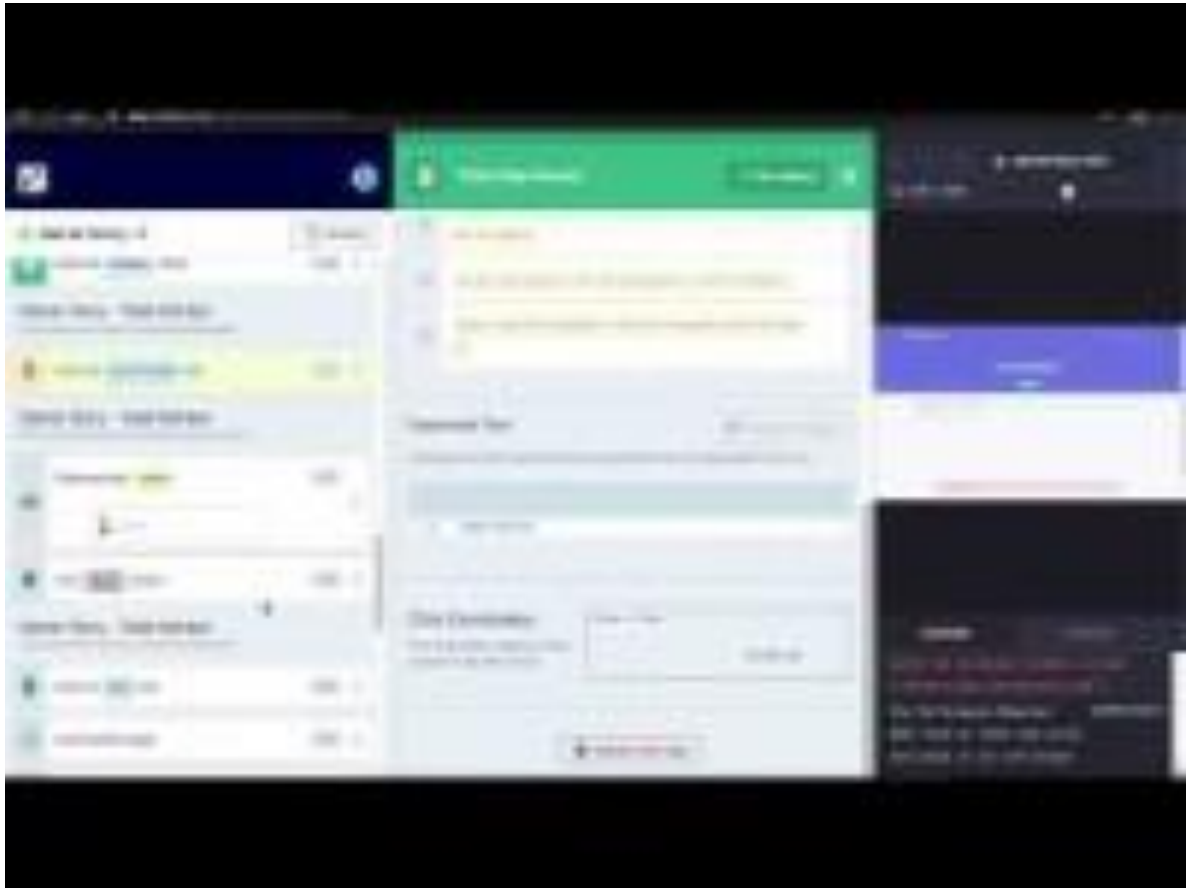
Test revalidation

- How can a test case be obsolete?
  - Their input/output relation is no longer correct due to changes in specification
  - They no longer test what they were designed to test due to modifications to the program
  - They are “structural” test cases that no longer contribute to structural coverage of the program

(Regression) test selection

- We now consider only corrective regression testing
- Strategies of (regression) test selection
  - Retest all
    - Can be costly, unless test execution is highly automated
  - Random selection
    - Better than no test
    - Hard to ensure coverage of the changed code
  - Selecting modification traversing tests
    - "Safe" regression test selection technique is preferred

Automated regression testing example



https://youtu.be/2jAy9cdblaE

Safe regression test selection

A technique that does not discard any test that will traverse a modified or impacted statement is known as "safe" regression test selection technique

- Assume program P has been tested by test set T against specification S
- Assume P is modified to P' to fix some bugs (i.e., specification S is not changed)

Question:

- What information is needed in order to select a safe regression test subset from T to verify P'?

General (regression) test selection process

- Establish traces between P and T
  - Execute P with test cases
  - Record program entities executed when running tests
- Compare P with P' to find differences
  - Identify program entities changed
- Select test cases from T that traverse the changed program and impacted entities

General (regression) test selection process (cont')

- Based on program entities traced and compared, the test selection methods can be classified into
  - Dynamic slicing based approach (statements)
  - Graph-walk approach (nodes in control flow graph/program dependence graph)
  - Firewall approach (OO classes, COTS components, etc.)
  - …

Graph-walk approach example

Code P

```
int  M (int x, int y){
  int z;
  if (x<y)
     z = f1 (x, y);
  else
     z = f2(x, y);
  return z;
}
```

```
int f1( int a, int b){
  if ((a+1) == b)
     return a*a;
  else
     return b*b; }
```

```
int f2( int a, int b){
  if (a == (b+1))
     return b*b;
  else
     return a*a; }
```

Code P’

If((a-1) == b)

Test set T of P

t1: <x = 1, y = 2>

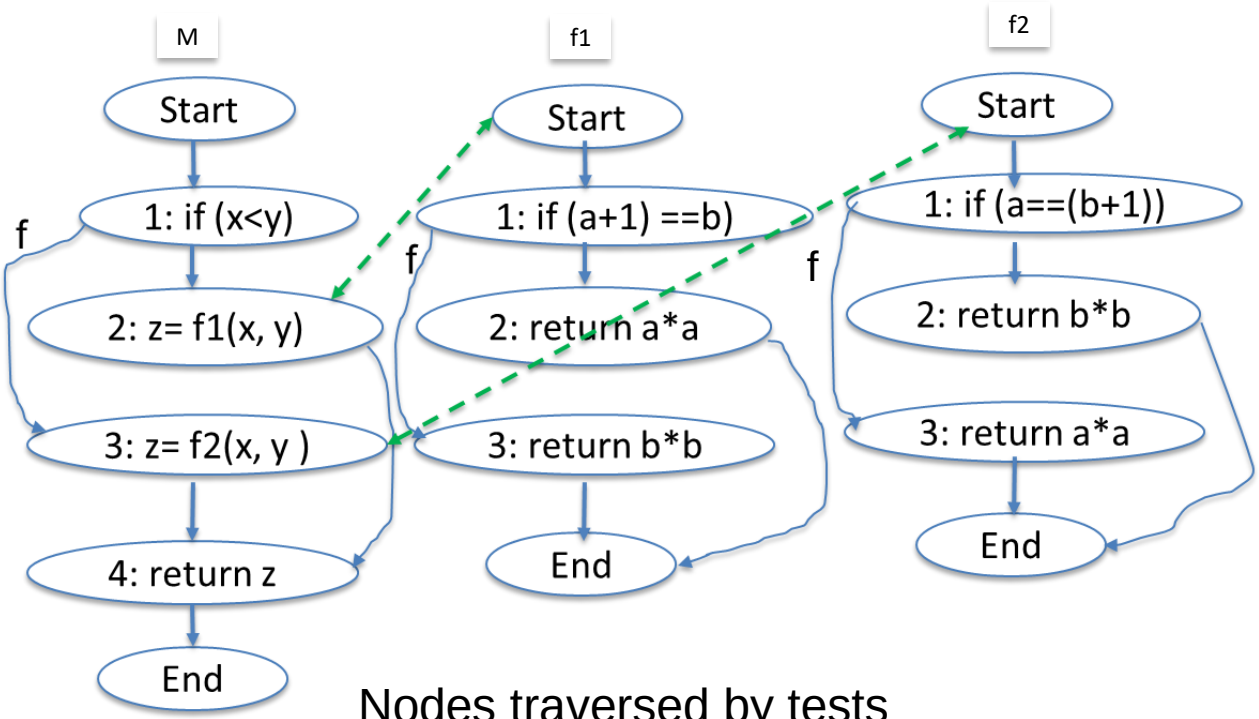
t2 : <x = 1, y = 3>

t3 : <x = 3, y = 1>

Which test cases to be included in the safe subset to test P’?

Graph-walk approach example – step 1

- Establish trace between test case and CFG (control flow graph) nodes



Test set T

t1: <x = 1, y = 2>

t2 : <x = 1, y = 3>

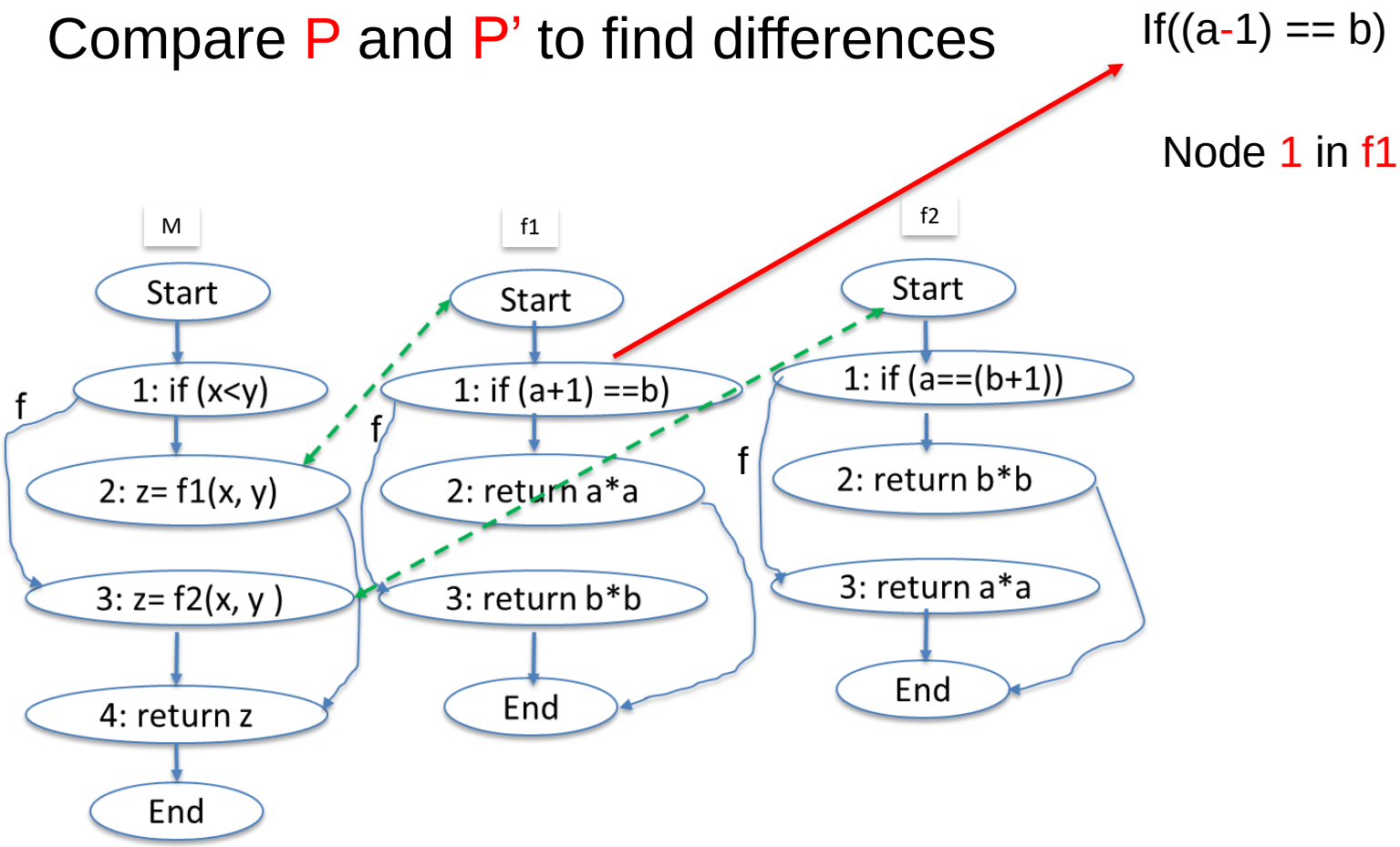
t3 : <x = 3, y = 1>

Nodes traversed by tests

Functions	Nodes			
	1	2	3	4
M	t1, t2, t3	t1, t2	t3	t1, t2, t3
f1	t1, t2	t1	t2	-
f2	t3	None	t3	-

Graph-walk approach example – step 2

- Compare P and P' to find differences



Graph-walk approach example – step 3

- Select test cases from T that traverse the changed CFG nodes

Test set T

t1: <x = 1; y = 2>

t2 : <x = 1, y = 3>

t3 : <x = 3, y = 1>

Set of tests that traverse a certain node

Functions	Nodes			
	1	2	3	4
M	t1, t2, t3	t1, t2	t3	t1,t2, t3
f1	t1, t2	t1	t2	-
f2	t3	None	t3	-

A technique that does not discard any test that will traverse a modified or impacted statement is known as “safe” regression test selection technique.

- Here, we did not cover the impacted statement
- Test oracles of the tests may need to be changed

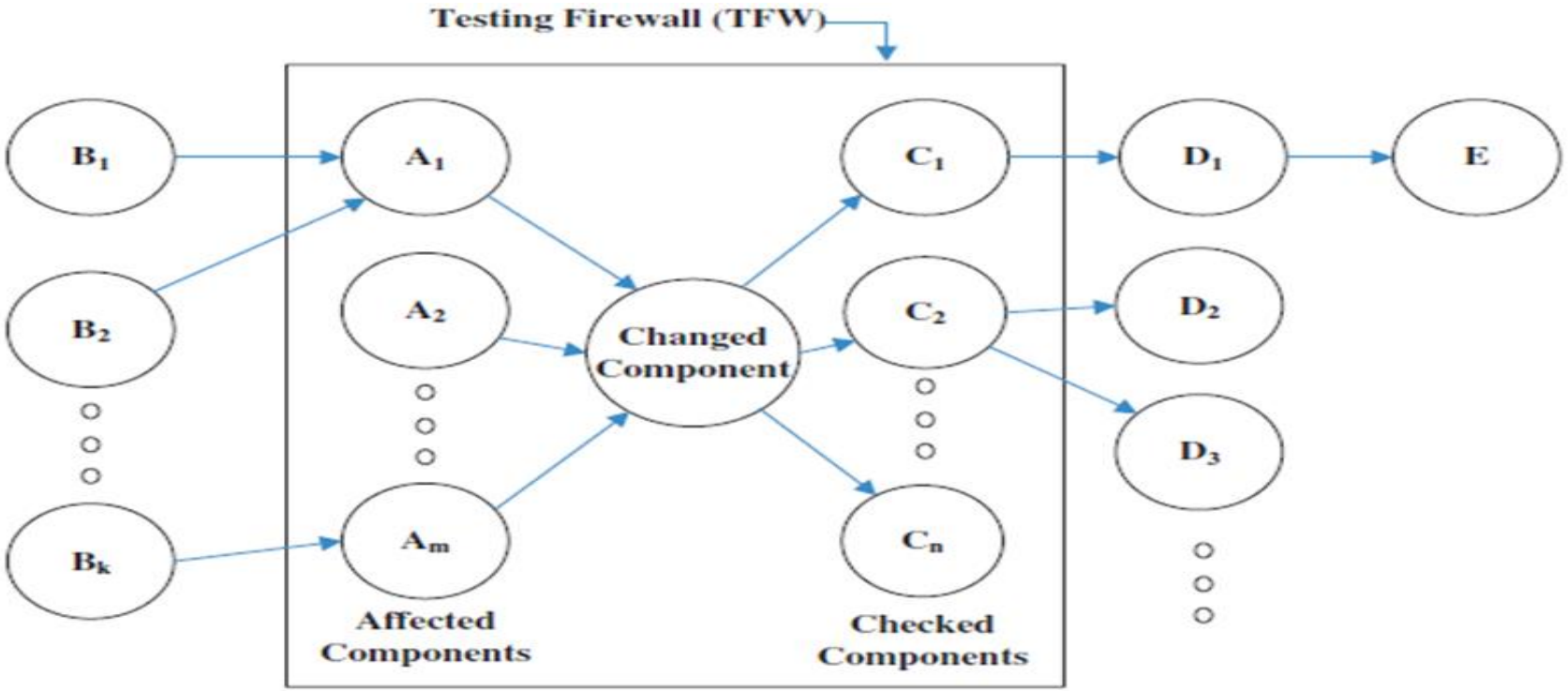
General (regression) test selection process (cont')

- Based on program entities traced and compared, the test selection methods can be classified into
  - Dynamic slicing based approach (statements)
  - Graph-walk approach (nodes in control flow graph/program dependence graph)
  - Firewall approach (OO classes, COTS components, etc.)
  - …

Firewall approach

What is a firewall?

- A ***firewall*** separates the classes that depend on the class that is changed from the rest of the classes



Firewall approach (cont')

Why use a firewall?

- The ***firewall*** approach is based on the first-level dependencies of modified components

- The most severe effects are in those components that send messages to the changed component or receive messages from the changed component

Process for determining “Firewall” in OO systems

1. Identify classes changed

2. Inheritance? Then descendants of the changed classes

3. Classes send messages to or receive messages from the changed class

One empirical study of using firewall approach [1]

- System studied
  - Distributed component-based J2EE sys. owned by Swedbank
  - 1.2M line of code from 27 000 classes
  - 2 new releases of the system

- Steps 1: Coverage analysis and establishing trace
  - Instrumentation of Java byte code and run-time collection of coverage data during original test
  - Used a tool called Emma [2]

One empirical study of using firewall approach [1] (Cont')

Step 2: Find classes in firewall

- Identify changes – produced MD5 signatures for each compiled class file. If code changes, class signature will be different.
- Own tool

Step 3: Extracting dependency between classes

- Dependency finder [3]

Step 3: Select test cases that traverse classes in firewall

Another empirical study of using firewall approach [4]

- Studied an OO telecommunication software system at ABB

- Compared two firewall approaches
  - One level from changed component
  - More than one level from the changed component, depending on their logical dependencies

Another empirical study of using firewall approach [4] (Cont')

- Observation
  - Extended firewall approach finds more faults than classical firewall approach, but more costly to run
- Recommendations
  - Routine incremental changes (one level approach)
  - Major release (> one level approach)

Test minimization

- To reduce redundancies in the safe subset
- If every entity covered by t2 is a subset of entities covered by another test case t1, then we remove t2
- The entities could be
  - Statement, CFG nodes, functions, etc.

Test minimization example

- t1 covers the following statement
- t2 also covers the same statement
- Then, we can remove t2

(a<b)||(a<c)

Risk of test minimization

- Test minimization is **risky** and is not necessarily safe
  - It depends on the modifications (from P to P'), the faults, the entities used

(a<b)||(a<c)

t2 may test the (a<c) part and t1 may test the (a<b) part

If the t2 is removed because this statement is covered by t1, then, we will miss the opportunity to test (a<c)

Test prioritization

- Different from test minimization

- Test prioritization
  - Is not going to remove any test cases from the safe subset
  - Is to rank regression tests based on some criteria

- The goal is to reveal faults early

Test prioritization approaches

- Unlike test prioritization of the first test
  - **More information available**
  - Fault finding effectiveness
  - Test coverage
  - Cost of running the test

- Prioritization strategies [5]
  - Coverage based (high coverage)
  - Cost-aware based (low cost to run, high fault-finding probability)
  - …

Summary

- We studied the following testing approaches
  - Integration
  - System
  - Acceptance

- We studied regression test
  - Selection
  - Minimization
  - Prioritization

References

- [1] White L., Abdullah K. A firewall approach for the regression testing of object-oriented software. Proc. of the Software Quality Week, 1997.
- [2] http://emma.sourceforge.net/
- [3] Skoglund M. and Runeson P. A Case study of The Class Firewall Regression Test Selection Technique on Large Scale Distributed Software System. Proc. of empirical software engineering conf. 2015.
- [4] White L. Jaber K. Robinson B. and Rajich V. Extended firewall for regression testing: an experience report. Journal of software maintenance and evolution. Vol. 20. 2008
- [5] Yoo S., Harman M. Regression testing minimization, selection and prioritization: a survey, Software testing, verification, and reliability, 2007.