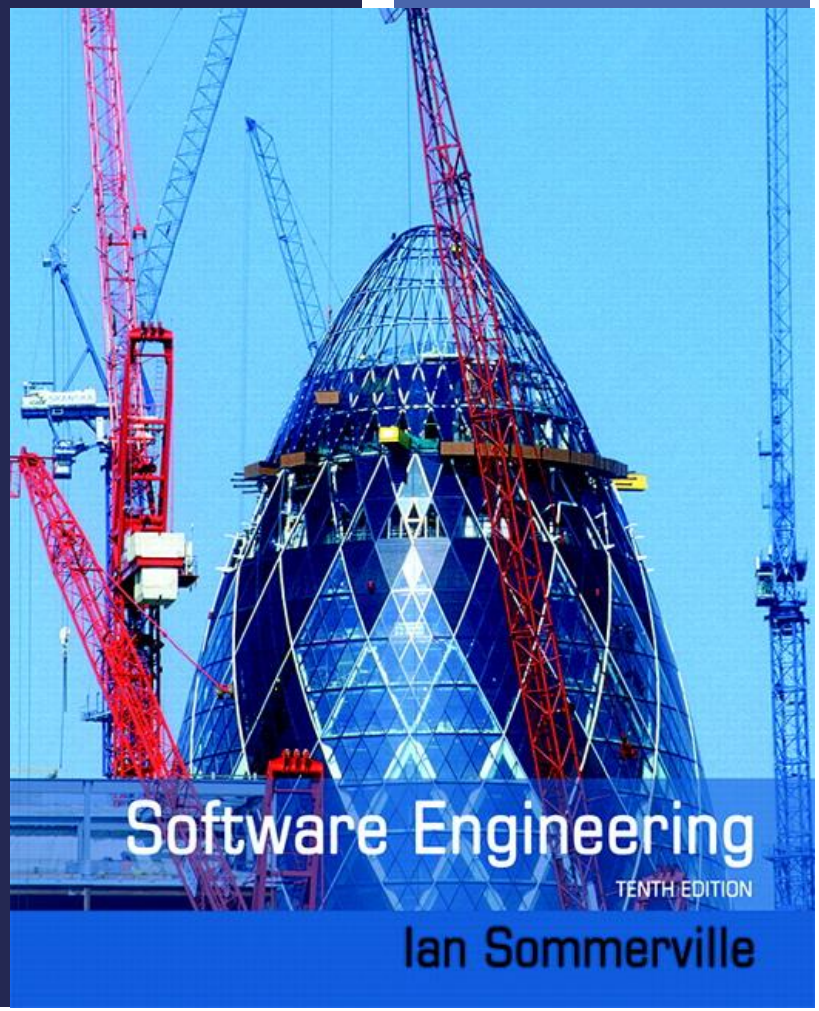
Software Engineering I

Chapters 8 & 9

SW Testing & Maintenance



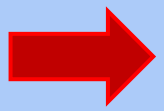
Chapter 8
Software Testing

Program testing

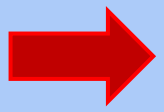
- Testing is intended to show that a program does what it is intended to do and to discover program defects before it is put into use.
- When you test software, you execute a program using artificial data.
- You check the results of the test run for errors, anomalies or information about the program's non-functional attributes.
- Can reveal the presence of errors NOT their absence.
- Testing is part of a more general verification and validation process, which also includes static validation techniques.

Program testing goals

- To demonstrate to the developer and the customer that the software meets its requirements.
  - For custom software, this means that there should be at least one test for every requirement in the requirements document.
  - For generic software products, it means that there should be tests for all of the system features, plus combinations of these features, that will be incorporated in the product release.
- To discover situations in which the behavior of the software is incorrect, undesirable or does not conform to its specification.
  - Defect testing is concerned with rooting out undesirable system behavior such as system crashes, unwanted interactions with other systems, incorrect computations and data corruption.

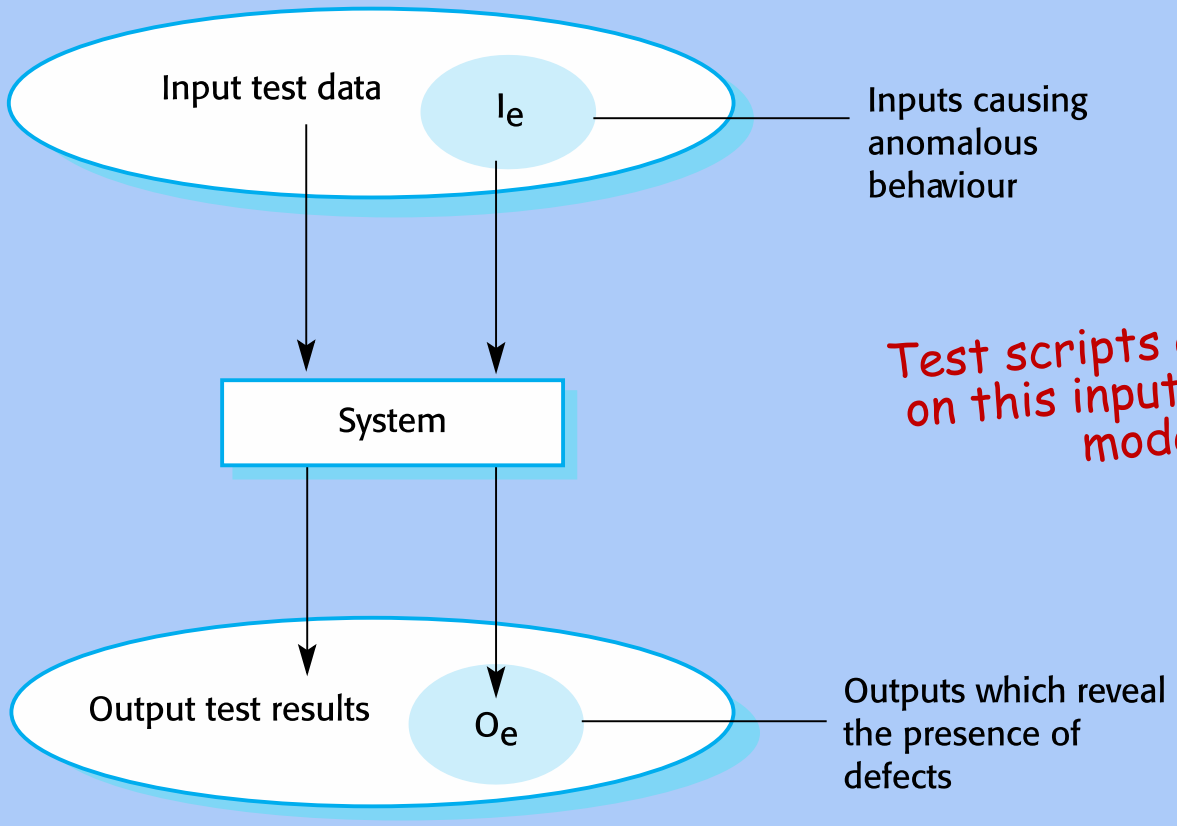


- Validation testing
  - You expect the system to perform correctly using a given set of test cases that reflect the system's expected use.



- Defect testing
  - The test cases are designed to expose defects.
  - The test cases in defect testing can be deliberately obscure and need not reflect how the system is normally used.

An input-output model of program testing



Test scripts are based on this input / output model

Verification vs validation

- **Verification**

  "Are we building the product right?"
  - The software should conform to its specification.

- **Validation**

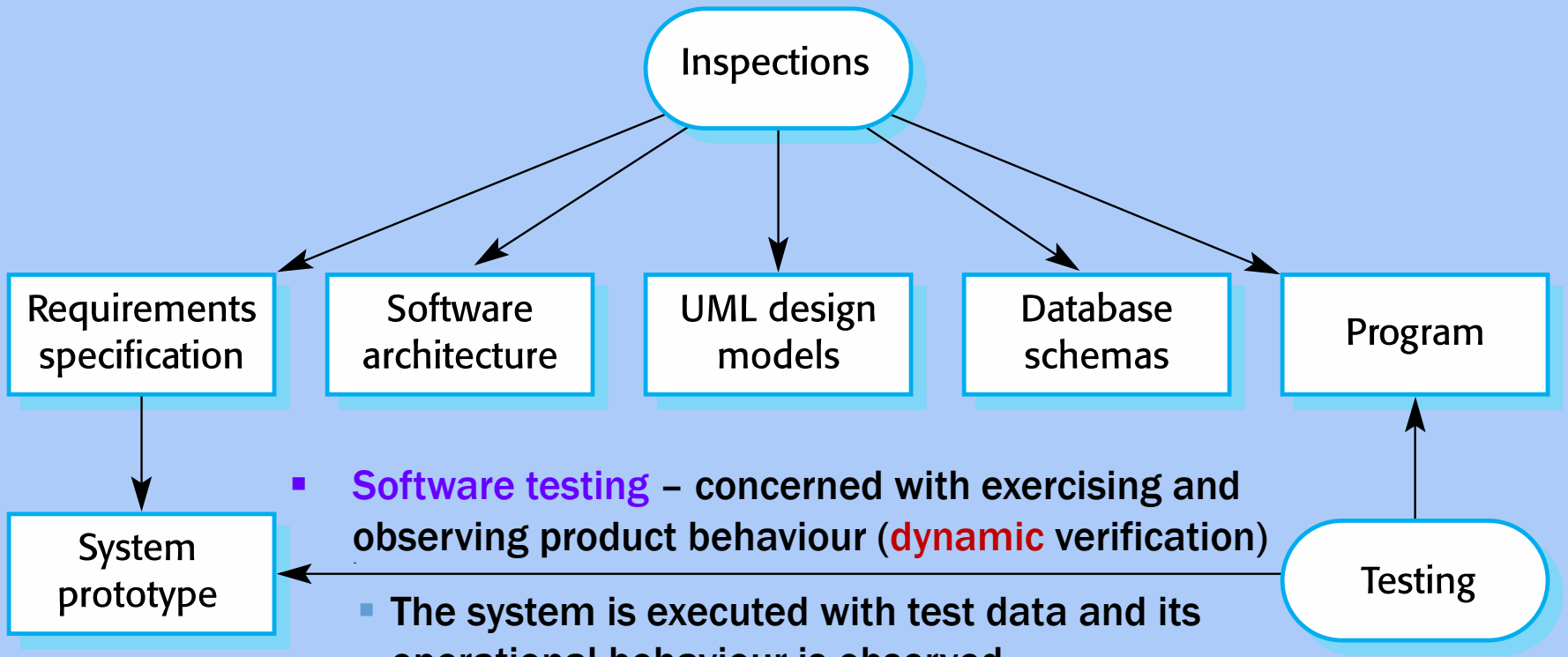
  "Are we building the right product?"
  - The software should do what the user really requires.

- Aim of V & V is to establish confidence that the system is 'fit for purpose'.

- Depends on system's purpose, user expectations and marketing environment
  - Software purpose
    - The level of confidence depends on how critical the software is to an organisation.
  - User expectations
    - Users may have low expectations of certain kinds of software.
  - Marketing environment
    - Getting a product to market early may be more important than finding defects in the program.

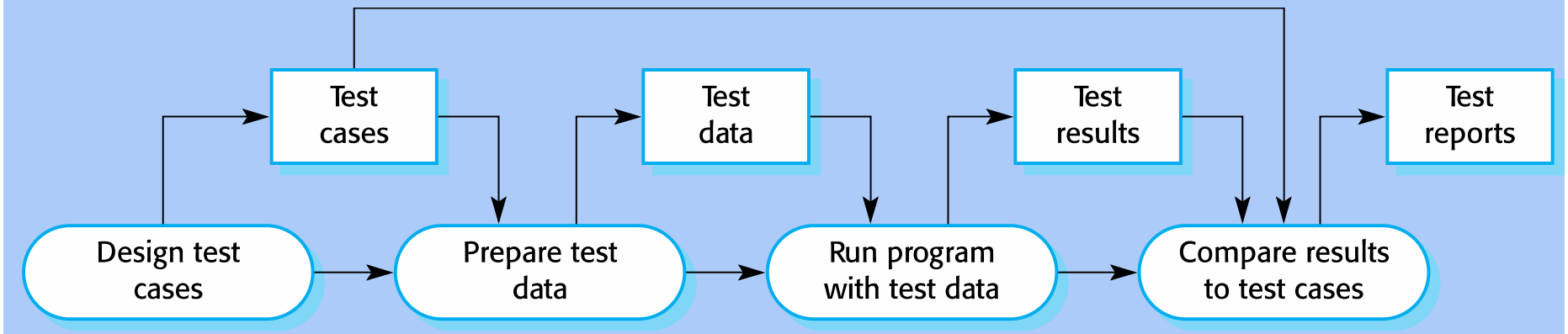
Inspections and testing

- **Software inspections** – concerned with analysis of the **static** system representation to discover problems (**static** verification)
  - May be supplemented by tool-based document and code analysis



- **Software testing** – concerned with exercising and observing product behaviour (**dynamic** verification)
  - The system is executed with test data and its operational behaviour is observed.

A model of the software testing process



Stages of testing

- Development testing, where the system is tested during development to discover bugs and defects.
- Release testing, where a separate testing team test a complete version of the system before it is released to users.
- User testing, where users or potential users of a system test the system in their own environment.

Types of testing

- Development testing includes all testing activities that are carried out by the team developing the system.
  - Unit testing, where individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.
  - Integration testing or Component testing (per textbook), where several individual units are integrated to create composite components. Component testing should focus on testing component interfaces.
  - System testing, where some or all of the components in a system are integrated and the system is tested as a whole. System testing should focus on testing component interactions.

Unit testing

Unit testing

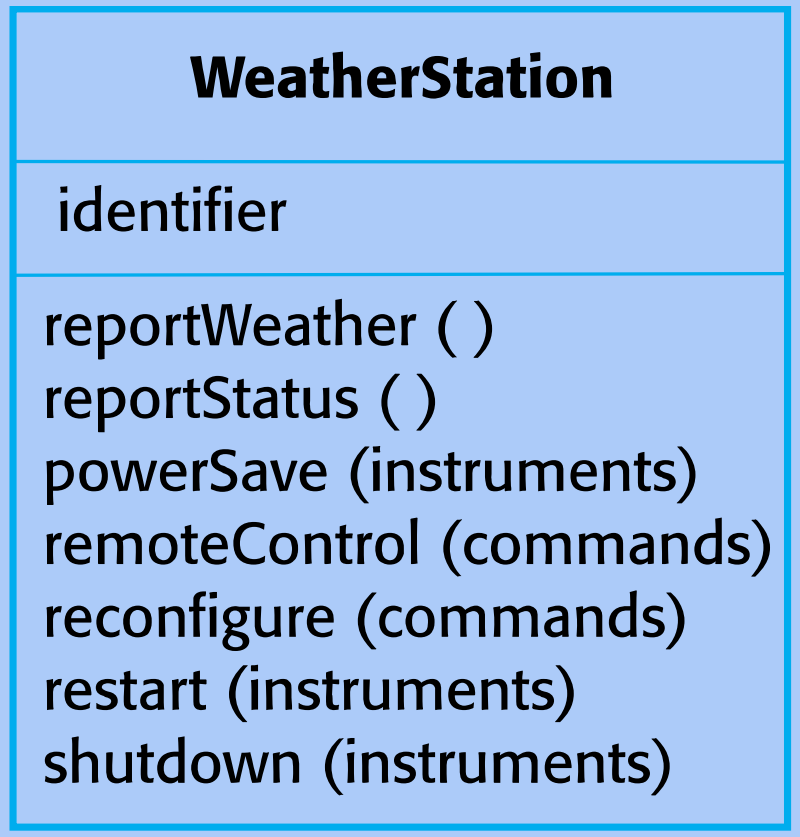
Unit testing

- Unit testing is the process of testing individual components in isolation.
- It is a defect testing process.
- Units may be:
  - Individual functions or methods within an object
  - Object classes with several attributes and methods
  - Composite components with defined interfaces used to access their functionality.

Object testing

- Complete test coverage of a class (a unit) involves
  - Testing all operations associated with an object
  - Setting and interrogating all object attributes
  - Exercising the object in all possible states.
- Inheritance makes it more difficult to design object class tests as the information to be tested is not localised.
  - Can a class be tested in isolation of other classes?
  - Unit testing is more challenging

The weather station object interface
("interface" as public methods or hooks)



- Need to define test cases for reportWeather(), reconfigure() and other methods.
- Using a state model, identify sequences of state transitions to be tested and the event sequences to cause these transitions
- For example:
  - Shutdown -> Running-> Shutdown
  - Configuring-> Running-> Testing -> Transmitting -> Running
  - Running-> Collecting-> Running-> Summarizing -> Transmitting -> Running

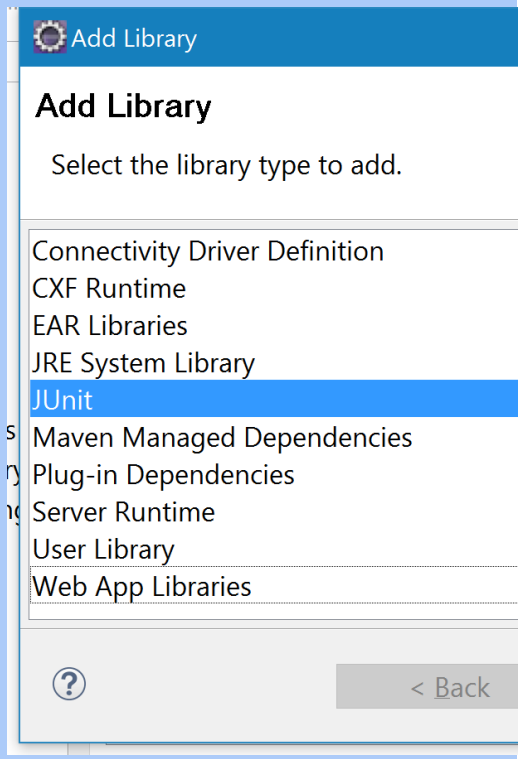
Automated testing

- Whenever possible, unit testing should be automated so that tests are run and checked without manual intervention.
- In automated unit testing, you make use of a test automation framework (such as JUnit) to write and run your program tests.
- Unit testing frameworks provide generic test classes that you extend to create specific test cases. They can then run all of the tests that you have implemented and report, often through some GUI, on the success of otherwise of the tests.

Automated test components

- A setup part, where you initialize the system with the test case, namely the inputs and expected outputs.
- A call part, where you call the object or method to be tested.
- An assertion part where you compare the result of the call with the expected result. If the assertion evaluates to true, the test has been successful if false, then it has failed.

- Setup Junit in Eclipse



JUnit Setup (1)

```
// Constructuor in Creature class
public Creature(String name, double height, String color) {
    this.name = name;
    this.height = height;
    this.color = color;
}

// Set name in Creature class
public void setName(String name) {
    this.name = name;
}

// Set name in Dragon class
@Override
public void setName(String name) {
    if (name.length() < 9) {
        this.name = "Dragon " + name;
    }
    else super.setName(name);
}
```

- Constructor of Creature as well as setName() sets the name of the creature to whatever parameter is passed.
- In the Dragon class, however, the setter is overridden and may set the dragon creature's name differently
  - Dragon constructor calls super(), setting the name to the passed parameter

Conflict!

  - Dragon setter may conditionally set the name using a different convention.

JUnit Setup (2)

```
// Constructuor in Creature class
public Creature(String name, double height, String color) {
   this.name = name;
   this.height = height;
   this.color = color;
}

// Set name in Creature class
public void setName(String name) {
   this.name = name;
}

// Set name in Dragon class
@Override
public void setName(String name) {
   if (name.length() < 9) {
      this.name = "Dragon " + name;
   }
   else super.setName(name);
}
```

- However, since most dragons have long names, this bug might never be caught!

- Dragon constructor calls super(), setting the name to the passed parameter

Conflict!

- Dragon setter may conditionally set the name using a different convention.

A method in a unit test case class, DragonTester

One of our test dragons has an unusually short name!

```
public void testSetters() {

  // Keep some test case dragons in an ArrayList

  // But first... let's get some instance variable values
  String[] names =    { "Deathwing", "Ysera", "Alexstrasza", "Saragosa" };
  double[] heights =  { 84.5, 72.0, 80.0, -71.0 };
  String[] colors =   { "black", "green", "red", "blue" };
  boolean[] fire =    { true, false, false, true };
  boolean[] canRide = { false, true, false, true };

  ArrayList<Dragon> testDragons = new ArrayList<Dragon>();
  for (int i = 0; i < names.length; i++) {
     testDragons.add(new Dragon(names[i], heights[i],
                                colors[i], fire[i], canRide[i]));
  }

  // Select a random Dragon to test
  Random random = new Random();
  int i = random.nextInt(names.length);

  Dragon testDragon = testDragons.get(i);
  System.out.println("Testing dragon " + i + ": " + testDragon);
```

A method in a unit test case class, DragonTester

```
public void testSetters() {
  ...

  // Test the setColor method
  String dragonColor = testDragon.getColor();  // color set by constructor
  testDragon.setColor(colors[i]);
  /* If we use setter to set the color that was set in the constructor,
     the colors should be equal                                          */
  assertEquals(dragonColor, testDragon.getColor());

 // Test the setName method
  String dragonName = testDragon.getName();  // name set by constructor
  testDragon.setName(names[i]);
  /* If we use setter to set the name that was set in the constructor,
     the names should be equal                                          */
  assertEquals(dragonName, testDragon.getName());
```

And a Driver to run the Unit Test classes

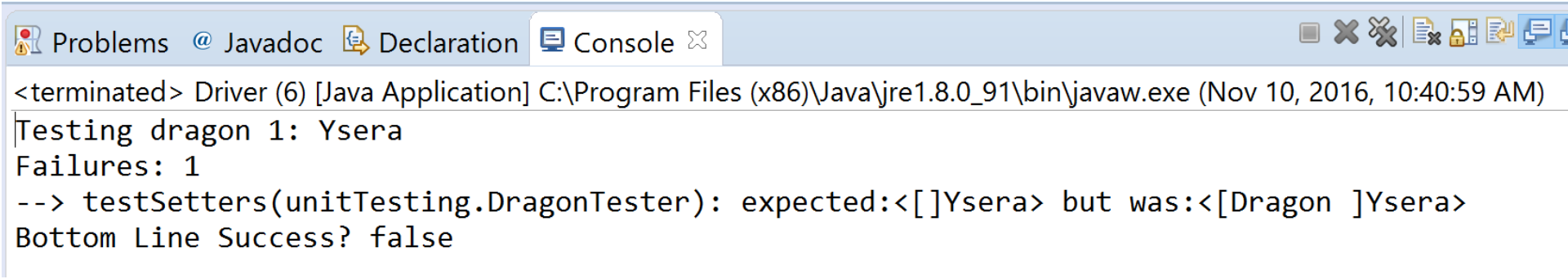
```
public class Driver {

   public static void main(String[] args) {

      Result result = JUnitCore.runClasses(DragonTester.class);
      System.out.println("Failures: " + result.getFailureCount());

      for (Failure failure : result.getFailures()) {
         System.out.println("--> " + failure);
      }
      System. out.println("Bottom Line Success? " +
      result.wasSuccessful());
   }
}
```

Run your unit test classes



Choosing unit test cases

- The test cases should show that, when used as expected, the component that you are testing does what it is supposed to do.
- If there are defects in the component, these should be revealed by test cases.
- This leads to 2 types of unit test case:
  - The first of these should reflect normal operation of a program and should show that the component works as expected.
  - The other kind of test case should be based on testing experience of where common problems arise. It should use abnormal inputs to check that these are properly processed and do not crash the component.

Equivalence partitions

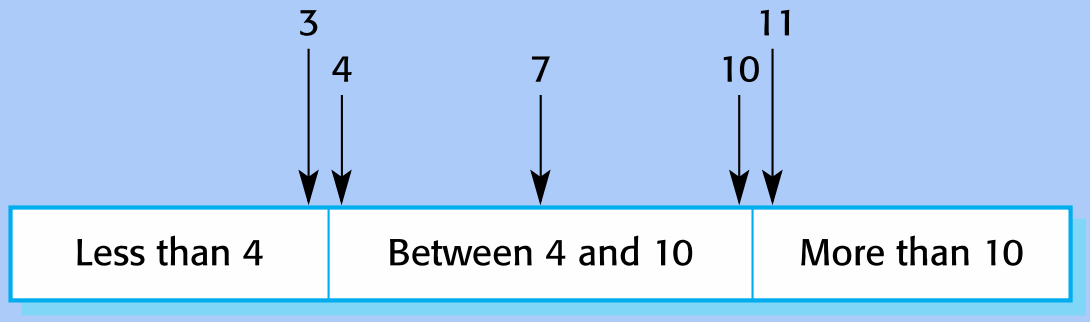
Equivalence partitioning or **equivalence** class **partitioning** (ECP) is a software testing technique that divides the input data of a software unit into **partitions** of equivalent data from which test cases can be derived.

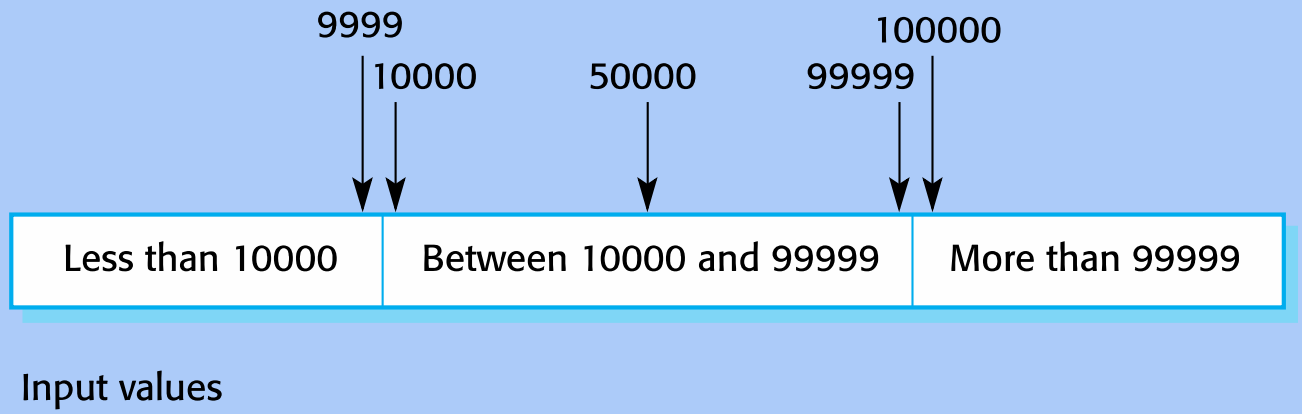
Example:

A program reads a file that contains between four and ten 5-digit numbers.

Boundary test cases
- **3, 4, 10, 11**
- **9999, 10000**

Midpoint test cases
- **7**
- **50000**





General testing guidelines

- Collections
  - Test software collections which have only a single value.
  - Use collections of different sizes in different tests.
  - Derive tests so that the first, middle and last elements of the collection are accessed.
  - Test with collections of zero length.
- Choose inputs that force the system to generate all error messages
- Design inputs that cause input buffers to overflow
- Repeat the same input or series of inputs numerous times
- Force invalid outputs to be generated
- Force computation results to be too large or too small.

General testing guidelines

- Collections
  - Test software collections which have only a single value.
  - Use collections of different sizes in different tests.
  - Derive tests so that the first, middle and last elements of the collection are accessed.
  - Test with collections of zero length.
- Choose inputs that force the system to generate all error messages
- Design inputs that cause input buffers to overflow
- Repeat the same input or series of inputs numerous times
- Force invalid outputs to be generated
- Force computation results to be too large or too small.

Integration testing

Interface testing

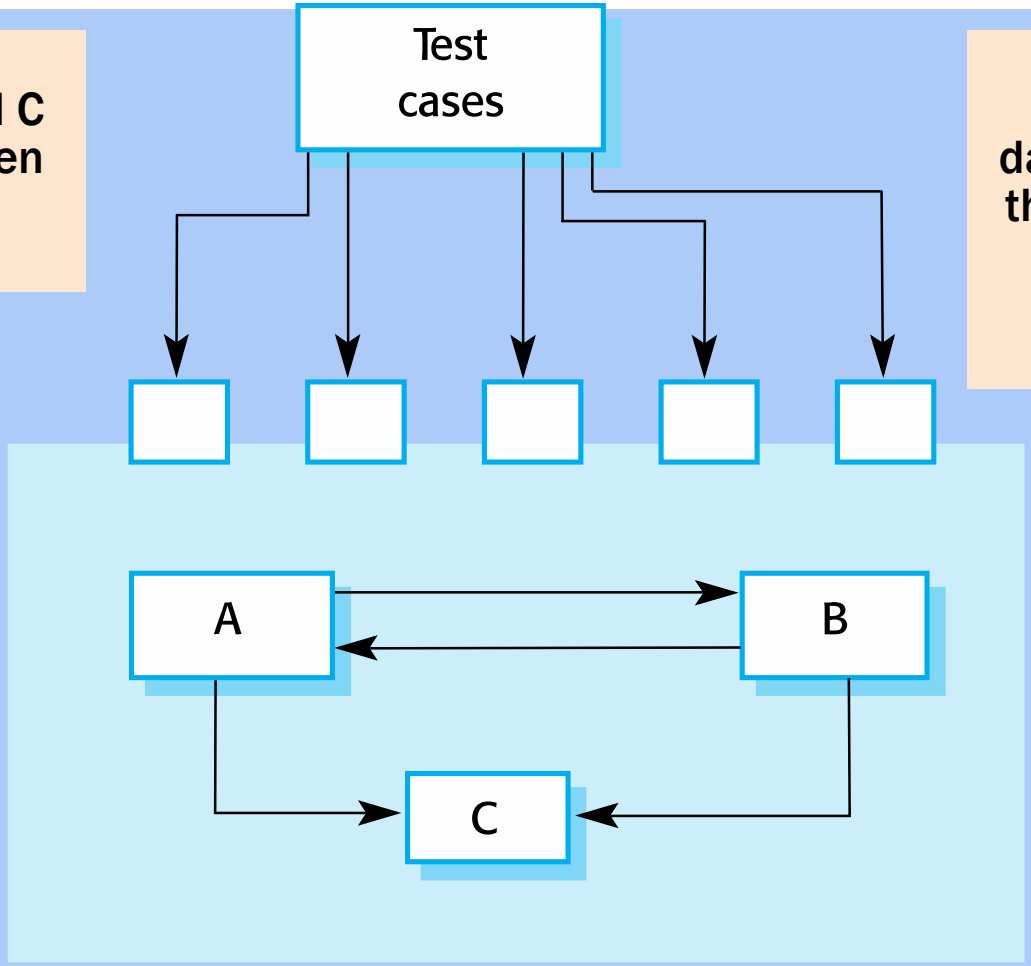
- Software components are often composite components that are made up of several interacting objects.
  - For example, in the weather station system, the reconfiguration component includes objects that deal with each aspect of the reconfiguration.
- You access the functionality of these objects through the defined component interface (aka API, aka exposed methods or functions)
- Testing composite components should therefore focus on showing that the component interface behaves according to its specification.
  - You can assume that unit tests on the individual objects within the component have been completed.

Interface testing

- Objectives are to detect faults due to interface errors or invalid assumptions about interfaces.
- Interface types
  - **Parameter interfaces**
    Data passed from one method or procedure to another.
  - **Shared memory interfaces**
    Data placed into memory by one subsystem and retrieved from there by another subsystem. (Frequent paradigm of embedded systems.)
  - **Procedural interfaces**
    Sub-system encapsulates a set of procedures to be called by other sub-systems. (e.g., PHP web form invoking a stored database procedure)
  - **Message passing interfaces**
    Sub-systems request services from other sub-systems (e.g., web services, JSON feeds)

Interface testing

Objects A, B and C have already been unit tested.



Test cases have data appropriate to the functionality of A, B and C as a subsystem

Focus on inter-object code, not intra-object code

Interface errors

- Interface misuse
  - A calling component calls another component and makes an error in its use of its interface e.g. parameters in the wrong order.
- Interface misunderstanding
  - A calling component embeds assumptions about the behaviour of the called component which are incorrect.
- Timing errors
  - The called and the calling component operate at different speeds and out-of-date information is accessed.

Interface testing guidelines

- Design tests so that parameters to a called procedure are at the extreme ends of their ranges.
- Always test pointer parameters with null pointers.
- Design tests which cause the component to fail.
- Use stress testing in message passing systems.
- In shared memory systems, vary the order in which components are activated.

System testing

System testing

- System testing during development involves integrating components to create a version of the system and then testing the integrated system.
- The focus in system testing is testing the interactions between components.
- System testing checks that components are compatible, interact correctly and transfer the right data at the right time across their interfaces.
- System testing tests the emergent behaviour of a system.

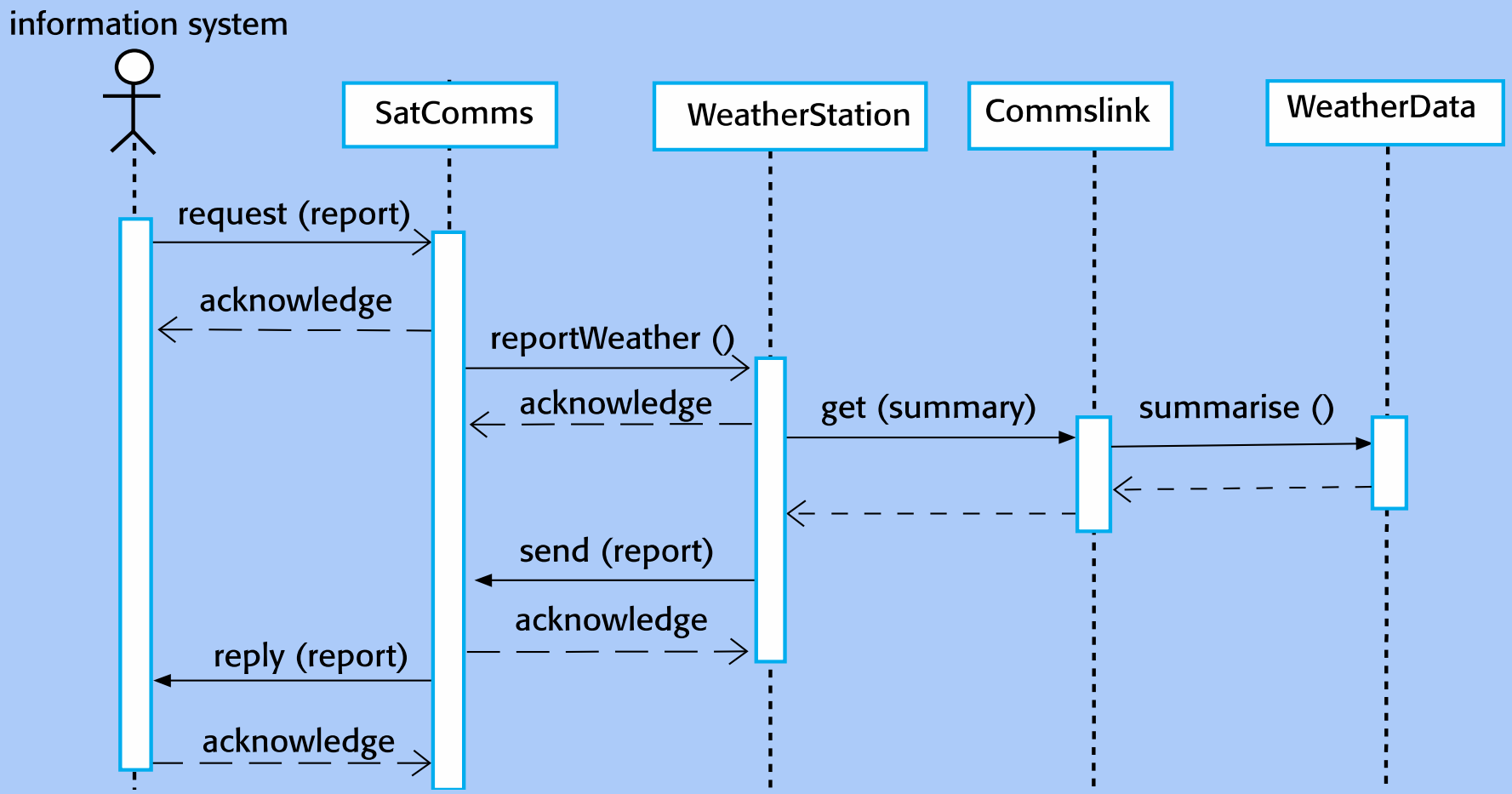
System and component testing

- During system testing, reusable components that have been separately developed and off-the-shelf systems may be integrated with newly developed components. The complete system is then tested.
- Components developed by different team members or sub-teams may be integrated at this stage. System testing is a collective rather than an individual process.
  - In some companies, system testing may involve a separate testing team with no involvement from designers and programmers.

Use-case testing

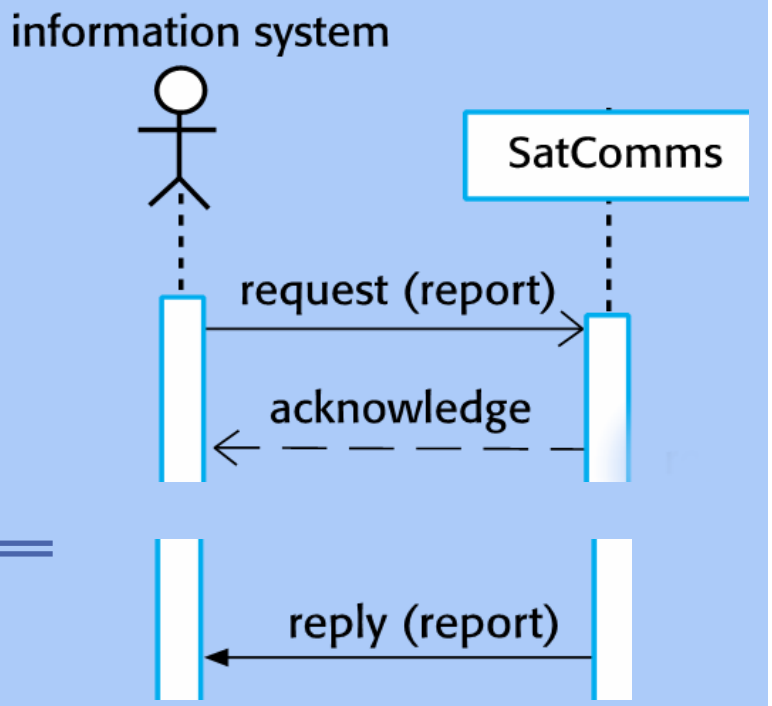
- The use-cases developed to identify system interactions can be used as a basis for system testing.
- Each use case usually involves several system components so testing the use case forces these interactions to occur.
- The sequence diagrams associated with the use case documents the components and interactions that are being tested.

Collect weather data sequence chart



Test cases derived from sequence diagram

- An input of a request for a report should have an associated acknowledgement.
- A report should ultimately be returned from the request.
  - You should create summarized data that can be used to check that the report is correctly organized and formatted. Expected Results



- An input request for a report to WeatherStation results in a summarized report being generated.
  - Can be tested by creating raw data corresponding to the summary that you have prepared for the test of SatComms and checking that the WeatherStation object correctly produces this summary. This raw data is also used to test the WeatherData object. Expected Results

Testing policies

- Exhaustive system testing is impossible so testing policies which define the required system test coverage may be developed.
- Examples of testing policies:
  - All system functions that are accessed through menus should be tested.
  - Combinations of functions (e.g. text formatting) that are accessed through the same menu must be tested.
  - Where user input is provided, all functions must be tested with both correct and incorrect input.

Test Scripts

System Test Case:
GOOGLE MAPS
USER STORY U2: Get Directions to a restaurant

Purpose: Verify the user story U2 (all parts).

Instructions: Anything in **RED** is mandatory, everything else is optional and should only be put in if it is needed to clarify how the test was performed.

Test Run Information: **Tester Name:** **Date(s) of Test:** Location/server being used: Google Maps Test Server A3	**Prerequisites for this test:** None **Software Versions:** Application: Google Maps beta 0.91 Browser [used & those COTS supports]: Safari v.2.1.4 Database: N/A Operating System: Mac OS10.4 **Required Configuration:** [browser setup, security or user ID roles] No special setup needed
NOTES and RESULTS:	

Inputs to each test step

Can include data to use for a test step (e.g. step 5) or can refer user to a document containing test data.

TEST SCRIPT STEPS/RESULTS

STEP	TEST STEP/INPUT	EXPECTED RESULTS	ACTUAL RESULTS	Requirements Validated	PAS S/FA IL
Get directions to a restaurant present in the database – Reqmts Validated: 3.7.1, 3.7.2, 3.7.5, 4.1-4.8					
1.	Enter restaurant name in the search window (e.g. McDonalds) that is present in the database	Able to enter text			
2.	Press the search button	Multiple results returned and pins are displayed on the map			
3.	Click on a single location	Map centers on that location and an information popup displays reviews and address, and link to get directions			
4.	Click on "get directions" link	Start address box appears			
5.	Type in a start address for your home.	Able to enter text			
6.	Press enter	Map shows a line from your home to restaurant. Directions in text are also displayed			
7.	Repeat steps, replacing step 6 with: Using mouse, press "Go" button	Same results as step 6		R3.7.56	
8.	Repeat steps 1-6 replacing start address with only city.	Directions are shown as in step 6, but only from the center of the city to the location		R3.7.55, R3.7.59	

System Test Case:
GOOGLE MAPS
USER STORY U2: Get Directions to a restaurant

Purpose: Verify the user story U2 (all parts).

Instructions: Anything in **RED** is mandatory, everything else is optional and should only be put in if it is needed to clarify how the test was performed.

Test Run Information: **Tester Name:** **Date(s) of Test:** Location/server being used: Google Maps Test Server A3	**Prerequisites for this test:** None **Software Versions:** Application: Google Maps beta 0.91 Browser [used & those COTS supports]: Safari v.2.1.4 Database: N/A Operating System: Mac OS10.4 **Required Configuration:** [browser setup, security or user ID roles] No special setup needed

NOTES and RESULTS:

Expected results for each step

Expected results may be listed here or may refer to an external document.

TEST SCRIPT STEPS/RESULTS					
STEP	**TEST STEP/INPUT**	**EXPECTED RESULTS**	**ACTUAL RESULTS**	**Requirements Validated**	**PASS/FAIL**
Get directions to a restaurant present in the database – Reqmts Validated: 3.7.1, 3.7.2, 3.7.5, 4.1-4.8					
1.	Enter restaurant name in the search window (e.g. McDonalds) that is present in the database	Able to enter text			
2.	Press the search button	Multiple results returned and pins are displayed on the map			
3.	Click on a single location	Map centers on that location and an information popup displays reviews and address, and link to get directions			
4.	Click on “get directions” link	Start address box appears			
5.	Type in a start address for your home.	Able to enter text			
6.	Press enter	Map shows a line from your home to restaurant. Directions in text are also displayed			
7.	Repeat steps, replacing step 6 with: Using mouse, press “Go” button	Same results as step 6		R3.7.56	
8.	Repeat steps 1-6 replacing start address with only city.	Directions are shown as in step 6, but only from the center of the city to the location		R3.7.55, R3.7.59	

System Test Case:
GOOGLE MAPS
USER STORY U2: Get Directions to a restaurant

Purpose: Verify the user story U2 (all parts).

Instructions: Anything in **RED** is mandatory, everything else is optional and should only be put in if it is needed to clarify how the test was performed.

Test Run Information: **Tester Name:** **Date(s) of Test:** Location/server being used: Google Maps Test Server A3	**Prerequisites for this test:** None **Software Versions:** Application: Google Maps beta 0.91 Browser [used & those COTS supports]: Safari v.2.1.4 Database: N/A Operating System: Mac OS10.4 **Required Configuration:** [browser setup, security or user ID roles] No special setup needed

NOTES and RESULTS:

Actual results for each step

Actual results may be recorded or described here.

TEST SCRIPT STEPS/RESULTS					
STEP	**TEST STEP/INPUT**	**EXPECTED RESULTS**	**ACTUAL RESULTS**	**Requirements Validated**	**PAS S/FA IL**
Get directions to a restaurant present in the database – Reqmts Validated: 3.7.1, 3.7.2, 3.7.5, 4.1-4.8					
1.	Enter restaurant name in the search window (e.g. McDonalds) that is present in the database	Able to enter text			
2.	Press the search button	Multiple results returned and pins are displayed on the map			
3.	Click on a single location	Map centers on that location and an information popup displays reviews and address, and link to get directions			
4.	Click on "get directions" link	Start address box appears			
5.	Type in a start address for your home.	Able to enter text			
6.	Press enter	Map shows a line from your home to restaurant. Directions in text are also displayed			
7.	Repeat steps, replacing step 6 with: Using mouse, press "Go" button	Same results as step 6			
8.	Repeat steps 1-6 replacing start address with only city.	Directions are shown as in step 6, but only from the center of the city to the location			

Highly validated systems may require another column to record initials of tester for each step – or may require tester to sign each page of testing document.

Screen shots may be required for critical steps such as step 6

System Test Case:
GOOGLE MAPS
USER STORY U2: Get Directions to a restaurant

Purpose: Verify the user story U2 (all parts).

Instructions: Anything in **RED** is mandatory, everything else is optional and should only be put in if it is needed to clarify how the test was performed.

Test Run Information: **Tester Name:** **Date(s) of Test:** Location/server being used: Google Maps Test Server A3	**Prerequisites for this test:** None **Software Versions:** Application: Google Maps beta 0.91 Browser [used & those COTS supports]: Safari v.2.1.4 Database: N/A Operating System: Mac OS10.4 **Required Configuration:** [browser setup, security or user ID roles] No special setup needed

NOTES and RESULTS:

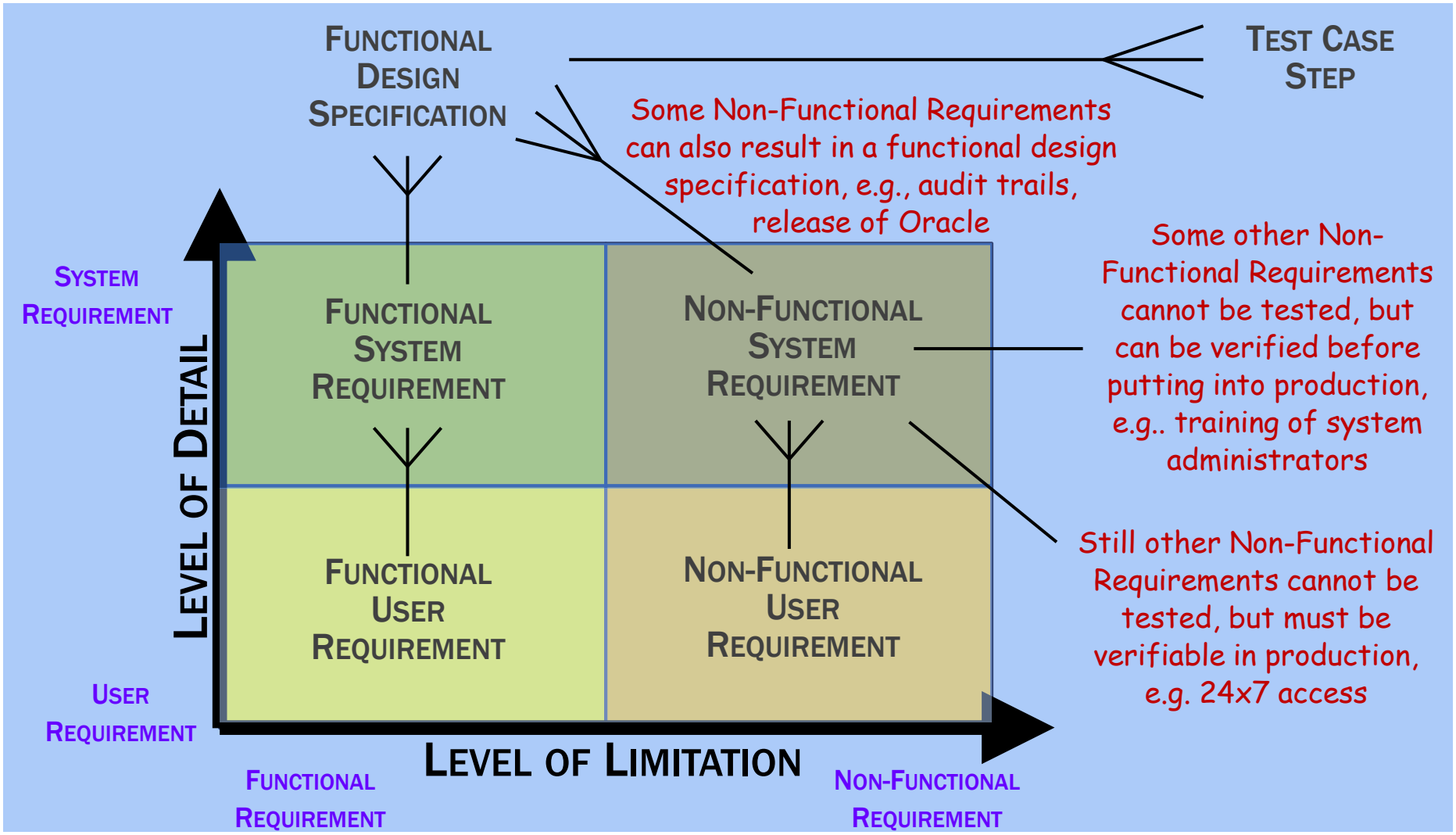
Traceability to requirements

TEST SCRIPT STEPS/RESULTS					
STEP	**TEST STEP/INPUT**	**EXPECTED RESULTS**	**ACTUAL RESULTS**	**Requirements Validated**	**PAS S/FA IL**
Get directions to a restaurant present in the database – Reqmts Validated: 3.7.1, 3.7.2, 3.7.5, 4.1-4.8					
1.	Enter restaurant name in the search window (e.g. McDonalds) that is present in the database	Able to enter text			
2.	Press the search button	Multiple results returned and pins are displayed on the map			
3.	Click on a single location	Map centers on that location and an information popup displays reviews and address, and link to get directions			
4.	Click on "get directions" link	Start address box appears			
5.	Type in a start address for your home.	Able to enter text			
6.	Press enter	Map shows a line from your home to restaurant. Directions in text are also displayed			
7.	Repeat steps, replacing step 6 with: Using mouse, press "Go" button	Same results as step 6		R3.7.56	
8.	Repeat steps 1-6 replacing start address with only city.	Directions are shown as in step 6, but only from the center of the city to the location		R3.7.55, R3.7.59	

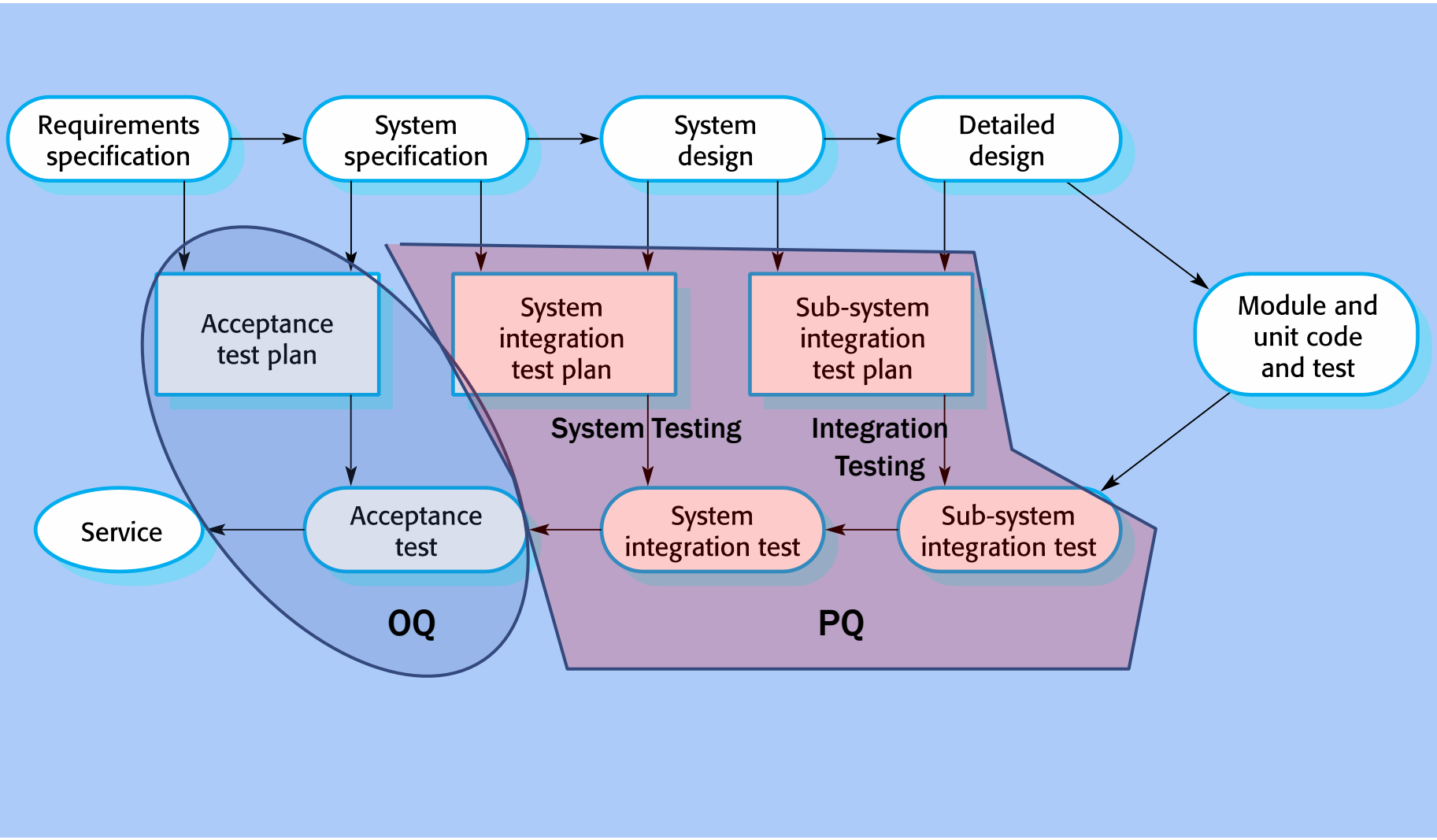
Traceability is often maintained in a separate document called a Traceability Matrix.

The "V" model of validation

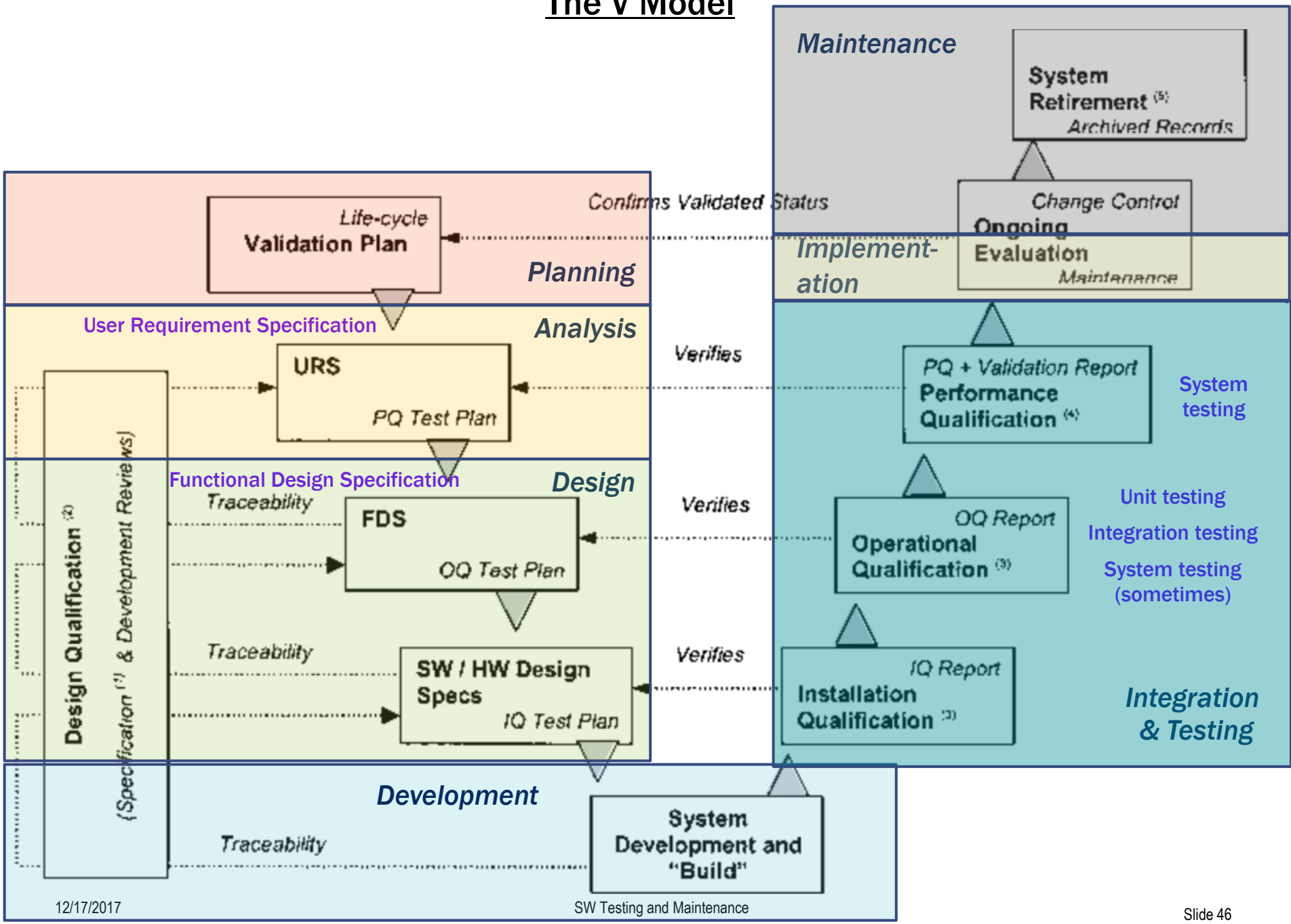
The Many-to-Many Complexity Problem



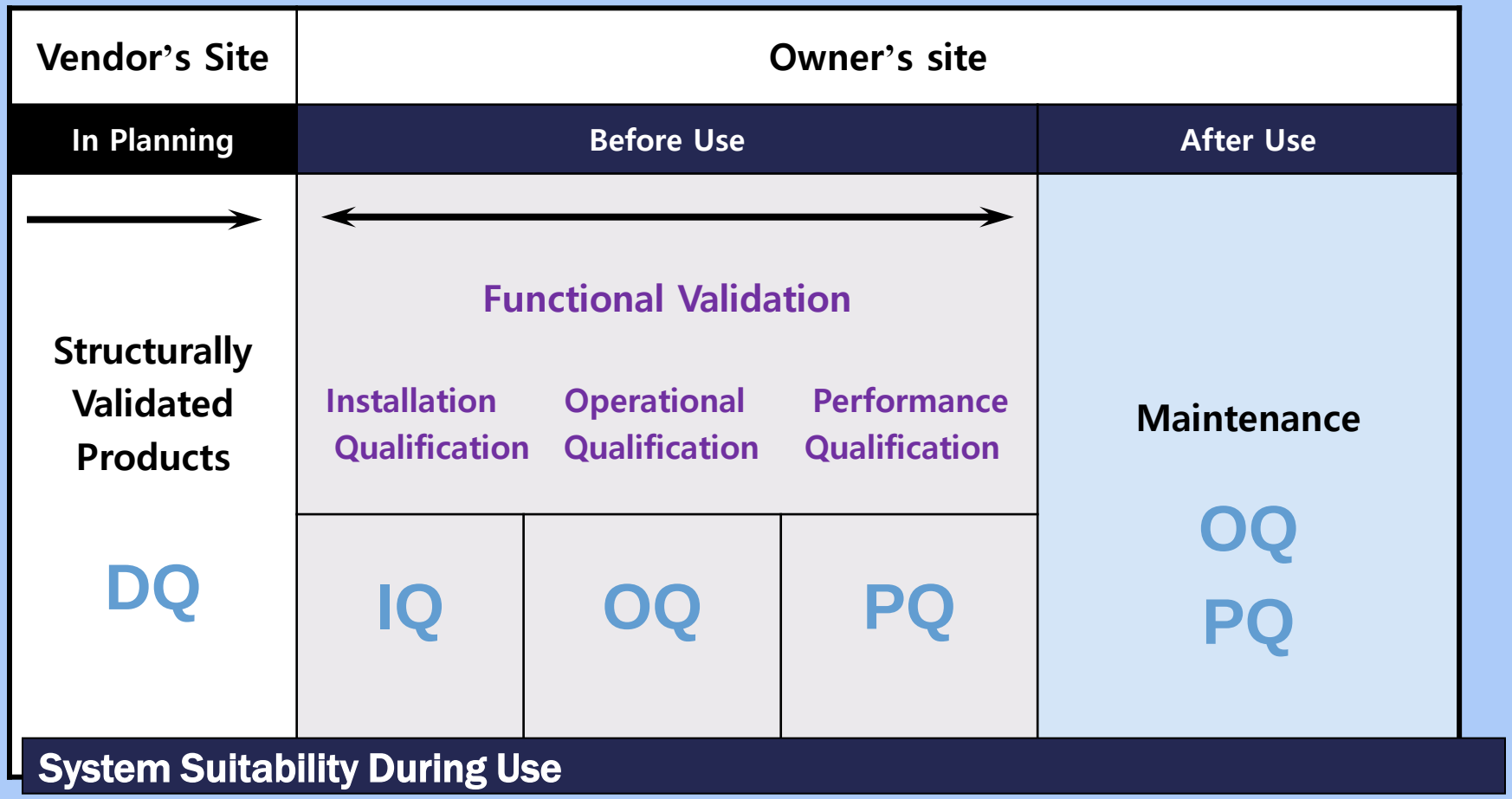
Testing phases in a plan-driven software process (V-model)



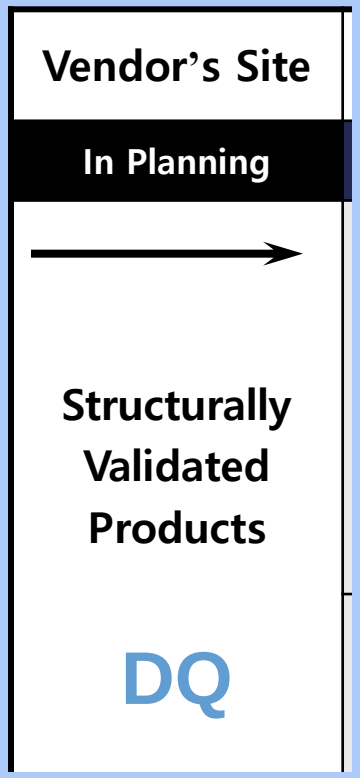
The V Model


Maintenance
System Retirement (5)
Archived Records
Change Control
Ongoing Evaluation
Maintenance
Implement-ation
Confirms Validated Status
Life-cycle Validation Plan
Planning
User Requirement Specification
Analysis
URS
PQ Test Plan
Verifies
PQ + Validation Report
Performance Qualification (4)
System testing
Functional Design Specification
Design
Traceability
FDS
OQ Test Plan
Verifies
OQ Report
Operational Qualification (3)
Unit testing
Integration testing
System testing (sometimes)
Design Qualification (2)
{Specification (1) & Development Reviews}
Traceability
SW / HW Design Specs
IQ Test Plan
Verifies
IQ Report
Installation Qualification (3)
Integration & Testing
Development
Traceability
System Development and "Build"

A formal software qualification process



Design Qualification:
Is product development done under best practices?

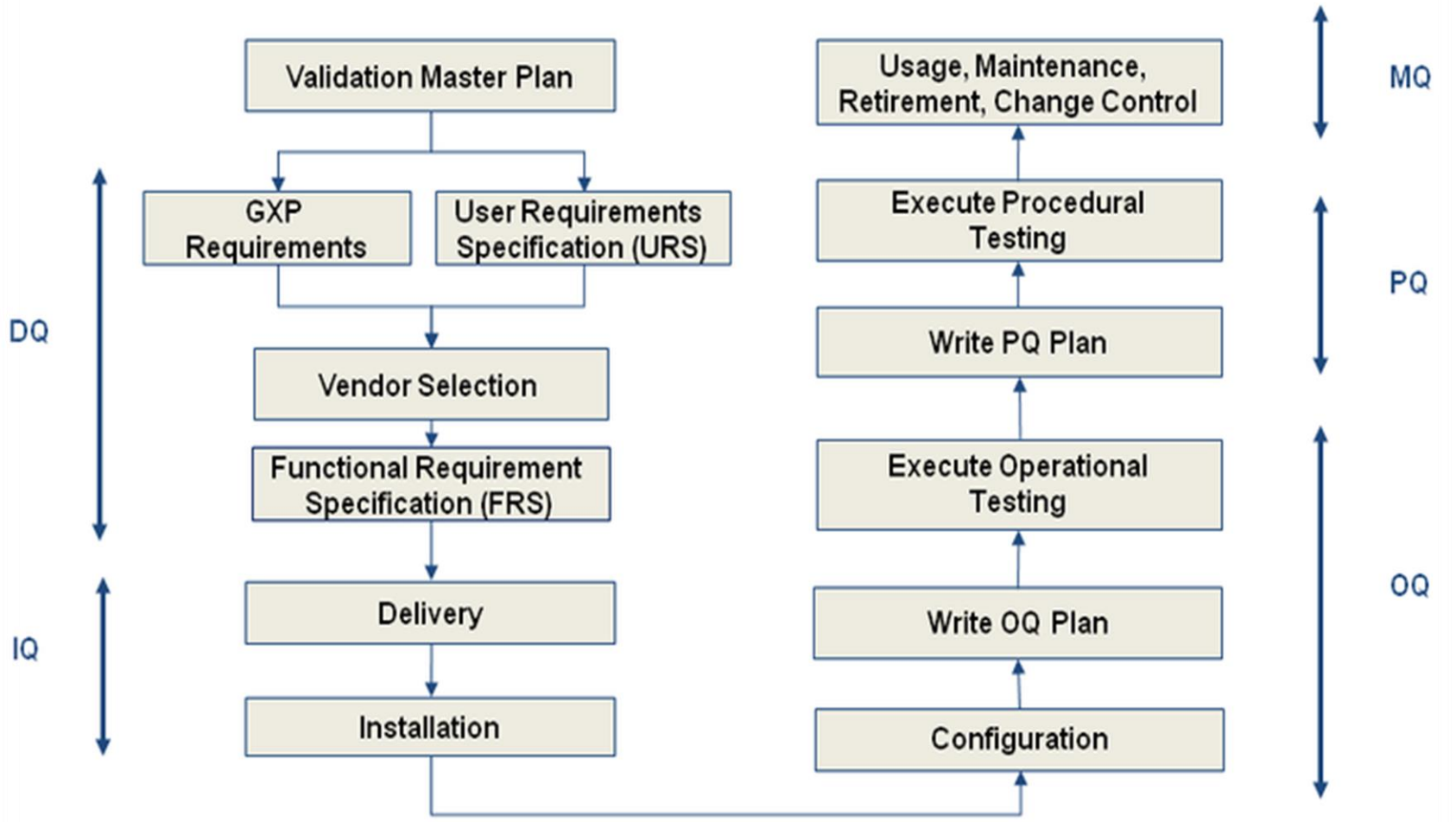


- Vendor:
  - Can be an external software provider (like SAP, Oracle)
    - Such vendors are often inspected and a Vendor Qualification Report is produced.
    - What development policies are in place?
    - Can all developers demonstrate that they are qualified? (resume / training records
    - What to do for Open Source software?
  - Can be an internal development team
    - All of the same checks are needed

Sample Traceability Matrix

Traceability Matrix

URS Reference Number	Description	Functional Specification Reference	Design Specialization Reference	Test or Verification Reference	Comments


Validation Master Plan
GXP Requirements
User Requirements Specification (URS)
Vendor Selection
Functional Requirement Specification (FRS)
Delivery
Installation
Usage, Maintenance, Retirement, Change Control
Execute Procedural Testing
Write PQ Plan
Execute Operational Testing
Write OQ Plan
Configuration
DQ
IQ
MQ
PQ
OQ

Release testing

Release testing

- Release testing is the process of testing a particular release of a system that is intended for use outside of the development team.
- The primary goal of the release testing process is to convince the supplier of the system that it is good enough for use.
  - Release testing, therefore, has to show that the system delivers its specified functionality, performance and dependability, and that it does not fail during normal use.
- Release testing is usually a black-box testing process where tests are only derived from the system specification.

Release testing and system testing

- Release testing is a form of system testing.
- Important differences:
  - A separate team that has not been involved in the system development, should be responsible for release testing.
  - System testing by the development team should focus on discovering bugs in the system (defect testing). The objective of release testing is to check that the system meets its requirements and is good enough for external use (validation testing).

Requirements based testing

- Requirements-based testing involves examining each requirement and developing a test or tests for it.
- Traceability matrices are good vehicles for ensuring this.

Performance testing

- Part of release testing may involve testing the emergent properties of a system, such as performance and reliability.
- Tests should reflect the profile of use of the system.
- Performance tests usually involve planning a series of tests where the load is steadily increased until the system performance becomes unacceptable.
- Stress testing is a form of performance testing where the system is deliberately overloaded to test its failure behaviour.

- Often required to test a non-functional performance requirement
- For discussion: should every system have a performance requirement?

User testing

User testing

- User or customer testing is a stage in the testing process in which users or customers provide input and advice on system testing.
- User testing is essential, even when comprehensive system and release testing have been carried out.
  - The reason for this is that influences from the user's working environment have a major effect on the reliability, performance, usability and robustness of a system. These cannot be replicated in a testing environment.

Types of user testing

- Alpha testing
  - Users of the software work with the development team to test the software at the developer's site.
- Beta testing
  - A release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers.
- Acceptance testing
  - Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment. Primarily for custom systems.

Agile methods and acceptance testing

- In agile methods, the user/customer is part of the development team and is responsible for making decisions on the acceptability of the system.
- Tests are defined by the user/customer and are integrated with other tests in that they are run automatically when changes are made.
- There is no separate acceptance testing process.
- Main problem here is whether or not the embedded user is ‘typical’ and can represent the interests of all system stakeholders.

Key points

- Testing can only show the presence of errors in a program. It cannot demonstrate that there are no remaining faults.
- Development testing is the responsibility of the software development team. A separate team should be responsible for testing a system before it is released to customers.
- Development testing includes unit testing, in which you test individual objects and methods component testing in which you test related groups of objects and system testing, in which you test partial or complete systems.

Key points

- When testing software, you should try to ‘break’ the software by using experience and guidelines to choose types of test case that have been effective in discovering defects in other systems.
- Wherever possible, you should write automated tests. The tests are embedded in a program that can be run every time a change is made to a system.
- Test-first development is an approach to development where tests are written before the code to be tested.
- Scenario testing involves inventing a typical usage scenario and using this to derive test cases.
- Acceptance testing is a user testing process where the aim is to decide if the software is good enough to be deployed and used in its operational environment.

Chapter 9
Software Evolution

Change Management

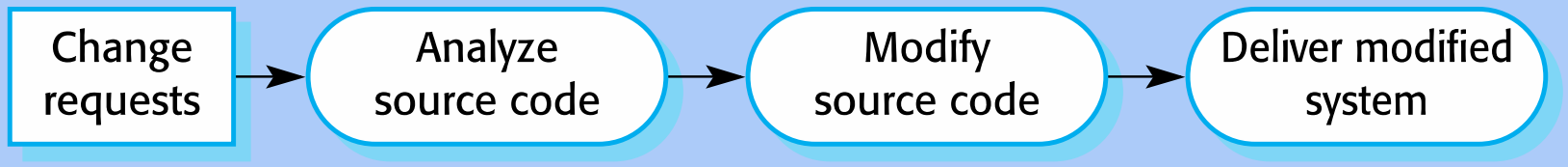
Change implementation

- Iteration of the development process where the revisions to the system are designed, implemented and tested.
- A critical difference is that the first stage of change implementation may involve program understanding, especially if the original system developers are not responsible for the change implementation.
- During the program understanding phase, you have to understand how the program is structured, how it delivers functionality and how the proposed change might affect the program.

Urgent change requests

- Urgent changes may have to be implemented without going through all stages of the software engineering process
  - If a serious system fault has to be repaired to allow normal operation to continue;
  - If changes to the system's environment (e.g. an OS upgrade) have unexpected effects;
  - If there are business changes that require a very rapid response (e.g. the release of a competing product).

The emergency repair process



Agile methods and evolution

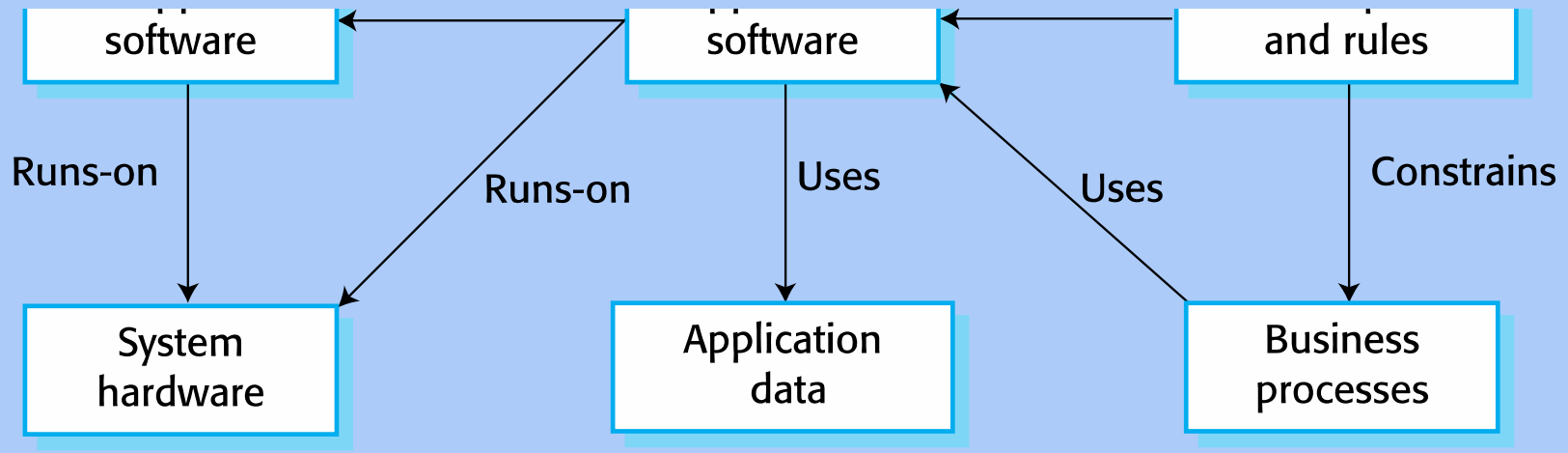
- Agile methods are based on incremental development so the transition from development to evolution is a seamless one.
  - Evolution is simply a continuation of the development process based on frequent system releases.
- Automated regression testing is particularly valuable when changes are made to a system.
- Changes may be expressed as additional user stories.

Legacy systems

Legacy systems

- Legacy systems are older systems that rely on languages and technology that are no longer used for new systems development.
- Legacy software may be dependent on older hardware, such as mainframe computers and may have associated legacy processes and procedures.
- Legacy systems are not just software systems but are broader socio-technical systems that include hardware, software, libraries and other supporting software and business processes.

The elements of a legacy system



- *System hardware:* Legacy systems may have been written for hardware that is no longer available.
- *Support software:* The legacy system may rely on a range of support software, which may be obsolete or unsupported.
- *Application software:* The application system that provides the business services is usually made up of a number of application programs.
- *Application data:* These are data that are processed by the application system. They may be inconsistent, duplicated or held in different databases.

Legacy system components

- *Business processes* These are processes that are used in the business to achieve some business objective.
- Business processes may be designed around a legacy system and constrained by the functionality that it provides.
- *Business policies and rules* These are definitions of how the business should be carried out and constraints on the business. Use of the legacy application system may be embedded in these policies and rules.

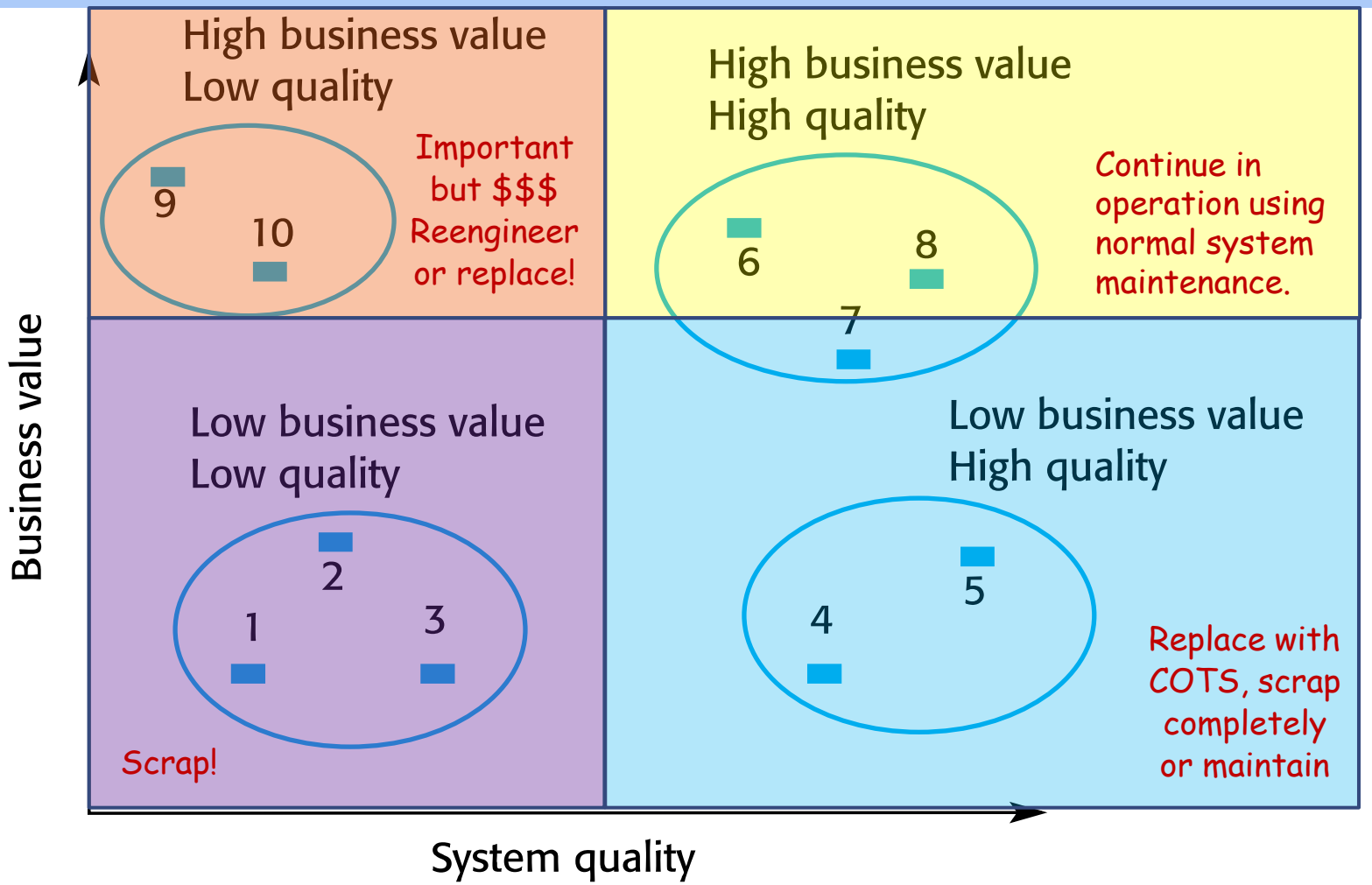
Legacy system replacement and change

- Legacy system replacement is risky and expensive so businesses continue to use these systems
- System replacement is risky for a number of reasons
  - Lack of complete system specification
  - Tight integration of system and business processes
  - Undocumented business rules embedded in the legacy system
  - New software development may be late and/or over budget
- Legacy systems are expensive to change for a number of reasons:
  - No consistent programming style
  - Use of obsolete programming languages with few people available with these language skills
  - Inadequate system documentation
  - System structure degradation
  - Program optimizations may make them hard to understand
  - Data errors, duplication and inconsistency

Legacy system management

- Organisations that rely on legacy systems must choose a strategy for evolving these systems
  - Scrap the system completely and modify business processes so that it is no longer required;
  - Continue maintaining the system;
  - Transform the system by re-engineering to improve its maintainability;
  - Replace the system with a new system.
- The strategy chosen should depend on the system quality and its business value.

Figure 9.13 An example of a legacy system assessment



Business value assessment

- Assessment should take different viewpoints into account
  - System end-users;
  - Business customers;
  - Line managers;
  - IT managers;
  - Senior managers.
- Interview different stakeholders and collate results.

Issues in business value assessment

- The use of the system
  - If systems are only used occasionally or by a small number of people, they may have a low business value.
- The business processes that are supported
  - A system may have a low business value if it forces the use of inefficient business processes.
- System dependability
  - If a system is not dependable and the problems directly affect business customers, the system has a low business value.
- The system outputs
  - If the business depends on system outputs, then the system has a high business value.

System quality assessment

- Business process assessment
  - How well does the business process support the current goals of the business?
- Environment assessment
  - How effective is the system's environment and how expensive is it to maintain?
- Application assessment
  - What is the quality of the application software system?

Business process assessment

- Use a viewpoint-oriented approach and seek answers from system stakeholders
  - Is there a defined process model and is it followed?
  - Do different parts of the organisation use different processes for the same function?
  - How has the process been adapted?
  - What are the relationships with other business processes and are these necessary?
  - Is the process effectively supported by the legacy application software?
- Example - a travel ordering system may have a low business value because of the widespread use of web-based ordering.

Factors used in environment assessment

Factor	Questions
Supplier stability	Is the supplier still in existence? Is the supplier financially stable and likely to continue in existence? If the supplier is no longer in business, does someone else maintain the systems?
Failure rate	Does the hardware have a high rate of reported failures? Does the support software crash and force system restarts?
Age	How old is the hardware and software? The older the hardware and support software, the more obsolete it will be. It may still function correctly but there could be significant economic and business benefits to moving to a more modern system.
Performance	Is the performance of the system adequate? Do performance problems have a significant effect on system users?
Support requirements	What local support is required by the hardware and software? If there are high costs associated with this support, it may be worth considering system replacement.
Maintenance costs	What are the costs of hardware maintenance and support software licences? Older hardware may have higher maintenance costs than modern systems. Support software may have high annual licensing costs.
Interoperability	Are there problems interfacing the system to other systems? Can compilers, for example, be used with current versions of the operating system? Is hardware emulation required?

Factors used in application assessment

Factor	Questions
Understandability	How difficult is it to understand the source code of the current system? How complex are the control structures that are used? Do variables have meaningful names that reflect their function?
Documentation	What system documentation is available? Is the documentation complete, consistent, and current?
Data	Is there an explicit data model for the system? To what extent is data duplicated across files? Is the data used by the system up to date and consistent?
Performance	Is the performance of the application adequate? Do performance problems have a significant effect on system users?
Programming language	Are modern compilers available for the programming language used to develop the system? Is the programming language still used for new system development?
Configuration management	Are all versions of all parts of the system managed by a configuration management system? Is there an explicit description of the versions of components that are used in the current system?
Test data	Does test data for the system exist? Is there a record of regression tests carried out when new features have been added to the system?
Personnel skills	Are there people available who have the skills to maintain the application? Are there people available who have experience with the system?

System measurement

- You may collect quantitative data to make an assessment of the quality of the application system
  - The number of system change requests; The higher this accumulated value, the lower the quality of the system.
  - The number of different user interfaces used by the system; The more interfaces, the more likely it is that there will be inconsistencies and redundancies in these interfaces.
  - The volume of data used by the system. As the volume of data (number of files, size of database, etc.) processed by the system increases, so too do the inconsistencies and errors in that data.
  - Cleaning up old data is a very expensive and time-consuming process

Software maintenance

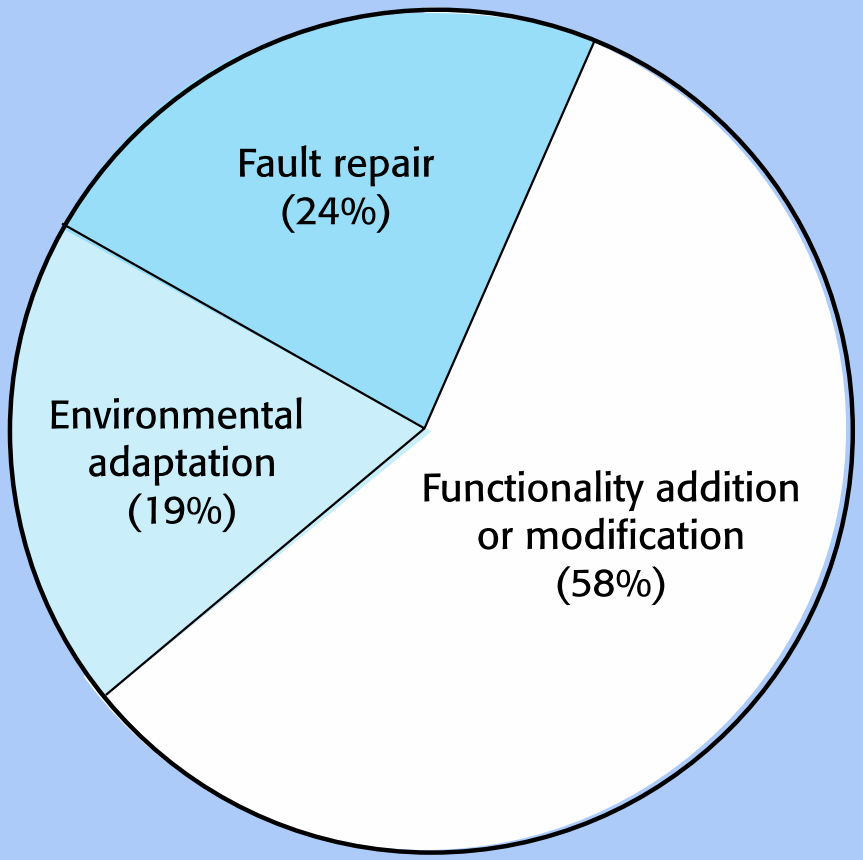
Software maintenance

- Modifying a program after it has been put into use.
- The term is mostly used for changing custom software. Generic software products are said to evolve to create new versions.
- Maintenance does not normally involve major changes to the system's architecture.
- Changes are implemented by modifying existing components and adding new components to the system.

Types of maintenance

- Fault repairs
  - Changing a system to fix bugs/vulnerabilities and correct deficiencies in the way meets its requirements.
- Environmental adaptation
  - Maintenance to adapt software to a different operating environment
  - Changing a system so that it operates in a different environment (computer, OS, etc.) from its initial implementation.
- Functionality addition and modification
  - Modifying the system to satisfy new requirements.

Maintenance effort distribution



Maintenance costs

- Usually greater than development costs (2* to 100* depending on the application).
- Affected by both technical and non-technical factors.
- Increases as software is maintained. Maintenance corrupts the software structure so makes further maintenance more difficult.
- Ageing software can have high support costs (e.g. old languages, compilers etc.).

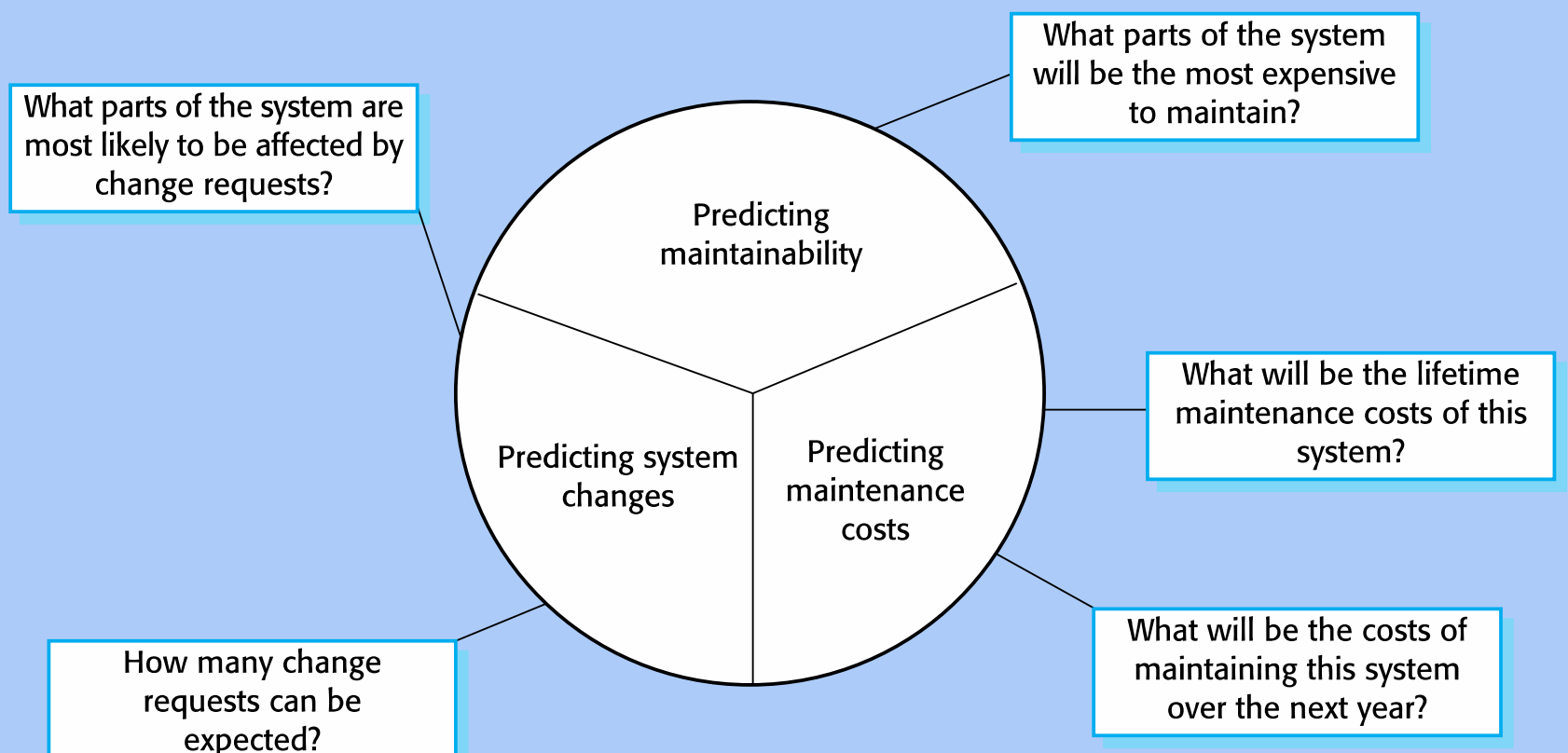
Maintenance costs

- It is usually more expensive to add new features to a system during maintenance than it is to add the same features during development
  - A new team has to understand the programs being maintained
  - Separating maintenance and development means there is no incentive for the development team to write maintainable software
  - Program maintenance work is unpopular
    - Maintenance staff are often inexperienced and have limited domain knowledge.
  - As programs age, their structure degrades and they become harder to change

Maintenance prediction

- Maintenance prediction is concerned with assessing which parts of the system may cause problems and have high maintenance costs
  - Change acceptance depends on the maintainability of the components affected by the change;
  - Implementing changes degrades the system and reduces its maintainability;
  - Maintenance costs depend on the number of changes and costs of change depend on maintainability.

Maintenance prediction



Change prediction

- Predicting the number of changes requires and understanding of the relationships between a system and its environment.
- Tightly coupled systems require changes whenever the environment is changed.
- Factors influencing this relationship are
  - Number and complexity of system interfaces;
  - Number of inherently volatile system requirements;
  - The business processes where the system is used.

Complexity metrics

- Predictions of maintainability can be made by assessing the complexity of system components.
- Studies have shown that most maintenance effort is spent on a relatively small number of system components.
- Complexity depends on
  - Complexity of control structures;
  - Complexity of data structures;
  - Object, method (procedure) and module size.

Process metrics

- Process metrics may be used to assess maintainability
  - Number of requests for corrective maintenance;
  - Average time required for impact analysis;
  - Average time taken to implement a change request;
  - Number of outstanding change requests.
- If any or all of these is increasing, this may indicate a decline in maintainability.

Key points

- It is often cheaper and less risky to maintain a legacy system than to develop a replacement system using modern technology.
- The business value of a legacy system and the quality of the application should be assessed to help decide if a system should be replaced, transformed or maintained.
- There are 3 types of software maintenance, namely bug fixing, modifying software to work in a new environment, and implementing new or changed requirements.

Bonus Material: Software Engineering

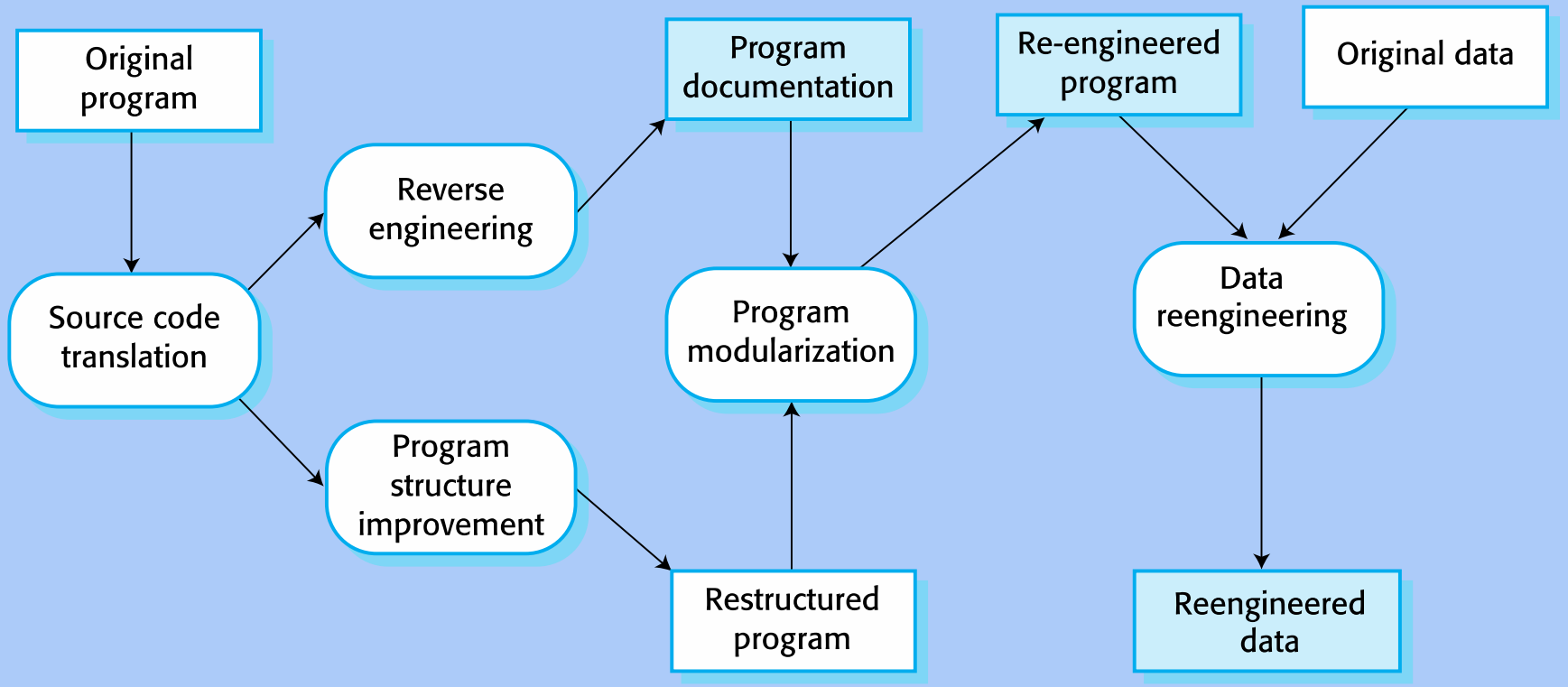
Software reengineering

- Restructuring or rewriting part or all of a legacy system without changing its functionality.
- Applicable where some but not all sub-systems of a larger system require frequent maintenance.
- Reengineering involves adding effort to make them easier to maintain. The system may be re-structured and re-documented.

Advantages of reengineering

- Reduced risk
  - There is a high risk in new software development. There may be development problems, staffing problems and specification problems.
- Reduced cost
  - The cost of re-engineering is often significantly less than the costs of developing new software.

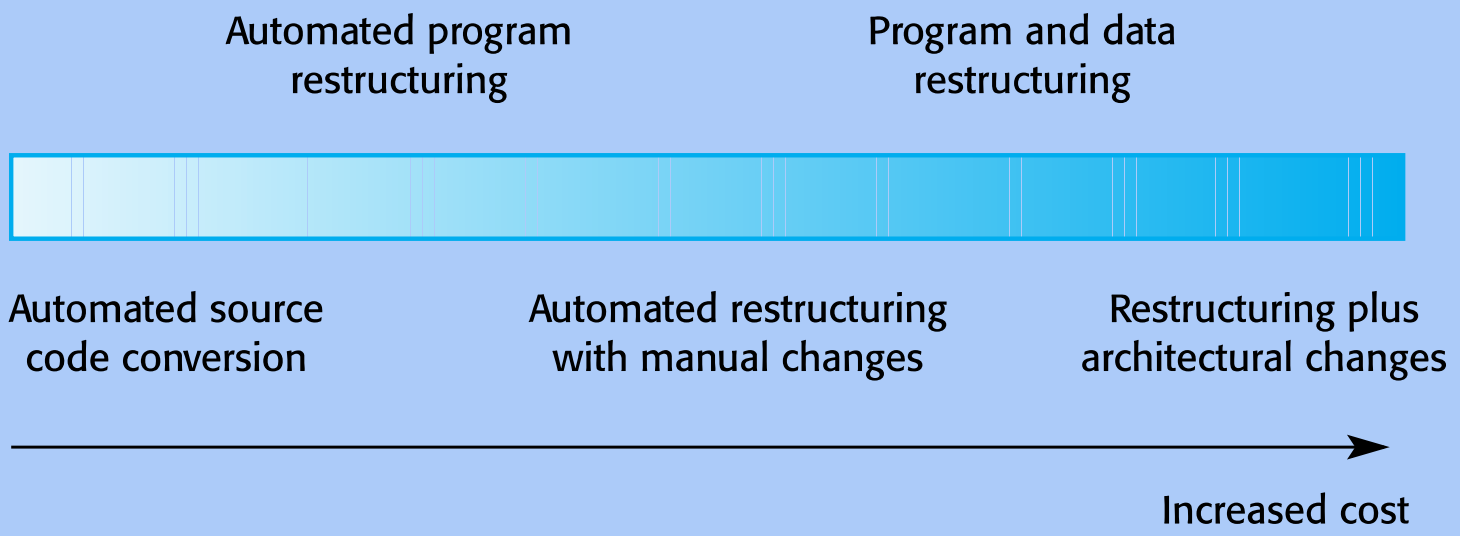
The reengineering process



Reengineering process activities

- Source code translation
  - Convert code to a new language.
- Reverse engineering
  - Analyse the program to understand it;
- Program structure improvement
  - Restructure automatically for understandability;
- Program modularisation
  - Reorganise the program structure;
- Data reengineering
  - Clean-up and restructure system data.

Reengineering approaches



Reengineering cost factors

- The quality of the software to be reengineered.
- The tool support available for reengineering.
- The extent of the data conversion which is required.
- The availability of expert staff for reengineering.
  - This can be a problem with old systems based on technology that is no longer widely used.

Refactoring

- Refactoring is the process of making improvements to a program to slow down degradation through change.
- You can think of refactoring as ‘preventative maintenance’ that reduces the problems of future change.
- Refactoring involves modifying a program to improve its structure, reduce its complexity or make it easier to understand.
- When you refactor a program, you should not add functionality but rather concentrate on program improvement.

Refactoring and reengineering

- Re-engineering takes place after a system has been maintained for some time and maintenance costs are increasing. You use automated tools to process and re-engineer a legacy system to create a new system that is more maintainable.
- Refactoring is a continuous process of improvement throughout the development and evolution process. It is intended to avoid the structure and code degradation that increases the costs and difficulties of maintaining a system.

Key points

- Software development and evolution can be thought of as an integrated, iterative process that can be represented using a spiral model.
- For custom systems, the costs of software maintenance usually exceed the software development costs.
- The process of software evolution is driven by requests for changes and includes change impact analysis, release planning and change implementation.
- Legacy systems are older software systems, developed using obsolete software and hardware technologies, that remain useful for a business.

Key points

- Software re-engineering is concerned with re-structuring and re-documenting software to make it easier to understand and change.
- Refactoring, making program changes that preserve functionality, is a form of preventative maintenance.

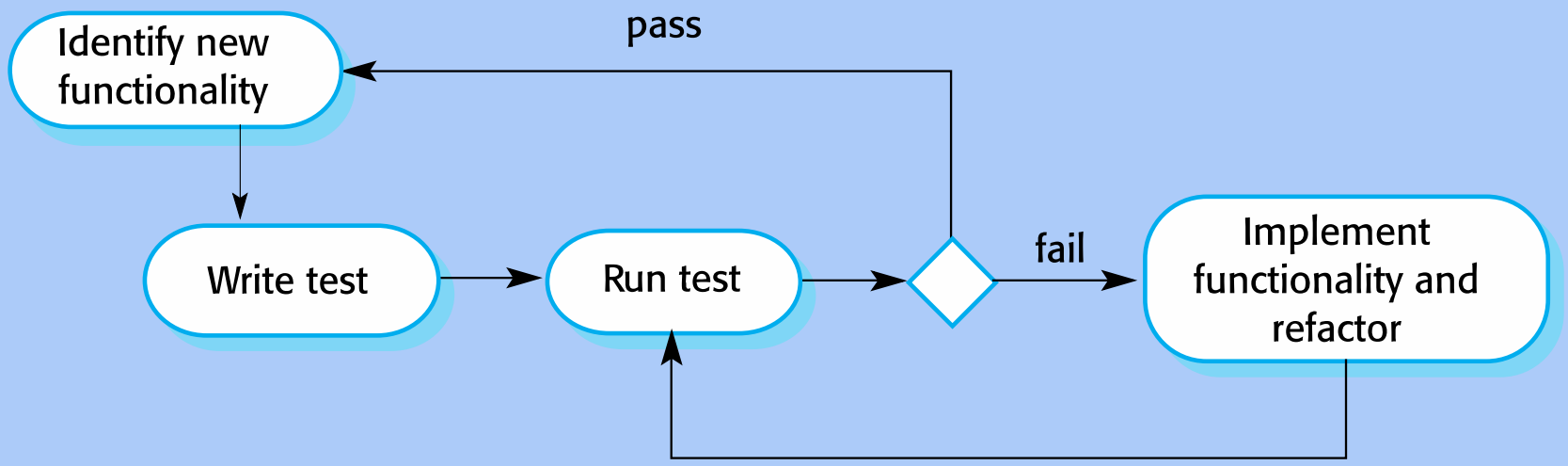
Supplemental Slides

Test-driven development

Test-driven development

- Test-driven development (TDD) is an approach to program development in which you inter-leave testing and code development.
- Tests are written before code and 'passing' the tests is the critical driver of development.
- You develop code incrementally, along with a test for that increment. You don't move on to the next increment until the code that you have developed passes its test.
- TDD was introduced as part of agile methods such as Extreme Programming. However, it can also be used in plan-driven development processes.

Test-driven development



TDD process activities

- Start by identifying the increment of functionality that is required. This should normally be small and implementable in a few lines of code.
- Write a test for this functionality and implement this as an automated test.
- Run the test, along with all other tests that have been implemented. Initially, you have not implemented the functionality so the new test will fail.
- Implement the functionality and re-run the test.
- Once all tests run successfully, you move on to implementing the next chunk of functionality.

Benefits of test-driven development

- Code coverage
  - Every code segment that you write has at least one associated test so all code written has at least one test.
- Regression testing
  - A regression test suite is developed incrementally as a program is developed.
- Simplified debugging
  - When a test fails, it should be obvious where the problem lies. The newly written code needs to be checked and modified.
- System documentation
  - The tests themselves are a form of documentation that describe what the code should be doing.

Regression testing

- Regression testing is testing the system to check that changes have not ‘broken’ previously working code.
- In a manual testing process, regression testing is expensive but, with automated testing, it is simple and straightforward. All tests are rerun every time a change is made to the program.
- Tests must run ‘successfully’ before the change is committed.

A usage scenario for the Mentcare system
What should we test?

George is a nurse who specializes in mental healthcare. One of his responsibilities is to visit patients at home to check that their treatment is effective and that they are not suffering from medication side effects.

On a day for home visits, George logs into the Mentcare system and uses it to print his schedule of home visits for that day, along with summary information about the patients to be visited. He requests that the records for these patients be downloaded to his laptop. He is prompted for his key phrase to encrypt the records on the laptop.

One of the patients that he visits is Jim, who is being treated with medication for depression. Jim feels that the medication is helping him but believes that it has the side effect of keeping him awake at night. George looks up Jim's record and is prompted for his key phrase to decrypt the record. He checks the drug prescribed and queries its side effects. Sleeplessness is a known side effect so he notes the problem in Jim's record and suggests that he visits the clinic to have his medication changed. Jim agrees so George enters a prompt to call him when he gets back to the clinic to make an appointment with a physician. George ends the consultation and the system re-encrypts Jim's record.

After, finishing his consultations, George returns to the clinic and uploads the records of patients visited to the database. The system generates a call list for George of those patients who He has to contact for follow-up information and make clinic appointments.

Features tested by scenario

- Authentication by logging on to the system.
- Downloading and uploading of specified patient records to a laptop.
- Home visit scheduling.
- Encryption and decryption of patient records on a mobile device.
- Record retrieval and modification.
- Links with the drugs database that maintains side-effect information.
- The system for call prompting.

The acceptance testing process