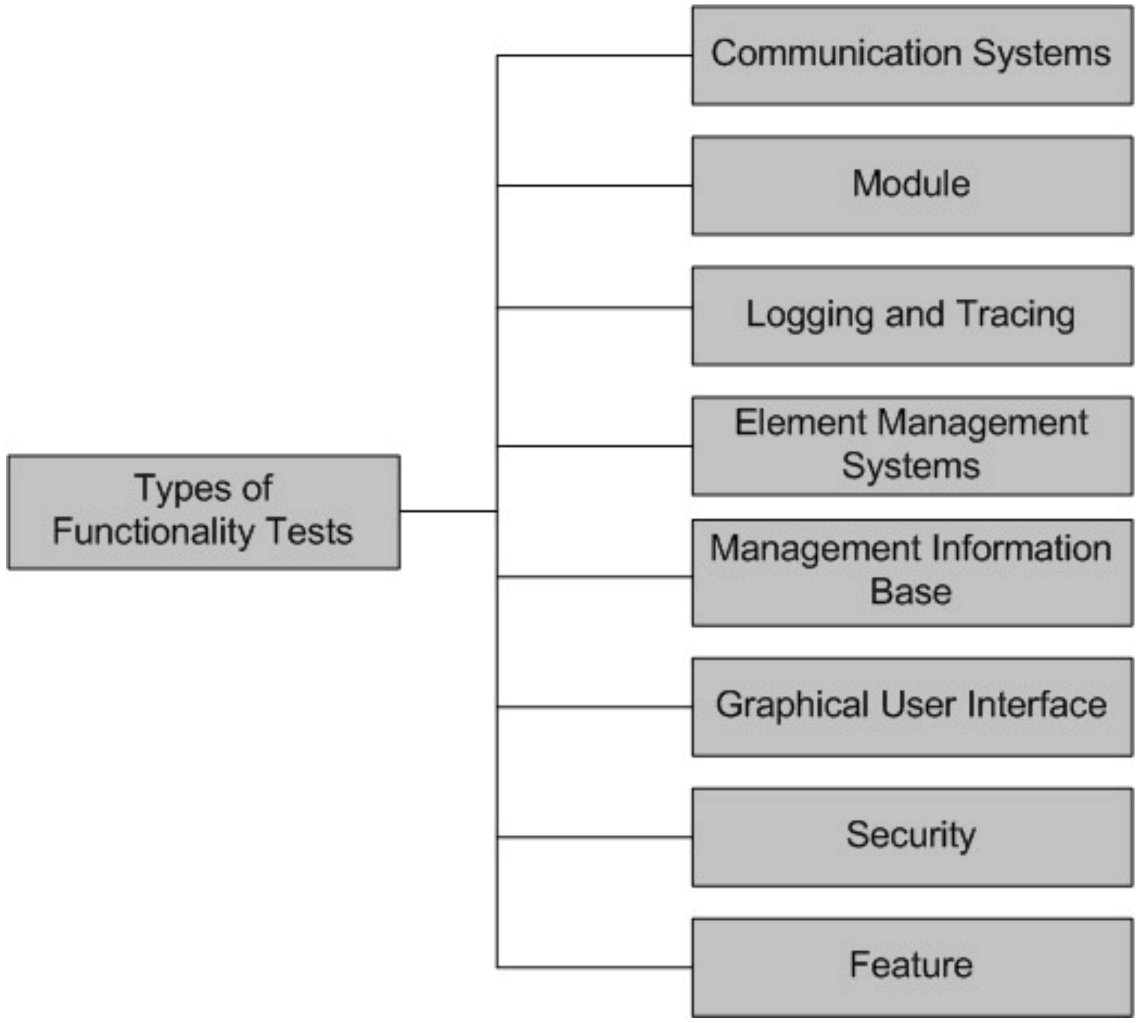
Lecture 20: Testing (4) & Maintenance

Functionality Tests



Robustness Tests

- Robustness means how much sensitive a system is to erroneous input and changes its operational environment

- Tests in this category are designed to verify how gracefully the system behaves in error situations and in a changed operational environment

Robustness Tests (2)

- Boundary value
  - Boundary value tests are designed to cover boundary conditions, special values, and system defaults
  - The tests include providing invalid input data to the system and observing how the system reacts to the invalid input.

Robustness Tests (3)

- Power cycling
  - Power cycling tests are executed to ensure that, when there is a power glitch in a deployment environment, the system can recover from the glitch to be back in normal operation after power is restored

Robustness Tests (4)

- High Availability
  - The concept of high availability is also known as **fault tolerance**
  - High availability tests are designed to verify the redundancy of individual modules, including the software that controls these modules.
  - The goal is to verify that the system gracefully and quickly recovers from hardware and software failures without adversely impacting the operation of the system
  - High availability is realized by means of proactive methods to maximize service up-time, and to minimize the downtime

Performance Tests

- Tests are designed to determine the performance of the actual system compared to the expected one;
- Tests are designed to verify response time, execution time, throughput, resource utilization and traffic rate;
- One needs to be clear about the specific data to be captured in order to evaluate performance metrics.

Performance Tests (2)

- For example, if the objective is to evaluate the response time, then one needs to capture
  - End-to-end response time (as seen by external user)
  - CPU time
  - Network connection time
  - Database access time
  - Network connection time
  - Waiting time

Scalability Tests

- Tests are designed to verify that the system can scale up to its engineering limits
- Scaling tests are conducted to ensure that the system response time remains the same, or increases by a small amount, as the number of users are increased.
- There are three major causes of these limitations:
  - data storage limitations
  - network bandwidth limitations
  - speed limit
- Extrapolation is often used to predict the limit of scalability

Stress Tests

- The goal of stress testing is to evaluate and determine the behavior of a software component while the offered load is in excess of its designed capacity;
- The system is deliberately stressed by pushing it to and beyond its specified limits;
- It ensures that the system can perform acceptably under worst-case conditions, under an expected peak load. If the limit is exceeded and the system does fail, then the recovery mechanism should be invoked;

Regression Tests

- In this category, new tests are not designed, instead, test cases are selected from the existing pool and executed

- The main idea in regression testing is to verify that no defect has been introduced into the unchanged portion of a system due to changes made elsewhere in the system

- During system testing, many defects are revealed and the code is modified to fix those defects

Documentation Tests

- Documentation testing means verifying the technical accuracy and readability of the user manuals, tutorials and the on-line help

- Documentation testing is performed at three levels:
  - ***Read test:*** In this test a documentation is reviewed for clarity, organization, flow, and accuracy without executing the documented instructions on the system
  - ***Hands-on test:*** Exercise the on-line help and verify the error messages to evaluate their accuracy and usefulness.
  - ***Functional test:*** Follow the instructions embodied in the documentation to verify that the system works as it has been documented.

Software Safety

- A *hazard* is a state of a system or a physical situation which when combined with certain environmental conditions, could lead to an *accident* or *mishap*
- An *accident* or *mishap* is an unintended event or series of events that results in death, injury, illness, damage or loss of property, or harm to the environment
- Software *safety* is defined in terms of hazards
- A software in isolation cannot do physical damage. However, a software in the context of a system and an embedding environment could be vulnerable

Software Safety (2)

Examples:

- A software module in a database application is not hazardous by itself, but when it is embedded in a missile navigation system, it could be hazardous
- If a missile takes a U-turn because of a software error in the navigation system, and destroys the submarine that launched it, then it is not a safe software

Software Safety Assurance

- There are two basic tasks performed by a **safety assurance** engineering team:
  - Provide methods for identifying, tracking, evaluating, and eliminating hazards associated with a system
  - Ensure that safety is embedded into the design and implementation in a timely and cost effective manner, such that the risk created by the user/operator error is minimized

Installation Testing

- Installation testing is to test the validity of written installation procedures for installing the system onto the target environment i.e. to verify if the installation procedures are accurate and with understandable instructions.

- Installation testing
  – Ensure the system is installed and set up properly.

User Testing

- User/customer testing 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 testing has been carried out;
  - Influences from the user's working environment can have a major effect on the reliability, performance, usability, and robustness of a system.

User Testing (2)

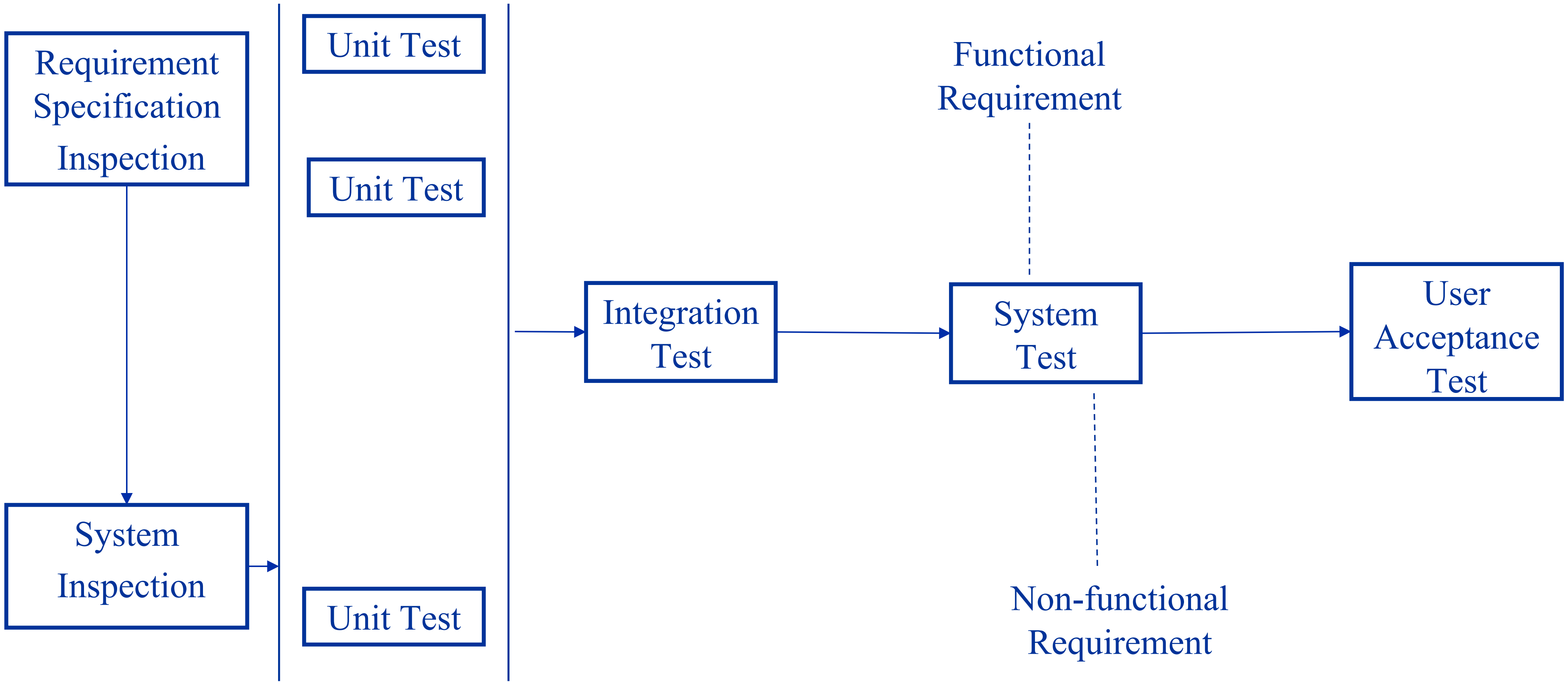
- Example:
  - For example, a system that is intended for use in a hospital is used in a clinical environment where other things are going on, such as patient emergencies and conversations with relatives. These all affect the use of a system, but developers cannot include them in their testing environment.

User Testing (3)

- There are three types of user testing:
  - Alpha testing, where a selected group of software users work closely with the development team to test early releases of the software;
  - Beta testing, where a release of the software is made available to larger group of users to allow them to experiment and to raise problems that they discover with the system developers.
  - Acceptance testing, where 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.

Recap

Test Phase


Requirement Specification Inspection
System Inspection
Unit Test
Unit Test
Unit Test
Integration Test
System Test
User Acceptance Test
Functional Requirement
Non-functional Requirement

Types of Tests

- Test object: Unit test, integration test, system test, user acceptance test.
- Test techniques: Black-box (specification-based), white-box (structure-based)
- Execution: Execution-based testing, Nonexecution-based testing;

Component level test/Unit test/Class test

- *Test object*:
  - components, program modules, functions, programs,
- *Objective*:
  - detect failures in the components,
- *Entry criteria*:
  - the component is available, compiled, and executable in the test environment;
  - the specifications are available and stable.
- *Exit criteria*:
  - the required coverage level has been reached;
  - defects found have been corrected;
  - the corrections have been verified;
  - regression tests have been executed on the last version and do not identify any regression;
  - traceability from requirements to test execution is maintained

Integration Level Testing

- *Test object*:
  - components, infrastructure, interfaces, database systems, and file systems.
- *Objective*:
  - detect failures in the interfaces and exchanges between components.
- *Entry criteria*:
  - at least two components that must exchange data are available, and have passed component test successfully.
- *Exit criteria*:
  - all components have been integrated and all message types (sent or received) have been exchanged without any defect for each existing interface;
  - statistics (such as *defect density*) are available;
  - the *defects* requiring correction have been corrected and checked as correctly fixed;
  - the impact of not-to-fix defects have been evaluated as not important.

Types of integration

- Integration of software components
  - typically executed during component integration tests
- Integration of software components with hardware components
- Integration of sub-systems
  - typically executed after the system tests for each of these sub-systems and before the systems tests of the higher-level system.

System Test

- *Test object*:
  - the complete software or system, its documentation, the software configuration and all the components that are linked to it
- *Objective*:
  - detect failures in the software, to ensure that it corresponds to the requirements and specifications, and that it can be accepted by the users.
- *Entry criteria*:
  - all components have been correctly integrated, all components are available.
- *Exit criteria*:
  - the level of coverage has been reached;
  - must-fix defects have been corrected and their fixes have been verified;
  - regression tests have been executed on the last version of the software and do not show any regression;
  - bi-directional traceability from requirements to test execution is maintained;
  - statistical data are available, the number of non-important defects is not large;
  - the summary test report has been written and approved.

Evolution

Software Change

- Software change is inevitable
  - New requirements emerge when the software is used;
  - The business environment changes;
  - Errors must be repaired;
  - New computers and equipment is added to the system;
  - The performance or reliability of the system may have to be improved.
  - …
- A key problem for all organizations is implementing and managing change to their existing software systems.

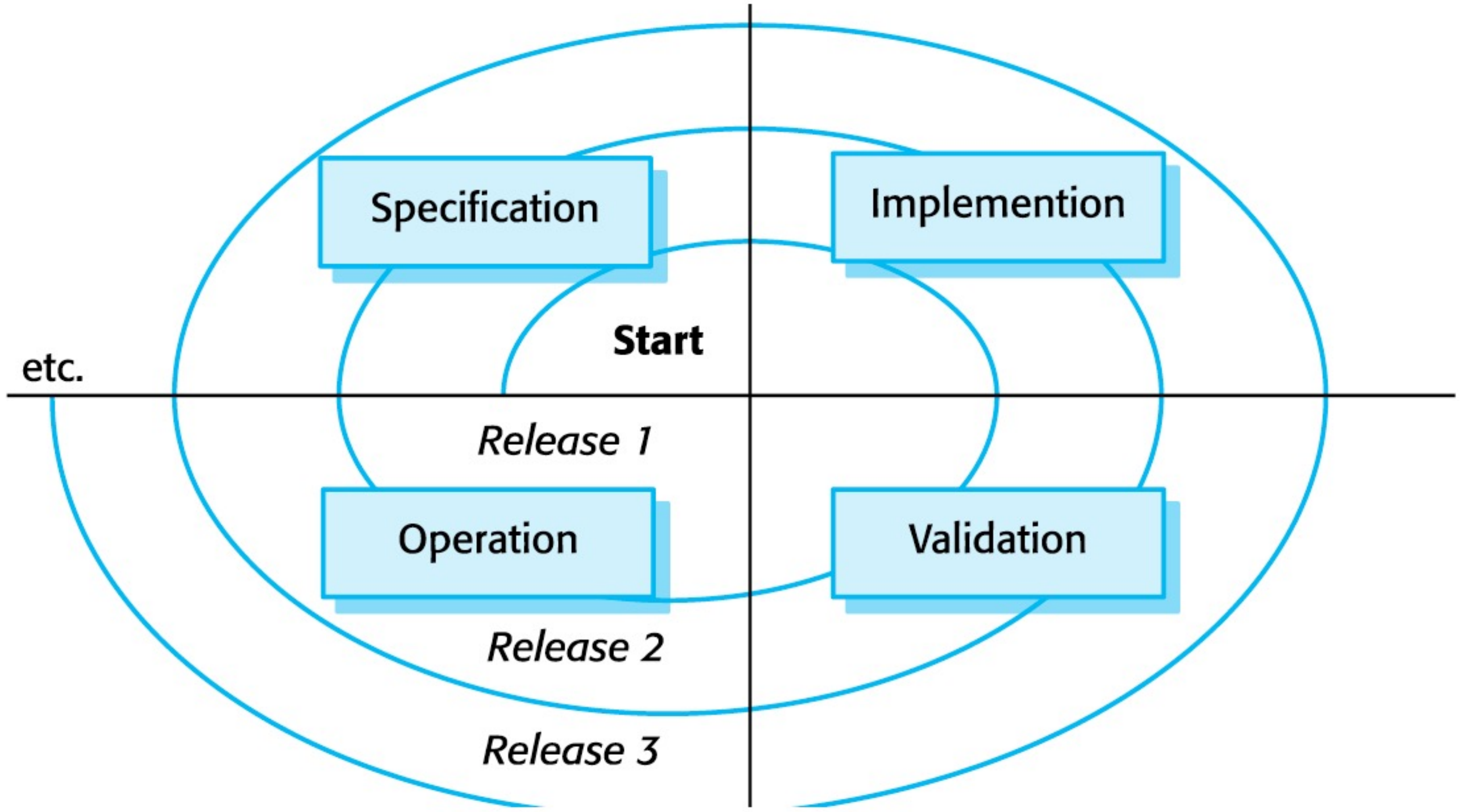
Importance of evolution

- Organizations have huge investments in their software systems - they are critical business assets;

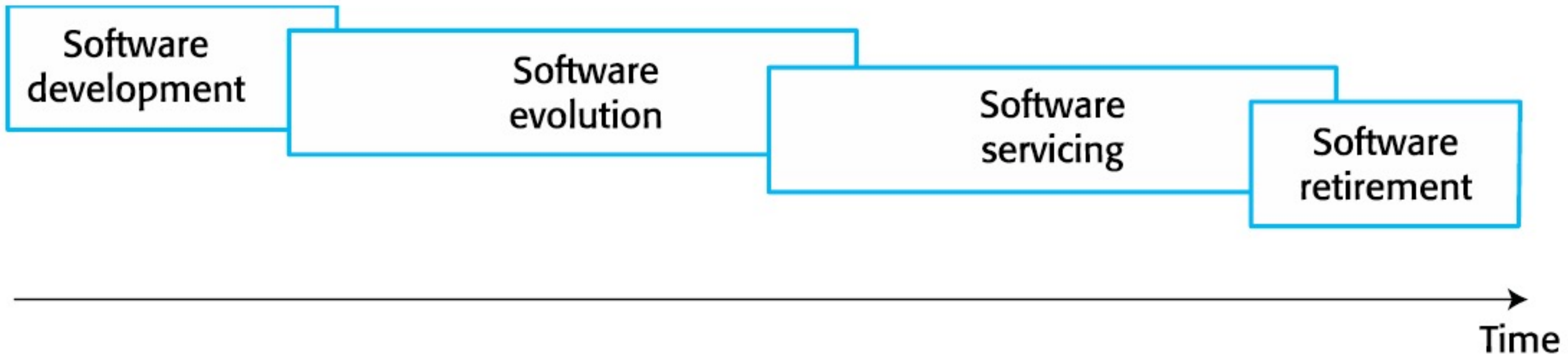
- To maintain the value of these assets to the business, they must be changed and updated;

- The majority of the software budget in large companies is devoted to changing and evolving existing software rather than developing new software.

A spiral model of development and evolution



Evolution and servicing



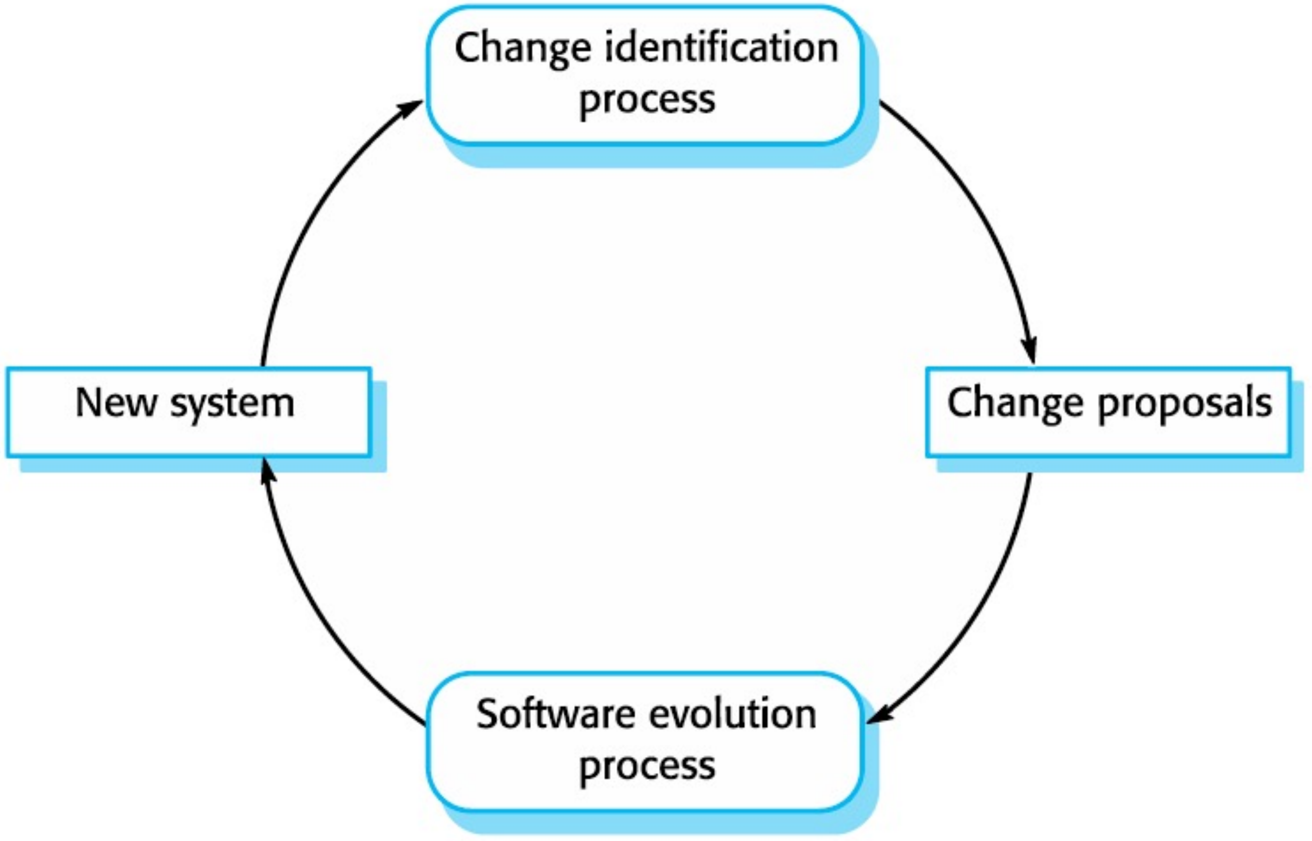
Evolution and servicing (2)

- Evolution
  - The stage in a software system's life cycle where it is in operational use and is evolving as new requirements are proposed and implemented in the system.
- Servicing
  - At this stage, the software remains useful but the only changes made are those required to keep it operational i.e. bug fixes and changes to reflect changes in the software's environment. No new functionality is added.
- Phase-out
  - The software may still be used but no further changes are made to it.

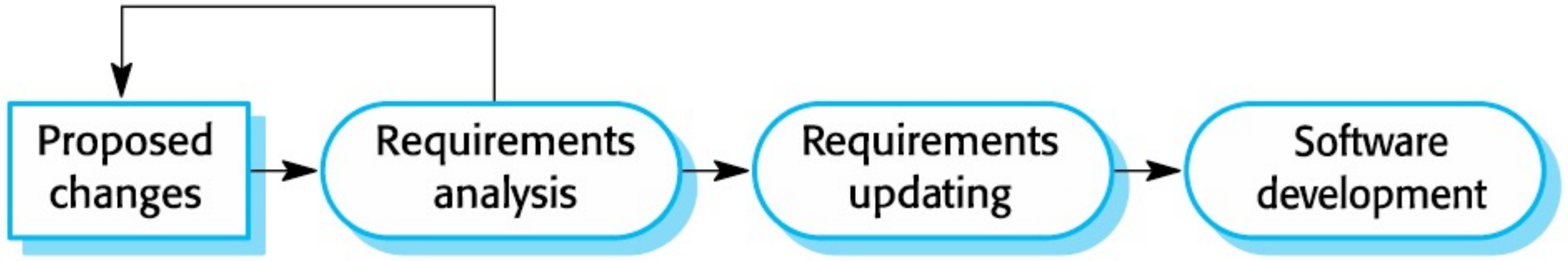
Evolution Process

- Software evolution processes depend on
  - The type of software being maintained;
  - The development processes used;
  - The skills and experience of the people involved.
- Proposals for change are the driver for system evolution.
  - Should be linked with components that are affected by the change, thus allowing the cost and impact of the change to be estimated.
- Change identification and evolution continues throughout the system lifetime.

Evolution Processes (2)



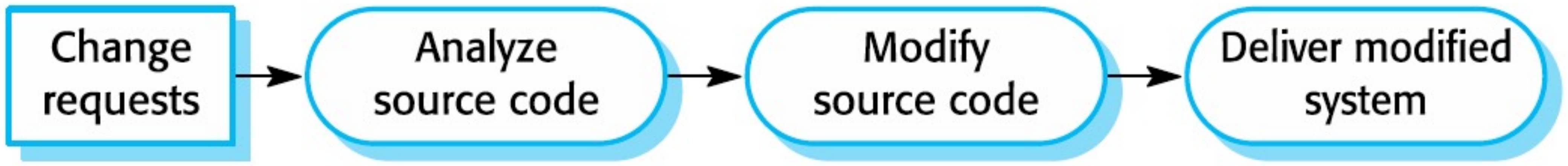
Change implementation



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.

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

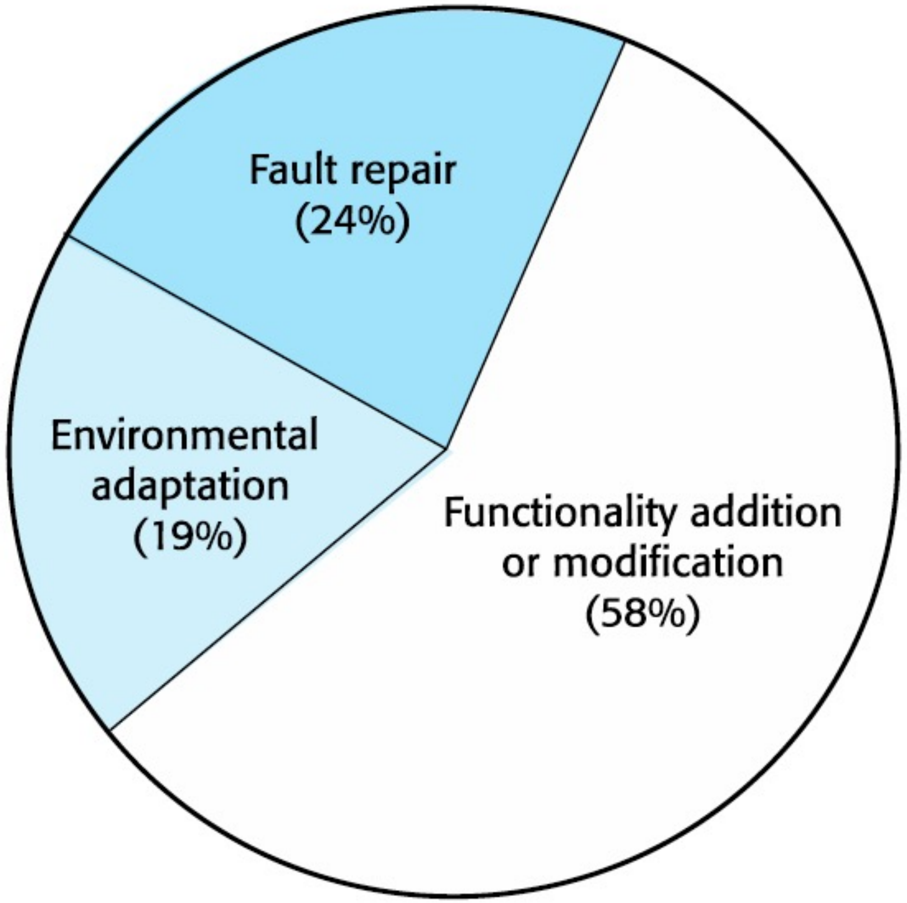
- Fault repairs
  - Changing a system to fix bugs/vulnerabilities and correct deficiencies in the way meets its requirements.

- Environmental adaption
  - 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.

Types of maintenance (2)

- 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.)

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;

Software rot

- Software rot, (i.e, code degradation) also known as bit rot, code rot, software erosion, software decay, or software entropy is either a slow deterioration of software quality over time or its diminishing responsiveness that will eventually lead to software becoming faulty, unusable, or in need of an upgrade.

Refactoring

- Refactoring is the process of making improvements to a program to slow down degradation through change.
- It means 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 should concentrate on program improvement.

Refactoring (2)

- Refactoring is an inherent part of agile methods.
- Refactoring is a continuous process of improvement throughout the development and evolution process.
- It is intended to avoid structure and code degradation that increases the costs and difficulties of maintaining a system

“Bad smells” in program code

- Duplicate code
  - The same or very similar code may be included at different places in a program. This can be removed and implemented as a single method or function that is called as required.
- Long methods
  - If a method is too long, it should be redesigned as a number of shorter methods.

“Bad smells” in program code (2)

- Switch (case) statements
  - These often involve duplication, where the switch depends on the type of a value. The switch statements may be scattered around a program.
- Data clumping
  - Data clumps occur when the same group of data items (fields in classes, parameters in methods) re-occur in several places in a program. These can often be replaced with an object that encapsulates all of the data.

"Bad smells" in program code (3)

- Speculative generality
  - This occurs when developers include generality in a program in case it is required in the future. This can often simply be removed.

Summary

- 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.

Summary (2)

- There are 3 types of software maintenance, namely bug fixing, modifying software to work in a new environment, and implementing new or changed requirements.