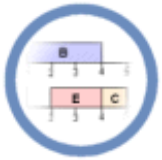




Lecture 13
Reliability (Part 3)

<lecturer, date>

Outline

- What to do about faults?
- Software engineering lifecycle
  - Water fall model
  - Spiral model
  - V-model
  - …
- Software testing
  - Unit testing
  - Integration testing
  - User acceptance testing
  - Performance testing
  - …

What to do about faults?

- Fault tolerance
  - to cope with the effects of faults
  - typically by redundancy

Previous lecture

- Verification and validation
  - to identify and eliminate faults

In this lecture

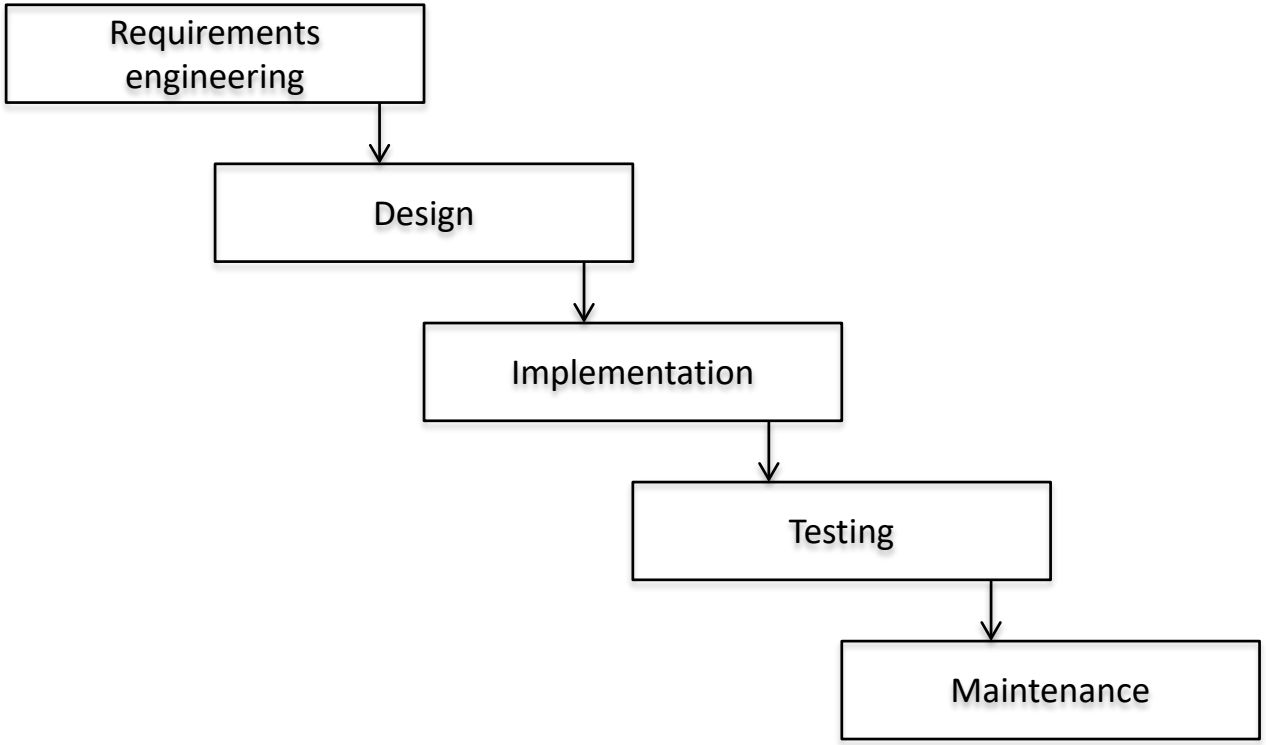
- Safety analysis
  - to focus on the most important faults

Outline

- What to do about faults?
- Software engineering lifecycle
  - Water fall model
  - Spiral model
  - V-model
  - ...
- Software testing
  - Unit testing
  - Integration testing
  - User acceptance testing
  - Performance testing
  - ...

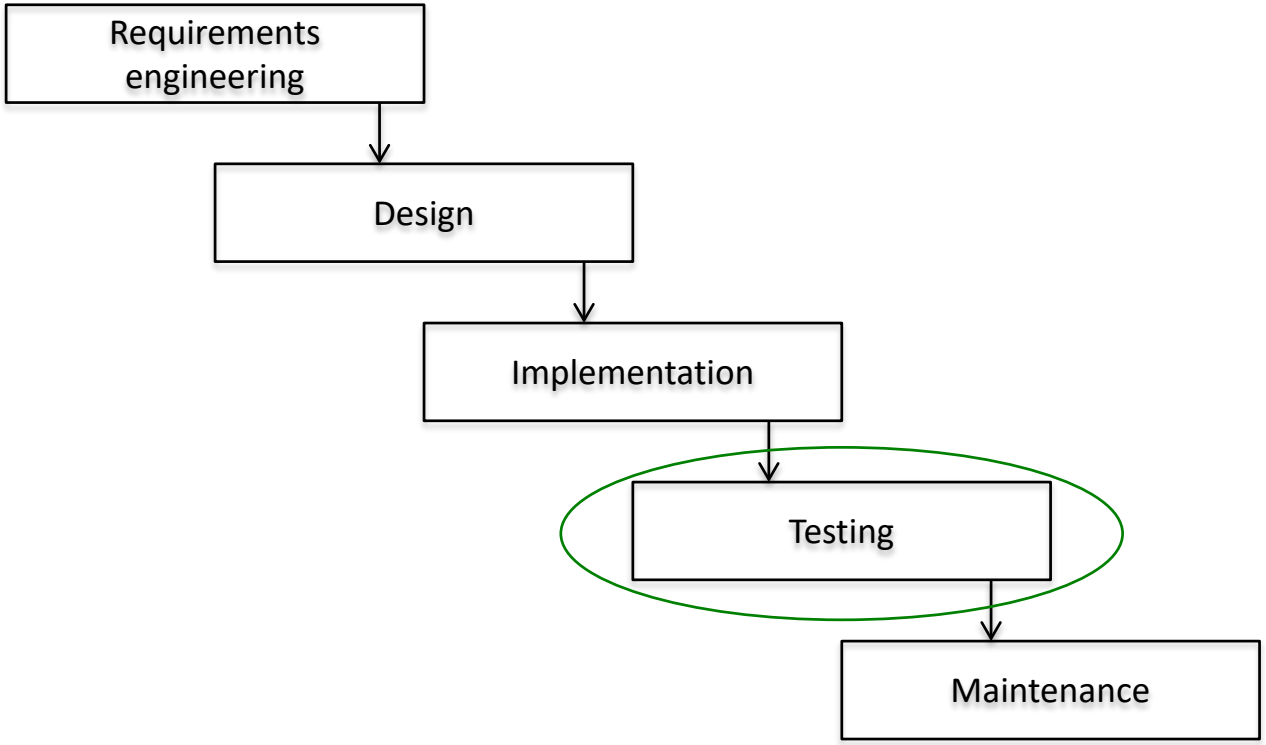
Water fall model

Software design process in which the progress is viewed as flowing steadily downwards like a waterfall sequentially.



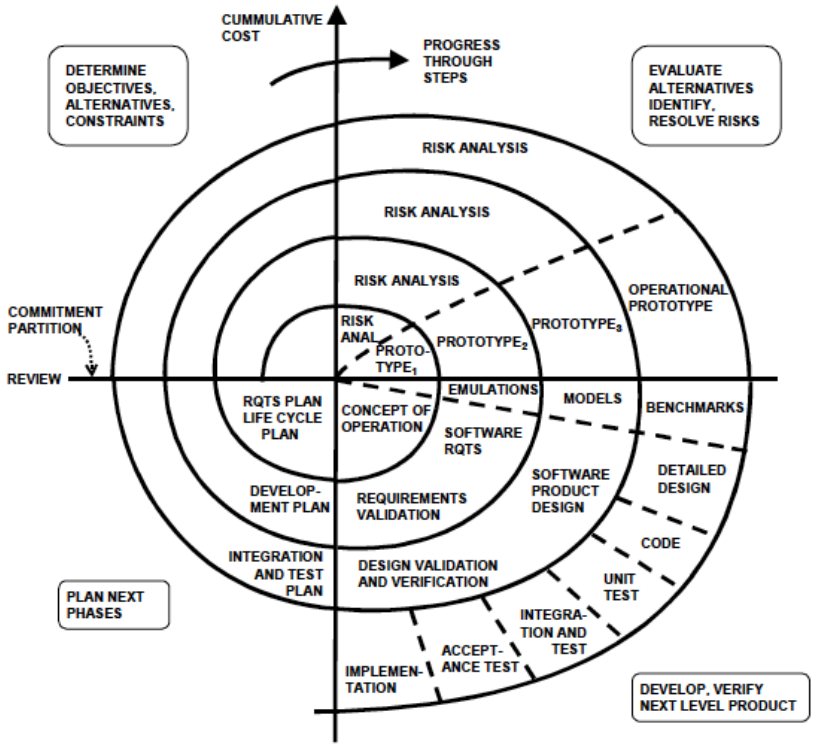
Water fall model

Software design process in which the progress is viewed as flowing steadily downwards like a waterfall sequentially.



Spiral model

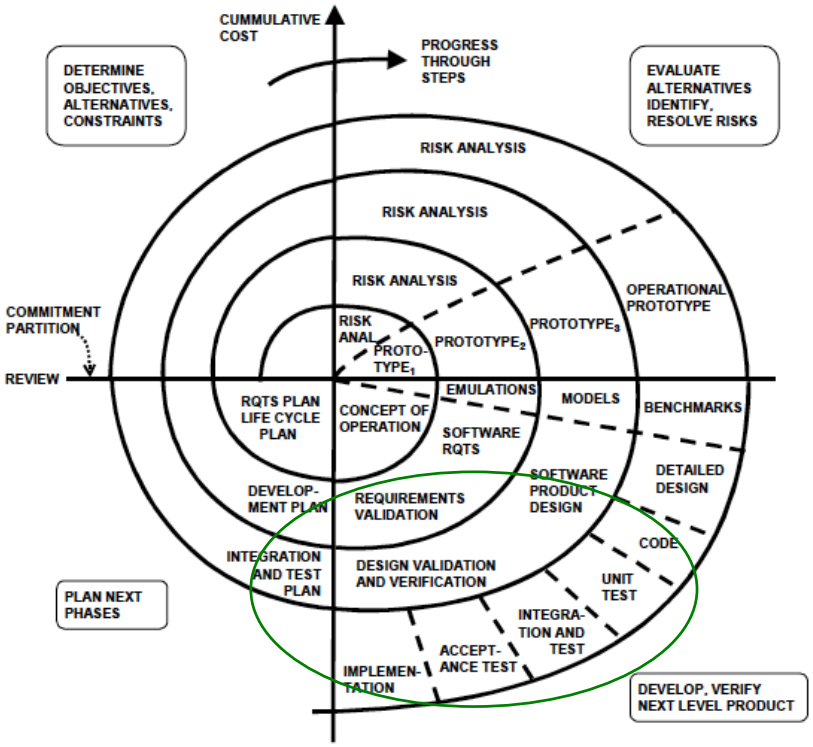
Software design process that enables developers to adopt elements of one or more software development models depending on the risk.



Taken from reference 4

Spiral model

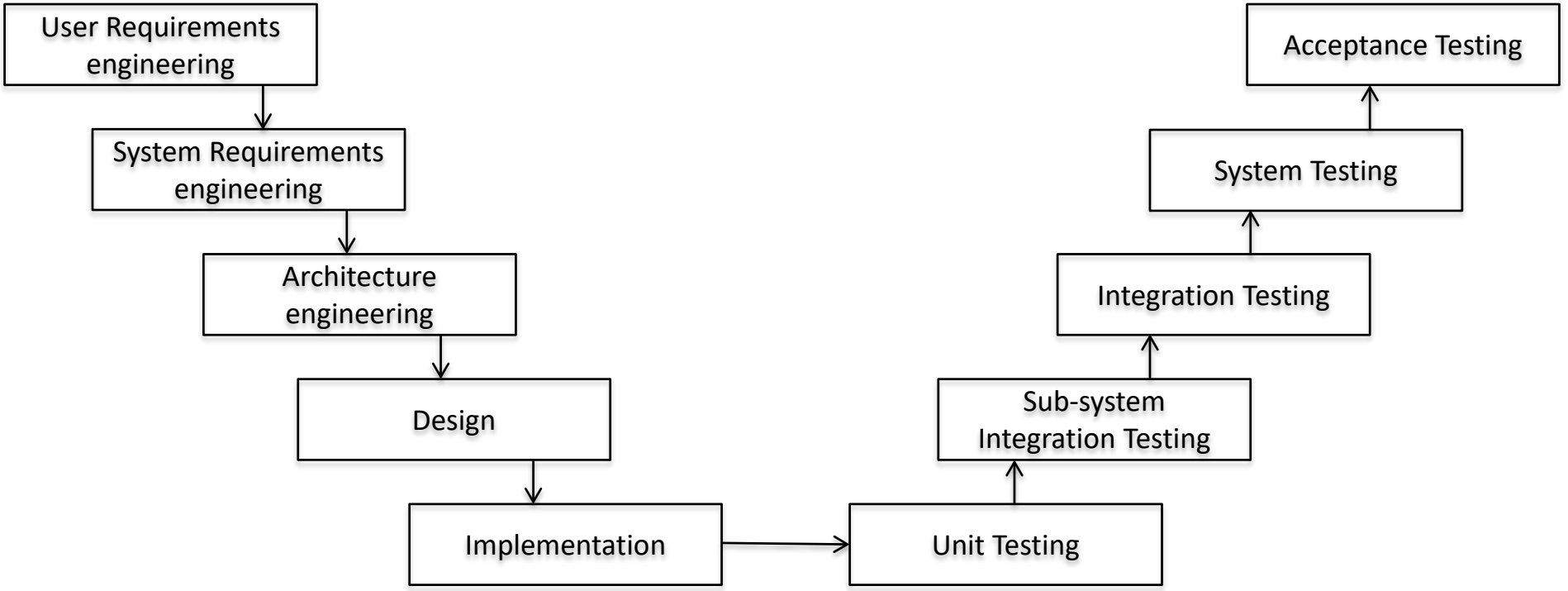
Software design process that enables developers to adopt elements of one or more software development models depending on the risk.



Taken from reference 4

V model

A variant of the waterfall model that emphasizes on verification and validation



Verification and Validation

- The process of ensuring that the software conforms to the specifications.
- Ensures that the system meets the requirements.
- Validation: ensures that the system meets the needs of the customer
- Verification: ensures that the system works “correctly”.

Outline

- What to do about faults?
- Software engineering lifecycle
  - Water fall model
  - Spiral model
  - V-model
  - …
- Software testing
  - Unit testing
  - Integration testing
  - User acceptance testing
  - Performance testing
  - …

Software Testing

- The process of investigating the quality of the software developed
- May involve executing the program, or analyzing the models
- Common types of testing
  - Unit testing: the specific subsystems works as specified
  - Integration testing: the subsystems when put together works as specified
  - User acceptance testing: the system works as desired by a user
  - Performance testing: the system/subsystem meets a specified performance objective

Test Automation Framework

- Test automation is the process of automating the testing process by using special software that can manage the test cases and their execution
- The testing framework is a software that is independent of the software being developed
- Available for testing code, as well as the GUI
- Many tools are available such as JUnit for testing Java programs

Unit Testing

- The process of testing whether the subsystems work as specified
- Involves executing the program for specific inputs
  - Boundary cases, e.g., large values
  - Exceptional cases, e.g., negative numbers
- Tool support
  - Junit: a unit testing framework for Java
  - Many major languages have support for unit testing e.g., Java, ABAP etc

Advantages of Unit Testing

- Enables early detection of faults
  - detect faults in the code and correct them before the complexity of the code increases
- Facilitates easy change
  - change code later and re-run the already prepared test cases
- Facilitates easier integration
  - programmers can develop test cases as they integrate subsystems
  - enables early detection of integration related problems

Unit Testing Java programs using JUnit

```
public class BowlingGame {

/* BowlingGame Score calculator constructor which require string
*as input
 * @param game Expected format "[nn][nn]..[nn]"
*/

public BowlingGame(String game)
        {
        }

/*getScore method returns a score of current Bowling game
*or -1 if  error
*/

public int getScore() {

                }
}
```

The function assertEquals tests whether the score is calculated correctly, i.e., if the calculated score = 81, 103, 110 and -1 for each of the associated cases.

```
import junit.framework.TestCase;

public class BowlingGameTest extends TestCase {

/**
If no game is provided, score should be -1 (error)
*/
public void testEmptyGame() {

        BowlingGame a1 = new BowlingGame("[1,5][3,6][7,2][3,6]
        [4,4][5,3][3,3][4,5][8,1][2,6]");
        assertEquals(81, a1.getScore());

        BowlingGame a6 = new BowlingGame("[10,0][4,6][7,2][3,6]
        [4,4][5,3][3,3][4,5][8,1][2,6]");
        assertEquals(103, a6.getScore());

        BowlingGame d3 = new BowlingGame("[0,10][0,10][0,10]
        [0,10][0,10][0,10][0,10][0,10][3,4][10,0][8,2]");
        assertEquals(110, d3.getScore());

        BowlingGame d = new BowlingGame("[][][][][][][][][][]");
        assertEquals(-1, d.getScore());

        }
}
```

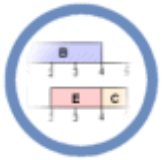
References

1) Basic concepts and taxonomy of dependable and secure computing, Avizienis, A. ; Laprie, J.-C. ; Randell, B. ; Landwehr, C., IEEE Transactions on Dependable and Secure Computing, 2004
2) An Experimental Evaluation Of The Assumption Of Independence In Multi-Version Programming, J. C. Knight , N. G. Leveson, IEEE Transactions on Software Engineering, 1986
3) N. G. Leveson, “High-pressure steam engines and computer software,” in Proceedings of the 14th International Conference on Software Engineering, 1992
4) B. Boehm, Spiral Development:Experience, Principles, and Refinements,http://www.sei.cmu.edu/reports/00sr008.pdf
5) Joint Software Systems Safety Engineering Handbook, http://www.system-safety.org/Documents/SOFTWARE_SYSTEM_SAFETY_HDBK_2010.pdf
6) Writing and running JUnit tests, http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2FgettingStarted%2Fqs-junit.htm

References

7) Architectures of test automation, Cem Kaner, http://www.kaner.com/pdfs/testarch.pdf





Lab 13
Reliability (Part 3)

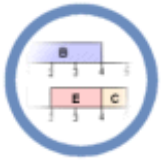
<lecturer, date>

Description

- Write a mobile application that takes temperature and time (hh:mm) as input, sorts the values in the increasing order of time and sets the temperature of the water tank controller when the corresponding time is reached.

- Test your application by writing unit test cases for the sorting function
- Test all possible cases such as:
  - Boundary conditions (such as an already sorted list)
  - Incorrect values (such as negative time)
  - Think of more ways of '*breaking*' your sorting algorithm





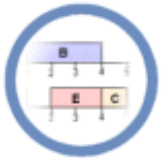
Seminar 13
Reliability (Part 3)

<lecturer, date>

Description

- Read the following article and write a short report along with your reflections:
  - On the Effectiveness of Test-first Approach to Programming Erdogmus, H, IEEE Transactions on Software Engineering, 2005, Available from: http://nparc.cisti-icist.nrc-cnrc.gc.ca/npsi/ctrl?action=shwart&index=an&req=5763742&lang=en
  - An Analysis and Survey of the Development of Mutation Testing , Yue and Mark Harman, IEEE Transactions on Software Engineering, 2011, Available from: http://crest.cs.ucl.ac.uk/fileadmin/crest/sebasepaper/JiaH10.pdf
- Discuss the article in the class.





Mini-project 13
Reliability (Part 3)

<lecturer, date>

Description

- Form groups of two, and run the test cases that you developed earlier in the lab on each others' code
  - Write a report summarizing the results
  - Discuss the results in the context of N-version programming
    - Did the two code fail on the same test case?
    - Is N-version programming effective?