Theme I
Software Processes

Quality in Software Engineering
Focus on Reliability, Maintainability and Flexibility

Learning Objective

- develop the foundations for practices and skills to develop reliable software by presenting basic definitions and terminology and testing as a technique to achieve this
- establish solid terminology of the different qualities or characteristics influencing ease of change, allowing to discuss techniques at a precise level

Table of content

- Software quality?
- Reliability
- Testing Terminology
- Automated Testing
- Maintainability
  - subqualities of maintainability
- Flexibility

What is quality?

Meaning of Software Quality

- varying views on meaning of software quality
  - reliability, portability
  - how well a development process is followed
  - => organization needs to define what quality means together with customers
- A possible definition:
  The more closely a software product meets its specified requirements, and those requirements meet the wants and the needs of its customers, the higher the quality is.

Working Software

- One important aspect is that the software does not fail to allow users to perform their work.

- **Reliability (ISO 9126)**
  The capability of the software product to maintain a specified level of performance when used under specified conditions.

How to achieve *reliability*?

- Evolution in programming languages
  - from Fortran to Java and on...
  - type system
- Review techniques
- Testing

Testing Terminology

- **Defect**

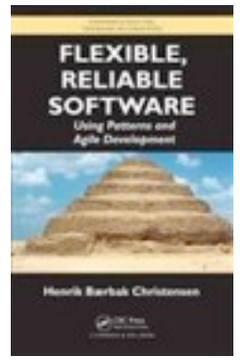
  A deviation from what is required in the respective phase of the software application

- Quality goals:
  - remove as many defects as is reasonably possible before project is completed
  - remove as many of these defects as early in the development process as possible

Defect Repair Cost

Rehearsal

Testing Terminology



- **Testing**
  Testing is the activity of executing a program with the intent of finding a defect.
  A successful test is one that finds a defect!

- **Test Case**
  A test case is a definition of input values and expected output values for the unit under test.

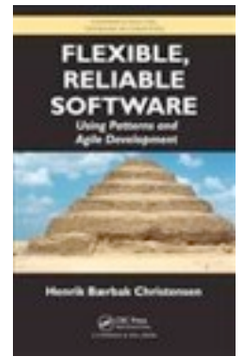
- **Unit under Test**
  The unit under test is some part of the system that we consider to be a whole.

Golden Rules of Testing

- Goal of testing:
  - maximize the number and severity of defects.

- Limits of testing
  - testing can only determine the presence of defects, never their absence.

Example

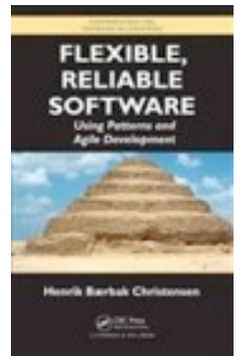



```
public class Date {
  public enum Weekday {
      MONDAY, TUESDAY, WEDNESDAY,
      THURSDAY, FRIDAY,
      SATURDAY, SUNDAY };

  /**
   * Construct a date object.
   * @param year the year as integer, i.e. year 2010 is 2010.
   * @param month the month as integer, i.e. januar is 1, december is 12.
   * @param dayOfMonth the day number in the month, range 1..31.
   * PRECONDITION: The date parameters must represent a valid date.
  */
  public Date(int year, int month, int dayOfMonth) {}

  /**
   * Calculate the weekday that this Date object represents.
   * @return the weekday of this date.
   */
   public Weekday dayOfWeek() {
     // Fake implementation, only for demonstrating testing.
     return Weekday.SATURDAY;
   }
}
```

How complete is the above implementation?

Give some test cases!!!

Test Case Table

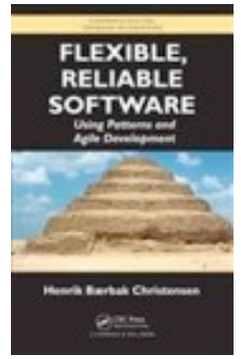


- shows:
  - unit under test
  - input values (all of them)
  - expected output
  - Example:

Unit under test: dayOfWeek	
Input	Expected output
year=2008, month=May, dayOfMonth=19	Monday
year=2008, month=Dec, dayOfMonth=25	Thursday
year=2010, month=Dec, dayOfMonth=25	Saturday

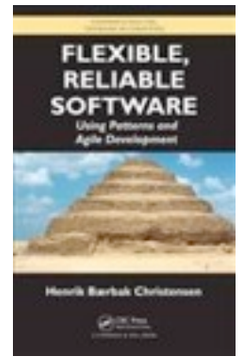
- A test case may:
  - pass: computed output equals expected output
  - fail

Final Terminology



- **Manual Testing**
  is a process in which suites of test cases are executed and verified manually by humans.
- **Automated Testing**
  is a process in which suites of test cases are executed and verified automatically by computer programs.
- **Regression Testing**
  is the repeated execution of test suites to ensure they still pass and the system does not regress after a modification.

Final Terminology



- **Production code**
  The production code is the code that defines the behavior implementing the software's requirements
- **Test code**
  The test code is the source code that defines the test cases for the production code.
- *Why can I improve reliability by writing more code?*
  - *Fact*: I know I make mistakes when I code.
  - *Fact*: To verify that my code has no defects I write more code.
- *Is this not absurd??*

Flexibility and Maintainability

- What is this?

```
public class X{private int y;public X(){y = 0;}public int z(){
return y;}public void z1(int z0){y += z0;}public static void main(
String[] args){X y=new X();y.z1(200);y.z1(3400);System.out.println
("Result is "+ y.z());}}
```

- What does it do?
- What abstraction is this code implementing?

Issue?

- The customers / executing software do not care if the code is
  - Readable / understandable / well documented
  - As long as it serves its purpose well...

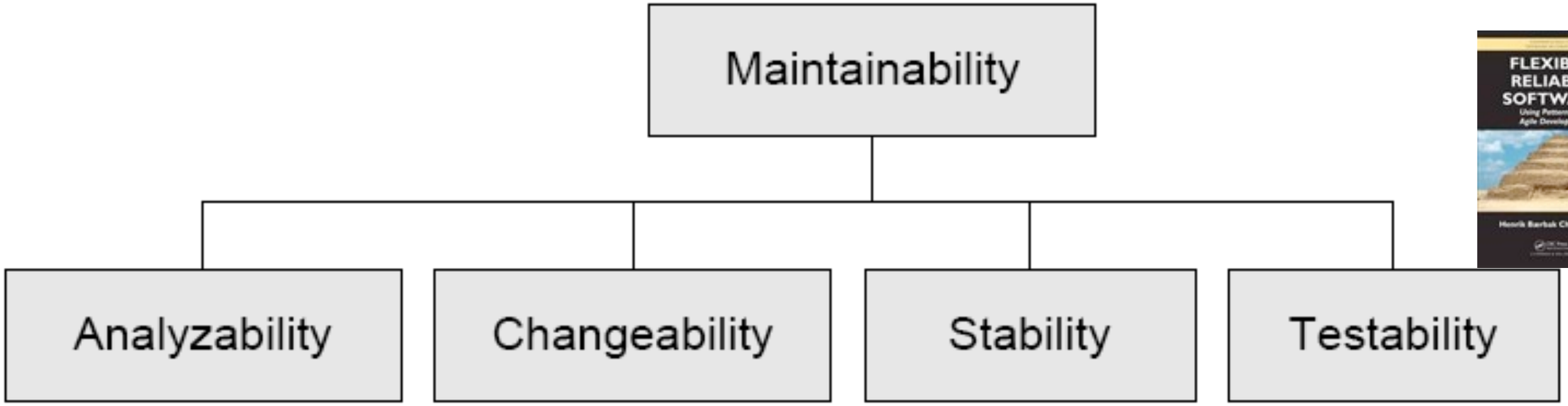
- However, developers do
  - Unless you are about to quit tomorrow

Software needs to be maintainable!!

Maintainability

- **Maintainability (ISO 9126)**
  The capability of the software product to be modified. Modifications may include corrections, improvements or adaptations of the software to changes in environment, and in requirements and function specifications.

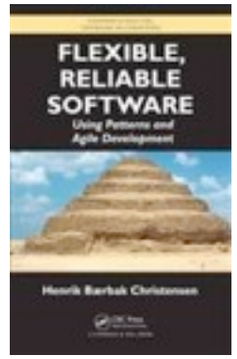
- Maintainability is a quality that our code has to a varying degree
  - Low maintainability -> high maintainability



- **Analyzability (ISO 9126)**
  The capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified.

Can I understand the code?
- Indentation
- Naming conventions for classes/methods
- Useful comments and documentation

- **Changeability (ISO 9126)**
  The capability of the software product to enable a specified modification to be implemented.



Cost of modifying the code
160x45 maze?

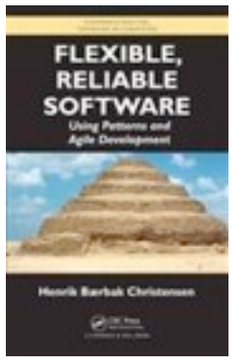
```
public class Maze {
  private boolean[] isWall = new boolean[2000];
  public void print() {
    for (int c = 0; c < 80; c++) {
      for (int r = 0; r < 25; r++) {
        char toPrint = (isWall[r*80+c] ? '#' : ' ');
        System.out.print(toPrint);
      }
      System.out.println();
    }
  }
  public void generate() {
    // generate the maze
  }
}
```

- **Stability (ISO 9126)**
  The capability of the software product to avoid unexpected effects from modifications of the system.

  - Changing a software unit may lead to failures in other units.

- **Testability (ISO 9126)**
  The capability of the software product to enable a modified system to be validated.

  - Everything can be tested?



- **Flexibility**

  The capability of the software product to support added/enhanced functionality purely by adding software units and not by modifying existing software units.

```
public class PointOfSale {

   private State state;
   public double calculateSalesTax(double price){
       switch (state) {
           case CALIFORNIA: return price * 8.25 / 100.0;
           case NEVADA: return price * 8.10 / 100.0;
           default:
              throw new RuntimeException("Unknown state");
       }
   }
   public enum State{
   CALIFORNIA, NEVADA}
   //rest of functionality omitted
}
```

Adding point of sale system for a new state?

Coupling and Cohesion

Metrics for maintainability of software

Ragnhild Van Der Straeten - ULB - Software Engineering and Project Management - 2012/2013

Measuring Software

- Programmers with some experience has a sense of good and bad software.
- Kent Beck and Martin Fowler also talk about *code smell*. But... what is good and what is bad?

- *Measure software according to some defined metric.*
- **Metric**
  Numerical measures that quantify the degree to which software or a process possesses a given attribute.

Examples of Metrics

- Defects/KLOC
- Average module size
- mean time to failure
- customer problems
- customer satisfaction
- ...

- Collected and analyzed throughout software project.

Coupling and Cohesion

- *Cohesion* is a measure of how strongly related and focused the responsibilities and provided behaviors of a software unit are.
  - The higher the better.
- *Coupling* is a measure on how strongly dependent one software unit is on other software units.
  - The weaker the better.

Cohesion and Coupling Heuristics

- It is good practice:
  - For two classes to either be not dependent on one another, or for one class to be only dependent on the interface of another class.
  - To keep attributes and the related methods together in one class.
  - For a class to capture one and only one abstraction - unrelated information to be kept in separate classes.
  - To distribute the system intelligence as uniformly as possible

Kinds of Class Coupling

- X inherits from Y.
- X has an attribute of class Y.
- X has a template attribute with a parameter of class Y.
- X has a method with an argument of class Y.
- X knows of a global variable of class Y.
- X knows of a method containing a local variable of class Y.
- X is friend of Y (in C++).

Trade-off

- Maintainable software generally has weak coupling and high cohesion.

- Weak coupling means one change does not influence all other parts of the software
  - lowering cost of change
- High cohesion means that a change is likely localized in a single subsystem, easier to spot
  - lowering the cost of change

Trade-off

- Need to find the right balance between coupling and cohesion.
  - Example: Making a subclass increases coupling (bad), but increases cohesion (good, when done right).
    - So adding tons of classes each overriding a single method might not be a good idea, even if the subclass semantics is right.
- Plays well together with encapsulation.