AARHUS UNIVERSITET

Software Engineering and Architecture

Flexibility and Maintainability

AARHUS UNIVERSITET

What is this?

- So – all you smart guys and girls – 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 every-day abstraction is this code implementing?
  - (Note: the code fragment in the book is incorrect ☹)

The point

- 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
  - Or in a consulting company ☺

AARHUS UNIVERSITET

What developers want...

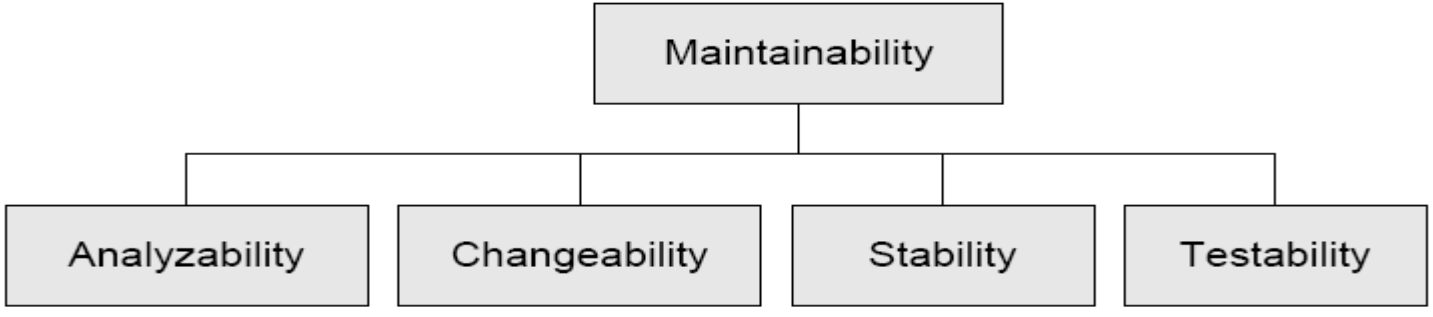
- We need software to be *maintainable*

> Definition: **Maintainability (ISO 9126)**
>
> The capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications.

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

Aspects

- Maintainability is influenced by a lot of sub qualities.



AARHUS UNIVERSITET

Analyzability

Definition: 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.

- Basically: can I *understand* the code?
  - Indentation
  - Intention-revealing names of methods
  - Follow language conventions
  - Useful comments and JavaDoc
  - Training!
    - To spot e.g. Design Patterns

AARHUS UNIVERSITET

Changeability

Definition: Changeability (ISO 9126)

The capability of the software product to enable a specified modification to be implemented.

- *Cost* of modifying the code
  - 160x45 maze?

Magic Constants

```
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

Definition: Stability (ISO 9126)

The capability of the software product to avoid unexpected effects from modifications of the system.

- In BASIC all variables are global
  - do not store some global property in variable i !
    - Why not?
- What ‘stability’ enhancing features have Java?

Testability

Definition: Testability (ISO 9126)

The capability of the software product to enable a modified system to be validated.

- Everything can be tested – right?
  - Not!
- Ariane rocket guidance system bug
  - Found when they launched it...
  - Overflow error due to 64-bit to 16-bit conversion
- Increasing testability is a major learning goal in SWEA

Flexibility

Definition: Flexibility

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

- A main theme of FRS !
- *Change by addition, not by modification...*

AARHUS UNIVERSITET

Coupling and Cohesion

Two metrics highly correlating to maintainability of software

To measure software

- Programmers with some experience has a sense of *good* and *bad* software.
- Some of the ”heavy guys” like Kent Beck and Martin Fowler also talks about *code smell*.
- But... what is *good* and what is *bad*?

- Not very scientific anyway ☺

- It is better to *measure* software according to some defined metric.

Examples of metrics

- A very simple, widely used, and next to useless metric is **kloc** = Kilo Lines of Code. It simply measures the quantity of code.

- Useless?
  - Is 2kloc better than 1kloc?

AARHUS UNIVERSITET

A maintainability measure

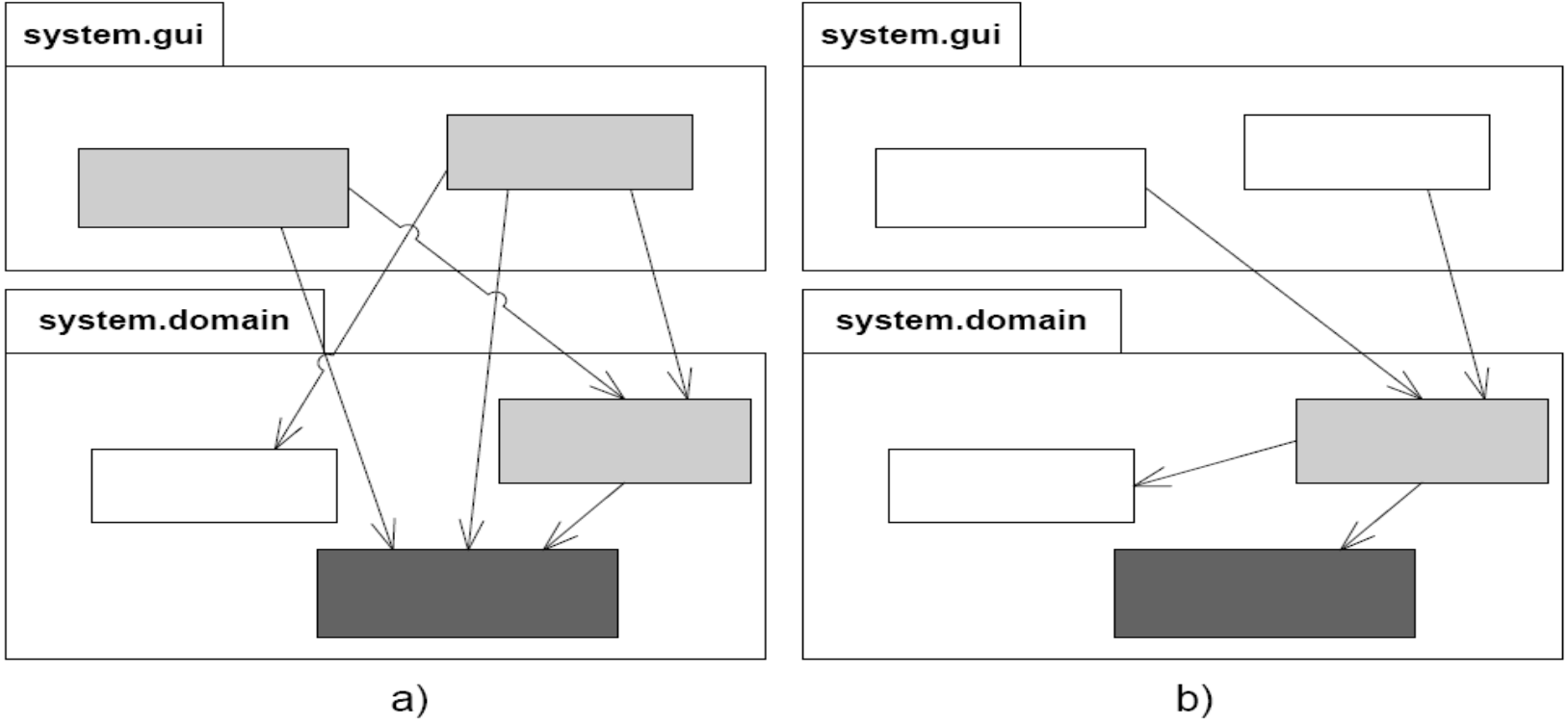
- Coupling (da: kobling):

> Definition: **Coupling**
>
> Coupling is a measure of how strongly dependent one software unit is on other software units.

- unit = a well delimited unit of code: class, package, module, method, application, etc.

- Low/loose coupling: few dependencies
- High/tight coupling: lot of dependencies

Example



AARHUS UNIVERSITET

Exercise

- Name some language constructs or techniques that generate dependencies between two classes.

- ?

War story

- In the ABC research project, a knowledge based system was able to guess at human activities based on knowledge of location of objects in a hospital setting.
- For instance co-location of a medicine tray, a nurse and a patient would trigger a ”the patient is receiving medicine” activity proposal.
- The ID used in the knowledgebase was RFID tag ID.
- Later, some programmer changed ID for persons to CPR identity instead ☺.

Rule of Thumb:

- Not that surprising:
- **Assign responsibility so coupling is low**
- Because
  - Local change has no/less impact
  - Easier to understand modules in isolation
  - Higher probability of reuse with few dependencies

Cohesion

- Cohesion (da: kohæsion/binding/samhørighed):

> **Definition: Cohesion**
>
> Cohesion is a measure of how strongly related and focused the responsibilities and provided behaviors of a software unit are.

- Example:
  - Unit X: all classes that begin with letters A, B, and C
  - Unit Y: all classes related to booking a flight seat

Rule of Thumb

- Also not surprising:

- **Assign responsibility so cohesion is high**

Discussion

- 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
  - lowing the cost of change

AARHUS UNIVERSITET

Law of Demeter

- A very concrete “law” that addresses the coupling measure is ***Law of Demeter****:*
    - *Do not collaborate with indirect objects*
- Also known as
    - ***Don’t Talk to Strangers***
- Example
- ~~p.getX().getY().getZ().fecthA().doSomething();~~

Lieberherr

- Within a method, messages should only be sent to
  - this
  - a parameter of the method
  - an attribute of this
  - an element of a collection which is an attribute of this
  - an object created within the method

- In other words: “never two dot’s in a call” ☺

War story

- Major Danish IT company
  - problem: dynamic configuration of user interface elements
  - solution:
    - configuration parameters in property file
    - read at run-time
    - if (dialogX.panelY.listboxZ.color == NONE) { ...}
  - ☹

AARHUS UNIVERSITET

Then what?

- Rule of Thumb:
  - *Assign the responsibility to the client's direct object to do the collaboration with indirect objects*
- Thus
  - `order.getItem(3).getPrice().addTax()`
  - should be replaced by
  - `order.addTaxToItem(3);`
- Liabilities?

AARHUS UNIVERSITET

Then what?

- Exercise:
  - Do we see this in the pay station?
  - In HotGammon/HotCiv?

- *Consequences*
  - ☺ Law of Demeter lowers direct coupling
  - ☹ Interfaces may bloat with too many method