Software Architecture Quality Attributes

Good or Bad?

Measurable criterions required...

What does “good” mean?

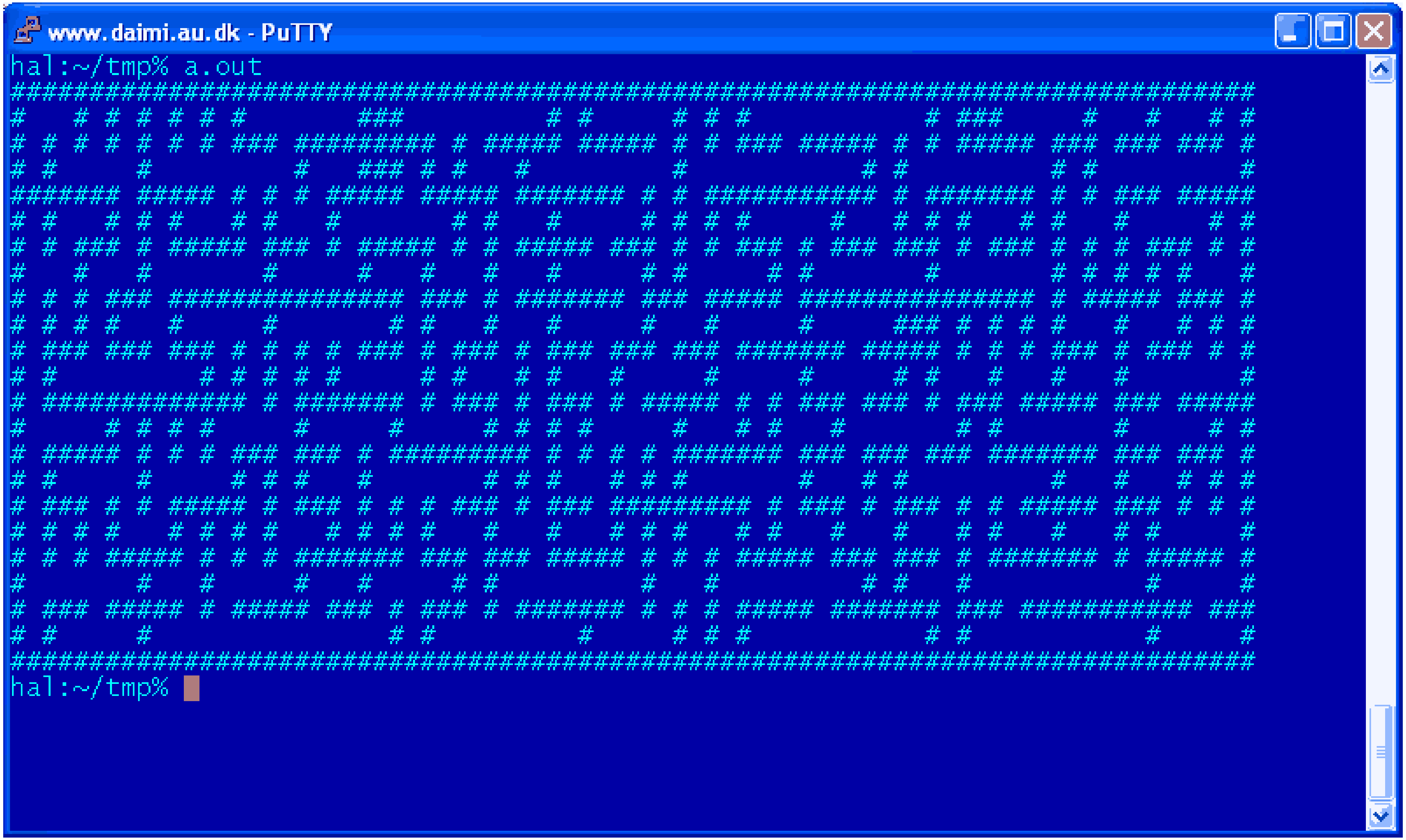
Question: Is this little C program an example of *good* or *bad* software?

```
int a[1817];main(z,p,q,r){for(p=80;q+p-80;p-
=2*a[p])for(z=9;z--;)q=3&(r=time(0) +r*57)/7,q=q?q-1?q-2?1-p%79?-
1:0:p%79-77?1:0:p<1659?79:0:p>158?-79:0,q?!a[p+q*2 ]?
a[p+=a[p+=q]=q]=q:0:0;for(;q++-1817;)printf(q%79?"%c":"%c\n"," #"[!
a[q-1]]);}
```

Exercise 1: Argue that this is a good program!

Exercise 2: Argue that this is a bad program !

(What did the C program do?)



```
hal:~/tmp% a.out
###############################################################################
#   # # # # # #       ###         # #     # # #           # ###     #   #   # #
# # # # # # # ### ######### # ##### ##### # # ### ##### # # ##### ### ### ### #
# #     #         #   ### # #   #         #           # #         # #         #
####### ##### # # # ##### ##### ####### # # ########### # ####### # # ### #####
# #   # # #   # #   #       # #   #     # # # #     #   # # #   # #   #     # #
# # ### # ##### ### # ##### # # ##### ### # # ### # ### ### # ### # # # ### # #
#   #   #       #     #   #   #   #     # #     # #       #       # # # # #   #
# # # ### ############### ### # ####### ### ##### ############### # ##### ### #
# # # #   #     #       # #   #   #     #   #     #     ### # # # #   #   # # #
# ### ### ### # # # # ### # ### # ### ### ### ####### ##### # # # ### # ### # #
# #         # # # # #     # #   # #   #     #     #     # #   #   #   #       #
# ############# # ####### # ### # ### # ##### # # ### ### # ### ##### ### #####
#     # # # #     #     #     # # # #     #   # #   #       # #       #     # #
# ##### # # # ### ### # ######### # # # # ####### ### ### ### ####### ### ### #
# #     #     # # #   #       # # #   # # #       #   # #         #   #   # # #
# ### # # ##### # ### # # # ### # ### ######### # ### # ### # # ##### ### # # #
# # # #   # # # #   # # # #   #   #   # # #   # #   #   #   # #   #   # #     #
# # # ##### # # # ####### ### ### ##### # # # ##### ### ### # ####### # ##### #
#       #   #     #   #     # #         #   #         # #   #           #     #
# ### ##### # ##### ### # ### # ####### # # # ##### ####### ### ########### ###
# #     #               # #         #     # # #           # #           #     #
###############################################################################
hal:~/tmp%
```

The need for measuring

The server should be highly available...

***My** software is really reusable!*

Our high performance server will...

These are simply claims ☺

Actual measurements on well defined scale is better...

Quality Attributes

The problem about ”good” or ”bad” is that they are subjective measures...

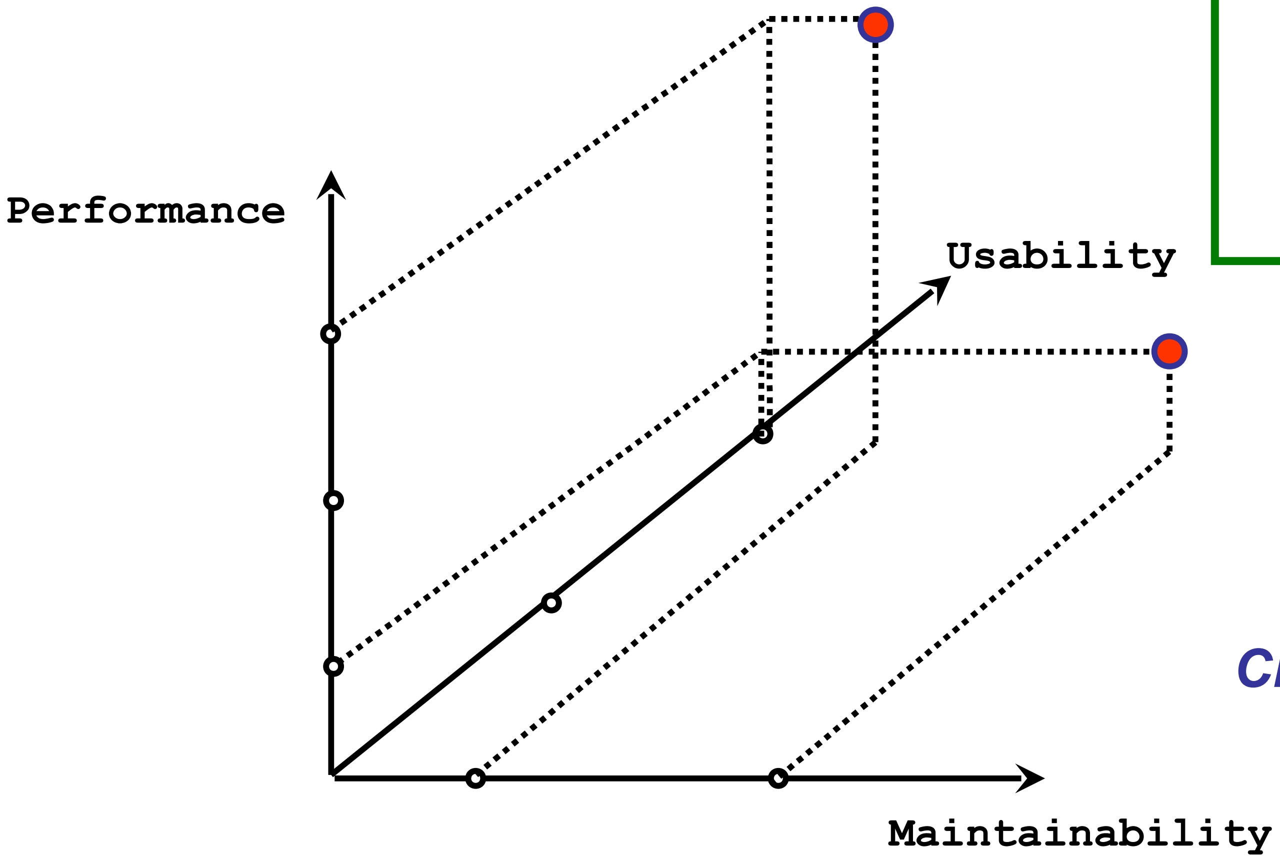
We need to *measure* our software. This requires

- that we define the aspects/**qualities** we measure
- that we agree on some kind of scale: a **metric**

Quality attributes (da: kvalitets-attributter)

AARHUS UNIVERSITET

Measuring quality



Quality Framework

Quality Attribute

Metric

Measurement

Choose alternatives

’Quality communities’

One aspects of qualities is that most of them have dedicated research communities associated:

- performance freaks (algorithm people, database, ...)
- usability freaks (HCI – human computer interface)
- security freaks
- cost freaks (managers ☺)
- reusability freaks (pattern community ☺)

...which has lead to lack of common vocabulary…

- user input, attack, event, failures, are all *stimulus*

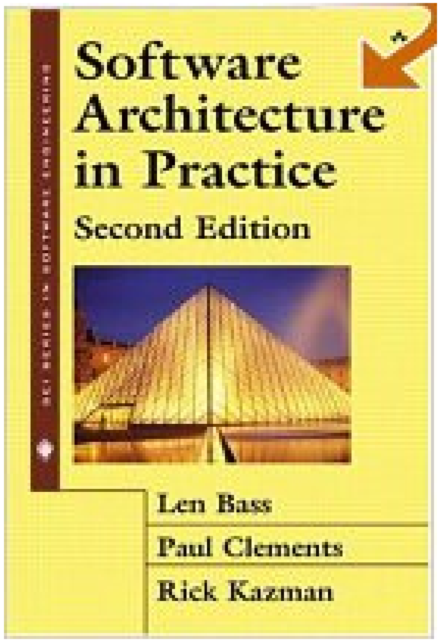
We need to provide common ground

SAiP's Contribution

The book *Software Architecture in Practice* has defined a framework that allow different architecturally qualities to be expressed in a similar form: A quality framework

Quality Attributes: set of qualities to consider

Quality Metric: A technique for measuring them



Quality framework (Bass et al.)

System quality attributes

- Availability
- Modifiability
- Performance
- Security
- Testability
- Usability

Business qualities

- Time to market
- Cost
- Projected lifetime
- Targeted market
- Roll-out schedule
- Integration with legacy sys.

Architectural qualities

- Conceptual integrity
- Correctness and completeness
- Buildability

Exercise

System quality attributes

- Availability
- Modifiability
- Performance
- Security
- Testability
- Usability

Which of these will have the greatest impact on your professional and personal lives?

Business qualities

- Time to market
- Cost
- Projected lifetime
- Targeted market
- Roll-out schedule
- Integration with legacy sys.

Architectural qualities

- Conceptual integrity
- Correctness and completeness
- Buildability

The System Qualities

Availability

- Concerned with the **probability that the system will be operational when needed**

Modifiability

- Concerned with the **ease with which the system supports change**

Performance

- Concerned with **how long it takes the system to respond** when an event occurs

The System Qualities

Security

- Concerned with the systems **ability to withstand attacks/threats**

Testability

- Concerned with the **ease with which the software can be made to demonstrate its faults**

Usability

- Concerned with **how easy it is for the user to accomplish a desired task** and the kind of user support the system provides

“We want them all”

Qualities in conflict

The conflict of qualities

Many qualities are in direct conflict – they must be balanced !

- modifiability and performance
  - many delegations costs in execution speed – and memory footprint
- cost and reusability
  - highly flexible software costs time, effort, and money
- security and availability
  - availability through redundancy – increase opportunities of attack
- etc.

Design Patterns in Perspective

Examples of Good turning Bad

Iteration

In a **distributed** system, the clients need to iterate over all order lines in a order object...

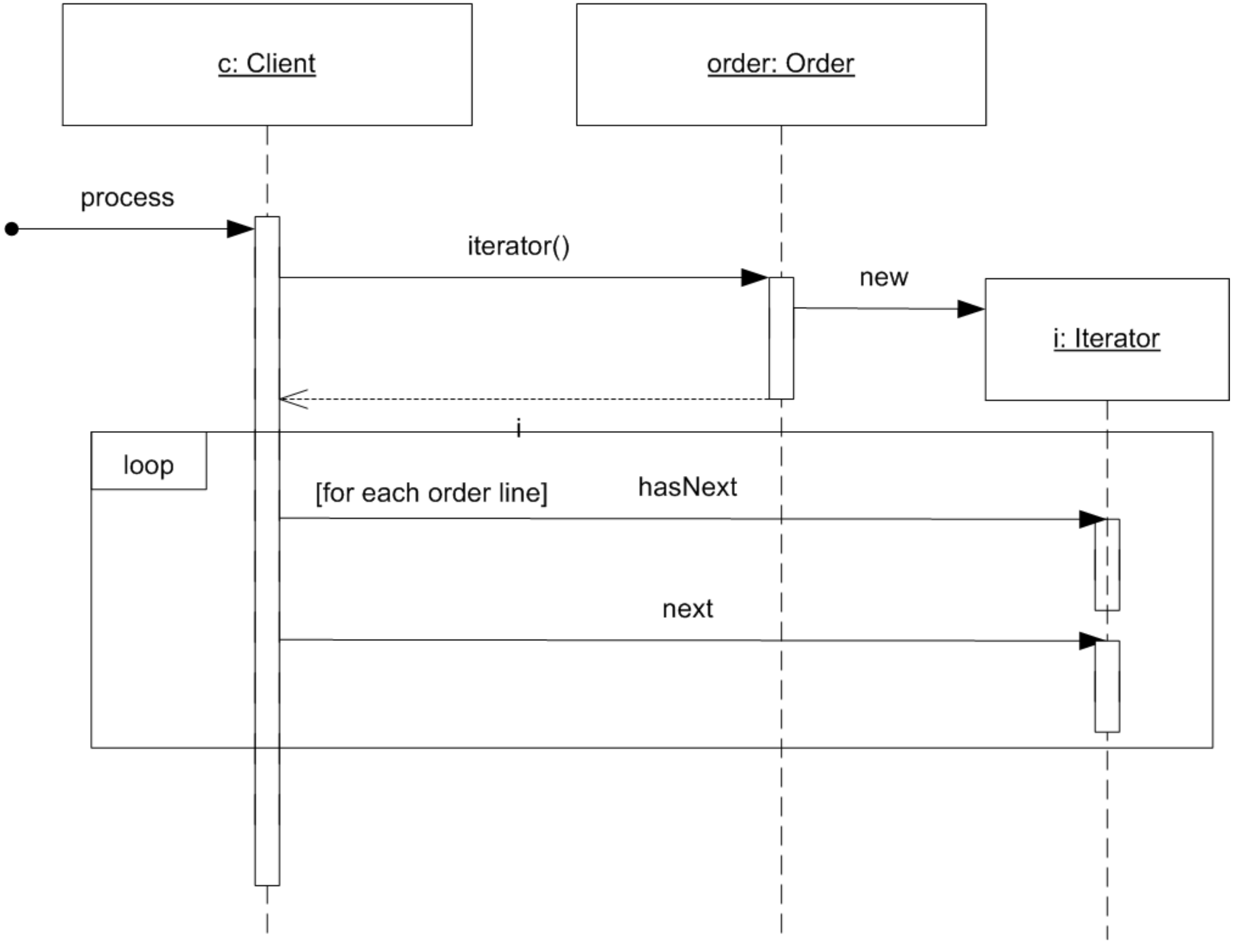
```
Iterator i = order.iterator();
while ( i.hasNext() ) {
  OrderLine entry = (OrderLine) i.next();
  [process entry]
}
```

Using RMI – Remote Method Invocation – the code looks exactly the same even when the Order object is on the server side !!! Great!

Iterator is a nice, flexible, design pattern ☺

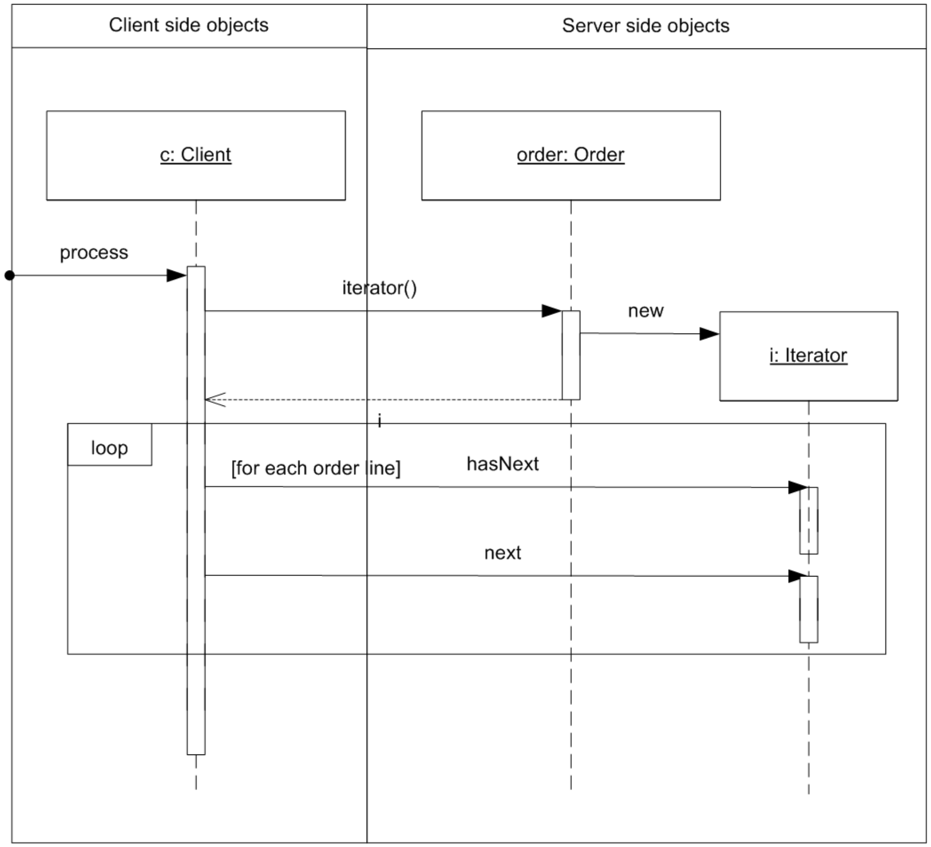
The sequence diagram

All is fine ...



... but

Let us consider the **deployment** of objects



... what about performance ?

Message-call is much more expensive over a network

- 2002 data: 20.000 msg/sec local; 220 msg/sec remote
- 90-times slow-down – probably even worse today.

The iterator pattern ensures an extreme slow-down compared to transfering all order line objects in a single network package !

Another example...

Singleton

- Ensure a class only has one instance, and provide a global point of access to it.

However, a singleton in a distributed system is a major headache!

- one server becomes two servers with a load balancer for scalability
  - now there are two singletons !
- really only one singleton
  - a major performance bottle-neck to ensure the system will never ever scale !

Actually – many consider singleton an anti-pattern!

Maintainability in Details

A key quality in design patterns, frameworks, and compositional designs...

(Repeating a previous slides show ☺)

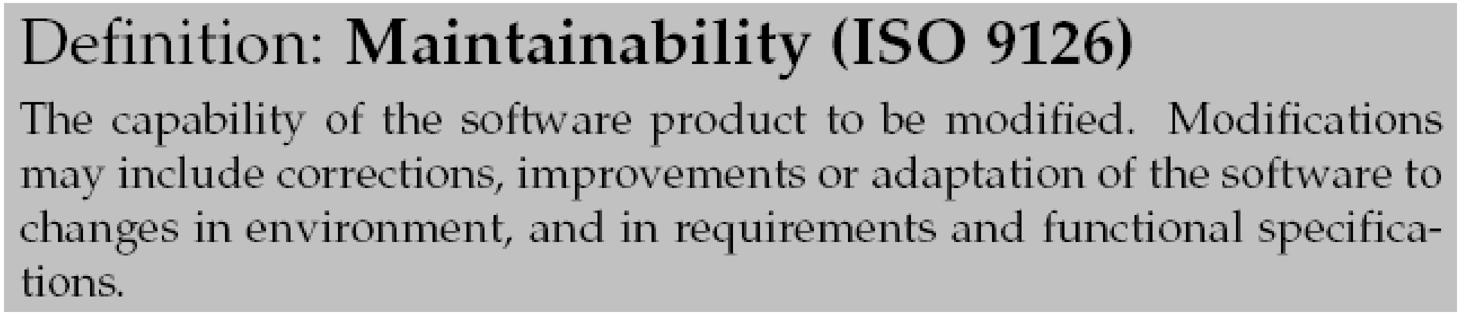
The Bass Framework

The Bass framework is rather coarse grained.

Another, much more detailed, framework is the ISO 9126 standard.

In particular it restate most qualities into *sub qualities.*

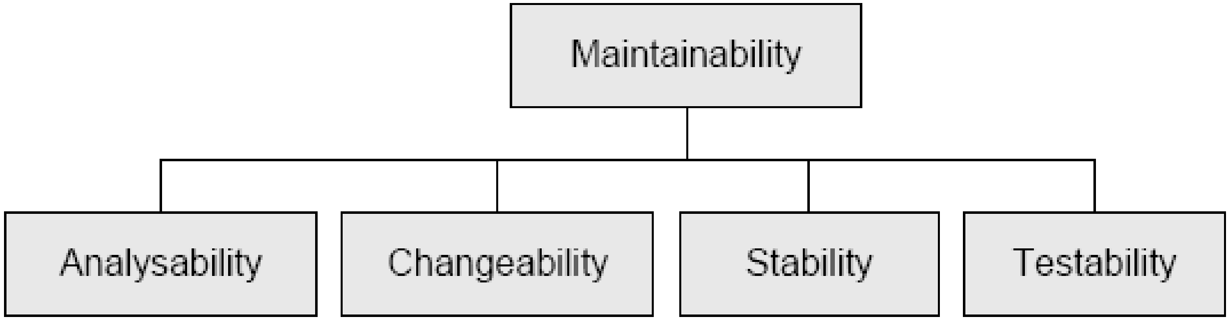
Sub Qualities of Maintainability



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.

Compare: *cost of ease of change (Bass et al.)*



Maintainability

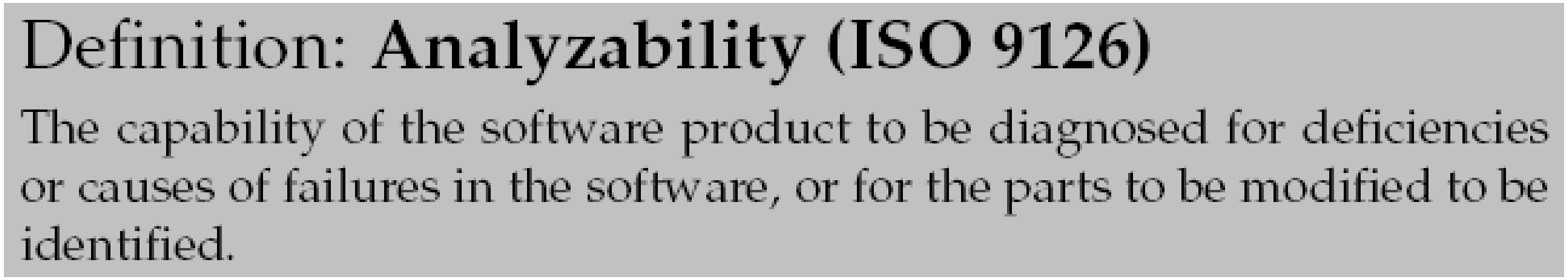
Analysability

Changeability

Stability

Testability

Analyzability

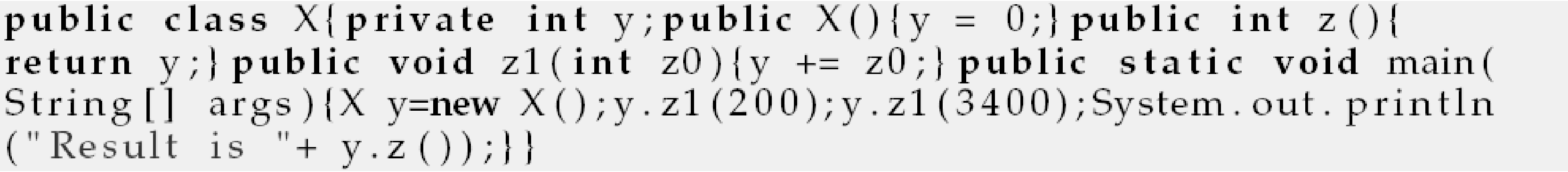


Definition: Analyzability (ISO 9126)

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.

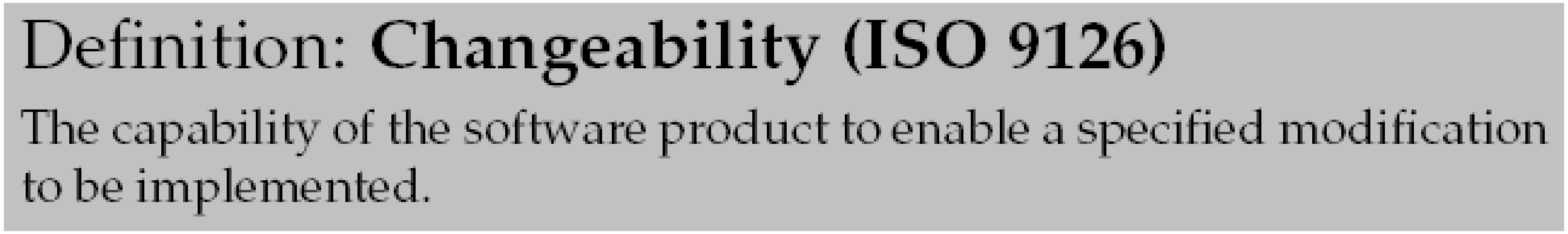
Examples:

- The Account program



```
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());}}
```

Changeability



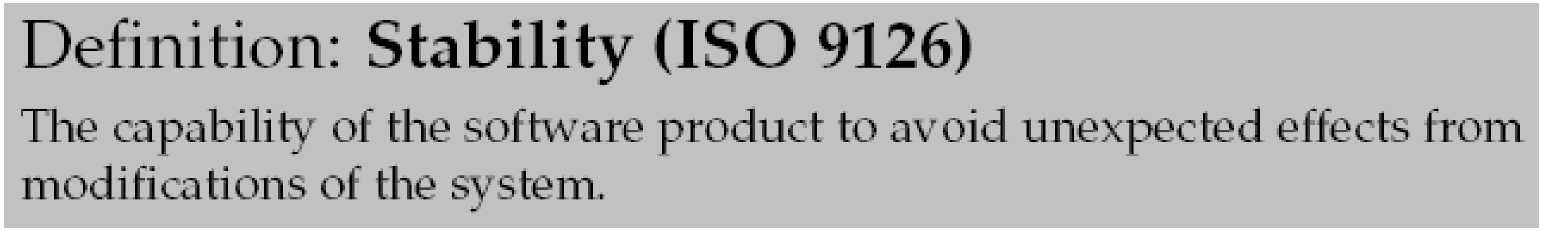
Definition: Changeability (ISO 9126)

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

Examples

- – The size of the Maze program is only present in terms of *magic numbers*
- – Consider using constants like MAZE_WIDTH instead
- – Delegation based techniques to encapsulate variability

Stability



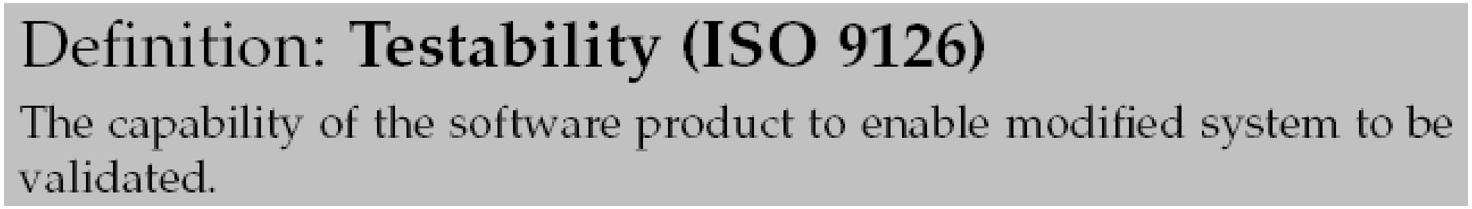
Definition: Stability (ISO 9126)

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

War story

- Person ID in ABC
  - ID = RFID tag; used by inference engine to detect ”activities”
  - Later a student programmer changed it CPR
    - Nothing worked during a demo for the physicians ☹

Testability



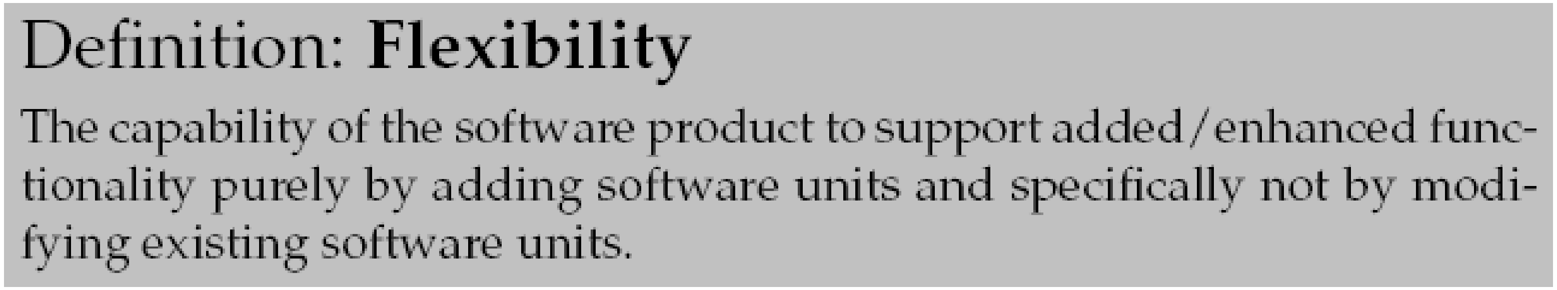
Definition: **Testability (ISO 9126)**

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

Examples

- The GammaTown pay station rate strategy
- ... And lots of others

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.

Note: Not an ISO definition, but my own ☺

Examples

- Almost everything in my book ...

Summary

Architectural Qualities: Maintainability is just *one* of several qualities of a software system

- Qualities often are in conflict
- Software architect must find the proper balance between them
- Maintainability (= techniques from this course) are not necessarily a primary driver!

Maintainability

- Has sub qualities
  - Stability, changeability, testability, analyzability
- Flexiblity: *change by addition, not by modification...*

Summary

There is no good or bad software!

- There is software that measure differently on the different qualities
  - Fast but difficult to change
  - Maintainable but performs less
  - Etc.

OK, OK: Software that is not maintainable, performs lousy, is in-secure, unreliable… *is* bad software…