# Introduction to Software Engineering

## 11. Software Quality

Mircea F. Lungu

Based on materials by Oscar Nierstrasz.


add description
All
S
W
Name ↓
Last Success
Last Failure
Last Duration
ese-2013-team1
6 hr 59 min - #20
13 hr - #15
2 min 57 sec
ese-2013-team2
17 hr - #24
22 hr - #22
2 min 4 sec
ese-2013-team3
53 min - #51
9 days 16 hr - #7
4 min 5 sec
ese-2013-team4
18 hr - #26
1 day 18 hr - #24
2 min 4 sec
ese-2013-team5
4 hr 45 min - #30
21 hr - #28
5 min 1 sec
ese-2013-team6
1 day 0 hr - #20
9 days 23 hr - #9
4 min 55 sec
ese-2013-team7
34 min - #31
2 days 0 hr - #20
3 min 46 sec
ese-2013-team8
13 hr - #25
4 days 13 hr - #21
2 min 17 sec
ese-2013-team9
4 hr 16 min - #21
6 days 21 hr - #11
2 min 19 sec
PomodoroBox
6 hr 27 min - #46
3 days 6 hr - #41
2 min 18 sec
TestJob
9 days 0 hr - #4
N/A
35 ms
Icon: S M L
Legend
RSS for all
RSS for failures
RSS for just latest builds

## What you will know…

- Can a correctly functioning piece of software still have poor quality?
- What's the difference between an external and an internal quality attribute?
- And between a product and a process attribute?
- Why should quality management be separate from project management?
- What are detection strategies

## Software Quality

## Which one would you choose?


What’s the difference between the two?
The quality of materials.
The working conditions.
The service.
The quality process.
The precision. The mean time to failure.
Thinking about this… try to define Software Quality.

## Software Quality is conformance to…

explicitly stated functional and performance requirements

explicitly documented development standards

implicit characteristics that are expected of all professionally developed software.



Quality assurance and quality control are often used in the manufacturing industry.

Pressman ch 17; Sommerville ch 30

I would add “implicit characteristics that are expected of that class of software”

## Did we already talk about software quality?

| | | |
|---|---|---|
| 1 | 18/Sep/13 | Introduction |
| 2 | 25/Sep/13 | Requirements Collection |
| 3 | 2/Oct/13 | Guest Lecture: Processes & Scrum |
| 4 | 9/Oct/13 | Usability and UI |
| 5 | 16/Oct/13 | Architecture |
| 6 | 23/Oct/13 | Good Software Design |
| 7 | 30/Oct/13 | Verification |
| 8 | 6/Nov/13 | UML |
| 9 | 13/Nov/13 | Software Quality |

…



We have been talking about quality already.
- Usability is quality. **of the Product.**
- GOOD is quality. **of the Software.**

The only thing that remains is **the process.**

## Steve McConnell, *Code Complete*

External Quality vs. Internal Quality



A definition in Steve McConnell's Code Complete divides quality into two pieces: internal and external quality characteristics. External quality characteristics are those parts of a product that face its users, where internal quality characteristics are those that do not. Surely, every thing faces some users. So we could consider the internal ones o be external to the developers.

# Quality is a struggle between…

- **customer** quality requirements (efficiency, reliability, etc.)
- **developer** quality requirements (maintainability, reusability, etc.)
- **organisation** quality requirements (standard conformance, portfolio management)

## Can we define quality formally?

> Some quality requirements are *hard to specify* in an unambiguous way
  - —directly measurable qualities (e.g., errors/KLOC),
  - —indirectly measurable qualities (e.g., usability).



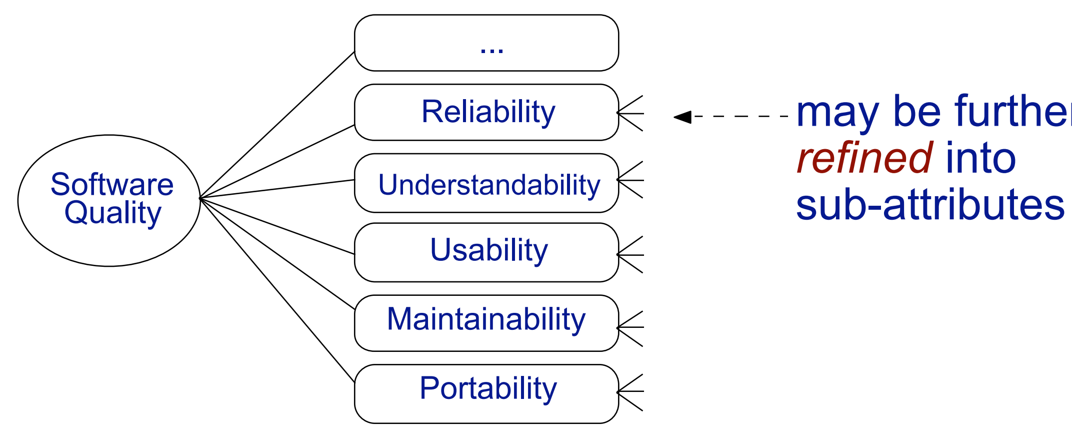
still vague… can we define it in a more concrete manner?

# Software Quality

Introduction
➡ Hierarchical Quality Model
Process Quality
Code Quality
  Cohesion and Coupling*
  Testing (unit, functional, integration)*
Software Metrics
  General Quality Evaluation
  Detection Strategies

## Hierarchical Quality Model

**Quality attributes / factors**






Define quality via hierarchical quality model, i.e. a number of *quality attributes* (a.k.a. quality factors, quality aspects, ...)

*Choose quality attributes (and weights) depending on the project context*

## Quality Attributes: External vs. Internal

> **External.** Derived from the relationship between the environment and the system (or the process). (To derive, the system or process must run)
> — e.g. Reliability, Robustness

> **Internal.** Derived immediately from the product or process description (To derive, it is sufficient to have the description)
> — Underlying assumption: internal quality leads to external quality (cfr. metaphor manufacturing lines)
> — e.g. Efficiency

# Correctness, Reliability, Robustness

***1. Correctness***

> A system is <u>correct</u> if it *behaves according to its specification*
  - —An *absolute property* (i.e., a system cannot be “almost correct”)
  - —... in theory and practice *undecidable*

***2. Reliability***

> The user may rely on the system behaving properly

> <u>Reliability</u> is the *probability* that the system will operate as expected over a specified interval
  - —A *relative property* (a system has a mean time between failure of 3 weeks)

***3. Robustness***

> A system is <u>robust</u> if it behaves reasonably *even in circumstances that were not specified*

> A *vague property* (once you specify the abnormal circumstances they become part of the requirements)

# Efficiency, Usability

***4. Efficiency*** (Performance)

> *Use of resources* such as computing time, memory
>
> —Affects user-friendliness and scalability
>
> —Hardware technology changes fast!
>
> —*First do it, then do it right, then do it fast*

> For process, resources are manpower, time and money
>
> —relates to the “productivity” of a process

# Efficiency, Usability ...

***5. Usability*** (User Friendliness, Human Factors)

> The *degree* to which the human users find the system (process) *both "easy to use" and useful*

- —Depends a lot on the target audience (novices vs. experts)
- —Often a system has various kinds of users (end-users, operators, installers)
- —Typically expressed in "amount of time to learn the system"

# Maintainability

> *External product attributes* (evolvability also applies to process)

***6. Maintainability***

> How easy it is to *change* a system after its initial release
>
> —software entropy $\Rightarrow$ maintainability gradually decreases over time

# Maintainability is often refined to…

***Evolvability*** (Adaptability)

> How much work is needed to *adapt* to changing requirements (both system and process)

***Portability***

> How much work is needed to *port* to new environment or platforms

***Understandability***

> How easy it is to *understand* the system
—internally: contributes to maintainability
—externally: contributes to usability

# Software Quality

# Process Quality

**Underlying assumption: a quality process leads to a quality product**

# How to evaluate your process?

**The Joel Test**

1. Do you use source control?
2. Can you make a build in one step?
3. Do you make daily builds?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
6. Do you have an up-to-date schedule?
7. Do you have a spec?
8. Do programmers have quiet working conditions?
9. Do you use the best tools money can buy?
10. Do you have testers?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?



http://www.joelonsoftware.com/articles/fog0000000043.html

# ISO

***ISO = International Organisation for Standardization***

> ISO main site: http://www.iso.ch/

# ISO 9000, 9001

ISO 9000 is an international set of standards for *quality management* applicable to a range of organizations from manufacturing to service industries.

ISO 9001 is a *generic model of the quality process*

> Applicable to organizations whose business processes range from design and development, to production, installation and servicing;

> ISO 9001 must be *instantiated for each organisation*



http://en.wikipedia.org/wiki/ISO_9000 -- ISO 9001 page has a good **Summary of ISO 9001:2008 in informal language**

**http://www.praxiom.com/iso-90003.htm**

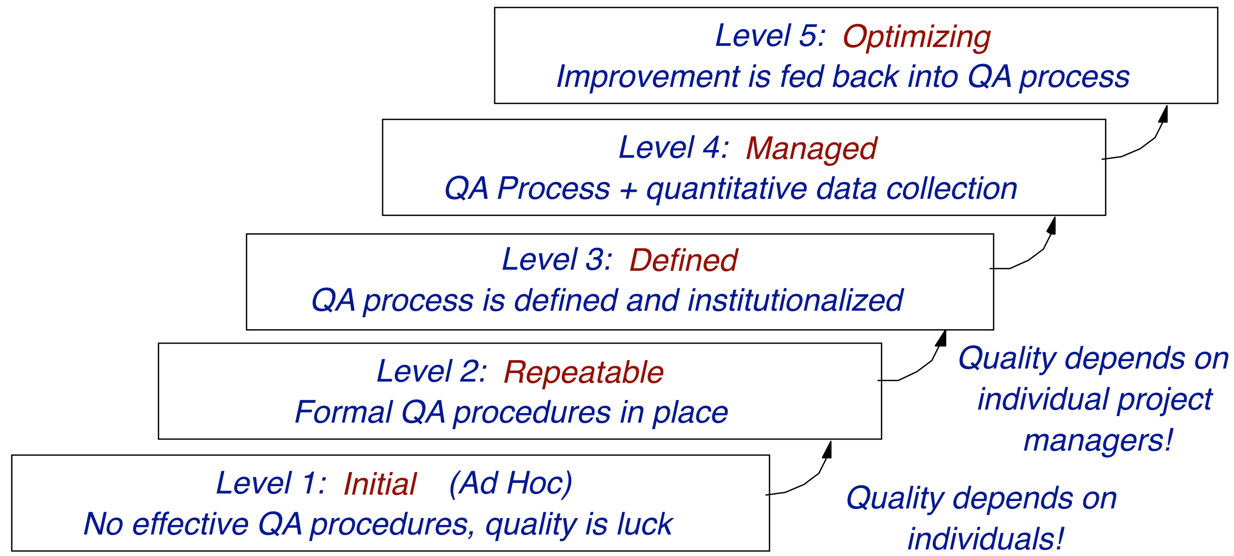
**Think about yourself. If somebody comes and asks you to estimate the effort for a new project it will be hard. The more you document the process, the higher the chance you'll be able to give a correct estimate upfront. ...**

# ISO 90003 (few of the points)

> **The quality policy is a formal statement** from management
> The business makes decisions about the quality system based on **recorded data.**
> The quality system is **regularly audited and evaluated** for conformance and effectiveness.
> The business has **created systems for communicating with customers** about product information, inquiries, contracts, orders, feedback, and complaints.
> The business **regularly reviews performance** through internal audits and meetings. The business determines whether the quality system is working and what improvements can be made. It has a documented procedure for internal audits.
> The business **deals with past problems and potential problems**. It keeps records of these activities and the resulting decisions, and monitors their effectiveness.
> The business has **documented procedures for dealing with actual and potential nonconformances** (problems involving suppliers, customers, or internal problems).

# Capability Maturity Model (CMM)

*The SEI process maturity model classifies how well contractors manage software processes*



# The Quality Plan

A quality plan should:

> set out desired product qualities and how these are assessed
  —define the most significant quality attributes

> define the quality assessment process
  —i.e., the controls used to ensure quality

> set out which organisational standards should be applied
  —may define new standards, i.e., if new tools or methods are used

*NB: Quality Management should be separate from project management to ensure independence*

## Software Quality Controls

1. Reviews
   - *Inspections* for defect removal (product)
   - *Progress assessment reviews* (product and process)
   - ***Quality reviews*** (product and standards)

2. Automated Software Assessment
   - *Measure* software attributes and compare to standards (e.g., defect rate, cohesion, etc.)

## Types of Quality Reviews

A quality review is carried out by a group of people who carefully examine part or all of a software system and its associated documentation.

> Reviews should be *recorded and records maintained*
- —Software or documents may be *"signed off"* at a review
- —Progress to the next development stage is thereby *approved*



Review meetings should:
typically involve 3-5 people
require a maximum of 2 hours advance preparation
last less than 2 hours

# Review Minutes

The review report should *summarize*:

1. *What* was reviewed
2. *Who* reviewed it?
3. *What* were the findings and conclusions?

The review should *conclude* whether the product is:

1. *Accepted* without modification
2. *Provisionally accepted*, subject to corrections (no follow-up review)
3. *Rejected*, subject to corrections and follow-up review

## Sample Review Checklists (I)

***Software Project Planning***

1. Is software scope unambiguously defined and bounded?
2. Are resources adequate for scope?
3. Have risks in all important categories been defined?
4. Are tasks properly defined and sequenced?
5. Is the basis for cost estimation reasonable?
6. Have historical productivity and quality data been used?
7. Is the schedule consistent?

...



Pressman pp 570-574

# Sample Review Checklists (II)

***Requirements Analysis***

1. Is information domain analysis complete, consistent and accurate?
2. Does the data model properly reflect data objects, attributes and relationships?
3. Are all requirements traceable to system level?
4. Has prototyping been conducted for the user/customer?
5. Are requirements consistent with schedule, resources and budget?

...

# Sample Review Checklists (III)

***Design***

1. Has modularity been achieved?
2. Are interfaces defined for modules and external system elements?
3. Are the data structures consistent with the information domain?
4. Are the data structures consistent with the requirements?
5. Has maintainability been considered?

...

# Sample Review Checklists (IV)

***Code***

1. Does the code reflect the design documentation?
2. Has proper use of language conventions been made?
3. Have coding standards been observed?
4. Are there incorrect or ambiguous comments?

...

## Sample Review Checklists (V)

***Testing***

1. Have test resources and tools been identified and acquired?
2. Have both white and black box tests been specified?
3. Have all the independent logic paths been tested?
4. Have test cases been identified and listed with expected results?
5. Are timing and performance to be tested?

## Review Results

Comments made during the review should be *classified*.

> ***No action.***
>
> —No change to the software or documentation is required.

> ***Refer for repair.***
>
> —Designer or programmer should correct an identified fault.

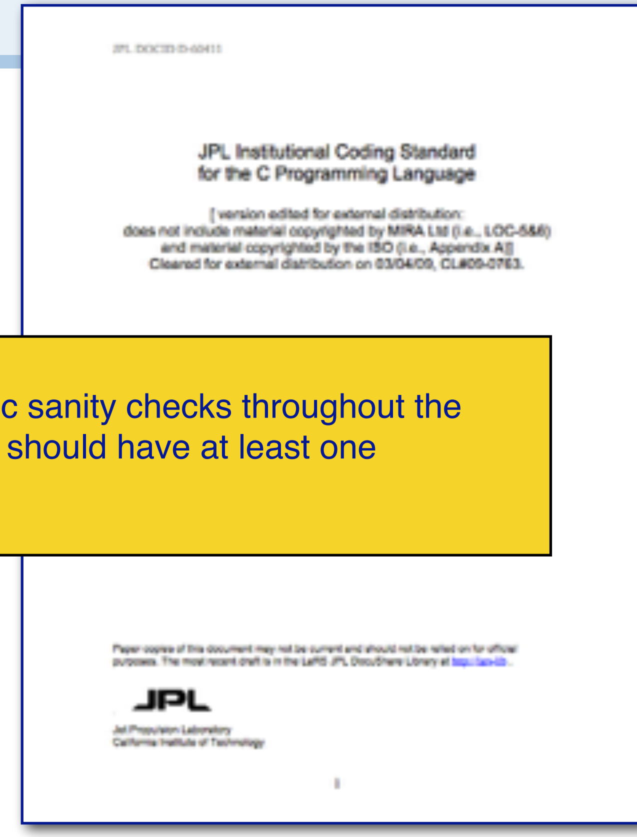
> ***Reconsider overall design.***
>
> —The problem identified in the review impacts other parts of the design.

*Requirements and specification errors may have to be referred to the client.*



Sommerville p 618

# Sample JPL Coding Convention


JPL Institutional Coding Standard
for the C Programming Language

[ version edited for external distribution:
does not include material copyrighted by MIRA Ltd (i.e., LOC-5&6)
and material copyrighted by the ISO (i.e., Appendix A)]
Cleared for external distribution on 03/04/09, CL#09-0763.

JPL

Jet Propulsion Laboratory
California Institute of Technology

**Rule 16 (use of assertions)**
Assertions shall be used to perform basic sanity checks throughout the code. All functions of more than 10 lines should have at least one assertion. [Power of Ten Rule 5]

http://lars-lab.jpl.nasa.gov/JPL_Coding_Standard_C.pdf

# Sample Java Code Conventions

### *10.3 Constants*

Numerical constants (literals) should not be coded directly, except for -1, 0, and 1, which can appear in a for loop as counter values.

Source: http://java.sun.com/docs/codeconv/CodeConventions.pdf

## Software Quality

- Introduction
- Hierarchical Quality Model
- Process Quality
- Code Quality
  - Cohesion and Coupling*
  - Testing (unit, functional, integration)*
- Software Metrics
  - General Quality Evaluation
  - Detection Strategies

## Software Quality

- Introduction
- Hierarchical Quality Model
- Process Quality
- Code Quality
  - ➡ Cohesion and Coupling*
  - Testing (unit, functional, integration)*
- Software Metrics
  - General Quality Evaluation
  - Detection Strategies

## Cohesion

***How well the parts of a component belong together***

> Cohesion is **weak** if elements are bundled simply because they perform similar or related functions (e.g., `java.lang.Math`).

> Cohesion is **strong** if all parts are needed for the functioning of other parts (e.g. `java.lang.String`).

—Strong cohesion *promotes maintainability* and adaptability by *limiting the scope of changes* to small numbers of components.



What do you talk about? Classes here. Modules / Components.
Eclipse plugins. If you have a set of plugins, this allows you to deploy only partially.
Inadequacy of formal definitions: it is in the eye of the beholder.

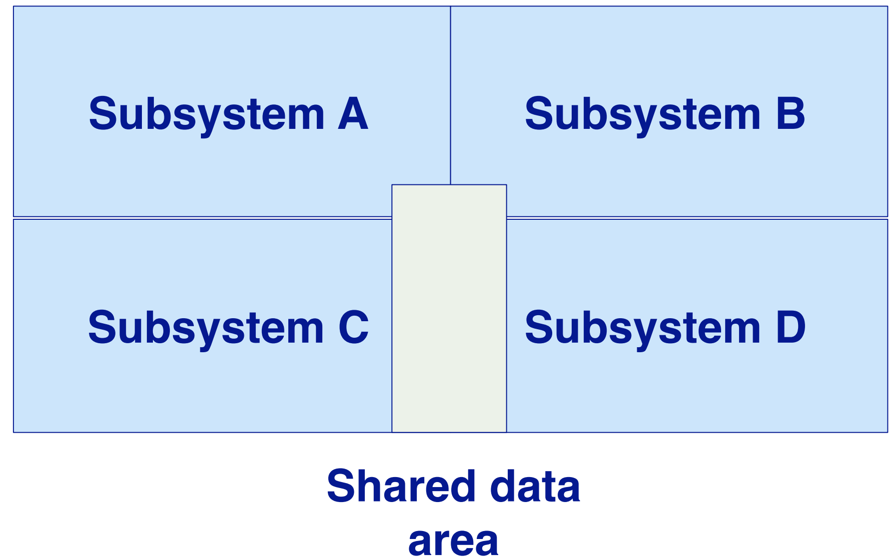
To think about. How would you measure cohesion?

# Coupling

**The *strength of the interconnections* between components**

- Coupling is **tight** between components if they depend heavily on one another, (e.g., there is a lot of communication between them).
- Coupling is **loose** if there are few dependencies between components.
  - —Loose coupling *promotes maintainability* and adaptability since *changes in one component are less likely to affect others.*
  - —*Loose coupling increases the chancesof reusability.*

## Tight Coupling

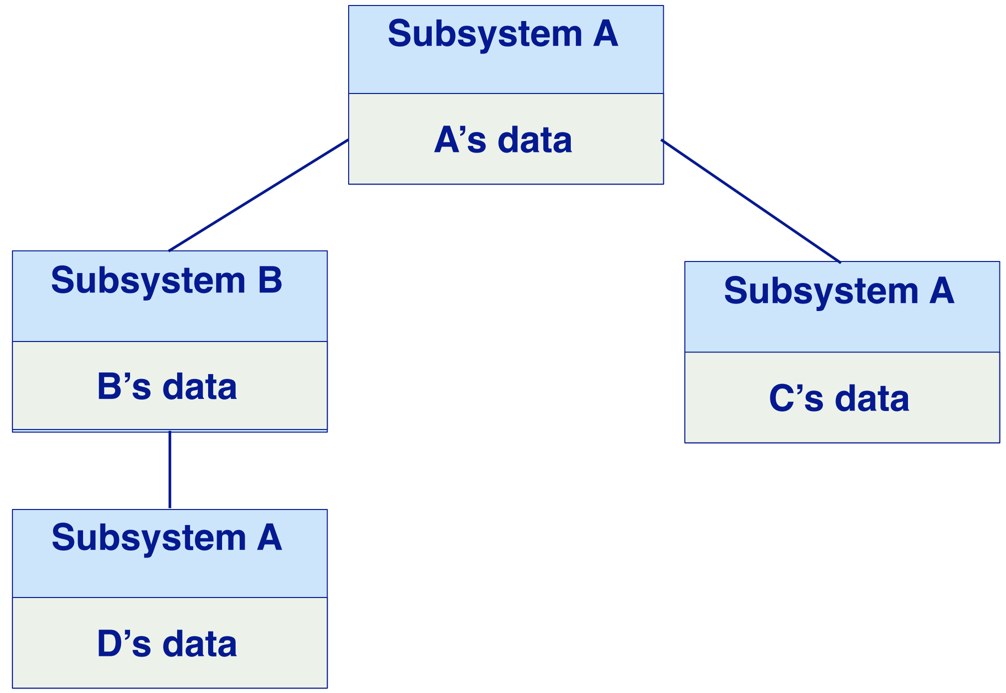







Classical structured programming. And classical using global variables.

# Loose Coupling








OO good idea: keep behavior close to the data.

## Software Quality

## The Testing Process

1. Unit testing:
   — Individual (stand-alone) *components* are tested to ensure that they operate correctly.
2. Module testing:
   — A collection of *related components* (a module) is tested as a group.
3. Sub-system testing:
   — The phase tests a *set of modules* integrated as a sub-system. Since the most common problems in large systems arise from sub-system interface mismatches, this phase focuses on testing these interfaces.

## The Testing Process ...

4. System testing:
   - This phase concentrates on (i) detecting errors resulting from unexpected interactions between sub-systems, and (ii) validating that the complete systems fulfils functional and non-functional requirements.
5. Acceptance testing (alpha/beta testing):
   - The system is tested with *real* rather than simulated data.

## Bottom-up Testing

> *Start by testing units* and modules
> *Test drivers* must be written to exercise lower-level components
> Works well for *reusable components* to be shared with other projects

Bottom-up testing will not uncover *architectural faults*

## Top-down Testing

> *Start with sub-systems*, where modules are represented by "stubs" / mocks
> Similarly test modules, representing functions as stubs
> *Coding and testing* are carried out as a *single activity*
> Design errors can be detected early on, avoiding expensive redesign
> Always have a running (if limited) system!

*BUT:* may be impractical for stubs to simulate complex components

## Software Quality

- Introduction
- Hierarchical Quality Model
- Process Quality
- Code Quality
  - Cohesion and Coupling*
  - Testing (unit, functional, integration)*
- Software Metrics
  - General Quality Evaluation
  - Detection Strategies

## Software Metrics

**Any type of measurement which relates to a software system, process or related documentation**

—Lines of code in a program

—the Fog index (calculates readability of a piece of documentation)

$$0.4 \times (\#\text{ words} / \#\text{ sentences}) + (\%\text{ words} \geq 3\text{ syllables})$$

—number of person-days required to implement a use-case

# Direct and Indirect Measures

***Direct Measures***

> *Measured* directly in terms of the observed attribute (usually by counting)
  - —Length of source-code, Duration of process, Number of defects discovered

***Indirect Measures***

> *Calculated* from other direct and indirect measures
  - —Module Defect Density = Number of defects discovered / Length of source
  - —Temperature (usually derived from the length of a liquid column)

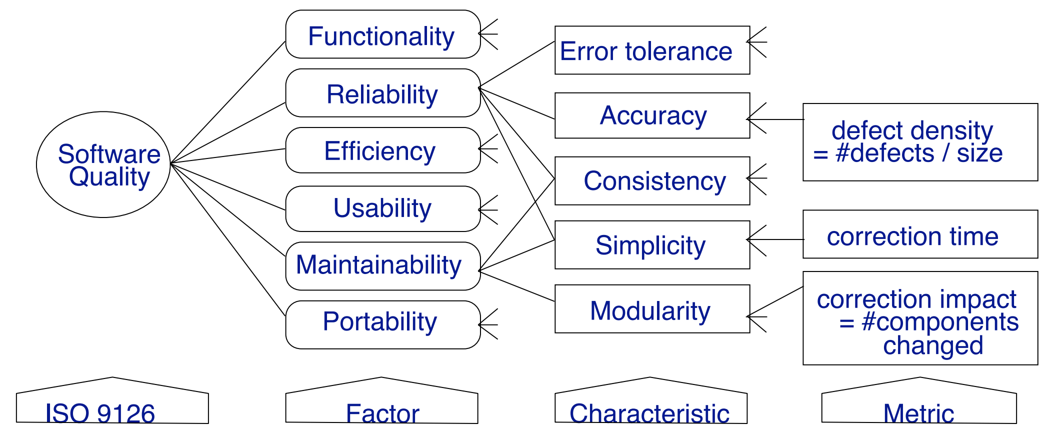
# Software Quality

- Introduction
- Hierarchical Quality Model
- Process Quality
- Code Quality
  - Cohesion and Coupling*
  - Testing (unit, functional, integration)*
- Software Metrics
  - General Quality Evaluation
  - Detection Strategies

# Quantitative Quality Model

***Quality according to ISO 9126 standard***

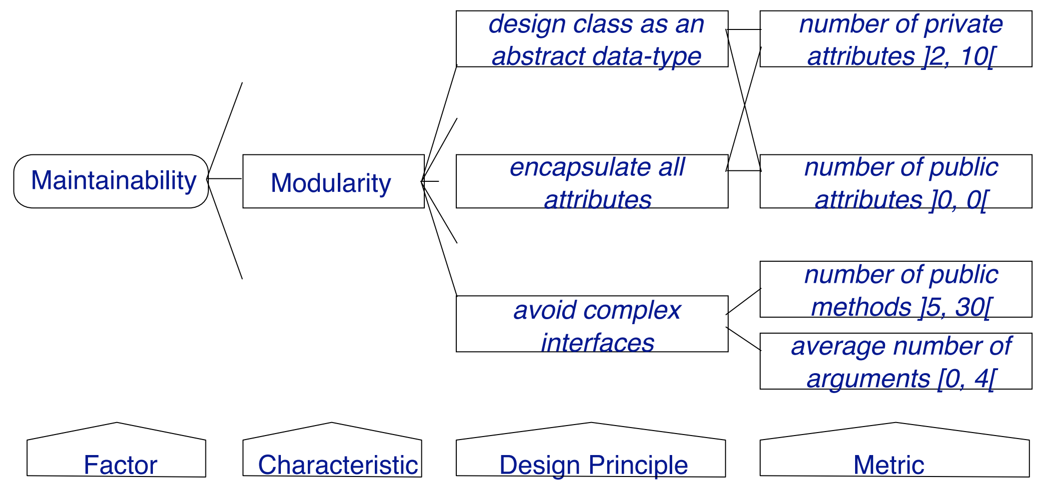
- Divide-and conquer approach via “hierarchical quality model”
- Leaves are simple metrics, measuring basic attributes



# “Define your own” Quality Model

*Define the quality model with the development team*

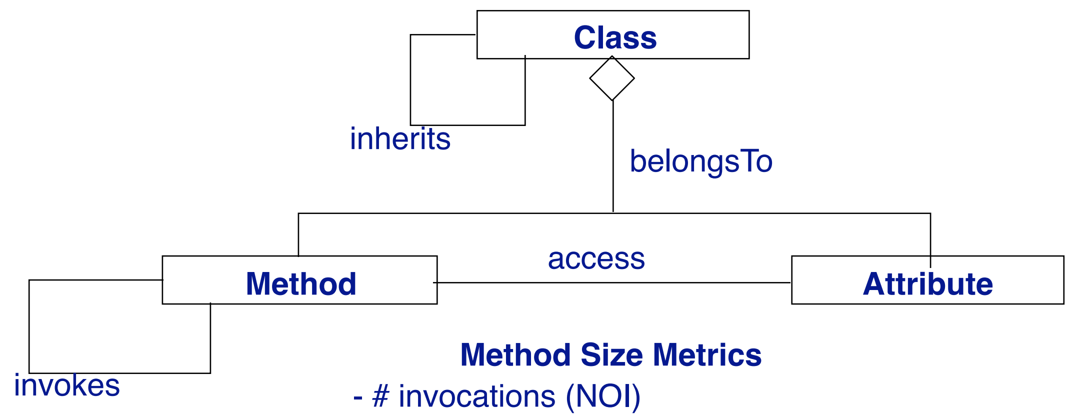
> Team chooses the characteristics, design principles, metrics ... and the thresholds



# Sample Size (and Inheritance) Metrics

**Inheritance Metrics**
- hierarchy nesting level (HNL)
- # immediate children (NOC)
- # inherited methods, unmodified (NMI)
- #overridden methods (NMO)

**Class Size Metrics**
- # methods (NOM)
- # attributes, instance/class (NIA, NCA)
- # Σ of method size (WMC)




**Method Size Metrics**
- # invocations (NOI)
- # statements (NOS)
- # lines of code (LOC)
- # arguments (NOA)

## Sample Coupling & Cohesion Metrics

The following definitions stem from [Chid91a], later republished as [Chid94a]

***Coupling Between Objects (CBO)***

CBO = number of other classes to which given class is coupled

Interpret as “number of other classes a class requires to compile”

***Lack of Cohesion in Methods (LCOM)***

LCOM = number of disjoint sets (= not accessing same attribute) of local methods



Researchers disagree whether coupling/cohesion methods are valid

Classes that are observed to be cohesive may have a high LCOM value due to accessor methods
Classes that are not much coupled may have high CBO value
no distinction between data, method or inheritance coupling

## Sample Quality Metrics (I)

***Productivity (Process Metric)***

> functionality / time

> functionality in LOC or FP; time in hours, weeks, months

—be careful to compare: the same unit does not always represent the same

> Does not take into account the quality of the functionality!

> metrics can be cheated
> LOC is the worst way of measuring productivity

# Sample Quality Metrics (II)

***Reliability (Product Metric)***

> mean time to failure =
> mean of probability density function PDF

- —for software one must take into account the fact that repairs will influence the rest of the function ⇒ quite complicated formulas

> average time between failures = # failures / time

- —time in execution time or calendar time
- —necessary to calibrate the probability density function

> mean time between failure = MTTF + mean time to repair

- —to know when your system will be available, take into account repair

## Software Quality

# Detection strategy

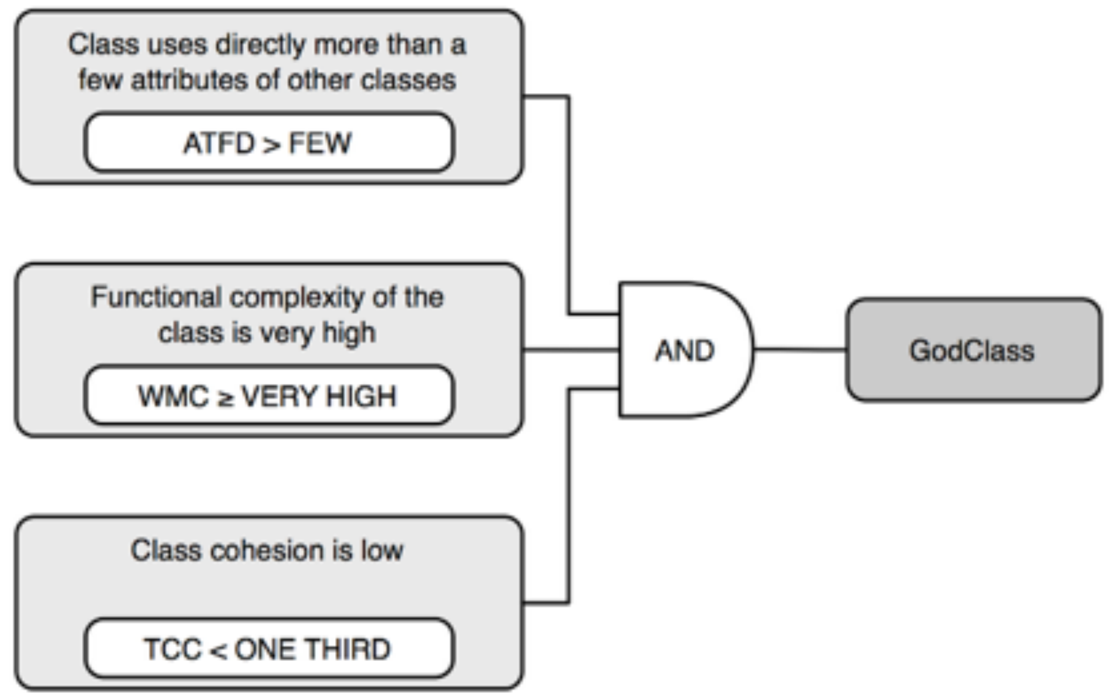
> A detection strategy is a *metrics-based predicate* to identify *candidate* software artifacts that *conform to* (or violate) a particular *design rule*

## Filters and composition

> A data filter is a predicate used to focus attention on a *subset of interest* of a larger data set
>
> — Statistical filters
>   - *I.e., top and bottom 25% are considered outliers*
>
> — Other relative thresholds
>   - *I.e., other percentages to identify outliers (e.g., top 10%)*
>
> — Absolute thresholds
>   - *I.e., fixed criteria, independent of the data set*

> A useful detection strategy can often be expressed as a *composition* of data filters

## God Class - centralizes intelligence



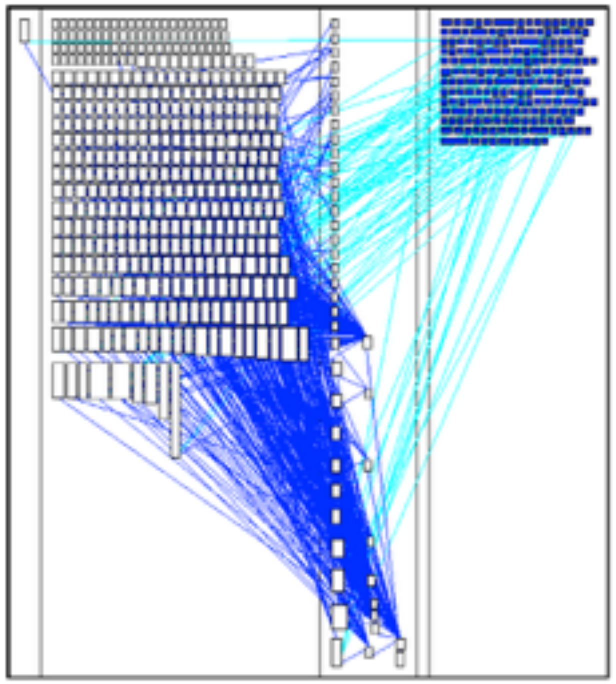



A God Class centralizes intelligence in the system:
- Impacts understandibility
- Increases system fragility

I want that you know the concepts, not the metrics.

# ModelFacade (ArgoUML)

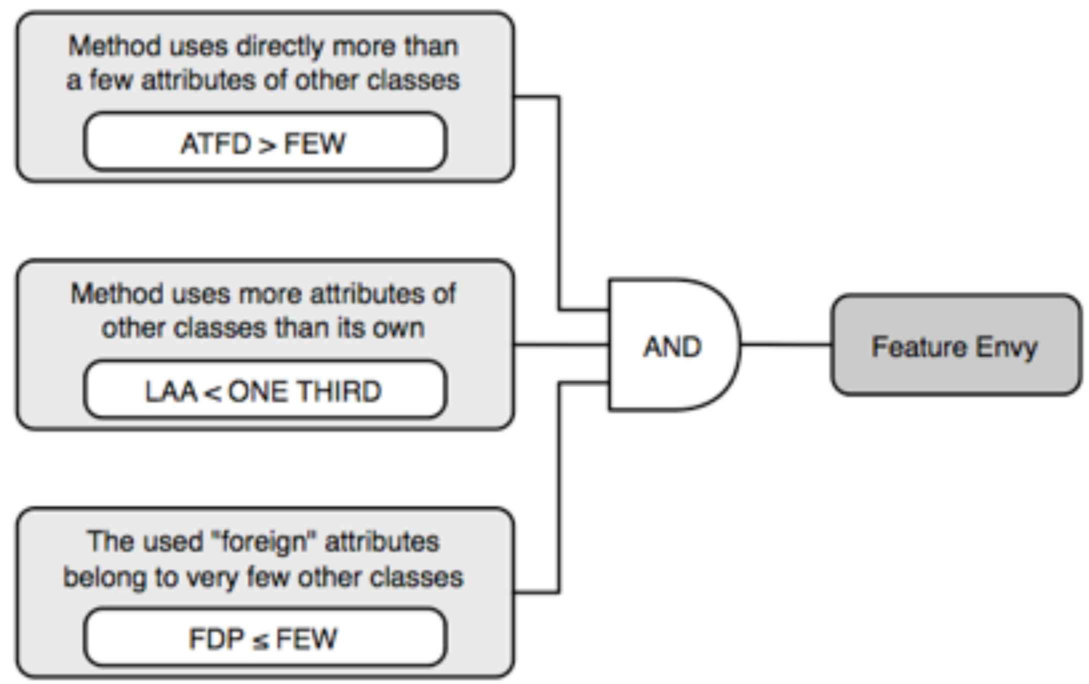
- 453 methods
- 114 attributes
- over 3500 LOC
- all methods and all attributes are static





Strictly speaking, this is a Facade rather than a God class, but it has become a “black hole” of functionality in the system.

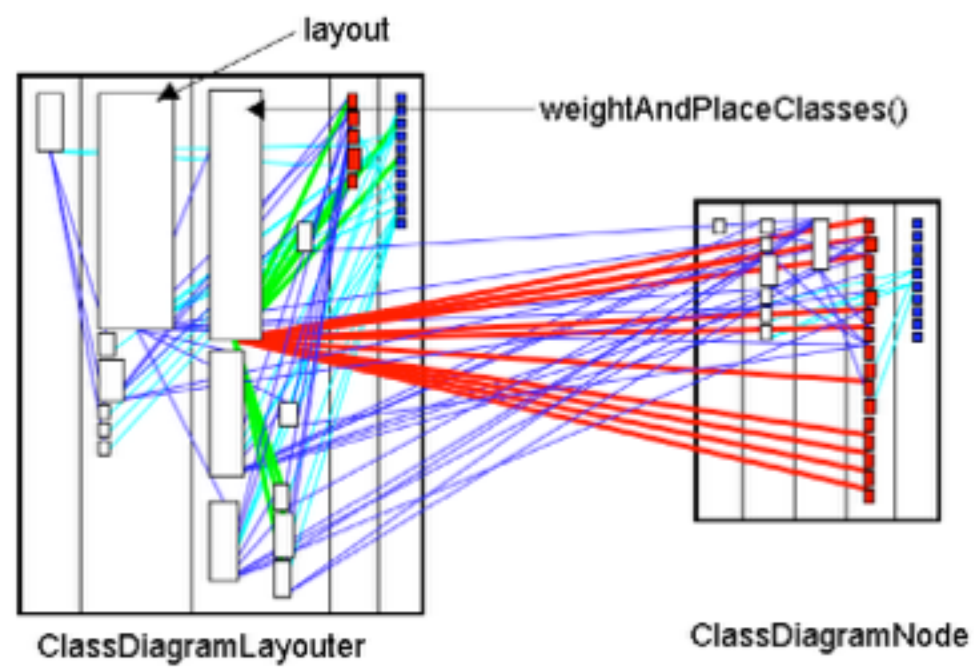
## Feature Envy






Methods that are more interested in data of other classes than their own [Fowler et al. 99]

# ClassDiagramLayouter

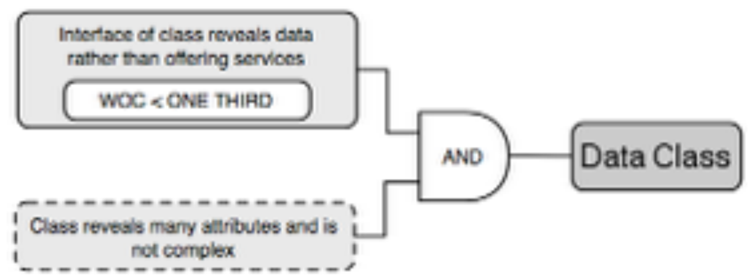




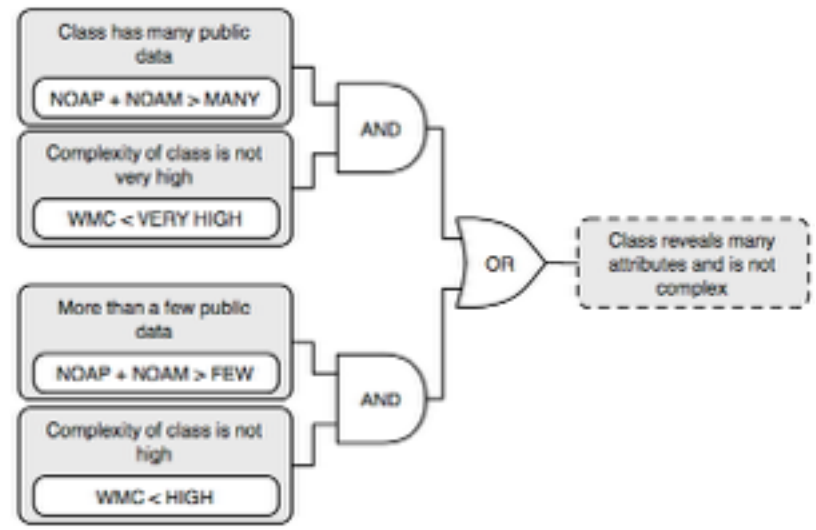

The weightAndPlaceClasses and layout methods are very large, and use many accessors and attributes of ClassDiagramNode. The latter has little behavior of its own. Likely fragments of code in the client methods can be extracted and moved to the data class.

# Data Class

> A Data Class provides data to other classes but little or no functionality of its own

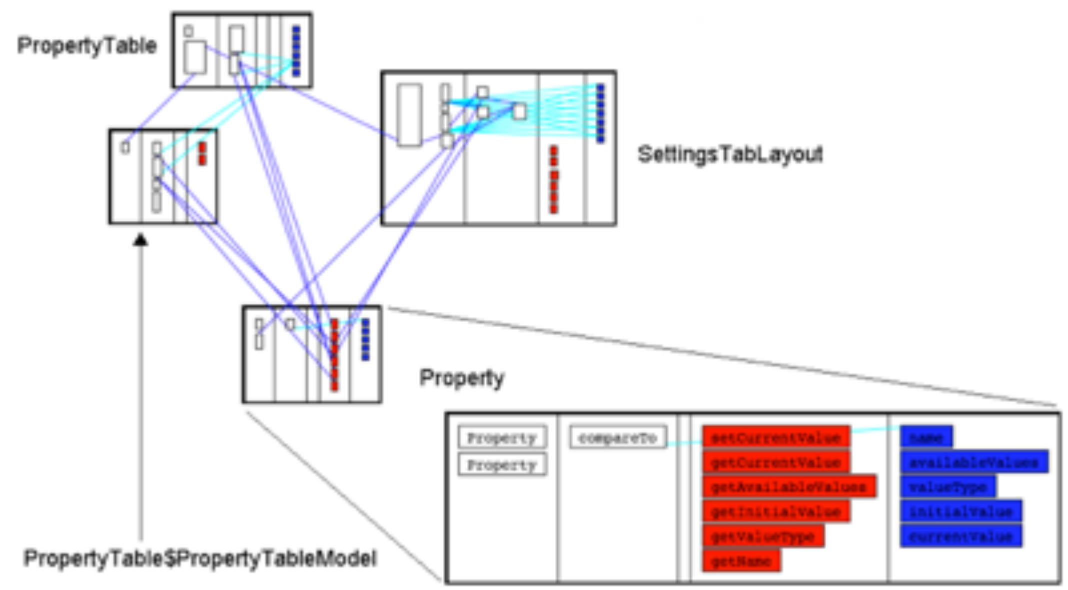


# Data Class (2)



Class has many public data
NOAP + NOAM > MANY
Complexity of class is not very high
WMC < VERY HIGH
More than a few public data
NOAP + NOAM > FEW
Complexity of class is not high
WMC < HIGH
AND
AND
OR
Class reveals many attributes and is not complex

# Property






Property is a classical data class with almost no behavior of its own. Sometimes data classes can be merged with their client class, and sometimes client methods or parts of client methods can be moved to the responsibility of the data class.

# Next time: Guest Lecture

Tudor Girba, Software Assessment

## Can you answer the following questions?

- > When and why does a project need a quality plan?
- > Why are coding standards important?
- > What would you include in a documentation review checklist?
- > How often should reviews be scheduled?
- > Would you trust software developed by an ISO 9000 certified company?
- > And if it were CMM level 5?

## Sources


> *Software Engineering*, I. Sommerville, 7th Edn., 2004.

> *Software Engineering — A Practitioner's Approach*, R. Pressman, Mc-Graw Hill, 5th Edn., 2001.

> *Fundamentals of Software Engineering*, C. Ghezzi, M. Jazayeri, D. Mandroli, Prentice-Hall 1991