IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Development environment & Introduction to software quality

J.-C. Bach

DCL – DevEnv – 2025-2026

Work under Creative Commons BY-SA license

Important notes

- If you do not understand something, please ask your questions. *We cannot answer the questions you do not ask...*
- If you disagree with us, please say it (politely)
- People don't learn computer science by only reading few academic slides: practicing is fundamental

Content of this lecture

An introduction to...

- ...integrated development environments (IDEs)
- ...(software) quality
- ...software metrology

Two practical sessions:

- debug (in Java, with Eclipse)
- static analysis, quality (with PMD Eclipse plugin)

⇒ we hope you will use what you will learn here in others UEs, modules, contexts (e.g. in MAPD)

Motivations

- Software (development process) complexity
- Heterogeneity: languages, libraries, tooling, technologies
- Short "sprints", frequent deliveries (agile methods)
- Difficulty to grasp all elements of a software systems
- Need of {quality,safety,security}

$\Rightarrow$ Need of tools to assist developers in the process development

Progress

1 Integrated Development Environment – IDE

2 Introduction to (software) quality

3 Introduction to software metrology

Integrated Development Environment (IDE)

- Integrated toolset to improve productivity and *quality* of software developments
- Language-specific or not
- IDE = helper
  - to write and to edit code (syntax highlightening, completion, refactoring, etc.)
  - to compile sources
  - to execute
  - to find errors (debugger, analysers, ...)
  - to create tests (unit tests frameworks)
  - to import/export various documents in various formats

Examples of IDE

- Eclipse: `https://www.eclipse.org`
- IntelliJ IDEA: `https://www.jetbrains.com/idea/`
- NetBean: `https://netbeans.org/`
- PyCharm: `https://www.jetbrains.com/pycharm/`
- Qt creator: `https://www.qt.io/`
- Visual Studio Code: `https://code.visualstudio.com/`
- Xcode: `https://developer.apple.com/xcode`
- ...

`https://en.wikipedia.org/wiki/Integrated_development_environment`

Tools often integrated in an IDE

- Version control management (Git, Subversion, etc.)
- Build systems and package managers (Gradle, Ant, Maven)
- Documentation management (Javadoc, ...)
- Generation of graphical interfaces
- Integration of UML tools (editors, code generators, reverse engineering, . . .)
- Application lifecycle managers and tasks managers (Mylyn)
- Project management, planning

Eclipse

- Usually Java-oriented...
- ...but also usable for many other languages like C++ (CDT), Python (pydev), ...
- JDT (= Java Development Toolkit) for Java compilation and analysis
- A lot of plugins
- Several specialized bundles : RCP, OSGi, JEE, ...

$\Rightarrow$ The IDE we know we can use during the lab sessions

Why Eclipse? Why not vscode? Where is my freedom?! Students on strike!

- A very mature product
  - widely used (it'changing)
  - well tooled
  - well documented
- Does what we need in our pedagogical context
- ...and we had to make a technical choice (it would probably be different if we had to choose today)
- Is it the best IDE in the world?
  - nope, the best IDE is the one you master and that is well-suited for your task
- I do not want to use Eclipse because I think that XYZ vscode is better
  - use the tools you prefer, as long you (learn to) use an IDE
  - some lab sessions might have constrained technical environment

How does Eclipse look? (*demo*)

- Perspective
  - set of tools for some needs (Java, PMD, version control, etc.)
  - usually a perspective associated to a plugin (PMD, Findbugs, ...)
- View
  - visual framework showing tools of the current perspective
  - example: error view in PMD, coverage view with EclEmma, . . .
- Finding views and perspectives: *Window* tab
- Another important tab: *Help* for the documentation

Eclipse extensions

- Bundles contain sets of extensions adapted to different tasks
- Possible to extend the current bundle
  - through the *market place* (probably the easiest way, *Help* tab)
  - through the *install new software* menu, the tool URL is needed
  - by hand (extensions are `.jar` archives)
  - through the source repository (Git, Subversion, ...), then build of the `.jar` archive
- Other IDEs usually also have plugins/extensions

Progress

What is quality?

- No unique definition: a lot of definitions
- Compliance with customers (and end-users) requirements
- "Respecting some relevant criteria"
- Quality criteria: consistency with requirements, performance (time, space), liability, security, testability, maintainability, durability, ergonomy, energy consumption, ...

$\Rightarrow$ a lot of criteria, sometime contradictory to each other

- Assumption that compliance to customer requirements is met
- Focus on internal qualities of software (style, structure, norms, etc.)

Improving quality

- Improving the product
  - validating: *do the right thing*
  - verifying: *do the thing right*
- Mastering and improving the development process
  - product quality depends on the process quality
  - ... and so do safety and security
- Improving project management, contractors, suppliers, etc. (out of scope)

$\Rightarrow$ Improving trust in software and in software development

Norms and standards

- Software standards: ISO 9126, replaced by ISO 25010
- Standard for software delivery: ISO 9001
- Classification: Capability Maturity Model (CMM)
- Good practices: CMMI, ITIL, COBIT, . . .
- Normalization organisms: AFNOR, IEEE, ITU, IETF, ISO, OASIS, W3C, . . .

Quality assurance (QA)

- Implements the rules that aim to improve quality
- What do you do to improve your code quality?

Quality assurance (QA)

- Implements the rules that aim to improve quality
- What do you do to improve your code quality?
- Usual QA activities
  - Code review / peer review
  - Review with checklist
  - Code analysis
  - Tests
  - Metrology
  - Audit (aims the process)

Quality assurance (QA)

- Implements the rules that aim to improve quality
- What do you do to improve your code quality?
- Usual QA activities
  - Code review / peer review
  - Review with checklist
  - Code analysis
  - Tests
  - Metrology
  - Audit (aims the process)
- There is often a gap between the formal norm and the industrial reality

$\Rightarrow$ Need of tools and methods to support those activities

Quality approaches in practice

- Normative approach
  - Code review
  - Coding conventions and style
- Static analysis (code is observed without being executed)
  - Code source or bytecode analysis, partial evaluation
  - Graph control analysis
  - Data flow analysis
  - Structural analysis
  - Formal verification
  - Metrology
- Dynamic analysis
  - profiling
  - software testing

Normative approach

- Coding style, coding conventions
- Common approach in software development
- Examples
  - Google conventions: `https://google.github.io/styleguide/a`
  - Oracle Java conventions: `https://www.oracle.com/technetwork/java/codeconventions-150003.pdf`
  - PostgreSQL coding conventions: `https://www.postgresql.org/docs/9.6/source.html`
- Easy to implement if done from start of the project
- Can be tooled (ex: checkstyle[1])

[1] `http://eclipse-cs.sourceforge.net/`

Good practices, coding style

- General and specific rules
- Examples
  - using *lowerCamelCase* style for variables and functions names
  - using symbolic constants (no *magic numbers*)
  - using interfaces
  - commenting code, at least API
  - not use (or avoid to use) `instanceof`
  - using private or protected attributes by default instead of public
  - using exceptions
  - ...

Static analysis

- Type checking
- Control-flow graph analysis: analysis of the graph representation of all paths that might be traversed through a program during its execution
  - useful to detect dead code, infinite loop, ...
  - ex: Checkstyle, PMD, FindBugs, ...
- Data-flow analysis: analysis of the possible set of values calculated at various points in a computer program
  - useful to detect unitialized variable, null pointer, ...
  - ex: JLint (`http://artho.com/jlint/`)

Formal verification

- Proving properties of a system
  - Model-checking
  - Proof
- Work with an abstraction/specification
- High entry costs
- Usually difficult to implement at large scale
- Tools
  - model-checking: Alloy, TLA+, LTSA, CADP, UPPAAL, ...
  - proof assistants: Coq, Isabelle/HOL, PVS
  - solvers: Z3, veriT, SMT, ...
- Out of the scope of this lecture set

Dynamic analysis

- Profiling
  - space (memory) or time complexity of a program
  - usage of a particular instruction
  - frequency and duration of function calls
  - ...

  $\Rightarrow$ usually to help optimization
  - needs instrumentation of code source or of binary executable + a profiler
- Testing
  - unit tests, regression tests, functional tests, ...
  - check behavior of a piece of software
  - developers (should) write tests during the whole development process
  - needs additional code (tests + framework) ... which can be incorrect

Progress

Metrology

- Defining metrics for source code
- Do not seek to find defaults but to provide numerical indicators. . . which are then evaluated
- Also concerns specifications and (UML) models
- Tooling
- Measurement can be done *a priori*, but usually done *a posteriori* (using heuristics and experimental validations)

Measurement validity

- A set of measurements is only valid in a given context
- Granularity: application, package, class, function, ...
- Problems:
  - meaning of metrics
  - validity of measurements
  - code metrics applied to models or objects (redefinition needed)

⇒ must be linked to quality indicators

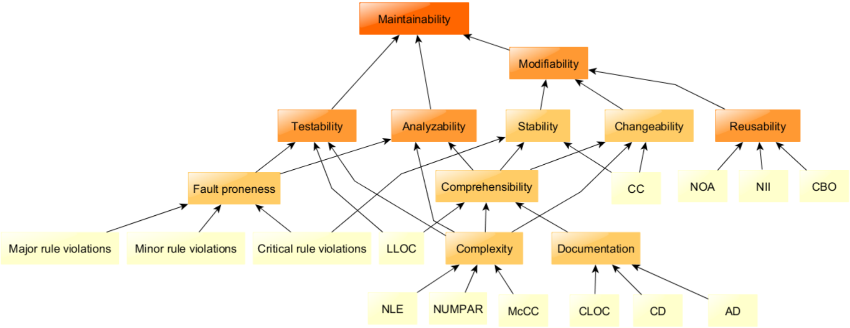
- Measurements are not exploitable alone: they require criteria

⚠ beware of traps and bias

Criteria to define software quality (ISO 25010)

- Functionality
- Maintainability
- Security
- Compatibility
- Portability
- Liability
- Performance
- Ergonomy

Quality indicators and measurements



- ISO 25010 criterion: maintainability
- Quality indicators for maintainability: modularity, reusability, analyzability, testability, modifiability
- Measurement of testability: logical lines of code (LLOC), cyclomatic complexity

Difficulty to establish a metric

- Absolute values?
- Ranges or thresholds, vague measurements
- Weighting depending on different levels and on different results of previous levels
- Example of *reusability*
  - ability of classes to be used
    - number of attributes, number of public methods, ...
  - ability of classes to be specialized
    - number of public attributes, number of inherited attributes, ...
  - ability of classes to be analysed
    - for one class: complexity, number of ancestors, coupling between objects, ...

Types of metrics

- What can be measured in code?

Types of metrics

- What can be measured in code?
- Simple metrics:
  - number of lines of code (LOC)
  - number of nested control structures (LEVL)

  $\Rightarrow$ relevant for code duplication and readability

Types of metrics

- What can be measured in code?
- Simple metrics:
  - number of lines of code (LOC)
  - number of nested control structures (LEVL)

  $\Rightarrow$ relevant for code duplication and readability
- More advanced metrics
  - coupling: relationships between two entities
  - cohesion: consistency of an entity ("there is a functional unity"), relationships within a module
  - call graph: control flow graph representing calling relationships between functions

  $\Rightarrow$ relevant for modularity and maintainability

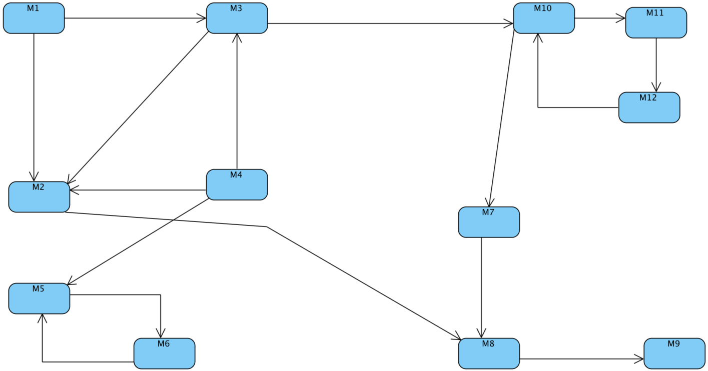
Modularity

- Some difficult questions:
  - how to decompose software?
  - what is the "good" granularity? (methods, classes, packages, etc.)
- To (try to) solve it, one can aim the "easyness" (or the cost) of maintenance and evolution of a piece of software
  - modular software have better chances to be reused
  - maintenance/evolution costs of monolithic software can be high

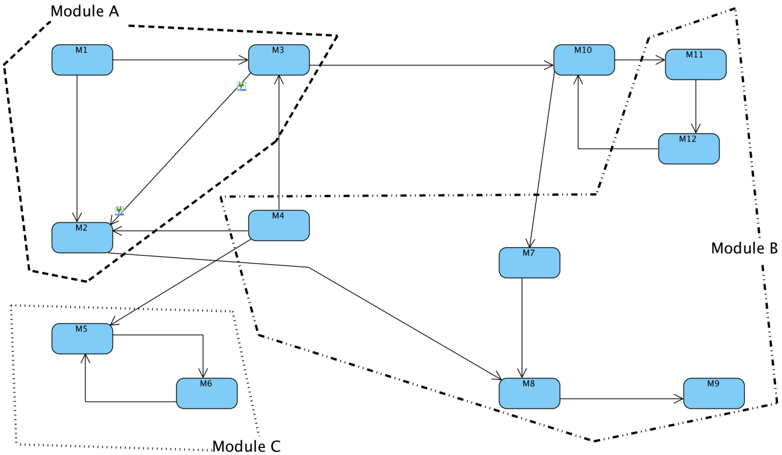
  $\Rightarrow$ modularity makes software evolution easier
- Concepts to try to control modularity and maintainability
  - coupling: greatly influences maintainability
  - cohesion: improves readability and maintenance

  $\Rightarrow$ Principle: *low coupling and high cohesion*

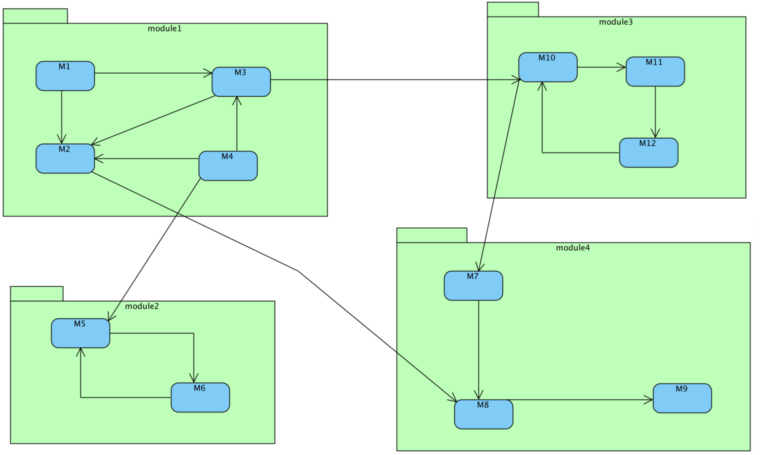
Modularity: example of a dependency graph



(Example) An ugly modularization



(Example) A better modularization



Measuring modularity: cohesion and coupling

- 16 links/arcs/dependencies
- Cohesion = # of internal dependencies
- Coupling = # of external dependencies
- Version #1
  - Cohesion = ?
  - Coupling = ?
- Version #2
  - Cohesion = ?
  - Coupling = ?

Measuring modularity: cohesion and coupling

- 16 links/arcs/dependencies
- Cohesion = # of internal dependencies
- Coupling = # of external dependencies
- Version #1
  - Cohesion = 8 : A=3 B=3 C=2 M10=0
  - Coupling = 8 : A–B=3 A–C=0 A–M10=1 B–C=1 B–M10=3 C–M0=0
- Version #2
  - Cohesion = ?
  - Coupling = ?

Measuring modularity: cohesion and coupling

- 16 links/arcs/dependencies
- Cohesion = # of internal dependencies
- Coupling = # of external dependencies
- Version #1
  - Cohesion = 8 : A=3 B=3 C=2 M10=0
  - Coupling = 8 : A–B=3 A–C=0 A–M10=1 B–C=1 B–M10=3 C–M0=0
- Version #2
  - Cohesion = 12 : 1=5 2=2 3=3 4=2
  - Coupling = 4 : m1–m2=1 m1–m3=1 m1–m4=1 m2–m3=0 m2–m4=0 m3–m4=1

Measuring modularity: cohesion and coupling

- 16 links/arcs/dependencies
- Cohesion = # of internal dependencies
- Coupling = # of external dependencies
- Version #1
  - Cohesion = 8 : A=3 B=3 C=2 M10=0
  - Coupling = 8 : A–B=3 A–C=0 A–M10=1 B–C=1 B–M10=3 C–M0=0
- Version #2
  - Cohesion = 12 : 1=5 2=2 3=3 4=2
  - Coupling = 4 : m1–m2=1 m1–m3=1 m1–m4=1 m2–m3=0 m2–m4=0 m3–m4=1

⇒ cohesion ++ ; coupling --

Metrology – practical session: PMD

PMD[2]

- static analyser
- uses a set of rules
- provides a lot of "indicators"
- can be customized
- can be complex to master

⇒ see more during the practical session (and do not hesitate to test it on your PetriNet project...)

[2] https://pmd.github.io/

Conclusion

- An introduction to IDEs and to software quality
- IDE = mandatory tool in a professional software development context
- Need to practice

- An introduction to metrology: be careful it is not an exact science
  - difficulty to establish and to interpret a metric
  - many tools, but not always consistent
  - relies on experiments (and biases)
  - there exists **much more** metrics than the ones presented during this lecture

Gentle reminder

⚠ Important notes

- If you do not understand something, please ask your questions. *We cannot answer the questions you do not ask...*
- If you disagree with us, please say it (politely)
- People don't learn computer science by only reading few academic slides: practicing is fundamental