Code quality (CQ) presentation

JM. Colley (JMC), LPNHE

Define CQ by use case (alone)

- Software project with single programmer
  - **no code distribution, publishing only result :** so just check your result !
  - executable/library distribution, to be credible you must provide :
    - Documentation
      - Installation procedure
      - API front-end software
    - Before the broadcast : serious high level testing

Define CQ by use case (friend)

- Software project led by 2 friends for life. What difference with (alone)
  - Certainly nothing, tasks are shared by friend and every thing is fine.

Define CQ by use case (team)

- Software project led by team with turn-over. What difference with (friend) :
  - Mr X is leaving project in 2 weeks, who can support the current refactoring a feature he was in charge of ?
  - Not me, I looked at his code it's incomprehensible there is no comment, he names these variables with 2 letters, there is plenty of commented code and hardly any test. He has been working on it for a month and has never updated with current dev branch
  - Oh, no we will be late again !

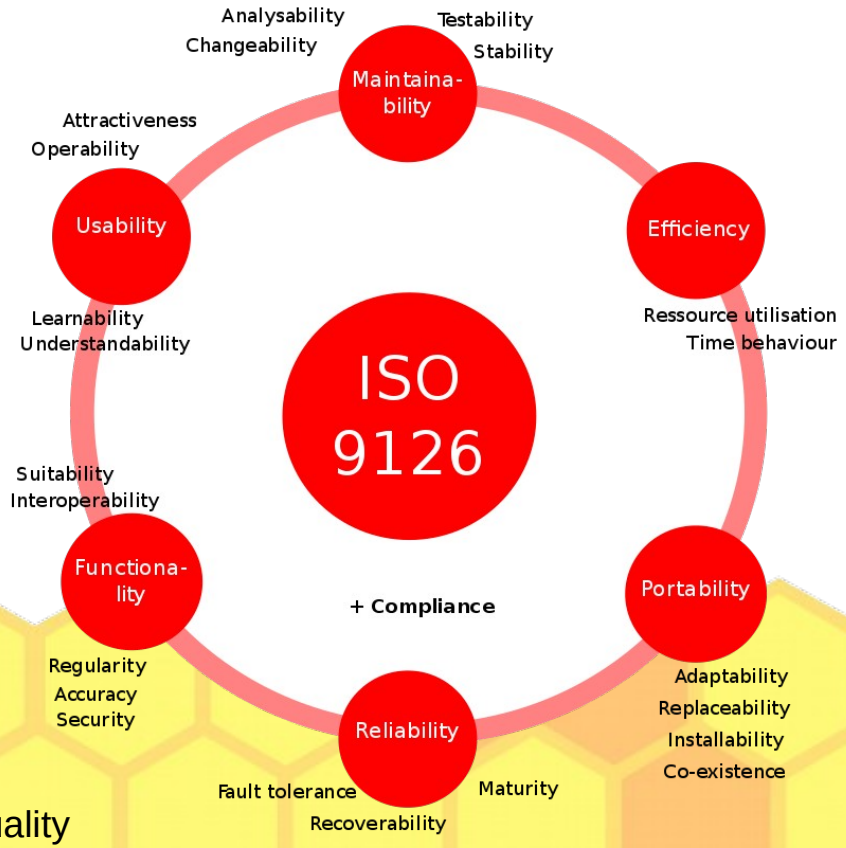
Define CQ by use case (team)

- So what do we need to prevent this : standardize the code
  - With some coding rules like :
    - Define namming convention, standard code presentation
    - Update API backend software with new code
    - Minimal test on new code
    - Permanently remove obsolete code
    - …
  - But also some coding organization rules
    - Description of implemented solution (in ticket, issue, document, ...)
    - Check coding rules regulary and integrate current modification to prevent merge problem for long feature developement
    - ...

Define CQ

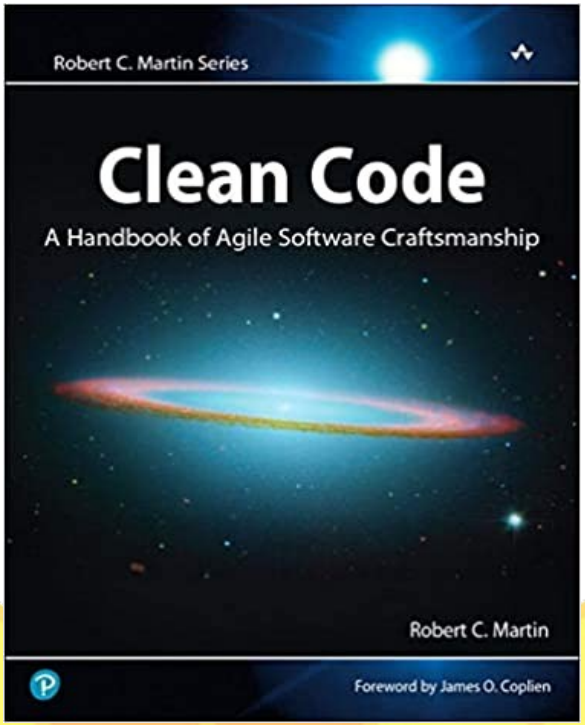
- Constraints appear when you let others use and develop your software
- Your software's response to these constraints is called software quality. 6 criteria are defined by ISO 9126:

Some criteria can be contradictory, the result must be a compromise



How to write quality software ?

- Difficult question, but it’s necessary to have good skill in software engineering, in software architecture in particular, testing experience, project management
- Knowledge good practice
  - Developement method : iterative (add features, refactoring, add features, ...)
  - Design pattern for oriented object approach
  - Seven testing principles and ISTQB
  - « Clean code » book by R. Martin Summary
  - Code review between dev in/out of the project
- Knowledge bad practice too
  - Anti-pattern
- And use the good tools ...



Evaluate CQ automatically ?

- Not easy for all criteria
- For « Maintainbility Reliability » you can apply specific langage tools and obtains some metrics to evalute your software
- 2 familly of tools, returns some CQ metrics, indicators
  - Static analysis
  - Dynamic analysis

Static analysis

- Don’t excute the code, just examine it
  - First tool for C nammed **lint**, by reference many tools take this name now
  - Can detect/evaluate : error, security problem, complexity, code duplication,coding rules like : namming convention, format, number of comments, …
  - In fact detect low level bad practice, integrate to compilator (enable warning option)
- Python examples
  - Pylint
  - Mypy (for error type if you use annotation)
  - with IDE (pydev, pycharm, sonarlint plugin)

Dynamic analysis

- Performed by executing program
  - Apply on specific case (find a bug, optimization) or tests suite
  - Can detect/evaluate : memory problem, coverage of tests, memory or time performance
- Python examples
  - Coverage.py
  - cProfile
  - With IDE (pycharm)

Others tools to help CQ

- Code formatter
  - black, ClangFormat (C++)
- Framework for test
  - Unittest, pytest, numpy.testing, Boost.Test(C++)
- Dashboard CQ
  - SonarQube (all languages)

Some words on SonarQube

- Provide a dashboard for a set of projects
- Each project has a set of metrics
- Provide static analysis tool (sonarlint) configurable for many langages, can be used in IDE but not in line command
- Can’t perform dynamic analysis, uses analysis reports from many tools
- Allows to define the **Quality Gate**, by a set of thresholds on the different metrics of the dashboard, so project has 2 status : **Passed** or **Failed**
- SonarQube can be update by github/gitlab to do continuous integration

Conclusions

- Code quality requires software engineering skill and experience
- a crash course on the subject is given for example by the book "clean code" by Robert Martin
- The practice of automatic testing is fundamental
- Do code reviews to detect high level bad practice
- Iterative developement and continuous evaluation of quality metrics seems the best way to write quality code.