



**Océ**

Bits&Chips 2011  
Embedded Systemen

18 november 2011  
Evoluon, Eindhoven

## Software quality at Océ: assurance using automated static code analysis



Edy Klomp, software engineer, Océ  
Paul Jansen, CEO, Tiobe

**Canon**  
CANON GROUP

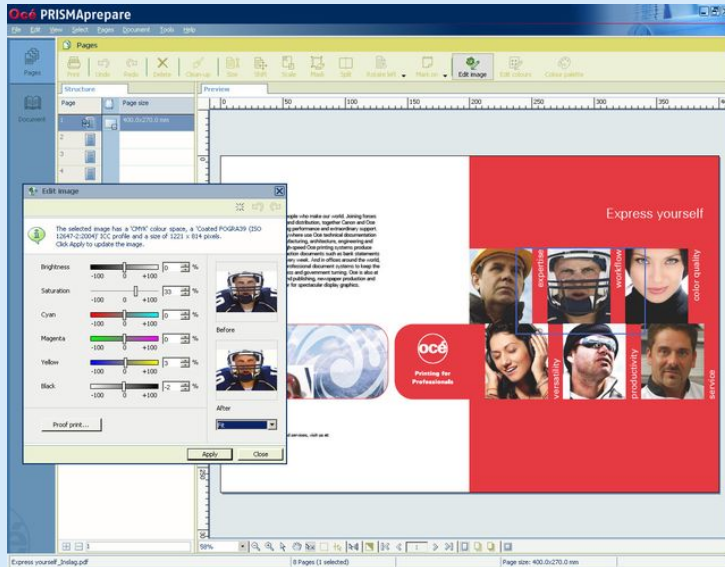
# Facts & Figures - Océ



- Founded in 1877
- Headquarters in Venlo, The Netherlands
- In 2010 Océ joined the Canon Group of companies with headquarters in Tokyo
- Around 20,000 people worldwide
- Total revenue 2010: € 2.7 billion
- Worldwide distribution in approx. 100 countries
- 10 R&D sites in 9 countries (1,600 people)
- Active in entire value chain of printing systems



# Facts & Figures - Océ products



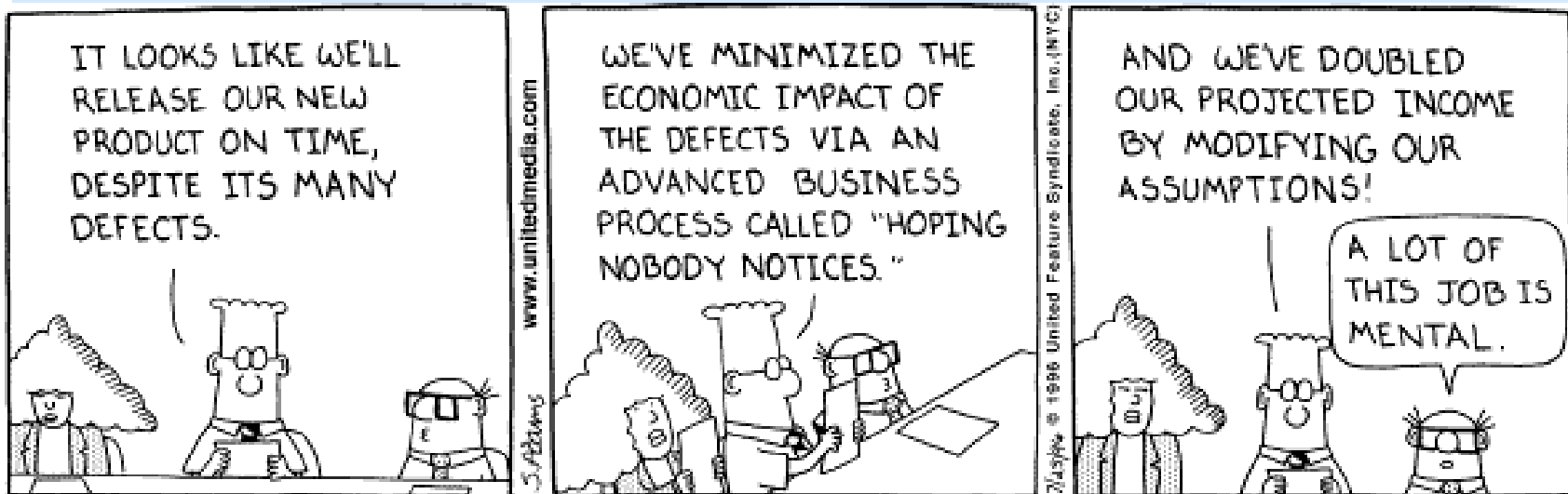
# Facts & Figures - Tiobe Software



- Founded in 2000
- Headquarters in Eindhoven
- Spin off of Philips Research
- Measuring and monitoring software code quality
- 140.2 Million LOCs checked each day for errors
- 620 different software projects



Quality must never be an outcome.



Dilbert ©2011, Universal Uclick

But: how do we assure software quality?

- Manual versus automated quality assurance
- What is static code analysis?
- TICS framework
- Static code analysis and TICS at Océ
- Benefits
- Challenges
- Observations and conclusions

## Pro manual:

- Low learning curve
- Flexible
- Good feeling on quality versus abstract score

## Pro automated:

- Reproducible
- Repeatable
- Objective (benchmarking possible)

- Automated testing
  - Dynamic
    - Functionals (unit/subsystem/product levels)
    - Non-functionals
  - Static
    - Compiler warnings
    - Static code analysis
    - Code metrics
    - Deep flow analysis

- Automated testing
  - Dynamic
    - Functionals (unit/subsystem/product levels)
    - Non-functionals
  - Static
    - Compiler warnings
    - Static code analysis
    - Code metrics
    - Deep flow analysis

# What is static code analysis?



- Automated analysis performed without execution
- Mostly used to check against coding standard
- Coding standard:
  - Style issues: use an indentation of 4 characters
  - Naming conventions: macros are in uppercase
  - Design rules: do not declare members public
  - Best practices: do not use goto statements
  - Programming errors: dereferences of null pointers
- Well known static analysis tools:
  - PC-Lint, QA-C, C++test, PMD, FxCop, Coverity

# Why static code analysis?



## Costs of repairing software defects



**Coding**



**Testing**



**Released**

## Key advantages:

- Automated
- Daily updated results
- Objective measures allow quantifying code quality
- Monitor trends
  - Coding standards compliance
  - Cyclomatic complexity
  - Lines Of Code change rates to detect “hotspots”
- Off-the-shelf solution: Tiobe TICS

**Company  
interested in  
Code Quality**

**Independent Layer TICS**

**Code  
Checking  
Tool 1**

**Code  
Checking  
Tool 2**

**Code  
Checking  
Tool 3**

**Code  
Checking  
Tool 4**

# TICS Framework within Océ



**Océ Technologies**

**Independent Layer TICS**

**C++test**

**TICS*c***

**FxCop &  
StyleCop**

**PMD**

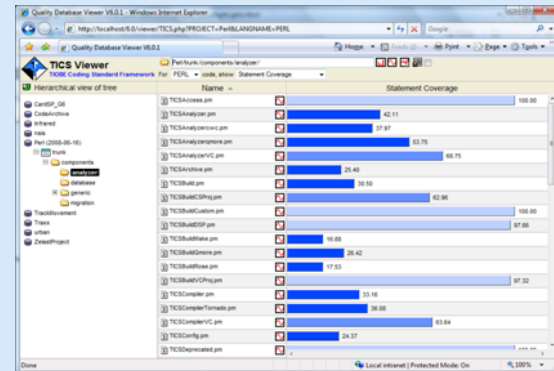
**Source  
Monitor**

# TICS: from bit to board



- TICS Enterprise Dashboard:
  - Target: board of management
  - World-wide real-time data
- TICS Intranet Viewer:
  - Target: local management
  - Drill down to sources
- TICS IDE Integration:
  - Target: software engineers
  - Detailed and before delivery

Project	Status	Metric 1	Metric 2	Metric 3	Metric 4	Metric 5	Metric 6	Metric 7	Metric 8	Metric 9	Metric 10
...	...	...	...	...	...	...	...	...	...	...	...



```
integers  
if (end($1) == 255) {  
    $1 = end($1) - 1;  
    // Sleep  
    case 1: if (1 <= end($1) == 0) {  
        1 <= end($1) = end($1) - 1;  
        $1 = end($1);  
        // Product  
        case 1: if (1 <= end($1) == 0) {  
            // ...  
        }  
    }  
}
```

- Problem: manage on violations of coding standard
- Confidence factor is measure of compliance
- Range between 0% and 100%
- Definition:
  - Confidence = Coverage \* Weighted Violation per rule
  - Coverage = %LOC that could be checked
  - Weighted Violation of a rule =  
Severity level factor \* #occurrences per LOC
- Target is 80%, Average of 620 projects: 74.84%

- Introduced in 2003
- 8.5 MLOC checked
- 48 projects
- Used at multiple sites
- Coding rules
  - Based on coding standards from outside Océ
  - Tuned towards Océ situation
  - Managed by a change control board
  - Synchronized over sites

- Confidence factor from TIOBE
  - Measuring the static quality of a program
  - Allows monitoring trends
- Starting point: coding standard violations
- Coding rules categories
  1. Programming errors, e.g. on comparing floats
  2. Potential programming error (as category 1, but exceptions possible)
  3. Maintenance/style, e.g. naming, indentation
  4. Generally considered good programming practices

# Coding standard challenges: specification



- Synchronization over languages: C, C++, C#, Java
- Language changes: C++11, C99, C# 4.0, Java 7
- Embedded versus controller/application software
- Usage of third-party libraries
- Agreement on rules
- Philosophy
  - “More rules is better”: I will filter myself
  - “Less rules is better”: I only want to see real issues

- Update interval:
  - Too frequent: “random” confidence fluctuations
  - Too infrequent: false positives annoyance
- Updates of rulesets via a well-communicated process
- Static code checkers need preprocessed input:
  - Dependency on build process
  - Dependency on compiler
  - Dependency on specific language extensions

- Effects on confidence
  - Changed level: + or -
  - Add or remove rule: + of -
  - Fix false positive: +
  - Fix false negative: -



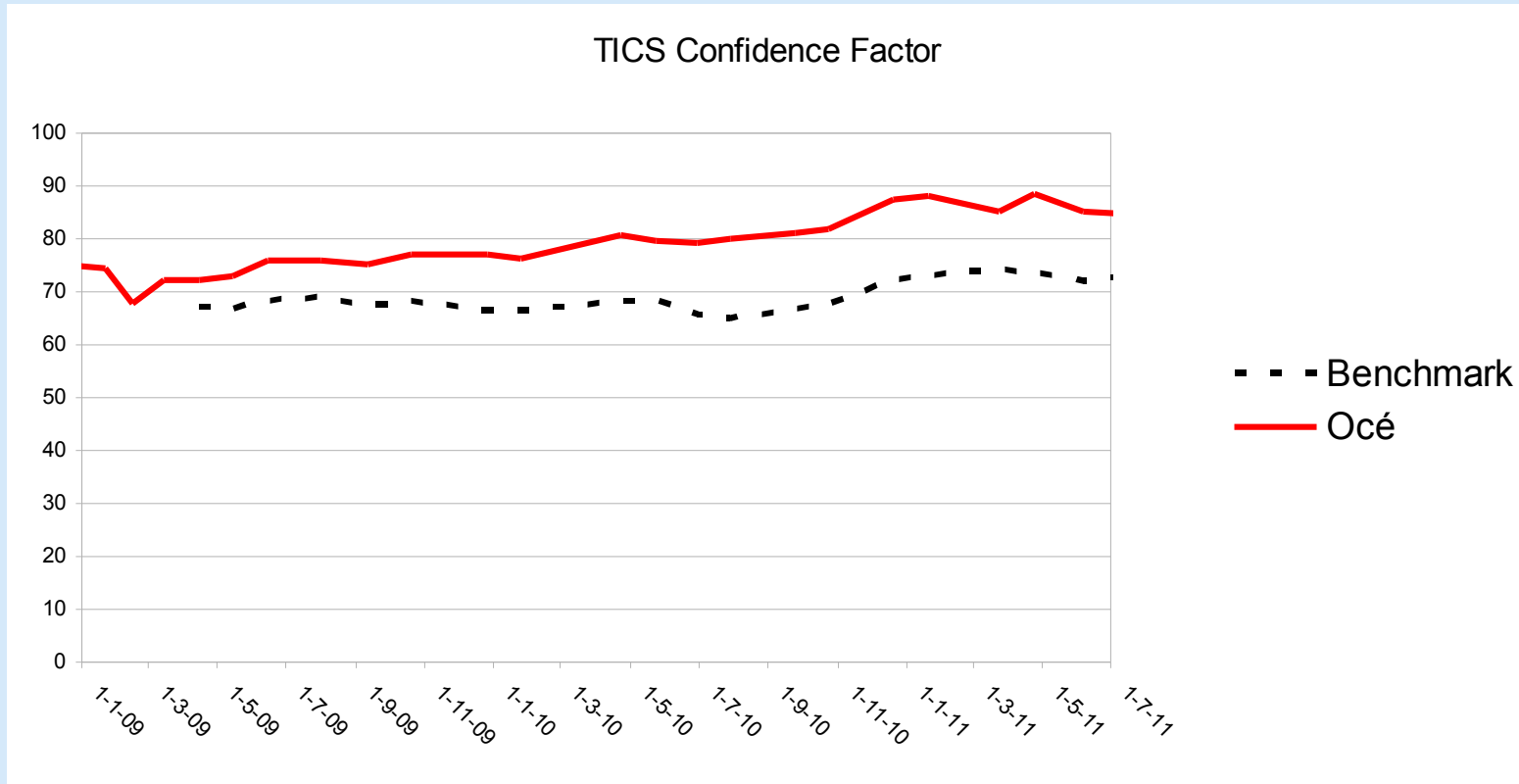
- Clash of interests
  - Latest rules to support new language constructs
  - Stable coding standard for projects nearing a release

- Communication is important when updating coding standards
- A 2-month update interval seems ideal
- Focus on the most important rules
- Important to keep developers involved
  - Avoid (better: discard) “taste” rules
  - Fix rules implementation defects fast
  - Increase speed of measurement runs

# Observations (2)



- Confidence *trend* is more important than the absolute score
- Confidence increasing over time

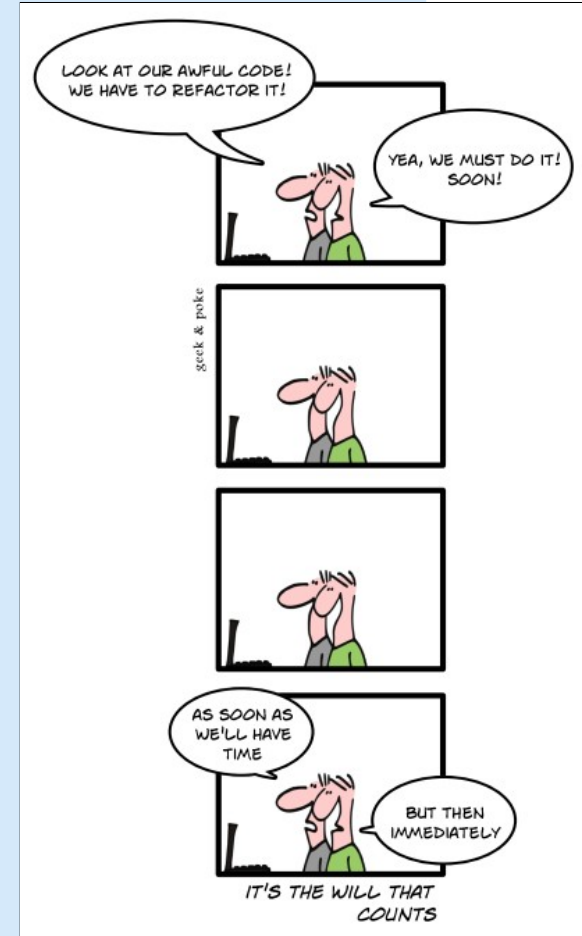


Static code analysis is not a walk in the park

- Long and bumpy road
- Continuous attention needed
- Beware of misuse
  - No tool satisfaction
  - Confidence factor should not be the only quality metric

But it does pay off

- Avoiding programming errors
- Improves consistency of code





**Printing for  
Professionals**