Software Engineering: Theory and Practice

Metrics

Tom Verhoeff

Eindhoven University of Technology
Department of Mathematics & Computer Science
Software Engineering & Technology

Feedback to `T.Verhoeff@TUE.NL`

GQM – TQM

- Goals (high-level, ultimate)
- Questions (specific)
- Metrics (how to)
- Total Quality Management (address quality everywhere)

Goals: What Do You Want to Achieve

- Quality control (objective vs. subjective, quantitative)
- Project planning & reporting
- Accountability (e.g. in case of an audit)
- Process improvement
- Measuring is not a goal in itself

Questions: What Do You Want to Know

- What is the quality of the system functionality?
- How much extra (time, staff) will it take to complete the system?
- What is the branch coverage of the tests?
- How complex is the system?
- How much time was spent on defect removal?

Metrics: What/How Do You Want to Measure

- What actually to measure?
- What measurement to use? Alternatives, trade-offs
- How to measure? (organize the measurement process)
- What to do with measurement results?

Measurement Process

1. Select metric(s) appropriate for goals and questions.

   Also consider: validation, calibration, tuning, staff training
2. Collect and store measurement data.
3. Consolidate and report: graphs, trend charts, ...
4. Interpret results w.r.t. goals and questions.
5. React: close the feedback loop.

What to measure

- Product-related
- Process-related

Fundamental Metrics

- Size of product (LOC, SLOC, FP, ABC, ...)
- Cost of project (Euro) [N.B. Not: product pricing]
- Duration of project (calendar months)
- Effort for project (person-months)
- Quality of product (number of remaining defects)
- Relationships, models
- Economy or diseconomy of scale: Effort = Size^e, $e < 1$ or $e > 1$

Size versus Complexity

- LOC, SLOC: (Source) Lines Of Code (SLOC = nonempty without comments)
- % of lines with comments
- # classes; # (public) methods or inst. var. per class
- # parameters or LOC per method
- depth of inheritance hierarchy
- # overridden methods
- % of duplicated code

Cyclomatic Complexity

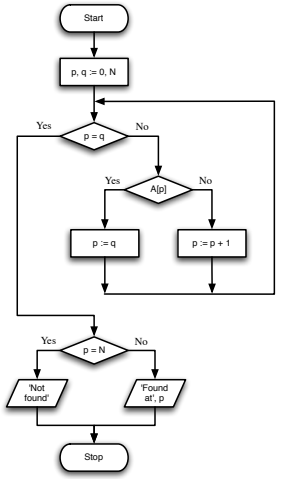
- McCabe 1976
- Measure for testability, understandability (maintainability)
- # linearly independent paths in flow graph
- # edges − # vertices + 2 ∗ # components
- # binary decisions + 1
- Typical reasonable upper bound to impose per module: 10

Code Metrics Example

Python code:

```
1 p, q = 0, N  # given A[0..N)
2
3 while p <> q :
4   if A[p] : p = q
5   else : p = p + 1
6
7 if p == N : print "Not found"
8 else : print "Found at", p
```

LOC = 8; SLOC = 6
A, B, C = 4, 2, 3
Cyclomatic complexity = 4



Measurement Tools

- Configuration management tools
- Defect/issue trackers
- Test tools (also measure coverage)
- Static code analyzers (e.g. JDepend)

Benchmark

- Point of reference, specifically for program/processor performance
- SPEC: Standard Performance Evaluation Corporation `www.spec.org`
- Dhrystone → SPEC CINT2000 (without floating point)
- Whetstone → SPEC CFP2000 (with floating point)

References

- "Applying the ABC Metric to C, C++, and Java" by Jerry Fitzpatrick. *C++ Report*, June 1997.