Improving quality of applications through monitoring – approaches and experiences

Tosin Daniel Oyetoyan
Trial lecture
Trondheim, Norway
June 26th 2015
Agenda

- Software Quality
- Software Quality Improvement Approaches
- Measurements and Metrics
- Estimations and Predictions
- Conclusion
Software Quality

- A complex and multifaceted concept
- Depends on the view (Garvin 1984)
  - Transcendental view: Something we can recognize
  - User view: fitness for purpose
  - Manufacturing view: conformance to specification
  - Product view: tied to inherent product characteristics
  - Value-based view: depends on the amount the customer is willing to pay for it

**Views and considerations** (Kitchenham 1996)

<table>
<thead>
<tr>
<th>User view</th>
<th>Product view</th>
<th>Manufacturer view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions provided by a product</td>
<td>Inherent characteristics of product</td>
<td>Constructed ‘right first time’</td>
</tr>
<tr>
<td>Usability (e.g. learning time), satisfaction,</td>
<td>Uses internal quality indicators</td>
<td>Process focused (ISO 9001, CMM)</td>
</tr>
<tr>
<td>reliability and performance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Influenced by constraints (delivery date,</td>
<td>Quality models (McCall, ISO) –</td>
<td>Defect counts, Rework costs (fixing</td>
</tr>
<tr>
<td>purchase cost, )</td>
<td>External quality factors</td>
<td>defects)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Software Quality

People

Business Management, Project Management, Development team, Customers, End users

Software

Project activities: planning, requirements analysis, design, implementation, testing, maintenance

Project deliverables

Project documentation, code, test documents, customer documents, productivity measurements

Software

Process

Waterfall, Iterative and Incremental Development (Spiral model), Agile (XP, Scrum, Crystal)
Agenda

- Software Quality
- Software Quality Improvement Approaches
- Measurements and Metrics
- Estimations and Predictions
- Experiences and Conclusion
Software Quality Assurance (SQA)

• Goal:
  – To improve software quality by appropriately monitoring the software and its development process (Humphrey 1989, van Vliet 2000)

• Key words
  – Improve (software quality)
  – Monitoring
    • Software product
    • Development process

Effective SQA Process (IEEE Std 730-2014)
• Identifies what to do
• How to do it well
• How to confirm it gets done right
• How to measure and track it
• How to learn from measures to manage and improve it, and
• How to encourage using it to improve software product quality
Software Quality Improvement

• Two areas of focus
  – Process improvement approaches
  – Product quality approaches

• Two stages
  – During development - different phases of the development process (requirement, design, implementation, testing)
  – After delivery - reflection on the project for learning (post mortem), post release issues (maintenance)

• Measurement
  – As a key to improving the quality of applications
  – Measurement provides means to understand, control, characterize, evaluate, predict and improve (Basili 1992, sFenton and Pfleeger 1997; Pressman 2000)
What to measure?

**Processes**
- Collection of software related activities (reviews, testing, etc.)

**Products**
- Any artifacts, deliverables or documents that result from a process activity

**Resources**
- Entities required by a process activity (personnel, materials, tools, methods)
Measurement as feedback

- We need feedback from the process, artifacts and resources to improve the quality of applications.

Examples:
1. How effective is the use of static analysis tool to detect defect in the code?
2. What is the error distribution by development phase for project X?
3. What is the estimated Mean Time To Failure of the product in the last release?
Quality in the software process

• Proponents assume that: (Halvorsen & Conradi 2001)
  – Quality(process) => Quality(Product)
  – If we improve the quality of the process, it will improve the quality of the application

• However, context could introduce confounding effects
  – software development is creative rather than a mechanical process
    • Influenced by individual skills and experience
    • Novelty of an application or time-to-market

Picture taken from: Halvorsen & Conradi 2001
Implied causal relation between SPI frameworks and product quality

Halvorsen & Conradi 2001 Taxonomy and mapping

<table>
<thead>
<tr>
<th>SPI Framework</th>
<th>Causal Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TQM</td>
<td>Not applicable</td>
</tr>
<tr>
<td>CMM</td>
<td>$F_1'(\text{Key Process areas}) \Rightarrow F_2'(\text{Maturity level}) \Rightarrow \text{Quality(Process)} \Rightarrow \text{Quality(Product)}$</td>
</tr>
<tr>
<td>ISO 9000</td>
<td>$F_1'(\text{Quality elements}) \Rightarrow F_2'(\text{Certification}) \Rightarrow \text{Quality(Process)} \Rightarrow \text{Quality(Product)}$</td>
</tr>
<tr>
<td>ISO/IEC 15504 (SPICE)</td>
<td>$F_1'(\text{Process attributes}) \Rightarrow F_2'(\text{Capability level}) \Rightarrow \text{Quality(Process)} \Rightarrow \text{Quality(Product)}$</td>
</tr>
<tr>
<td>QIP/GQM/EF</td>
<td>$F'(\text{Experience reuse}) \Rightarrow \text{Quality(Process)} \Rightarrow \text{Quality(Product)}$</td>
</tr>
<tr>
<td>SPIQ</td>
<td>$F'(\text{Experience reuse}) \Rightarrow \text{Quality(Process)} \Rightarrow \text{Quality(Product)}$</td>
</tr>
</tbody>
</table>

- TQM - Total Quality Management
- CMM - Capability Maturity Model
- SPICE – Software Process Improvement Capability dEtermination.
- QIP - Quality Improvement Paradigm,
- EF - Experience Factory,
- GQM - Goal Question Metric.
- SPIQ - Software Process Improvement for better Quality

Further Work

We have seen that the proposed taxonomy seems reasonable from a theoretical standpoint. However, there are many issues that can only be answered through practical use and further investigation:

- **Cost** – An evaluation of the costs/benefits incurred from use of the taxonomy, for example in terms of training and competence building. We also need more cost data on implementation of the various frameworks.
- **Effectiveness** – Which parts of the framework seem to be the most useful, for whom and for what? We may apply the Goal Question Metric (GQM) method on the taxonomy itself, in order to assess its usability to help making right SPI decisions.
- **Ease-of-use** – Which parts or subsets of the taxonomy appear to be the easiest or hardest to use?
Example: Capability Maturity Model (CMM)

- Defines the maturity level of development process which is an indicator of the quality of the product

- Level 1: Initial
  - Ad hoc
  - Requirements management
  - Software product planning
  - Software project tracking and oversight
  - Software subcontract management
  - Software quality assurance
  - Software configuration management

- Level 2: Repeatable
  - Organization process focus
  - Organization process definition
  - Training program
  - Integrated software management
  - Software product engineering
  - Intergroup coordination
  - Peer review

- Level 3: Defined
  - Quantitative process management
  - Software quality management

- Level 4: Managed
  - Defect prevention
  - Technology change management
  - Process change management

- Level 5: Optimizing

***There are examples and counter examples about the relationships between the maturity level and product improvement (e.g. Herbsleb & Goldenson 1996)***
Quality in the software product

• Proponents assume that
  – Quality(Product Internal properties) => Quality(Product external properties)

• Quality models (External quality factors)
  – McCall 1977
  – Boehm 1978
  – ISO 9126
Decomposing external quality factors to measurable entity

- **Factor**: Maintainability, Correctability, Testability, Expandability
- **Criteria**: Defect counts, Degree of testing, Change counts
- **Metric**: Closure time, Isolate/fix time, Defect rate, Statement coverage, Branch coverage, Test plan completeness, Resource prediction, Effort expenditure, Change effort, Change size, Change rate

**Examples:**
- What is the response time (change effort) to add a module (modify) to a chosen architecture?
- What is the defect rate of the product after reusing a design framework?

*Picture adapted from: Fenton and Pfleeger 1997*
Agenda

- Software Quality
- Software Quality Improvement Approaches
- Measurements and Metrics
- Estimations and Predictions
- Conclusion
Goal Question Metric (GQM) Approach

- What to measure and how to measure it? GQM
- Measuring what is needed and not what is convenient or easy to measure (driven Top-down)
- To be successful, the goal for measurement must be clear and focused (e.g. Improving software quality)
- Define set of questions to be answered to determine whether the goal is being met (e.g. How effective is the inspection process?)
- Decide what should be measured to be able to answer the questions adequately (e.g. Avg. defects detected per lines of code)
**Example: GQM**

- **AT&T Experience:** Measure the effectiveness of inspections *(Barnard and Price 1994)*

<table>
<thead>
<tr>
<th>Goal</th>
<th>Questions</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan</td>
<td>How much does the inspection process cost?</td>
<td>Average effort per KLOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Percentage of reinspections</td>
</tr>
<tr>
<td></td>
<td>How much calendar time does the inspection process take?</td>
<td>Average effort per KLOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Total KLOC inspected</td>
</tr>
<tr>
<td>Monitor and Control</td>
<td>What is the quality of the inspected code?</td>
<td>Average faults detected per KLOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Average inspection rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Average preparation rate</td>
</tr>
<tr>
<td></td>
<td>To what degree did the staff conform to the procedures?</td>
<td>Average inspection rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Average preparation rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Average lines of code inspected</td>
</tr>
<tr>
<td></td>
<td>What is the status of the inspection process?</td>
<td>Total KLOC inspected</td>
</tr>
<tr>
<td>Improve</td>
<td>How effective is the inspection process?</td>
<td>Average removal efficiency</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>What is the productivity of the inspection process?</td>
<td>Average effort per fault detected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

**Claimed Results**

- Defect removal efficiency > 70%
- Reduced cost of removing defects with code inspection by 300%
- Productivity of code inspection doubled
Internal and External measures

In measurement program, we have a set of internal attributes that we want to understand against an external attributes.

<table>
<thead>
<tr>
<th>Products</th>
<th>Internal</th>
<th>External</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification</td>
<td>Size, reuse, modularity, syntactic correctness</td>
<td>Comprehensibility, maintainability</td>
</tr>
<tr>
<td>Designs</td>
<td>Size, coupling, cohesiveness</td>
<td>Complexity, maintainability</td>
</tr>
<tr>
<td>Code</td>
<td>Size, coupling, cohesiveness, control-flow structuredness</td>
<td>Reliability, usability, maintainability</td>
</tr>
<tr>
<td>Test data</td>
<td>Size, coverage level</td>
<td>Quality</td>
</tr>
</tbody>
</table>
## Internal and External measures

<table>
<thead>
<tr>
<th>Processes</th>
<th>Internal</th>
<th>External</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructing Specification</td>
<td>Time, Effort, number of requirement changes</td>
<td>Quality, cost, stability</td>
</tr>
<tr>
<td>Detailed design</td>
<td>Time, Effort, number of specification faults found</td>
<td>Cost, cost-effectiveness</td>
</tr>
<tr>
<td>Testing</td>
<td>Time, Effort, number of coding faults found</td>
<td>Cost, cost-effectiveness, stability</td>
</tr>
</tbody>
</table>
# Internal and External measures

<table>
<thead>
<tr>
<th>Resources</th>
<th>Internal</th>
<th>External</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personnel</td>
<td>Age, price</td>
<td>Productivity, experience, intelligence</td>
</tr>
<tr>
<td>Teams</td>
<td>Size, communication level, structuredness,</td>
<td>Productivity, quality</td>
</tr>
<tr>
<td>Offices</td>
<td>Size, temperature, light</td>
<td>Comfort, quality</td>
</tr>
</tbody>
</table>
Agenda

- Software Quality
- Software Quality Approaches
- Measurements and Metrics
- Estimations and Predictions
- Conclusion
Estimations and Prediction models

• Various concerns at each life-cycle phase
  – What is the likely size of the requirements in current version?
  – What is the likely complexity of the current product based on previous releases?
  – How much testing effort would be needed for the current software release?

Picture adapted from: Fenton and Pfleeger 1997
Estimation and Prediction models

- Cost and effort (e.g. COCOMO, Function point)
- Reliability (e.g. Jelinski-Moranda)
- Defect
Cost, Effort and Reliability models

Cost & Effort

- e.g. COCOMO model
- Effort (person-month) is a function of size
- Basic, intermediate and advanced
- All take the form: \( E = a \times \text{Size}^b \times F \)
- Where: \( a \) and \( b \) are calibrated based on the type of software under construction (organic, semi-detached or embedded)
- \( F \) (effort adjustment factor) is product of the cost driver values

Reliability model

- e.g. Jelinski-Moranda model
- \( F_i(t_i) = 1 - e^{-\lambda_i t_i} \)
- \( \lambda_i = (N-i+1)\phi \)
- Where \( N = \) initial number of defects and \( \phi = \) failure rate
- Both \( N \) and \( \phi \) are estimated to predict the next failure time \( t_i \) (e.g. using MLE)

Key concepts in discussing reliability:
- Defect
- Failure
- Time (Execution time & calendar time)

Three kinds of time intervals:
- **MTTF**: Mean Time To Failure
- **MTTR**: Mean Time To Repair
- **MTBF**: Mean Time Between Failures (\( = \) MTTF + MTTR)
Software defects

- Can occur in any of the development phase

Defects
- Requirements defect
- Design defect
- Code defect
- Test case defect
- Operational defect

Why analyze and predict defect?
- Assess project progress and plan defect detection activities
- Assess product quality for improvement
- Assess process performance and improve capability
Defects from project & process factors

- Requirements adequacy
- Size, complexity
- COTS & reused code
- Development team capability
  - Problem solving, designing, coding
- Development team experience
  - Application domain, language & tools, platform
- Process maturity

COTS: Commercial-Off-The-Shelf
Measurement and defect data sources

Verification: Are we building the product right?
Validation: Did we build the right product?

Figure adapted from: Clark & Zubrow 2001
Defect Prediction techniques

A panel at IEEE Metrics 2002 (Shull et al. 2002) concluded that manual software reviews can only find $\approx 60\%$ of defects.
Defect prediction: Project Management

Orthogonal defect classification

- Used to identify areas of process improvement based on analysis of defect types, triggers, impact, and source
- Types: Algorithm, Assignment, Checking, Timing, Interface,…
- Triggers: testing, inspection, conditions of operational use
Defect proneness evaluation: Product Assessment

- **Goal:** To predict the defect-proneness of a software artifact

**Independent variables**
- Size (e.g. LOC), complexity (e.g. McCabe & Halstead)
- Fan-in/Fan-out
- Object-oriented (e.g. WMC, RFC, CBO, DIH, NOC)
- Process (e.g. code churn, code delta, developer metrics)
- Code reuse (e.g. type of import)
- Social-network or graph (e.g. size, density, betweenness centrality, closeness centrality, reachability)

**Modeling techniques**
- **Statistical** (logistic, linear, univariate, multivariate regression, Moving averages, Exponential smoothing)
- **Machine learning** (Naïve bayes, decision tree, random forest, neural network, support vector machine, clustering)

**Dependent variables**
- Defect-proneness (classification)
- Defect counts (ranking)
- Defect-severity (classification)
- Defect-density (classification)

**Granularity**
- Method, Class, File, Package
- Pre or post release

**Requires historical defect data**
Defect prediction: Process Improvement

Defect prevention

- Perform root cause analysis of most frequently occurring defects
- Take sample of defect reports for in-depth causal analysis
- Can be used prior to launching new projects or starting new phases of a project
Other quality modeling approach

• **Code smells**
  – God class
  – Long method
  – Duplicated method (clone)
  – Feature envy

• **Anti-patterns**
  – Structural metrics
    • Degenerated inheritance
    • Circular dependency
    • Subtype knowledge
    • Reachability distribution

• **Investigate against change- and defect-proneness**

Analysis
  o Fisher’s exact
  o proportion tests
  o Correlation
  o Rule-based
  o Bayesian Belief Network

Post Mortem & Error report analysis

PMA
- Learning from experience
- Both successes and failures in software projects can help to improve the process
- Applied at the end of the project
- Questions: Which aspect of the project worked well and should be repeated?
  - Which aspect worked badly and should be avoided?
  - Which aspect is merely OK but leaves room for improvement
- Used by Apple Computers, Microsoft

Error report analysis (ERA)
- A source for SPI
- Uses the defect tracking system (DTS)
- e.g. Li et al. (2010) improved the DTS of 2 companies using ERA technique
- They also showed that cost drivers for maintenance activities are different in the 2 organizations.
Agenda

- Software Quality
- Software Quality Improvement Approaches
- Measurements and Metrics
- Estimations and Predictions
- Conclusion
## Process improvement

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements process improvement, increased customer satisfaction, better project planning,</td>
<td>SPIs power lies in the project team Flexibility is needed to achieve the benefits of standard conformance and agile practices</td>
<td>Non-representative</td>
</tr>
<tr>
<td>Shorter project completion time</td>
<td>Focus is on continuous improvement of the performance of ongoing project</td>
<td>Inconclusive</td>
</tr>
<tr>
<td>High return on investment of 677% for Motorola – rework cost</td>
<td>~40-70% defect reduction Significant increase in productivity (Sutherland et al., Jarvis &amp; Gristock.)</td>
<td>Insufficient validation in general</td>
</tr>
<tr>
<td>improvement in delivery rate and quality (defects/function points)</td>
<td></td>
<td>Less than favorable experiences</td>
</tr>
</tbody>
</table>

Designed for large businesses
# Product quality

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Static code metrics: Useful, Generalizable, Easy to use and widely used</td>
<td>Most complexity metrics correlate with LOC</td>
</tr>
<tr>
<td>Success in various telecom, energy, technology, and government markets, including organizations such as NASA, Compagnie Financière Alcatel (Alcatel); Chevron Corporation; LogLogic, Inc.; and Northrop Grumman Corporation. GB Tech, Inc</td>
<td>Causality is mostly missed and uncertainties not modeled</td>
</tr>
<tr>
<td>Model can be reused across projects</td>
<td>Finding large number of defects may not improve reliability as only few defects lead to most of the observed failures</td>
</tr>
<tr>
<td></td>
<td>conflicts, contradictions and Anomalies with code smell detection</td>
</tr>
<tr>
<td></td>
<td>Project effort for refactoring smell</td>
</tr>
</tbody>
</table>
Summary and conclusion

• There is no silver bullet to software quality improvement
• Process, product and the resources need to be monitored for improving quality
• Measurement for quality improvement should be goal-driven
• Prediction models for quality improvement are useful but context is important
• Using both approaches of inspections (manual & tool-based) and prediction model are useful
• Software development is a creative process and confounding effects must be evaluated
References

• Books

References

• Articles

References


THANK YOU
Q&A?