Best Practices for Software Process and Product Measurement

Dr. Bill Curtis
Executive Director, CISQ
SVP & Chief Scientist, CAST Research Labs

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Dr. Bill Curtis is Senior Vice President Chief Scientist of CAST Software, where he heads CAST Research Labs. CAST is a leading provider of technology for measuring the structural quality of business applications. He leads the production of CAST’s annual CRASH Report on the global state of structural quality in business applications. He is also the Director of the Consortium for IT Software Quality, a Special Interest Group of the Object Management Group (OMG) where he leads an international effort to establish standards for automating measures of software quality characteristics. Dr. Curtis is a Fellow of the Institute of Electrical and Electronics Engineers (IEEE) for his career contributions to software process improvement and measurement.

Dr. Curtis is a co-author of the Capability Maturity Model for Software (CMM), the People CMM, and the Business Process Maturity Model. He was also a member of the Product Suite Team for CMMI. He is a globally recognized expert in software measurement, process improvement, and workforce development. He has been leader in the application of statistics to software engineering. He was the Chief Process Officer at Borland Software Corporation. Until it’s acquisition by Borland he was a Co-founder and Chief Scientist of TeraQuest in Austin, Texas, the premier provider of CMM-based services. He is a former Director of the Software Process Program in the Software Engineering Institute at Carnegie Mellon University. Prior to joining the SEI, Dr. Curtis directed research on advanced user interface technologies and the software design process at MCC, the US’s fifth generation computer research center in Austin, Texas. Prior to MCC he developed a global software productivity and quality measurement system at ITT’s Programming Technology Center. Prior to ITT he evaluated software development methods in GE Space Division, worked in organizational effectiveness at Weyerhaeuser, and taught statistics at the University of Washington. He has co-authored four books, is on the editorial boards of several journals, and has published over 150 papers on software development and management. Dr. Curtis earned a Ph.D. from Texas Christian University and a M.A. from The University of Texas. He earned his B.A. from Eckerd College in mathematics, psychology, and theater.
I. Measuring Software Size and Productivity 3
   Break
II. Adopting Measures to Development Methods 33
   Lunch
III. Measuring Software Improvement Programs 61
   Break
IV. Measuring Software Product Quality 89
   Adjourn
V. References 115
Section 1
Measuring Software Size and Productivity

1. Practical Software Measurement
2. Size and functional measures
3. Measuring software productivity
4. Analyzing software productivity
Badly defined measures

Too many measures

Unreliable data

Use in personal appraisals

Poorly matched to process maturity

Why Does Measurement Languish?
Measurement Process Guidance

- Based on decades of software measurement best practice
- Best guidance for starting a measurement program
- Compatible with ISO 15939 – *Software Measurement Process*
- Free guides & licensed training and consulting available

- [http://psmsc.com/](http://psmsc.com/)

# Measurement Categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Issue</th>
<th>Example measure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Schedule &amp; Progress</strong></td>
<td>Milestone completion</td>
<td>% Actual vs. planned dates</td>
</tr>
<tr>
<td></td>
<td>Earned value</td>
<td>Actual vs. planned completions</td>
</tr>
<tr>
<td><strong>Resources &amp; Cost</strong></td>
<td>Personnel effort</td>
<td>Person-hours</td>
</tr>
<tr>
<td></td>
<td>Facilities used</td>
<td>Test equipment &amp; hours</td>
</tr>
<tr>
<td><strong>Product Size</strong></td>
<td>Size of coded product</td>
<td>Function Points</td>
</tr>
<tr>
<td></td>
<td>Size of documentation</td>
<td>Pages of manuals</td>
</tr>
<tr>
<td><strong>Product Quality</strong></td>
<td>Functional quality</td>
<td>Defects per KLOC</td>
</tr>
<tr>
<td></td>
<td>Structural quality</td>
<td>Violations of reliability rules</td>
</tr>
<tr>
<td><strong>Process Performance</strong></td>
<td>Efficiency</td>
<td>Defect detection efficiency</td>
</tr>
<tr>
<td></td>
<td>Compliance</td>
<td>Audit results</td>
</tr>
<tr>
<td><strong>Product Value</strong></td>
<td>Return on investment</td>
<td>Increase in target revenue</td>
</tr>
<tr>
<td></td>
<td>Risk avoidance</td>
<td>Cost of system outage</td>
</tr>
<tr>
<td><strong>Customer Satisfaction</strong></td>
<td>Account growth</td>
<td>Repeat business within 1 year</td>
</tr>
<tr>
<td></td>
<td>Self-sufficiency</td>
<td>Number of helpdesk calls</td>
</tr>
</tbody>
</table>

Productivity Analysis Objectives

- Improvement
- Estimation
- Productivity Analysis
- Benchmarking
- Managing Vendors
Productivity Analysis Measures

Size
- Instructions
- Functions
- Requirements

Primary Measures

Effort
- Hours
- Roles
- Phases

Productivity Analysis
**Software Productivity**

Software Productivity = \[
\frac{\text{Size of software produced}}{\text{Total effort expended to produce it}}
\]

**Release Productivity**

Release Productivity = \[
\frac{\text{Size of software \{ developed, deleted, or modified \}}}{\text{Total effort expended on the release}}
\]
## Software Size Measures

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Most frequently used. Different definitions of a line can cause counts to vary by 10x. Smaller programs often accomplish the same functionality with higher quality coding.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirements-based</th>
<th>Use Case Points, Story Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case Points have not become widely used and need more development. Story points are subjective to each team and are susceptible to several forms of bias.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functions</th>
<th>Function Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Popular in IT. Several counting schemes (IFPUG, NESMA, Mark II, COSMIC, etc.). Manual counting is expensive and subjective—certified counters can differ by 10%. Automated FPs taking root.</td>
<td></td>
</tr>
</tbody>
</table>
```c
#define LOWER 0    /* lower limit of table */
#define UPPER 300   /* upper limit */
#define STEP 20     /* step size */

main()     /* print a Fahrenheit-Celsius conversion table */
{
    int fahr;
    for(fahr=LOWER; fahr<=UPPER; fahr=fahr+STEP)
        printf("%4d %6.1f
", fahr, (5.0/9.0)*(fahr-32));
}
```

The image also includes a bar chart showing the number of lines of code and the number of votes received. The chart indicates that the majority of the votes are for 6 or 7 lines of code. The reference mentioned is B. Park (1992).
Functional view of software

- Functional size can be estimated from external inputs and outputs
- Upfront functional analysis provides basis for good estimates
- Repository of FP data provides basis for calibrating estimates

Unadjusted Function Points

Unadjusted Function Points vs. EIs + EOs

\[ R^2 = .95 \]
\[ y = 7.79x + 43.50 \]
Effort — Weakest Measure

After the fact estimates
- Memory lapses
- Time-splicing
- Inconsistency

Under-reporting
- Contract issues
- HR issues
- Impressions

Lack of normalization
- Roles included
- Phases included
- Hours in P-Year

Unreliable, Inconsistent
**Assumption:** Productivity is a stable number

**Reality:** Productivity is unstable, tending to decline

Unless you take action !!!
Carry-forward Rework

- Release N
  - Develop N
  - Rework N
- Unfixed defects release N

- Release N+1
  - Develop N+1
  - Rework N+1
  - Rework N+0
- Unfixed defects release N

- Release N+2
  - Develop N+2
  - Rework N+2
  - Rework N+1
  - Rework N+0

- Unfixed defects release N+1
Example of Quality Impact

Project A (Plodders)
- 20 developers, 3 months
- $120k per FTE
- 3 FPs per staff month
- 180 FPs delivered
  - $3,333/FP cost

Project B (Better, Faster, Cheaper)
- 20 developers, 3 months
- $120k per FTE
- 4 FPs per staff month
- 240 FPs delivered
  - $2,500/FP cost

Project B is 25% more productive

However !!!
- 2 critical violations per FP
- $500 per fix
- Cost for 360 fixes = $180k
- Total Cost to Own = $780k
  - $4,333/FP of TCO

- 5 critical violations per FP
- $500 per fix
- Cost for 1200 fixes = $600k
- Total Cost to Own = $1,200k
  - $5,000/FP of TCO

Project A is 13.4% more productive
Size of both functional and non-functional code segments

Must add future effort to fix bugs into productivity calculations

Corrective effort in future releases for defects injected in this release

Automated Enhancement Points

Automated Technical Debt

Quality-Adjusted Productivity

Effort & Cost

Productivity

Estimation

Benchmarks

Value & ROI

Etc.
1) Segment baselines
2) Beware sampling effects
3) Understand variation
4) Evaluate demographics
5) Investigate distributions
6) Account for maturity effects
7) Beware external data sets
Evaluate Demographics

- Routine reports
- Custom ERP
- Web-based apps
- Innovative mobile
- COBOL apps

Function Points

Productivity
## Segment Baselines

Multiple baselines are usually the most valid

<table>
<thead>
<tr>
<th>Year</th>
<th>Projects</th>
<th>Productivity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total Corporate</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1981</td>
<td>28</td>
<td>2342</td>
</tr>
<tr>
<td>1980</td>
<td>21</td>
<td>1939</td>
</tr>
</tbody>
</table>
3 — Beware Sampling Effects

![Graph showing lines of code/person-years from 1980 to 1982.](image)

- **Baselines**
  - Lots of small projects
  - Lots of large projects
**Product factors:**
- timing constraints
- memory utilization
- CPU occupancy
- resource constraints
- complexity
- product size

**Project factors:**
- concurrent development
- development computer
- requirements definition
- requirements churn
- programming practices
- personnel experience
- client experience
- client participation

R²
- \( R^2_{\text{product}} = .16 \)
- \( R^2_{\text{project}} = .49 \)
- \( R^2_{\text{model}} = .65 \)

Investigate Distributions

Relative productivity (% of mean)

Software Engineering Practices Usage

Low    Medium    High
Which organization will be required to spend more effort on correcting software flaws?
7 — Caution for External Data Sets

Productivity
- Us
- DACS

- Data definitions
- Data collection
- Data validity
- Data age
- Demographics

Size
Section 2
Measuring to Manage Software Projects

1) Tracking traditional projects
2) Iterative and Agile measurement
3) Team Software Process measurement
Measuring Project Progress

“The measures say we are on schedule”

“But the code keeps growing. We should have measured the requirements growth”

“And all the defect reports remaining to be fixed will keep us far behind schedule”
Requirements Change Impact

Number of requirements affected

- **Requirements Added**
- **Requirements Changed**
- **Requirements Deleted**

- **February**: 10, 20
- **March**: 10, 12
- **April**: 20, 25
- **May**: 60, 70
- **June**: 20, 22
- **July**: 10, 14
- **August**: 20, 24
### Phase Defect Removal Model

<table>
<thead>
<tr>
<th>PHASE</th>
<th>Escapes Previous Phase / KSLOC</th>
<th>Defect Injection / KSLOC</th>
<th>Subtotal / KSLOC</th>
<th>Removal Effectiveness</th>
<th>Defect Removal / KSLOC</th>
<th>Escapes at phase exit / KSLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rqmts</td>
<td>0</td>
<td>1.2</td>
<td>1.2</td>
<td>0%</td>
<td>0</td>
<td>1.2</td>
</tr>
<tr>
<td>Design</td>
<td>1.2</td>
<td>18.0</td>
<td>19.2</td>
<td>76%</td>
<td>14.6</td>
<td>4.6</td>
</tr>
<tr>
<td>Code &amp; U.T.</td>
<td>4.6</td>
<td>15.4</td>
<td>20.0</td>
<td>71%</td>
<td>14.2</td>
<td>5.8</td>
</tr>
<tr>
<td>Integration</td>
<td>5.8</td>
<td>0</td>
<td>5.8</td>
<td>67%</td>
<td>3.9</td>
<td>1.9</td>
</tr>
<tr>
<td>Sys. Test</td>
<td>1.9</td>
<td>0</td>
<td>1.9</td>
<td>58%</td>
<td>1.1</td>
<td>0.8</td>
</tr>
<tr>
<td>Operation</td>
<td>0.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### SCRUM Board

- Scrumboard displays the path to completion for each story by showing task status.
- Story points are an estimate of days for each story.
- Agile estimating is by feature (story) rather than by task.
- ‘Done’ does not always mean all tasks are finished.
- Kanban methods restrict the work in progress to a limited number of paths.

#### Table: Scrumboard Details

<table>
<thead>
<tr>
<th>Story Description</th>
<th>Not Started</th>
<th>In Progress</th>
<th>To Verify</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td>As a CFO, I would like to see a monthly sales trend. [JPN]</td>
<td>6</td>
<td>- Impeded 8</td>
<td>Modify UI to allow selection of new report</td>
<td>8</td>
</tr>
<tr>
<td>Update application documentation and rebuild help file</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>As an Accountant, I would like to create invoices for all outstanding charges for all customers</td>
<td>4</td>
<td>Write code to select appropriate data for invoicing</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Write Unit Tests to check data selection criteria</td>
<td></td>
<td>Design Invoice template</td>
<td>Research reporting engines</td>
<td>0</td>
</tr>
<tr>
<td>Implement UI</td>
<td>8</td>
<td>Wireframe UI for the Invoice Run</td>
<td>Review the viable choices for Reporting Engines with the team for input</td>
<td>0</td>
</tr>
<tr>
<td>Implement and run Functional Tests</td>
<td>8</td>
<td></td>
<td>Learn API of the reporting engine</td>
<td>0</td>
</tr>
<tr>
<td>Update documentation</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Burndown Chart and Velocity

- Burndown chart displays story completion by day
- Burndown chart indicates actual versus estimated progress
- Velocity is the rate at which stories are completed
- Velocity indicates sustainable progress
- Velocity results provide one source of historical data for improving estimates
- Story points without the context of productivity factors are dangerous for estimating
Analyzing Velocity

**Burndown Chart**

**Item Cycle Times**

**Team Velocity**

**Story Points per Hour**

![Burndown Chart](image)

![Item Cycle Times](image)

![Team Velocity](image)

![Story Points per Hour](image)
Measuring and Managing Flow

Cumulative Flow Diagram

Days to delivery

Track and correlate multiple measures across sprints

Predictive models
Sources and Measures

Data sources

- Project tracking
- Code control
- Bug tracking
- Build system
- Test environment
- Deployment tools
- Operational monitoring

Measures

- Stories submitted
- Stories pulled back
- Growth in size of stories
- Stories added mid-sprint
- Stories under review
- Failed tests
- Non-development tasks
- Assists to other projects
Recommended Books

1. Agile Metrics in Action
   - Author: Christopher W. H. Davis
   - Publisher: O'Reilly

2. Software Development Metrics
   - Author: David Nicolette
   - Publisher: Addison-Wesley

3. Metrics and Models in Software Quality Engineering
   - Second Edition
   - Author: Stephen H. Kan
   - Foreword by Capers Jones
   - Publisher: CRC Press

4. Site Reliability Engineering
   - Author: Niall Richard Murphy
   - Publisher: O'Reilly
Personal Software Process (PSP)


**PSP 0**
- Current process
- Time recording
- Defect recording
- Defect type standards

**PSP 0.1**
- Coding standard
- Size measurement
- Process improvement proposal

**PSP 1**
- Size estimating
- Test report

**PSP 1.1**
- Task planning
- Schedule planning

**PSP 2**
- Code reviews
- Design reviews

**PSP 2.1**
- Cyclic development
- Design templates

**Cyclic Personal Process**

**Personal Quality Management**

**Personal Planning**

**Baseline Personal Process**

---

Team Software Process (TSP)

- Built from personal processes of team members
- Well defined team roles
- Project launch workshops for planning
- Team measurement and tracking
- Post-mortems for improvement
- Team application of lean principles

The ability to predict the amount of time required to produce a piece of software is dramatically improved by estimating at the individual level first and then combining the estimates rather than developing a single team estimate averaged over individuals.
Defect Detection at Intuit

Consequently test changes from defect detection into correctness verification.

PSP/TSP shifts defect detection from the test phase back up into development.

- Personal reviews: 33%
- Compile: 33%
- Team reviews: 19%
- Unit test: 15%
- Test: 14%

Fagan (2005)
Dramatic reductions:

- Delivered defects
- Days per KLOC
- Schedule deviation
- Test defects/kloc
- Variation in results

Section 3
Measuring Software Improvement Programs

1) Improvement program methods
2) Maturity-based measurement
3) Improvement program results
CMMI Maturity Transitions

Level 5: Optimizing
- Innovation management

Level 4: Quantitatively Managed
- Capability management

Level 3: Defined
- Process management

Level 2: Managed
- Project management

Level 1: Initial
- Inconsistent management

Chrisis, Konrad, & Shrum (2005). CMMI.
Enhanced Maturity Framework

Level 1
Initial
Inconsistent management

Level 2
Stabilized
Work unit management

Level 3
Standardized
Organizational management

Level 4
Optimized
Capability management

Level 5
Innovating
Innovation management

Innovate to new level of capability
Optimize standardized process
Standard integrated value chain
Control baselines and commitments
Maturity Level Transformation

Level 1
Ad Hoc, inconsistent development practices
Relationship culture

Level 2
Project managers stabilize projects
Local culture

Level 3
Organization develops standard processes
Common culture

Level 4
Process and results managed statistically
Precision culture

Level 5
Proactive improvements
Agile culture

Level 5
Opportunistic improvements
Empowered culture

Organization
Projects
Individual

Trust
Discipline
Measure-maturity mismatch slows improvement and creates dysfunction
## Average Improvement Results

<table>
<thead>
<tr>
<th>Improvement benefit</th>
<th>Orgs</th>
<th>Annual median</th>
<th>Annual range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Productivity growth</td>
<td>4</td>
<td>35%</td>
<td>9% - 67%</td>
</tr>
<tr>
<td>Pre-test defect detection</td>
<td>3</td>
<td>22%</td>
<td>6% - 25%</td>
</tr>
<tr>
<td>Time to market</td>
<td>2</td>
<td>↓ 19%</td>
<td>15% - 23%</td>
</tr>
<tr>
<td>Field error reports</td>
<td>5</td>
<td>↓ 39%</td>
<td>10% - 94%</td>
</tr>
<tr>
<td>Return on investment</td>
<td>5</td>
<td>5.0:1</td>
<td>4:1 - 8.8:1</td>
</tr>
</tbody>
</table>
Schedule Performance

Schedule performance index

CMM maturity level

Air Force study of contractors

Flowe & Thordahl (1994)
Cost Performance

Air Force study of contractors

Flowe & Thordahl (1994)
Crosby’s Cost of Quality Model

**Project Cost**

**Cost of Quality**

**Conformance**
- Appraisal
  - Reviews
    - system
    - requirements
    - design
    - test plan
    - test scripts
  - Walkthroughs
  - Testing (first time)
  - Independent
  - Verification and Validation (first time)
  - Audits
- Prevention
  - Training
  - Policies, Procedures, and methods
  - Tools
  - Planning
  - Quality Improvement Projects
  - Data Gathering and Analysis
  - Root Cause Analysis
  - Quality Reporting

**Nonconformance**
- Fixing Defects
- Reworking any Document
- Re-updating Source Code
- Re-reviews
- Re-tests
- Lab costs for re-tests
- Patches (Internal and External)
- Engineering Changes
- Change Control Boards
- External Failures and fines
- Customer Support
- Help Desks

**Performance**

Generating plans, documentation
Development
- requirements
- design
- code
- integration

Raytheon's Cost of Quality

- **Performance** — *cost of building it right first time*
- **Nonconformance** — *cost of rework*
- **Appraisal** — *cost of testing*
- **Prevention** — *cost of preventing nonconformance*

### Year | Level | Perform | Nonconf. | Appraise | Prevent
--- | --- | --- | --- | --- | ---
1988 | 1 | 34% | 41% | 15% | 7%
1990 | 2 | 55% | 18% | 15% | 12%
1992 | 3 | 66% | 11% | 23% | 
1994 | 4 | 76% | 6% | 18%

Dion (1993), Haley (1995)
Raytheon’s Productivity

Growth relative to 1989 baseline

Lydon (1995)
Raytheon’s Cost Reduction

Reduction relative to 1989 baseline

Lydon (1995)
Raytheon’s Cost Predictability

Actual cost
Budgeted cost

over-run
under-run

Haley (1995)
Section 4
Measuring Quality and Technical Debt

1) Structural quality measurement
2) Technical debt
3) Consortium for IT Software Quality (CISQ)
CMMI’s primary assumption:

“The quality of a software system is governed by quality of the process used to develop it.”

- Watts Humphrey

CMMI’s product quality problem:

1. Assessments do not verify product quality
2. Compliance focus undermines CMMI’s primary assumption
3. Product quality focus at Level 4 in CMM was lost in quantitative process control in CMMI
4. The assumptions underlying control charts are violated by software data
Six Sigma’s Challenge

Six Sigma projects must have significant benefits

- Huge benefits
- Large benefits
- Good benefits
- Okay benefits
- Small benefits
- Now what?

Ultimately we run out of projects with significant enough benefits to continue the program.... so what is the solution that allows continual improvement?

**Process focus** – process improvement – *Six Sigma*

**Product focus** – product improvement – *Design for 6σ*

Product focus supplements product focus to unlock even more value from software.
“As higher levels of assurance are demanded…testing cannot deliver the level of confidence required at a reasonable cost.”

“The correctness of the code is rarely the weakest link.”

“…a failure to satisfy a non-functional requirement can be critical, even catastrophic…non-functional requirements are sometimes difficult to verify. We cannot write a test case to verify a system’s reliability…The ability to associate code to non-functional properties can be a powerful weapon in a software engineer’s arsenal.”
CAST’s Application Intelligence Platform

Language Parsers
- Oracle PL/SQL
- Sybase T-SQL
- SQL Server T-SQL
- IBM SQL/PSM
- C, C++, C#
- Pro C
- Cobol
- CICS
- Visual Basic
- VB.Net
- ASP.Net
- Java, J2EE
- JSP
- XML
- HTML
- Javascript
- VBScript
- PHP
- PowerBuilder
- Oracle Forms
- PeopleSoft
- SAP ABAP, Netweaver
- Tibco
- Business Objects
- Universal Analyzer for other languages

Application Analysis

Detected Violations
- Expensive operation in loop
- Static vs. pooled connections
- Complex query on big table
- Large indices on big table
- Empty CATCH block
- Uncontrolled data access
- Poor memory management
- Opened resource not closed
- SQL injection
- Cross-site scripting
- Buffer overflow
- Uncontrolled format string
- Unstructured code
- Misuse of inheritance
- Lack of comments
- Violated naming convention
- Highly coupled component
- Duplicated code
- Index modified in loop
- High cyclomatic complexity

Quality Measurements
- Performance
- Robustness
- Security
- Transferability
- Changeability

Evaluation of 1200+ coding & architectural rules

Application meta-data

Language Parsers

Oracle PL/SQL
Sybase T-SQL
SQL Server T-SQL
IBM SQL/PSM
C, C++, C#
Pro C
Cobol
CICS
Visual Basic
VB.Net
ASP.Net
Java, J2EE
JSP
XML
HTML
Javascript
VBScript
PHP
PowerBuilder
Oracle Forms
PeopleSoft
SAP ABAP, Netweaver
Tibco
Business Objects
Universal Analyzer for other languages

Application Analysis

Detected Violations
- Expensive operation in loop
- Static vs. pooled connections
- Complex query on big table
- Large indices on big table
- Empty CATCH block
- Uncontrolled data access
- Poor memory management
- Opened resource not closed
- SQL injection
- Cross-site scripting
- Buffer overflow
- Uncontrolled format string
- Unstructured code
- Misuse of inheritance
- Lack of comments
- Violated naming convention
- Highly coupled component
- Duplicated code
- Index modified in loop
- High cyclomatic complexity

Quality Measurements
- Performance
- Robustness
- Security
- Transferability
- Changeability

Evaluation of 1200+ coding & architectural rules

Application meta-data
Modern Apps are a Technology Stack

1. **Unit Level**
   - Code style & layout
   - Expression complexity
   - Code documentation
   - Class or program design
   - Basic coding standards
   - Developer level

2. **Technology Level**
   - Single language/technology layer
   - Intra-technology architecture
   - Intra-layer dependencies
   - Inter-program invocation
   - Security vulnerabilities
   - Development team level

3. **System Level**
   - Integration quality
   - Architectural compliance
   - Risk propagation
   - Application security
   - Resiliency checks
   - Transaction integrity
   - Function point
   - Effort estimation
   - Data access control
   - SDK versioning
   - Calibration across technologies
   - IT organization level

- Transaction Risk
- Data Flow
Architecturally Complex Defects

A structural flaw involving interactions among multiple components that reside in different application layers.

Most architecturally complex defects involve an architectural hotspot – a badly-built component that should be replaced.
The impact of violation 284 in Component B is very widely propagated throughout the system – it presents very high risk.

Violation 342 in component A

The impact of violation 342 in Component A is only narrowly propagated in the system – it presents low risk.

Violation 284 in component B

Propagated Risk Index

A ranking of the risk created by each violation based on its severity and the breadth of its impact in the system.

Remediating violation 284 in Component B will have the greater impact on reducing risk and cost.
- Customers care most about the dependability of their transactions.
- The risk of a transaction is determined by the number and severity of violations along its path.
- Transactions can be ranked by risk to prioritize their remediation.
- The Transaction Risk Index can be further tuned using the operational frequency of each transaction.

**Transaction Risk Index**

A ranking of transactions based on the risk created by the number and severity of violations along the transaction path.
## Risk-Based Prioritization

<table>
<thead>
<tr>
<th>Violation</th>
<th>Health Factor</th>
<th>Severity</th>
<th>Propagated Risk Index</th>
<th>In a high risk transaction?</th>
<th>Frequency of path execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>#148</td>
<td>Security</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#371</td>
<td>Robustness</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#062</td>
<td>Performance</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#284</td>
<td>Robustness</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#016</td>
<td>Changeability</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#241</td>
<td>Security</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#173</td>
<td>Transferability</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#359</td>
<td>Transferability</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Testing finds <1/3 of structural defects

Static analysis before testing is 85% efficient

With testing and static analysis combined, test schedules are reduced by 50%
Reducing Operational Incidents & Costs

Study of structural quality measures and maintenance effort across 20 customers in a large global system integrator

TQI increase of .24 decreased corrective maintenance effort by 50%
Reducing Operational Losses

Large international investment bank
Business critical applications
Factors Affecting Structural Quality

Percentage of variation in structural quality scores accounted for by various factors

<table>
<thead>
<tr>
<th>Factor</th>
<th># apps</th>
<th>Robust</th>
<th>Security</th>
<th>Perform</th>
<th>Change</th>
<th>Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>1606</td>
<td>20%</td>
<td>3%</td>
<td>16%</td>
<td>26%</td>
<td>7%</td>
</tr>
</tbody>
</table>
The Technical Debt Metaphor

**Technical Debt** — the future cost of fixing defects remaining in code at release, a component of the cost of ownership

- **Business Risk**
  - **Opportunity cost** — benefits that could have been achieved had resources been put on new capability rather than retiring technical debt
  - **Liability from debt** — business costs related to outages, breaches, corrupted data, etc.

- **Technical Debt**
  - **Interest on the debt** — continuing IT costs attributable to the violations causing technical debt, i.e., higher maintenance costs, greater resource usage, etc.
  - **Principal borrowed** — cost of fixing problems remaining in the code after release that must be remediated

- **Structural quality problems in production code**

Today’s talk focuses on the principal

70% of Technical Debt is in IT Cost (Transferability, Changeability)

30% of Technical Debt is in Business Risk (Robustness, Performance, Security)

Health Factor proportions are mostly consistent across technologies

Join CISQ! www.it-cisq.org

CISQ
Consortium for IT Software Quality

The Consortium for IT Software Quality (CISQ) is an IT industry leadership group comprised of IT executives from the Global 2000, system integrators, outsourced service providers, and software technology vendors committed to introducing a computable metrics standard for measuring software quality & size. CISQ is a neutral, open forum in which customers and suppliers of IT application software can develop an industry-wide agenda of actions for improving IT application quality to reduce cost and risk.

CISQ Sponsors

ADVANCING THE MEASUREMENT OF SOFTWARE SIZE, QUALITY, AND RISK

Become a sponsor to lend thought leadership
Join CISQ to stay current
References
References — 1


