Software Quality Assurance: V Management

Dr. Holger Giese
Software Engineering Group
Room E 3.165
Tel. 60-3321
Email: hg@upb.de
Outline

I  Introduction
II  Software Life Cycle
III  Quality Control
IV  Infrastructure
V  Management
VI  Standards
VII  Conclusion & Summary
V Management

V.1 Project Progress Control
V.2 Quality Metrics
V.3 Cost of Quality
V.4 Discussion & Summary
V.5 Bibliography
V.1 Project Progress Control

Overview:

- The components of project progress control
- Progress control of internal projects and external participants
- Implementation of project progress control regimes
- Computerized tools for software progress control

[Galin2004]
Components of Project Progress Control

(1) Control of risk management activities
(2) Project schedule control
(3) Project resource control
(4) Project budget control

[Galina2004]
(1) Control of Risk Management Activities

- Initial list of risk items come from the contract reviews and project plan
- Systematic risk management activities required:
  - Periodic assessment about the state of the software risk items
  - Based on this reports the project managers are expected to intervene and help arrive at a solution in the more extreme cases

(see also short excursus on risk management in chapter II) [Galin2004]
(2) Project Schedule Control [Galin2004]

- Compliance of the project with its approved and contracted timetables
- Control is based mainly on milestone reports which are set (in part) to facilitate identification of delays and other periodic reports
- Milestones set in contracts, especially dates for delivery, receive special emphasis
- Focus on critical delays (which may effect final completion of the project)
- Management interventions:
  - Allocation of additional resources
  - Renegotiating the schedule with the customer
(3) Project Resource Control

- Main control items:
  - Human resources
  - Special development and testing equipment (real-time systems; firmware)

- Control is based on periodic resource usage

- True extent of derivations can only be assessed from the point of view of the project progress

- Internal composition of the resource also counts (percentage of senior staff involved, …)

[Galina2004]
Project Budget Control

Main budget items:
- Human resources
- Development and testing facilities
- Purchase of COTS software
- Purchase of hardware
- Payments to subcontractors

Control is based on the milestone reports and other periodic reports.

Usually budget control has the highest priority, but only the combination of all control aspects ensures the required...
Progress control of internal projects and external participants

- **Problem:** In practice project control provides only a limited view of the progress of *internal software development* and an even more limited view on the progress made by *external project participants*.

- **Internal projects** have by definition no external customer and therefore tend to occupy a lower place among management's priorities. Therefore, the full range of project progress control should be employed.

- More significant efforts are required in order to achieve acceptable levels of control for an *external project participant* due to the more complex communication and coordination.
Implementation of project progress control regimes

Procedures:
- Allocation of responsibilities for
  - Person or management for progress control
  - Frequency of reporting from each of the unit levels and administrative level
  - Situation requiring the project leader to report immediately to management
  - Situation requiring lower management to report immediately to upper management
- Management audits of project progress which deals mainly with
  1. How well progress reports are transmitted by project leaders and by lower- to upper-level management
  2. Specific management control activities to be initiated

Remarks:
- Project progress control may be conducted on several managerial levels in large software development organizations ⇒ coordination becomes essential
- Project leaders base there progress reports on information gathered from team leaders
Computerized Project Progress Control

- Required for non trivial projects
- Automation can reduce costs considerably

[Galin2004]
Computerized Control of risk management

- Lists of software risk items by category and their planned solution dates
- Lists of exceptions of software risk items – overrun solution dates

[Galin2004]
Computerized Project Schedule Control

- Classified lists of delayed activities.
- Classified lists of delays of critical activities – delays that can affect the project’s completion date.
- Updated activity schedules, according to progress reports and correction measures.
- Classified lists of delayed milestones.
- Updated milestones schedules, according to progress reports and applied correction measures.

[Galil 2004]
Computerized Project Resource Control

- Project resources allocation plan
  - for activities and software modules
  - for teams and development units
  - for designated time periods, etc.

- Project resources utilization – as specified above

- Project resources utilization exceptions – as specified above

- Updated resource allocation plans generated according to progress reports and reaction measures applied

[Galin2004]
Computerized Project Budget Control

- Project budget plans
  - for activity and software module
  - for teams and development units
  - for designated time periods, etc.

- Project budget utilization reports — as specified above

- Project budget utilization deviations – by period or accumulated – as specified above

- Updated budget plans generated according to progress reports ports and correction measures
V.2 Quality Metrics

“You can’t control what you can’t measure” [DeMarco1982]

Overview:

- Objectives of quality measurement
- Classification of software quality metrics
- Process metrics
- Product metrics
- Implementation of software quality metrics
- Limitations of software metrics
- The function point method

(see also earlier outlook in chapter I-38 GQM: Goal-Question-Metric)
Definition

IEEE definition of software quality metrics:

- A **quantitative measure** of the degree to which an item possesses a given quality attribute.

- A **function** whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given quality attribute.

[Galina2004]
Objectives

- **Facilitate** management control, planning and managerial intervention. Based on:
  - Deviations of actual from planned performance.
  - Deviations of actual timetable and budget performance from planned.

- **Identify** situations for development or maintenance process improvement (preventive or corrective actions). Based on:
  - Accumulation of metrics information regarding the performance of teams, units, etc.

[Galin2004]
Requirements

General requirements
- Relevant
- Valid
- Reliable
- Comprehensive
- Mutually exclusive

Operative requirements
- Easy and simple
- Does not require independent data collection
- Immune to biased interventions by interested parties
Classifications

Classification by phases of software system:
- **Process metrics**: metrics related to the software development process
- **Maintenance metrics**: metrics related to software maintenance (product metrics in [Galin2004])
- **Product metrics**: metrics related to software artifacts

Classification by subjects of measurements
- Quality
- Timetable
- Effectiveness (of error removal and maintenance services)
- Productivity
Software Size/Volume Measures

- **KLOC**: classic metric that measures the size of software by thousands of code lines.

- **Number of function points (NFP)**: a measure of the development resources (human resources) required to develop a program, based on the functionality specified for the software system.

[Galin2004]
## Error Counted Measures

[Galin2004]

<table>
<thead>
<tr>
<th>Error severity class</th>
<th>Number of Errors</th>
<th>Relative Weight</th>
<th>Weighted Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>$D = b \times c$</td>
</tr>
<tr>
<td>low severity</td>
<td>42</td>
<td>1</td>
<td>42</td>
</tr>
<tr>
<td>medium severity</td>
<td>17</td>
<td>3</td>
<td>51</td>
</tr>
<tr>
<td>high severity</td>
<td>11</td>
<td>9</td>
<td>99</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>70</strong></td>
<td>***</td>
<td><strong>192</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NCE</strong></td>
<td><strong>70</strong></td>
<td>***</td>
<td>***</td>
</tr>
<tr>
<td><strong>WCE</strong></td>
<td>***</td>
<td><strong>192</strong></td>
<td></td>
</tr>
</tbody>
</table>

Number of code errors (NCE) vs. weighted number of code errors (WCE)
Process Metrics Categories

- Software process quality metrics
  - Error density metrics
  - Error severity metrics
- Software process timetable metrics
- Software process error removal effectiveness metrics
- Software process productivity metrics

[Galin2004]
## Error Density Metrics

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Calculation formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>CED</td>
<td>Code Error Density</td>
<td>CED = NCE / KLOC</td>
</tr>
<tr>
<td>DED</td>
<td>Development Error Density</td>
<td>DED = NDE / KLOC</td>
</tr>
<tr>
<td>WCED</td>
<td>Weighted Code Error Density</td>
<td>WCED = WCE / KLOC</td>
</tr>
<tr>
<td>WDED</td>
<td>Weighted Development Error Density</td>
<td>WDED = WDE / KLOC</td>
</tr>
<tr>
<td>WCEF</td>
<td>Weighted Code Errors per Function Point</td>
<td>WCEF = WCE / NFP</td>
</tr>
<tr>
<td>WDEF</td>
<td>Weighted Development Errors per Function Point</td>
<td>WDEF = WDE / NFP</td>
</tr>
</tbody>
</table>

- **NCE** = The number of code errors detected by code inspections and testing.
- **NDE** = total number of development (design and code) errors detected in the development process.
- **WCE** = weighted total code errors detected by code inspections and testing.
- **WDE** = total weighted development (design and code) errors detected in development process.
## Error Severity Metrics

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Calculation formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCE</td>
<td>Average Severity of Code Errors</td>
<td>ASCE = \frac{WCE}{NCE}</td>
</tr>
<tr>
<td>DED</td>
<td>Average Severity of Development Errors</td>
<td>ASDE = \frac{WDE}{NDE}</td>
</tr>
</tbody>
</table>

- **NCE** = The number of code errors detected by code inspections and testing.
- **NDE** = Total number of development (design and code) errors detected in the development process.
- **WCE** = Weighted total code errors detected by code inspections and testing.
- **WDE** = Total weighted development (design and code) errors detected in development process.
## Software Process Timetable Metrics

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Calculation formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTO</td>
<td>Time Table Observance</td>
<td>$\text{TTO} = \frac{\text{MSOT}}{\text{MS}}$</td>
</tr>
<tr>
<td>ADMC</td>
<td>Average Delay of Milestone Completion</td>
<td>$\text{ADMC} = \frac{\text{TCDAM}}{\text{MS}}$</td>
</tr>
</tbody>
</table>

- **MSOT** = Milestones completed on time.
- **MS** = Total number of milestones.
- **TCDAM** = Total Completion Delays (days, weeks, etc.) for all milestones.

[Galin2004]
## Error Removal Effectiveness Metrics

[Galin2004]

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Calculation formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>DERE</td>
<td>Development Errors Removal Effectiveness</td>
<td>( \text{DERE} = \frac{\text{NDE}}{\text{NDE} + \text{NYF}} )</td>
</tr>
<tr>
<td>DWERE</td>
<td>Development Weighted Errors Removal Effectiveness</td>
<td>( \text{DWERE} = \frac{\text{WDE}}{\text{WDE} + \text{WYF}} )</td>
</tr>
</tbody>
</table>

NDE = total number of development (design and code) errors) detected in the development process.
WCE = weighted total code errors detected by code inspections and testing.
WDE = total weighted development (design and code) errors detected in the development process.
NYF = number software failures detected during a year of maintenance service.
WYF = weighted number of software failures detected during a year of maintenance service.
# Process Productivity Metrics

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Calculation formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevP</td>
<td>Development Productivity</td>
<td>DevP = ( \frac{\text{DevH}}{\text{KLOC}} )</td>
</tr>
<tr>
<td>FDevP</td>
<td>Function point Development Productivity</td>
<td>FDevP = ( \frac{\text{DevH}}{\text{NFP}} )</td>
</tr>
<tr>
<td>CRe</td>
<td>Code Reuse</td>
<td>Cre = ( \frac{\text{ReKLOC}}{\text{KLOC}} )</td>
</tr>
<tr>
<td>DocRe</td>
<td>Documentation Reuse</td>
<td>DocRe = ( \frac{\text{ReDoc}}{\text{NDoc}} )</td>
</tr>
</tbody>
</table>

DevH = Total working hours invested in the development of the software system.
ReKLOC = Number of thousands of reused lines of code.
ReDoc = Number of reused pages of documentation.
NDoc = Number of pages of documentation.

[Galin2004]
Maintenance Metrics Categories (1/2)

Help desk service (HD):
software support by instructing customers regarding the method of application of the software and solution for customer implementation problems (depends to a great extent on “user friendliness”)

Related metrics:
- HD quality metrics:
  - HD calls density metrics - measured by the number of calls.
  - HD calls severity metrics - the severity of the HD issues raised.
  - HD success metrics – the level of success in responding to HD calls.
- HD productivity metrics.
- HD effectiveness metrics.

[Galin2004]
Product Metrics Categories (2/2)

Corrective maintenance:
Correction of software failures identified by customers/users or detected by the customer service team prior to their discovery by the customer (directly related to the software development quality)

Related metrics:
- Corrective maintenance quality metrics.
  - Software system failures density metrics
  - Software system failures severity metrics
  - Failures of maintenance services metrics
  - Software system availability metrics
- Corrective maintenance productivity metrics
- Corrective maintenance effectiveness metrics

[Galit2004]
# HD Calls Density Metrics

[Galin2004]

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Calculation formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDD</td>
<td>HD calls density</td>
<td>HDD = \frac{\text{NHYC}}{\text{KLMC}}</td>
</tr>
<tr>
<td>WHDD</td>
<td>Weighted HD calls density</td>
<td>WHYC = \frac{\text{WHYC}}{\text{KLMC}}</td>
</tr>
<tr>
<td>WHDF</td>
<td>Weighted HD calls per function point</td>
<td>WHDF = \frac{\text{WHYC}}{\text{NMFP}}</td>
</tr>
</tbody>
</table>

- \text{NHYC} = the number of HD calls during a year of service.
- \text{KLMC} = Thousands of lines of maintained software code.
- \text{WHYC} = weighted HD calls received during one year of service.
- \text{NMFP} = number of function points to be maintained.
Severity of HD Calls Metrics and HD Success Metrics

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Calculation formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASHC</td>
<td>Average severity of HD calls</td>
<td>ASHC = ( \frac{WHYC}{NHYC} )</td>
</tr>
</tbody>
</table>

\( NHYC \) = the number of HD calls during a year of service.

\( WHYC \) = weighted HD calls received during one year of service.

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Calculation formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDS</td>
<td>HD service success</td>
<td>HDS = ( \frac{NHYOT}{NHYC} )</td>
</tr>
</tbody>
</table>

\( NHYOT \) = Number of yearly HD calls completed on time during one year of service.

\( NHYC \) = the number of HD calls during a year of service.
# HD Productivity and Effectiveness Metrics

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Calculation formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDP</td>
<td>HD Productivity</td>
<td>HDP = \frac{HDYH}{KLNC}</td>
</tr>
<tr>
<td>FHDP</td>
<td>Function Point HD Productivity</td>
<td>FHDP = \frac{HDYH}{NMFP}</td>
</tr>
<tr>
<td>HDE</td>
<td>HD effectiveness</td>
<td>HDE = \frac{HDYH}{NHYC}</td>
</tr>
</tbody>
</table>

HDYH = Total yearly working hours invested in HD servicing of the software system.
KLMC = Thousands of lines of maintained software code.
NMFP = number of function points to be maintained.
NHYC = the number of HD calls during a year of service.
Failures Density Metrics

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Calculation formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSFD</td>
<td>Software System Failure Density</td>
<td>SSFD = NYF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>KLMC</td>
</tr>
<tr>
<td>WSSFD</td>
<td>Weighted Software System Failure Density</td>
<td>WFFFD = WYF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>KLMC</td>
</tr>
<tr>
<td>WSSFF</td>
<td>Weighted Software System Failures per Function point</td>
<td>WSSFF = WYF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NMFP</td>
</tr>
</tbody>
</table>

NYF = number of software failures detected during a year of maintenance service.
WYF = weighted number of yearly software failures detected during one year of maintenance service.
NMFP = number of function points designated for the maintained software.
KLMC = Thousands of lines of maintained software code.
## Failure Severity and Failures of Maintenance Services Metrics

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Calculation formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSSF</td>
<td>Average Severity of Software System Failures</td>
<td>[ ASSSF = \frac{\text{WYF}}{\text{NYF}} ]</td>
</tr>
</tbody>
</table>

- **NYF** = number of software failures detected during a year of maintenance service.
- **WYF** = weighted number of yearly software failures detected during one year.

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Calculation formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRepF</td>
<td>Maintenance Repeated repair Failure metric -</td>
<td>[ MRepF = \frac{\text{RepYF}}{\text{NYF}} ]</td>
</tr>
</tbody>
</table>

- **NYF** = number of software failures detected during a year of maintenance service.
- **RepYF** = Number of repeated software failure calls (service failures).

---

[Galin2004]
# Availability Metrics

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Calculation formula</th>
</tr>
</thead>
</table>
| FA   | Full Availability  | \[
|      | \text{FA} = \frac{\text{NYSerH} - \text{NYFH}}{\text{NYSerH}}\] |
| VitA | Vital Availability | \[
|      | \text{VitA} = \frac{\text{NYSerH} - \text{NYVitFH}}{\text{NYSerH}}\] |
| TUA  | Total Unavailability| \[
|      | \text{TUA} = \frac{\text{NYTFH}}{\text{NYSerH}}\] |

\(\text{NYSerH}\) = Number of hours software system is in service during one year.  
\(\text{NYFH}\) = Number of hours where at least one function is unavailable (failed) during one year, including total failure of the software system.  
\(\text{NYVitFH}\) = Number of hours when at least one vital function is unavailable (failed) during one year, including total failure of the software system.  
\(\text{NYTFH}\) = Number of hours of total failure (all system functions failed) during one year.  

\text{Note: } \text{NYFH} \geq \text{NYVitFH} \geq \text{NYTFH} \text{ and } 1 - \text{TUA} \geq \text{VitA} \geq \text{FA}
Software Corrective Maintenance
Productivity and Effectiveness Metrics

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Calculation formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMaiP</td>
<td>Corrective Maintenance Productivity</td>
<td>CMaiP = ( \frac{CMaiYH}{KLMC} )</td>
</tr>
<tr>
<td>FCMP</td>
<td>Function point Corrective Maintenance Productivity</td>
<td>FCMP = ( \frac{CMaiYH}{NMFP} )</td>
</tr>
<tr>
<td>CMaiE</td>
<td>Corrective Maintenance Effectiveness</td>
<td>CMaiE = ( \frac{CMaiYH}{NYF} )</td>
</tr>
</tbody>
</table>

- **CMaiYH** = Total yearly working hours invested in the corrective maintenance of the software system.
- **NYF** = number of software failures detected during a year of maintenance service.
- **NMFP** = number of function points designated for the maintained software.
- **KLMC** = Thousands of lines of maintained software code.
Product Metrics Categories

- **Product metrics** can also be used for general predictions or to identify anomalous components.

- Classes of product metric
  - **Dynamic metrics** which are collected by measurements made of a program in execution;
  - **Static metrics** which are collected by measurements made of the system representations;

- Dynamic metrics help assess efficiency and reliability; static metrics help assess complexity, understandability and maintainability.

[Sommerville2004]
Static Metrics

Static metrics have an indirect relationship with quality attributes (see also static analysis)

- Maintainability
- Reliability
- Portability
- Usability

- Number of procedure parameters
- Cyclomatic complexity
- KLOC
- Number of error messages
- Length of the user manual

You need to try and derive a relationship between these metrics and properties such as complexity, understandability, and maintainability.

[Sommerville2004]
## Software Product Metrics

<table>
<thead>
<tr>
<th>Software metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fan in/Fan-out</td>
<td>Fan-in is a measure of the number of functions or methods that call some other function or method (say X). Fan-out is the number of functions that are called by function X. A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive knock-on effects. A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components.</td>
</tr>
<tr>
<td>Length of code</td>
<td>This is a measure of the size of a program. Generally, the larger the size of the code of a component, the more complex and error-prone that component is likely to be. Length of code has been shown to be one of the most reliable metrics for predicting error-proneness in components.</td>
</tr>
<tr>
<td>Cyclomatic complexity</td>
<td>This is a measure of the control complexity of a program. This control complexity may be related to program understandability. I discuss how to compute cyclomatic complexity in Chapter 22.</td>
</tr>
<tr>
<td>Length of identifiers</td>
<td>This is a measure of the average length of distinct identifiers in a program. The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.</td>
</tr>
<tr>
<td>Depth of conditional nesting</td>
<td>This is a measure of the depth of nesting of if-statements in a program. Deeply nested if statements are hard to understand and are potentially error-prone.</td>
</tr>
<tr>
<td>Fog index</td>
<td>This is a measure of the average length of words and sentences in documents. The higher the value for the Fog index, the more difficult the document is to understand.</td>
</tr>
</tbody>
</table>
## Object-oriented Metrics

<table>
<thead>
<tr>
<th>Object-oriented metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth of inheritance tree</td>
<td>This represents the number of discrete levels in the inheritance tree where sub-classes inherit attributes and operations (methods) from super-classes. The deeper the inheritance tree, the more complex the design. Many different object classes may have to be understood to understand the object classes at the leaves of the tree.</td>
</tr>
<tr>
<td>Method fan-in/fan-out</td>
<td>This is directly related to fan-in and fan-out as described above and means essentially the same thing. However, it may be appropriate to make a distinction between calls from other methods within the object and calls from external methods.</td>
</tr>
<tr>
<td>Weighted methods per class</td>
<td>This is the number of methods that are included in a class weighted by the complexity of each method. Therefore, a simple method may have a complexity of 1 and a large and complex method a much higher value. The larger the value for this metric, the more complex the object class. Complex objects are more likely to be more difficult to understand. They may not be logically cohesive so cannot be reused effectively as super-classes in an inheritance tree.</td>
</tr>
<tr>
<td>Number of overriding operations</td>
<td>This is the number of operations in a super-class that are over-ridden in a sub-class. A high value for this metric indicates that the super-class used may not be an appropriate parent for the sub-class.</td>
</tr>
</tbody>
</table>
The Measurement Process

A software measurement process may be part of a quality control process.

1. Define software quality metrics
2. Select components to be assessed
3. Data collection
4. Identify anomalous measurement
5. Analyse anomalous measurement

Data collected during this process should be maintained as an organisational resource.

Once a measurement database has been established, comparisons across projects become possible.

[Sommerville2004]
(1) Defining Software Quality Metrics

- Define an attribute to be measured
- Define the metrics that measure the attributes
- Determine comparative target values (indicators)
- Define method of reporting and metrics data collection
- Software quality metrics procedures and work instructions
- Application of software quality metrics
- Metrics data collection
- Metrics data for managerial control applications

Developments in the organization and its environment

Changes of metrics
Changes in comparative target values (indicators)
Changes in metrics data collection
Analysis of metrics performance and effects of environmental changes

Metrics data for analysis of metrics' performance

[Galin2004]
(3) Data Collection

- A metrics programme should be based on a set of product and process data.
- Data should be collected *immediately* (not in retrospect) and, if possible, *automatically*.
- Three types of automatic data collection
  - Static product analysis;
  - Dynamic product analysis;
  - Process data collation.

[Sommerville2004]
Data Accuracy

- Don’t collect unnecessary data
  - The questions to be answered should be decided in advance and the required data identified.

- Tell people why the data is being collected.
  - It should not be part of personnel evaluation.

- Don’t rely on memory
  - Collect data when it is generated not after a project has finished.

[Sommerville2004]
(5) Measurement Analysis [Sommerville2004]

- It is not always obvious what data means
  - Analysing collected data is very difficult.
- Professional statisticians should be consulted if available.
- Data analysis must take local circumstances into account.
Measurement Surprises [Sommerville2004]

Reducing the number of faults in a program leads to an increased number of help desk calls

- The program is now thought of as more reliable and so has a wider more diverse market. The percentage of users who call the help desk may have decreased but the total may increase;

- A more reliable system is used in a different way from a system where users work around the faults. This leads to more help desk calls.
Limitations of Quality Metrics

- **Budget** constraints in allocating the necessary resources.
- **Human factors**, especially opposition of employees to evaluation of their activities.
- **Validity** Uncertainty regarding the data's, partial and biased reporting.

Metrics assumptions:
- A software property can be measured.
- The relationship exists between what we can measure and what we want to know. We can only measure internal attributes but are often more interested in external software attributes.
- This relationship has been formalised and validated.
- It may be difficult to relate what can be measured to desirable external quality attributes.

"Not everything that counts is countable; and not everything that is countable counts."
Examples of Software Metrics that exhibit Severe Weaknesses

- Parameters used in development process metrics:
  - KLOC, NDE, NCE.

- Parameters used in product (maintenance) metrics:
  - KLMC, NHYC, NYF.

[Galin2004]
Factors Affecting Parameters used for Development Process Metrics

a. Programming style (KLOC).
b. Volume of documentation comments (KLOC).
c. Software complexity (KLOC, NCE).
d. Percentage of reused code (NDE, NCE).
e. Professionalism and thoroughness of design review and software testing teams: affects the number of defects detected (NCE).
f. Reporting style of the review and testing results: concise reports vs. comprehensive reports (NDE, NCE).

[Galin2004]
Factors Affecting Parameters used for Maintenance Metrics

a. Quality of installed software and its documentation (NYF, NHYC).

b. Programming style and volume of documentation comments included in the code be maintained (KLMC).

c. Software complexity (NYF).

d. Percentage of reused code (NYF).

e. Number of installations, size of the user population and level of applications in use: (NHYC, NYF).

[Galvin2004]
Factors Affecting Parameters used for Product Metrics

a. Programming style
b. Architectural styles/Architectures

c. Thorough understanding of the domain (more intensive use of inheritance)

d. Level of reuse and use of frameworks
The Function Point Method

The function point estimation process:

- **Stage 1:** Compute crude function points (CFP).
- **Stage 2:** Compute the relative complexity adjustment factor (RCAF) for the project. RCAF varies between 0 and 70.
- **Stage 3:** Compute the number of function points (FP):
  \[ FP = CFP \times (0.65 + 0.01 \times RCAF) \]
### Crude Function Points: Calculation

<table>
<thead>
<tr>
<th>Software system components</th>
<th>Complexity level</th>
<th>Total CFP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Simple</td>
<td>average</td>
</tr>
<tr>
<td><strong>Count</strong></td>
<td><strong>Weight Factor</strong></td>
<td><strong>Points</strong></td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>C=AxB</td>
</tr>
<tr>
<td>User inputs</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>User outputs</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>User online queries</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Logical files</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>External interfaces</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**Total CFP**

---

Additional Information

Short excursus: function point method (2/8)

[Galin2004]
# Relative Complexity Adjustment Factor

<table>
<thead>
<tr>
<th>No</th>
<th>Subject</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Requirement for reliable backup and recovery</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>2</td>
<td>Requirement for data communication</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>3</td>
<td>Extent of distributed processing</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>4</td>
<td>Performance requirements</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>5</td>
<td>Expected operational environment</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>6</td>
<td>Extent of online data entries</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>7</td>
<td>Extent of multi-screen or multi-operation online data input</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>8</td>
<td>Extent of online updating of master files</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>9</td>
<td>Extent of complex inputs, outputs, online queries and files</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>10</td>
<td>Extent of complex data processing</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>11</td>
<td>Extent that currently developed code can be designed for reuse</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>12</td>
<td>Extent of conversion and installation included in the design</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>13</td>
<td>Extent of multiple installations in an organization and variety of customer organizations</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>14</td>
<td>Extent of change and focus on ease of use</td>
<td>0 1 2 3 4 5</td>
</tr>
</tbody>
</table>

**Total = RCAF**

*Short excursus: function point method (3/8)*
Example: ATTEND MASTER

ATTEND MASTER is a basic employee attendance system for small to medium-sized businesses.

Data Flow Diagram:
### Example: CFP

<table>
<thead>
<tr>
<th>Software system components</th>
<th>Simple</th>
<th>average</th>
<th>complex</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Count</td>
<td>Weight Factor</td>
<td>Points</td>
<td>Count</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User inputs</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>---</td>
</tr>
<tr>
<td>User outputs</td>
<td>---</td>
<td>4</td>
<td>---</td>
<td>2</td>
</tr>
<tr>
<td>User online queries</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Logical files</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>---</td>
</tr>
<tr>
<td>External interfaces</td>
<td>---</td>
<td>5</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Total CFP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Example: RCAF

<table>
<thead>
<tr>
<th>No.</th>
<th>Subject</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Requirement for reliable backup and recovery</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>2</td>
<td>Requirement for data communication</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>3</td>
<td>Extent of distributed processing</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>4</td>
<td>Performance requirements</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>5</td>
<td>Expected operational environment</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>6</td>
<td>Extent of online data entries</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>7</td>
<td>Extent of multi-screen or multi-operation online data input</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>8</td>
<td>Extent of online updating of master files</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>9</td>
<td>Extent of complex inputs, outputs, online queries and files</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>10</td>
<td>Extent of complex data processing</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>11</td>
<td>Extent that currently developed code can be designed for reuse</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>12</td>
<td>Extent of conversion and installation included in the design</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>13</td>
<td>Extent of multiple installations in an organization and variety of customer organizations</td>
<td>0 1 2 3 4 5</td>
</tr>
<tr>
<td>14</td>
<td>Extent of change and focus on ease of use</td>
<td>0 1 2 3 4 5</td>
</tr>
</tbody>
</table>

**Total = RCAF**

41
Example: Final Calculation

FP = CFP x (0.65 + 0.01 x RCAF)

FP = 81 x (0.65 + 0.01 x 41) = 85.86
Advantages & Disadvantages

Main advantages
- Estimates can be prepared at the pre-project stage.
- Based on requirement specification documents (not specific dependent on development tools or programming languages), the method’s reliability is relatively high.

Main disadvantages
- FP results depend on the counting instruction manual.
- Estimates based on detailed requirements specifications, which are not always available.
- The entire process requires an experienced function point team and substantial resources.
- The evaluations required result in subjective results.
- Successful applications are related to data processing. The method cannot yet be universally applied.

[Galin2004]
V.3 Cost of Quality

Overview:

- Objectives of cost of software quality metrics
- The classic model of cost of software quality
- Galin’s extended model for cost of software quality
- Application of a cost of software quality system
- Problems in the application of cost of software quality metrics
Objectives

In general – it enables management to achieve **economic control** over SQA activities and outcomes. The specific objectives are:

- **Control organization-initiated costs** to prevent and detect software errors.
- Evaluation of the **economic damages** of software failures as a basis for revising the SQA budget.
- Evaluation of **plans to increase or decrease** of SQA activities or to invest in SQA infrastructure on the basis of past economic performance.
Performance Comparisons

- Control Budgeted expenditures (for SQA prevention and appraisal activities).
- Previous year’s failure costs
- Previous project’s quality costs (control costs and failure costs).
- Other department’s quality costs (control costs and failure costs).
Evaluating SQA Systems

Cost metrics examples:

- Percentage of cost of software quality out of total software development costs.
- Percentage of software failure costs out of total software development costs.
- Percentage of cost of software quality out of total software maintenance costs.
- Percentage of cost of software quality out of total sales of software products and software maintenance.

[Galin2004]
Model of Software Quality Costs

Cost of software quality

- Costs of Control costs
  - Prevention costs
  - Appraisal costs

- Costs of Failure of control costs
  - Internal failure costs
  - External failure costs

[Galina2004]
Prevention Costs

- **Investments in development of SQA infrastructure components**
  - Procedures and work instructions
  - Support devices: templates, checklists etc
  - Software configuration management system
  - Software quality metrics

- **Regular implementation of SQA preventive activities:**
  - Instruction of new employees in SQA subjects
  - Certification of employees
  - Consultations on SQA issues to team leaders and others

- **Control of the SQA system through performance of:**
  - Internal quality reviews
  - External quality audits
  - Management quality reviews
Appraisal Costs

Costs of reviews:
- Formal design reviews (DRs)
- Peer reviews (inspections and walkthroughs)
- Expert reviews

Costs of software testing:
- Unit, integration and software system tests
- Acceptance tests (carried out by customers)

Costs of assuring quality of external participants

[Galin2004]
Internal Failure Costs

- Costs of redesign or design corrections subsequent to design review and test findings
- Costs of re-programming or correcting programs in response to test findings
- Costs of repeated design review and re-testing (regression tests)

[Galin2004]
External Failure Costs

Typical external failure costs cover:

- Resolution of customer complaints during the warranty period.
- Correction of software bugs detected during regular operation.
- Correction of software failures after the warranty period is over even if the correction is not covered by the warranty.
- Damages paid to customers in case of a severe software failure.
- Reimbursement of customer's purchase costs.
- Insurance against customer's claims.

Typical examples of hidden external failure costs:

- Reduction of sales to customers that suffered from software failures.
- Severe reduction of sales motivated by the firm's damaged reputation.
- Increased investment in sales promotion to counter the effects of past software failures.
- Reduced prospects to win a tender or, alternatively, the need to underprice to prevent competitors from winning tenders.
Galin’s Extended Model of Software Quality Costs

- Cost of software quality
  - Costs of Control costs
    - Prevention costs
    - Appraisal costs
    - Managerial preparations and control costs
  - Costs of Failure of control costs
    - Internal failure costs
    - External failure costs
    - Managerial failure costs
Managerial Preparation and Control Costs

- Costs of carrying out contract reviews
- Costs of preparing project plans, including quality plans
- Costs of periodic updating of project and quality plans
- Costs of performing regular progress control
- Costs of performing regular progress control of external participants’ contributions to projects

[Galin2004]
Managerial Failure Costs

- **Unplanned costs** for professional and other resources, resulting from underestimation of the resources in the proposals stage.

- **Damages** paid to customers as compensation for late project completion, a result of the *unrealistic schedule* in the Company’s proposal.

- **Damages** paid to customers as compensation for late completion of the project, a result of management’s *failure to recruit* team members.

- **Domino effect**: Damages to other projects planned to be performed by the same teams involved in the delayed projects. The domino effect may induce considerable hidden external failure costs.

[Galín2004]
Application

- Definition of a cost of software quality model and specification of cost items.
- Definition of the method of data collection for each cost item.
- Application of a cost of software quality system, including thorough follow up.
- Actions taken in response to the findings.

[Galin2004]
Cost of Software Quality Balance by Quality Level

Minimal total cost of software quality

Total control costs

Total failure of control costs

Optimal software quality level

Quality costs

Software quality level
Problems of Application

**General problems:**
- Inaccurate and/or incomplete identification and classification of quality costs.
- Negligent reporting by team members
- Biased reporting of software costs, especially of “censored” internal and external costs.
- Biased recording of external failure costs - “camouflaged” compensation of customers for failures.

**Problems arising when collecting data on managerial costs:**
- Contract review and progress control activities are performed in a “part-time mode”. The reporting of time invested is usually inaccurate and often neglected.
- Many participants in these activities are senior staff members who are not required to report use of their time resources.
- Difficulties in determination of responsibility for schedule failures.
- Payment of overt and formal compensation usually occurs quite some time after the project is completed, and much too late for efficient application of the lessons learned.

[Galín2004]
V.4 Discussion & Summary

- **Components of Project Progress Control**
  requires (1) control of risk management activities, (2) project schedule control, (3) project resource control, and (4) project budget control. Even though budget control has usually the highest priority, only the combination of all control tasks ensures the required coverage of risks.

- **Metrics** facilitate management control, planning and managerial intervention based on deviations of actual from planned performance to identify situations for development or maintenance process improvement (preventive or corrective actions).
Discussion & Summary

- Metrics must be relevant, valid, reliable, comprehensive, and mutually exclusive. In addition, it must be easy and simple, should not require independent data collection, and should be immune to biased interventions by interested parties.

- Available metrics for control are **process metrics**, which are related to the software development process, **maintenance metrics**, which are related to software maintenance (called product metrics in [Galin2004]), and **product metrics**, which are related to software artifacts.

- Metrics are used to measure **quality**, accordance with **timetables**, **effectiveness** (of error removal and maintenance services), and **productivity**.
Discussion & Summary

- A software measurement process may should be part of a quality control process which (1) defines the employed software quality metrics, (2) selects components to be assessed, (3) collects the data, (4) ensures that anomalous measurements are identified, and (5) are analyzed. But due to the complex interdependencies unexpected effects are possible (measurement surprises).

- The optimal software quality level is reached when the sum of the total failure of control costs and total control costs is minimal.
V.5 Bibliography


