Towards a Catalog Format for Software Metrics

Eric Bouwers, Arie van Deursen, Joost Visser

Presented at the 5th International Workshop on Emerging Trends in Software Metrics (WeTSOM 2014) held at ICSE 2014, Hyderabad, India
Collect detailed *technical findings* about software-intensive systems

Translate into *actionable information* for *high-level management*

Using methods from academic and self-funded *research*
Metrics in Context: 
Software Risk Assessments

System Supplier
System Client
SRA Client

Sessions
Final Report
system

SRA Consultant
SRA Analyst

Provides
Uses
Receives

Participates
Participates

Writes
Contributes to
Analyzes

ICSM 2009
Metrics in Context: Continuous Monitoring & Rating
Software Engineering Metrics: What Do They Measure and How Do We Know?

Cem Kaner, Senior Member, IEEE, and Walter P. Bond

Abstract—Construct validity is about the question, how we know that we’re measuring the attribute that we think we’re measuring? This is discussed in formal, theoretical ways in the computing literature (in terms of the representational theory of measurement) but rarely in simpler ways that foster application by practitioners. Construct validity starts with a thorough analysis of the construct, the attribute we are attempting to measure. In the IEEE Standard 1061, direct measures need not be validated. "Direct" measurement of an attribute involves a metric that depends only on the value of the attribute, but few or no software engineering attributes or tasks are so simple that measures of them can be direct. Thus, all metrics should be validated. The paper continues with a framework for evaluating proposed metrics, and applies it to two uses of bug counts. Bug counts capture only a small part of the meaning of the attributes they are being used to measure. Multidimensional analyses of attributes appear promising as a means of capturing the quality of the attribute in question. Analysis fragments run throughout the paper, illustrating the breakdown of an attribute or task of interest into sub-attributes for grouped studies.


1 INTRODUCTION

We hear too often that few companies establish measurement programs, that fewer succeed with them, or that many of the companies who have established metrics programs have them in order to conform to criteria established in the Capability Maturity Model. [1]

One could interpret this as evidence of the immaturity and unprofessionalism of the field or of resistance to the high cost of metrics programs (Fenton [1] estimates a cost of 4% of the development budget). In some cases, these explanations are undoubtedly correct. In other cases, however, metrics programs are resisted or rejected because they do more harm than good.

Robert Austin [2] provided an excellent discussion of the problems of measurement distortion and dysfunction in general. In this paper, we explore one aspect of the problem of dysfunction. We assert that Software Engineering as a field presents an approach to measurement that underemphasizes measurement validity (the condition that the measurement actually measures the attribute in question). This has a likely consequence: if a project or company is managed according to the results of measurements, and those metrics are inadequately validated, insufficiently understood, and not tightly linked to the attributes they are intended to measure, measurement distortions and dysfunctional should be commonplace.

After justifying our basic assertion, we lay out a model for evaluating the validity and risk of a metric, and apply it to a few metrics common in the field. Not surprisingly (given our main assertion), serious problems will show up.

In the final section of this paper, we suggest a different approach: the use of multidimensional evaluation to obtain measurement of an attribute of interest. The idea of multidimensional analysis is far from new [3], but we will provide detailed examples that appear to have been used effectively at the line manager level, in the field. A pattern of usability and utility emerges from these examples that, we hope, could stimulate further practical application.

2 WHAT ARE WE MEASURING?

2.1 Defining Measurement

To provide context for the next two sections, we need a definition of measurement. To keep the measurement definitions in one place, we present several current definitions here. We will distinguish between them later.

• “Measurement is the assignment of numbers to objects or events according to rule. [4] The rule of assignment can be any consistent rule. The only rule not allowed would be random assignment, for randomness amounts in effect to a nounrule.” [5, p. 47]
• “Measurement is the process of empirical, objective, assignment of numbers to properties of objects or events of the real world in such a way as to describe them.” [6, p. 6]
• “Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to characterize them according to clearly defined rules.” [7, p. 5]
• Measurement is “the act or process of assigning a number or category to an entity to describe an attribute of that entity.” [8, p. 2]
Ten Questions to Ask when Evaluating a Metric

1. Purpose?
2. Scope?
3. Which attribute?
4. Scale of attribute?
5. Variability of attribute?
6. Metric definition?
7. Scale of Metric?
8. Variability of instrument for metric?
9. Relation attribute and metric value?
10. Natural / foreseeable side effects of use?
# Proposed Catalog Format for Software Metrics

<table>
<thead>
<tr>
<th>Name</th>
<th>full name</th>
<th>(abbreviation)</th>
<th>Level</th>
<th>Base/Derived</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity</td>
<td>what is the subject of the measurement?</td>
<td></td>
<td>Type</td>
<td>Internal/External</td>
</tr>
<tr>
<td>Attribute</td>
<td>what characteristic does it quantify?</td>
<td></td>
<td>Range</td>
<td>...</td>
</tr>
<tr>
<td>Definition</td>
<td>what is the measurement procedure?</td>
<td></td>
<td>Expected value</td>
<td>...</td>
</tr>
<tr>
<td>Rationale (theoretical)</td>
<td>why does this metric quantify the defined attribute of the entity?</td>
<td>Variability</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Implications (practical)</td>
<td>Definition of undesired metric values and explanation of the implications of these undesired metrics values, e.g. why does the metric matter?</td>
<td>Scale type</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Applicable in context</td>
<td>When is this metric useful?</td>
<td>Related metrics</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Solution Strategies</th>
<th>Solution Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;solution1&gt;</td>
<td>Treating/solving</td>
</tr>
<tr>
<td>&lt;solution2&gt;</td>
<td>Treating/solving</td>
</tr>
</tbody>
</table>
## Main Page

Welcome to the software metrics wiki! Our goal is to collect and categorize all Software Metrics to provide you with a one-place stop to:

- Find software metrics you can use in your project
- Identify common solutions to problems you find with software metrics
- Get to know new software metrics

All software metrics in this catalog are described in a catalog format which was first introduced in this proposal. Since this introduction, we have been working on setting up the basic structure of the wiki and adding descriptions of software metrics. The following shows a selection of the software metrics that have already been added:

<table>
<thead>
<tr>
<th>Component Size Uniformity</th>
<th>Measures</th>
<th>Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of Code</td>
<td></td>
<td>Software System</td>
</tr>
<tr>
<td>System Breakdown Optimality</td>
<td></td>
<td>Software System</td>
</tr>
<tr>
<td>Duplication Percentage</td>
<td></td>
<td>Software System</td>
</tr>
<tr>
<td>Component Balance</td>
<td></td>
<td>Software System</td>
</tr>
<tr>
<td>Duplicated Lines</td>
<td></td>
<td>Software System</td>
</tr>
<tr>
<td>Project Scope Prognosis</td>
<td></td>
<td>Software Project</td>
</tr>
<tr>
<td>Project Size</td>
<td></td>
<td>Software Project</td>
</tr>
<tr>
<td>Estimation Shift</td>
<td></td>
<td>Software Project</td>
</tr>
<tr>
<td>Enhancement Rate</td>
<td></td>
<td>Project Iteration</td>
</tr>
<tr>
<td>Changed Product Backlog Items</td>
<td></td>
<td>Product Backlog</td>
</tr>
<tr>
<td>Added Removed Product Backlog Items</td>
<td></td>
<td>Product Backlog</td>
</tr>
<tr>
<td>Statement Coverage</td>
<td></td>
<td>Test Suite</td>
</tr>
</tbody>
</table>
Example: Lines of Code for Volume

Lines of Code

(Redirected from Lines Of Code)

Contents [hide]
1 Definition
2 Rationale
3 Implications
4 Applicable in Context
5 Solution strategies
6 Related Metrics
7 Validation

Definition [edit]
Count all the lines in a Software System which do not solely consist of comments and/or whitespace.

Rationale [edit]
Every line that is needed to parse/compile the Software System contributes to the volume of the

<table>
<thead>
<tr>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Abbreviation</td>
</tr>
<tr>
<td>Entity</td>
</tr>
<tr>
<td>Attribute</td>
</tr>
<tr>
<td>Level</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Range</td>
</tr>
<tr>
<td>Expected Value</td>
</tr>
<tr>
<td>Variability</td>
</tr>
<tr>
<td>Scale Type</td>
</tr>
</tbody>
</table>
Implications

A larger Lines of Code leads to higher maintenance costs, lower Analyzability and lower Testability.

Applicable in Context

For any Software System which needs to be maintained.

Solution strategies

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Solution Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove newlines, effectively placing all code on a single line.</td>
<td>Treating</td>
</tr>
<tr>
<td>Introduce abstractions, which can factor out common code.</td>
<td>Solving</td>
</tr>
<tr>
<td>Use libraries to solve common problems.</td>
<td>Solving</td>
</tr>
</tbody>
</table>

Related Metrics

<table>
<thead>
<tr>
<th>Metric name</th>
<th>type of relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duplicated Lines</td>
<td>Positive Correlation</td>
</tr>
<tr>
<td>Duplication Percentage</td>
<td>Normalization Factor</td>
</tr>
<tr>
<td>Function Points</td>
<td>Positive Correlation</td>
</tr>
</tbody>
</table>

Validation

<table>
<thead>
<tr>
<th>Study</th>
<th>Validation type</th>
</tr>
</thead>
</table>

Category: SoftwareMetric
Definition
Divide the Duplicated Lines of a Software System by its Lines of Code

Rationale
The normalized lines of duplication indicate which percentage of the system consists of Duplicated Lines

Implications
A high number of duplicated lines is an indication of bad design.

Applicable in Context
For any Software System which needs to be maintained. However, in refactoring situations a higher level of Duplication might be warranted.

Solution strategies

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Solution Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reorder statements in a duplicated piece of code</td>
<td>Treating</td>
</tr>
<tr>
<td>Refactor code to extract common pieces of functionality</td>
<td>Solving</td>
</tr>
</tbody>
</table>

Related Metrics

<table>
<thead>
<tr>
<th>Metric name</th>
<th>type of relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duplicated Lines</td>
<td>Base metric</td>
</tr>
<tr>
<td>Lines of Code</td>
<td>Normalization Factor</td>
</tr>
</tbody>
</table>

Validation

<table>
<thead>
<tr>
<th>Study</th>
<th>Validation type</th>
</tr>
</thead>
<tbody>
<tr>
<td>McConnel</td>
<td>External Validity, Experience based</td>
</tr>
</tbody>
</table>

Category: SoftwareMetric

Example: Duplication %
Statement Coverage

Definition

Given a test suite $T$ and a system $S$, statement coverage is defined as the percentage of statements from $S$ that are executed when test suite $T$ is run.

Rationale

Statement coverage gives an indication to what extent a Software System has been exercised by a test suite.

Implications

In general, high statement coverage is preferable to low statement coverage.
Applicable in Context

(see Andrew Glover, Don't be fooled by the coverage report”, and Arie van Deursen, "Test Coverage: Not for Managers?”, and Arie van Deursen: "Line Coverage: Lessons Learned from JUnit")

- Improving functional test suites: If coverage achieved by a functional test suite T of specific modules or components is low, then this information can be used to come up with additional functional test cases for T.
- Effort planning: If modifications are to be made in a software component, this will be harder (more costly, slower) for components that have a low statement coverage.
- Coverage trends: If statement coverage trends go down as the software is under maintenance, this may mean that inadequate testing is done during maintenance.
- Reviews of change (pull) requests: If a code modification is proposed, statement coverage can be used to understand which parts of the change have been exercised by the accompanying test suite (and its modification).

Solution strategies

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Solution Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoke top level methods just to get a high coverage without adequate result checking</td>
<td>Treating</td>
</tr>
<tr>
<td>Examine uncovered code and expand test suite where appropriate</td>
<td>Solving</td>
</tr>
</tbody>
</table>

Related Metrics

<table>
<thead>
<tr>
<th>Metric name</th>
<th>type of relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision Coverage</td>
<td>Correlated</td>
</tr>
<tr>
<td>Line Coverage</td>
<td>Different percentages, same metric.</td>
</tr>
<tr>
<td>Lines of Code</td>
<td>Normalization Factor</td>
</tr>
</tbody>
</table>

Validation

<table>
<thead>
<tr>
<th>Study</th>
<th>Validation type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inzemtseva and Holmes: Coverage is not strongly correlated with test suite effectiveness. ICSE 2014.</td>
<td>Empirical Study</td>
</tr>
</tbody>
</table>
Discussion & Outlook

• What do we need to know about a metric to make an informed adoption decision?
  – Context, related metrics, validation

• Semantic wiki to collect such information

• We’re using it – so can you!
  http://softwaremetrics.referata.com/wiki/Main_Page