Measurement and Experiment in Software Engineering

Bill Curtis

Presented by: Lei Wang
CS5391 Spring 2003
Abstract

- Review the contributions of measurement and experimentation to the state of the art in software engineering.
- Discuss the role of measurement in developing theoretical models.
- Concern for reliability and validity are stressed.
Abstract

- Current approaches to measuring software characteristics are presented as examples.
  - Software complexity metrics related to
    - Control flow
    - Module interconnectedness
    - Halstead’s Software Science
  - Experimental methods in evaluating cause-effect relationship.
  - Example programs of experimental research.

- Conclusion
Introduction

- The core of rigorous scientific procedures
  - The development of measurement techniques.
  - The determination of cause-effect relationship.

- Modeling the process underlying a software task.
Science and Measurement

- Scientific discipline can be classified by the degree to which their analytical approach is theoretical rather than correlational. Margenau, 1950 and Torgerson, 1958.

- The development of scientific theory involves relating theoretical constructs to observe data.
Figure 1: The structure of theory in science
Science and Measurement

- Measurement quantifies a property of the construct.

- Reliability concerns the extent to which measures are accurate and repeatable.
  - Internal consistency
  - Stability

- Validity concerns whether a measure represents what it was designed to access.
  - Content validity
  - Predictive validity
  - Construct validity
### Science and Measurement

<table>
<thead>
<tr>
<th>SCALE</th>
<th>APPROPRIATE OPERATIONS</th>
<th>DESCRIPTION</th>
<th>EXAMPLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOMINAL</td>
<td>=, ≠</td>
<td>CATEGORIES</td>
<td>SEX, RACE JERSEY NUMBERS</td>
</tr>
<tr>
<td>ORDINAL</td>
<td>&lt;, &gt;</td>
<td>RANK ORDERINGS</td>
<td>NARDNESS OF MIERIAL RANK OF CLASS</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>+, –</td>
<td>EQUIVALENT INTERVALS BETWEEN NUMBERS</td>
<td>TEMPERATURE(${\text{F}^\circ}$ AND $\text{C}^\circ$) CALENDAR TIME</td>
</tr>
<tr>
<td>RATING</td>
<td>÷</td>
<td>EQUIVALENT INTERVALS AND ABSOLUTE ZERO</td>
<td>TEMPERATURE($\text{K}^\circ$) WEIGHT LINES OF CODE</td>
</tr>
</tbody>
</table>

Table 1: The operations listed for each scale are appropriate for all scales listed beneath it.
Measurement of software characteristics

- Uses for software metrics
  - Management information tools
  - Measures of software quality
  - Feedback to software personnel

- Measures of process and product
  - They are different.
  - It will be difficult to develop a metric which can represent both.
  - Mainly deal with measures of product in this paper.
Omnibus approaches to quantifying software

- Matric quantify numerous factors underlying concept
  - Boehm, Brown *et al.* see Figure 2.
  - McCall *et al.*

- Higher level constructs in each system represent
  - The current behavior of the software
  - The ease of changing the software
  - the ease of converting or interfacing the system
Omnibus approaches to quantifying software

Figure 2: The Boehm et al. software quality model
Omnibus approaches to quantifying software

Figure 3: McCall et al.’s tradeoff analysis among software quality factors.
Software Complexity

- Computational complexity
  - Relies on formal mathematical analysis
  - The quantity aspects of the solutions to computational problem

- Psychological complexity
  - The understanding that software development and maintenance are largely human activities.
  - Concern the characteristics of software which affect programmer performance
Software Complexity

- Definition of Complexity:
  Complexity is a characteristic of the software interface which influences the resources another system will expend or commit while interacting with the software.

- The focus of complexity not merely on the software, but on the software’s interactions with other system.

- Explicit criteria are not specified.

- Complexity will have different operational definitions depending on the criterion under study.
Software Complexity

Steps in modeling an aspect of software complexity:

1. define(and quantify) the criterion the metric will be developed to predict.

2. develop a model of process in the interacting system which will affect this criterion.

3. identify the properties of software which affect the operation of these process.

4. quantify these software characteristics.

5. validate this model with empirical research.
Control Structure

- Many metrics having a theoretical base in graph theory to measure software complexity by accessing the control flow.

- Discussing of McCabe’s metric
  - Define complexity in relation to the decision structure of a program.
  - Complexity metric is the classical graph-theory cyclomatic number indicating the number of regions in a graph. Demonstrated in Figure 4.

- The different counting methods developed by Basili, Reiter and Myers.
Control Structure

Figure 4: Computation of McCabe’s $V(G)$. 
Halstead’s theory of software science

\[ V = (N_1 + N_2) \log_2(\eta_1 + \eta_2) \]

\[ E = \frac{\eta_1 N_2 (N_1 + N_2) \log_2(\eta_1 + \eta_2)}{2\eta_2} \]

Where \( V \) is volume, \( E \) is effort, and

\( \eta_1 \) number of unique operators

\( \eta_2 \) number of unique operands

\( N_1 \) total frequency of operators

\( N_2 \) total frequency of operands
Experiment evaluation of Halstead and McCabe’s metrics

- Both superior to LOC for predicting the time to find and fix error.

- Better prediction of outcome may occur when more disciplined software development practices reduces the often dramatic performance differences among programmers (see Figure 5).

- Both to be valid measures of psychological complexity.

- Useful in actual practice.
Figure 5: Scatterplot of Halstead’s $V$ against development time from Sheppard et al.
Interconnectedness

Several metrics developed to access the complexity of the interconnectedness among the parts comprising a software system

Myers, 1976

Model system complexity by developing a dependency matrix among pairs of modules base on whether there is a interface between them

Present two important consideration: **Strength** of a module and **Coupling** between modules.

Yau et al., 1979, Working on validating a generic type model.
Experimental Evaluation of Software Characteristics

- Cause-Effect Relationships
  - focus on the evaluation of conditional statements and control flow.

- Conditional Statement
  - Nested conditionals – more visible and comprehensible to programmer.
  - Branch-to-label conditionals – Obscure the visibility of embedded conditions.
Conditional Statement

- Nested conditions:
  IF [condition] THEN [process 1]
  OTHERWISE [process 2].

- Branch-to-label conditions:
  IF [condition] GOTO L1
  [process 2]
  L1 [process 1].

- Branch-to-label conditions place a greater cognitive load on programmers.
**Conditional Statement**

<table>
<thead>
<tr>
<th>IF-GOTO</th>
<th>NESTED BEGIN-END</th>
<th>NESTED IF-NOT-END</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF (Condition 1)</td>
<td>IF (Condition 1) THEN</td>
<td>IF (Condition 1)</td>
</tr>
<tr>
<td>GOTO L1</td>
<td>BEGIN</td>
<td>IF (Condition 2)</td>
</tr>
<tr>
<td>IF (Condition 3)</td>
<td>IF (Condition 2) THEN</td>
<td>(Process 1)</td>
</tr>
<tr>
<td>GOTO L2</td>
<td>BEGIN (Process 1)</td>
<td>NOT (Condition 2)</td>
</tr>
<tr>
<td>(Process 4) STOP</td>
<td>END</td>
<td>(Process 2)</td>
</tr>
<tr>
<td>L1 IF(Condition 2)</td>
<td>ELSE</td>
<td>END (Condition 2)</td>
</tr>
<tr>
<td>GOTO L3</td>
<td>BEGIN (Process 2)</td>
<td>NOT (Condition 1)</td>
</tr>
<tr>
<td>(Process 2) STOP</td>
<td>END</td>
<td>IF (Condition 3)</td>
</tr>
<tr>
<td>L2 (Process 3) STOP</td>
<td>ELSE</td>
<td>(Process 3)</td>
</tr>
<tr>
<td>L3 (Process 1) STOP</td>
<td>BEGIN (Process 1)</td>
<td>NOT (Condition 3)</td>
</tr>
</tbody>
</table>
<pre><code>                          | IF (Condition 3) THEN                                                            | (Process 4)                        |
                          | BEGIN (Process 3)                                                                | END (Condition 3)                  |
                          | END                                                                               | END (Condition 1)                  |
</code></pre>

Table 2: Conditional Structures Investigated by Sime, Green and Guest.
Conditional Statement

- **IF-GOTO** cause more semantic(algorithmic) errors.
- Nested language cause primarily syntactic(grammatical) errors.
- Errors were debugged ten times faster in the **IF-NOT-END** condition.
- Sheiderman and Mayer’s Model: A language design should seek to simplify both the design and expression of an algorithm.
Conditional Statement

- Two types of information: sequence and taxon (Sime, *et al.*, 1977)

- Sequence information can easily extracted from nested language.

- Backward tracing is performed much more easily with the `IF-NOT-END` construct.

- Writing procedure and automated tool reduce the number of syntactic errors.

- More experiment on Conditional Statement: Arblaster, Sheiderman, Green, etc.
Conditional Statement

- The superiority of nested over branch-to-label conditions.
- The advantage of redundant expression of controlling conditions at the entrance to each conditional branch.
- The benefits of a software practice may vary with the nature of the task.
- A standard procedure for generating the syntax of a conditional statement can improve coding speed and accuracy.
Control Flow

- Review only experiments related to structured coding.
- The control structures generally allowed under structured coding are shown in Figure 6.
- It is not clear that structured coding will improve the productivity of programmers.
- Structured coding should reduce the costs of maintenance.
# Control flow

## Sequence

<table>
<thead>
<tr>
<th>BEGIN</th>
<th>process 1</th>
<th>process 2</th>
<th>END</th>
</tr>
</thead>
</table>

## Repetition

<table>
<thead>
<tr>
<th>WHILE CONDITION B0</th>
<th>process</th>
<th>ENDB0</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>REPEAT</th>
<th>process</th>
<th>UNTIL CONDITION</th>
</tr>
</thead>
</table>
Control flow

Figure 6: Constructs in structured control flow.
Summary

- Measurement and experimentation are complementary process.

- Results are far more impressive when they emerge from a program of research rather than from one-shot studies.

- The rigors of measurement and experimentation require serious consideration of processes underlying software phenomena.

- No substitute for sound experimental evidence in arguing the benefits of a particular software engineering practice or in comparing the relative merits of several practices.