Chap 4. Using Metrics To Manage Software Risks

1. Introduction
2. Software Measurement Concepts
3. Case Study: Measuring Maintainability
4. Metrics and Quality
1. Introduction

**Definition**

Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world so as to describe them according to specified rules.

- There are two broad use of measurement: assessment and prediction:
  - Predictive measurement of some attribute $A$ relies on a mathematical model relating $A$ to some existing measures of attributes $A_1, \ldots, A_n$.
  - Assessment is more straightforward and relies on the current status of the attribute.

- There are 3 classes of software metrics: *process, product, and project*
  - *Process metrics*: measure process effectiveness; Example: defect removal effectiveness.
  - *Product metrics*: measure product characteristics such as size, cost, defect count etc.
  - *Project metrics*: used to keep track of project execution; Examples: development time, development effort, productivity etc.
-Software metrics provide a *quantitative vehicle for evaluating and managing quality factors and risks* related to a given software product.

-The software artifacts concerned by metrics include *analysis*, and *design models*, as well as *program code*.

-Metrics can be used at early stages as leading quality indicators of the software architecture design. They can also be used to *drive an iterative design process* (such as the Rational Unified Process).

-Metrics may be collected either dynamically or statically.
  - *Dynamic* metrics require execution of the software system, which restrict their applicability to later phases of the development.
  - *Static* metrics, in contrast can be collected and used at early stages of the design.
2. Software Measurement Concepts

- Measurement always targets specific software attribute or concept:
  - Examples: complexity, cohesion, coupling, size, time, effort, maintainability etc.

- In software measurement studies, a distinction is made between internal and external attributes:
  - **Internal attributes**: are those which can be measured purely in terms of the product, process, or project itself. Examples: size for product and elapsed time for process.
  - **External attributes**: are those which can only be measured with respect to how the product, process, or project relates to other entities in its environment. Examples: reliability for product and productivity for project (e.g., people).

- Software Managers and Users would like to measure and predict external attributes.
  - External attributes are easy to interpret but hard to measure directly, while internal attributes are hard to interpret but relatively easy to collect directly.
-In practice, measurement of external attributes are derived indirectly from internal (attributes) measures, through correlation or statistical analysis such as regression or Bayesian probabilistic models.

Example:

\[ \text{Product Cost} = f(\text{effort, time}); \text{Effort (person/month)} = g(\text{size}) \]
3. Case Study: Measuring Maintainability

- Important aspects of maintainability include understandability, flexibility, reusability, and testability.

- Complex code is difficult to understand, and thereby to maintain and evolve. Complex code increases the cost of testing, because the likelihood of faults is higher.

- Complexity is mastered by applying the principle of “divide and conquer”, which typically underlies another common design principle, namely modular design.

- Good modular design requires high cohesion of modules, and less coupling between them.
  - Less cohesion means more complexity.
  - Strong coupling means reduced reusability.

- Several software product metrics have been proposed to evaluate the complexity factors that affect the creation, comprehension, modification, and maintenance of a piece of software.
<table>
<thead>
<tr>
<th>Metrics</th>
<th>Available at Design</th>
<th>Constructs/Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyclomatic complexity (CC)</td>
<td>N</td>
<td>Method/Complexity</td>
</tr>
<tr>
<td>Lines of Code (LOC)</td>
<td>N</td>
<td>Method/Size, complexity</td>
</tr>
<tr>
<td>Comment percentage (CP)</td>
<td>N</td>
<td>Method/Complexity</td>
</tr>
<tr>
<td>Weighted methods per class (WMC)</td>
<td>Y</td>
<td>Class,Method/Complexity</td>
</tr>
<tr>
<td>Response for a class (RFC)</td>
<td>N</td>
<td>Class, Method/Complexity</td>
</tr>
<tr>
<td>Lack of cohesion of methods (LCOM)</td>
<td>N</td>
<td>Class/Cohesion</td>
</tr>
<tr>
<td>Coupling between objects classes (CBO)</td>
<td>Y</td>
<td>Class/Coupling</td>
</tr>
<tr>
<td>Depth of inheritance tree (DIT)</td>
<td>Y</td>
<td>Inheritance/Complexity</td>
</tr>
<tr>
<td>Number of children (NOC)</td>
<td>Y</td>
<td>Inheritance/Complexity</td>
</tr>
</tbody>
</table>

8
**Cyclomatic Complexity (CC)**
- Also called McCabe complexity metric
- Evaluate the *complexity of algorithms* involved in a method.
- Give a count of the number of test cases needed to test a method comprehensively;
- Use a control flow graph (CFG) to describe the software module or piece of code under study:
  - *Each node* corresponds to a block of sequential code.
  - *Each edge* corresponds to a path created by a decision.

- CC is defined as the number of edges minus the number of nodes plus 2: \( CC = \text{edges} - \text{nodes} + 2 \)

\[
CC = e - n + 2 = 8 - 7 + 2 = 3
\]

*Low CC means reduced testing, and better understandability.*
**Primitive Operations of Structured Programming**

- **Sequence**:
  
  $y = 2 + x$

  $CC = 1 - 2 + 2 = 1$

- **If/Then/Else**:

  if ($x > 2$) $y = 2x$
  
  else $y = 2$

  $CC = 4 - 4 + 2 = 2$

- **While**:

  while ($x > 2$) $y = 2x$

  $CC = 3 - 3 + 2 = 2$

- **For Loop**:

  for (int $i = 0; i < 5; i++$)

  $x = x + i$

  $CC = 5 - 5 + 2 = 2$
Size

-The size of a piece of code can be measured using different metrics.

\(\text{(Basic) Lines of code (LOC)}\) count all lines, including comments;
\(\text{Non-comment non-blank (NCNB)}\) counts all lines except comments and blanks.
\(\text{Executable statements (EXEC)}\) count the number of executable statements.

Examples:

\[
\begin{align*}
\text{if } x & > 2 \\
\text{then } y & = x + z; \\
\text{/* evaluates … */} \\
\text{if } x & > 2 \\
\text{then } y & = x + z; \\
x & = 2z;
\end{align*}
\]

\(\text{LOC}=2\), \(\text{NCNB}=2\), \(\text{EXEC}=1\) \quad \text{LOC}=5\), \(\text{NCNB}=3\), \(\text{EXEC}=2\)

High size decreases understandability, and therefore increases risk and faults.
Comment Percentage (CP)

- Is obtained by the total number of comments divided by the total number of lines of code less the number of blank lines.

Example:

/*evaluates...*/
if \(x>2\)
then \(y=x+z\);

\(x=2z;\)

/*computes...*/
\(z=x^2-y;\)

\(CP = \frac{2}{8-2} = 33\%\)

Higher comment percentage means better understandability and maintainability.
**Weighted Methods per Class (WMC)**

- Is measured either by counting the number of methods associated with a class, or by summing the complexities (CC) of the methods.

\[
WMC = \sum_{i=1}^{n} c_i, \quad c_i = CC_i
\]

**Example:**

<table>
<thead>
<tr>
<th>Class</th>
<th>Methods</th>
<th>WMC</th>
</tr>
</thead>
</table>
| Person     | name: Name
            employeeID: Integer
            title: String
            getContactInformation(): ContactInformation
            getPersonalRecords(): Personalrecords | 2   |
**Response For a Class (RFC)**
- Measure the number of methods that can be invoked in response to a message to an object of the class or by some methods in the class; this includes all the methods accessible in the class hierarchy.

**Example:**

### RFC (StoreDepartments)
- $\text{RFC (StoreDepartments)} = 2 = 1 + 3 = 6$

### RFC (Clothing)
- $\text{RFC (Clothing)} = 1 + 2 = 3$

### RFC (Appliances)
- $\text{RFC (Appliances)} = 3 + 2 = 5$

*Higher RFC value is a predictor of larger number of communications with other classes, so more complexity.*
Lack of Cohesion (LCOM)
- Measure the cohesion or lack of a class; evaluate the dissimilarity of methods in a class by instance variables or attributes.
- LCOM is measured by counting the number of pairs of methods that have no attributes in common, minus the number of methods that do. A negative difference corresponds to LCOM value of zero.

Example:

Low cohesion is a sign of high complexity, and shows that the class can be subdivided.
High cohesion indicates simplicity and high potential for reuse.

<table>
<thead>
<tr>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>type: int</td>
</tr>
<tr>
<td>reading: int</td>
</tr>
<tr>
<td>mode: boolean</td>
</tr>
</tbody>
</table>

Class Device {
    int reading, type;
    boolean mode=false;

    public int update (int a) {return a + reading; }
    public int compute(int x, int y) {return x*y*type - reading;}
    public void test (int t) { if  t ==1   mode=true; }
}

LCOM(Device) = 2-1 =1
**Coupling Between Object Classes (CBO)**

- Measure the number of classes to which a class is coupled.
- Class A is coupled to class B iff A uses B’s methods or instance variables.
- Coupling is calculated by counting the number of distinct non-inheritance related class hierarchies on which a class depends.

**Example:**

```
StoreDepartments
    manager
    employees
    display()
    credit()

Supplier
    products

Warehouse
    stock

JacketDepartment
    customer_type
    size_range
    exchange()

SlacksDepartment
    customer_type
    size_range
    exchange()
    purchase()
```

- \( CBO(\text{StoreDepartments}) = 1 \)
- \( CBO(\text{Warehouse}) = 1 \)
- \( CBO(\text{Supplier}) = 0 \)

High coupling means increased dependency among the classes; this restricts reusability.

Useful for determining reusability.
**Depth of Inheritance Tree (DIT)**
-Measure the number of ancestor classes of a given class involved in an inheritance relation.

**Example:**

```
Department
  StoreDepartments
    manager
    employees
    display()
    credit()
  Appliances
    category
    delivery()
    service()
    parts_ordering()
    DIT (Appliances) = 2
    DIT (StoreDepartments) = 1
    DIT (Department) = 0
```

Greater value of DIT means more methods to be inherited, so increased complexity; but at the same time that means increased reusability; so a trade-off must be made here.
Number of Children (NOC)

-Measure the number of immediate subclasses of a class in an inheritance hierarchy.

**Example:**

- **NOC(Division) = 1**
- **NOC(StoreDepartments) = 2**
- **NOC(Appliances) = 0**

*High NOC means high reuse, but may also be the sign of improper abstraction or misuse of inheritance. High NOC may also be the sign of increased complexity. So a trade-off must be made for this metric.*
EXAMPLE: compute relevant CK metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Bill</th>
<th>Purchase</th>
<th>Warning Ltrs</th>
<th>Periodic Msgs</th>
<th>Message</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coupling Between Objects</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Metric</td>
<td>Bill</td>
<td>Purchase</td>
<td>Warning Ltrs</td>
<td>Periodic Msgs</td>
<td>Message</td>
<td>Customer</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------</td>
<td>----------</td>
<td>--------------</td>
<td>---------------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>Weighted Methods/Class</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Number of Children</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Depth of Inheritance Tree</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Response for a Class</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Coupling Between Objects</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Lack of Cohesion in Methods</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
4. Metrics and Quality

*Metrics can be useful indicators of unhealthy code and design, pointing out areas where problems are likely to occur, by focusing on specific quality attributes.*

<table>
<thead>
<tr>
<th>Metric</th>
<th>Source</th>
<th>OO Construct</th>
<th>Objectives</th>
<th>Quality Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>Traditional</td>
<td>Method</td>
<td>Low</td>
<td>Testability, Understandability</td>
</tr>
<tr>
<td>LOC</td>
<td>Traditional</td>
<td>Method</td>
<td>Low</td>
<td>Understandability, Reusability, Maintainability</td>
</tr>
<tr>
<td>CP</td>
<td>Traditional</td>
<td>Method</td>
<td>~20-30%</td>
<td>Understandability, Maintainability</td>
</tr>
<tr>
<td>WMC</td>
<td>New OO</td>
<td>Class/Method</td>
<td>Low</td>
<td>Testability, Reusability</td>
</tr>
<tr>
<td>DIT</td>
<td>New OO</td>
<td>Inheritance</td>
<td>Low (Trade-off)</td>
<td>Reuse, Understandability, Maintainability</td>
</tr>
<tr>
<td>NOC</td>
<td>New OO</td>
<td>Inheritance</td>
<td>Low (Trade-off)</td>
<td>Reusability, Testability</td>
</tr>
<tr>
<td>CBO</td>
<td>New OO</td>
<td>Coupling</td>
<td>Low</td>
<td>Usability, Maintainability, Reusability</td>
</tr>
<tr>
<td>RFC</td>
<td>New OO</td>
<td>Class/Method</td>
<td>Low</td>
<td>Usability, Reusability, Testability</td>
</tr>
<tr>
<td>LCOM</td>
<td>New OO</td>
<td>Class/Cohesion</td>
<td>Low High</td>
<td>Complexity, Reusability</td>
</tr>
</tbody>
</table>