Software Engineering Features - Models, Methods, Tools, Standards, and Metrics

From SEBoK
Software Engineering Features - Models, Methods, Tools, Standards, and Metrics

Lead Author: Tom Hilburn

In recent decades software has become ubiquitous. Almost all modern engineered systems include significant software subsystems; this includes systems in the transportation, finance, education, healthcare, legal, military, and business sectors. Along with the increase in software utility, capability, cost, and size there has been a corresponding growth in methods, models, tools, metrics and standards, which support software engineering.

Chapter 10 of the SWEBOK discusses modeling principles and types, and the methods and tools that are used to develop, analyze, implement, and verify the models. The other SWEBOK chapters on the software development phases (e.g., Software Design) discuss methods and tools specific to the phase. Table 1 identifies software engineering features for different life-cycle phases. The table is not meant to be complete; it simply provide examples. In Part 2 of the SEBoK there is a discussion of models and the following is one of the definitions offered: “an abstraction of a system, aimed at understanding, communicating, explaining, or designing aspects of interest of that system” (Dori 2002).

For the purposes of Table 1 we extend the definition of a model to some aspect of the software system or it development. So as an example, we list “Project Plan” as a model in the Software Management area. The idea is that the Project Plan provides a model of how the project is going to be carried out; the project team organization, the process to be used, the work to be done, the project schedule, and the resources needed.

<table>
<thead>
<tr>
<th>Life-Cycle Activity</th>
<th>Models</th>
<th>Methods &amp; Tools</th>
<th>Standards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Management</td>
<td>• Life-Cycle Process Model</td>
<td>• Effort, Schedule and Cost Estimation</td>
<td>• [IEEE 828]</td>
</tr>
<tr>
<td></td>
<td>• Work Breakdown Structure</td>
<td>• Risk Analysis</td>
<td>• [IEEE 1058]</td>
</tr>
<tr>
<td></td>
<td>• Constructive Cost Model (COCOMO)</td>
<td>• Data Collection</td>
<td>• [IEEE 1540]</td>
</tr>
<tr>
<td></td>
<td>• Project Plan</td>
<td>• Project Tracking</td>
<td>• [IEEE 12207]</td>
</tr>
<tr>
<td></td>
<td>• Configuration Management (CM) Plan</td>
<td>• CM Management</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Risk Management Plan</td>
<td>• Iterative/Incremental Development</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Agile Development</td>
<td></td>
</tr>
</tbody>
</table>
### Software Requirements

- Functional Model
- User Class Model
- Data Flow Diagram
- Object Model
- Formal Model
- User Stories

### Software Design

- Architectural Model
- Structure Diagram
- Object Diagram
- Class Specification
- Data Model
- Detailed Design Document
- Pseudocode
- Flow Chart
- Program Code
- Unit Test Plan
- Integration Test Plan

### Software Construction

- System Test Plan
- Reliability Model
- Software Maintenance Process
- Software Maintenance Process

### Software Testing

- Requirements Elicitation
- Prototyping
- Structural Analysis
- Data-Oriented Analysis
- Object-Oriented Analysis
- Object Modeling Language (OML)
- Formal Methods
- Requirements Specification
- Requirements Inspection
- Structured Design
- Object-Oriented Design
- OML
- Modular Design
- Integrated Development Environment (IDE)
- Database Management System (DBMS)
- Design Review
- Refinement
- Detailed Design
- Functional Programming
- Object-Oriented Programming
- IDE
- DBMS
- Black Box/White Box Testing
- Basic Path Testing
- Unit Testing
- Code Review
- Proof of Correctness
- Software Reuse
- Integration
- Integration Testing
- Usability Testing
- System Testing
- Acceptance Testing
- Regression Testing
- Reliability Testing
- Non Functional Software Testing
- Automated Testing Tools
- Maintenance Change
- Impact Analysis
- Inventory Analysis
- Restructuring
- Reverse Engineering
- Re-engineering

### Software Metric

A software metric is a quantitative measure of the degree a software system, component, or process possesses a given attribute. Because of the abstract nature of software and special problems with software schedule, cost, and quality, data collection and the derived metrics are an essential part of software engineering. This is evidenced by the repeated reference to measurement and metrics in the SWEBOK. Table 2 describes software metrics that are collected and used in different areas of programming.
software development. As in Table 1 the list is not meant to be complete, but to illustrate the type and range of measures used in practice.

Table 2: Software Metrics * (SEBoK Original)

<table>
<thead>
<tr>
<th>Category</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management Metrics</td>
<td>• Size: Lines of Code (LOC*), Thousand Lines of Code (KLOC)</td>
</tr>
<tr>
<td></td>
<td>• Size: Function points, Feature Points</td>
</tr>
<tr>
<td></td>
<td>• Individual Effort: hours</td>
</tr>
<tr>
<td></td>
<td>• Task Completion Time: hours, days, weeks</td>
</tr>
<tr>
<td></td>
<td>• Project Effort: person-hours</td>
</tr>
<tr>
<td></td>
<td>• Project Duration: months</td>
</tr>
<tr>
<td></td>
<td>• Schedule: earned value</td>
</tr>
<tr>
<td></td>
<td>• Risk Projection: risk description, risk likelihood, risk impact</td>
</tr>
<tr>
<td></td>
<td>• Defect Density - Defects/KLOC (e.g., for system test)</td>
</tr>
<tr>
<td></td>
<td>• Defect Removal Rate - defect removed/hour (for review and test)</td>
</tr>
<tr>
<td>Software Quality Metrics</td>
<td>• Test Coverage</td>
</tr>
<tr>
<td></td>
<td>• Failure Rate</td>
</tr>
<tr>
<td></td>
<td>• Change requests received, open, closed</td>
</tr>
<tr>
<td></td>
<td>• Change request frequency</td>
</tr>
<tr>
<td>Software Requirements Metrics</td>
<td>• Effort required to implement a requirement change</td>
</tr>
<tr>
<td></td>
<td>• Status of requirements traceability</td>
</tr>
<tr>
<td></td>
<td>• User stories in the backlog</td>
</tr>
<tr>
<td></td>
<td>• Cyclomatic Complexity</td>
</tr>
<tr>
<td></td>
<td>• Weighted Methods per Class</td>
</tr>
<tr>
<td>Software Design Metrics</td>
<td>• Cohesion - Lack of Cohesion of Methods</td>
</tr>
<tr>
<td></td>
<td>• Coupling - Coupling Between Object Classes</td>
</tr>
<tr>
<td></td>
<td>• Inheritance - Depth of Inheritance Tree, Number of Children</td>
</tr>
<tr>
<td></td>
<td>• Mean Time Between Changes (MTBC)</td>
</tr>
<tr>
<td></td>
<td>• Mean Time To Change (MTTC)</td>
</tr>
<tr>
<td>Software Maintenance and Operation</td>
<td>• System Reliability</td>
</tr>
<tr>
<td></td>
<td>• System Availability</td>
</tr>
<tr>
<td></td>
<td>• Total Hours of Downtime</td>
</tr>
</tbody>
</table>

*Note: Even though the LOC metric is widely used, using it comes with some problems and concerns: different languages, styles, and standards can lead to different LOC counts for the same functionality; there are a variety of ways to define and count LOC – source LOC, logical LOC, with or without comment lines, etc.; and automatic code generation has reduced the effort required to produce LOC.

References

Works Cited


**Additional References**


