**Requirements**

- System Design
  - (Architecture, High-level Design)

- Module Design
  - (Program Design, Detailed Design)

- Implementation of Units (classes, procedures, functions)

**Implementation**

- Module Testing
  - (Integration testing of units)

- System Testing
  - (Integration testing of modules)

- Acceptance Test
  - (Release testing)

**Verify**

- System Design
- Module Design
- Implementation
- Requirements
- Verify Specification
- Verify System Design
- Verify Module Design
- Verify Implementation

**Supporting**

- Project Management
- Software Quality Assurance (SQA)
- Supporting Tools
- Education

**Maintenance**
Quality factors

- Correctness
- Reliability
- Efficiency
- Usability
- Integrity
- Maintainability
- Flexibility
- Testability
- Security

- Portability
- Reusability
- Interoperability
- Survivability
- Safety
- Manageability
- Supportability
- Replaceability
- Functionality

Measuring these requires both research, experience and imagination.
Remember Reliability?

- The probability that the software executes with no failures during a specified time interval
  - Approximation: MTTF/(1+MTTF)
  - Example
  - Easier to manage: Failure intensity, [failures / hours of execution time]
  - Another approximation: \( \lambda = (1-R)/t \)
  - Example
<table>
<thead>
<tr>
<th>Relevance</th>
<th>Efficiency</th>
<th>Learnability</th>
<th>Attitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>• number of good and bad features recalled by users</td>
<td>• time to complete a task</td>
<td>• ratio of successes to failures (over time)</td>
<td>• percent of favorable/unfavorable user comments</td>
</tr>
<tr>
<td>• number of available commands not invoked by users</td>
<td>• percent of task completed</td>
<td>• time spent in errors</td>
<td>• number of good and bad features recalled by users</td>
</tr>
<tr>
<td>• number of available commands invoked by users</td>
<td>• percent of task completed per unit time (speed metric)</td>
<td>• percent or number of errors</td>
<td>• number of users preferring the system</td>
</tr>
<tr>
<td>• number of times user needs to work around a problem</td>
<td>• time spent in errors</td>
<td>• number of commands used</td>
<td>• number of times user loses control of the system</td>
</tr>
<tr>
<td>• percent of task completed</td>
<td>• frequency of help and documentation use</td>
<td>• frequency of help and documentation use</td>
<td>• number of times the user is disrupted from a work task</td>
</tr>
<tr>
<td></td>
<td>• time spent using help or documentation</td>
<td>• time spent using help or documentation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• number of repetitions of failed commands</td>
<td>• number of repetitions of failed commands</td>
<td></td>
</tr>
</tbody>
</table>
Measurement - metrics

Most common use:

• Measurement – directly measured on:
  • Document, no of pages
  • Design, no of model elements
  • Code, no of lines
  • Process, iteration length
  • Quality, avg no of hours to learn a system

• Metrics – is a combination of measurements, eg.
  number of faults found in test/hours of testing
Cyclomatic complexity has a foundation in graph theory and is computed in the following ways:

1. Cyclomatic complexity $V(G)$, for a flow graph, $G$, is defined as:

   $$V(G) = E - N + 2P$$

   - $E$: number of edges
   - $N$: number of nodes
   - $P$: number of disconnected parts of the graph

2. Cyclomatic complexity $V(G)$, for a flow graph, $G$, with only binary decisions, is defined as:

   $$V(G) = b + 1$$

   - $b$: number of binary decisions
Examples of Graphs and calculation of McCabe’s Complexity Metric

*Student Worksheet*
Control-flow

Basic block

E = 9
N = 8
P = 1

B = 2

V = 3
Software metrics

- Usage-based metrics
- Verification & Validation metrics
- Volume metrics
- Structural metrics
- Effort metrics
- Direct measurement
- Indirect measurement

Note: Pedagogical model only!
Usage based metrics - example

- **Description:** Number of good and bad features recalled by users.

- **How to obtain data:** Set up a test scenario. Let test users run the scenario. Collect number of good and bad features in a questionnaire afterwards.

- **How to calculate the metric:** Take the average of number of good and no. bad features. Two values.

- **Relevant quality factor:** Relevance – many good and few bad features indicates a good match with the users’ mind-set.
Verification and validation metrics - example

• Description: Rate of severe defects found in inspection of design description.

• How to obtain data: Perform an inspection according to your process. Make sure that severity is in the classification scheme.

• How to calculate the metric: Divide the number of defects classified with highest severity with total number of defects in the Inspection record.

• Relevant quality factor: Safety – a high proportion of severe defects in design indicates fundamental problems with the solution and/or competence.
Volume metrics - example

- Description: Number on non-commented lines of code.
- How to obtain data: Count non-commented lines of the code with a tool.
- How to calculate the metric: See above.
- Relevant quality factor: Reliability – it is often hard to understand a large portion of code, the fault density is often higher for large modules.
Structural metrics - example

• Description: Maximum depth of inheritance tree.

• How to obtain data: Count the depth of the inheritance tree for all classes with a tool.

• How to calculate the metric: Take the maximum value of the classes.

• Relevant quality factor: Understandability – It is hard to determine how a change in a higher class will affect inherited/overridden methods.
Effort metrics - example

• Description: Time spent in testing.
• How to obtain data: Make sure that testing activities are distinguished in time reporting forms. Make sure that all project activities are reported.
• How to calculate the metric: Sum the number of hours for all activities in testing for all people involved.
• Relevant quality factor: Testability – a comparably long testing time indicates low testability.
The Goal Question Metric approach

- Outside the written exam we can use a top-down approach: Goal-Question-Metric (GQM)

<table>
<thead>
<tr>
<th>Goal</th>
<th>Purpose</th>
<th>Improve</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Issue</td>
<td>the timeliness of</td>
</tr>
<tr>
<td></td>
<td>Object (process)</td>
<td>change request processing</td>
</tr>
<tr>
<td></td>
<td>Viewpoint</td>
<td>from the project manager's viewpoint</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Question</th>
<th>What is the current change request processing speed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metrics</td>
<td>Average cycle time</td>
</tr>
<tr>
<td></td>
<td>Standard deviation</td>
</tr>
<tr>
<td></td>
<td>% cases outside of the upper limit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Question</th>
<th>Is the performance of the process improving?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metrics</td>
<td>Current average cycle time</td>
</tr>
<tr>
<td></td>
<td>$\times 100$</td>
</tr>
<tr>
<td></td>
<td>Baseline average cycle time</td>
</tr>
<tr>
<td></td>
<td>Subjective rating of manager's satisfaction</td>
</tr>
</tbody>
</table>

Basili, Caldiera, Rombach (1994)
### Research

<table>
<thead>
<tr>
<th>Metric</th>
<th>Threshold Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Self Transitions</td>
<td>-</td>
</tr>
<tr>
<td>Transitions/State</td>
<td>Middle level state</td>
</tr>
<tr>
<td></td>
<td>4 - 5</td>
</tr>
<tr>
<td>State Depth</td>
<td>3</td>
</tr>
</tbody>
</table>

\[
\text{Rank} = 1.2 + 0.007\text{NonSelfTransitions} + 0.17\text{Transitions/state} + 0.25\text{StateDepth}
\]

*Rezaei, Ebersjö, Sandahl, Staron*

*Identifying and managing complex modules in executable software design models*

*IWSM Mensura 2014 conference*