Non-Linear Procedure Models

**Evolutionary and Iterative Development**

- **Analysis of Requirements**
- **Use on Target System**
- **Defined Steps**
- **Preliminary Results Used**
- **Complete Plan**

- **Rapid Prototyping**
- **Evolutionary Development**
- **Iterative Development**
- **Incremental Development**

---

**Incremental Development**

- The total extension of a system under development remains open; it is realised in stages of expansion. The first stage is the core system. Each stage of expansion extends the existing system and is subject to a separate project. Providing a new stage of expansion typically includes (as with iterative development) an improvement of the old components.

- **Note**: (to maximise confusion) IEEE calls our "iterative" incremental:

- **Incremental Development**

  - A software development technique in which requirements definition, design, implementation, and testing occur in an overlapping, iterative (rather than sequential) manner, resulting in incremental completion of the overall software product.

- IEEE 610.12 (1990)

---

**The Spiral Model**

- **Analysis of Requirements**
- **Use on Target System**
- **Defined Steps**
- **Preliminary Results Used**
- **Complete Plan**

- **Rapid Prototyping**
- **Evolutionary Development**
- **Iterative Development**
- **Incremental Development**

---

**The Spiral Model**

- An approach to project planning and control that is based on the development cycle, which is represented by the spiral. The cycle consists of the following steps:
  1. Planning
  2. Risk Identification
  3. Implementation
  4. Review

- **Goals**
  - To reduce the risk of project failure by focusing on the most important aspects first.
  - To improve the predictability of project outcomes.

---

**Software Metrics**

- **Capabilities for following tasks/questions.**
  - What is evolutionary, incremental, iterative?
  - What's the fundamental idea of the spiral model? Where's the spiral?
  - What is the difference between procedure and process model?
  - What are the constituting elements of "V-Modell XT"? What project types does it support, what is the consequence? What is tailoring in the context of "V-Modell XT"?
  - What are examples of agile process models? What are their principles? Describe XP, Scrum.
  - What is a nominal, . . . , absolute scale? What are their properties?
  - Which properties make a metric useful?
  - What's the difference between objective, subjective, and pseudo metrics?
  - Compute LOC, cyclomatic complexity, LCOM, . . . for this software.

---

**Content:**

- Non-linear procedure models cont'd, process models (V-Modell XT, Scrum, . . . )
- Scales, metrics

Risk – a problem, which did not occur yet, but on occurrence threatens important project goals or results. Whether it will occur, cannot be surely predicted.

Ludewig & Lichter (2013)

\[ \text{riskvalue} = p \cdot K \]

\( p \): probability of problem occurrence,

\( K \): cost in case of problem occurrence.

\( 10^{9} \), \( 10^{10} \), \( 10^{11} \)

10

\( 10^{-3} \)

\( 0.01 \), \( 0.1 \), \( 1 \), \( 10 \), \( 100 \), \( 500 \)

incidence probability

\( \frac{1}{10^{-3}} \)

Acceptable risks

Unacceptable risks

Extreme risks

Avionics requires: "Average Probability per Flight Hour for Catastrophic Failure Conditions of \( 10^{-9} \) or ‘Extremely Improbable’ (AC 25.1309-1)."

• "problems with \( p = 500 \cdot 10^{-3} = 0.5 \) are not risks, but environment conditions to be dealt with"

The Spiral Model (Boehm, 1988)

- Repeat until end of project (successful completion or failure):
  1. Determine the set \( R \) of risks threatening the project; if \( R = \emptyset \), the project is successfully completed
  2. Assign each risk \( r \in R \) a risk value \( v(r) \)
  3. For the risk \( r_0 \) with the highest risk value, \( r_0 = \max \{ v(r) \mid r \in R \} \), find a way to eliminate this risk, and go this way; if there is no way to eliminate the risk, stop with project failure

Advantages:

• We know early if the project goal is unreachable,

Note:

Risk can by anything; e.g. open technical questions (→ prototype?), but also lead developer leaving the company (→ invest in documentation), changed market situation (→ adapt appropriate features), . . .

Wait, Where’s the Spiral?

A concrete process using the Spiral Model could look as follows:

\[ \text{t} (\text{cost, project progress}) \]

- Fix goals, conditions,
- Risk analysis,
- Develop and test,
- Plan next phase,
To systematically compare and improve industrial products, we need to precisely describe and assess the products and the process of creation.

This common practice for many material goods, e.g., cars:
- fuel consumption,
- size of trunk,
- fixed costs per year,
- time needed to change a headlight's light bulb,
- clearance (accuracy of fit and gaps of, e.g., doors)... Note: all these key figures are models of products — they reduce everything but the aspect they are interested in.

Less common practice for immaterial goods like Software.
- It should be — (objective) measures are central to engineering approaches.
- Yet: it's not that easy for software.

Excursion: Scales

• measuring maps elements from a set \( A \) to a scale \( M \):
  \[ m: A \rightarrow M \]
• we distinguish (i) nominal scale
  • operations: \( = \) (and \( \neq \))
(iii) interval scale (with units)
  • operations: \( = \), \(<\), \(\leq\), \(\geq\), \(\neq\), \(\neq\)

(iv) rational scale (with units)
  • operations: \( = \), \(<\), \(\leq\), \(\geq\), \(\neq\), \(\neq\), \(\neq\)

(v) absolute scale
  • a rational scale where \( M \) comprises the key figures itself

Nominal Scale

• operations: \( = \) (and \( \neq \))
• that is, there is no (natural) order between elements of \( M \),
• the lexicographic order can be imposed, but is not related to measured information
• general example:
  - nationality, gender, car manufacturer, geographic direction, . . .
  - Autobahn number, train number, . . .

Ordinal Scale

• operations: \( = \), \(<\), \(<\), \(\leq\), \(\geq\), \(\neq\), \(\neq\), \(\neq\)
• there is a (natural) order between elements of \( M \), but no (natural) notion of distance or average
• general example:
  - strongly agree > agree > disagree > strongly disagree
  - administrative ranks: Chancellor > Minister
  - ranking list, leaderboard: finishing number tells us who was, e.g. faster, than who; but nothing about how much faster 1st was than 2nd

Interval Scale

• operations: \( = \), \(<\), \(<\), \(\leq\), \(\geq\), \(\neq\), \(\neq\), \(\neq\)
• there's a (natural) notion of difference \( \Delta \) :
  \[ M \times M \rightarrow \mathbb{R} \]
• but no (natural) 0
• general example:
  - temperature in Celsius (no zero),
  - year dates, two persons, born \( B_1 \), died \( D_1 \), \( B_2 \), died \( D_2 \) (all dates beyond, say, 1900) — if \( \Delta(B_1, D_1) = \Delta(B_2, D_2) \), they reached the same age

Software and Process Metrics

• measuring maps elements from \( A \) to \( M \)

Software and Types of Scales

• measuring maps elements from \( A \) to \( M \)
Rational Scale

- operations:
  - =
  - <, >
  - min/max
  - percentiles
  - ∆
  - proportion
  - 0
  - the (natural) zero induces a meaning for proportion

- general example:
  - age ("twice as old"), finishing time, weight, pressure, ...
  - price, speed, distance from Freiburg, ...

Software engineering example:
  - runtime of a program for certain inputs

Absolute Scale

- M = N₀
  - a rational scale where M comprises the key figures itself
  - absolute scale has median, but in general not an average

- general example:
  - seats in a bus, number of public holidays, number of inhabitants of a country, ...
  - "average number of children per family: 1.203" – what is a 0.203-child? the absolute scale has been viewed as a rational scale, makes sense for certain purposes

Software engineering example:
  - number of known errors

Communicating Figures

Median and Box-Plots

- arithmetic average: 2785.6
- median: 127, 139, 152, 213, 13297
- a boxplot visualises 5 aspects of data at once (whiskers sometimes defined differently, with "outliers")

40.0003 0.0002 0.0001 0.000

median: 2,078
average: 7,033.027

LOC lecture's *.tex files

Software Metrics

- metric — A quantitative measure of the degree to which a system, component, or process possesses a given attribute. See: quality metric.

IEEE 610.12 (1990)

- quality metric — (1) A quantitative measure of the degree to which an item possesses a given quality attribute.(2) A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given quality attribute.
**Metric Space**

**Definition.**

A function \( d : X \times X \to \mathbb{R} \) is called a metric on \( X \) if and only if, for each \( x, y, z \in X \),

- \( d(x, y) \geq 0 \) (non-negative)
- \( d(x, y) = 0 \iff x = y \) (identity of indiscernibles)
- \( d(x, y) = d(y, x) \) (symmetry)
- \( d(x, z) \leq d(x, y) + d(y, z) \) (triangle inequality)

\((X, d)\) is called a metric space.

**Software Metrics: Motivation and Goals**

Important motivations and goals for using software metrics:

- Support decisions
- Quantify experience, progress, etc.
- Assess the quality of products and processes
- Predict cost/effort, etc.

Metrics can be used:

- Descriptive or prescriptive:
  - "the current average LOC per module is \( N \)" vs. "a procedure must not have more than \( N \) parameters"
- A descriptive metric can be diagnostic or prognostic:
  - "the current average LOC per module is \( N \)" vs. "the expected test effort is \( N \) hours"
- Note: prescriptive and prognostic are different things.

Examples for diagnostic/guiding use:

- Measure time spent per procedure before starting "optimisations",
- Focus testing effort accordingly, e.g. guided cyclomatic complexity,
- Develop measures indicating architecture problems, (analyse,) then focus refactoring

**Requirements on Useful Metrics**

A thing which is subject to the application of a metric is called a proband. The value \( m(P) \) yielded by a given metric \( m \) on a proband \( P \) is called the valuation yield ('Bewertung') of \( P \).

In order to be useful, a (software) metric should be:

- Differentiated – worst case: same valuation for all probands
- Comparable – ordinal scale, better: rational (or absolute) scale
- Reproducible – multiple applications of a metric to the same proband should yield the same valuation
- Available – valuation yields need to be in place when needed
- Relevant – wrt. overall needs
- Economical – worst case: doing the project gives a perfect estimation of duration, but is expensive; irrelevant metrics are not economical (if not available for free)
- Plausible – (\( \rightarrow \) pseudo-metric)
- Robust – developers cannot arbitrarily manipulate the yield; antonym: subvertible

**Requirements on Useful Metrics: Examples**

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Positive Example</th>
<th>Negative Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Differentiated</td>
<td>Program length in LOC</td>
<td>CMM/CMMI level below 2</td>
</tr>
<tr>
<td>Comparable</td>
<td>Cyclomatic complexity</td>
<td>Review (text)</td>
</tr>
<tr>
<td>Reproducible</td>
<td>Memory consumption</td>
<td>Grade assigned by inspector</td>
</tr>
<tr>
<td>Available</td>
<td>Number of developers</td>
<td>Number of errors in the code (not only known ones)</td>
</tr>
<tr>
<td>Relevant</td>
<td>Expected development cost; number of errors</td>
<td>Number of subclasses (NOC)</td>
</tr>
<tr>
<td>Economical</td>
<td>Number of discovered errors in code</td>
<td>Highly detailed timekeeping</td>
</tr>
<tr>
<td>Plausible</td>
<td>Cost estimation following COCOMO (to a certain amount)</td>
<td>Cyclomatic complexity of a program with pointer operations</td>
</tr>
<tr>
<td>Robust</td>
<td>Grading by experts</td>
<td>Almost all pseudo-metrics (Ludewig and Lichter, 2013)</td>
</tr>
</tbody>
</table>

**Software Metrics: Blessing and Curse**

Application domains for software metrics:

- Cost metrics (including duration)
- Error metrics
- Volume/Size metrics
- Quality metrics

Being good wrt. to a certain metric in general is not an asset on its own. In particular critical: pseudo-metrics for quality (\( \rightarrow \) in a minute).

**Kinds of Metrics**

<table>
<thead>
<tr>
<th>Metric Type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedural</td>
<td>Measures total time spent per procedure</td>
</tr>
<tr>
<td>Structural</td>
<td>Measures the architecture of the software</td>
</tr>
<tr>
<td>Behavioral</td>
<td>Measures how the software behaves under different conditions</td>
</tr>
<tr>
<td>Static</td>
<td>Measures the structure of the code without execution</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Measures the behavior of the software during execution</td>
</tr>
</tbody>
</table>

**Benefits of Metrics**

- Improved decision-making
- Enhanced communication and understanding
- Better resource allocation
- Enhanced resource management
- Improved project planning
- Enhanced project control

**Challenges of Metrics**

- Difficulty in defining metrics
- Difficulty in collecting data
- Difficulty in interpreting results
- Difficulty in implementing metrics
- Difficulty in maintaining metrics
- Difficulty in changing metrics

**Benefits of Metrics**

- Improved decision-making
- Enhanced communication and understanding
- Better resource allocation
- Enhanced resource management
- Improved project planning
- Enhanced project control

**Challenges of Metrics**

- Difficulty in defining metrics
- Difficulty in collecting data
- Difficulty in interpreting results
- Difficulty in implementing metrics
- Difficulty in maintaining metrics
- Difficulty in changing metrics
### Some Objective Metrics, Base Measures


<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>Lines of Code</td>
</tr>
<tr>
<td>DLOC</td>
<td>Delivered Lines of Code</td>
</tr>
<tr>
<td>DIME</td>
<td>Dimensions in Management Engineering</td>
</tr>
</tbody>
</table>

#### Base Measures

- **Code Size**: LOC, DLOC
- **Program Size**: DIME
- **Headcount**: Number of employees
- **Unit Count**: Delivered units (modules, classes, etc.)
- **Effort**: Time spent on tasks
- **Cost**: Financial resources allocated
- **Quality**: Assessment of software characteristics
- **Health Condition**: Measure of software quality

#### Advantages

- Exact, reproducible, can be obtained automatically
- Not subvertable, plausible results, applicable to complex characteristics
- Yields relevant, directly usable statements on not directly visible characteristics

#### Disadvantages

- Hard to comprehend, pseudo-objective
- Assessment costly, quality of results depends on inspector
- Not always relevant, often subvertable, no interpretation

#### Examples

- **Productivity**: Average/median lines of code, productivity (lines per hour), .. .
- **Usability**: Ease of use, interface design, feedback from users
- **Severity of Error**: Criticality, risk, impact on system
- **Cost Estimation**: Following COCOMO
- **Quality Assessment**: Error weighting, error distribution

#### Notes

- *Some Subjective Metrics*
- *Objective Metrics*
- *Pseudo Metrics*
- *Metrics Defined for Version Control*
10 + 2 = 3

\( P(v) \rightarrow v \)

In a directed graph, the number of entries and exits are:

- Number of nodes: \( N \)
- Number of edges: \( E \)

\(|V| \cdot |E| = \sum \text{true positives} \cdot \sum \text{true negative} \cdot \sum \text{false positive} \cdot \sum \text{false negative} \)

The cyclomatic number is defined as:

\( V \cdot E \)

The cyclomatic complexity of a program is the number of entry or exit points. McCabe Complexity Cont'd

Definition.

\( \text{Cyclomatic Complexity} \) [McCabe, 1976]

\[ (v) \rightarrow v \]

Two main approaches:

- Expert review, grading
- Pseudo-metrics, derived measures

Due to the difficulty in measuring, people want to measure productivity (as defined above).

Pseudo-metrics, derived measures can be used to measure productivity.

Not robust, easily subvertible (see exercises).

Pseudo-metric: measure of program complexity.

M 

I 

C 

E 

G 

O 

S 

T 

Pseudo-Metrics Cont'd

Due to the difficulty in measuring, people want to measure productivity (as defined above).

Pseudo-metrics, derived measures can be used to measure productivity.

Not robust, easily subvertible (see exercises).

Pseudo-metric: measure of program complexity.

M 

I 

C 

E 

G 

O 

S 

T 

Pseudo-Metrics Cont'd
### If in doubt, use the simpler measure.

(vi) the metric.

Apply correction and detection of the pseudo-metric, how to compute the metric.

(v) Develop for all parameters of the definition.

(iv) Develop a model to be represented.

(iii) Fix a size error rate for error.

••

(ii) From each goal, derive goals which need to be answered to check whether the goal is reached.

(i) Identify significant effort causing errors during quality assurance, and errors found at least errors preventive maintenance).

...there seems to be an agreement that it is far more important to focus on empirical validation (or refutation) of the proposed metrics than to propose new ones, ...

...there is a relatively easy to compute number of paths, number of decision points. ...McCabe Complexity Cont'd
Now, Which Metric Should We Use?

It is often useful to collect some basic measures before they are actually required, in particular if collection is cheap:

- size of newly created and changed code,
- size of separate documentation,
- effort for coding, review, testing, verification, fixing, maintenance, ... for restructuring (preventive maintenance), ...
- errors, at least errors found during quality assurance, and errors reported by customer for recurring problems causing significant effort: is there a (pseudo-)metric which correlates with the problem?

Measures derived from the above basic measures:

- error rate per release,
- error density (errors per LOC),
- average effort for error detection and correction,
- ...

If in doubt, use the simpler measure.

LOC and changed lines over time (obtained by statsvn).  

References


