A Comparison

"Formal specifications may become for software engineers what, say, differential equations are for engineers of other fields."

Bertrand Meyer

Organisational Issues

- Two-fold lecture
  - For Computational Logic:
    - broadening the area towards software engineering
  - For “Vertiefungsrichtung Softwaretechnologie”
    - lecture on specialized topic in software engineering
    - Germans: please ask for help when required!
- Exercises:
  - A few theoretical exercises
  - A lot of project work in the lab (details announced later)
- Computational Logic students: examination through lab project
- Supporting material on the web:
  http://www-st.inf.tu-dresden.de/fs
  - Slides in PDF format
  - Lecture notes in PDF format
- Sorry, no lecture on December 4.
Outline (1)
1. Formal Specification of Software: Why and When?
   1.1 Purpose of Specifications
   1.2 Specification Examples
   1.3 Formality
   1.4 Overview of Formal Specification Languages
   1.5 Reminder on Mathematical Notation
2. Software Models in Software Engineering (mainly for CL!)
   2.1 Large Software Projects
   2.2 Models and Modeling from the Software Engineering Point of View
   2.3 Application Areas of Specification Languages
3. Algebraic Specification (Abstract Data Types)
   3.1 Syntax of Data Types: Signatures
   3.2 Algebraic Semantics: Heterogeneous Algebras
   3.3 Axiomatic Specification
   3.4 Deduction and Evaluation: Tools
   3.5 Structured Specifications
   3.6 Constructive vs. Observational Specification

Outline (2)
4. UML Class Diagrams as a Formal Specification Language
   4.1 Object-oriented Specification with the Unified Modeling Language (mainly for CL)
   4.2 UML Class Diagrams: Formal Syntax and Semantics
   4.3 Object-oriented Specification and Object-oriented Programs
5. The Object Constraint Language (OCL) for UML
   5.1 Object Constraints in OCL: Principles and Syntax
   5.2 Formal Semantics of OCL
   5.3 Collection Types
   5.4 Methodical Use of OCL
   6.1 Statecharts in UML: Sketch of Formal Semantics
   6.2 State Models and OCL
   6.3 Sequence Diagrams in UML: Sketch of Formal Semantics
7. Outlook
   7.1 Overview of the Formal Specification Language Z
   7.2 IT Security and Formal Specification
   7.3 Practical Usability of Formal Methods

Waterfall Model (and Effort Distribution)
Why do we need specifications?

- High level management of projects:
  - Development contracts rely on specifications.
  - Procurement decisions rely on specifications.
- Detailed management of projects:
  - Resource planning is intrinsically related to specifications.
  - The same holds for time planning, of course.
- Specifications are key to:
  - Software documentation,
  - Software understanding,
  - Software adaptation.

Languages for Specification

- Practice: natural language & graphical notations
- Functional aspect:
  - Global functional aspects: data flow diagrams, use case diagrams
  - Local functional aspects: “pseudo-code”
- Structural aspect: class diagrams, entity-relationship diagrams, block diagrams, physical deployment diagrams
- Dynamic aspect: state machines (with input/output)
- Behavioural aspect: interface definition languages, sequence diagrams, process descriptions
- Non-functional aspect: tables with measurable values
- Development process aspect: workflow description formalisms

Unified Modeling Language (UML)

- UML is a 2nd generation language for object-oriented modeling
- UML is an industry standard of OMG (Object Management Group)
- Originally developed by the company “Rational”
Specification from the “formal” viewpoint

- B. Meyer, On Formalism in Specifications:
  "Specification is the software lifecycle phase concerned with precise definition of the tasks to be performed by the system."

- For most of this lecture:
  - specification = problem, design or module/class usage
    - functional aspect

- There are other variants of specifications which can be treated formally as well.

The Seven Sins of the Specifier

- Noise (dt. Rauschen)
  - Remorse (dt. Reue bzw. Nachtrag)
- Silence (dt. Schweigen bzw. Auslassung)
- Contradiction (dt. Widerspruch)
- Overspecification (dt. Überspezifikation)
- Ambiguity (dt. Mehrdeutigkeit)
- Forward reference (dt. Vorwärtsverweis)
- Wishful thinking (dt. Wunschdenken)

Example Specification (1)

The program's input is a stream of characters whose end is signaled with a special end-of-text character, ET. There is exactly one ET character in each input stream. Characters are classified as:

- break characters – BL (blank) or NL (new line);
- nonbreak characters – all others except ET;
- the end-of-text indicator – ET.

A word is a nonempty sequence of nonbreak characters. A break is a sequence of one or more break characters. Thus, the input can be viewed as a sequence of words separated by breaks, with possibly leading and trailing breaks, and ending with ET.
Example Specification (2)

The program's output should be the same sequence of words as in the input, with the exception that an oversize word (i.e., a word containing more than MAXPOS characters, where MAXPOS is a positive integer) should cause an error exit from the program (i.e., a variable, Alarm, should have the value TRUE).

Up to the point of an error, the program's output should have the following properties:
1. A new line should start only between words and at the beginning of the output text, if any.
2. A break in the input is reduced to a single break character in the output.
3. As many words as possible should be placed on each line (i.e., between successive NL characters).
4. No line may contain more than MAXPOS characters (words and BLs)."

Meyer's mathematical specification (excerpt)

Short breaks. Let $a$ be a sequence of characters. We define $\text{SINGLE\_BREAKS}(a)$ as the set of subsequences of $a$ such that no two consecutive characters are break characters.

$\text{SINGLE\_BREAKS}(a) = \{ s \in \text{SUBSEQUENCE}(a) \mid \forall i \in [2..\text{length}(s) \cdot s(i-1) \in \text{BREAK\_CHAR} \Rightarrow s(i) \notin \text{BREAK\_CHAR} \}$

Next, we define $\text{COMPACTED}(a)$ as the subset of $\text{SINGLE\_BREAKS}(a)$ containing those sequences of maximum length:

$\text{COMPACTED}(a) = \{ s \in \text{SINGLE\_BREAKS}(a) \mid \text{MAX\_SET}(\text{SINGLE\_BREAKS}(a), \text{length}) \}$

UML Use Case Diagram

- format text
- <<include>
- break lines
- fill lines
- reduce breaks
- <<include>
Motivation for Formal Specification

• Formal specification languages intend to
  – achieve the same level of precision as a purely mathematical
    approach
  – apply structuring concepts known from programming
    languages
  – make specifications machine-processable
  – keep the specifications as readable as possible.

• In general, special education will always be required to deal with
  high-precision specifications.

Formality

Definition:
A language is called formal if it has a precisely defined
semantics that relates the syntactical form of a
specification in the language with a precisely defined
semantic domain.

A language is informal if it is not formal, i.e. if its
semantics rely on human interpretation of the
specifications, in particular on the chosen identifiers.

A language is semi-formal if it contains formal as well
as informal sublanguages or language layers.

• Warning: The above definition is the theoretician's view. For most
  practitioners, however, everything is formal which puts some
  restriction onto the form of the text/diagram being written!

Formal Specification Language

Definition
A formal specification language is a triple

\(<\text{Syn}, \text{Sem}, \text{Sat}>\)

where \(\text{Syn}\) and \(\text{Sem}\) are sets and \(\text{Sat} \subseteq \text{Syn} \times \text{Sem}\) is

\(\text{Syn}\) is called the syntactic domain of the language,
\(\text{Sem}\) is called its semantic domain and \(\text{Sat}\) is called its

satisfaction relation.
Application Areas for Formal Specification

- Highly reliable systems for safety-critical applications
  - Small but extremely important area
- Definition of standards
- "Local" enhancements of semi-formal specifications
  - "The best of both worlds."
- Contract-proof definition of "black box" software (componentware)
  - Increasing importance as software becomes the subject of lawsuits
- Semantics definition for semi-formal specification languages

History of Formal Specification Languages

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hoare logic</td>
<td>VDM</td>
<td>Z</td>
</tr>
<tr>
<td>Dijkstra calculus</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>Abstract data types</td>
<td>OBJ</td>
<td>CafeOBJ, Maude</td>
</tr>
<tr>
<td>Clear</td>
<td>Larch</td>
<td>CASL</td>
</tr>
</tbody>
</table>