Foundations of System Development

Prof. Dr. Martin Wirsing

04.10.2002

Contents

- **Data-oriented specification development**
  (Abstract data types, change of data structures and refinement, CASL)
- **Specification of dynamic systems**
  (Transition systems, model based specifications, invariants, Pre- and Post-conditions, Z, refinement)
- **Validation and refinement of concurrent, dynamic and reactive systems**
  (Features of processes, temporal logic, model checking, TLA, refinement)

System Development and Formal Specification

Prof. Dr. Martin Wirsing

04.10.2002

Introduction

Goals

- understanding the process of System Development and the importance of validation and verification
- show (technical) problems in System Development
- give an overview of formal methods for System Development

Related courses

- Object-Oriented Software Development
- Formal Object-Oriented Software Development
- Temporal Logic
- Model Checking Practical Course
- Software Engineering Practical Course

Introductory Examples

- Spacecraft Mariner I was destroyed on 22 of Juli 1962, 18–20 million dollars were lost. The program included following code-fragment:

  if *not* in radar contact with the rocket
  then
  do not correct its flight path

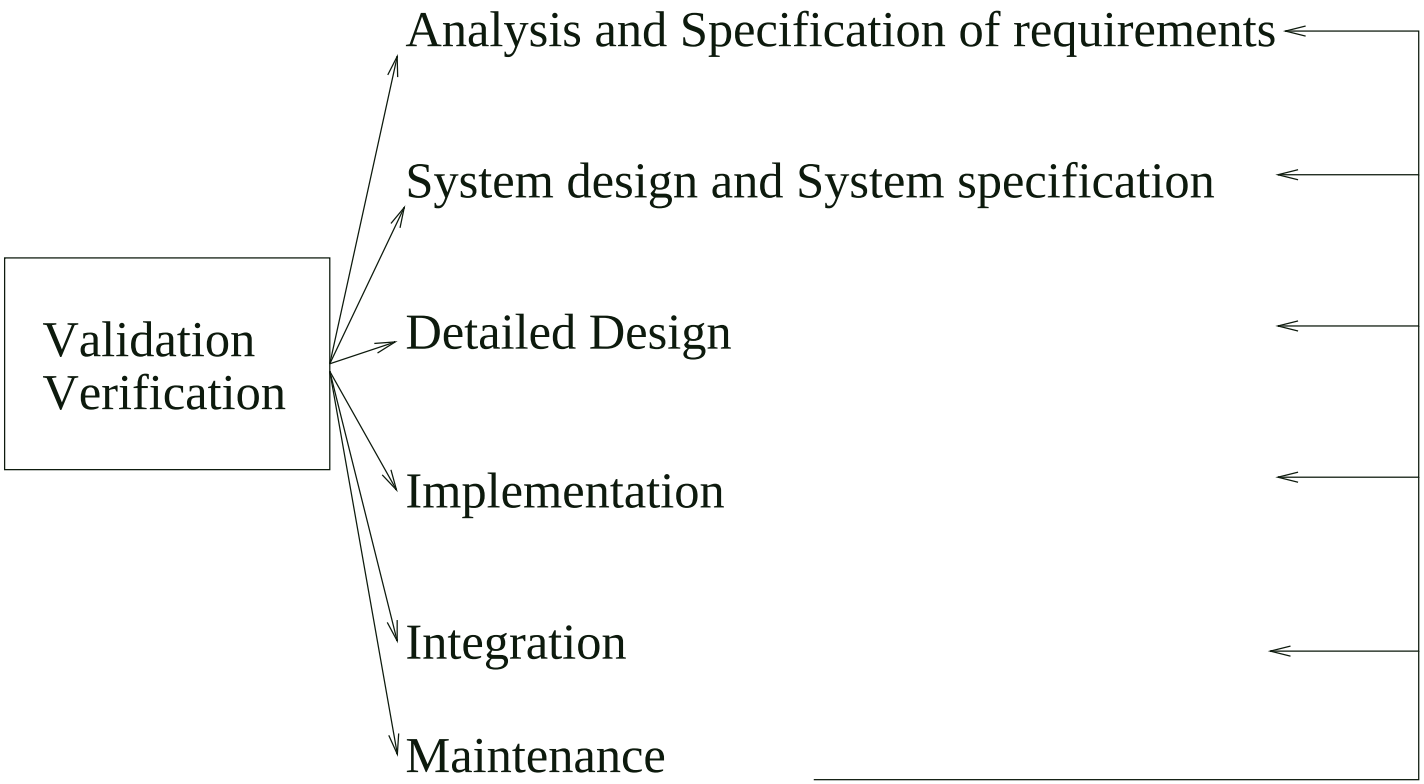
- Similar software bugs destroyed space rocket Ariane 5 in spring 1996
- The "Deutsche Telekom ltd." lost plenty money, because of a wrong calculation of telephone bills.
- The Windows Calculator (Version 3.1–8.3) had following computer bug:

$$\begin{array}{r} 2.01 \\ -2.00 \\ \hline 0.00 \end{array}$$

- A bug in Therac 25, a medical attendance tool for cancer, lead to erythema(burnings) and cases of death

The Software Lifecycle

System-Development-Process



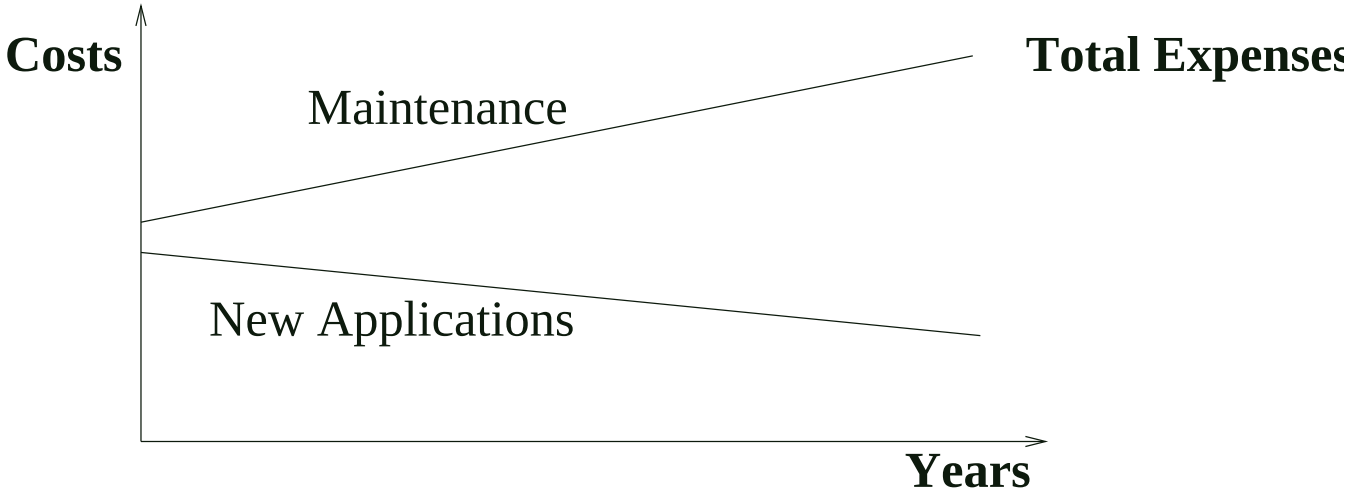
- **Analysis and Specification of Requirements.** Creates a document, that describes what the system should do, but not how.
- **System design and system specification.** Specification of system architecture and component tasks.
- **Detailed Design.** Refinement of the design to be able to generate code from it.
- **Implementation.** Implementing the designed components in a certain programming language.
- **Integration.** Assembling system components.
- **Maintenance.** Bug-fixing and changing the running system.
- **Validation.** Test whether a product fulfils the user-requirements ("Do we build the right product?").
- **Verification.** Test whether every process of the system development fulfils the intention of the previous process ("Do we build the product in the right way?").
- **Testing.** The process of executing a whole system to test if it fulfils the requirements. Is part of validation.

Costs of software within the lifecycle

While hardware costs become more and more cheaper, the costs for software explode. We differ two kinds of costs:

- Costs of Software-Construction (a),
- Costs for maintenance, for bug fixing or change of requirements (b).

The costs a) amount to 20% or less, the costs of maintenance b) amount to at least 80%.



Reusability

- Advantages:
  - saves time and costs,
  - less bug fixing,
  - longer used software is more robust,
  - Reuseability on all process levels particulary on the specification level.
- Problems:
  - "not invented here" Syndrom, which means that you do not trust software from others,
  - Reuseability is difficult to handle if you need to adapt the code to a new problem.
  - the development of generic software is expensive.

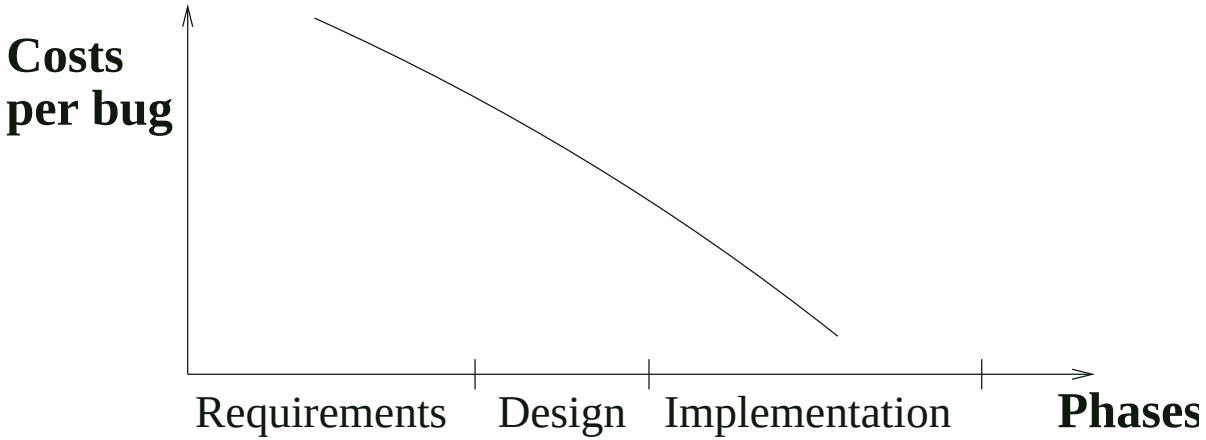
In the last years Application Programming Interfaces (API) were pushed. But through better notations (e.g.: Java Beans,EJB,COM) component development is more and more in demand.

System Development Problems

Development Process Step	Notation
Specification of Requirements	natural languages with diagrams and tables
Design Specification	graphic notation and in natural language; formal spec. in security-critical systems
Detailed Specification	Pseudo Code and natural language.
Implementation	Programming Languages

Problems:

1. Natural languages are not precise enough
2. Different notations in each phase. Errors at the interfaces between the phases.
3. Errors in the requirements definition phase are more expensive than errors in latter phases



Examples

- **Lack of Precision.**
  - "The interface for the system should be user-friendly.The virtual interface should stand on a few simple concepts, that are clear to understand and offer a simple user guidance."
  - "The volt value of the bulbs is always an integer between 3 and 6 Volt." Does that mean: "volt value ≥ 3 and volt value ≤ 6" or "volt value > 3 and volt value < 6" ?
- **Incompleteness** "The system should control the temperatures hourly, that are measured from the sensors at the running reactors. These values should be saved for the last three months. The command function of `AVERAGE` is to display the average temperature of one reactor for a certain day."

  What will be displayed if the average temperature of today is requested at 14 o' clock: error or average temperature until 14 o' clock?
- **Ambiguity.** "The operator label consists of an operator name and a password.The password consists of eight digits. It should be presented on a display and saved in a login file, when a operator logs on. Does this refer to the operator name or the password?"

Formal Methods in Computer Science

- **1960:** Formal Languages and State Machines
  finite state machines, Chomsky-Hierarchie, Turing machines, regular expressions.
- **1970:** Semantics of Programming Languages (Scott, Strachey)
  semantics of λ-calculus, denotational semantics.
- **1969–1975:** Proving Programs, Assertion Calculus (Dijkstra, Hoare, Floyd) describes properties of the state before and after the execution of the program.
- **1975:** Formal Specification (Guttag, ADJ)
  Abstract description of program features through axioms.
- **1970:** Formal Program Development (CIP, Burstall/Darlington, Hoare)
  Relationship between abstract levels of description, program transformation and change of data structures.
- **1980:** Temporal Logic for description of the dynamic behaviour of systems.
- **1990:** Integration of pragmatical graphical notations with formal technics.
- **1995:** Great results of temporal logic by using Model Checking for Verification.

Model-oriented and Algebraic Specifications

Generally, specifications are classified in model-oriented, algebraic and temporal specifications. Today the model-oriented specification languages VDM [Jones 90], B and Z are mainly used. The algebraic specifications are represented by Larch, LOTOS, Maude and a new language CASL. Temporal logic languages are LTL, CTL, TLA.

Model-oriented Specifications

describe imperative concepts through an assertion such as:
$\{\Phi\}P\{\Psi\}$

Factorial Function

e.g.: the assertion
$\{n \geq 0\}\ s := n; \quad Q\{s = fac(n)\}$
is a specification of a program Q, that calculates the factorial function. In this specification s is a variable, n a "mathematical" or "logical" variable, whose value is not allowed to be changed during the execution of Q. fac represents the factorial function.

Axiomatic Specifications

aim at the functional behaviour of computation structures. A data structure is described by its interface or signature, which means the declaration of its data types, operations and its characteristic properties.

Stack Example:

The computing structure of a stack is given by the name for the data type of the stack and by the data types of the stack elements. Characteristical functions are:

$$
\begin{array}{ll}
\text{empty:} & \text{Stack} \\
\text{push:} & \text{Data} \times \text{Stack} \rightarrow \text{Stack} \\
\text{top:} & \text{Stack} \rightarrow \text{Data} \\
\text{pop:} & \text{Stack} \rightarrow \text{Stack}
\end{array}
$$

Characteristical properties are:

$$\mathsf{top}(\mathsf{push}(d, s)) = d$$
$$\mathsf{pop}(\mathsf{push}(d, s)) = s$$

An imperative version of a stack specification includes the following **procedures**:

$New(s)$ assigns the empty stack to the variable s,

$Push(s, n)$ adds the symbol n to the stack s,

$Pop(s)$ removes the first stack symbol,

$Top(s, m)$ assigns the first symbol of the stack s to the variable m,

$IsEmpty(s, b)$ assigns the value $true$ to the variable b, if s is empty, otherwise b gets the value $false$.

Prejudices versus formal methods.

1. Formal methods make testing superfluous.
   Wrong: The transitions "real world to system" or also "informal description to formal description" can not be proven.
2. Formal methods make natural language descriptions unnecessary.
   Wrong: The analysis of requirements starts always with natural languages, extended by technical terms of the application. The natural language is used as comment for the specification.
3. A PhD is necessary to understand formal languages.
   Wrong: Formal specification is a formal notation like any other programming language.

Examples for proving systems

System	Developer	Remarks
LCF	R. Milner	Logic for Computable Functions
PVS	John Rushby, N. Shankar (SRI)	Interactiv prover for higher order logic, since 1980
Isabelle	L. Paulson, T. Nipkow	Advancement of LCF Cambridge, since 1985
Larch	DEC Palo Alto / MIT Guttag, Horning	since 1989
NUPRL	Constable, Cornell U.	Advancement of LCF, since 1986
Modelchecker	Different systems for proving the features of finite state machines	since 1990
Z/Eves	Dan Craigen	Analysis tool for Z, since 1990
HOL-CASL	Till Mossakowski, Univ. Bremen	Parser and Prover, since 2000

Summary

- System Development can be divided into process steps, starting with analysis of requirements and ending up in maintenance. In all process steps, validation and verification is necessary to prove the results.
- Lack of Precision, Incompleteness and Ambiguity are often sources of errors, that can be simpler discovered with formal analysis.
- Basic formal technics are:
  - axiomatic specifications to describe data and functional behaviour,
  - model-oriented specification to describe state-based behaviour,
  - temporal-logic specification to describe dynamic and reactive behaviour.