ETH Zürich
Chair of Software Engineering

Multirequirements

Bertrand Meyer

ETH Zurich
ИТМО
Eiffel Software

SECR, Moscow, 2 November 2011

About these slides

This printout is provided for the private convenience of SECR 2011 participants having attended my keynote. The keynote was centered around a live demo, obviously not included here.

The slides are not intended for use by, and would make little sense to, people not having attended the talk and the demo.

SEL at ITMO

Software **E**ngineering **L**aboratory |
Лаборатория Программной Инженерии

Создано в июне 2011

Ещё остаются открытые позиции!

- Аспиранты и Кандидаты (на полной ставке)
- Временные гранты ("sabbaticals") для исследователей, 2 до 6 месяцев

3

What this is about

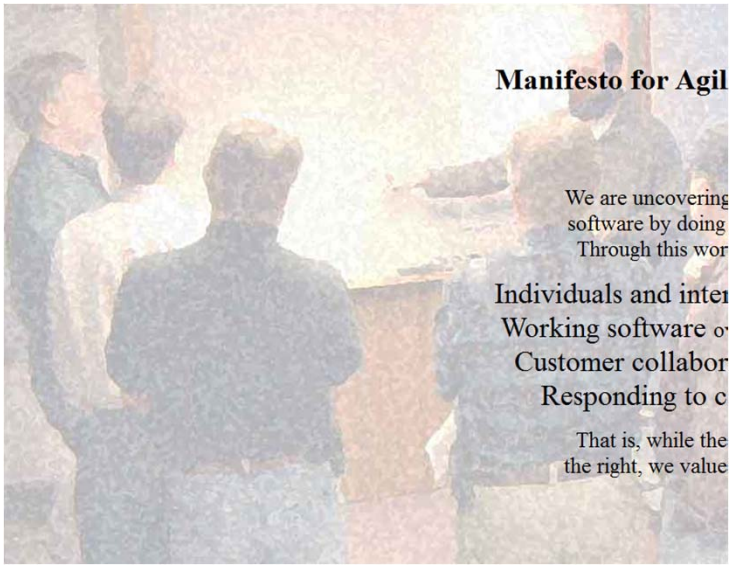
1. A reasoned look at the role of requirements
2. An introduction to object-oriented analysis
3. A reminder on basic laws of software engineering
4. A reminder on the importance of contracts
5. A primer on seamless development
6. The integration of these ideas into the concept of multirequirements

4

Big Upfront Requirements

5

Individual interactions over processes and tools?



6

Ariane 5, 1996

37 seconds into flight, exception in Ada program not processed; mission aborted

Exception was caused by an incorrect conversion: a 64-bit real value was incorrectly translated into a 16-bit integer

Reused program element was correct – for Ariane 4!

See: Jean-Marc Jézéquel and Bertrand Meyer

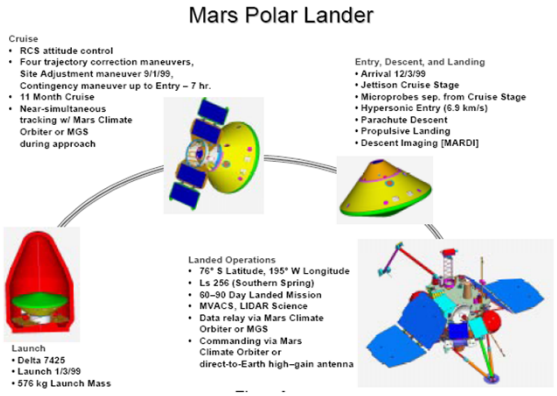
Design by Contract: The Lessons of Ariane

IEEE Computer, January 1997

7

While the Mars Climate Orbiter was lost

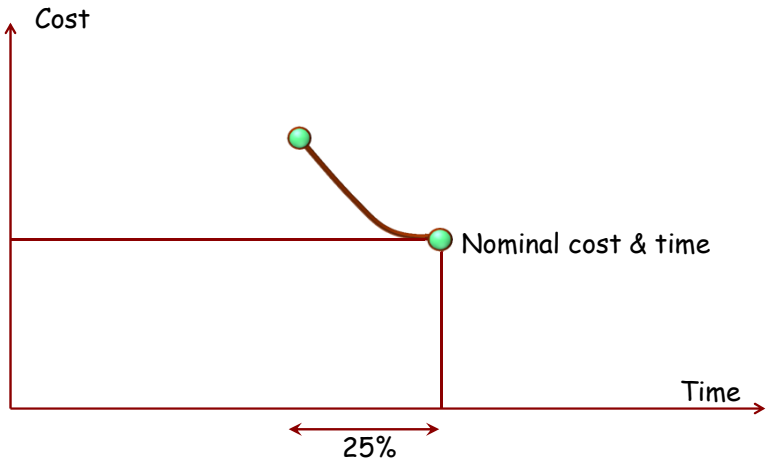
"The peer review … indicates that one team used English units (e.g., inches, feet and pounds) while the other used metric units… This information was critical to the maneuvers to place the spacecraft in the proper Mars orbit. "



8 8

Software engineering has laws

Boehm, McConnell, Putnam, Capers Jones…



9

Revolutionary ideas?

There have only been a few game-changing ideas in the history of software engineering:

- Structured programming
- Object technology
- ??? [CMM | agile | empirical SE] ???

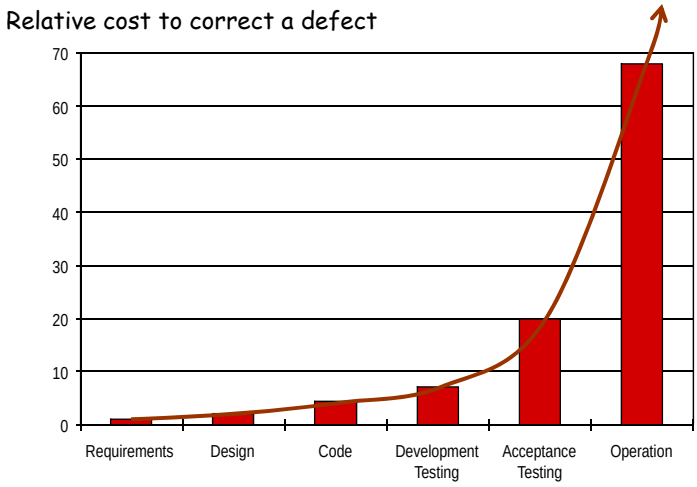
For every complex problem there is an answer that is clear, simple, and wrong.

H.L. Mencken

10

Statements about requirements: Boehm

Source: Boehm 81



11

Use cases and user stories...

... are not a proper tool for quality requirements

12

User stories

X	f (x)
0	0
1	1
2	4
3	9
4	16
...	...

13

Results of using use cases as requirements

14

Requirements change!

... but so will code, designs, documentation, and all other products of the software lifecycle!

What this means is **not** that we should forgo requirements, (even "Big Upfront Requirements") but that requirements are a software product:

- Subject to change process
- Subject to quality assurance
- Under configuration management

15

What use cases and user stories are good for

They are ways to **validate** the user requirements

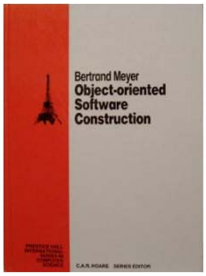
Use cases are to requirements (specifications) what tests are to programs

The task of requirements is to **abstract** from user stories

Object technology

Ask not first what the system does:

Ask what it does it to!



17

What is object-oriented analysis?

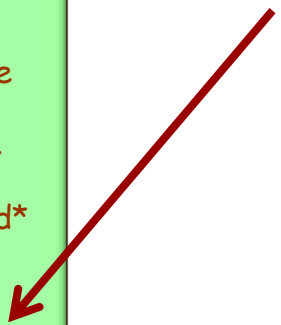
- **Classes** around object types (not just physical objects but also important concepts of the application domain)
- **Abstract Data Types** approach
- **Deferred** classes and features
- Inter-component relations: "**client**" and inheritance
- **Inheritance** — single, multiple and repeated for classification.
- **Contracts** to capture the *semantics* of systems: properties other than structural.
- **Libraries** of reusable classes

18

15 quality goals for requirements

- Justified
- Correct
- Complete
- Consistent
- Unambiguous
- Feasible
- Abstract
- Delimited

- Interfaced
- Readable
- Modifiable
- Verifiable
- Prioritized*
- Endorsed
- Traceable



Marked attributes are part of IEEE 830

19

Requirement styles

1. **"Structured English"**
   Informal but following some organization rules
   For guidance: use IEEE 830
2. **Formal**
   Based on mathematical formalism
   Subject to proofs and formal manipulation
3. **Graphical**
   e.g. UML

20

Multirequirements

Can we combine all three?

The basic problem will still be **traceability**

1. **"Structured English"**
   Informal but following some organization rules
   For guidance: use IEEE 830
2. **Formal**
   Based on mathematical formalism
   Subject to proofs and formal manipulation
3. **Graphical**
   e.g. UML

21

Technology

EiffelStudio

Diagram Tool

EIS (Eiffel Information System):

- Outgoing
- Incoming

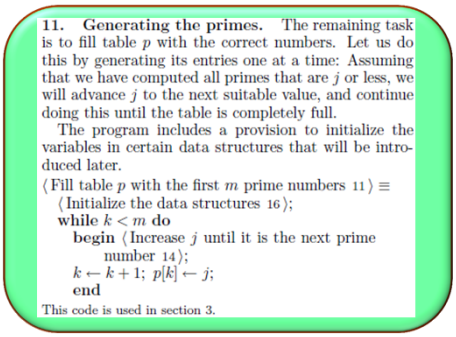
22

Related idea: Literate Programming

Knuth's proposal:

- Intersperse program text & natural language
- Record the history of a program development

Led to the development of TEX



11. **Generating the primes.** The remaining task is to fill table p with the correct numbers. Let us do this by generating its entries one at a time: Assuming that we have computed all primes that are j or less, we will advance j to the next suitable value, and continue doing this until the table is completely full.

The program includes a provision to initialize the variables in certain data structures that will be introduced later.

⟨ Fill table p with the first m prime numbers 11 ⟩ ≡
  ⟨ Initialize the data structures 16 ⟩;
  while $k < m$ **do**
    begin ⟨ Increase j until it is the next prime number 14 ⟩;
    $k \leftarrow k + 1$; $p[k] \leftarrow j$;
    end

This code is used in section 3.

Limitations:

- Top-down development, not O-O
- Documentation contains program, not other way around
- Tied to the personality of its inventor
- No support for change

23

Related idea: single-product principle in Eiffel

Single-Model Principle

All the information about a software system should be in the software text

Supported in EiffelStudio by Diagram Tool, multiple views of a class (contract, interface, inheritance…) & other techniques

24

Opposite idea: Model-Driven Development

Separate model from program

In principle:

- Program generated automatically

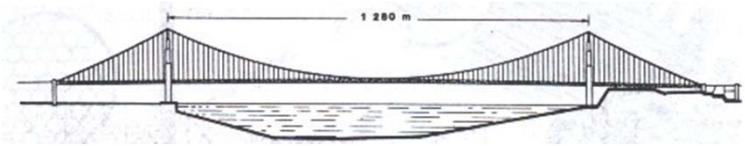
In reality:

- What about debugging?
- What about change?

25

Description and implementation

A bridge



A drawing of a bridge

Description-Implementation Porosity

A program text

```
private static boolean endsWith(String str, String suffix,
boolean ignoreCase) {
    if (str == null || suffix == null) {
        return (str == null && suffix == null);
    }
    if (suffix.length() > str.length()) {
        return false;
    }
    int strOffset = str.length() - suffix.length();
    return str.regionMatches(ignoreCase, strOffset);
 }
```

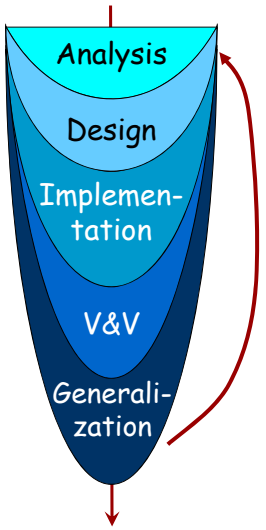
28

A ~~program text~~ specification (VDM)

```
AccNum = token;
CustNum = token;
Balance = int;
Overdraft = nat;
AccData :: owner : CustNum
        balance : Balance
state Bank of
        accountMap : map AccNum to AccData
        overdraftMap : map CustNum to Overdraft
inv mk_Bank(accountMap,overdraftMap) ==
        for all a in set rng accountMap & a.owner in set
                dom overdraftMap and
                a.balance >= -overdraftMap(a.owner)
```

Seamless development as practiced in Eiffel

- Single notation, tools, concepts, principles
- Continuous, incremental development
- Keep model, implementation and documentation consistent
- Reversibility: go back and forth



Example classes:

PLANE, ACCOUNT, TRANSACTION…

STATE, COMMAND…

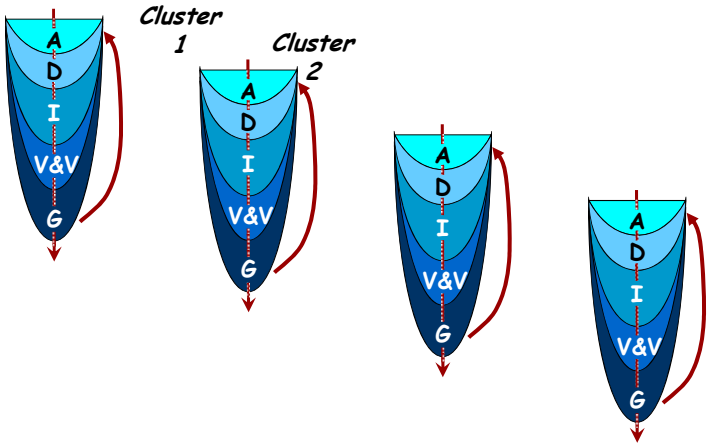
HASH_TABLE…

TEST_DRIVER…

TABLE…

30

The cluster model and risk minimization



31

IEEE standard

IEEE Std 830-1998
(Revision of
IEEE Std 830-1993)

IEEE Std 830-1998

IEEE Recommended Practice for Software Requirements Specifications

IEEE Computer Society

Sponsored by the
Software Engineering Standards Committee

20 October 1998 SH94654

32

Recommended structure

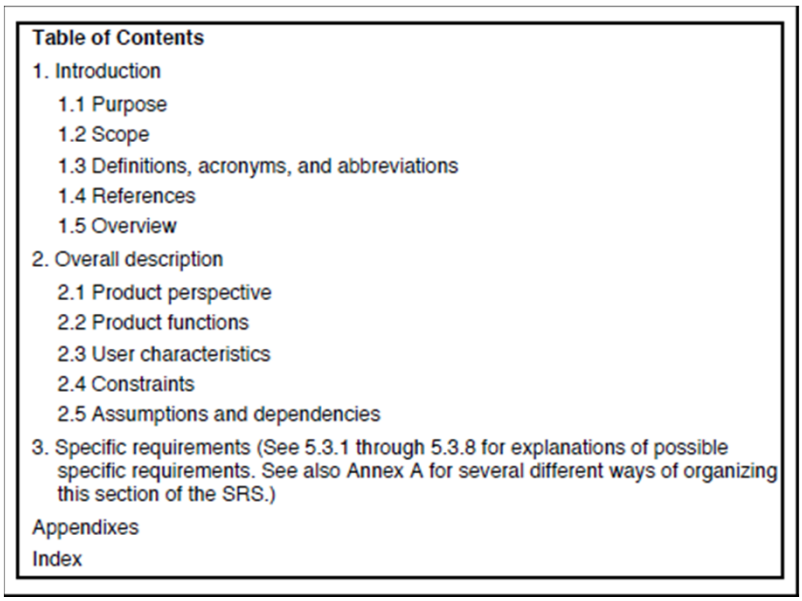


Table of Contents

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions, acronyms, and abbreviations
   - 1.4 References
   - 1.5 Overview
2. Overall description
   - 2.1 Product perspective
   - 2.2 Product functions
   - 2.3 User characteristics
   - 2.4 Constraints
   - 2.5 Assumptions and dependencies
3. Specific requirements (See 5.3.1 through 5.3.8 for explanations of possible specific requirements. See also Annex A for several different ways of organizing this section of the SRS.)

Appendixes

Index

33

Eiffel Information System

Ensure traceability between Eiffel software text and any other document (Word, PDF, …) with a notion of hyperlink

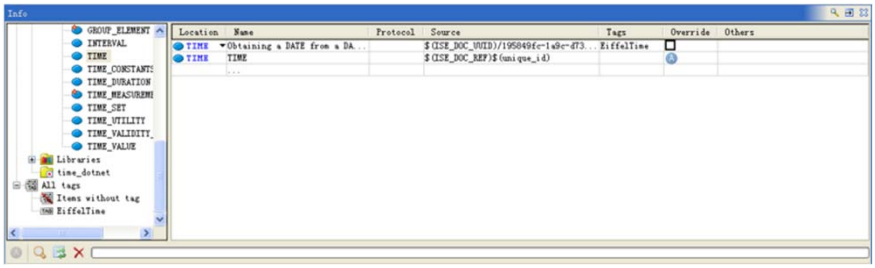
Two directions:

- Outgoing
- Incoming

34

Outgoing

EIS: "name=Project Requirement", "src=$(PROJ)/docs/requirements.pdf", "protocol=PDF", "nameddest=4.1", "tag=requirement"



Multirequirements

Integrate many views of requirements:

- Textual (English)
- Graphical (UML, BON)
- Precise (object-oriented text, in Eiffel)
- Rigorous (contracts)

Make sure all views are closely interconnected

Ensure traceability

Technology

Eiffel language: analysis, design, implementation, maintenance

Design by Contract

EiffelStudio 6.8

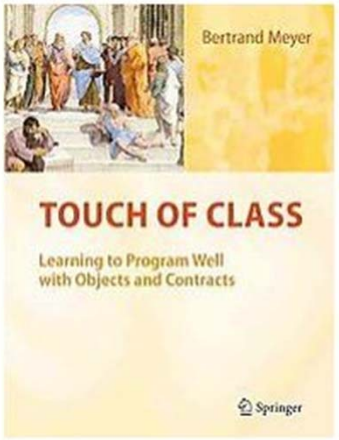
- Diagram tool
- Eiffel Information System

37

Teaching these ideas

At ETH: since 2003, beginning in the first year

Textbook: "Touch of Class", Springer



38

Multirequirements: the key ideas

- Seamless development
- Multiple views (textual, formal, graphical)
- Consistency and traceability
- Object-oriented
- Design by Contract as the formal method
- Eiffel contracts as the formal notation
- BON or UML as the graphical notation
- An IDE (here EiffelStudio) as requirements environment

39

For more

se.ethz.ch

www.eiffel.com

bertrandmeyer.com (blog)

40