

1

# Introduction to Programming

Bertrand Meyer

Lecture 23:  
From Programming to Software Engineering

Last revised 20 January 2006

Chair of Software Engineering Introduction to Programming – Lecture 23

2

## Software engineering (1)

The processes, methods, techniques, tools and languages for developing **quality** operational software.

Chair of Software Engineering Introduction to Programming – Lecture 23

3

## Software engineering (2)

The processes, methods, techniques, tools and languages for developing **quality** operational software that may need to

- Be of large size
- Be developed and used over a long period
- Involve many developers
- Undergo many changes and revisions

Chair of Software Engineering Introduction to Programming – Lecture 23

4

## Operating systems: source lines

Operating System	Year	Lines of code (millions)
Windows 3.1	1991	3 M
Windows NT	1993	4 M
Windows 95	1995	15 M
Windows 98	1998	18 M
Windows 2000	2000	40 M
Windows XP	2001	> 45 M

Chair of Software Engineering Introduction to Programming – Lecture 23

5

## Not just Windows

Operating System	Year	Lines of code (millions)
Linux	1992	10,000
Unix V7	1992	10,000
Windows 3.1	1991	3 M
Windows NT	1993	4 M
Solaris 7	1995	12 Mio
Windows 95	1995	15 M
Red Hat 6.2	1998	17 M
Windows 98	1998	18 M
Red Hat 7.1	2001	30 M
Windows 2000	2000	40 M
Windows XP	2001	> 45 M

Chair of Software Engineering Introduction to Programming – Lecture 23

6

## Not just operating systems

System	Year	Lines of Code (in millions)
Mercury	1960	~5
Gemini	1965	~10
Apollo	1970	~20
Lunar mission control	1970	~35
NASA space programs	1970	~45
Space shuttle	1975	~60
EWSD-APS DBP-14	1980	~10
EWSD-APS WM/4.2	1985	~20
Exchange systems for speech and data communication	1990	~30
EWSX V3 for broadband	1995	~50

Chair of Software Engineering Introduction to Programming – Lecture 23

## The basic issue

7

Developing software systems that are

- On time and within budget
- Of high immediate quality
- Possibly large and complex
- Extensible

## US Software industry, 1998

8

Standish Group: "Chaos" Report

250,000 Software projects, \$275 billions

- Project results:
  - 28% abandoned (1994: 31%)
  - 27% successful (1994: 16%)

The rest: "challenged"

- Smaller projects have higher chances of success

## NIST report on testing (May 2002)

9

- Financial consequences, on developers and users, of "insufficient testing infrastructure"

\$ 59.5 B.

(Finance \$ 3.3 B, Car and aerospace \$ 1.8 B. etc.)



## Software quality factors

10

- External: of interest to customers
  - Examples: Reliability, Extensibility
- Internal: of interest to developers
  - Examples: Modularity, Style

## Some internal factors

11

- Modularity
- Observation of style rules
- Consistency
- Structure
- ...

## Some external factors

12

Product quality (immediate):

- Reliability
- Efficiency
- Ease of use
- Ease of learning

Process quality:

- Timeliness
- Cost-effectiveness

Product quality (long term):

- Extensibility
- Reusability
- Portability

## External factors: reliability

13

Reliability = Correctness + Robustness + Integrity

Chair of Software Engineering Introduction to Programming – Lecture 23

## Reliability

14

- **Correctness**  
The system's ability to perform its functions according to the specification, in cases covered by the specification
- **Robustness**  
The system's ability to handle erroneous cases safely
- **Integrity**  
The system's ability to protect its users, its data and itself against hostile uses

Chair of Software Engineering Introduction to Programming – Lecture 23

## External factors

15

**Product quality (immediate):**

- Reliability
- Efficiency
- Ease of use
- Ease of learning

**Process quality:**

- Timeliness
- Cost-effectiveness

**Product quality (long term):**

- Extendibility
- Reusability
- Portability

Chair of Software Engineering Introduction to Programming – Lecture 23

## Software tasks

16

- Requirements analysis
- Specification
- Design
- Implementation
- Validation & Verification (V&V)
- Management
- Planning and estimating
- Measurement

Chair of Software Engineering Introduction to Programming – Lecture 23

## Requirements analysis

17

- Understanding user needs
- Understanding constraints on the system
  - Internal constraints: class invariants
  - External constraints

Chair of Software Engineering Introduction to Programming – Lecture 23

## Validation & Verification

18

- **Verification:** checking that you have built the system right  
(followed all rules)
- **Validation:** checking that you have built the right system  
(satisfied user needs)

Chair of Software Engineering Introduction to Programming – Lecture 23

## Software lifecycle models

19

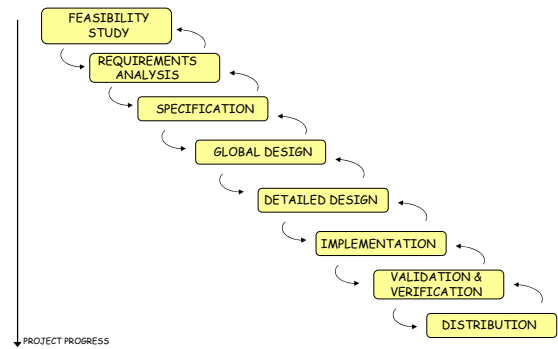
Describe an overall distribution of the software construction into tasks, and the ordering of these tasks

They are models in two ways:

- Provide an abstracted version of reality
- Describe an ideal scheme, not always followed in practice

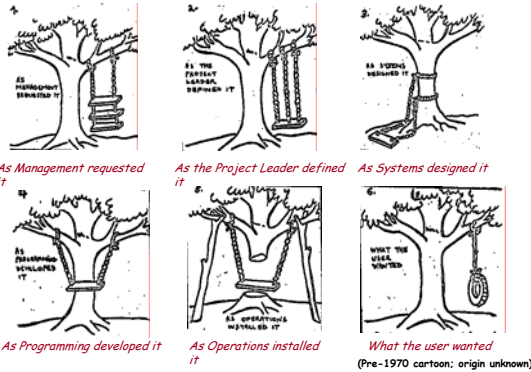
## The waterfall model (Royce, 1970)

20



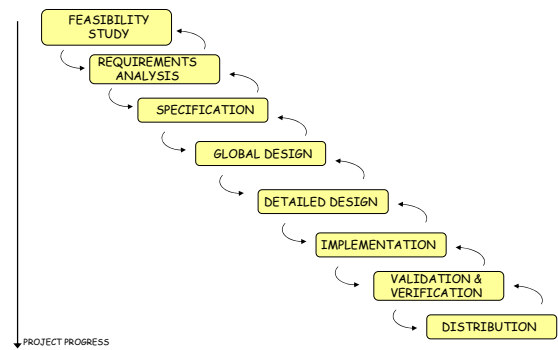
## Lifecycle: what not to achieve

21



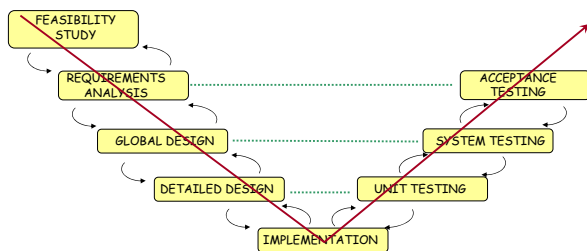
## The waterfall model

22



## A more realistic version

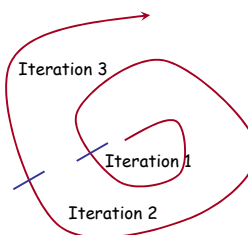
23



## The spiral model

24

- Apply a waterfall-like approach to successive prototypes



## ⦿ The problem with prototyping 25

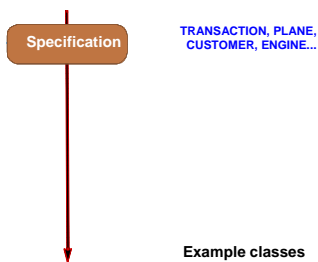
- Software development is hard because of the need to reconcile conflicting criteria, e.g. portability and efficiency
- A prototype typically sacrifices some of these criteria
- Risk of shipping the prototype

## ⦿ Seamless, incremental development 26

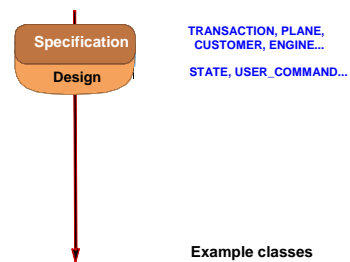
The Eiffel view:

- Single set of notation, tools, concepts, principles throughout
- Eiffel is as much for analysis & design as for implementation & maintenance
- Continuous, incremental development
- Keep model, implementation and documentation consistent
- Reversibility: can go back and forth

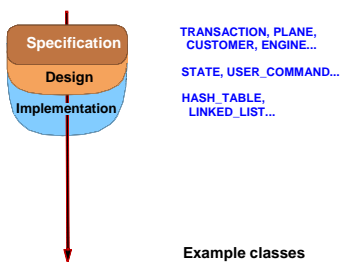
## ⦿ Seamless development (1) 27



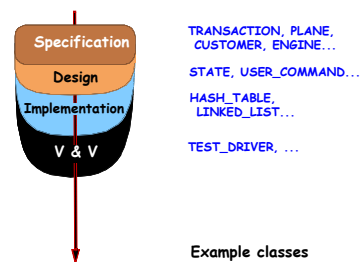
## ⦿ Seamless development (2) 28



## ⦿ Seamless development (3) 29

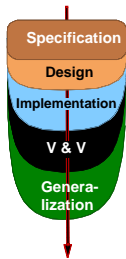


## ⦿ Seamless development (4) 30



## Seamless development (5)

31



TRANSACTION, PLANE,  
CUSTOMER, ENGINE...  
STATE, USER\_COMMAND...  
HASH\_TABLE,  
LINKED\_LIST...  
TEST\_DRIVER, ...  
AIRCRAFT, ...  
Example classes

## Generalization

32

- Prepare for reuse
- For example:
  - Remove built-in limits
  - Remove dependencies on specifics of project
  - Improve documentation, contracts...
  - Extract commonalities and revamp inheritance hierarchy
- Few companies have the guts to provide the budget for this

## Antoine de Saint-Exupéry

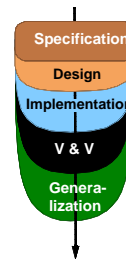
33

*It seems that the sole purpose of the work of engineers, designers, and calculators in drawing offices and research institutes is to polish and smooth out, lighten this seam, balance that wing until it is no longer noticed, until it is no longer a wing attached to a fuselage, but a form fully unfolded, finally freed from the ore, a sort of mysteriously joined whole, and of the same quality as that of a poem. It seems that perfection is reached, not when there is nothing more to add, but when there is no longer anything to remove.*

(Terre des Hommes, 1937)

## Seamless development

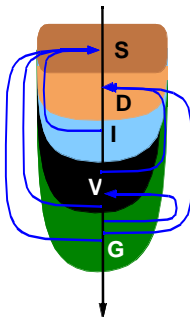
34



TRANSACTION, PLANE,  
CUSTOMER, ENGINE...  
STATE, USER\_COMMAND...  
HASH\_TABLE,  
LINKED\_LIST...  
TEST\_DRIVER, ...  
AIRCRAFT, ...  
Example classes

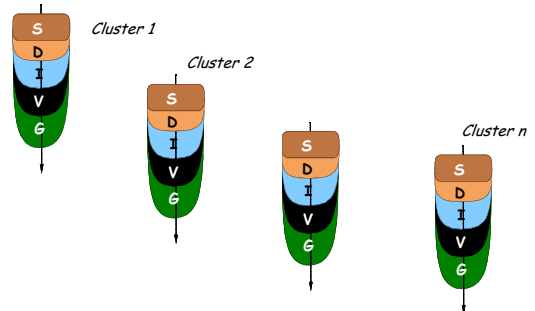
## Reversibility

35



## The cluster model

36



## Agile methods and extreme programming 37

- De-emphasize process and reuse
- Emphasize the role of tests to guide the development
- Emphasize the role of refactoring

## Validation and Verification 38

- Not just testing:
  - Static Analysis** tools explore code for possible deficiencies, e.g. uninitialized variables
- Should be performed throughout the process, not just at the end

## Software engineering tools 39

- Development environments (compiler, browser, debugger, ...): "IDE"
- Documentation tools
- Requirements gathering tools
- Analysis and design tools
- Configuration & version management (CVS, Source Safe...) (also "make" etc.)
- Formal development and proof tools
- Integrated CASE (Computer-Aided Software Engineering) environments

## Configuration management 40

- Aim: make sure that versions used for the various components of a system are compatible
- When poorly done, one of the biggest sources of software catastrophes
- Good tools exist today, e.g. CVS, Source Safe.
- Any project not using one of these tools is managed by a complete idiot

## Formal methods 41

- Use mathematics as the basis for software development
- A software system is viewed as a mathematical theory, progressively refined until directly implementable
- Every variant of the theory and every refinement step is **proved**
- Proof supported by computerized tools
- Example: *Atelier B*, security system of newest Paris Metro line

## Metrics 42

Things to measure:

- Product attributes: lines of code, number of classes, complexity of control structure ("cyclomatic number"), complexity and depth of inheritance structure, presence of contracts...
- Project attributes: number of people, person-months, costs, time to completion, time of various activities (analysis, design, implementation, V&V etc.)

Taking good measurements helps take good measures

## Cost models

43

- Attempt to evaluate cost of software development ahead of project, based on estimate of parameters
- Example: *COCOMO* (Constructive Cost Model), Barry Boehm

L: 1000 \* Delivered Source Instructions (KDSI)

Program type	Effort (pm)	Time
Application	$2.4 * L * 1.05$	$2.5 * pm * 0.38$
Utility	$3.0 * L * 1.12$	$2.5 * pm * 0.35$
System	$3.6 * L * 1.20$	$2.5 * pm * 0.32$

## Software reliability models

44

- Estimate number of bugs from
  - Characteristics of program
  - Number of bugs found so far
- Variant: "Fault injection"

## Project management

45

- Team specialties: customer relations, analyst, designer, implementer, tester, manager, documenter...
- What role for the manager: managerial only, or technical too?
- "Chief Programmer teams"

## Software engineering

46

- In the end it's code
- Don't underestimate the role of tools, language and, more generally, technology
- Bad management kills projects  
Good technology makes projects succeed

## Programming languages

47

- Not just for talking to your computer!
- A programming language is a tool for thinking

## A bit of history

48

- "Plankalkül", Konrad Zuse, 1940s
- Fortran (FORmula TRANSLator), John Backus, IBM, 1954
- Algol, 1958/1960

## Some FORTRAN code

49

```
100 IF (N) 150, 160, 170
150 A (I) = A (I) ** 2
    READ ("I6") N
    GOTO 100
C   THE NEXT ONE IS THE TOUGH CASE
160 A (I) = A (I) + 1
    READ ("I6") N
    GOTO 100
170 STOP
    END
```

## Algol

50

- International committee, Europeans and Americans; led to IFIP. Algol 58, Algol 60.
- Influenced by (and reaction against) FORTRAN; also influenced by LISP (see next). Recursive procedures, dynamic arrays, block structure, dynamically allocated variables
- New language description mechanism: BNF (for Algol 60).

## Algol W and Pascal

51

- Successors to Algol 60, designed by Niklaus Wirth from ETH
- Algol W introduced record structures
- Pascal emphasized simplicity, data structures (records, pointers).
- Small language, widely adopted for teaching.
- Helped trigger the PC revolution through Turbo Pascal from Borland (Philippe Kahn)

## C

52

- 1968: Brian Kernighan and Dennis Richie, AT&T Bell Labs
- Initially, closely connected with Unix
- Emphasis on low-level machine access: pointers, address arithmetic, conversions
- Frantically adopted by industry in the 80s and 90s

## Lisp and functional languages

53

- LISP Processing, 1959, John McCarthy, MIT then Stanford
- Fundamental mechanism is recursive function definition
- Numerous successors, e.g. Scheme (MIT)
- Functional languages: Haskell, Scheme

## LISP "lists"

54

- A list is of the form  $(x_1 x_2 \dots)$  where each  $x_i$  is either
- An atom (number, identifier etc.)
  - (Recursively) a list:

Examples:

- $()$
- $(x_1 x_2)$
- $(x_1 (x_2 x_3) x_4 (x_5 (x_6 () x_7)))$

$((x_1 x_2))$  is not the same as  $(x_1 (x_2))$

## LISP function application and definition

55

The application of function  $f$  to arguments  $a, b, c$  is written

$(f\ a\ b\ c)$

Example function definition (Scheme):

```
(define (factorial n)
  (if (eq? n 0)
      1
      (* n (factorial (- n 1)))))
```

Chair of Software Engineering

Introduction to Programming – Lecture 23



## Basic functions

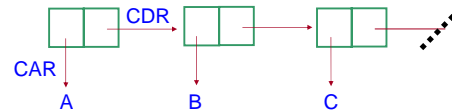
56

Let  $my\_list = (A\ B\ C)$

$(CAR\ my\_list) = A$

$(CDR\ my\_list) = (B\ C)$

$(CONS\ A\ (B\ C)) = (A\ B\ C)$



Chair of Software Engineering

Introduction to Programming – Lecture 23



## Functions working on lists

57

```
(define double-all (list)
  (mapcar
    '(lambda (x) (* 2 x)) list))
```

```
(define (mapcar function f)
  (if (null? ls) '()
      (cons
        (function (car ls))
        (mapcar function (cdr ls)))))
```

Chair of Software Engineering

Introduction to Programming – Lecture 23



## Object-oriented programming

58

- Simula 67: Algol 60 extensions for simulation, University of Oslo, 1967 (after Simula 1, 1964). Kristen Nygaard, Ole Johan Dahl
- Grew into a full-fledged programming language
- Smalltalk (Xerox PARC) added ideas from Lisp and innovative user interface ideas. Alan Kay, Adele Goldberg, Daniel Bobrow

Chair of Software Engineering

Introduction to Programming – Lecture 23



## "Hybrid" languages

59

- Objective-C, around 1984: Smalltalk layer on top of C
- C++, around 1985: "C with classes"

Made O-O acceptable to mainstream industry

Key moment: first OOPSLA, 1986

Chair of Software Engineering

Introduction to Programming – Lecture 23



## Java and C#

60

- C++ with enough restrictions to permit type safety and garbage collection
- Java initially marketed for applets in connection with the explosion of the Internet, 1995
- C# adds "delegates" (agent-like mechanism)

Chair of Software Engineering

Introduction to Programming – Lecture 23





- First version goes back to mid-eighties, first demonstrated at OOPSLA 86
- Emphasis on software engineering principles: information hiding, Design by Contract, static typing (through genericity), full application of O-O principles
- Has found its main application area in mission-critical industrial applications



End of lecture 23