Introduction to Programming

Bertrand Meyer

Last revised 3 February 2004
Lecture 26:
From Programming
to
Software Engineering
Software engineering (1)

The processes, methods, techniques, tools and languages for developing quality operational software.
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
Operating systems: source lines

- Windows 3.1: 3 M
- Windows NT: 4 M
- Windows 95: 15 M
- Windows 98: 18 M
- Windows 2000: 40 M
- Windows XP: > 45 M
Not just Windows

Lines of code (millions)


Unix V7: 10,000
Windows 3.1: 3 M
Windows NT: 4 M
Windows 95: 15 M
Windows 98: 18 M
Solaris 7: 12 Mio.
Windows 2000: 40 M
Windows XP: > 45 M
Red Hat 7.1 30 M
Red Hat 6.2 17 M

Windows 90:
Linux: 10,000

Chair of Software Engineering
Introduction to Programming – Lecture 26
Not just operating systems

Lines of Code (in millions)

- Mercury
- Gemini
- Apollo
- Lunar mission control
- Space shuttle
- NASA spacecraft programs
- EWSX V3 for broadband
- Exchange systems for speech and data communication
- EWSD-APS WM/4.2
- EWSD-APS DBP-14

The basic issue

Developing software systems that are

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

- **External**: of interest to customers
  - Examples: Reliability, Extendibility

- **Internal**: of interest to developers
  - Examples: Modularity, Style
Some internal factors

- Modularity
- Observation of style rules
- Consistency
- Structure
- …
Some external factors

Product quality (immediate):
- Reliability
- Efficiency
- Ease of use
- Ease of learning

Process quality:
- Timeliness
- Cost-effectiveness

Product quality (long term):
- Extendibility
- Reusability
- Portability
External factors: reliability

Reliability = Correctness + Robustness + Integrity
Reliability

- 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
**External factors**

**Product quality (immediate):**
- Reliability
- Efficiency
- Ease of use
- Ease of learning

**Process quality:**
- Timeliness
- Cost-effectiveness

**Product quality (long term):**
- Extendibility
- Reusability
- Portability
Software tasks

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

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

- Verification: checking that you have built the system right
  (followed all rules)

- Validation: checking that you have built the right system
  (satisfied user needs)
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)

FEASIBILITY STUDY

REQUIREMENTS
ANALYSIS

SPECIFICATION

GLOBAL DESIGN

DETAILED DESIGN

IMPLEMENTATION

VALIDATION & VERIFICATION

DISTRIBUTION
Lifecycle: what not to achieve

1. As Management requested it
2. As the Project Leader defined it
3. As Systems designed it
4. As Programming developed it
5. As Operations installed it
6. What the user wanted

(Pre-1970 cartoon; origin unknown)
The waterfall model

- FEASIBILITY STUDY
- REQUIREMENTS ANALYSIS
- SPECIFICATION
- GLOBAL DESIGN
- DETAILED DESIGN
- IMPLEMENTATION
- VALIDATION & VERIFICATION
- DISTRIBUTION

PROJECT PROGRESS
A more realistic version

FEASIBILITY STUDY

REQUIREMENTS ANALYSIS

GLOBAL DESIGN

DETAILED DESIGN

IMPLEMENTATION

ACCEPTANCE TESTING

SYSTEM TESTING

UNIT TESTING

ACCEPTANCE TESTING

A more realistic version
The spiral model

- Apply a waterfall-like approach to successive prototypes
The problem with prototyping

- 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

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)

Specification

TRANSACTION, PLANE, CUSTOMER, ENGINE...

Example classes
Seamless development (2)

Example classes

TRANSACTION, PLANE, CUSTOMER, ENGINE...

STATE, USER_COMMAND...
Seamless development (3)

Example classes

- TRANSACTION, PLANE, CUSTOMER, ENGINE...
- STATE, USER_COMMAND...
- HASH_TABLE, LINKED_LIST...
Seamless development (4)

Example classes

Specification
Design
Implementation

TRANSACTION, PLANE, CUSTOMER, ENGINE...

STATE, USER_COMMAND...

HASH_TABLE, LINKED_LIST...

TEST_DRIVER, ...
Seamless development (5)

Example classes

- TRANSACTION, PLANE, CUSTOMER, ENGINE...
- STATE, USER_COMMAND...
- HASH_TABLE, LINKED_LIST...
- TEST_DRIVER, ...
- AIRCRAFT, ...

Design

Implementation

Specification

Generalization

V & V
Generalization

- 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
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

Example classes

- TRANSACTION, PLANE, CUSTOMER, ENGINE...
- STATE, USER_COMMAND...
- HASH_TABLE, LINKED_LIST...
- TEST_DRIVER, ...
- AIRCRAFT, ...
Reversibility

Chair of Software Engineering
The cluster model

Cluster 1

Cluster 2

Cluster n
Agile methods and extreme programming

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

- 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

- 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

- 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

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

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

<table>
<thead>
<tr>
<th>Program type</th>
<th>Effort (pm)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>$2.4 \times L \times 1.05$</td>
<td>$2.5 \times pm \times 0.38$</td>
</tr>
<tr>
<td>Utility</td>
<td>$3.0 \times L \times 1.12$</td>
<td>$2.5 \times pm \times 0.35$</td>
</tr>
<tr>
<td>System</td>
<td>$3.6 \times L \times 1.20$</td>
<td>$2.5 \times pm \times 0.32$</td>
</tr>
</tbody>
</table>

$L$: 1000 * Delivered Source Instructions (KDSI)
Software reliability models

- Estimate number of bugs from
  - Characteristics of program
  - Number of bugs found so far

- Variant: “Fault injection”
Project management

- 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

- 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

- Not just for talking to your computer!

- A programming language is a tool for thinking
A bit of history

- “Plankalkül”, Konrad Zuse, 1940s
- Fortran (FORmula TRANSlator), John Backus, IBM, 1954
- Algol, 1958/1960
Some FORTRAN code

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
- 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

- 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)
- 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

- LISt Processing, 1959, John McCarthy, MIT then Stanford

- Fundamental mechanism is recursive function definition

- Numerous successors, e.g. Scheme (MIT)

- Functional languages: Haskell, Scheme
A list is of the form \((x_1 \ x_2 ... \) where each \(x_i\) is either

- An atom (number, identifier etc.)
- (Recursively) a list:

Examples:
- ()
- (x1 x2)
- (x1 (x2 x3) x4 (x5 (x6 () x7)))

\(((x_1 \ x_2))\) is not the same as \((x_1 \ (x_2))\)
LISP function application and definition

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

\[(f \ a \ b \ c)\]

Example function definition (Scheme):

\[
\text{(define (factorial n)}
\begin{align*}
\text{if (eq? n 0)} & \quad 1 \\
\text{(\ast n (factorial (- n 1)))})
\end{align*}
\)
Basic functions

Let \( \text{my\_list} = (A \ B \ C) \)

\((\text{CAR my\_list}) = A\)
\((\text{CDR my\_list}) = (B \ C)\)

\((\text{CONS A (B C)}) = (A \ B \ C)\)
Functions working on lists

\[\text{(define double-all (list)}\]
\[\text{(mapcar}\]
\[\text{'(lambda (x) (\ast 2 x)) list}))))\]

\[\text{(define (mapcar function f)}\]
\[\text{(if (null? ls) '()}\]
\[\text{(cons}\]
\[\text{(function (car ls))}\]
\[\text{(mapcar function (cdr ls))) ) )}\]
Object-oriented programming


- 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
"Hybrid" languages

- 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
Java and C#

- 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)
- 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
Evaluation

- Extremely encouraging results
- Lecture videos appreciated
- Completed textbook will help...
- Advance slides will help...
- Better TRAFFIC will help...
- German-language lectures might help...

... but it won’t ever be the first time again!
Thank you *very* much...

... to

- Students
- Assistants (including Hilfsassistenten), esp. Michela & Susanne
- Patrick Schoenbach for TRAFFIC
- N.E.T.
- FILEP project (rectorate)
- all on mailing list
End of lecture 26