CIS 422/522
Software Methodologies I

Software Engineering (van Vliet)

Chapter 1

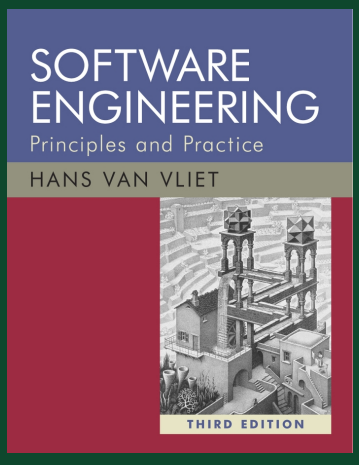
Professor: Juan J. Flores
jflore10@uoregon.edu

UNIVERSITY OF OREGON

1

Software Engineering

Principles and Practice
Third Edition



Hans van Vliet
Vrije Universiteit

ISBN: 978-0-470-3146-9

UNIVERSITY OF OREGON

2

Chapter 1
Introduction

3

Software engineering
The beginning

- 1968/69 NATO conferences: introduction of the term Software Engineering
- Idea: software development is not an art, or a bag of tricks
- Build software like we build bridges

4

4

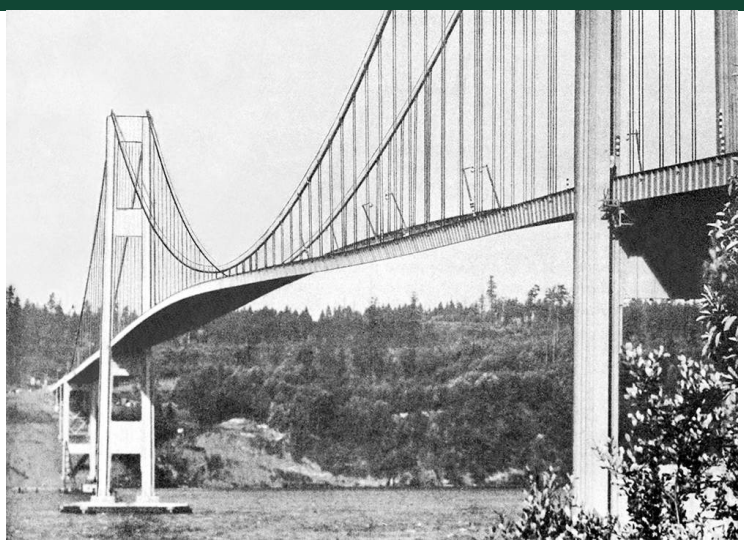
Current status

- a lot has been achieved
- but …
- some of our bridges still collapse

5

5

Tacoma Narrows bridge



6

6

Tacoma Narrows bridge

7

7

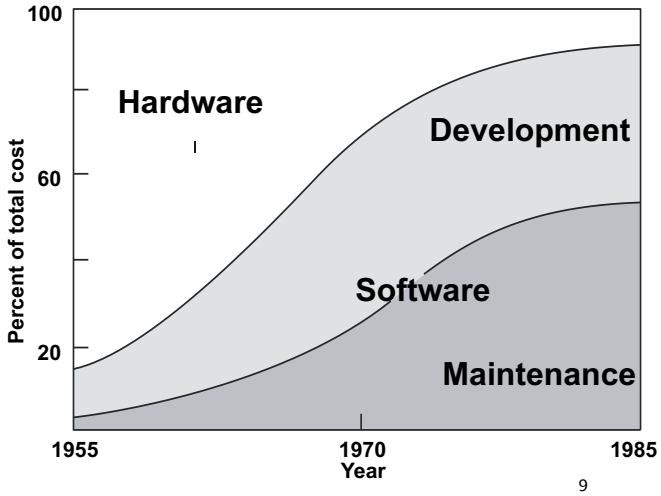
Same bridge, a while later

8

8

Relative distribution of software/hardware costs

- Why does software maintenance cost so much?



9

9

Definition (IEEE)

Software Engineering is the application of
- Systematic
- Disciplined
- Quantifiable

approach to the
- Development
- Operation
- maintenance

of software

=> application of engineering to software

10

10

Central themes

- SE is concerned with BIG programs
- Complexity is an issue
- Software evolves
- Development must be efficient
- Work as a team
- Software must effectively support users
- Involves different disciplines
- SE is a balancing act

11

11

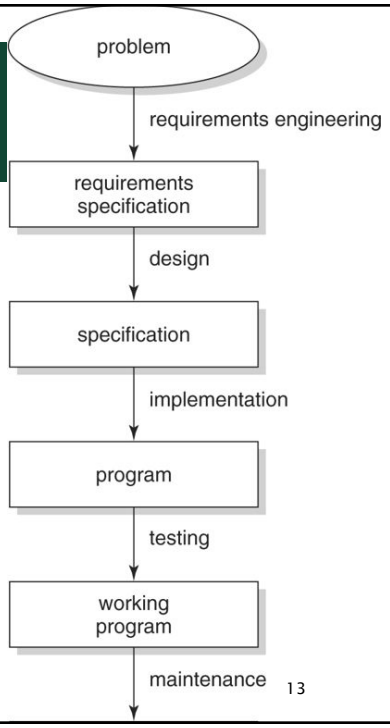
Building software ~ Building bridges?

- Yes, and no
- Software is logical, rather than physical
- Progress is hard to see (speed ≠ progress)
- Software is not continuous

12

12

Simple life cycle model



13

13

Requirements Engineering

- Yields a description of the desired system:
  - which functions
  - possible extensions
  - required documentation
  - performance requirements
- Includes a feasibility study

A **feasibility study** is an **analysis** that takes all of a project's relevant factors into account—including economic, technical, legal, and scheduling considerations—to ascertain the likelihood of completing the project successfully.

- Resulting document: requirements specification

14

14

Design

- Earliest design decisions captured in *software architecture*
- Decomposition into parts/components
- What are the functions of and interfaces between, those components?
- Emphasis on *what* rather than *how*
- Resulting document: *specification*

15

15

Implementation

- Focus on individual components
- Goal: a working, flexible, robust, … piece of software
- *Not a bag of tricks*
- Programming languages have modules and/or classes

16

16

Testing

- Does the software do what it is supposed to do?
- Are we building the right system? (*validation*)
- Are we building the system right? (*verification*)
- Start testing activities in phase 1, on day 1

17

17

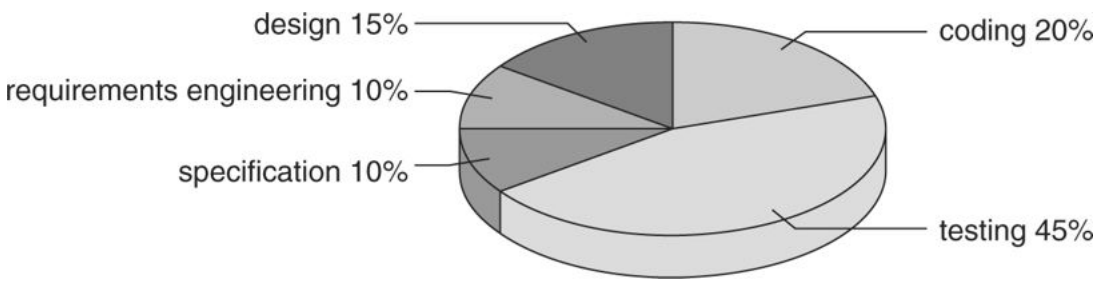
Maintenance

- Correcting errors found after the software has been delivered
- Adapting the software to changing requirements, changing environments, ...

18

18

Global distribution of effort



19

19

Global distribution of effort

- Rule of thumb: 40-20-40 distribution of effort
- Trend:
  - enlarge requirements specification/design slots
  - reduce test slot
- Maintenance alone consumes 50-75% of total effort

20

20

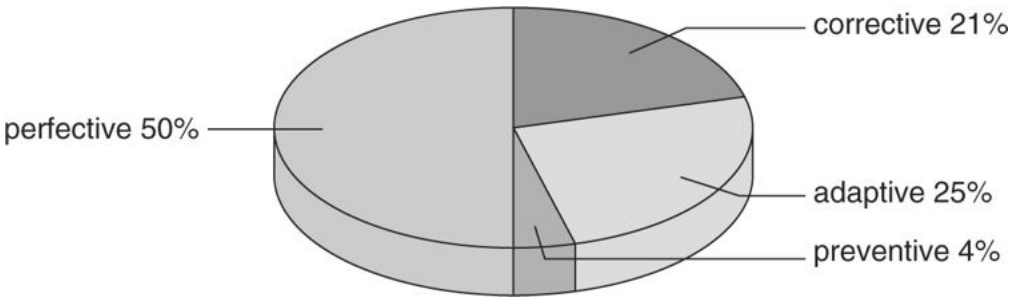
Kinds of maintenance activities

- *Corrective maintenance*: correcting errors
- *Adaptive maintenance*: adapting to changes in the environment (both hardware and software)
- *Perfective maintenance*: adapting to changing user requirements

21

21

Distribution of maintenance activities



22

22

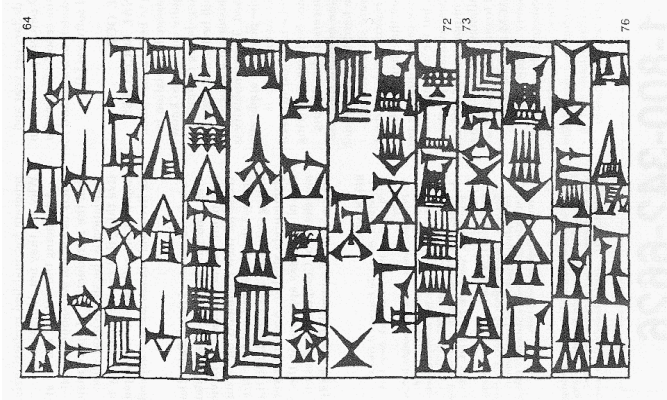
Ponder this situation

- You are a tester
- The product you are testing does not yet meet the testing requirements
- Your manager wants to
  - Ship the product now,
  - Continue testing so that the next release will meet the testing requirements.
- What do you do?
  - Discuss the issue with your manager?
  - Talk to the manager's boss?
  - Talk to the customer?

23

23

Hammurabi's Code



24

24

Hammurabi's Code (translation)

64: If a builder build a house for a man and does not make its construction firm, and the house which he has built collapses and causes the death of the owner of the house, that builder shall be put to death.

73: If it causes the death of a son of the owner of the house, they shall put to death a son of that builder.

25

25

Software Engineering Ethics - Principles

- Act consistently with the public interest
- Act in a manner that is in the best interest of the client and employer
- Ensure that products meet the highest professional standards possible
- Maintain integrity in professional judgment

26

26

Software Engineering Ethics - Principles

- Managers shall promote an ethical approach
- Advance the integrity and reputation of the profession
- Be fair to and supportive of colleagues
- Participate in lifelong learning and promote an ethical approach

27

27

Quo Vadis?

- It takes at least 15-20 years for a technology to become mature (true for computer science: UNIX, relational databases, structured programming, …)
- Software engineering has made tremendous progress
- There is no silver bullet, though

28

28

Recent developments

- Rise of agile methods
- Shift from producing software to using software
- Success of Open Source Software
- Software development becomes more heterogeneous

29

29

The Agile Manifesto

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

30

30

Producing software ⇒ Using software

- Builders build pieces, integrators integrate them
- Component-Based Development (CBSD)
- Software Product Lines (SPL)
- Commercial Off-The-Shelves (COTS)
- Service Orientation (SOA)

31

31

Open Source: crowdsourcing

1. Go to LEGO site
2. Use CAD tool to design your favorite castle
3. Generate bill of materials
4. Pieces are collected, packaged, and sent to you

5. Leave your model in LEGO's gallery
6. Most downloaded designs are prepackaged

- ❑ No requirements engineers needed!
- ❑ Gives rise to new business model

32

32

Heterogeneity

- Old days: software development department had everything under control
- Nowadays:
  - Teams scattered around the globe
  - Components acquired from others
  - Includes open source parts
  - Services found on the Web

33

33

SE Principles

- Build with and for reuse
- Define software artifacts rigorously
- Establish a software process that provides flexibility
- Manage quality as formally as possible
- Minimize software components interaction
- Produce software in a stepwise fashion
- Plan for change and manage it
- Make tradeoffs explicit and document them
- Identify uncertainty manage it

34

34

SUMMARY

- Software engineering is a balancing act, where trade-offs are difficult
- Solutions are not right or wrong; at most they are better or worse
- Most of maintenance is (*inevitable*) evolution
- There are many life cycle models, and they are all *models*