



CpSc 872 Software Design & Implementation



History

Historical Overview of Software Engineering

2

Historical overview Computing evolution: 1st phase

- Batch orientation
- Limited computer usage
- Hardware much more expensive than software: **bundled**
- Custom software



Computing evolution: 2nd phase

- Multi-user
- Real-time systems
- DBMS
- Product software

4

Computing evolution: 3rd phase

- Distributed systems
- Embedded "intelligence"
- hardware is **cheap**
- Consumer impact

5

Computing evolution: 4th phase

- Powerful desk-top system
- **OO technology**
- Expert systems
- AI
- Parallel computing
- Network computing



6

Software today ^{Everyone's a user!}

- computer programs are ubiquitous
- software realizes the potential of hardware
- mega-software companies
- software superstores
- Internet



7

What is software engineering ?

- the establishment and use of sound engineering principles to obtain economical software that is reliable and works efficiently on real machines.
[Fritz Bauer, 1969]

8

What is software engineering ?

- A **multi-person** construction of **multi-version** software [Parnas 1987]



9

What is software engineering?

- (1) the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; the application of engineering to software.
- (2) the study of approaches to (1)
[IEEE 1993]

10

Why do we need Software Engineering

- Large software projects demand it
- To control Software and maintenance costs
- To “**recycle**” software

11

Large Software Projects

- Initially projects were small
 - Single programmer
 - Limited communication required
 - Finite Development Period
 - Defined project objectives
 - Problem understood by **one** person

12

Large Software Projects

- Gradually projects became Larger
 - Varied project members (Managers, Analysts, Programmers)
 - Extensive interpersonal communication required
 - Long Development Period
 - Non Specific project objectives
 - Problem entirely understood by **no one person**

13

Large Software Projects

- Projects universally over budget and behind schedule, so called "**Software Crisis**"
- Briggs-Meyers estimation rule
- Software Engineering was an attempt to control and **solve** these problems

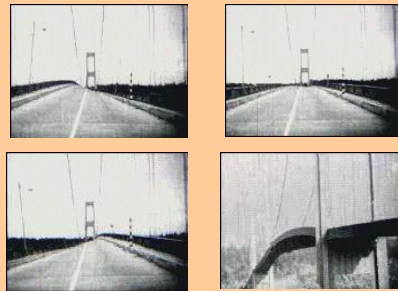
14

Recycling Software

- Software reuse
 - Avoids re-inventing the wheel
 - Economical
- Accomplished via engineering processes
 - Documentation
 - Quality
 - OO
 - **Components**

15

Hardware does fail: The Tacoma Narrows



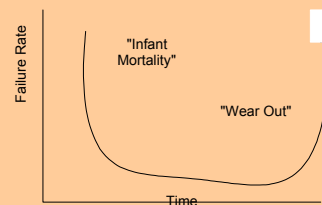
16

Software vs Hardware

- Software is developed or engineered- NOT manufactured
- Frequently software is custom built
- Software doesn't wear out
 - Hardware Bathtub failure curve
 - Software steady state failure curve (with spikes)

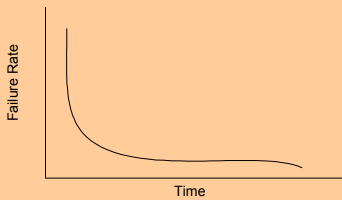
17

Hardware failure curve



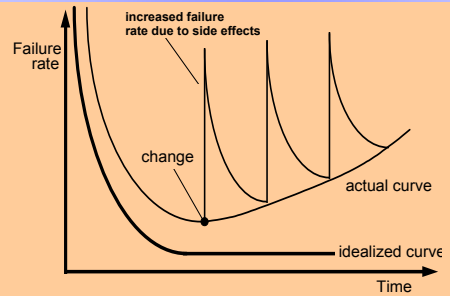
18

Software Failure rate (idealised)



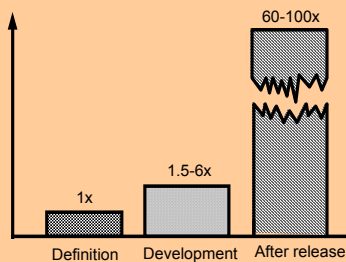
19

The effects of change:



20

The Cost of Change



21

The ideal Software Engineer

- Must be a good Programmer
- Familiar with several design approaches
- Know several computer languages
- Have excellent communication skills
- Have the ability to manage
- Problem solver
- Professional



22

The Software Process



23

The Software Process

- The process we follow to build, deliver, and evolve the software product, from the inception of an idea all the way to the delivery and the final retirement of the system, is called a *software production process*.



24

The Software Process

- A process is essential to any discipline whose goal is to produce products
- Software development differs from hardware
- QUES: How to organise a software production process?
- We put forward the following models...

25

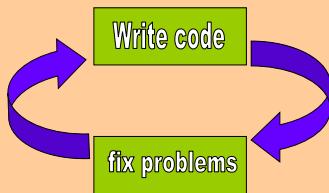
Software Life Cycle Models

- code-and-fix
- The Waterfall model (p. 47 in UML book)
- The Prototyping model
- Incremental model/Iterative Model
- Spiral Model (p. 49 in UML book)
- Components model
- Fourth generation model
- Formal Methods model

*aka: classic,
linear sequential*

26

Code-and-fix



27

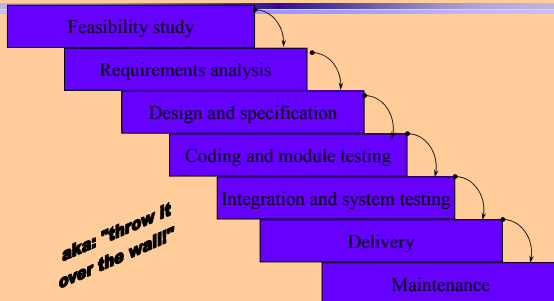
The Waterfall Model



- First appeared in 1950's, popular in 1970's
- Process structured as a cascade of phases where output of one is input of next.
- Many variants of model depending on
 - Organisation
 - Specific project
- Underlying phases same for all

28

The Waterfall Model



*aka: "throw it
over the wall"*

29

Feasibility Study



- Analyse the problem
 - Problem definition
 - Identify alternative solutions
 - Resources: Costs & Delivery dates for each

30

Requirements Analysis & Specification

- RA: what must the software do?
- Requirements Specification Document
 - Documents the results of the analysis
 - Customer uses it to verify expectations

31

Design and Analysis

- Description of software architecture
 - Decompose the system into modules or components
 - Function of each module
 - Relationships between modules

32

Coding and Module testing

- Writing the program
 - Originally the only recognised phase
 - Subject to company wide standards
- Unit testing



33

Integration and system testing

- Integration
 - Tests the connectivity of modules/units
 - integration testing tests as application is built
- System testing
 - High level testing of the assembled application
 - system is put into actual use but with understanding and forgiving users (alpha testing).

34

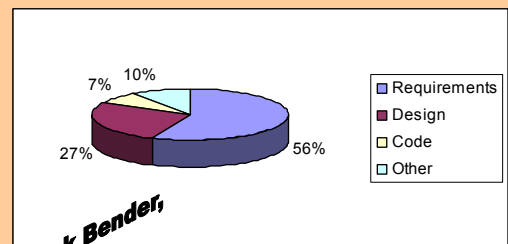
Delivery and Maintenance

- Delivery
 - Stage 1, distribute to select group of customers (Beta testing)
 - Stage 2, official release.
- Maintenance: any change to the software



35

Source of software errors



36

Conclusions on Maintenance

- Requirements analysis big source of error.
 - Difficult to capture exactly
 - Change over time
- Many errors not found until maintenance
 - Very costly to fix at this stage
 - One of SE's goals is to catch faults earlier

37

An evaluation of the model

- Two fundamental contributions
 - Firstly, the SDP should be subject to discipline, planning, and management
 - Secondly, coding should wait until requirements, analysis and design

38

An evaluation of the model

- Waterfall model can only be approximated in practice. It is characterised as
 - Linear
 - Rigid
- No room for feedback during cycle

39

Conclusion about waterfall model

- While the waterfall model does have its weaknesses it has given discipline and structure to software development

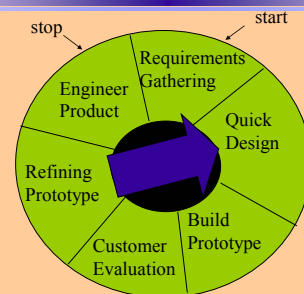
40

The Prototyping model

- Based on the "Do it twice" principle [Brooks 1975]
- First version is a "quick design" prototype to
 - assess the feasibility of the product
 - verify and refine the requirements
- Prototype is evaluated and requirements refined
- Real system is then developed

41

The Prototyping model



42

An evaluation of the model

- Model goes some way to address the rigidity problems of Waterfall
- Does not eliminate gap between definition of requirements and delivery of software
- No mechanism for anticipating change

43

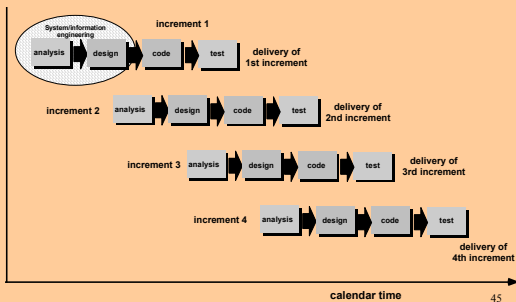
Software process models; the incremental model:

- apply **analysis, design, code, test** iteratively, increasing functionality at each iteration
- e.g.: word processor developed incrementally might deliver basic file mgmt and editing in first increment, more sophisticated editing with second increment
- first deliverable is usually “core” product

ACDSee

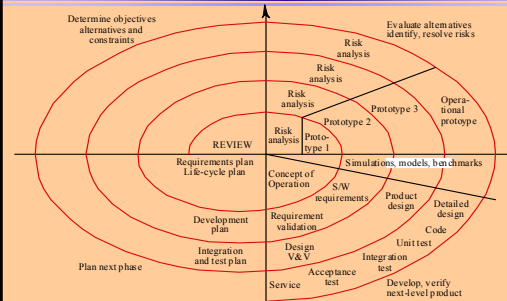
44

Incremental



45

Boehm, 1988 Spiral model



46

Software process models; component assembly model:

- OO provides the technical framework for this model
- OO paradigm emphasizes the creation of classes => components
- properly designed components are reusable across different applications & platforms

47

Software process models; formal methods model:

- mathematical specification of computer software
- use formal methods to: **specify, develop and verify** a computer based system by applying rigorous mathematical notation
- not a mainstream approach yet offers the promise of **defect-free** software

48