Software Engineering Fundamentals

Introduction

Dr. Petru Florin Mihancea

Based on:
I. Sommerville - Software Engineering 8, Ch.1 Introduction
R. Pressman - Software Engineering, Ch. 1 The Product

1. What is Software Engineering ?

The beginning ...

The "software crisis" started in the 60's

- **Major programming projects were years late**
- **Software costs much higher than predicted**
- **Unreliable programs**
- **Difficult to maintain, understand and change the software**
- **etc.**

The causes (1)

The major cause of the software crisis is that the **machines have become several orders of magnitude more powerful!** To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now **we have gigantic computers, programming has become an equally gigantic problem**.

Edsger Dijkstra

Complexity

Much larger and more complex software products than previously developed

Expectations

Hardware prices were going down, prices for software were increasing dramatically

Change

Need to maintain and evolve programs

The causes (2)

Larger and **more complex** programs became possible, BUT ...

... the **informal** way of developing software became insufficient!

Software engineering

... is an engineering discipline that is concerned with **all the aspects of software production** from the early stages of system specification to maintaining the system after it has gone into use

Firstly defined in 1968 at the NATO Software Engineering Conference

Why a different discipline is needed?

1. What is the difference between Software Engineering and Computer Science ?

Computer science

Concerned with theories and methods that underlies computer and software systems

Software engineering

Concerned with the **practical problem of producing software**

- It is not just concerned with the technical processes of software development but also with activities such as software project management and with the development of tools, methods and theories to support software production
- **Software engineers** apply theories, methods and tools but they use them selectively and always try to discover solutions to problems even when there are no applicable theories and methods
- **Software engineers** look for solutions within given constraints e.g., financial

2. What is a software product ?

Developer myth: The only deliverable work product for a successful project is the working program

WRONG!

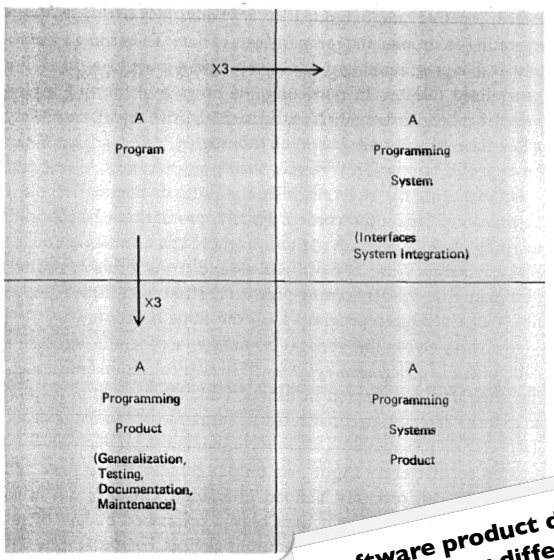
A. Instructions (computer programs) that when executed provide desired function + data structures that enable programs to manipulate information

+

B. Documents that describe the operation and use of the programs (e.g. documentation)

Real products are more than just coding ...

Programs vs Software



The Mythical Man-Month

Software product development involves many different activities

3. Which are the main activities to apply in order to build a software product ?

Fundamental activities

1. **Software specification**
   defining the software to be produced
2. **Software development**
   designing and programming
3. **Software validation**
   checking to ensure that it is what the customer required
4. **Software evolution**
   modification to adapt to customer / market requirements

A software process is a set of activities and associated results that produce a software product

Some processes

Various ways in which these activities can be "chained"

A. Waterfall approach

separate steps for specification, design, implementation, etc.

complete a phase and only after move to the next one

B. Iterative development

interleaves specification, development, validation activities, etc.

fast development of an initial system and refine it with the customer

Various pros/cons - see later :)

What fundamentals ?

Processes

what are the phases and how is the flow of a development process

Methods

how to accomplish different phases of a development process

Tools

what are the tools to support the involved activities

Other programming language mechanisms

used in professional software development

Computer-Aided Software Engineering Tools

A wide range of different types of programs that are used to support software process activities

Upper CASE

front-end activities e.g. requirements and design

...

Lower CASE

back-end activities e.g. programming, debugging, testing

...

2. Questions about Software Engineering

A. What is a software product ?

Computer programs and their documentation

describing their operation and usage (e.g., design, user manuals)

Types of software products

Generic

developed by an organisation and sold on an open market; the organisation controls the specification; e.g. Word, Excel

Customised

developed especially for a particular customer according to their specification; e.g. air traffic control

Currently, more and more generic products are built which can also be adapted for specific customers e.g., Enterprise Resource Planning (ERP)

Couldn't a software product be developed / built like other things human build?

Distinguishing characteristics of software

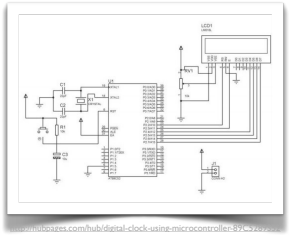
1. Software is developed or engineered, it is not manufactured in the classical sense

2. Software does not "wear out"

3. Although the industry is moving towards component-based assembly, most software continues to be custom built

Software is developed or engineered, it is not manufactured in the classical sense (I)

for hardware



manufacturing

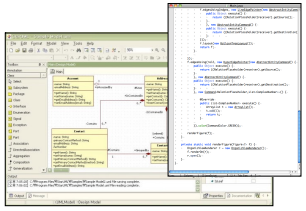
http://www.timetools.co.uk

a physical form

engineered

Software is developed or engineered, it is not manufactured in the classical sense (II)

for software



Logical "product"

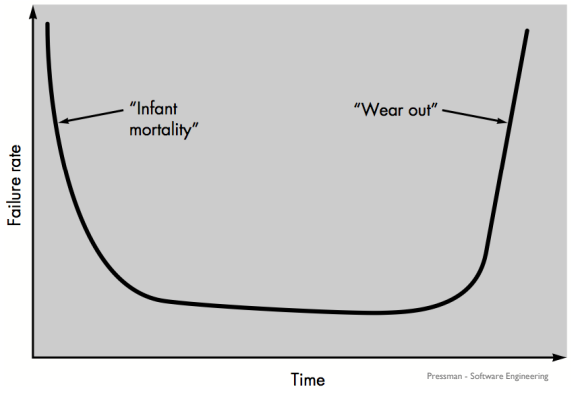
engineered

Both activities require the construction of a product but the approaches are different

Software costs are concentrated in engineering

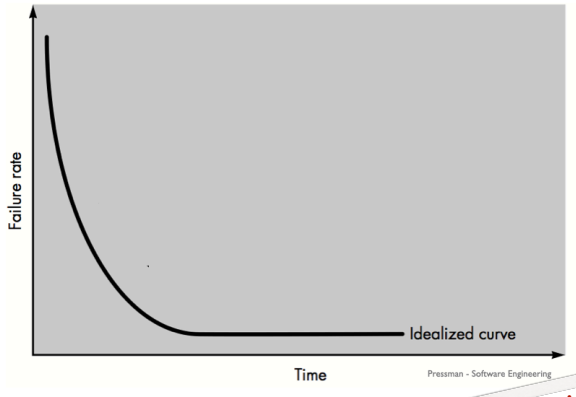
Software does not "wear out" (I)

for hardware



Software does not "wear out" (II)

for software



But it deteriorates in a different way

Software must evolve

Software systems must be continually adapted to new requirements else they become progressively less satisfactory

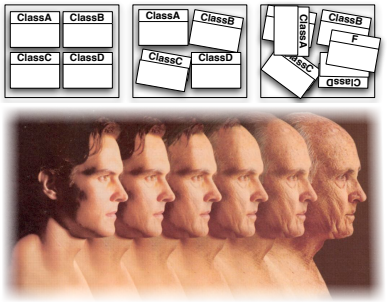
One of the Lehman's software evolution laws

Why ?

Software aging

Programs, like people, get old!

D. Parnas - Software aging



How ?

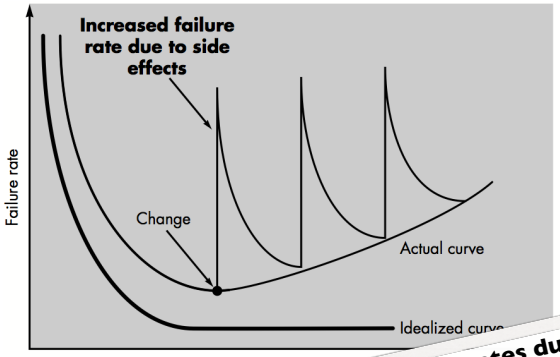
Changes are not necessary performed by the original developers

Those that modify a system do not fully understand it and thus, may deteriorate the program (including its internal structure)

After several such modifications *nobody* understand the modified product :)

Software does not "wear out" (III)

for software

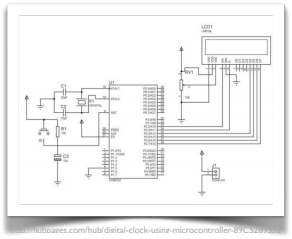


Software deteriorates due to repeated changes

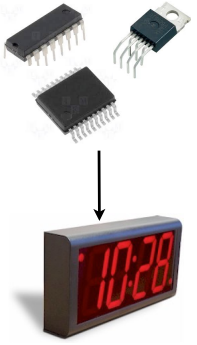
Software maintenance considerably more difficult than hardware maintenance

Software is still custom built

for hardware



select standard components, having well defined functions, interfaces, etc.



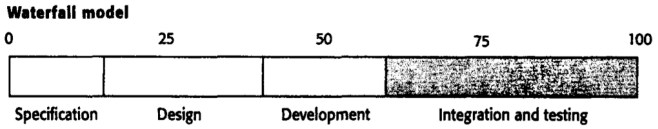
http://www.timetools.co.uk

In the software world, this component reuse is something that has only begun to be achieved

We need more than algorithmic libraries e.g., frameworks

B. What are the software costs? (I)

How is the software cost distributed over different activities e.g., specification, design, development, testing?



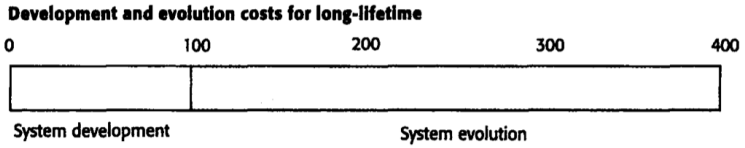
Coding is NOT the main cost :)

And these are only initial production costs!

What are the software costs? (II)

Developer myth: Once we write the program and get it work our work is done

WRONG!



Always, design your product for change!

C. What is quality for software?

Functional suitability e.g., *correctness* - it behaves/ functions as per software requirement specifications

Maintainability - software should be written in such a way that it may evolve to meet the changing needs of customers e.g., *modularity, testability*

Usability - it must be usable, without efort, by the type of user for whom it is designed e.g., *user interface aesthetics, learnability*

Efficiency - it should not make wasteful use of system resources such as memory and processor e.g., *time behaviour*

Portability, Reliability, Security, ...

The expected non-functional quality factors depend on the application and not possible to optimise it for all these factors

3. Myths about software development

Developer myth: Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down

WRONG!

Software engineering is NOT about documents,

it is about creating quality

Better quality leads to reduce rework, reduce rework results in faster delivery times (and costs).

Developer myth: Until I get the program "running" I have no way of assessing its quality

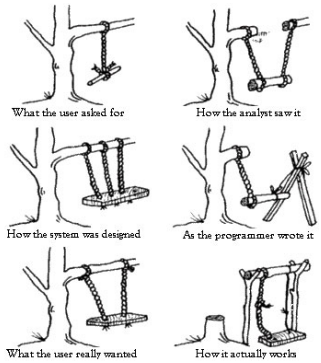
WRONG!

Software reviews can be applied from the inception of the project and found to be effective in finding certain classes of problems

Some design analyses can be run while writing the code

Customer myth: A general statement of objectives is sufficient to begin writing programs - we can fill details later

WRONG!



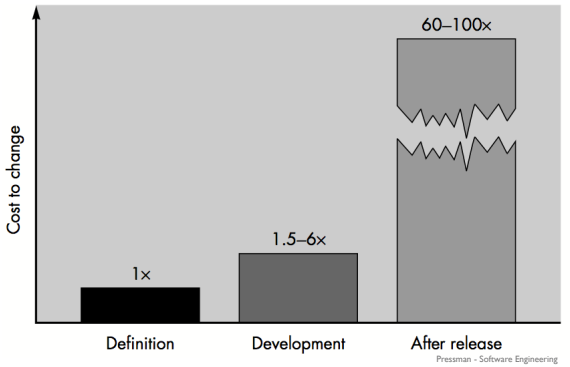
A poor up-front definition is a major cause of failed projects

Communication with the customer is mandatory

Detailed (formal) description of domain, function, behaviour, performance, interfaces, validation criteria are essential

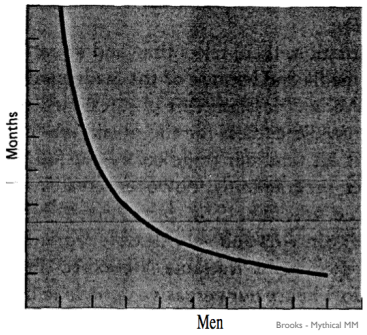
Customer myth: Project requirements continually change, but change can be easily accommodated because software is flexible

WRONG!

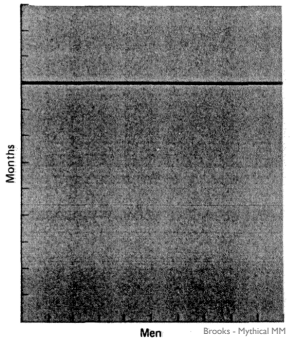


Management myth: If we get behind schedule, we can add more programmers and catch up

WRONG!



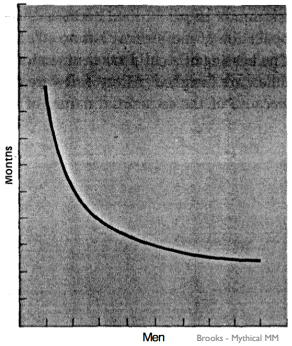
Completely independent subtasks



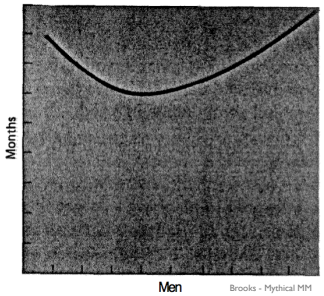
Non decomposable task

Management myth: If we get behind schedule, we can add more programmers and catch up

WRONG!



Partially independent tasks
(extra time for communication)



... and when too much communication is required :)

Brook's law - Adding manpower to a late software project makes it later

Dr. Petru Florin Mihancea