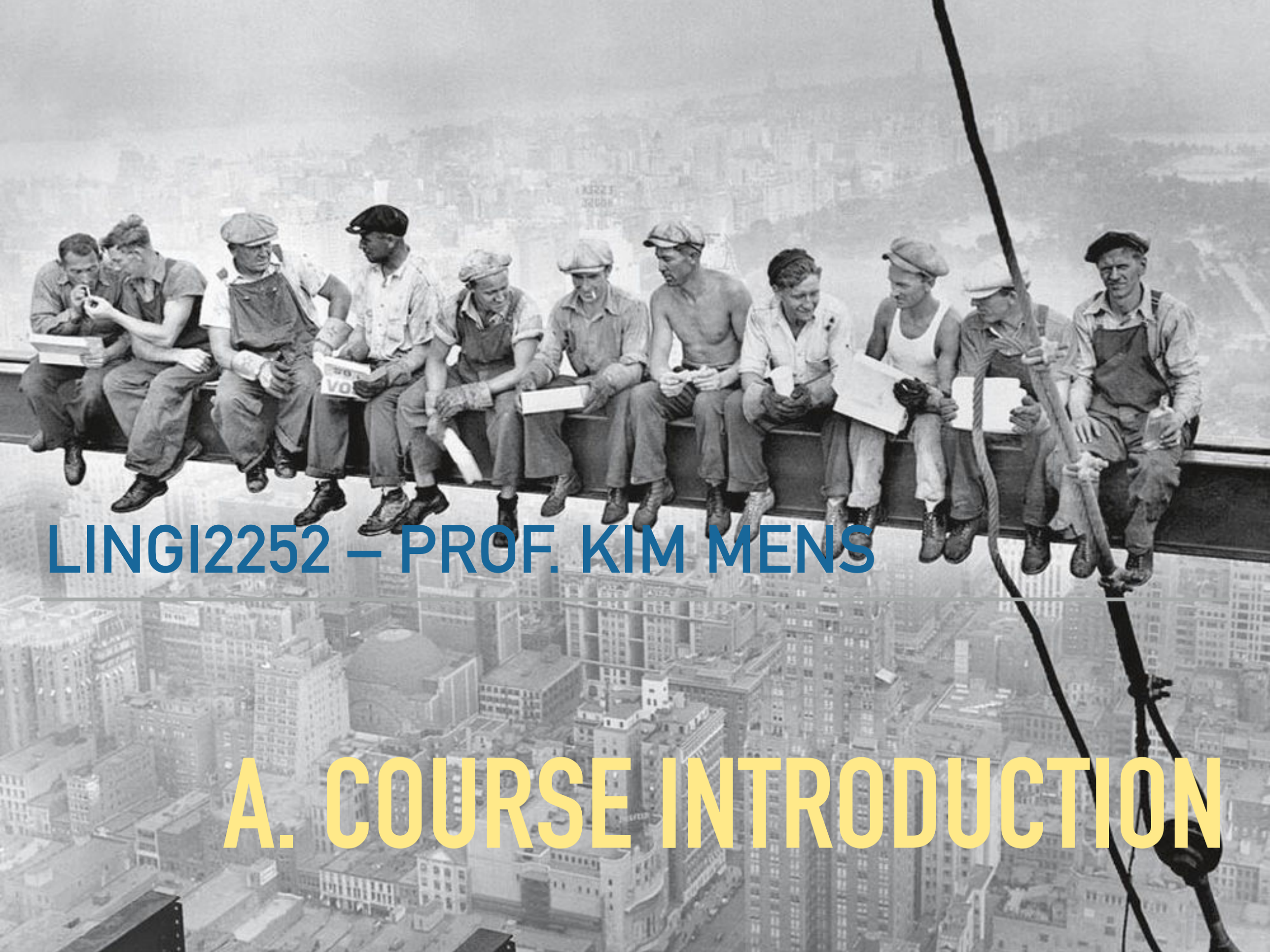
LINGI2252 – PROF. KIM MENS

SOFTWARE MAINTENANCE & EVOLUTION

LINGI2252 – PROF. KIM MENS

INTRODUCTION & PRELIMINARIES



LINGI2252 – PROF. KIM MENS

A. COURSE INTRODUCTION

COURSE THEMES

Software Maintenance

Software Evolution

Software Reuse

LEARNING OUTCOMES

Gain familiarity with the concepts of **software evolution**, **reuse** and **maintenance**.

Gain *hands-on experience* with techniques to build more maintainable and reusable software.

Identify the *issues and challenges* associated with software evolution and assess their impact.

Discuss *(dis)advantages* and *trade-offs* of different types and techniques for software reuse.

COURSE CONTENTS

Concepts and definitions

Domain modelling & feature-oriented domain analysis

Software reuse & object-oriented programming

Bad smells and refactoring

Software patterns

Design heuristics

Libraries & application frameworks

An industrial case study

Reflection, aspect-oriented programming and context-oriented programming

COURSE ORGANISATION

Theory sessions covering the different course topics

Practical sessions to apply the concepts in practice

developing and evolving a maintainable and reusable software system

Missions to complete the application developed during the practical sessions

COURSE EVALUATION

[10%] Obligatory participation during practical sessions

reviewing work of other groups

[40%] intermediate missions in-between practical sessions

presentation and demo of deliverables produced

2 missions (10% + 20%) throughout semester

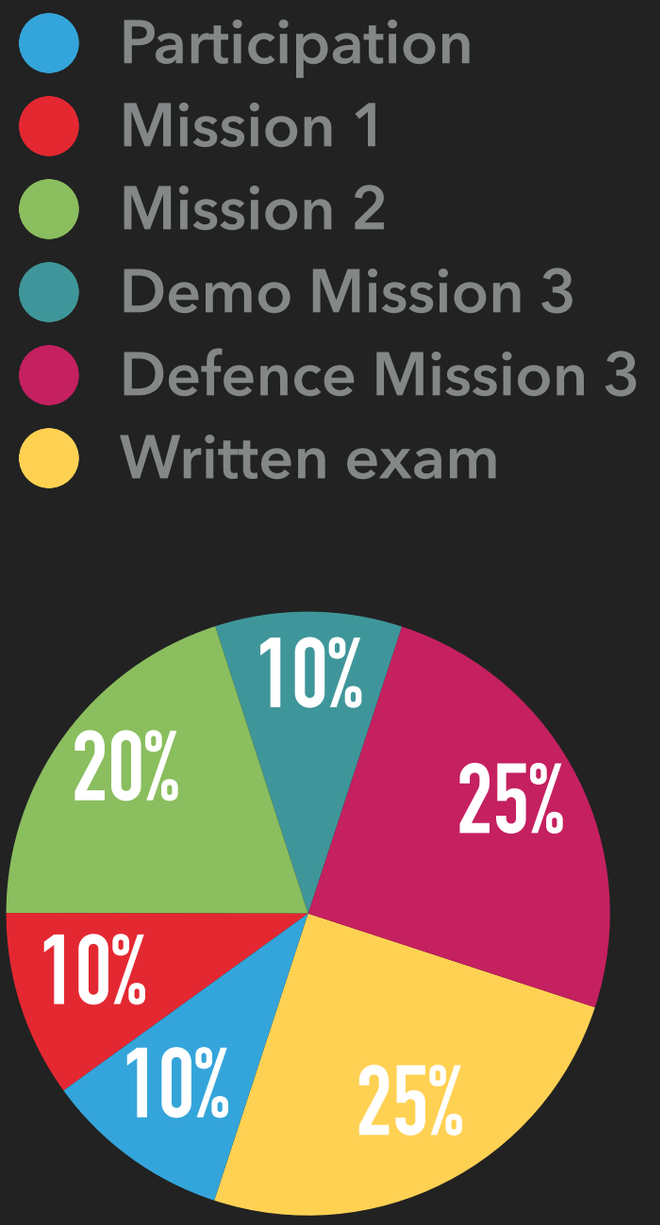
demo of 3rd mission (10%) at end of semester

[50%] during exam session

[25%] written exam

[25%] presentation and discussion

on 3rd mission + overall lessons learned throughout course



LINGI2252 – PROF. KIM MENS

B. SOFTWARE MAINTENANCE

WHY SOFTWARE FAILS

Billions of $ per year wasted on preventable mistakes

Biggest tragedy : software failure is mostly predictable and avoidable.

Organisations don't see preventing failure as important

even though this can harm or destroy the organisation

YEAR	COMPANY	OUTCOME (COSTS IN US $)
2005	Hudson Bay Co. [Canada]	Problems with inventory system contribute to $33.3 million* loss.
2004–05	UK Inland Revenue	Software errors contribute to $3.45 billion* tax-credit overpayment.
2004	Avis Europe PLC [UK]	Enterprise resource planning (ERP) system canceled after $54.5 million† is spent.
2004	Ford Motor Co.	Purchasing system abandoned after deployment costing approximately $400 million.
2004	J Sainsbury PLC [UK]	Supply-chain management system abandoned after deployment costing $527 million.†
2004	Hewlett-Packard Co.	Problems with ERP system contribute to $160 million loss.
2003–04	AT&T Wireless	Customer relations management (CRM) upgrade problems lead to revenue loss of $100 million.
2002	McDonald's Corp.	The Innovate information-purchasing system canceled after $170 million is spent.
2002	Sydney Water Corp. [Australia]	Billing system canceled after $33.2 million† is spent.
2002	CIGNA Corp.	Problems with CRM system contribute to $445 million loss.
2001	Nike Inc.	Problems with supply-chain management system contribute to $100 million loss.
2001	Kmart Corp.	Supply-chain management system canceled after $130 million is spent.
2000	Washington, D.C.	City payroll system abandoned after deployment costing $25 million.
1999	United Way	Administrative processing system canceled after $12 million is spent.
1999	State of Mississippi	Tax system canceled after $11.2 million is spent; state receives $185 million damages.
1999	Hershey Foods Corp.	Problems with ERP system contribute to $151 million loss.
1998	Snap-on Inc.	Problems with order-entry system contribute to revenue loss of $50 million.
1997	U.S. Internal Revenue Service	Tax modernization effort canceled after $4 billion is spent.
1997	State of Washington	Department of Motor Vehicle (DMV) system canceled after $40 million is spent.
1997	Oxford Health Plans Inc.	Billing and claims system problems contribute to quarterly loss; stock plummets, leading to $3.4 billion loss in corporate value.
1996	Arianespace [France]	Software specification and design errors cause $350 million Ariane 5 rocket to explode.
1996	FoxMeyer Drug Co.	$40 million ERP system abandoned after deployment, forcing company into bankruptcy.
1995	Toronto Stock Exchange [Canada]	Electronic trading system canceled after $25.5 million** is spent.
1994	U.S. Federal Aviation Administration	Advanced Automation System canceled after $2.6 billion is spent.
1994	State of California	DMV system canceled after $44 million is spent.
1994	Chemical Bank	Software error causes a total of $15 million to be deducted from 100 000 customer accounts.
1993	London Stock Exchange [UK]	Taurus stock settlement system canceled after $600 million** is spent.
1993	Allstate Insurance Co.	Office automation system abandoned after deployment, costing $130 million.
1993	London Ambulance Service [UK]	Dispatch system canceled in 1990 at $11.25 million**; second attempt abandoned after deployment, costing $15 million.**
1993	Greyhound Lines Inc.	Bus reservation system crashes repeatedly upon introduction, contributing to revenue loss of $61 million.
1992	Budget Rent-A-Car, Hilton Hotels, Marriott International, and AMR [American Airlines]	Travel reservation system canceled after $165 million is spent.

http://spectrum.ieee.org/computing/software/why-software-fails

YEAR	COMPANY	OUTCOME (COSTS IN US $)
2005	Hudson Bay Co. [Canada]	Problems with inventory system contribute to $33.3 million* loss.
2004–05	UK Inland Revenue	Software errors contribute to $3.45 billion* tax-credit overpayment.
2004	Avis Europe PLC [UK]	Enterprise resource planning (ERP) system canceled after $54.5 million† is spent.
2004	Ford Motor Co.	Purchasing system abandoned after deployment costing approximately $400 million.
2004	J Sainsbury PLC [UK]	Supply-chain management system abandoned after deployment costing $527 million.†
2004	Hewlett-Packard Co.	Problems with ERP system contribute to $160 million loss.
2003–04	AT&T Wireless	Customer relations management (CRM) upgrade problems lead to revenue loss of $100 million.
2002	McDonald's Corp.	The Innovate information-purchasing system canceled after $170 million is spent.
2002	Sydney Water Corp. [Australia]	Billing system canceled after $33.2 million† is spent.
2002	CIGNA Corp.	Problems with CRM system contribute to $445 million loss.
2001	Nike Inc.	Problems with supply-chain management system contribute to $100 million loss.
2001	Kmart Corp.	Supply-chain management system canceled after $130 million is spent.
2000	Washington, D.C.	City payroll system abandoned after deployment costing $25 million.
1999	United Way	Administrative processing system canceled after $12 million is spent.
1999	State of Mississippi	Tax system canceled after $11.2 million is spent; state receives $185 million damages.
1999	Hershey Foods Corp.	Problems with ERP system contribute to $151 million loss.
1998	Snap-on Inc.	Problems with order-entry system contribute to revenue loss of $50 million.
1997	U.S. Internal Revenue Service	Tax modernization effort canceled after $4 billion is spent.
1997	State of Washington	Department of Motor Vehicle (DMV) system canceled after $40 million is spent.
1997	Oxford Health Plans Inc.	Billing and claims system problems contribute to quarterly loss; stock plummets, leading to $3.4 billion loss in corporate value.

YEAR	COMPANY	OUTCOME (COSTS IN US $)
2005	Hudson Bay Co. [Canada]	Problems with inventory system contribute to $33.3 million* loss.
2004–05	UK Inland Revenue	Software errors contribute to $3.45 billion* tax-credit overpayment.
2004	Avis Europe PLC [UK]	Enterprise resource planning (ERP) system canceled after $54.5 million† is spent.
2004	Ford Motor Co.	Purchasing system abandoned after deployment costing approximately $400 million.
2004	J Sainsbury PLC [UK]	Supply-chain management system abandoned after deployment costing $527 million.†
2004	Hewlett-Packard Co.	Problems with ERP system contribute to $160 million loss.
2003–04	AT&T Wireless	Customer relations management (CRM) upgrade problems lead to revenue loss of $100 million.
2002	McDonald's Corp.	The Innovate information-purchasing system canceled after $170 million is spent.
2002		
2002		
2001		…n loss.
2001		
2000		
1999		
1999		…n damages.
1999		
1998		
1997		
1997	State of Washington	Department of Motor Vehicle (DMV) system canceled after $40 million is spent.
1997	Oxford Health Plans Inc.	Billing and claims system problems contribute to quarterly loss; stock plummets, leading to $3.4 billion loss in corporate value.

The national economic impacts of software defects are significant. In the USA the cost of software defects has been estimated to be $59 billion, that is 0.6% of the gross domestic product.

Source: National Institute of Standards & Technology (NIST): The Economic Impacts of Inadequate Infrastructure for Software Testing
www.nist.gov/director/prog-ofc/report02-3.pdf

THE IMPORTANCE OF MAINTENANCE

Rates of software engineering failure

Requirements	Very High
Specification	Low
Design	Low
Implementation	Low
Installation	High
Operation	Enormous
Maintenance	Very High

40 - 75%

TABLE I. Estimations of Percentage of Total Costs Represented by Maintenance Costs

Maintenance Cost (percentage)	Study Estimation
40	[17 and 18]
40–60	[6, 7, 8, 10, 11, 16, 19, 26]
67	[28]
70	[6]
75	[1, 12]

Source: Guimaraes, T. 1983. Managing application program maintenance expenditures. *Commun. ACM* 26, 10 (Oct. 1983), 739-746

Standard Software: 25 bugs per 1000 lines of program.
Good Software: 2 errors per 1000 lines.
Space Shuttle Software: < 1 errors per 10000 lines.

Example Handy (Cellular Phone):
200 000 lines of program: up to 600 errors.

Windows-95: 10 Mill. lines: up to 200 000 errors. See also

SOFTWARE MAINTENANCE IS HARD

Even *after deployment*, software systems may need to undergo changes, for example to fix problems or improve the system.

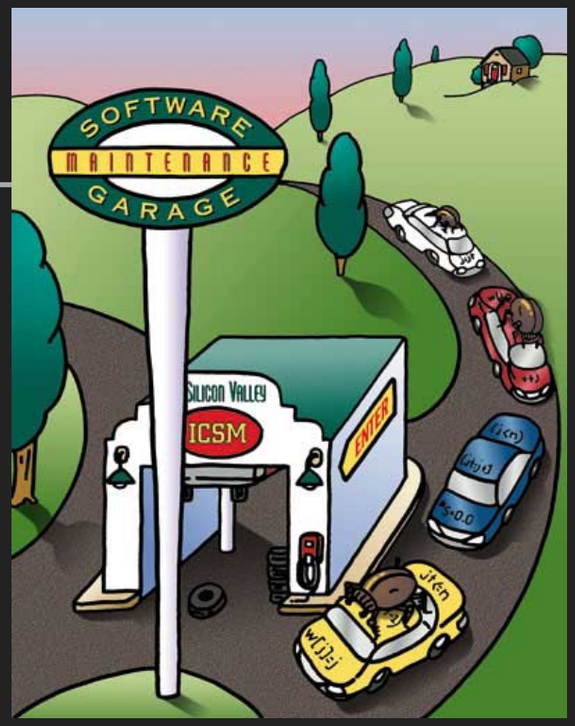
This activity is called "software maintenance".

Software maintenance is a crucial, but critical, activity in the life cycle of a system.

It's often harder to maintain a system than to develop it.

But it's even *harder to design a maintainable system*, because it's hard to foresee all future changes.

DEFINITION OF SOFTWARE MAINTENANCE



According to [IEEE Std 610.12-1990]

"**Software maintenance** is the process of modifying a software system or component after delivery to …

(Software) **maintainability** is the ease with which a software system or component can be modified to …

… *correct faults, improve performance or other attributes, or adapt to a changed environment.*"

TYPES OF SOFTWARE MAINTENANCE

Different types of maintenance can be distinguished, depending on the goal of the maintenance activity:

- Adaptive maintenance
- Corrective maintenance
- Perfective maintenance
- Preventive maintenance

Definitions provided by the [IEEE Std 610.12-1990] standard

ADAPTIVE MAINTENANCE

Adapting to changes in the environment (either hardware and software)

According to [IEEE Std 610.12-1990], adaptive maintenance is

> "Software maintenance performed to make a computer program usable in a changed environment."

Occurs when, as a result of external influences or strategic changes, a software system needs to be adapted to remain up to date.

Examples:

> The government decides to change the VAT rate from 21% to 19%.
>
> An insurance company decides to offer a new kind of insurance.
>
> A company decides to introduce an online system to allow clients to access its services. This online system needs to be integrated into their normal ordering system.

CORRECTIVE MAINTENANCE

Correcting errors that cause the software to behave in undesired or unexpected ways.

According to [IEEE Std 610.12-1990], corrective maintenance is

> "Software maintenance performed to correct faults in hardware or software."

Often occurs after deployment when customers detect problems that were not discovered during initial testing of the system. These errors should be fixed.

PERFECTIVE MAINTENANCE

Improving the quality of a software system.

According to [IEEE Std 610.12-1990], perfective maintenance is

> "Software maintenance performed to improve the performance, maintainability or other attributes of a computer program."

Occurs after the system has been in place and running fine for a while, and end users start asking for minor tweaks improvements that could improve the way the system works.

Examples :

> Better input forms, shortcut commands, better help system or error reporting, making the system more responsive, …

PREVENTIVE MAINTENANCE

Proactively change the software to avoid future problems

According to [IEEE Std 610.12-1990], preventive maintenance is

> "Software maintenance performed for the purpose of preventing problems before they occur."

TYPES OF SOFTWARE MAINTENANCE

WHEN? \ WHY?	Correction	Enhancement
Proactive	Preventive	Perfective
Reactive	Corrective	Adaptive

A STUDY OF SOFTWARE MAINTENANCE IN INDUSTRY

Between 1977 and 1980, Lientz & Swanson carried out a large study of the maintenance of application software in 487 companies

They found that, on average, development and systems staff spent *half of their time* on maintenance.

The larger the company, the more time was spent on maintenance.

Source: Lientz & Swanson, 1980

MAINTENANCE EFFORT

Corrective Maintenance : 21,7%

- Emergency fixes: 12,4%
- Routine debugging: 9,3%

Adaptive Maintenance : 23,6%

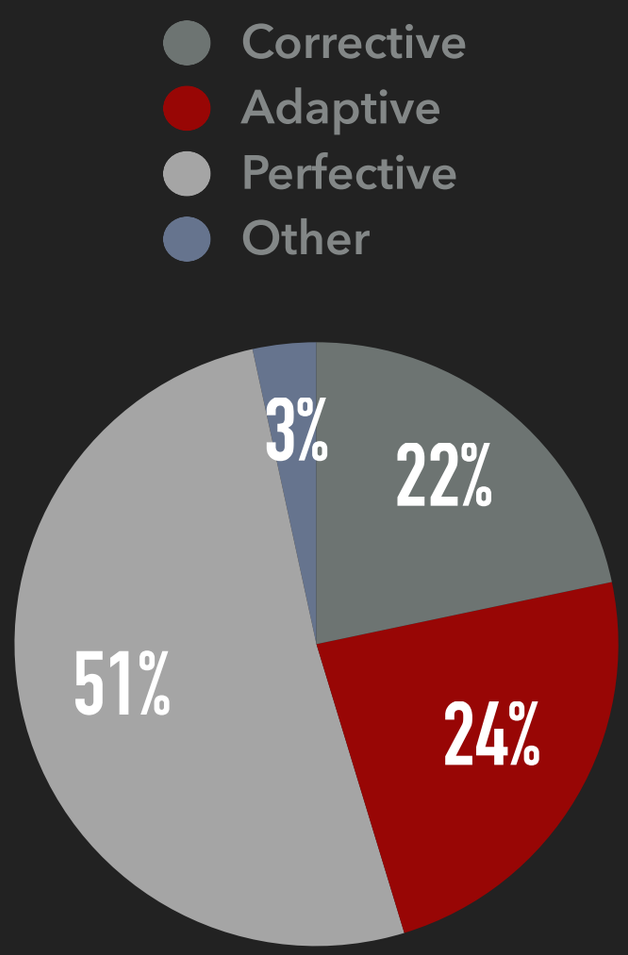
- Accommodating changes to data inputs and files: 17,4%
- Accommodating changes to hardware and software: 6,2%

Perfective Maintenance : 51,3%

- Customer enhancements: 41.8%
- Improvements to documentation: 5.5%

Other (Preventive Maintenance) : 3,4%

According to the study, maintenance effort, was broken out as follows:



MAINTENANCE REASONS

Changed user requirements

- user-requested extensions and modifications

Bug fixes

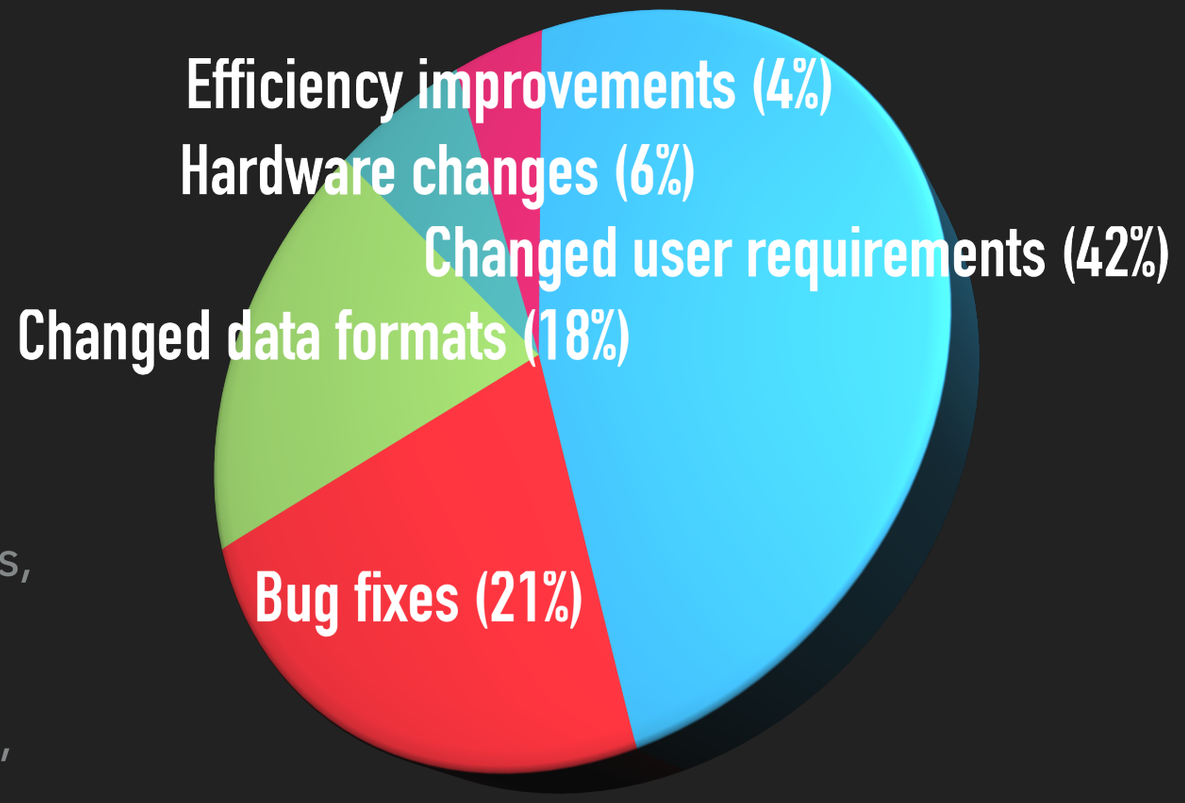
- scheduled routine fixes
- emergency fixes (more costly due to heavy pressure)

Changed data formats

- Y2K, Euro, tax rates, postal codes, phone numbers, ...
- new standards: UML, XML, COM, DCOM, CORBA, ActiveX, WAP

Hardware changes

Efficiency improvements



MAIN CAUSES OF MAINTENANCE PROBLEMS

Poor quality of the software documentation

Poor software quality (e.g., unstructured code, too large components, inadequate design)

Insufficient knowledge about the system and its domain

- (maybe unavailable due to personnel turnover)

Ineffectiveness of maintenance team

- low productivity, low motivation, low skill levels, competing demands for programmer time

KEY TO MAINTENANCE IS IN DEVELOPMENT

Higher code quality

⇒ less (corrective) maintenance

Anticipating changes

⇒ less (adaptive and perfective) maintenance

Better tuning to user needs

⇒ less (perfective) maintenance

Less code

⇒ less maintenance

LINGI2252 – PROF. KIM MENS

C. SOFTWARE EVOLUTION

SOFTWARE AGEING

“Programs, like people, get old. We can’t prevent ageing, but we can understand its causes, take steps to limit its effects, temporarily reverse some of the damage it has caused, and prepare for the day when the software is no longer viable.”

“A sign that the software engineering profession has matured will be that we lose our preoccupation with the first release and focus on the long-term health of our products. Researchers and practitioners must change their perception of the problems of software development. Only then will software engineering deserve to be called ‘engineering’.”

Source: Parnas, 1994 [http://ieeexplore.ieee.org/document/296790/]

REASONS WHY SOFTWARE AGES

maintenance activities

ignorant surgery and architectural erosion

inflexibility from the start

insufficient or inconsistent documentation

deadline pressure

duplicated functionality (code duplication)

lack of modularity

PLUS ÇA CHANGE,
PLUS C'EST LA MÊME CHOSE.

Jean-Baptiste Alphonse Karr

CHANGE IS INEVITABLE

New requirements emerge when software is being used

Even as it is being developed !

Business environments change

Feedback loop : the changed software may even be the reason why the environment changes !

Errors must be repaired

New computers and equipment are added to the system

The performance or reliability of the system may have to be improved

New technology (new standards, new OS, new software versions, ...)

CHANGE INDUCES TECHNICAL DEBT

As a change is started on a software system, often there is a need to make other coordinated changes at the same time in other parts of the software.

These other required changes, if *not* completed immediately, incur a kind of *debt* that must be paid at some point in the future.

This technical debt is "the extra development work that arises when code that is easy to implement in the short run is used instead of applying the best overall solution".

CHANGE INDUCES TECHNICAL DEBT

In other words, technical debt is an accumulation of poor and lazy implementation choices that slowly makes the code hard to maintain or evolve.

Just like financial debt, if technical debt is not repaid, the uncompleted changes accumulate interest on top of interest, because of increased entropy.

The longer you wait to make the changes, the harder it becomes.

CHANGE IS HARD TO ANTICIPATE

Many *changes cannot be anticipated* at design time

> "The fundamental problem, supported by 40 years of hard experience, is that many changes actually required are those that the original designers cannot even conceive of." [Bennett & Rajlich 2000]

The *Heisenberg principle* of software development

- requirements start changing from the moment you start building or using a software system

Key challenge for organisations

- implementing and managing change to their existing software systems

MAINTENANCE VS. EVOLUTION

Software maintenance typically does not involve major changes to the system's architecture

Changes are implemented by modifying existing components and adding new components to the system

Software evolution is a broader term that encompasses both software maintenance and bigger changes

At different phases of the software life-cycle

Evolution is intrinsic to the very nature of software development

SOFTWARE EVOLUTION

Evolution may arise

during software development

where the design evolves and matures as the understanding of the problem to be solved and how to solve it gradually increases

during software maintenance

after deployment, in the continuing process of adapting the system to the changing needs of its users and usage environment

when the system's continuous evolution made it too complex to maintain

the system may requiring a more extensive *restructuring*, *redesign* or even a full *reimplementation* or *replacement*

DEFINITION OF SOFTWARE EVOLUTION

Software evolution is …

> … all programming activity that is intended to generate a new software version from an earlier operational version
> [Lehman & Ramil 2000]

EVOLUTION MECHANISMS

Different types of evolution can be distinguished according to the mechanism how evolution is achieved

Manual : changes applied manually by a software developer

Generic : write sufficiently generic and abstract components that are broadly adaptable

Generation : generate lower-level representation from a higher-level specification of the software

a.k.a. *vertical transformation* or *refinement*: from more abstract to more concrete

Transformation : old components are transformed into a newer version

a.k.a. *horizontal transformation* or *restructuring*: transformation at the same level of abstraction

Configuration : different variants of a software component are available up front but the actual selection of which one to use is based on a desired configuration

e.g. software product lines, software-as-a-service, context-oriented programming, …

STATIC VS. DYNAMIC SOFTWARE EVOLUTION

Static evolution

- changes are applied manually by a human programmer
- part of the software gets adapted or replaced by a programmer and the evolved software is redeployed

Dynamic evolution

- changes are applied automatically at runtime
- to better suit the current needs of the software system
- by automatically generating, adapting, transforming or selecting parts of the software
- e.g., self-adaptive systems, metaprogramming, context-oriented programming, …

PROGRAMMING PARADIGMS . . .

Many programming paradigms offer dedicated mechanisms to support *software evolution* and *reuse* at the language level

Object-oriented programming offers abstractions like *inheritance* to enable *code reuse* and *extensibility*

Event-driven programming allows a program to trigger certain actions in response to dynamically occurring *events* such as user inputs or actions

Service-oriented programming was originally proposed as a new paradigm to promote *software reuse* by having *services* with a well-defined interface as the main unit of modularisation

Component-based development is a *reuse-based* approach to defining, implementing and composing loosely coupled independent *components* into software systems

. . . PROGRAMMING PARADIGMS

Aspect-oriented programming allows to implement "cross-cutting" concerns, not central to the business logic, separately from the base functionality, and then "weave" these back into the code

Metaprogramming is the writing of computer programs that treat other programs as their data, thus allowing them to read, generate, analyse or transform other programs, or even modify themselves while running. More specifically, reflection is the ability of a computer program to examine, introspect, and modify its own structure and behaviour at runtime.

Context-oriented programming models context as a first class language citizen and enables programs to adapt their behaviour dynamically to changing contexts.

... (and many more)

THE LAWS OF SOFTWARE EVOLUTION

After major empirical studies, from 1974 onwards, Lehman and Belady proposed a number of “laws” that apply to many evolving software systems.

The laws describe a balance between forces that drive new development and forces that slow down progress.

influenced by thermodynamics

reflect established observations and empirical evidence

Over the past decades the laws have been revised and extended several times:

M. M. Lehman, L. Belady. Program Evolution: Processes of Software Change. Academic Press, London, 1985, 538pp.

THE LAWS OF SOFTWARE EVOLUTION

There is no such thing as a “finished” computer program.

Lehman and Belady were the first to recognise the phenomenon of software evolution.

Their laws of software evolution are based on a study of the evolution of IBM 360 mainframe OS and led, over a period of 20 years, to the formulation of eight Laws of Software Evolution.

> M. M. Lehman. Laws of Software Evolution Revisited. Lecture Notes in Computer Science 1149, pp. 108-124, Springer Verlag, 1997

THE EIGHT LAWS OF SOFTWARE EVOLUTION

Law 1: Continuing change

Law 2: Increasing complexity

Law 3: Self regulation

Law 4: Conservation of organisational stability

Law 5: Conservation of familiarity

Law 6: Continuing growth

Law 7: Declining quality

Law 8: Feedback system

LAW 1 : CONTINUING CHANGE

Law 1: Continuing change

> A program that is used in a real-world environment must be continually adapted, or else become progressively less satisfactory.

Reasons:

- Evolution of the environment (“operational domain”)
- Hence, increasing mismatch between the system and its environment
- Continuous need for change because requirements and environment continuously evolving

LAW 2 : INCREASING COMPLEXITY

Law 2: Increasing complexity

> As a program is evolved its complexity increases with time unless specific work is done to maintain or reduce it.

Reasons:

Inspired by the second law of entropy in thermodynamics.

Unaddressed technical debt increases entropy.

Small changes are applied in a step-wise process; each ‘patch’ makes sense locally, not globally

Effort needed to address accumulated technical debt; a more significant restructuring or refactoring may be needed

LAW 6 : CONTINUING GROWTH

Law 6: Continuing Growth

> Functional content of a program must be continually increased to maintain user satisfaction over its lifetime.

Related to the first law, but with focus on functional requirements

> often one cannot afford to omit existing functionality
>
> “omitted attributes will become the bottlenecks and irritants in usage as the user has to replace automated operation with human intervention. Hence they also lead to demand for change”

LAW 7 : DECLINING QUALITY

Law 7: Declining Quality

> "Evolving programs will be perceived as of declining quality unless rigorously maintained and adapted to a changing operational environment."

Related the first law, but with focus on observed reliability

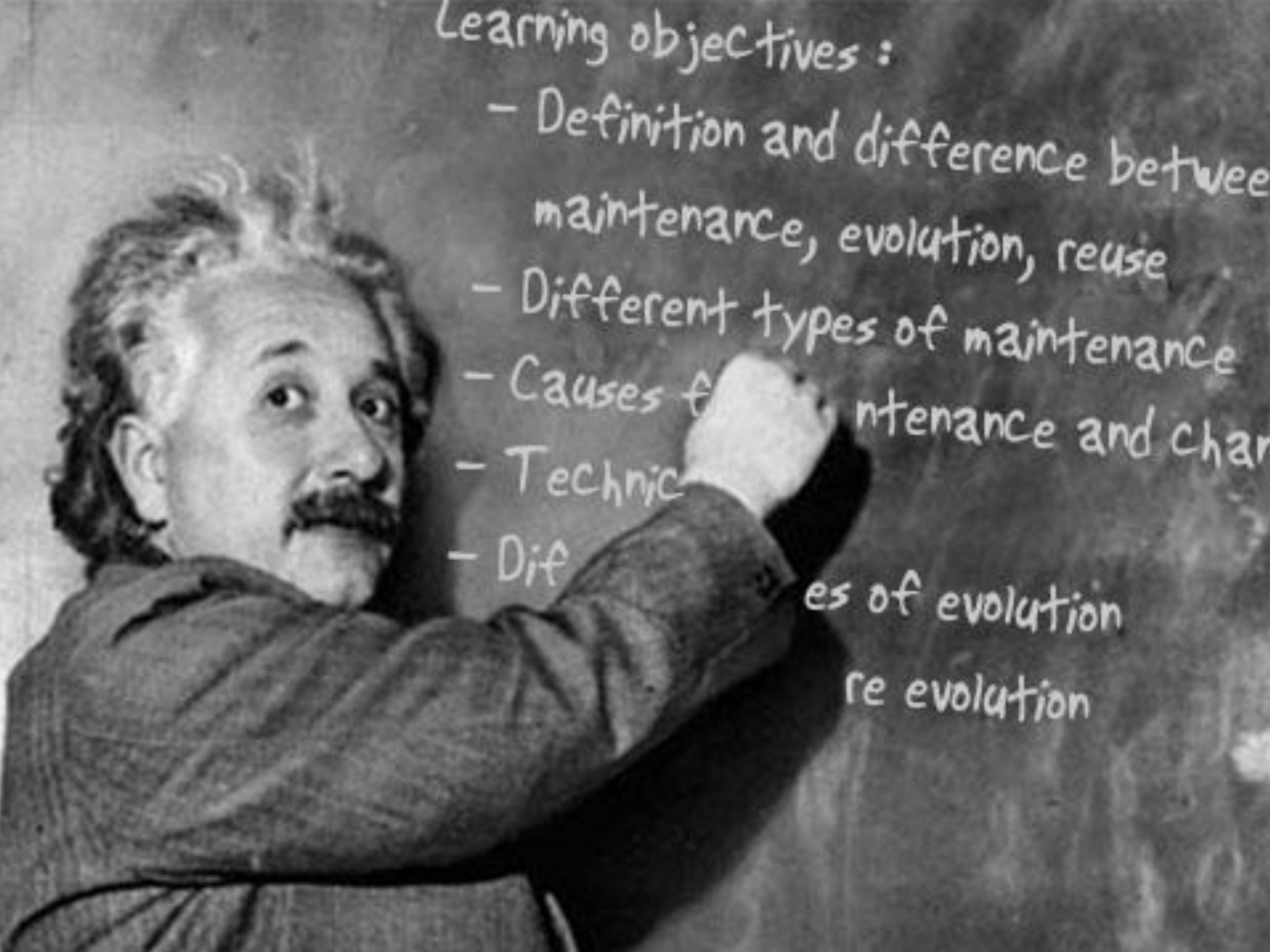
APPLICABILITY OF THE LAWS

These laws of software evolution seem to be generally applicable to large, tailored systems developed by large organisations.

- Confirmed in later work by Lehman on the FEAST project

No proof yet whether they are applicable to other types of software as well

- Systems that incorporate a significant number of “off the shelf“ components
- Small organisations
- Medium sized systems
- Open source software


Learning objectives :
- Definition and difference betwee
maintenance, evolution, reuse
- Different types of maintenance
- Causes f ntenance and char
- Technic es of evolution
- Dif re evolution

POSSIBLE QUESTIONS

1. Define and explain, in your own words, the difference between software maintenance and software evolution.
2. List and explain the different types of software maintenance. Given an illustrative example of at least three different types of particular maintenance activities.
    1. adaptive, corrective, perfective, preventive
    2. proactive or reactive, correction or enhancement
3. Discuss the need for and possible reasons for software maintenance, change and evolution.
4. Give some main causes of maintenance problems.
5. Define and explain, in your own words, what technical debt is.
6. What different types of software evolution can be distinguished?
7. Pick one of the laws (1, 2, 6 or 7) of software evolution and explain it.