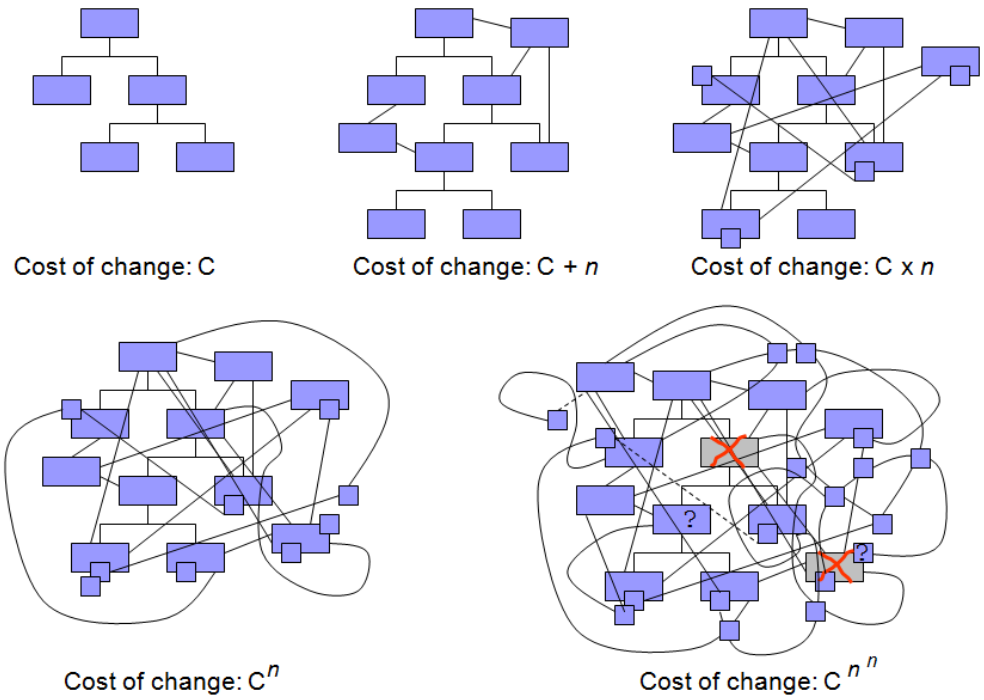
SOEN 6431
Software Comprehension and Maintenance

Introduction to Software Evolution and Software Aging

Premise

"All known compound objects decay and become more complex with the passage of time. Software is no exception."



Terminology

Maintenance/Evolution

- Software Maintenance – When changing an existing system after the initial release.
- The maintenance phase – the time from first release to retirement of a system, often many years.
- Almost all industrial SW development is "maintenance", systems are seldom created from scratch
- The term "Software Evolution" is often used instead

However there is a semantic difference.

Lowell Jay Arthur distinguish the two terms as follows:

- "Software maintenance means to preserve from failure or decline."
- "Software evolution means a continuous change from lesser, simpler, or worse state to a higher or better state."

Keith H. Bennett and Lie Xu use the term:

- "maintenance for all post-delivery support and evolution to those driven by changes in requirements."

Terminology: Reengineering

Reengineering is done to transform an existing "lesser or simpler" system into a new "better" system.

Reengineering is the examination, analysis and restructuring of an existing software system to reconstitute it in a new form, and the subsequent implementation of the new form.

Chikofsky and Cross II defines reengineering as:

- "the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form."

Terminology: Reengineering

Jacobson and Lindstorm defined following formula:

Reengineering = Reverse engineering + Δ + Forward engineering.

Reverse engineering is the activity of defining a more abstract, and easier to understand, representation of the system

The core of reverse engineering is the process of examination, not a process of change, therefore it does not involve changing the software under examination.

The third element "forward engineering," is the traditional process of moving from high-level abstraction and logical, implementation-independent designs to the physical implementation of the system.

The second element Δ captures alteration that is change of the system.

Re-engineering/ Reverse Engineering

Re-engineering = Reverse Engineering (+ actions/intentions) + Forward Engineering

Reverse engineering = extracting more abstract representations (e.g. extract UML from code)

Forward engineering = the opposite, develop more detailed design (or code) from higher level documents

Actions/intentions = e.g. re-structuring, documentation, verification, performed on a higher-level representation of the system (a model), resulting from the reverse engineering.

Legacy System

- *A sum of money, or a specified article, given to another by will; anything handed down by an ancestor or predecessor. — Oxford English Dictionary*
- A **legacy system** is a piece of software that:
  - you have *inherited*, and is *valuable* to you.
- Typical **problems** with legacy systems:
  - original developers *not available*
  - *outdated* development methods used
  - extensive patches and *modifications* have been made
  - *missing* or outdated documentation

=> So further evolution and development may be prohibitively expensive !

Terminology: Legacy System

A legacy system is an old system which is valuable for the company which often developed and owns it.

It is the phase out stage of the software evolution model of Rajlich and Bennet described earlier.

Bennett used the following definition:

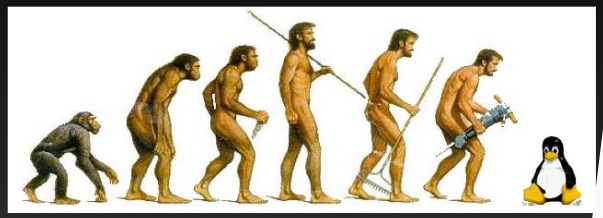
- "large software systems that we don't know how to cope with but that are vital to our organization."

Similarly, Brodie and Stonebraker: define a legacy system as

- "any information system that significantly resists modification and evolution to meet new and constantly changing business requirements."

Terminology: Legacy System

- There are a number of options available to manage legacy systems. Typical solution include:
  - **Freeze**: The organization decides no further work on the legacy system should be performed.
  - **Outsource**: An organization may decide that supporting software is not core business, and may outsource it to a specialist organization offering this service.
  - **Carry on maintenance**: Despite all the problems of support, the organization decides to carry on maintenance for another period.
  - **Discard and redevelop:** Throw all the software away and redevelop the application once again from scratch.
  - **Wrap:** It is a black-box modernization technique – surrounds the legacy system with a software layer that hides the unwanted complexity of the existing data, individual programs, application systems, and interfaces with the new interfaces.
  - **Migrate:** Legacy system migration basically moves an existing, operational system to a new platform, retaining the legacy system's functionality and causing minimal disruption to the existing operational business environment as possible.



Evolution and Maintenance

"Evolution is what happens while you're busy making other plans."

Usually, we consider evolution to begin once the first version has been delivered:

- Maintenance *is the planned set of tasks to effect changes.*
- Evolution *is what actually happens to the software.*

Software Evolution

In 1965, Mark Halpern used the term *evolution* to define the dynamic growth of software.

The term evolution in relation to application systems took gradually in the 1970s.

Lehman and his collaborators from IBM are generally credited with pioneering the research field of software evolution.

Lehman formulated a set of observations that he called laws of evolution.

These laws are the **results of studies** of the evolution of large-scale proprietary or closed source system (CSS).

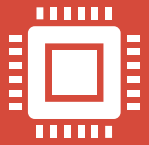
The laws concern what Lehman called E-type systems:

- Monolithic systems produced by a team within an organization that solves a real world problem and have human users.

Software Evolution: Laws of Lehman

Continuing change (1st) – A system will become progressively less satisfying to its user over time, unless it is continually adapted to meet new needs.

Increasing complexity (2nd) – A system will become progressively more complex, unless work is done to explicitly reduce the complexity.



Self-regulation (3rd) – The process of software evolution is self regulating with respect to the distributions of the products and process artifacts that are produced.

Conservation of organizational stability (4th) – The average effective global activity rate on an evolving system does not change over time, that is the average amount of work that goes into each release is about the same.

Software Evolution: Laws of Lehman

Conservation of familiarity (5th) – The amount of new content in each successive release of a system tends to stay constant or decrease over time.

Continuing growth (6th) – The amount of functionality in a system will increase over time, in order to please its users.

Declining quality (7th) – A system will be perceived as losing quality over time, unless its design is carefully maintained and adapted to new operational constraints.

Feedback system (8th) – Successfully evolving a software system requires recognition that the development process is a multi-loop, multi-agent, multi-level feedback system.

Applicability of Lehman's laws

Lehman's laws seem to be generally applicable to large, tailored systems developed by large organisations.

Confirmed in early 2000's by work by Lehman

It is not clear how they should be modified for

Systems that incorporate a significant number of COTS components	Small organisations	Medium sized systems

Software Evolution: FOSS System

PIRZADA POINTED OUT THE DIFFERENCES BETWEEN THE EVOLUTION OF THE UNIX OS AND SYSTEM STUDIED BY LEHMAN

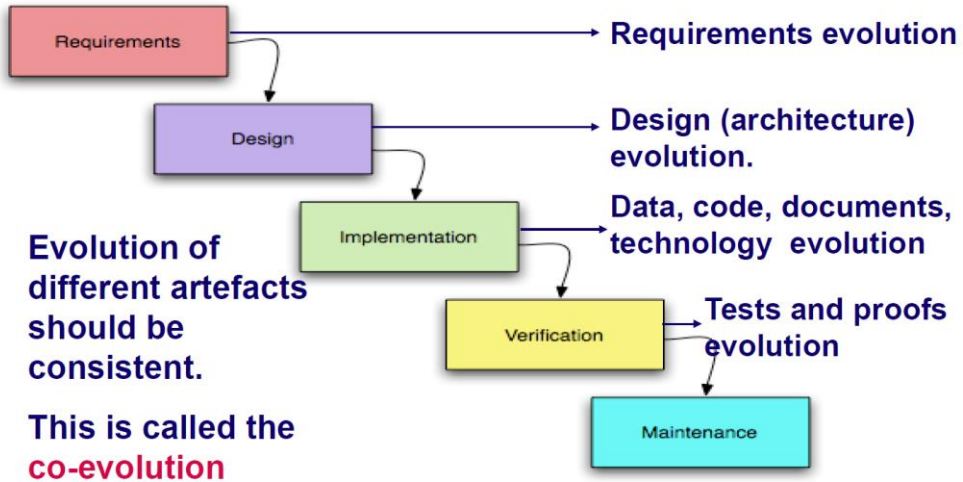
PIRZADA ARGUED THAT THE DIFFERENCES IN ACADEMIC AND INDUSTRIAL S/W DEVELOPMENT COULD LEAD TO A DIFFERENCES IN THE EVOLUTIONARY PATTERN.

IN CIRCA 2000, EMPIRICAL STUDY OF FREE AND OPEN SOURCE SOFTWARE (FOSS) EVOLUTION WAS CONDUCTED BY GODFREY AND TU.

THEY FOUND THAT THE GROWTH TRENDS FROM 1994-1999 FOR THE EVOLUTION OF FOSS LINUX OS TO BE SUPER-LINEAR, THAT IS GREATER THAN LINEAR.

ROBLES AND HIS COLLABORATOR CONCLUDED THAT LEHMAN'S LAWS, 3, 4, AND 5 ARE NOT FITTED TO LARGE SCALE FOSS SYSTEM SUCH AS LINUX.

"FOSS IS MADE AVAILABLE TO THE GENERAL PUBLIC WITH EITHER RELAXED OR NON-EXISTENT INTELLECTUAL PROPERTY RESTRICTIONS. THE FREE EMPHASIZES THE FREEDOM TO MODIFY AND REDISTRIBUTE UNDER THE TERMS OF THE ORIGINAL LICENSE WHILE OPEN EMPHASIZES THE ACCESSIBILITY TO THE SOURCE CODE."



What is actually evolving?

Software maintenance is a **natural continuation** of the development process (specification, design, implementation, testing). Hence the term evolution applied especially when the transition from development is seamless.

Development and maintenance **costs vary** from application to application

Investing in development leads to reduction of both maintenance costs and overall project costs.

Importance of Software Evolution

Organization have made huge investments in their software systems – they are critical business assets.

To maintain the value of these assets to the business, they must be changed and updated.

The majority of the software budget in large companies is devoted to evolving existing software rather than developing new software.

Studies indicate that up to 75% of all software professionals are involved in some form of maintenance/ evolvability activity.

What is your opinion?

- Can we compare: Aging of software with aging of people?
  - Does software age?
  - What are characteristics of aging?
  - Are they applicable in some form to both software and people?

What is Software "Aging" – Opinion #1

- *"It does **not** make sense to talk about software aging!"*

  Software is a mathematical product; mathematics does not decay with time.

  if a theorem was correct 200 years ago, it will be correct tomorrow.

  If a program is correct today, it will be correct 100 years from now.

  If a program is wrong 100 years from now, it must have been wrong when it was written.

  All of the above statements are true, but are they really relevant?

Software Aging

"Programs, like people, get old. We can't prevent aging, 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. … (We must) lose our preoccupation with the first release and focus on the long term health of our products."

Parnas, D.L. Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on Volume , Issue , 16-21 May 1994 Page(s):279 - 287

Software **Does Age** (Opinion #2)

- *Software aging is gaining in significance because:*
  - *of the growing economic importance of software,*
  - *software is the "capital" of many high-tech firms .*
  - *"If only the software had been designed using today's languages and techniques …"*
- *Like a young jogger scoffing at an 86 year old man (ex-champion swimmer) and saying that he should have exercised more in his youth!*

Software Aging - Causes

Lack of Movement: *Aging caused by the failure of the product's owners to modify it to meet changing needs.*

- Unless software is frequently updated, its user's will become dissatisfied and change to a new product.
- Excellent software developed in the 60's would work perfectly well today, but nobody would use it.
- That software has aged even though nobody has touched it. Actually, it has aged because nobody bothered to touch it.

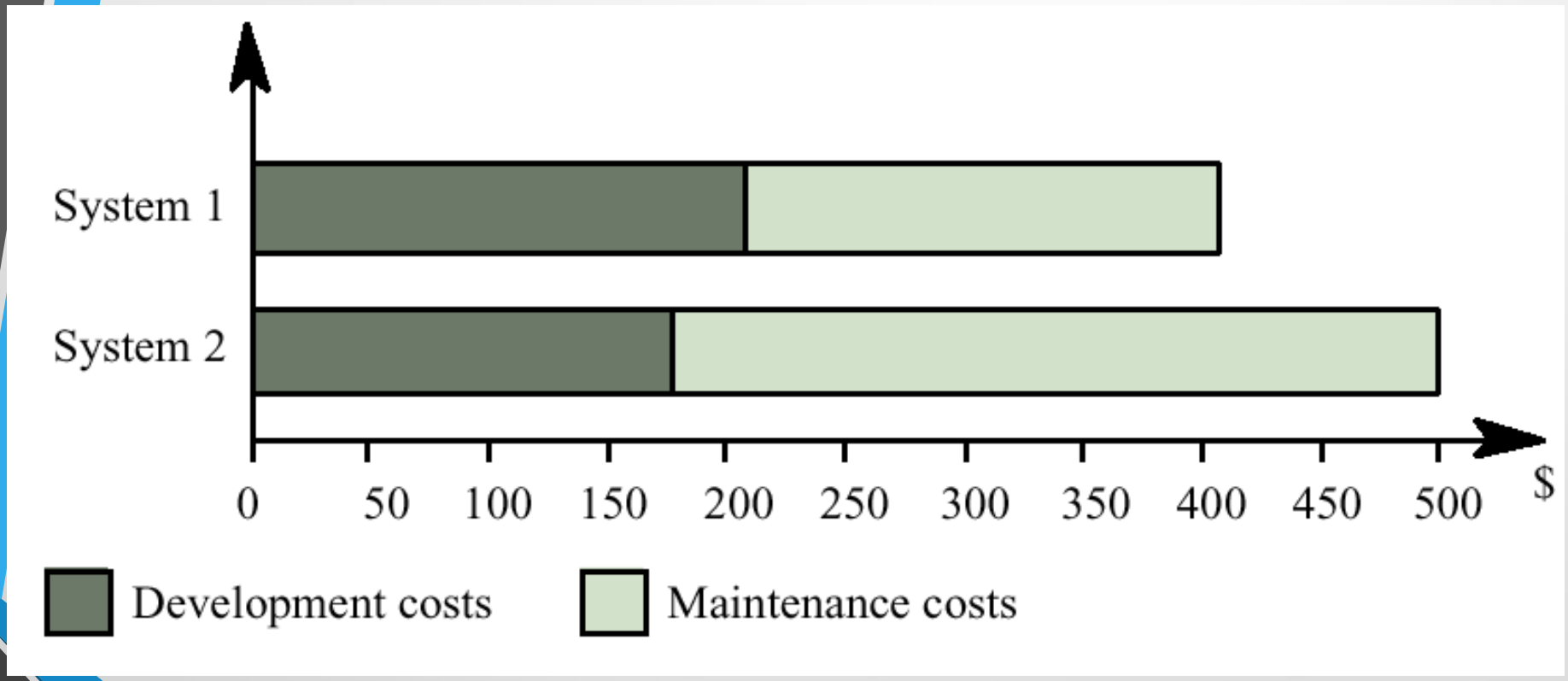
Software Aging - Causes

Ignorant Surgery: *Aging caused as a result of changes that are made.*

This "one-two punch" can lead to rapid decline in the value of a software product.

- One must upgrade software to prevent aging.
- Changing software can cause aging too.
- Changes are made by people who do not understand the software. Hence, software structure degrades.
- After many such changes nobody understands the software:
  - the original designers no longer understand the modified software,
  - those who made the modification still do not understand the software.
- Changes take longer and introduce new bugs.
- Inconsistent and inaccurate documentation makes changing the software harder to do.

Development versus maintenance costs



A more detailed look at what is Software Maintenance?

Software evolves **continuously** due to demands for changes:

- New requirements surface
- Existing requirements need be modified
- Errors found need be fixed

Activities to make corrections:

- If there are discrepancies between the expected behavior of a system and the actual behavior, then some activities are performed to eliminate or reduce the discrepancies.

In some cases 90% of software costs are maintenance costs

Four categories:

- **Perfective maintenance:** changes required as a result of user requests (a.k.a. evolutive maintenance)
- **Adaptive maintenance:** changes needed as a consequence of operated system, hardware, or DBMS changes
- **Corrective maintenance:** the identification and removal of faults in the software
- **Preventative maintenance:** changes made to software to make it more maintainable

Two Types of Maintenance Activities – Bug Fixing

- Corrective Maintenance
  - Identify and remove defects
  - Correct actual errors

- *Preventive Maintenance*
  - *Identify and detect latent faults*
  - *Systems with safety concerns*

- *Emergency Maintenance*
  - *Unscheduled corrective maintenance*

  (Risks due to reduced testing)

Two Types of Maintenance Activities – Development/Migration

Post-Delivery

- Perfective Maintenance
  - Improve performance, dependability, maintainability
  - Update documentation

- *Adaptive Maintenance*
  - *Adapt to a new/upgraded environment (e.g., hardware, operating system, middleware)*
  - *Incorporate new capability*

Development/Migration

Five Types of Software Maintenance

- Corrective Maintenance
  - Identify and remove defects
  - Correct actual errors

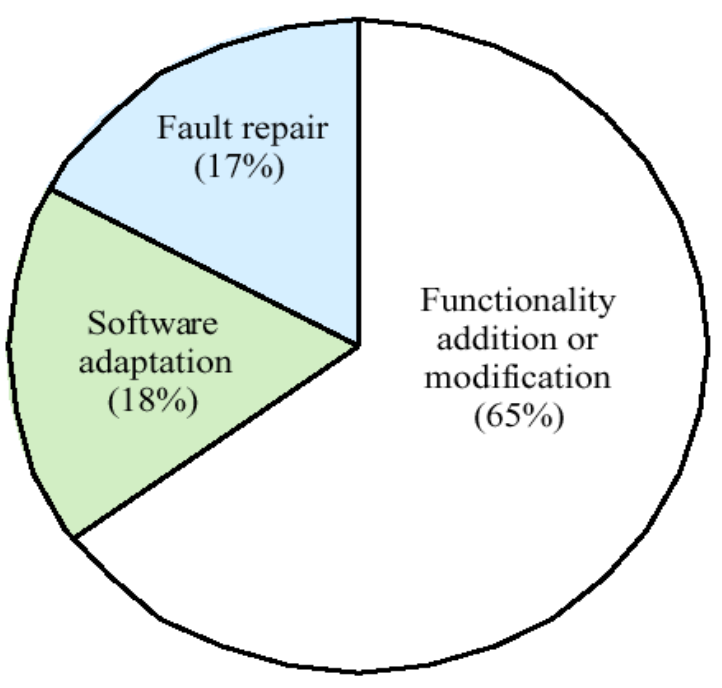
- *Adaptive Maintenance*
  - *Adapt to a new/upgraded environment (e.g., hardware, operating system, middleware)*
  - *Incorporate new capability*

- *Preventive Maintenance*
  - *Identify and detect latent faults*
  - *Systems with safety concerns*

- Perfective Maintenance
  - Improve performance, dependability, maintainability
  - Update documentation

- *Emergency Maintenance*
  - *Unscheduled corrective maintenance*

(Risks due to reduced testing)



Maintenance cost

Why maintenance costs are higher than development costs?

Factors:

- Team stability: development teams break up after delivery
- Contractual responsibility: different teams or organizations have the responsibility for maintenance
- Staff skills: more experienced software engineers tend to avoid maintenance
- Program age and structure: not structured in the first place, the program copes poorly with changes and its structure degrades

Video