Lehman's laws of software evolution

Three categories of software:

- An S-program is written according to an exact **specification** of what that program can do
- A P-program is written to implement certain **procedures** that completely determine what the program can do (the example mentioned is a program to play chess)
- An E-program is written to perform some real-world activity; how it should behave is strongly linked to the **environment** in which it runs, and such a program needs to adapt to varying requirements and circumstances in that environment

*Lehman, Meir M. (1980). "Programs, Life Cycles, and Laws of Software Evolution". Proc. IEEE. **68** (9): 1060–1076. doi:10.1109/proc.1980.11805*

Lehman's laws of software evolution

- **Continuing Change** — an E-type system must be continually adapted or it becomes progressively less satisfactory
- **Increasing Complexity** — as an E-type system evolves, its complexity increases unless work is done to maintain or reduce it
- **Self Regulation** — E-type system evolution processes are self-regulating with the distribution of product and process measures close to normal
- **Conservation of Organisational Stability** (invariant work rate) - the average effective global activity rate in an evolving E-type system is invariant over the product's lifetime
- **Conservation of Familiarity** — as an E-type system evolves, all associated with it, developers, sales personnel and users, for example, must maintain mastery of its content and behaviour to achieve satisfactory evolution. Excessive growth diminishes that mastery. Hence the average incremental growth remains invariant as the system evolves.
- **Continuing Growth** — the functional content of an E-type system must be continually increased to maintain user satisfaction over its lifetime
- **Declining Quality** — the quality of an E-type system will appear to be declining unless it is rigorously maintained and adapted to operational environment changes
- **Feedback System** — E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base

Software change

- Software change is inevitable
  - New requirements emerge when the software is used;
  - The business 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.
- A key problem for all organizations is implementing and managing change to existing software systems.

3

Importance of evolution

- Organisations have 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.
- A significant proportion of the software budget in large companies is devoted to changing and evolving existing software rather than developing new software.

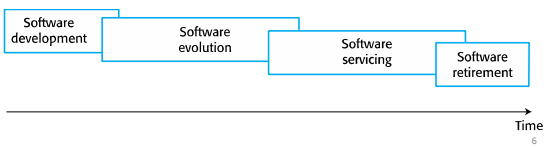
4

Software enhancement &maintenance projects

- Here are some examples of enhancement and maintenance projects:
  - Adding a new feature or function to an existing system
  - Implementing a business policy change
  - Correcting a problem with the current system or improving the performance of operational software
  - Porting (moving software components) operational software to a different hardware platform

Evolution and servicing

- Evolution
  - The stage in a software system's life cycle where it is in operational use and is evolving as new requirements are proposed and implemented in the system.
- Servicing
  - At this stage, the software remains useful but the only changes made are those required to keep it operational i.e. bug fixes and changes to reflect changes in the software's environment. No new functionality is added.
- Phase-out
  - The software may still be used but no further changes are made to it.

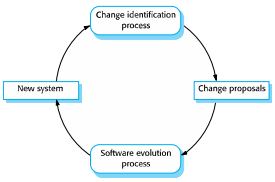


6

Evolution processes

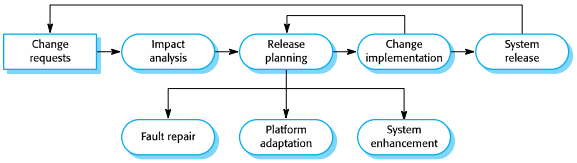
- Software evolution processes depend on
  - The type of software being maintained;
  - The development processes used;
  - The skills and experience of the people involved.
- Proposals for change are the driver for system evolution.
  - Should be linked with components that are affected by the change, thus allowing the cost and impact of the change to be estimated.
- Change identification and evolution continues throughout the system lifetime.

Change identification & evolution processes



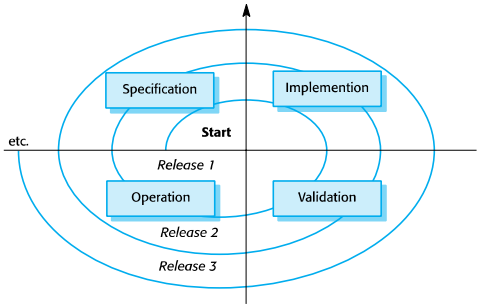
7

The software evolution process



8

A spiral model of development and evolution



9

Change implementation

- Iteration of the development process where the revisions to the system are designed, implemented and tested.
- A critical difference is that the first stage of change implementation **may involve program understanding**, especially if the original system developers are not responsible for the change implementation. **This can be expensive.**
- During the program understanding phase, you have to understand how the program is structured, how it delivers functionality and **how the proposed change might affect the program**.

10

Urgent change requests

- **Urgent** changes may have to be implemented without going through all stages of the software engineering process
  - If a serious system fault has to be repaired to allow normal operation to continue;
  - If changes to the system's environment (e.g. an OS upgrade) have unexpected effects;
  - If there are business changes that require a very rapid response (e.g. the release of a competing product).

The emergency repair process

11

Agile methods and evolution

- Agile methods are based on incremental development so the transition from development to evolution is a **seamless** one.
  - Evolution is simply a continuation of the development process based on frequent system releases.
- **Automated regression testing** is particularly valuable when changes are made to a system.
- Changes may be expressed as **additional user stories.**
- But can **design decay in agile environments inflate evolution/maintenance costs?**

12

Handover problems

- Where the development team have used an agile approach but the evolution team is unfamiliar with agile methods and prefer a plan-based approach.
  - The evolution team may expect detailed documentation to support evolution and this is not produced in agile processes.
- Where a plan-based approach has been used for development but the evolution team prefer to use agile methods.
  - The evolution team may have to start from scratch developing automated tests and the code in the system may not have been refactored and simplified as is expected in agile development.
- Recall that DevOps seeks to foster more commonality between development and ops, thereby overcoming some of these problems.

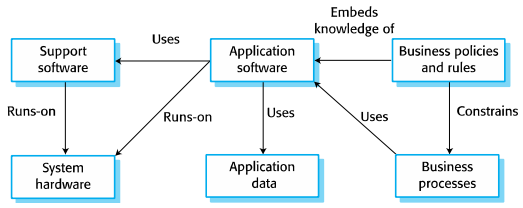
13

Legacy systems

- Legacy systems are **older systems** that **rely on languages and technology that are no longer (commonly) used** for new systems development.
- Legacy software may be dependent on older **hardware**, such as mainframe computers and may have associated **legacy processes and procedures**.
- Legacy systems are not just software systems but are broader socio-technical systems that include hardware, software, libraries and other supporting software and business processes.

14

The elements of a legacy system



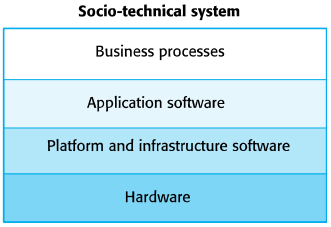
15

Legacy system components

- *System hardware* Legacy systems may have been written for hardware that is no longer available.
- *Support software* The legacy system may rely on a range of support software, which may be obsolete or unsupported.
- *Application software* The application system that provides the business services is usually made up of a number of application programs.
- *Application data* These are data that are processed by the application system. They may be inconsistent, duplicated or held in different databases.
- *Business processes* These are processes that are used in the business to achieve some business objective.
- Business processes may be designed around a legacy system and constrained by the functionality that it provides.
- *Business policies and rules* These are definitions of how the business should be carried out and constraints on the business. Use of the legacy application system may be embedded in these policies and rules.

16

Legacy system layers



17

Legacy system replacement

- Legacy system **replacement is risky and expensive** so businesses continue to use these systems
- System replacement is risky for a number of reasons
  - Lack of complete system specification
  - Tight integration of system and business processes
  - Undocumented business rules embedded in the legacy system
  - New software development may be late and/or over budget

18

Legacy system change

- Legacy systems are expensive to change for a number of reasons:
  - No consistent programming style
  - Use of obsolete programming languages with few people available with these language skills
  - Inadequate system documentation
  - System structure degradation
  - Program optimizations may make them hard to understand
  - Data errors, duplication and inconsistency

19

Legacy system management

- Organisations that rely on legacy systems must choose a **strategy for evolving** these systems
  - Scrap the system completely and modify business processes so that it is no longer required;
  - Continue maintaining the system;
  - Transform the system by re-engineering to improve its maintainability;
  - Replace the system with a new system.
- The **strategy** chosen should depend on the system quality and its business value.

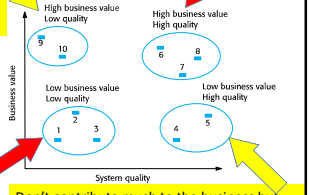
20

An example of a legacy system assessment

Important business contribution so they cannot be scrapped. However, their low quality means that it is expensive to maintain them. These systems should be reengineered to improve their quality.

Have to be kept in operation. However, their high quality means that you don't have to invest in transformation or system replacement

- For example, assume that an organization has 10 legacy systems.
- You should assess the quality and the business value of each of these systems.
- You may then create a chart showing relative business value and system quality.



Keeping these systems in operation will be expensive and the rate of the return to the business will be fairly small. These systems should be scrapped

Don't contribute much to the business but may not be very expensive to maintain. Its not worth replacing these systems so normal system maintenance may be continued if expensive changes are not required

Business value assessment

- Assessment should take **different viewpoints** into account
  - System end-users;
  - Business customers;
  - Line managers;
  - IT managers;
  - Senior managers.
- Interview different stakeholders and collate results.

22

Issues in business value assessment

- The use of the system
  - If systems are only used occasionally or by a small number of people, they may have a low business value.
- The business processes that are supported
  - A system may have a low business value if it forces the use of inefficient business processes.
- System dependability
  - If a system is not dependable and the problems directly affect business customers, the system has a low business value.
- The system outputs
  - If the business depends on system outputs, then the system has a high business value.

23

System quality assessment

- Business process assessment
  - How well does the business process support the current goals of the business?
- Environment assessment
  - How effective is the system's environment and how expensive is it to maintain?
- Application assessment
  - What is the quality of the application software system?

24

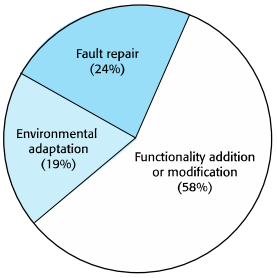
Software maintenance

- Modifying a program after it has been put into use.
- The term is mostly used for changing custom software. Generic software products are said to evolve to create new versions.
- Maintenance does not normally involve major changes to the system's architecture.
- Changes are implemented by modifying existing components and adding new components to the system.

31

Types of maintenance

- Fault repairs
  - Changing a system to fix bugs/vulnerabilities and correct deficiencies in the way meets its requirements.
- Environmental adaptation
  - Maintenance to adapt software to a different operating environment
  - Changing a system so that it operates in a different environment (computer, OS, etc.) from its initial implementation.
- Functionality addition and modification
  - Modifying the system to satisfy new requirements.



Sample distribution of maintenance costs

32

Types of Maintenance

- Corrective
  - correcting faults in system behaviour
  - caused by errors in coding, design or requirements
- Adaptive
  - due to changes in operating environment
  - e.g., different hardware or Operating System
- Perfective
  - due to changes in requirements
  - Often triggered by organizational, business or user learning
- Preventive
  - dealing with legacy systems

Maintenance costs

- Usually greater than development costs (2X to (100+X) depending on the application).
- Affected by both technical and non-technical factors.
- Increases as software is maintained.
- Maintenance corrupts the software structure so makes further maintenance more difficult.
- Ageing software can have high support costs (e.g. old languages, compilers etc.).

34

Development vs. Maintenance

- Team Stability
  - After a system has been delivered, it is normal for the development team to be broken up and people work on new projects
  - The new team or the individuals responsible for system maintenance do not understand the system or the background to system design decisions
  - A lot of the effort during maintenance process is taken up with understanding the existing system before implementing changes to it
- Contractual Responsibility
  - The contract to maintain the system is usually separate from the system development contract
  - The maintenance contract may be given to a different company rather than the original system developer
  - No incentive to write easy maintainable software
  - Reducing development cost may increase maintenance cost
- Staff Skills
  - Maintenance staff may be relatively inexperienced and unfamiliar with the application domain
  - Maintenance has a poor image among software engineering
- Program Age and Structure
  - As programs age, their structure tends to be degraded by change, so they become harder to understand and modify

Maintenance is Hard

- Key design concept not captured
- Systems not robust under **change**
- Poor **documentation**
  - of code
  - of design process and rationale
  - of system's evolution
- "stupid" code **features** may not be so stupid
  - Work-arounds of artificial constraints may no longer be documented (e.g., Operating System bugs, undocumented features, memory limits, etc.)
- Poor management attitudes (**culture**)
  - Maintenance not "cool"
  - It is just "patchy code"
  - Easier/less important than design (does not need similar level of support – tools, modelling, documentation, management, etc.)

Maintenance costs

- It is usually more expensive to add new features to a system during maintenance than it is to add the same features during development
  - A new team has to understand the programs being maintained
  - Separating maintenance and development means there is no incentive for the development team to write maintainable software
  - Program maintenance work is unpopular
    - Maintenance staff are often inexperienced and have limited domain knowledge.
  - As programs age, their structure degrades and they become harder to change

37

Maintenance costs example

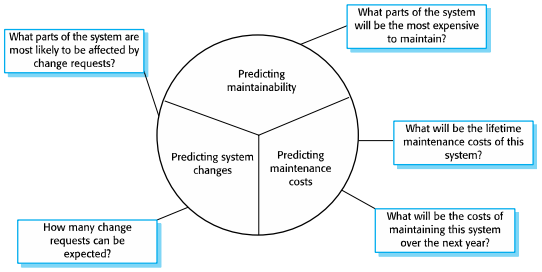
- Some Maintenance Statistics
  - Maintenance may consume >40% of total costs
  - Typical developer's activity (from Lients and Swanston's review of 487 companies):
    - 48% Maintenance
    - 46.1% New Development
    - 5.9% other
- Huge quantities of legacy code out there:
  - US/DoD maintains more than 1.4 billion LOC (Line Of Code) for non-combat information systems, over more that 1700 data centres.
  - Estimated to cost billions of $$s per annum.

Maintenance prediction

- Maintenance prediction is concerned with assessing which parts of the system may cause problems and have high maintenance costs
  - Change acceptance depends on the maintainability of the components affected by the change;
  - Implementing changes degrades the system and reduces its maintainability;
  - Maintenance costs depend on the number of changes and costs of change depend on maintainability.

39

Maintenance prediction



40

Change prediction

- Predicting the number of changes requires an understanding of the relationships between a system and its environment.
- Tightly coupled systems require changes whenever the environment is changed.
- Factors influencing this relationship are
  - Number and complexity of system interfaces;
  - Number of inherently volatile system requirements;
  - The business processes where the system is used.

41

Complexity metrics

- Predictions of maintainability can be made by assessing the complexity of system components.
- Studies have shown that most maintenance effort is spent on a relatively small number of system components.
- Complexity depends on
  - Complexity of control structures;
  - Complexity of data structures;
  - Object, method (procedure) and module size.
- Complexity metrics tell us about the difficulty or error proneness associated with changing a particular segment of software.

42

Process metrics

- Process metrics may be used to assess maintainability
  - Number of requests for corrective maintenance;
  - Average time required for impact analysis;
  - Average time taken to implement a change request;
  - Number of outstanding change requests.
- If any or all of these is increasing, this may indicate a decline in maintainability.

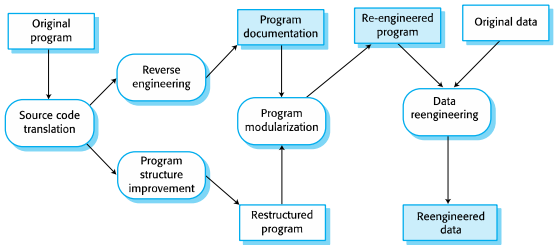
43

Software reengineering

- **To make legacy software systems easier to maintain, you can reengineer these systems to improve their structure and understandability.**
- Restructuring or rewriting part or all of a legacy system without changing its functionality.
- Applicable where some but not all sub-systems of a larger system require frequent maintenance.
- Reengineering involves adding effort to make them easier to maintain.
- The system may be re-structured and re-documented.

44

The reengineering process



45

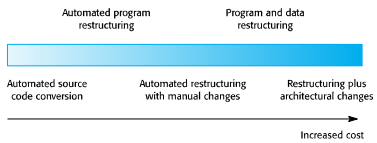
Advantages of reengineering

- Reduced risk
  - There is a high risk in new software development. There may be development problems, staffing problems and specification problems.
- Reduced cost
  - The cost of re-engineering is often significantly less than the costs of developing new software.

46

Reengineering cost factors

- The quality of the software to be reengineered.
- The tool support available for reengineering.
- The extent of the data conversion which is required.
- The availability of expert staff for reengineering.
  - This can be a problem with old systems based on technology that is no longer widely used.



- There is a spectrum of possible approaches to reengineering.
- Costs increase from left to right so that source code translation is the cheapest option. Reengineering as part of architectural migration is the most expensive.

47

Refactoring

- Refactoring is the process of making improvements to a program to slow down degradation through change.
- You can think of refactoring as 'preventative maintenance' that reduces the problems of future change.
- Refactoring involves modifying a program to improve its structure, reduce its complexity or make it easier to understand.
- When you refactor a program, you should not add functionality but rather concentrate on program improvement.

48

Refactoring and reengineering

- **Re-engineering** takes place after a system has been maintained for some time and maintenance costs are increasing.
  - You use automated tools to process and re-engineer a legacy system to create a new system that is more maintainable.
- **Refactoring** is a continuous process of improvement throughout the development and evolution process.
  - It is intended to avoid the structure and code degradation that increases the costs and difficulties of maintaining a system.

49

Key points

- Software development and evolution can be thought of as an integrated, iterative process that can be represented using a spiral model.
- For custom systems, the costs of software maintenance usually exceed the software development costs.
- The process of software evolution is driven by requests for changes and includes change impact analysis, release planning and change implementation.
- Legacy systems are older software systems, developed using obsolete software and hardware technologies, that remain useful for a business.

50

Key points

- It may be cheaper and less risky to maintain a legacy system than to develop a replacement system using modern technology.
- The business value of a legacy system and the quality of the application should be assessed to help decide if a system should be replaced, transformed or maintained.
- There are 3 types of software maintenance, namely bug fixing, modifying software to work in a new environment, and implementing new or changed requirements.
- Software re-engineering is concerned with re-structuring and re-documenting software to make it easier to understand and change.
- Refactoring, making program changes that preserve functionality, is a form of preventative maintenance.

51