XVI. Software Evolution

Andrea Polini

Objectives

- To explain why change is inevitable if software systems are to remain useful
- To discuss software maintenance and maintenance cost factors
- To describe the processes involved in software evolution
- To discuss an approach to assessing evolution strategies for legacy systems

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 is added to the system;
  - The performance or reliability of the system may have to be improved.
- A key problem for organisations is implementing and managing change to their existing software systems.

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

Program evolution dynamics

- **Program evolution** dynamics is the study of the processes of system change.
- After major empirical studies, Lehman and Belady proposed that there were a number of 'laws' which applied to all systems as they evolved.
- There are sensible observations rather than laws. They are applicable to large systems developed by large organisations. Perhaps less applicable in other cases.

Lehman's Law

Law	Description
Continuing change	A program that is used in a real-world environment necessarily must change or become progressively less useful in that environment.
Increasing complexity	As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplifying the structure.
Large program evolution	Program evolution is a self-regulating process. System attributes such as size, time between releases and the number of reported errors is approximately invariant for each system release.
Organisational stability	Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.
Conservation of familiarity	Over the lifetime of a system, the incremental change in each release is approximately constant.
Continuing growth	The functionality offered by systems has to continually increase to maintain user satisfaction.
Declining quality	The quality of systems will appear to be declining unless they are adapted to changes in their operational environment.
Feedback system	Evolution processes incorporate multi-agent, multi-loop feedback systems and you have to treat them as feedback systems to achieve significant product improvement.

Applicability of Lehman's Law

- Lehman’s laws seem to be generally applicable to large, tailored systems developed by large organisations.
  - Confirmed in more recent work by Lehman on the FEAST project (see further reading on book website).
- It is not clear how they should be modified for
  - Shrink-wrapped software products;
  - Systems that incorporate a significant number of COTS components;
  - Small organisations;
  - Medium sized systems.

Software Maintenance

- Modifying a program after it has been put into use.
- 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.

Maintenance is inevitable

- The system requirements are likely to change while the system is being developed because the environment is changing. Therefore a delivered system won't meet its requirements!
- Systems are tightly coupled with their environment. When a system is installed in an environment it changes that environment and therefore changes the system requirements.
- Systems MUST be maintained therefore if they are to remain useful in an environment.

Types of Maintenance

- Maintenance to repair software faults (corrective)
  - Changing a system to correct deficiencies in the way meets its requirements.
- Maintenance to adapt software to a different operating environment (adaptive)
  - Changing a system so that it operates in a different environment (computer, OS, etc.) from its initial implementation.
- Maintenance to add to or modify the system's functionality (perfective)
  - Modifying the system to satisfy new requirements

Maintenance Costs

- Usually greater than development costs (2* to 100* 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.).

Maintenance Cost Factors

- Team stability
  - Maintenance costs are reduced if the same staff are involved with them for some time.
- Contractual responsibility
  - The developers of a system may have no contractual responsibility for maintenance so there is no incentive to design for future change.
- Staff skills
  - Maintenance staff are often inexperienced and have limited domain knowledge.
- Program age and structure
  - As programs age, their structure is degraded and they become harder to understand and change.

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.

Change Prediction

- Predicting the number of changes requires and 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.

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.

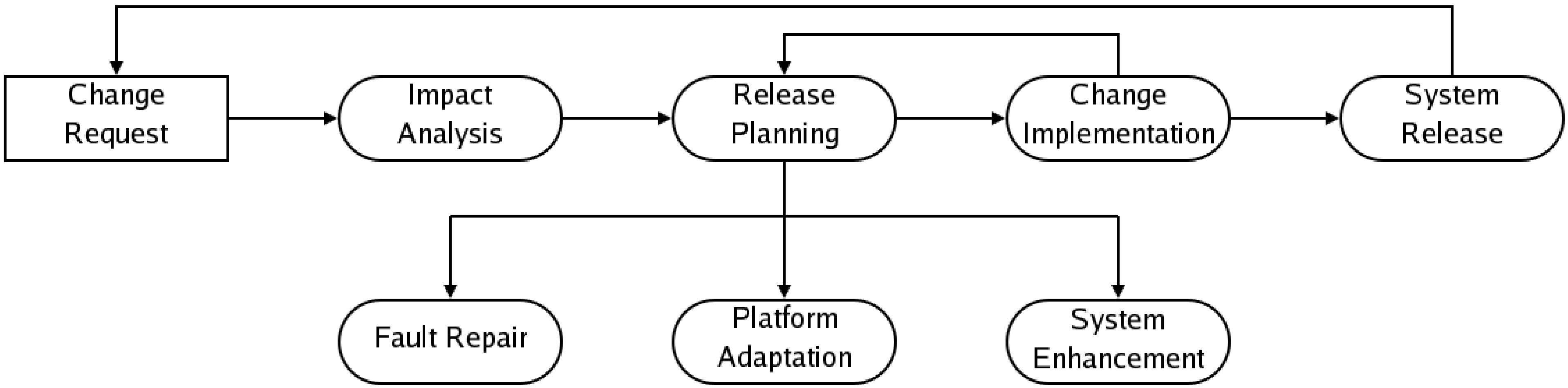
Process Metrics

- Process measurements 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.

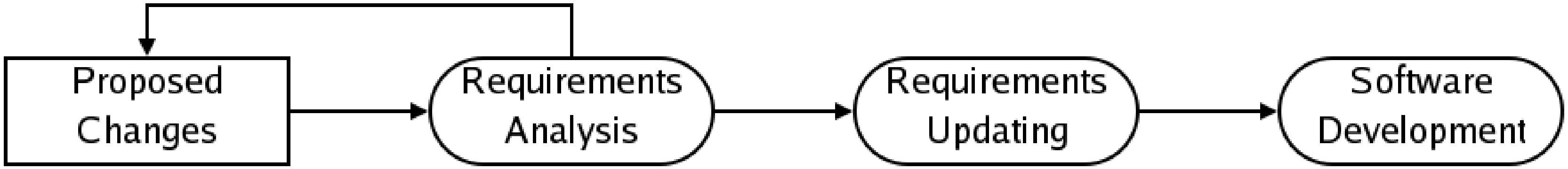
Evolution Processes – Motivations

- 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. Change identification and evolution continue throughout the system lifetime
  - Existing requirements not implemented in previous release
  - Request for new requirements
  - Bur repairs

The System Evolution Process

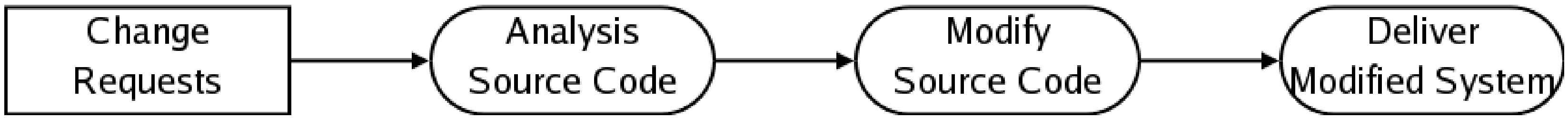


- Revision of the Development process where the revision are designed, implemented and tested
  - Basic difference is that the initial stage is program understanding
- During the change implementation requirements analysis could lead to the identification of new changes and the revision of those planned



Emergency Repair Process

- 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;
  - 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).



- Emergency repair leads faster to ageing effects
  - Possible solution: "fix and then rework". But....

System Re-engineering

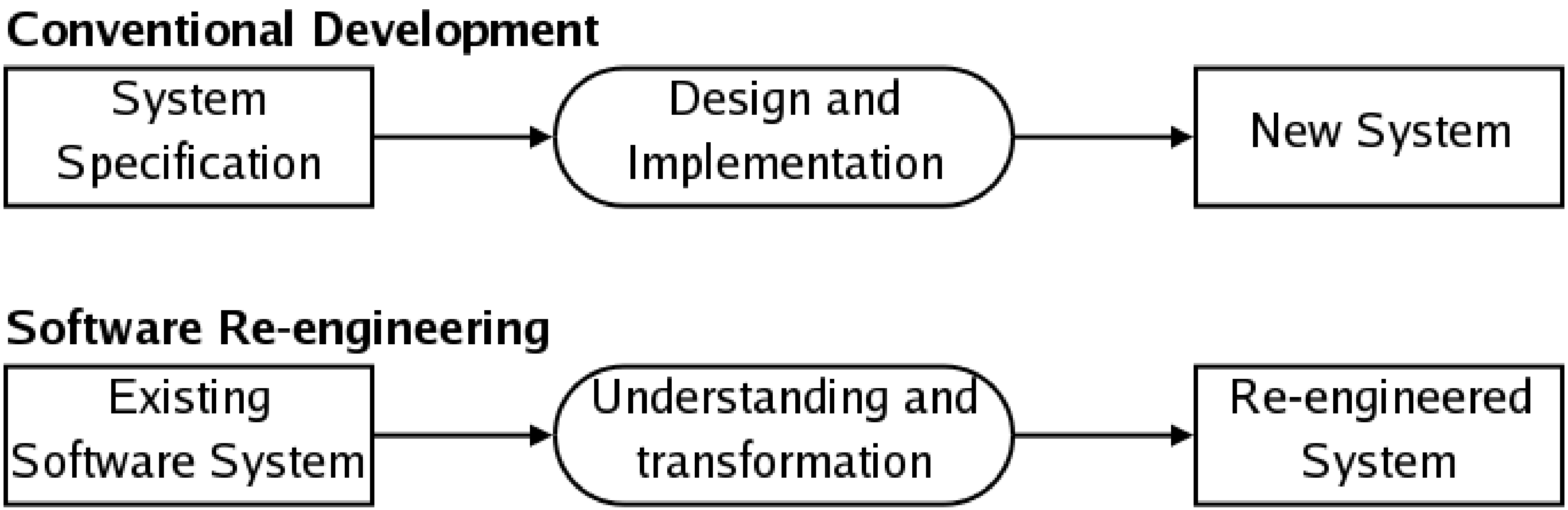
- System re-engineering concerns with **re-structuring or re-writing part or all of a legacy system** without changing its functionality.
- Applicable where some but not all sub-systems of a larger system require **frequent maintenance.**
- Re-engineering involves **adding effort to make them easier to maintain.** The system may be re-structured and re-documented.

Advantages of re-engineering

- Two key advantages over redevelopment:
- **Reduced risk**
  - There is a high risk in new software development. There may be development problems, staffing problems and specification problems. Planning is difficult and delays are expensive
- **Reduced cost**
  - The cost of re-engineering is often significantly less than the costs of developing new software.

Conventional Development vs. Re-engineering

- Development start with requirements, and proceeds through the studied phases
- In re-engineering the old system acts as a specification for the new system



Typical Activities of the Re-engineering Process

- Source code translation
  - Convert code to a new language.
- Reverse engineering
  - Analyse the program to understand it;
- Program structure improvement
  - Restructure automatically for understandability;
- Program modularisation
  - Reorganise the program structure;
- Data reengineering
  - Clean-up and restructure system data.

Re-engineering 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

- Limit to re-engineering
  - e.g. Converting code written using functional languages to OO

Legacy system evolution

- 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.

Legacy systems – business and technical perspective

- **Low quality, low business value**
  - These systems should be scrapped.
- **Low-quality, high-business value**
  - These make an important business contribution but are expensive to maintain. Should be re-engineered or replaced if a suitable system is available.
- **High-quality, low-business value**
  - Replace with COTS, scrap completely or maintain.
- **High-quality, high business value**
  - Continue in operation using normal system maintenance.

Identify Quality

- Business quality: usage, supported activities, dependability
- Technical quality:
  - Environmental
  - Application

Factors for Environmental Assessment

Factor	Questions
Supplier stability	Is the supplier is still in existence? Is the supplier financially stable and likely to continue in existence? If the supplier is no longer in business, does someone else maintain the systems?
Failure rate	Does the hardware have a high rate of reported failures? Does the support software crash and force system restarts?
Age	How old is the hardware and software? The older the hardware and support software, the more obsolete it will be. It may still function correctly but there could be significant economic and business benefits to moving to more modern systems.
Performance	Is the performance of the system adequate? Do performance problems have a significant effect on system users?
Support requirements	What local support is required by the hardware and software? If there are high costs associated with this support, it may be worth considering system replacement.
Maintenance costs	What are the costs of hardware maintenance and support software licences? Older hardware may have higher maintenance costs than modern systems. Support software may have high annual licensing costs.
Interoperability	Are there problems interfacing the system to other systems? Can compilers etc. be used with current versions of the operating system? Is hardware emulation required?

Factors for Application Assessment

Factor	Questions
Understandability	How difficult is it to understand the source code of the current system? How complex are the control structures that are used? Do variables have meaningful names that reflect their function?
Documentation	What system documentation is available? Is the documentation complete, consistent and up-to-date?
Data	Is there an explicit data model for the system? To what extent is data duplicated in different files? Is the data used by the system up-to-date and consistent?
Performance	Is the performance of the application adequate? Do performance problems have a significant effect on system users?
Programming language	Are modern compilers available for the programming language used to develop the system? Is the programming language still used for new system development?
Configuration management	Are all versions of all parts of the system managed by a configuration management system? Is there an explicit description of the versions of components that are used in the current system?
Test data	Does test data for the system exist? Is there a record of regression tests carried out when new features have been added to the system?
Personnel skills	Are there people available who have the skills to maintain the application? Are there only a limited number of people who understand the system?