IT3205: Fundamentals of Software Engineering (Compulsory)

BIT – 2nd Year

Semester 3

UCSC

IT3205: Fundamentals of Software Engineering

Software Maintenance

Duration: 3 hours

Learning Objectives

- Describe the types of software maintenance
- Describe the software maintenance process
- Describe activities of configuration management

Detailed Syllabus

7.1 Evolving nature of software

7.1.1 Different types of maintenance

- 7.1.1.1 Fault repair
- 7.1.1.2 Software adaptation
- 7.1.1.3 Functionality addition or modification

7.1.2 Maintenance prediction

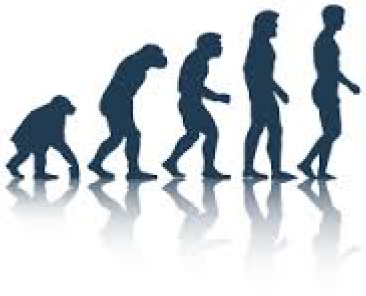
7.1.3 Re-engineering

Detailed Syllabus

7.2 Configuration Management (CM)

- 7.2.1 Importance of CM
- 7.2.2 Configuration items
- 7.2.3 Versioning

7.1 EVOLVING NATURE OF SOFTWARE



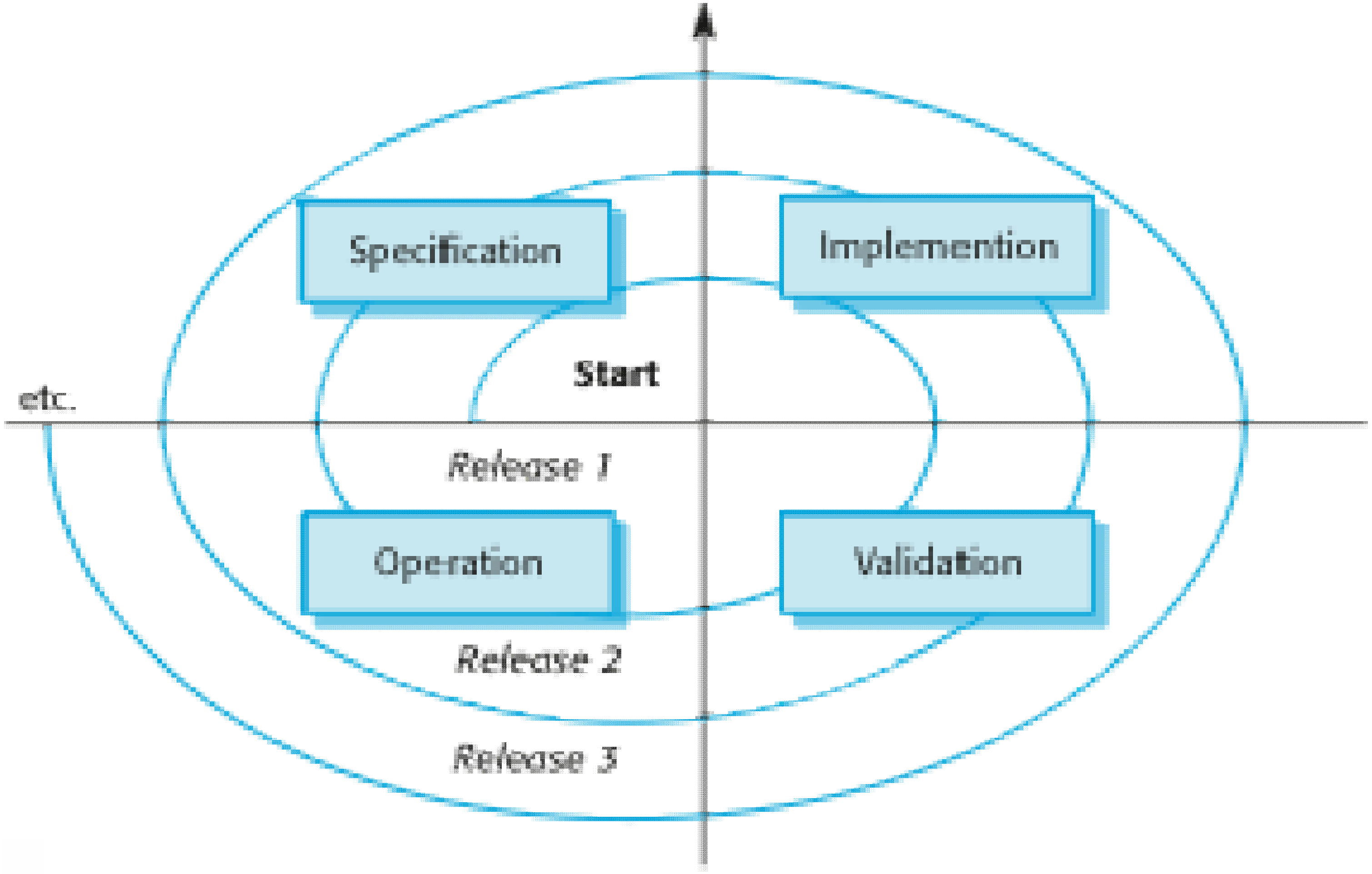
Software change

- 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 all organizations is implementing and managing change to their existing software systems.

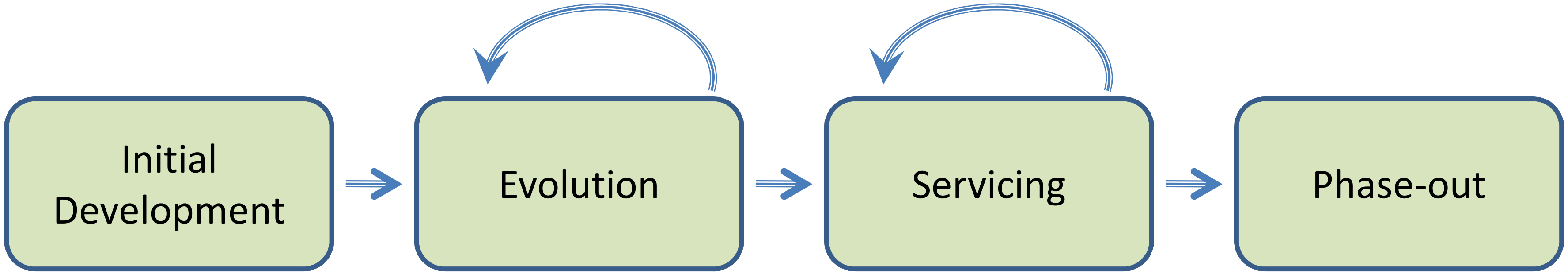
Importance of Evolution

- Organizations 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.

A spiral model of Development and Evolution



Evolution and Servicing



Initial Development

Evolution

Servicing

Phase-out

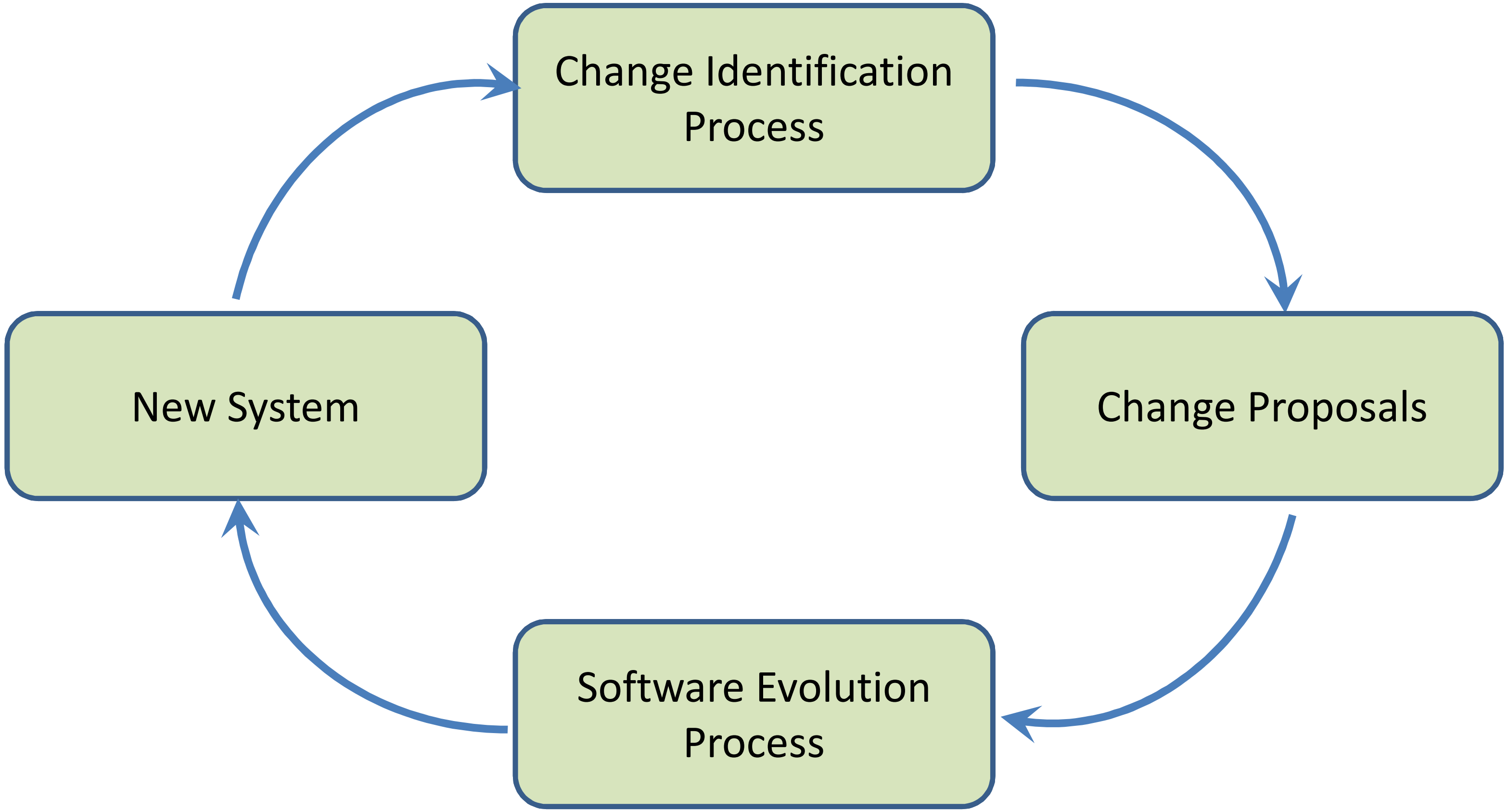
Evolution and Servicing

- **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.

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 and Evolution Processes



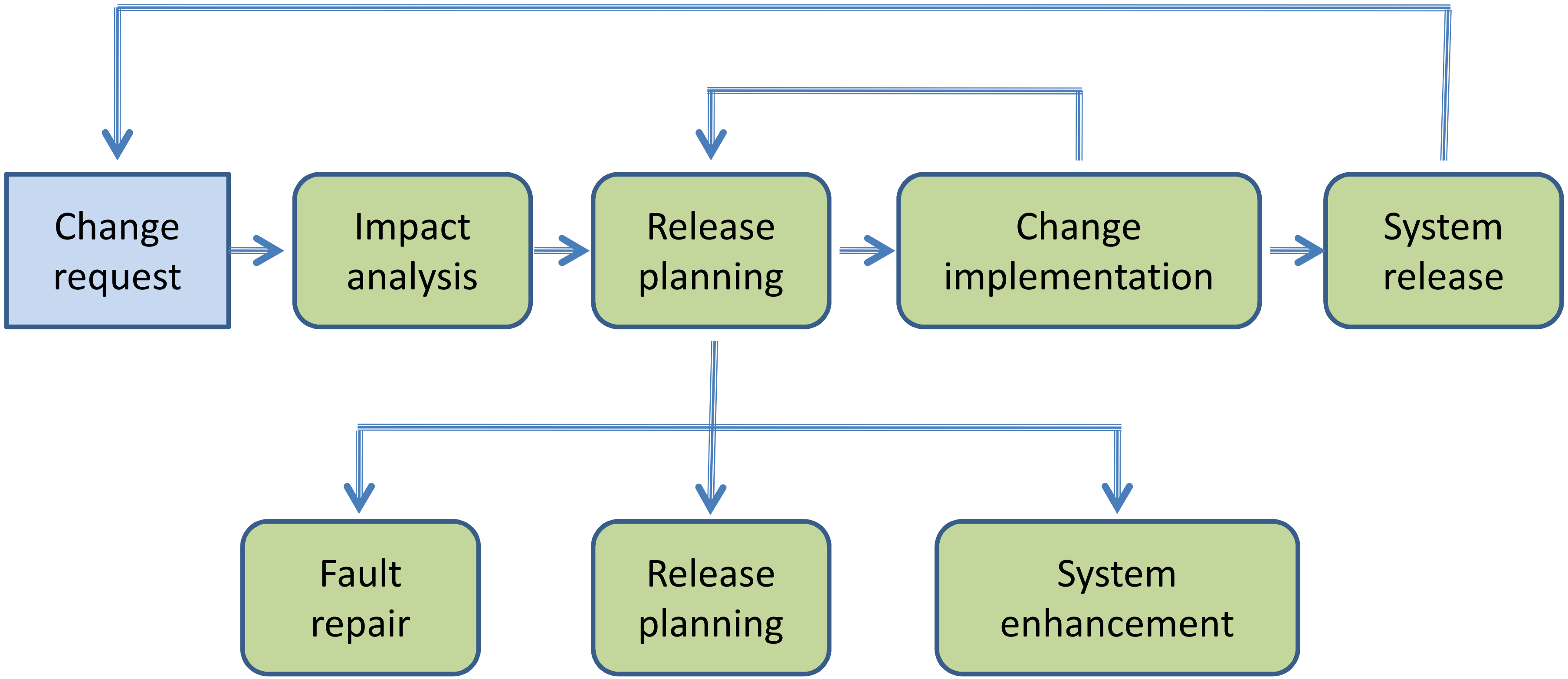
Change Identification Process

Change Proposals

Software Evolution Process

New System

The Software Evolution Process


Change request

Impact analysis

Release planning

Change implementation

System release

Fault repair

Release planning

System enhancement

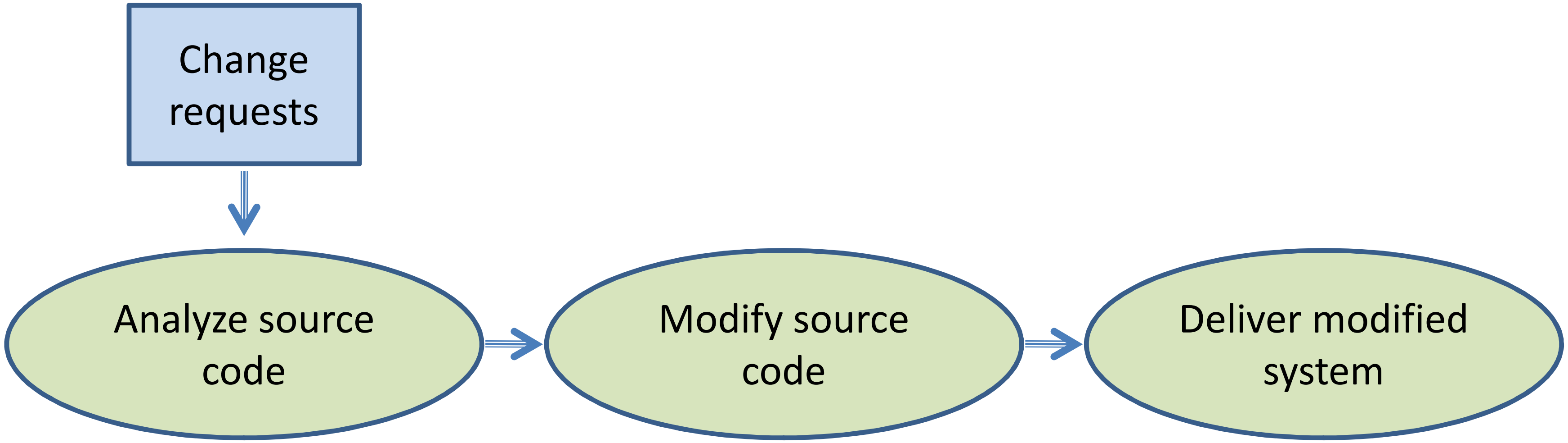
Change Implementation

- 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.
- 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.

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



Change requests

Analyze source code

Modify source code

Deliver modified system

Program evolution dynamics

- *Program evolution dynamics* is the study of the processes of system change.
- After several 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.
    - It is not clear if these are applicable to other types of software system.

Change 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 changed if they are to remain useful in an environment.

Lehman's Laws

Law	Description
Continuing change	A program that is used in a real-world environment must necessarily change, or else 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.
Organizational stability	Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.

Lehman's Laws

Law	Description
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 decline unless they are modified to reflect changes in their operational environment.
Feedback system	Evolution processes incorporate multiagent, multiloop feedback systems and you have to treat them as feedback systems to achieve significant product improvement.

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 on the FEAST project.
- 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.

Reasons for changes

- Errors in the existing system
  - The performance or reliability of the system may have to be improved.
- Changes in requirements
  - New requirements emerge when the software is used.
- Technological advances
  - New computers and equipment is added to the system.
- Legislation and other changes
- The change in the business environment

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.

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

- **Corrective Maintenance**
  - to repair software faults. Changing a system to correct deficiencies in the way meets its requirements.
- **Adaptive Maintenance**
  - Maintenance to add to or modify the system's functionality
  - Modifying the system to satisfy new requirements
  - Modifying the system to suit new operation environment
- **Perfective Maintenance**
  - Modifying the system to satisfy new requirements.
  - Improving programs performance, structure, reliability etc. Making changes to avoid future problems or prepare for future changes

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.).

Reasons for high Maintenance cost

- **Program age and structure**
  - The design and the programming practices used in developing old systems may not be flexible, hence modifying existing systems may be difficult.
- **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 responsible for the maintenance of the system may not understand the design decisions and hence lot of effort during the maintenance process is taken up with understanding the existing system.

Reasons for high Maintenance cost

- **Contractual responsibility**
  - The maintenance contract may be given to a difference company rather than the original system developer. This factor along with the lack of team stability, means that there is no incentive for a development team to write the software so that it is easy to change.
- **Staff skills**
  - Maintenance is seen as a less skilled process than system development and is often allocated to junior staff members. Programming languages and techniques used in old systems may be obsolete, hence new staff members may not like to learn these languages and techniques.

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.

Software Re-engineering

- 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.

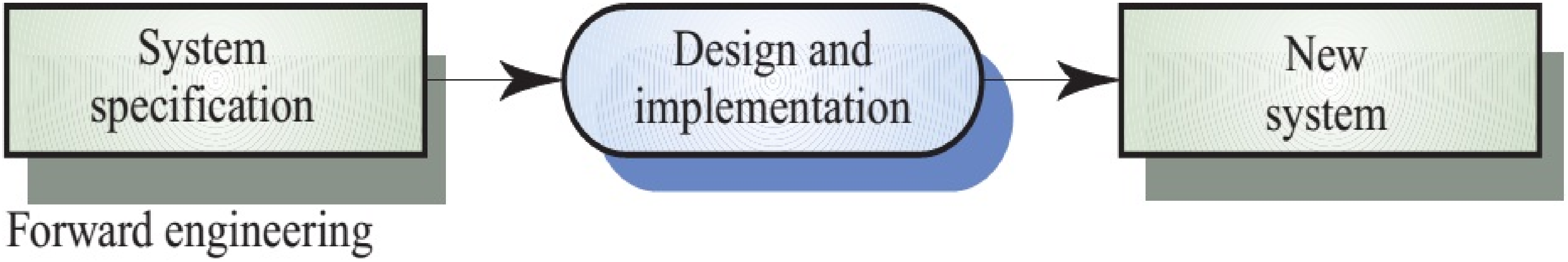
Software Re-engineering

- **What?**
    - Re-structuring or re-writing part or all of an existing system without changing its functionality
- **When?**
    - When some but not all sub-systems of a larger system require frequent maintenance
    - When hardware or software support becomes obsolete
- **How?**
    - The system may be re-structured and re-documented to make it easier to maintain
- **Why?**
    - Reduced risk
        - New software development carries high risk.
    - Reduced cost
        - The cost of re-engineering is often significantly less than the costs of developing new software

Advantages of re-engineering

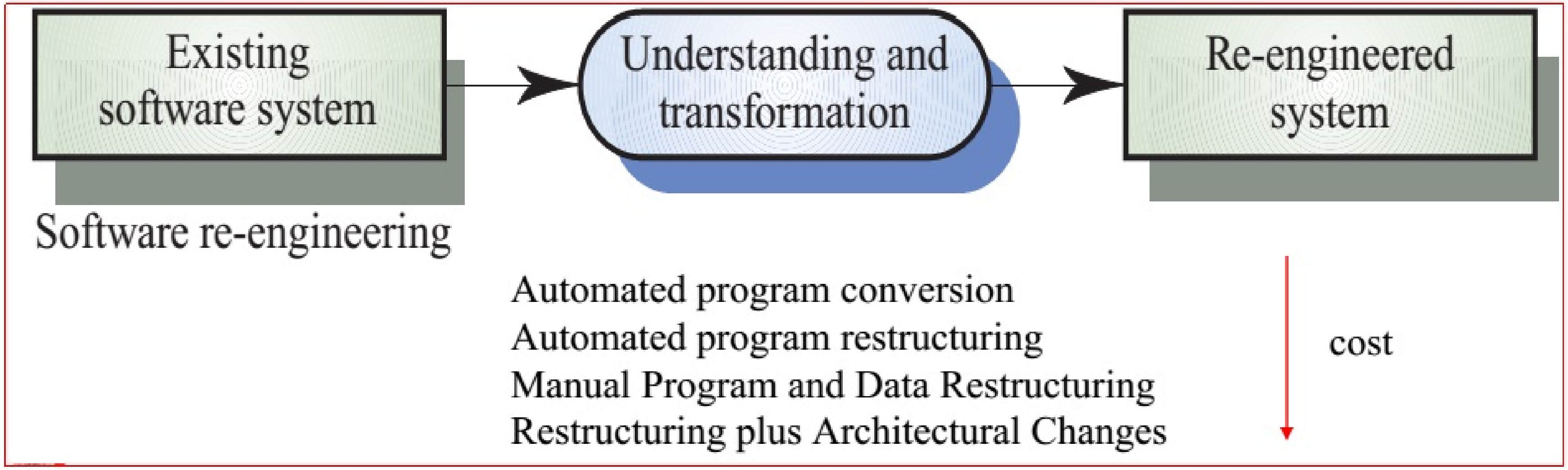
- **Reduced risk**
  - There is a high risk in new software development. There may be development problems, staffing problems and specification problems.
- **Reduced cost**
  - The cost of re-engineering is often significantly less than the costs of developing new software.

Forward engineering and re-engineering



System specification → Design and implementation → New system

Forward engineering



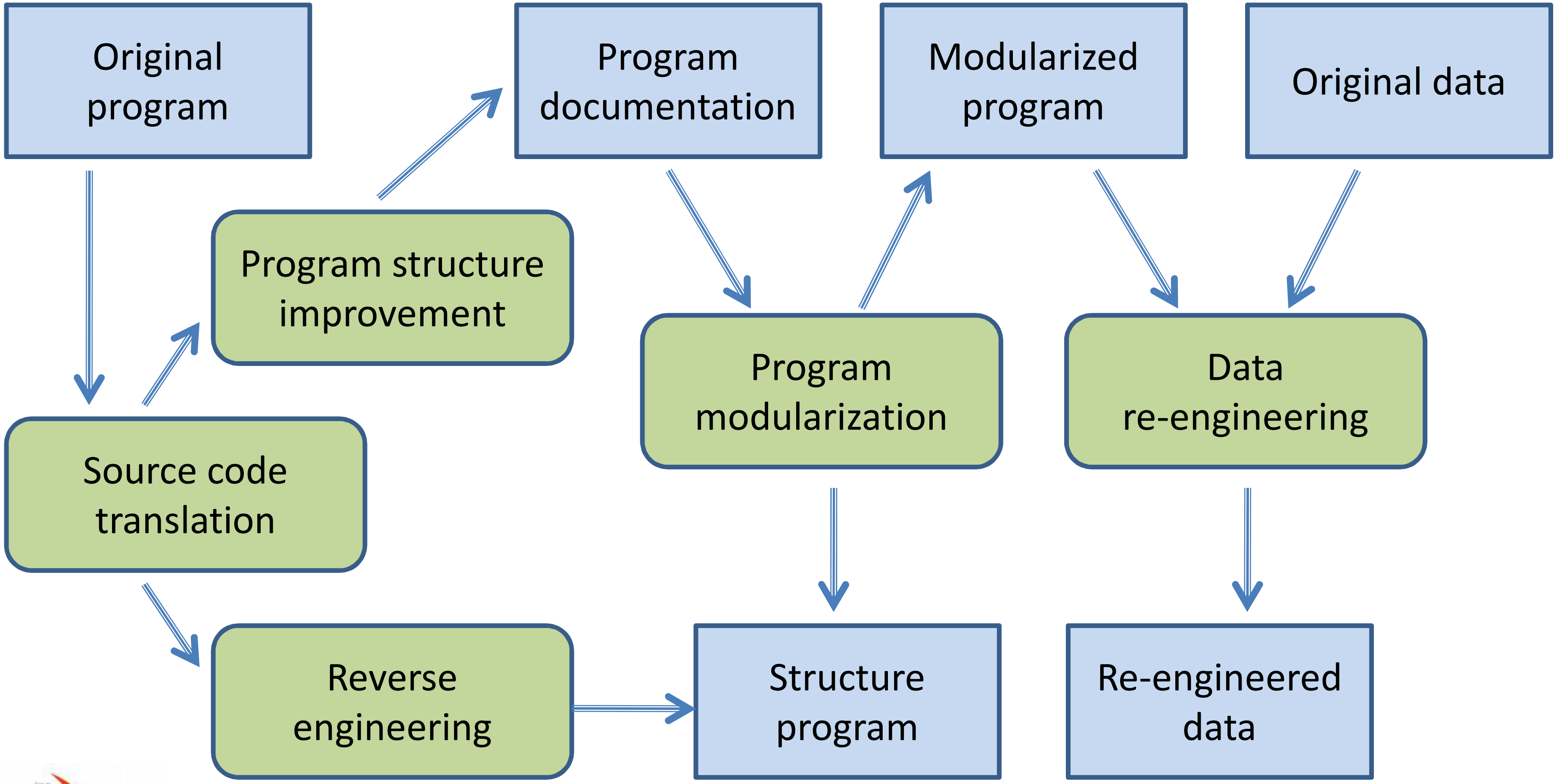
Existing software system → Understanding and transformation → Re-engineered system

Software re-engineering

Automated program conversion
Automated program restructuring
Manual Program and Data Restructuring
Restructuring plus Architectural Changes

cost

The re-engineering process



Re-engineering activities

- **Source code translation**
  - The program is converted from an old programming language to a more modern version of the same language or to different language.
- **Reverse engineering**
  - The program is analyzed and information extracted from it which help to document its organization and functionality.
- **Program structure improvement**
  - The control structure of the program is analyzed and modified to make it easier to read and understand.

Re-engineering activities

- **Program modularization**
  - Related parts of the program are grouped together and, where appropriate, redundancy is removed. In some cases this stage may involve architectural transformation.
- **Data re-engineering**
  - The data processed by the program is changed to reflect program changes.

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.

Preventative maintenance by 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.

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.

7.2 CONFIGURATION MANAGEMENT (CM)

Configuration management

- CM is the development and application of standards and procedures for managing an evolving software product.
- You need to manage evolving systems because, as they evolve, many different versions of the software are created.
- These versions incorporate proposals for change, corrections of faults and adaptations for different hardware and operating systems.
- There may be several versions under development and in use at the same time. You need to keep track of these changes that have been implemented and how these changes have been included in the software.

Configuration Management

- All products of the software process may have to be managed
  - Specifications
  - Designs
  - Programs
  - Test data
  - User manuals
- Thousands of separate documents are generated for a large software system

Configuration Management

- CM Plan
  - A CM plan described the standards and procedures which should be used for configuration management. The starting point for developing the plan should be a set of general, company-wide CM standards and these should be adapted as necessary for each specific project

The contents of a CM plan

1. The definitions of what entities are to be managed and a formal scheme for identifying these entities.
2. A statement of who takes responsibility for the CM procedures and for submitting controlled entities to the CM team.
3. The CM policies that are used for change control and version management.
4. A description of the records of the CM process which should be maintained.
5. A description of the tools to be used for CM and the process to be applied when using these tools.
6. A definition of the configuration database which should used to record configuration information.

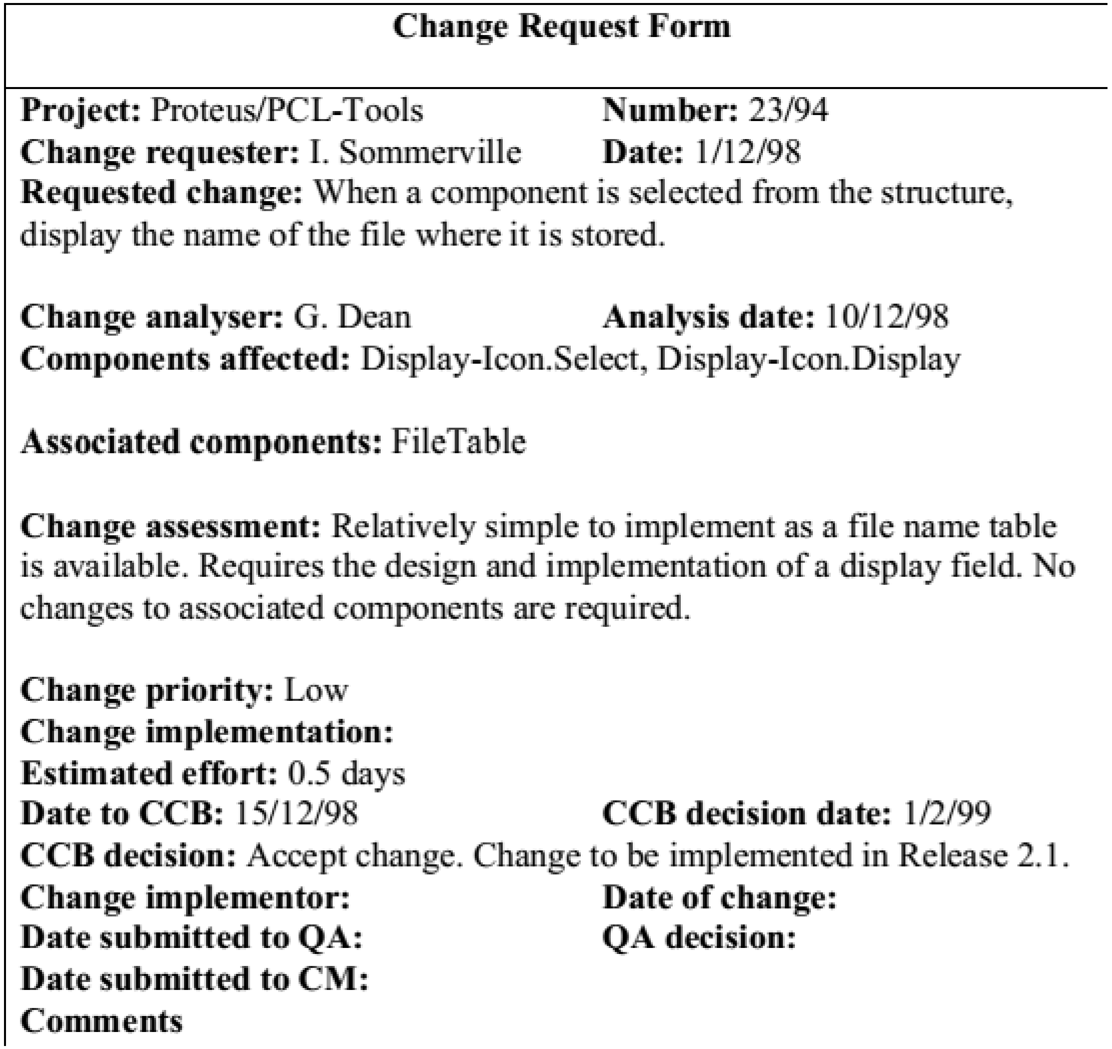
The configuration database

- All CM information should be maintained in a configuration database
- allow queries
  - Who has a particular system version?
  - What platform is required for a particular version?
  - What versions are affected by a change to component X?
  - How many reported faults in version T?
- The CM database should preferably be linked to the software being managed
  - When a programmer downloads a program it is ‘booked’ out to him/her
  - Could be linked to a CASE tool

The change management process

```
Request change by completing a change request form
Analyze change request
If change is valid then
    Assess how change might be implemented
    Assess change cost
    Submit request to change control board
   If change is accepted then
        repeat
                make changes to software
                submit changed software for quality approval
        until   software quality is adequate
        create new system version
   else
        reject change request
else
   reject change request
```

Change request form



Change Request Form

Project: Proteus/PCL-Tools **Number:** 23/94
Change requester: I. Sommerville **Date:** 1/12/98
Requested change: When a component is selected from the structure, display the name of the file where it is stored.

Change analyser: G. Dean **Analysis date:** 10/12/98
Components affected: Display-Icon.Select, Display-Icon.Display

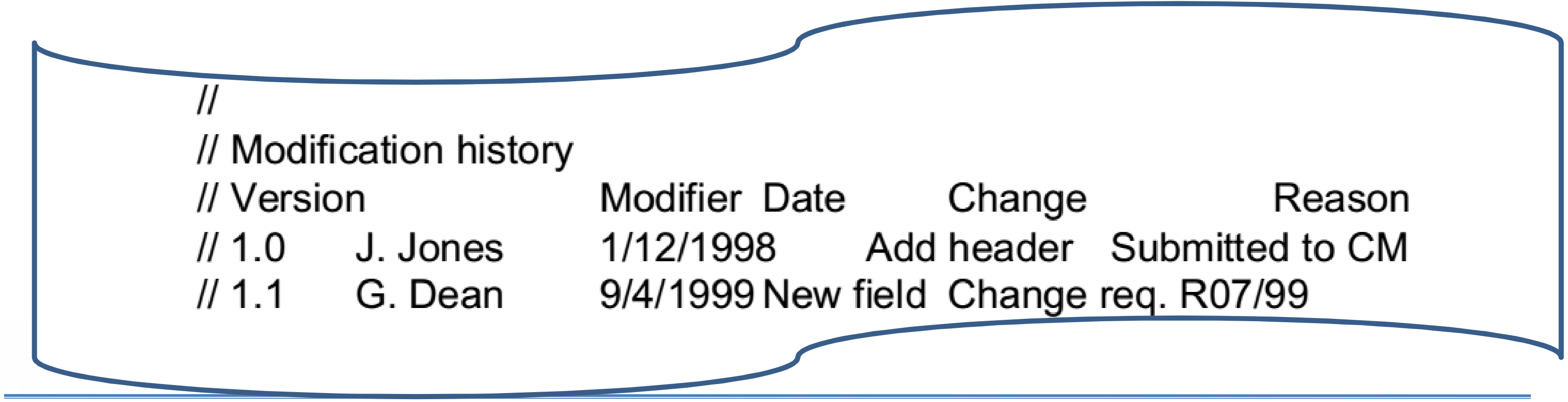
Associated components: FileTable

Change assessment: Relatively simple to implement as a file name table is available. Requires the design and implementation of a display field. No changes to associated components are required.

Change priority: Low
Change implementation:
Estimated effort: 0.5 days
Date to CCB: 15/12/98 **CCB decision date:** 1/2/99
CCB decision: Accept change. Change to be implemented in Release 2.1.
Change implementor: **Date of change:**
Date submitted to QA: **QA decision:**
Date submitted to CM:
Comments

Derivation history

- Record of changes applied to a document or code component
- Should record, in outline, the change made, the rationale for the change, who made the change and when it was implemented
- May be included as a comment in code. If a standard prologue style is used for the derivation history, tools can process this automatically



```
//
// Modification history
// Version                Modifier  Date        Change           Reason
// 1.0      J. Jones      1/12/1998        Add header   Submitted to CM
// 1.1      G. Dean       9/4/1999 New field  Change req. R07/99
```

Versions/variants/releases

- **Version**
  - An instance of a system which is functionally distinct in some way from other system instances
- **Variant**
  - An instance of a system which is functionally identical but non-functionally distinct from other instances of a system
- **Release**
  - An instance of a system which is distributed to users outside of the development team

Versions/variants/releases

- Version Numbering
  - Simple naming scheme uses a linear derivation

    e.g. V1, V1.1, V1.2, V2.1, V2.2 etc.
  - Better way is Attribute Naming
    - Examples of attributes are Date, Creator, Programming Language, Customer, Status etc
    - AC3D (language =Java, platform = NT4, date = Jan 1999)

Summary

- 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.
- Lehman’s laws, such as the notion that change is continuous, describe a number of insights derived from long-term studies of system evolution.

Summary

- There are 3 types of software maintenance, namely Corrective, Adaptive and Perfective maintenance.
- 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.
- Configuration management is essential to maintain the versions of the system.