Software Maintanance

Lesson Outcomes

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

Lehman’s Laws

- Law of Continuing Change
  - A program that is used in a real-world environment necessarily must change or become progressively less useful in that environment
  - I.e. change is inevitable. Even during development, the requirements change!
  - Systems are tightly coupled with their environment. When a system is installed in an environment , changes in the environment may leads to changes in the system.
  - Maintenance is unavoidable
- BUT…Lehman’s Law of 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.

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

Importance of evolution

- A key problem for organisations is implementing and managing change to their existing software systems.
- 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.

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

- 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

Reasons for high Maintenance cost

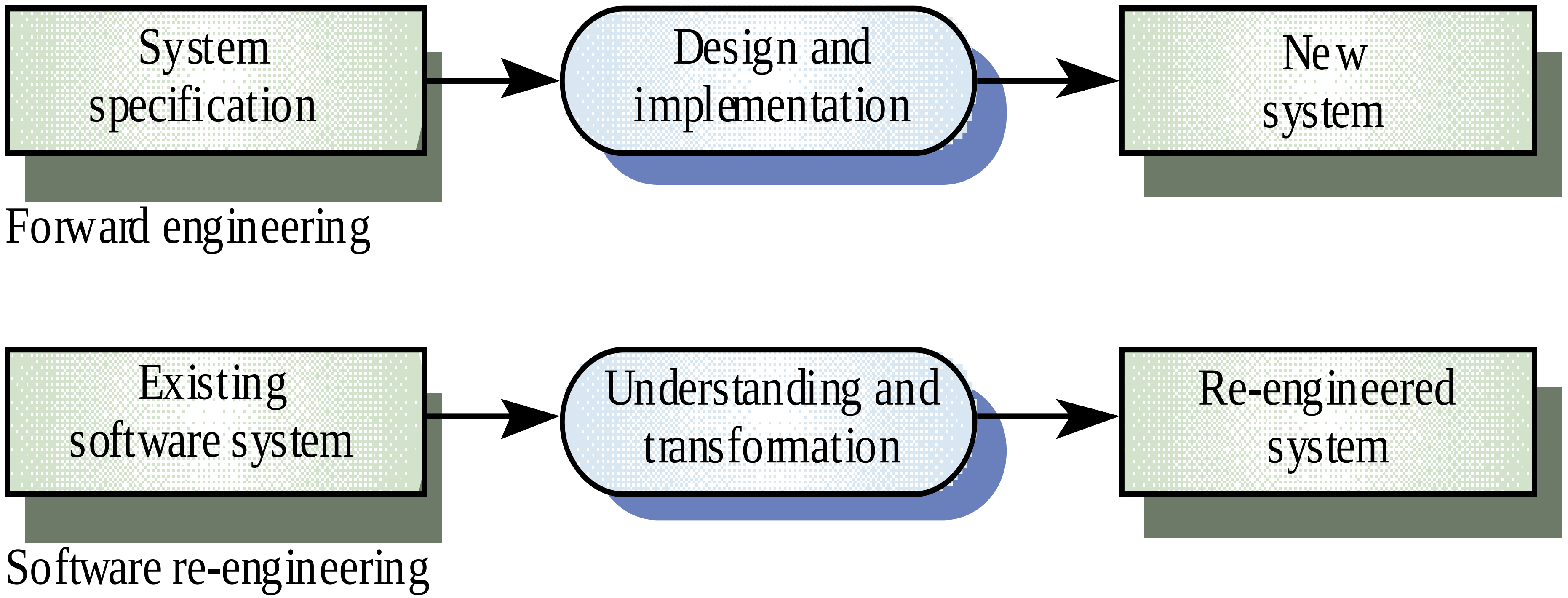
- ***Program age and structure -*** The design and the programming practices used in developing old systems may not be flexible, hence modifying existing systems may be difficult**.**

- ***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.

- ***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.

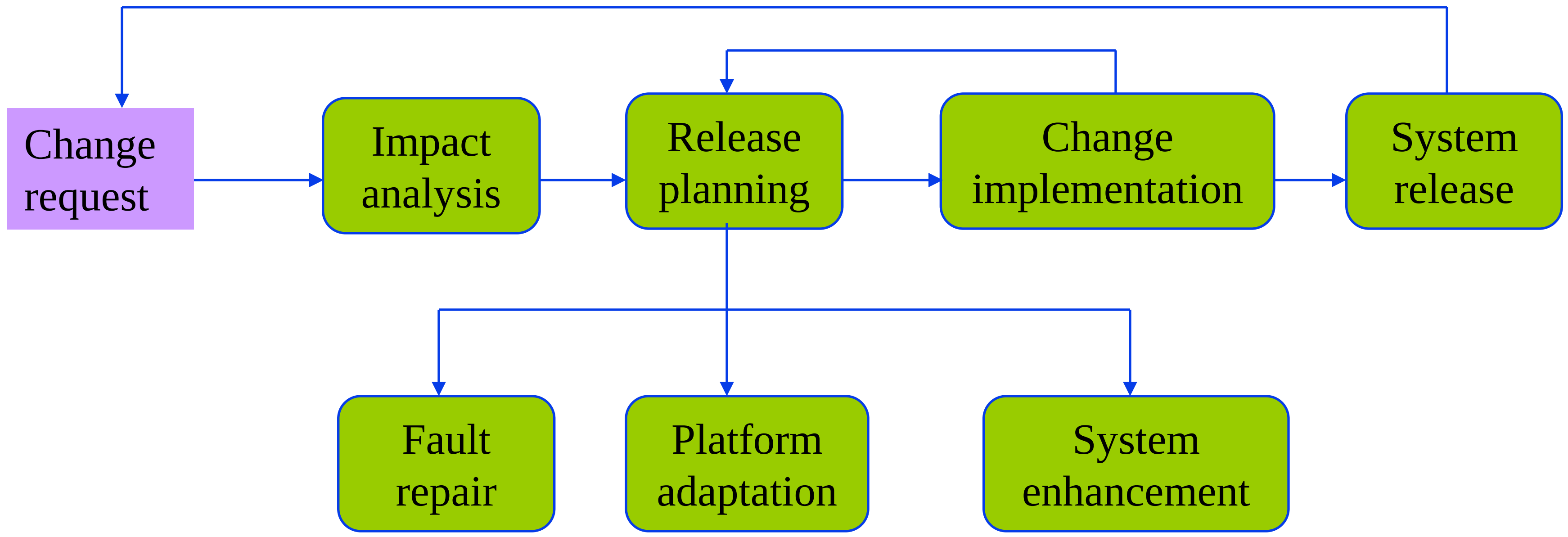
- ***Staff skills -*** Maintenance is seen as a less skilled process than system development and is often allocated to junior staff members. Programming languages and techniques used in old systems may be obsolete, hence new staff members may not like to learn these languages and techniques.

Forward engineering and re-engineering



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

An overview of the maintenance process



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

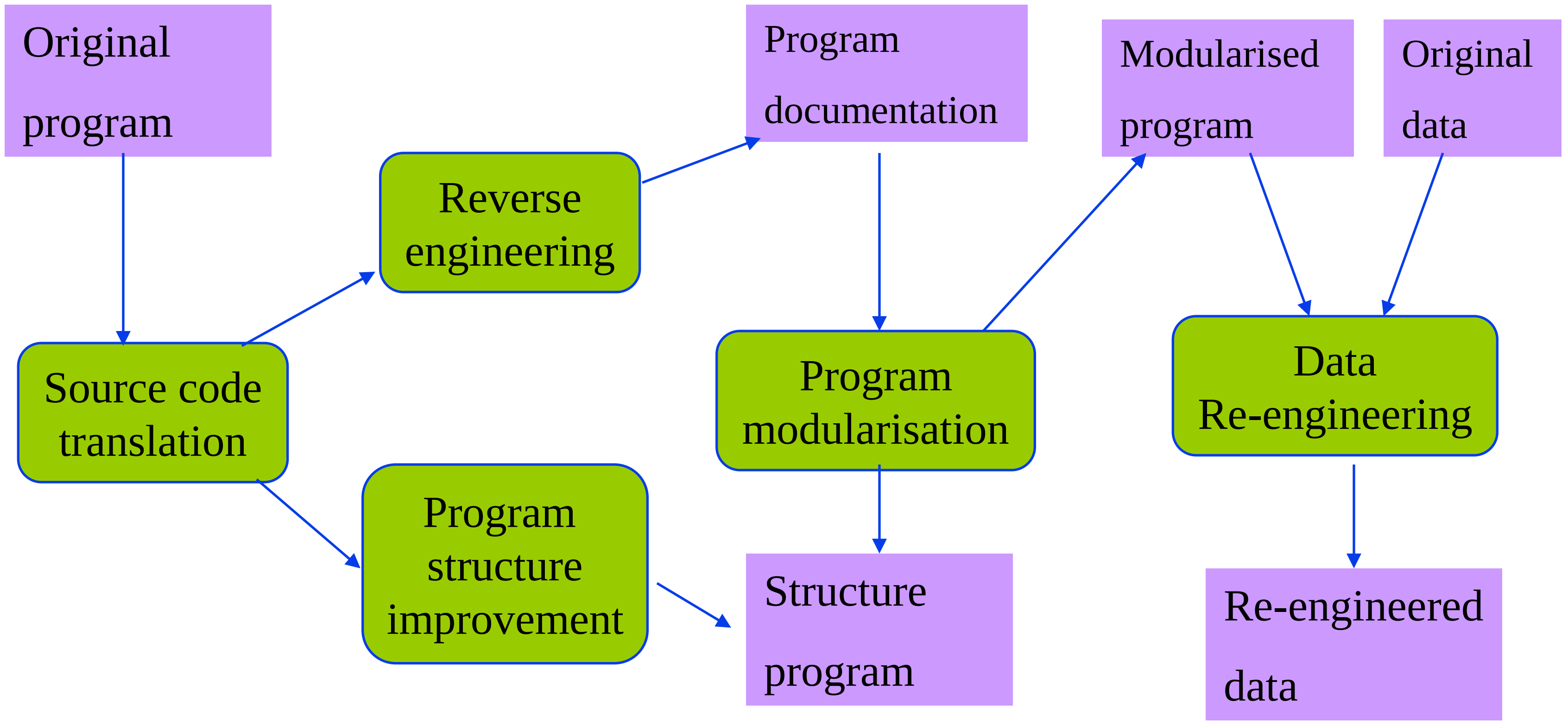
- **Why?**
  - Reduced risk
    - » New software development carries high risk.
  - Reduced cost
    - » The cost of re-engineering is often significantly less than the costs of developing new software

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.

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.

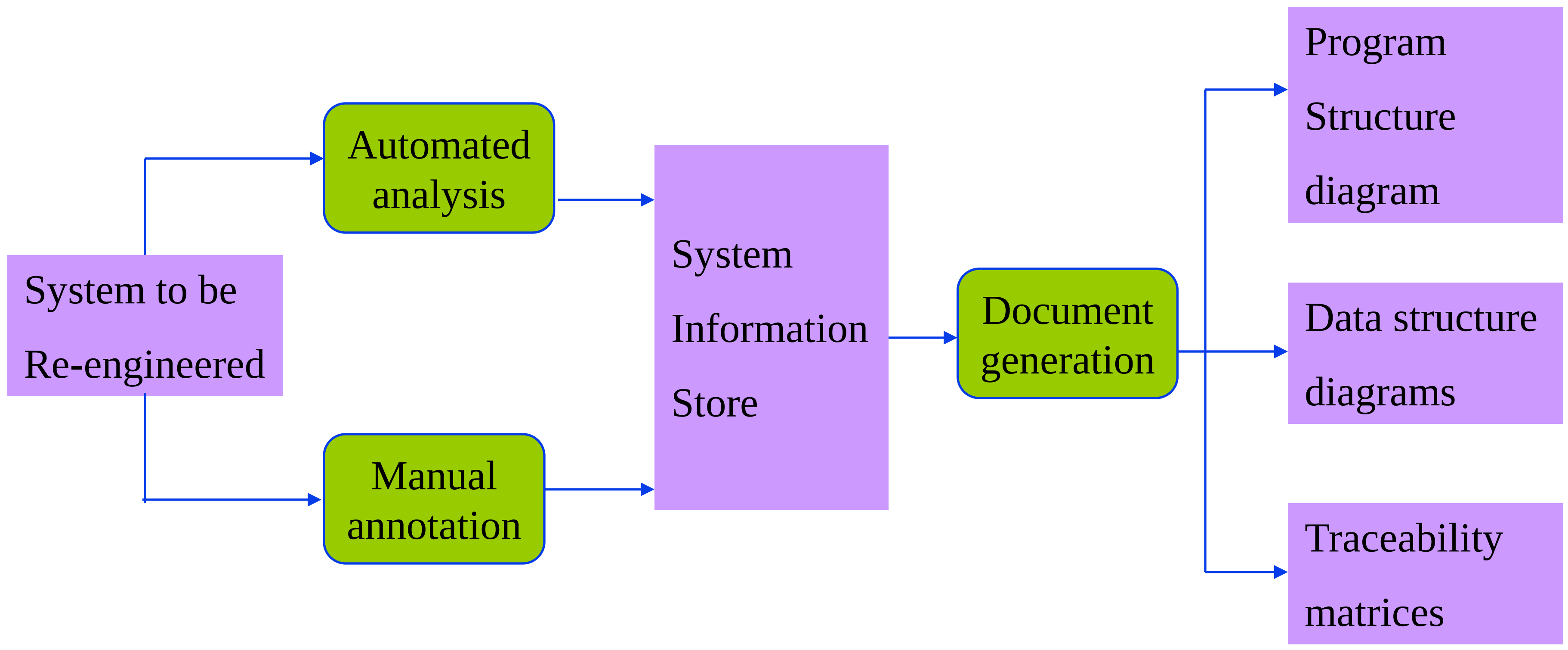
Program modularization – Related parts of the program are grouped together and, where appropriate, redundancy is removed. In some cases this stage may involve architectural transformation.

Data re-engineering – The data processed by the program is changed to reflect program changes.

Reverse engineering

- Analysing software to gain an understanding of its design and specification
- May be part of a re-engineering process **or** to re-specify a system for re-implementation
- Builds a program data base and generates information from this
- Program understanding tools (browsers, CASE tools, cross-reference generators, etc.) may be used in this process
- Why?
  - Original code may have been written under limitations which no longer apply e.g. memory needs, performance etc
  - Maintenance tends to corrupt the structure of a program. It becomes harder and harder to understand
  - The program may be automatically restructured to remove unconditional branches or complex conditional statements

The reverse engineering process



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 contd.

- 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
- 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.

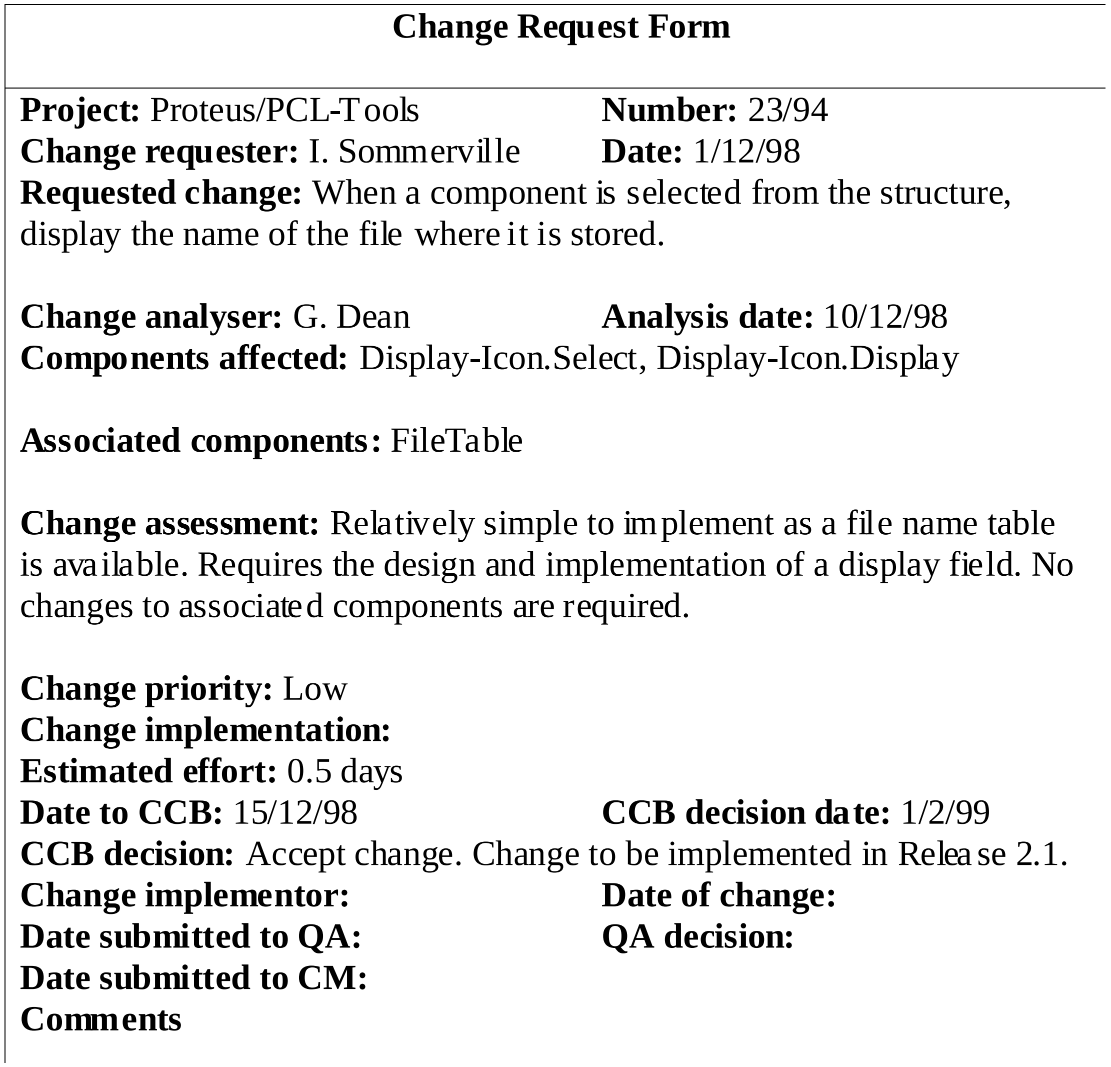
The configuration database

- 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
- 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)