SS ZG622: Software Project Management (Lecture #15)

BITS Pilani
Pilani|Dubai|Goa|Hyderabad

T V Rao, BITS-Pilani Off-campus Centre, Hyderabad

Text Books

T1: Bob Hughes, Mike Cotterel, and Rajib Mall, Software Project Management, 5th Edition, McGraw Hill, 2011

T2: Pressman, R.S. Software Engineering : A Practitioner's Approach, 7th Edition, McGraw Hill, 2010

R1: Sommerville, I., Software Engineering, Pearson Education, 9th Ed., 2010

R2: Capers Jones., Software Engineering Best Practices, TMH ©2010

R3: Robert K. Wysocki, Effective Software Project Management, John Wiley & Sons © 2006

R4: George Stepanek, Software Project Secrets : Why Software Projects Fail, Apress ©2012

R5: A Guide to the Project Management Body of Knowledge (PMBOK® Guide), Fifth Edition by Project Management Institute Project Management Institute © 2013

R6: Jake Kouns and Daniel Minoli, Information Technology Risk Management in Enterprise Environments. John Wiley & Sons © 2010

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

L15: Software Project Management –

Software Evolution, Maintenance, Re-engineering, Change Management

Source Courtesy: Some of the contents of this PPT are sourced from materials provided by publishers of prescribed books

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 changing and evolving existing software rather than developing new software.

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.
  - Other types of software systems, e.g. COTS, small applications, may evolve differently.

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.

Software Evolution

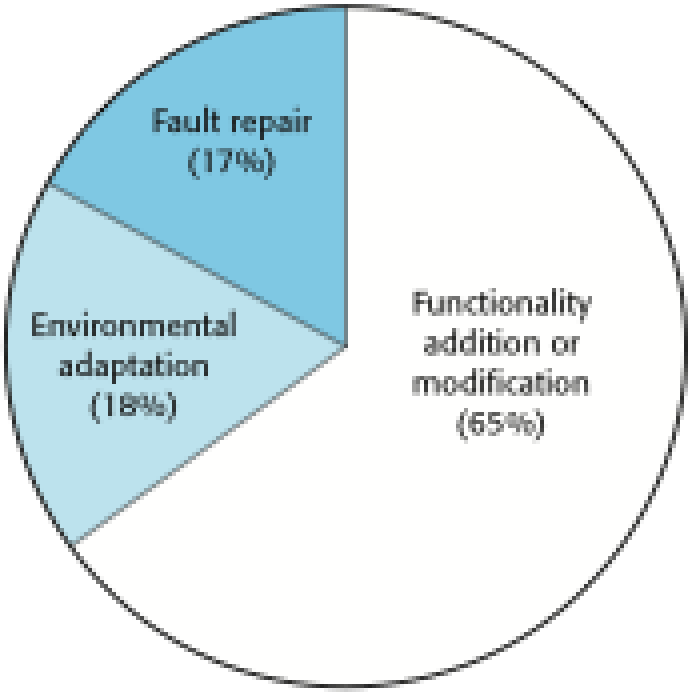
- Software development and evolution can be thought of as an integrated, iterative process that can be represented using a spiral model.
- For custom systems, the costs of software maintenance usually exceed the software development costs.
- The process of software evolution is driven by requests for 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.

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.

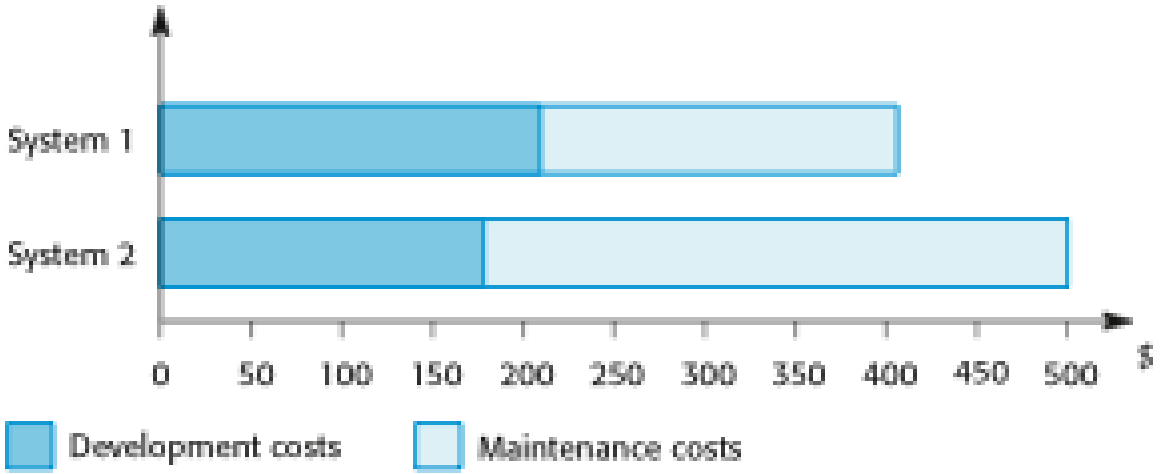
Types of maintenance

- Maintenance to repair software faults
  - Changing a system to correct deficiencies in the way meets its requirements.
- 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.
- Maintenance to add to or modify the system's functionality
  - Modifying the system to satisfy new requirements.



The chart is based on information from several surveys as collated by Sommerville. Some authors categorize maintenance as corrective, adaptive, perfective forms.

Maintenance costs



Good management practices recommend considering lifetime costs rather than development 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.).

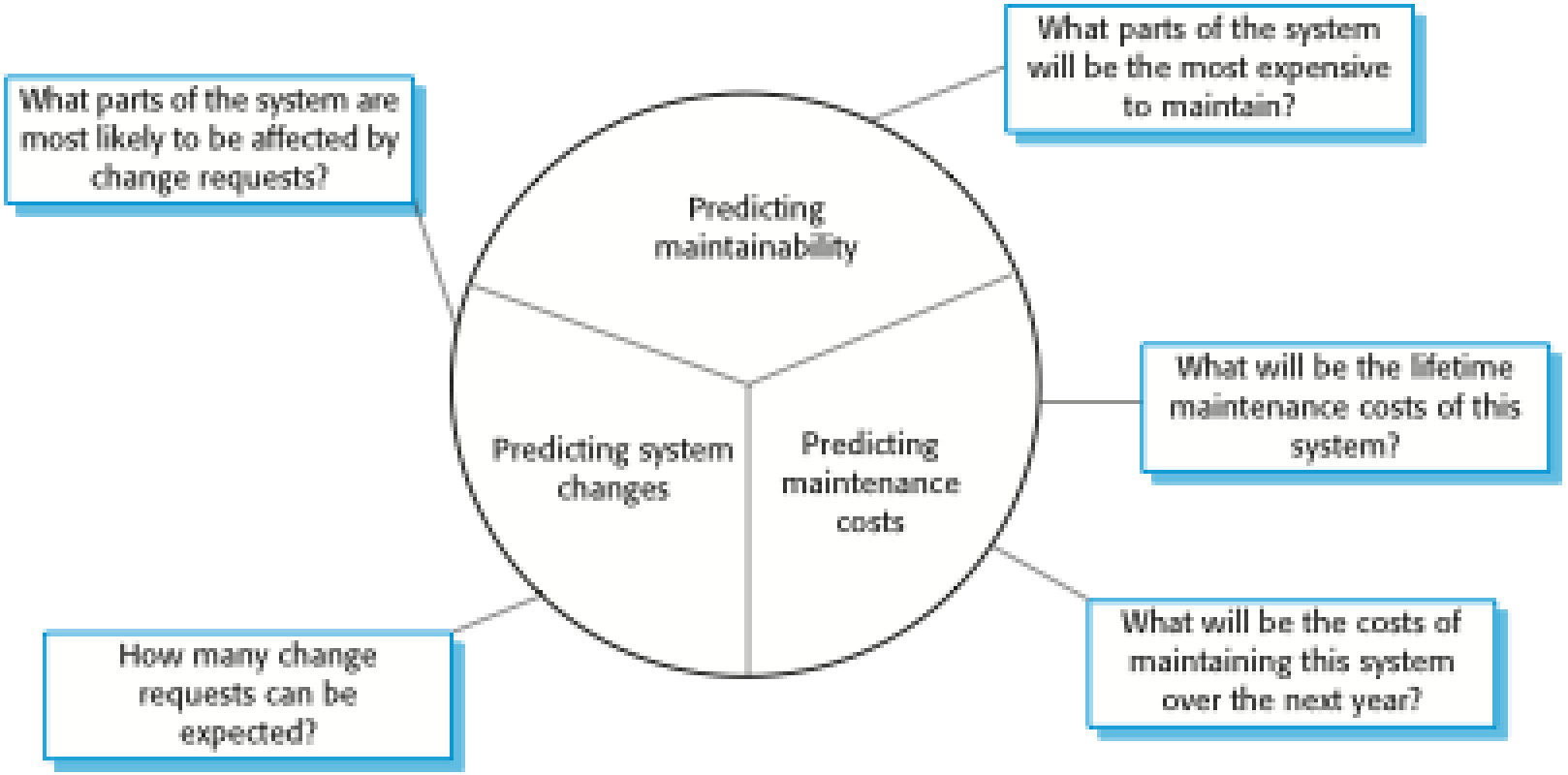
Preventative maintenance by refactoring

- 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.

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



- 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.

Maintainable Software

- Maintainable software exhibits effective modularity
- It makes use of design patterns that allow ease of understanding.
- It has been constructed using well-defined coding standards and conventions, leading to source code that is self-documenting and understandable.
- It has undergone a variety of quality assurance techniques that have uncovered potential maintenance problems before the software is released.
- It has been created by software engineers who recognize that they may not be around when changes must be made.
  - *Therefore, the design and implementation of the software must "assist" the person who is making the change*

Software Supportability

- According to software-supportability.org, supportability of software is

  "the capability of supporting a software system over its whole product life. This implies satisfying any necessary needs or requirements, but also the provision of equipment, support infrastructure, additional software, facilities, manpower, or any other resource required to maintain the software operational and capable of satisfying its function."

- The software should contain facilities to assist support personnel when a defect is encountered in the operational environment (and make no mistake, defects *will* be encountered).
- Support personnel should have access to a database that contains records of all defects that have already been encountered—their characteristics, cause, and cure.

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.

Metrics for Maintainability

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 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.

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.

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.

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?

Business process assessment

- Use a viewpoint-oriented approach and seek answers from system stakeholders
  - Is there a defined process model and is it followed?
  - Do different parts of the organisation use different processes for the same function?
  - How has the process been adapted?
  - What are the relationships with other business processes and are these necessary?
  - Is the process effectively supported by the legacy application software?
- Example - a travel ordering system may have a low business value because of the widespread use of web-based ordering.

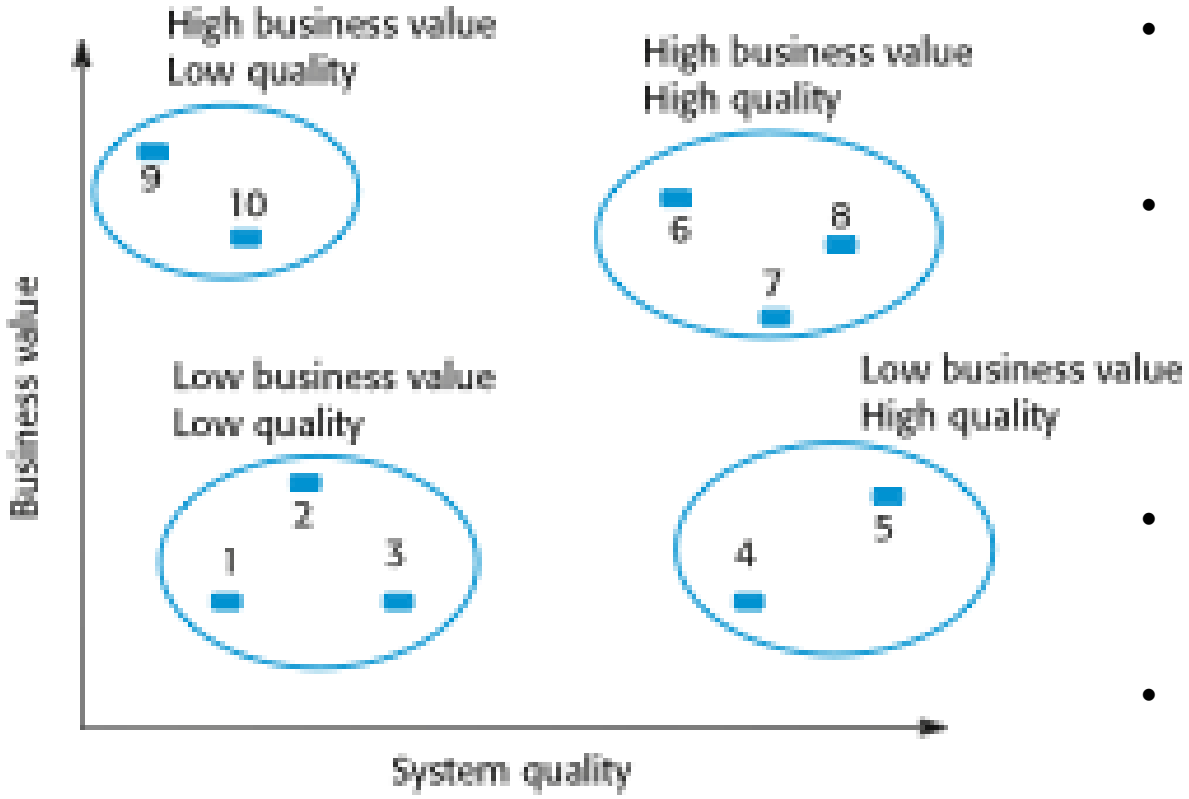
Factors used in environment assessment

Factor	Questions
Supplier stability	Is the supplier 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 a more modern system.
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, for example, be used with current versions of the operating system? Is hardware emulation required?

Factors used in application assessment

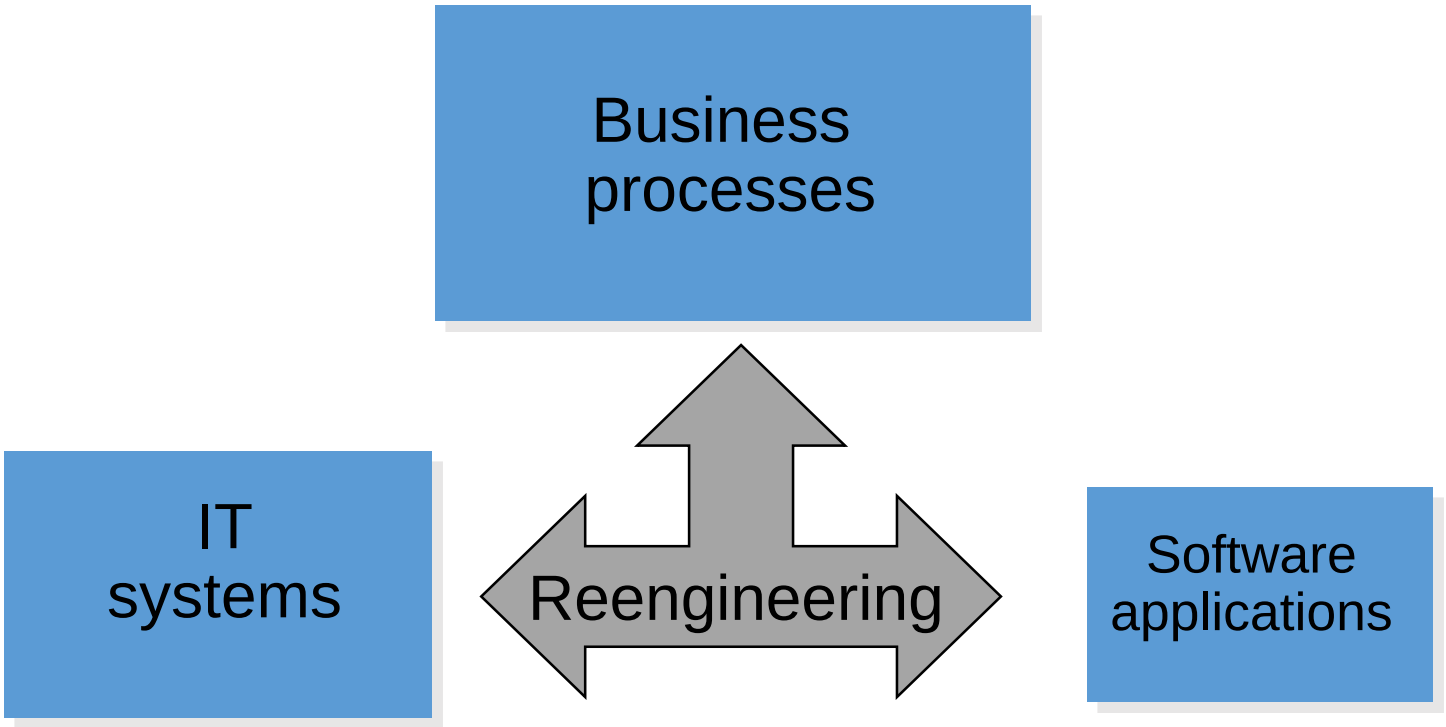
Factor	Questions
Understandability	How difficult is it to understand the source code of the current system? How complex are the control structures that are used? Do variables have meaningful names that reflect their function?
Documentation	What system documentation is available? Is the documentation complete, consistent, and current?
Data	Is there an explicit data model for the system? To what extent is data duplicated across 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 people available who have experience with the system?

An example legacy system assessment



- 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.

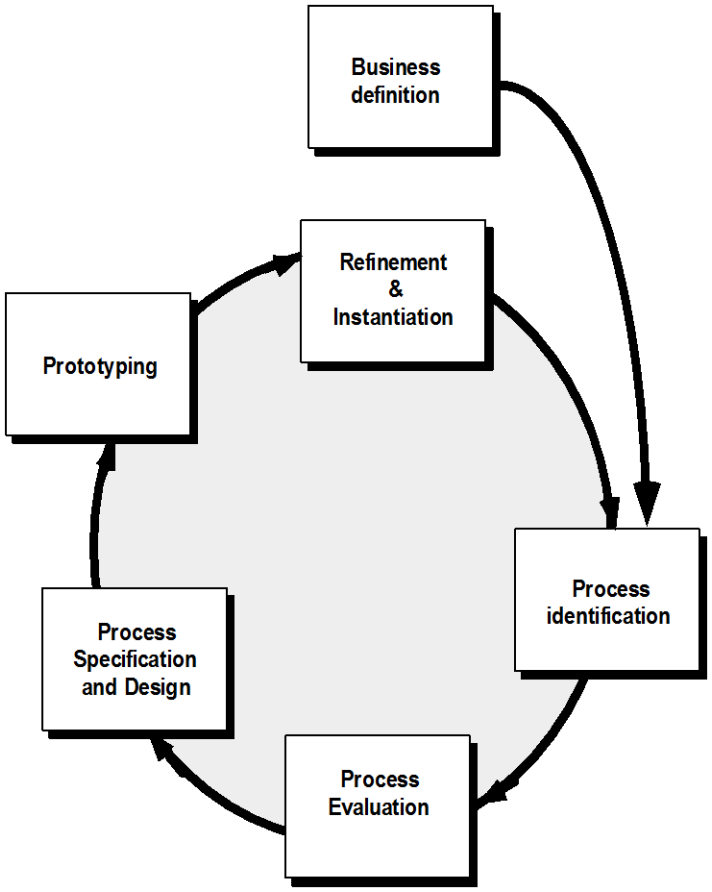
Reengineering



Business Process Reengineering

- **Business definition.** Business goals are identified within the context of four key drivers: cost reduction, time reduction, quality improvement, and personnel development and empowerment.
- **Process identification.** Processes that are critical to achieving the goals defined in the business definition are identified.
- **Process evaluation.** The existing process is thoroughly analyzed and measured.
- **Process specification and design.** Based on information obtained during the first three BPR activities, use-cases are prepared for each process that is to be redesigned.
- **Prototyping.** A redesigned business process must be prototyped before it is fully integrated into the business.
- **Refinement and instantiation.** Based on feedback from the prototype, the business process is refined and then instantiated within a business system.

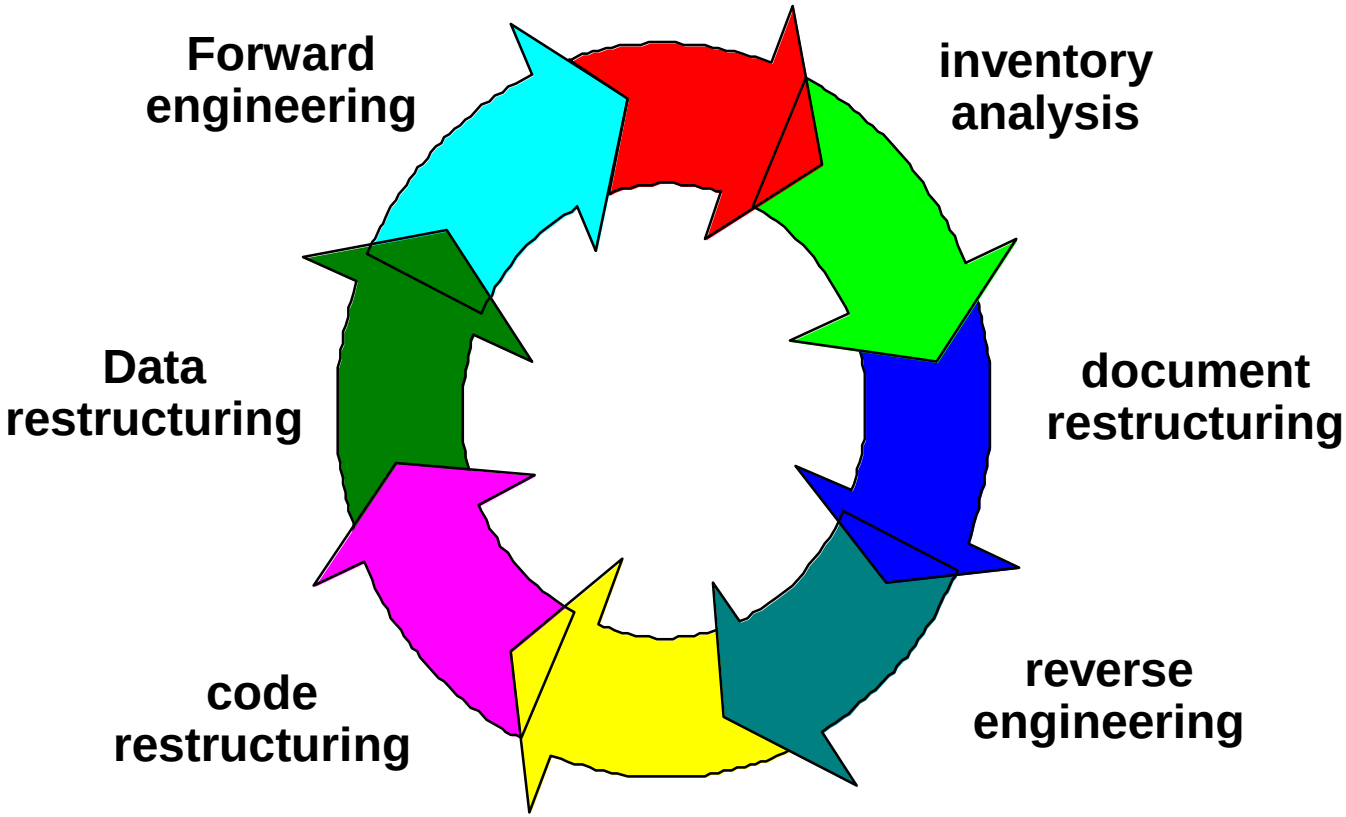
Business Process Reengineering



BPR Principles

- Organize around outcomes, not tasks.
- Have those who use the output of the process perform the process.
- Incorporate information processing work into the real work that produces the raw information.
- Treat geographically dispersed resources as though they were centralized.
- Link parallel activities instead of integrated their results. When different
- Put the decision point where the work is performed, and build control into the process.
- Capture data once, at its source.

Software Reengineering



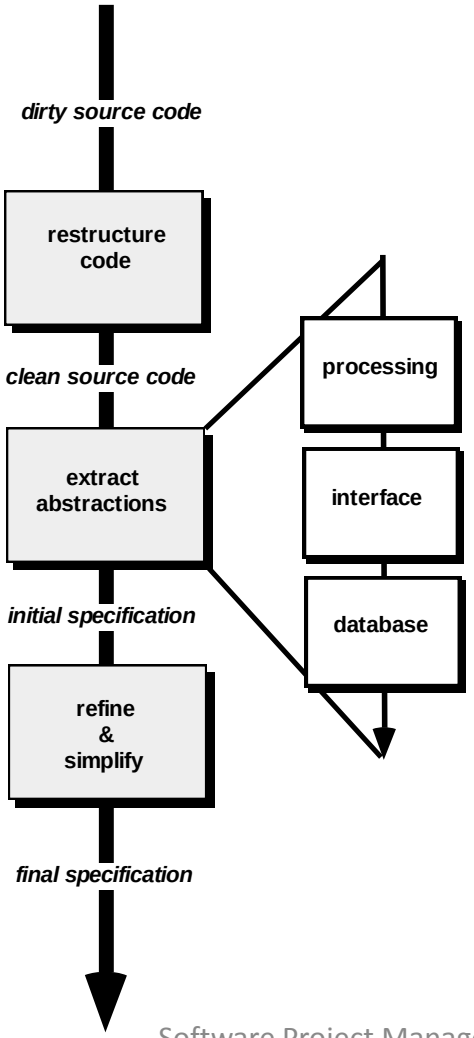
Inventory Analysis

- build a table that contains all applications
- establish a list of criteria, e.g.,
  - name of the application
  - year it was originally created
  - number of substantive changes made to it
  - total effort applied to make these changes
  - date of last substantive change
  - effort applied to make the last change
  - system(s) in which it resides
  - applications to which it interfaces, ...
- analyze and prioritize to select candidates for reengineering

Document Restructuring

- Weak documentation is the trademark of many legacy systems.
- But what do we do about it? What are our options?
- Options …
  - *Creating documentation is far too time consuming.* If the system works, we'll live with what we have. In some cases, this is the correct approach.
  - *Documentation must be updated, but we have limited resources.* We'll use a "document when touched" approach. It may not be necessary to fully redocument an application.
  - *The system is business critical and must be fully redocumented.* Even in this case, an intelligent approach is to pare documentation to an essential minimum.

Reverse Engineering



Code Restructuring

- Source code is analyzed using a restructuring tool.
- Poorly design code segments are redesigned
- Violations of structured programming constructs are noted and code is then restructured (this can be done automatically)
- The resultant restructured code is reviewed and tested to ensure that no anomalies have been introduced
- Internal code documentation is updated.

Data Restructuring

- Unlike code restructuring, which occurs at a relatively low level of abstraction, data structuring is a full-scale reengineering activity
- In most cases, data restructuring begins with a reverse engineering activity.
  - Current data architecture is dissected and necessary data models are defined.
  - Data objects and attributes are identified, and existing data structures are reviewed for quality.
  - When data structure is weak (e.g., flat files are currently implemented, when a relational approach would greatly simplify processing), the data are reengineered.
- Because data architecture has a strong influence on program architecture and the algorithms that populate it, changes to the data will invariably result in either architectural or code-level changes.

Forward Engineering

- Forward Engineering re-creates an existing application using sound principles
- It may not just create a modern equivalent to an old program. It more than that
- Generally forward engineering is integrated into reengineering effort.
- Redeveloped program extends capabilities of older program.

Forward Engineering

Why Forward Engineering may be feasible for an existing program?

1. The cost to maintain one line of source code may be several times the cost of initial development of that line.

2. Redesign of the software architecture (program and/or data structure), using modern design concepts, can greatly facilitate future maintenance.

3. Because a prototype of the software already exists, development productivity should be much higher than average.

4. The user now has experience with the software. Therefore, new requirements and the direction of change can be ascertained with greater ease.

5. CASE tools for reengineering will automate some parts of the job.

6. A complete software configuration (documents, programs and data) will exist upon completion of preventive maintenance.

Economics of Reengineering-I

- A cost/benefit analysis model for reengineering has been proposed by *Sneed*. Nine parameters are defined:
  - P_1 = current annual maintenance cost for an application.
  - P_2 = current annual operation cost for an application.
  - P_3 = current annual business value of an application.
  - P_4 = predicted annual maintenance cost after reengineering.
  - P_5 = predicted annual operations cost after reengineering.
  - P_6 = predicted annual business value after reengineering.
  - P_7 = estimated reengineering costs.
  - P_8 = estimated reengineering calendar time.
  - P_9 = reengineering risk factor (P_9 = 1.0 is nominal).
  - L = expected life of the system.

Economics of Reengineering-II

- The cost associated with continuing maintenance of a candidate application (i.e., reengineering is not performed) can be defined as

$$C_{maint} = [P_3 - (P_1 + P_2)] \times L$$

- The costs associated with reengineering are defined using the following relationship:

$$C_{reeng} = [P_6 - (P_4 + P_5) \times (L - P_8) - (P_7 \times P_9)]$$

- Using the costs presented in equations above, the overall benefit of reengineering can be computed as

$$\text{cost benefit} = C_{reeng} - C_{maint}$$

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Software Project Management –

Software Configuration Management, Tools, Build, Releases

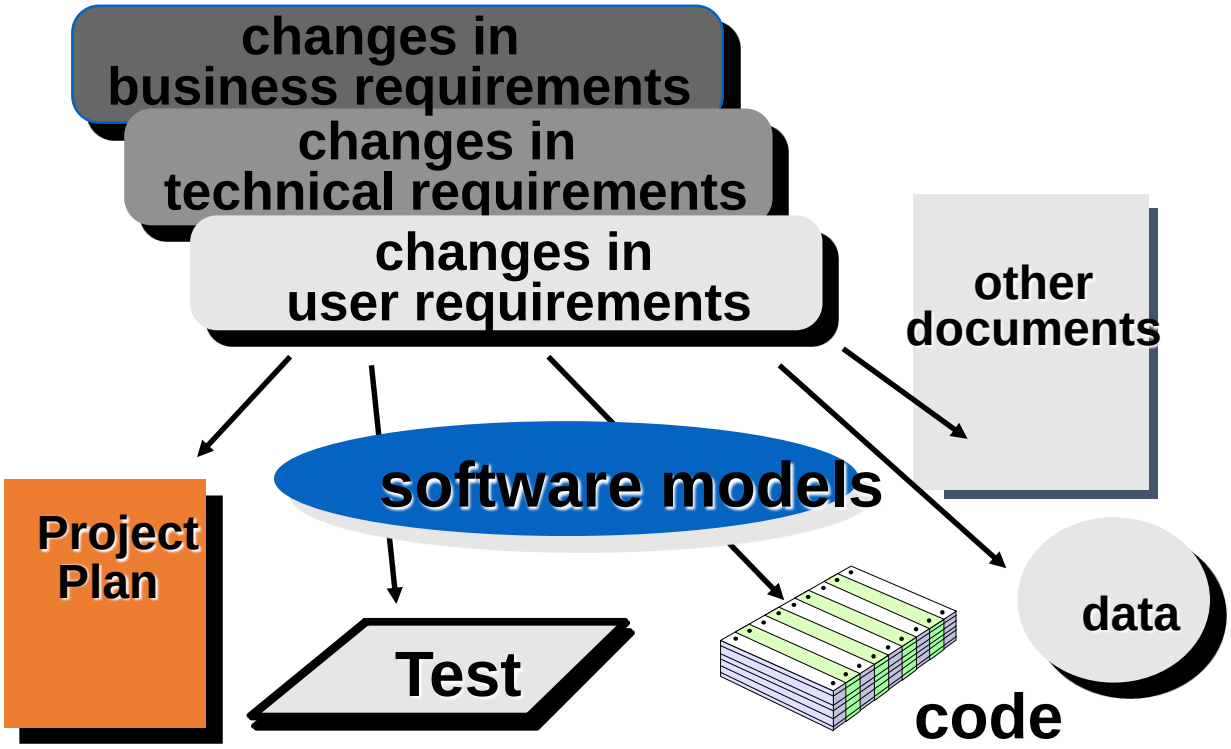
Source Courtesy: Some of the contents of this PPT are sourced from materials provided by publishers of prescribed books

The “First Law”

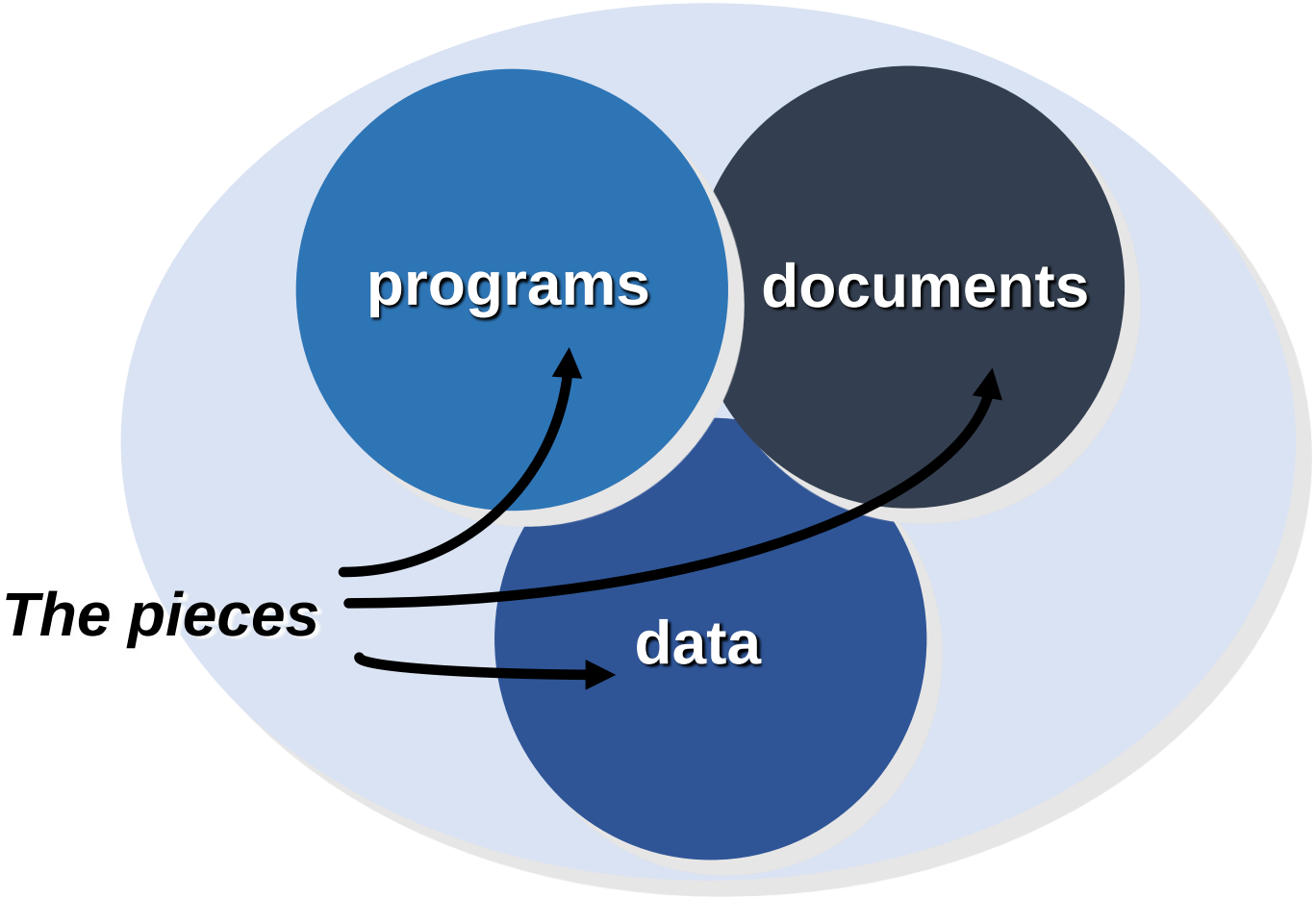
No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle.

Bersoff, et al, 1980

What Are These Changes?



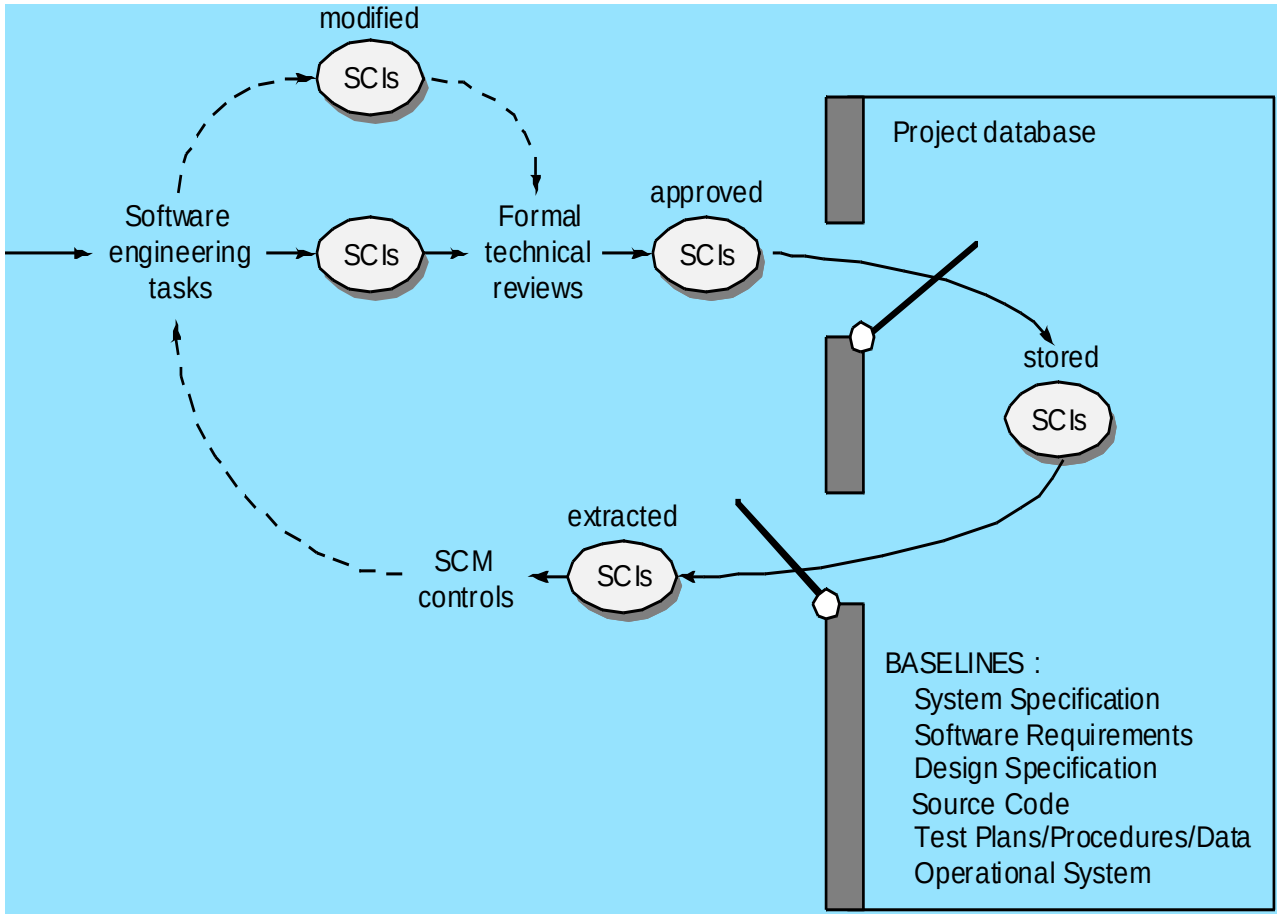
The Software Configuration



SCM Major Areas

Area	Description
Configuration Identification	This requires identifying the configuration items to be controlled, and implementing a sound configuration management system, including a repository where documents and source code are placed under controlled access. It includes a mechanism for releasing documents or code, a naming convention and version numbering system for documents and code, and baseline/release planning. The version and status of each configuration item should be known.
Configuration Control	This involves tracking and controlling change requests, and controlling changes to the configuration items. Any changes to the work products are controlled, and authorized by a change control board or similar mechanism. Problems or defects reported by the test groups or customer are analysed, and any changes made are subject to change control. The version of the work product is known, and the constituents of a particular release are known and controlled. The previous versions of releases can be recreated as the source code constituents are fully known.
Configuration Auditing	This includes audits of the baselines to verify integrity of the baseline and audits of the configuration management system itself and verification that standards and procedures are followed. The results of the audits are communicated to the affected groups and corrective action taken.
Status Accounting	This involves data collection and report generation. These reports include the software baseline status, the summary of changes to the software baseline, problem report summaries, and change request summaries

Baselines

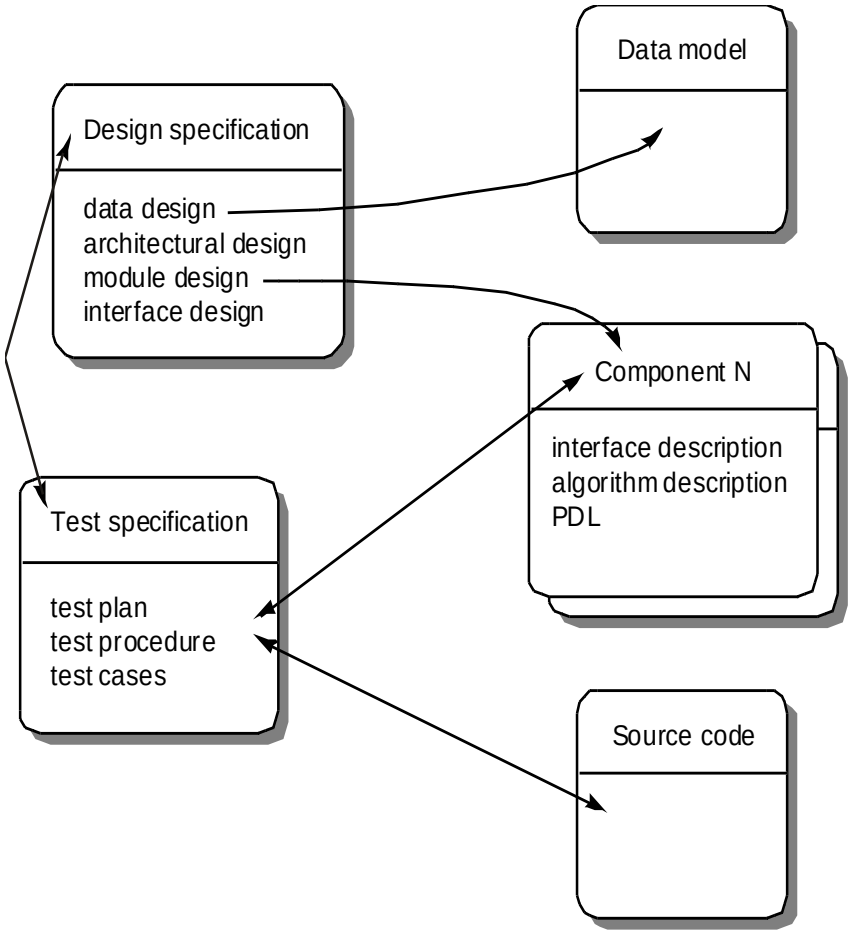


The IEEE (IEEE Std. No. 610.12-1990) defines a baseline as:

A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.

A baseline is a milestone in the development of software that is marked by the delivery of one or more software configuration items and the approval of these SCIs that is obtained through a formal technical review

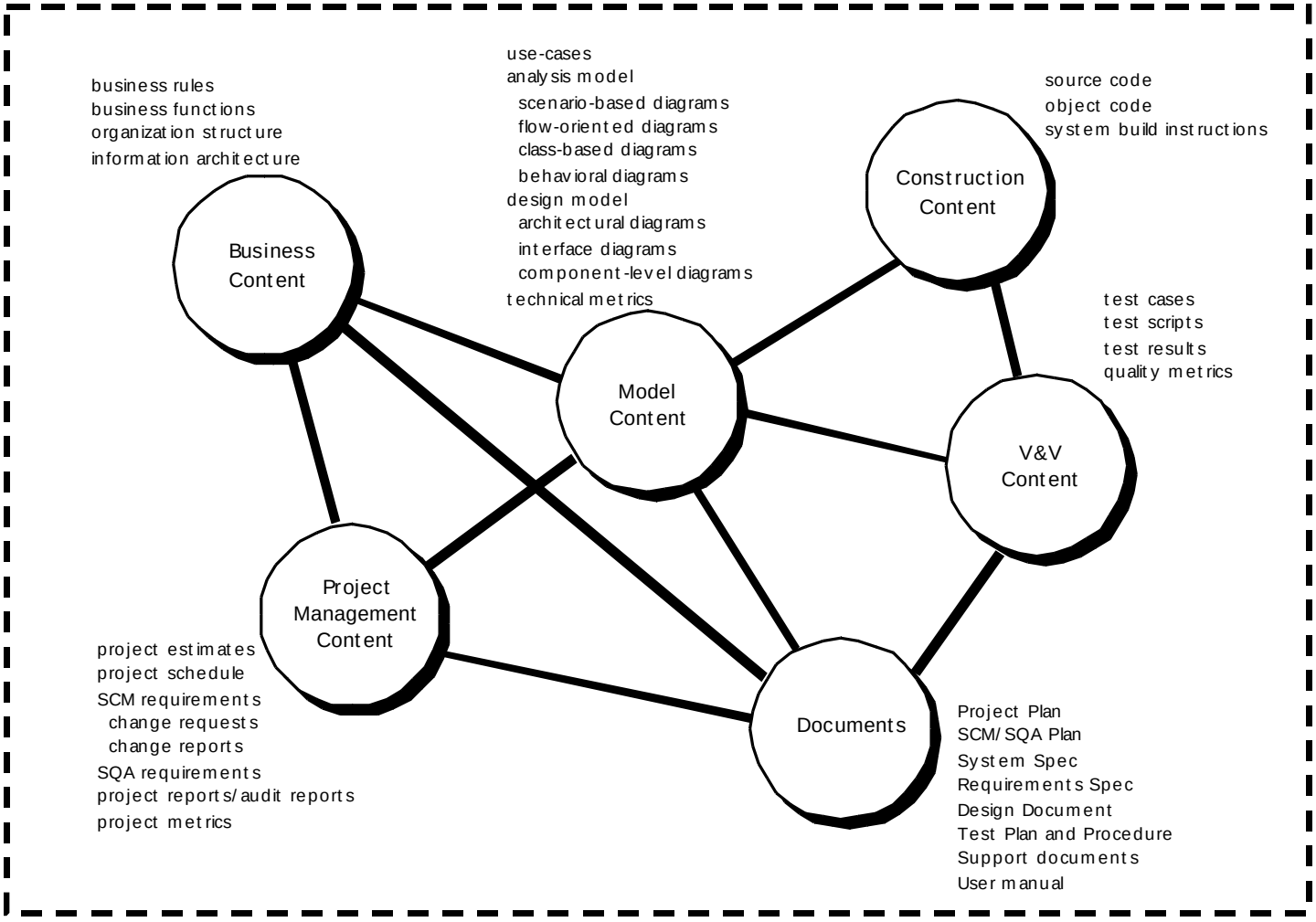
Software Configuration Objects



SCM Repository

- The SCM repository is the set of mechanisms and data structures that allow a software team to manage change in an effective manner
- The repository performs or precipitates the following functions [For89]:
  - Data integrity
  - Information sharing
  - Tool integration
  - Data integration
  - Methodology enforcement
  - Document standardization

Repository Content



Repository Features

- **Versioning.**
  - saves all of these versions to enable effective management of product releases and to permit developers to go back to previous versions
- **Dependency tracking and change management.**
  - The repository manages a wide variety of relationships among the data elements stored in it.
- **Requirements tracing.**
  - Provides the ability to track all the design and construction components and deliverables that result from a specific requirement specification
- **Configuration management.**
  - Keeps track of a series of configurations representing specific project milestones or production releases. Version management provides the needed versions, and link management keeps track of interdependencies.
- **Audit trails.**
  - establishes additional information about when, why, and by whom changes are made.

SCM Elements

According to Susan Dart, a configuration management system has four elements

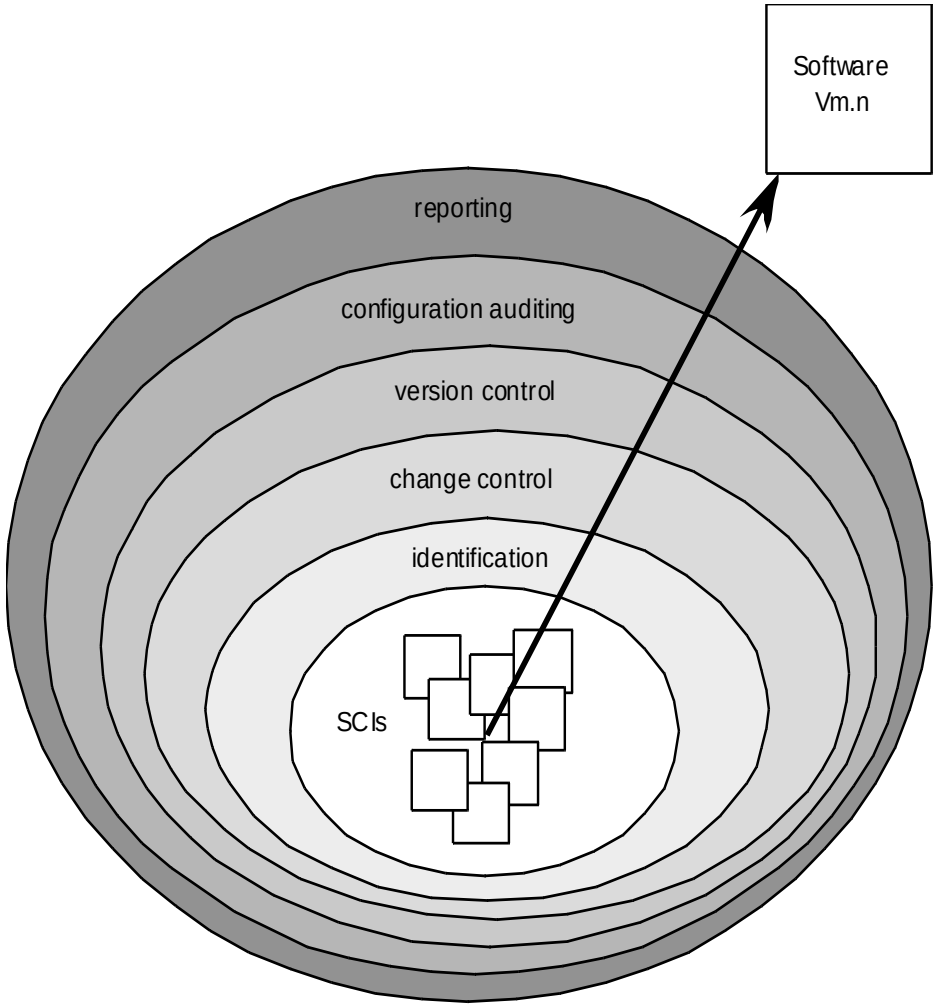
- *Component elements*—a set of tools coupled within a file management system (e.g., a database) that enables access to and management of each software configuration item.
- *Process elements*—a collection of procedures and tasks that define an effective approach to change management (and related activities) for all constituencies involved in the management, engineering and use of computer software.
- *Construction elements*—a set of tools that automate the construction of software by ensuring that the proper set of validated components (i.e., the correct version) have been assembled.
- *Human elements*—to implement effective SCM, the software team uses a set of tools and process features (encompassing other CM elements)

The SCM Process

Addresses the following questions …

- How does a software team identify the discrete elements of a software configuration?
- How does an organization manage the many existing versions of a program (and its documentation) in a manner that will enable change to be accommodated efficiently?
- How does an organization control changes before and after software is released to a customer?
- Who has responsibility for approving and ranking changes?
- How can we ensure that changes have been made properly?
- What mechanism is used to appraise others of changes that are made?

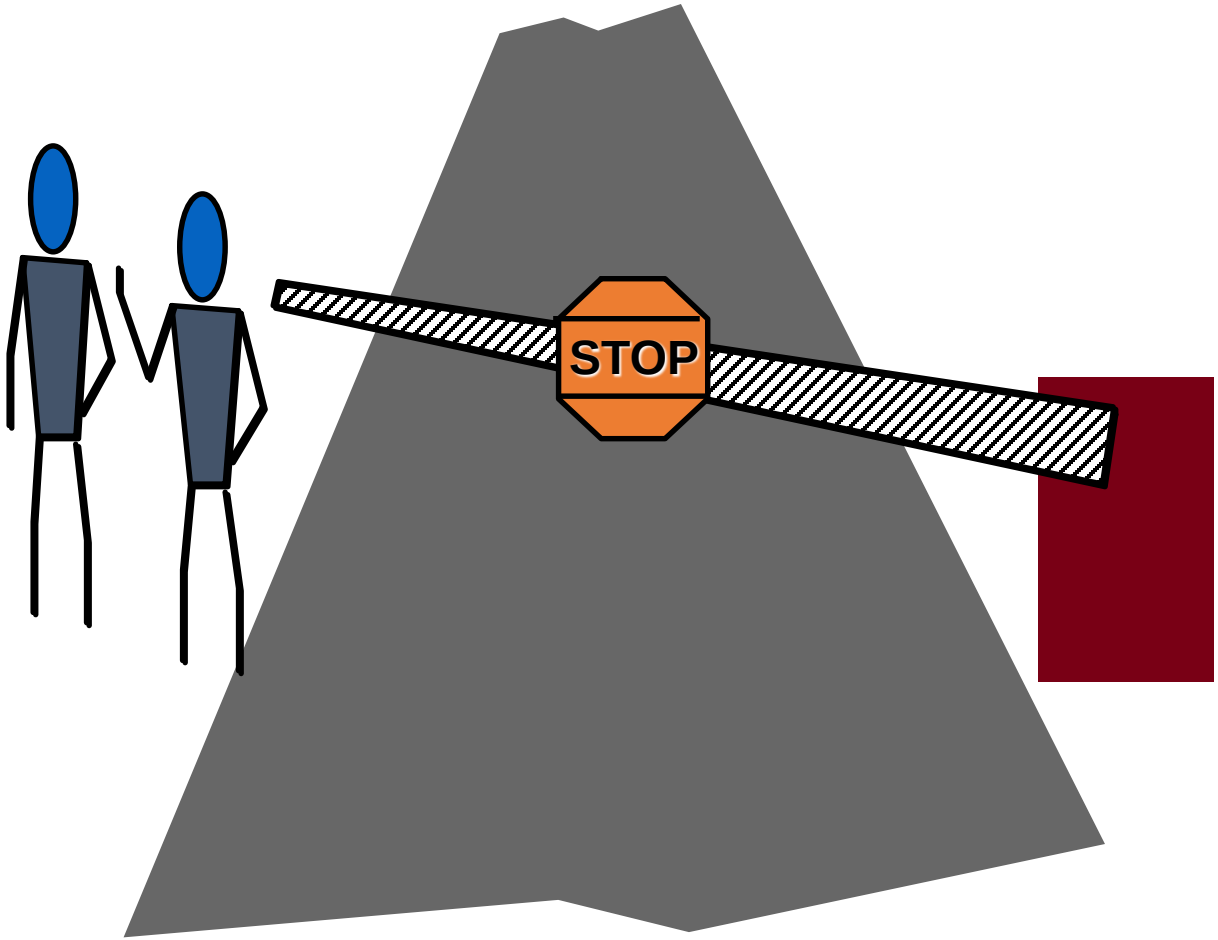
The SCM Process



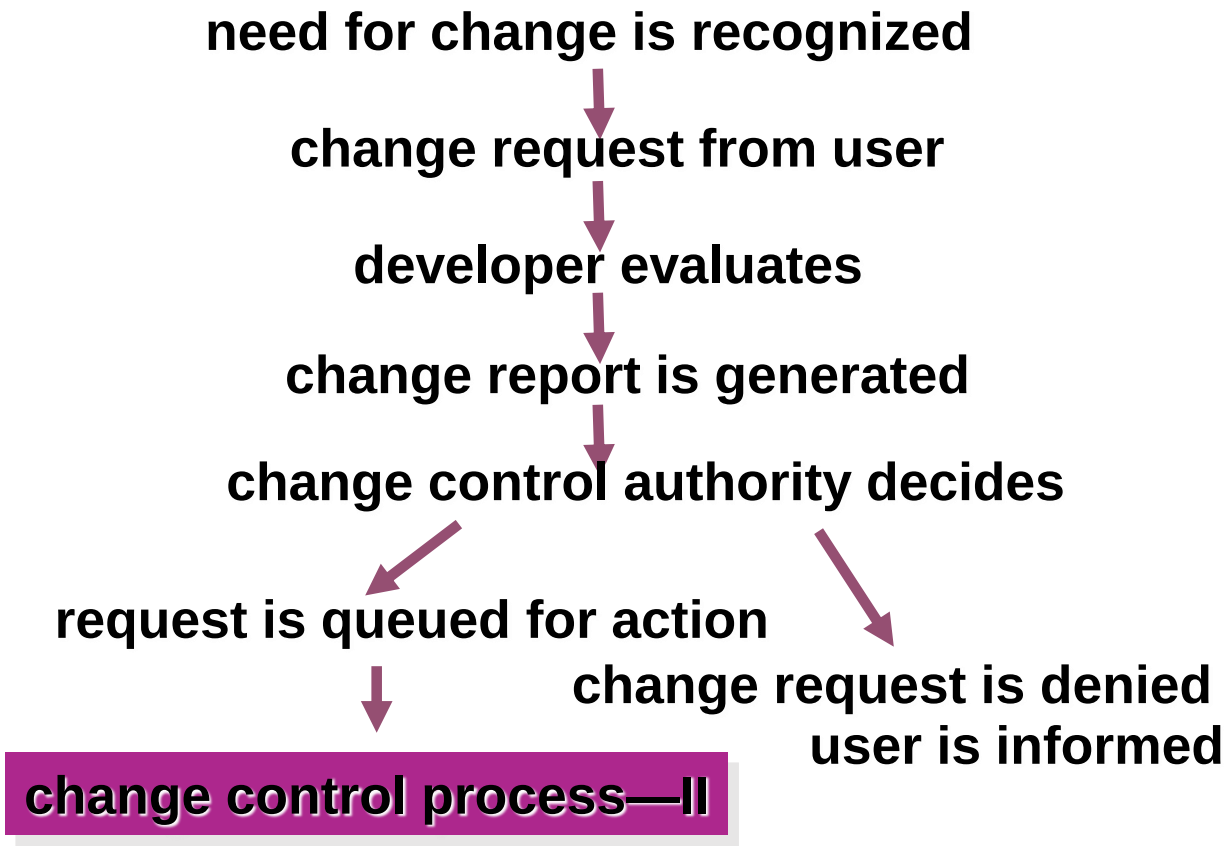
Version Control

- Version control combines procedures and tools to manage different versions of configuration objects that are created during the software process
- A version control system implements or is directly integrated with four major capabilities:
  - a *project database (repository)* that stores all relevant configuration objects
  - a *version management* capability that stores all versions of a configuration object (or enables any version to be constructed using differences from past versions);
  - a *make facility* that enables the software engineer to collect all relevant configuration objects and construct a specific version of the software.
  - an *issues tracking* (also called *bug tracking*) capability that enables the team to record and track the status of all outstanding issues associated with each configuration object.

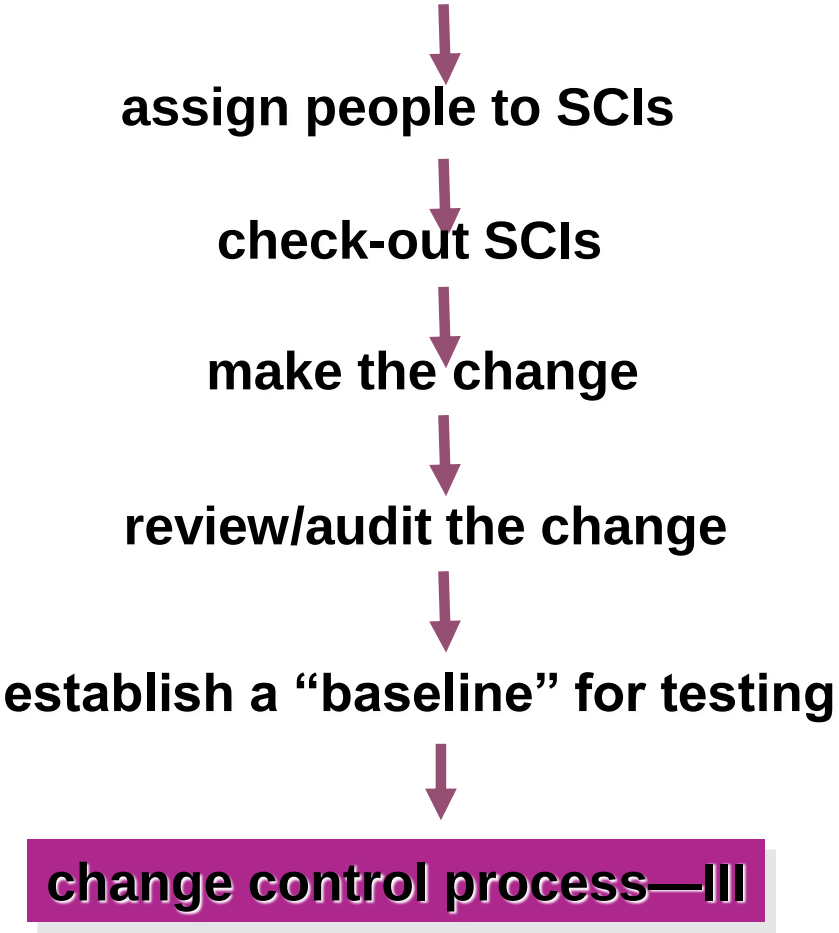
Change Control



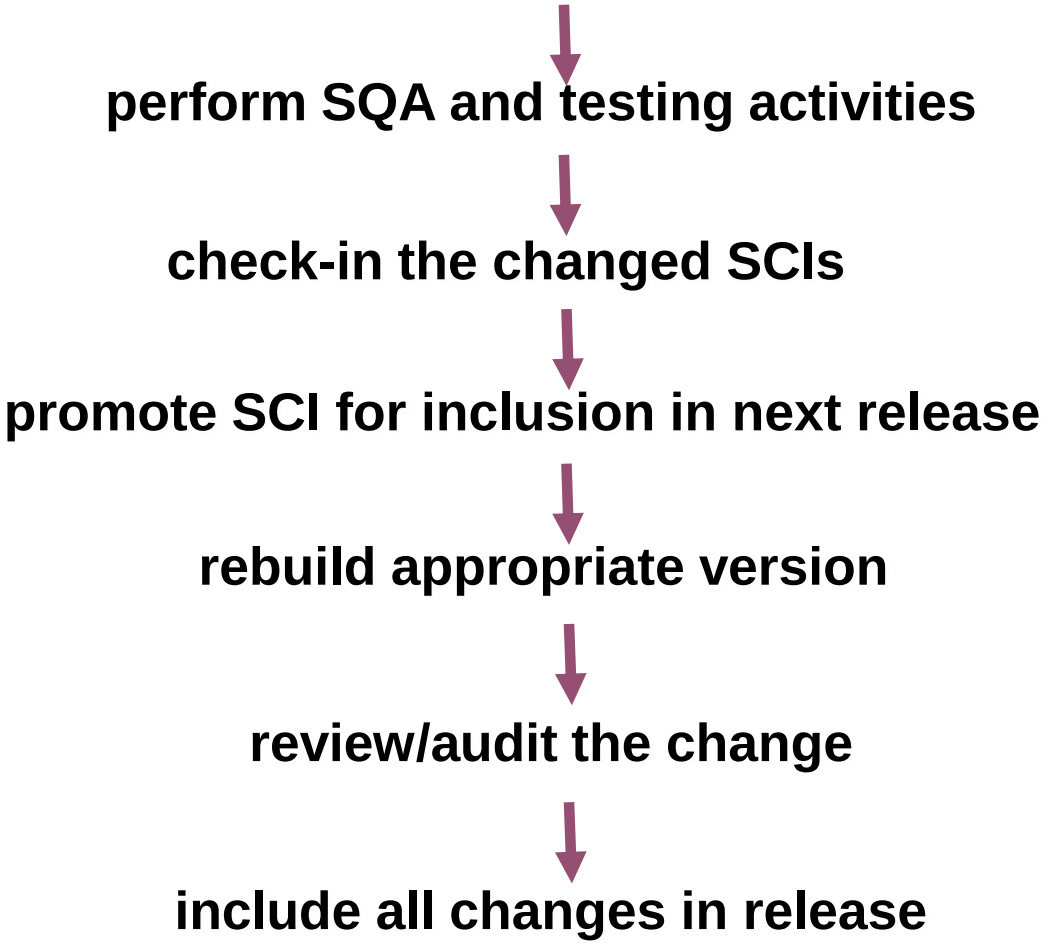
Change Control Process—I



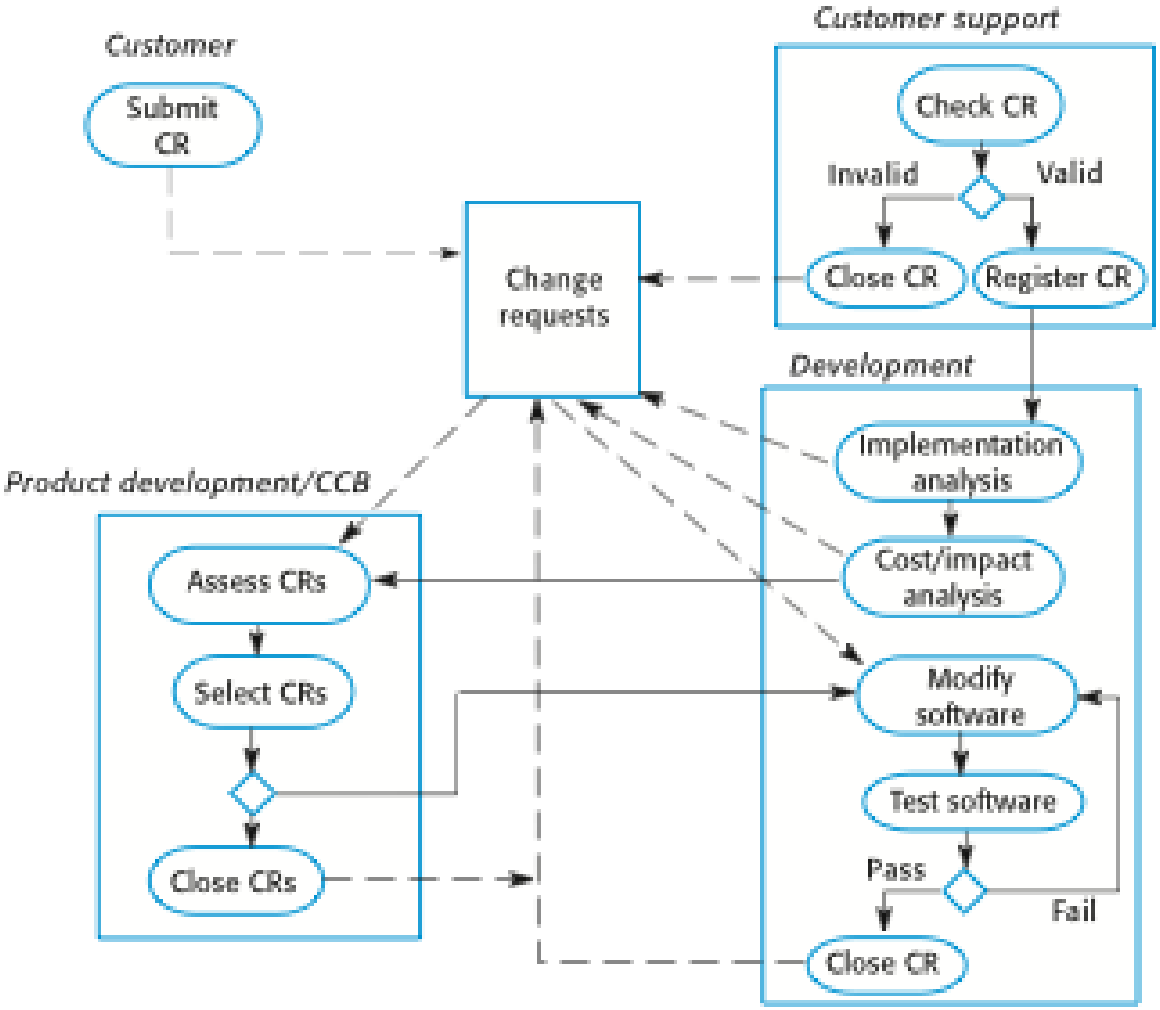
Change Control Process—II



Change Control Process—III



The change management process



Sample change request form

Change Request Form

Project: SICSA/AppProcessing **Number:** 23/02
Change requester: I. Sommerville **Date:** 20/01/09
Requested change: The status of applicants (rejected, accepted, etc.) should be shown visually in the displayed list of applicants.

Change analyzer: R. Looek **Analysis date:** 25/01/09
Components affected: ApplicantListDisplay, StatusUpdater

Associated components: StudentDatabase

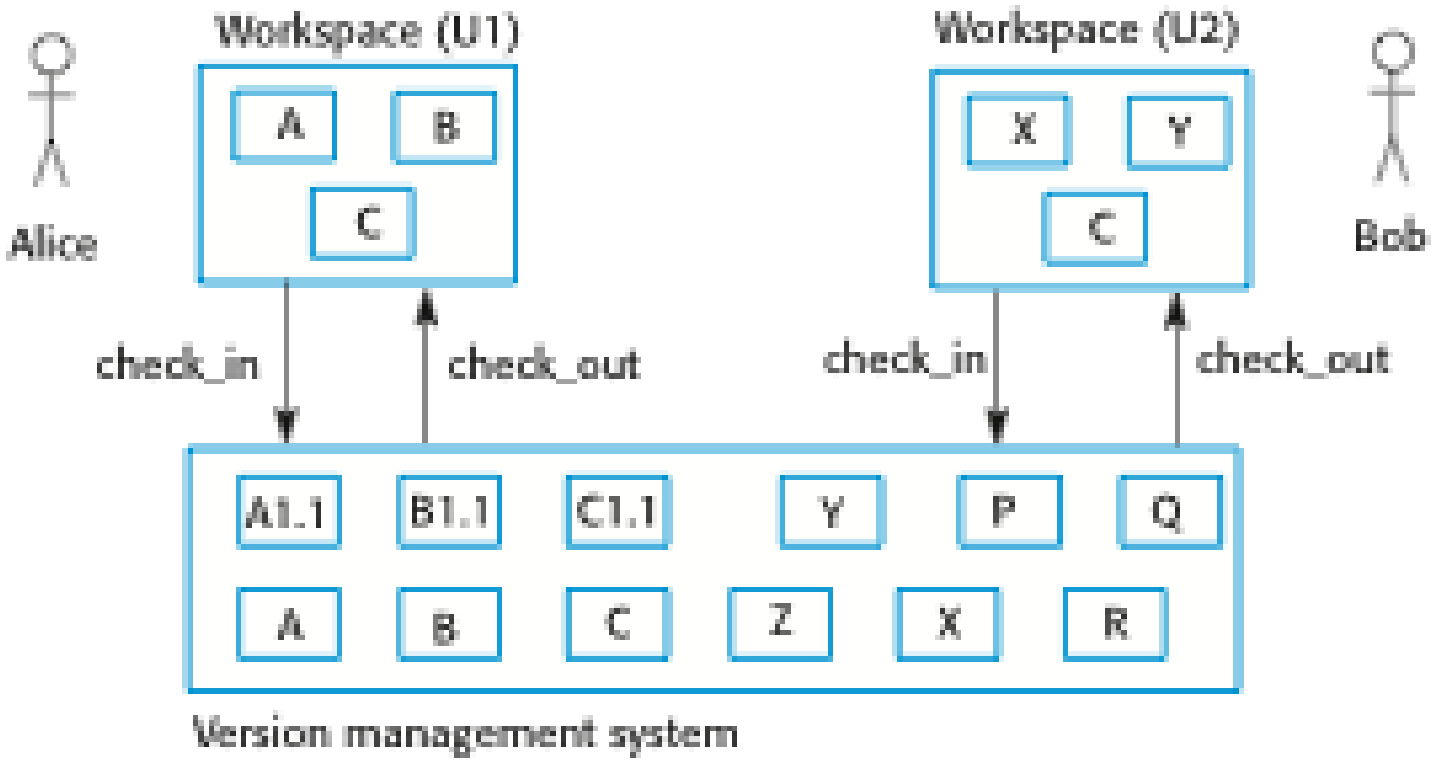
Change assessment: Relatively simple to implement by changing the display color according to status. A table must be added to relate status to colors. No changes to associated components are required.

Change priority: Medium
Change implementation:
Estimated effort: 2 hours
Date to app. team: 28/01/09 **CCB decision date:** 30/01/09
Decision: Accept change. Change to be implemented in Release 1.2
Change implementor: **Date of change:**
Date submitted to QA: **QA decision:**
Date submitted to CM:
Comments:

Factors in change analysis

- The consequences of not making the change
- The benefits of the change
- The number of users affected by the change
- The costs of making the change
- The product release cycle

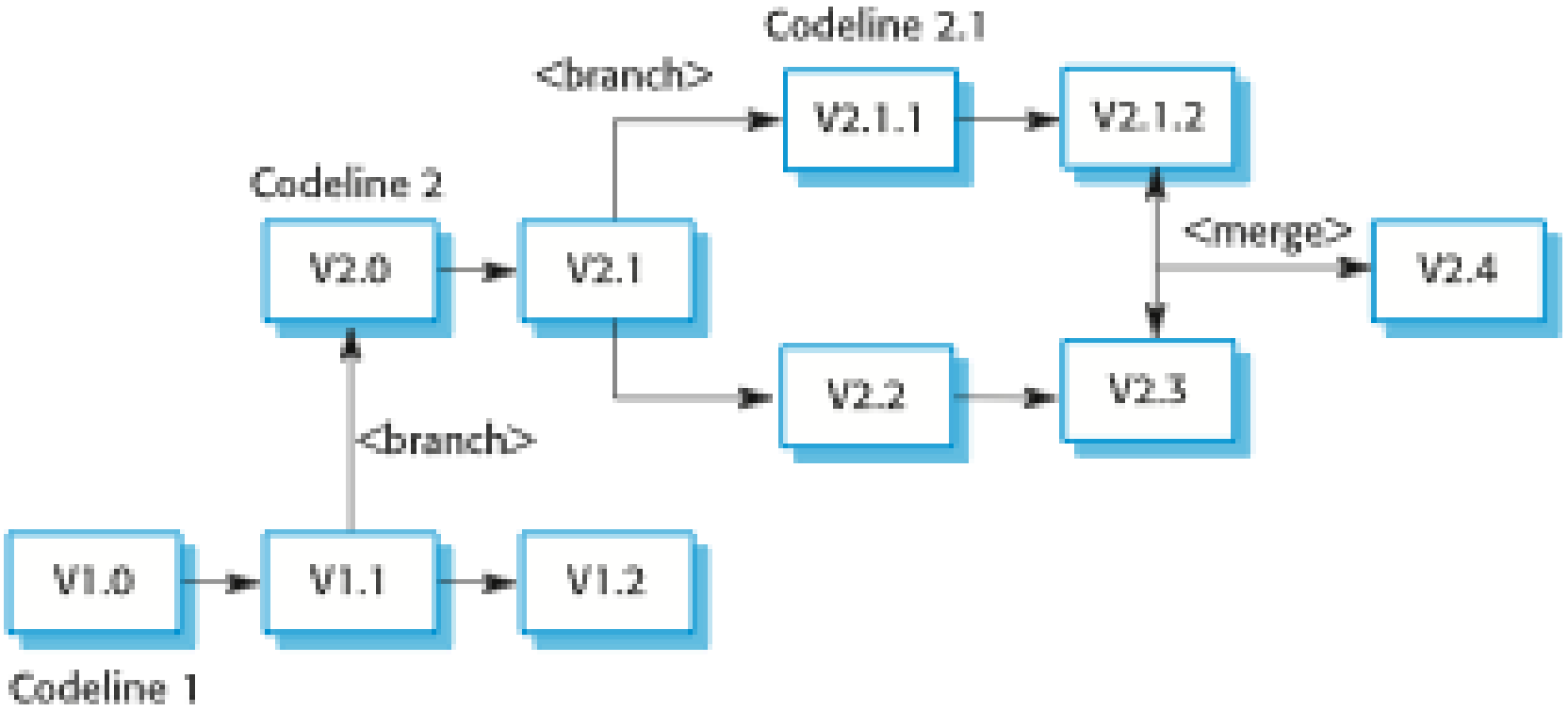
Check-in and check-out from a version repository



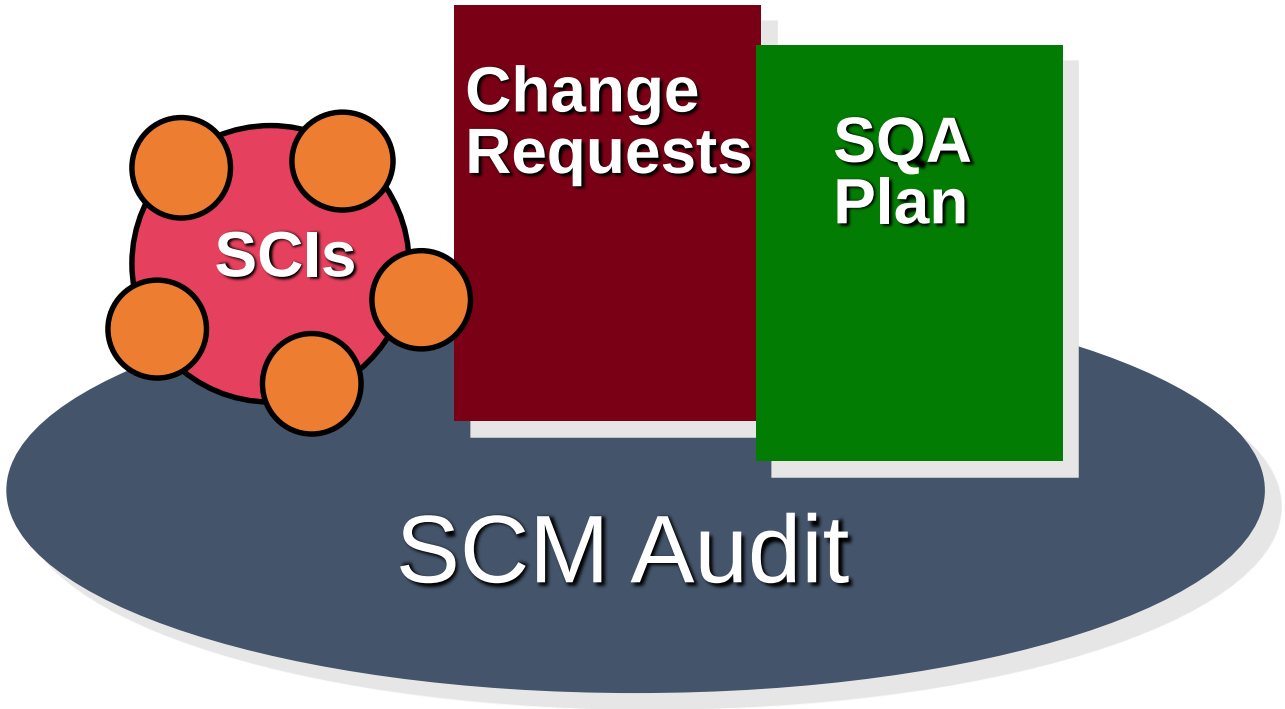
Codeline branches

- Rather than a linear sequence of versions that reflect changes to the component over time, there may be several independent sequences.
  - This is normal in system development, where different developers work independently on different versions of the source code and so change it in different ways.
- At some stage, it may be necessary to merge codeline branches to create a new version of a component that includes all changes that have been made.
  - If the changes made involve different parts of the code, the component versions may be merged automatically by combining the deltas that apply to the code.

Branching and merging



Auditing



SCM Audit

Has the change specified by the ECO(Engineering Change Order) been made without modifications?

Has a FTR been conducted to assess technical correctness?

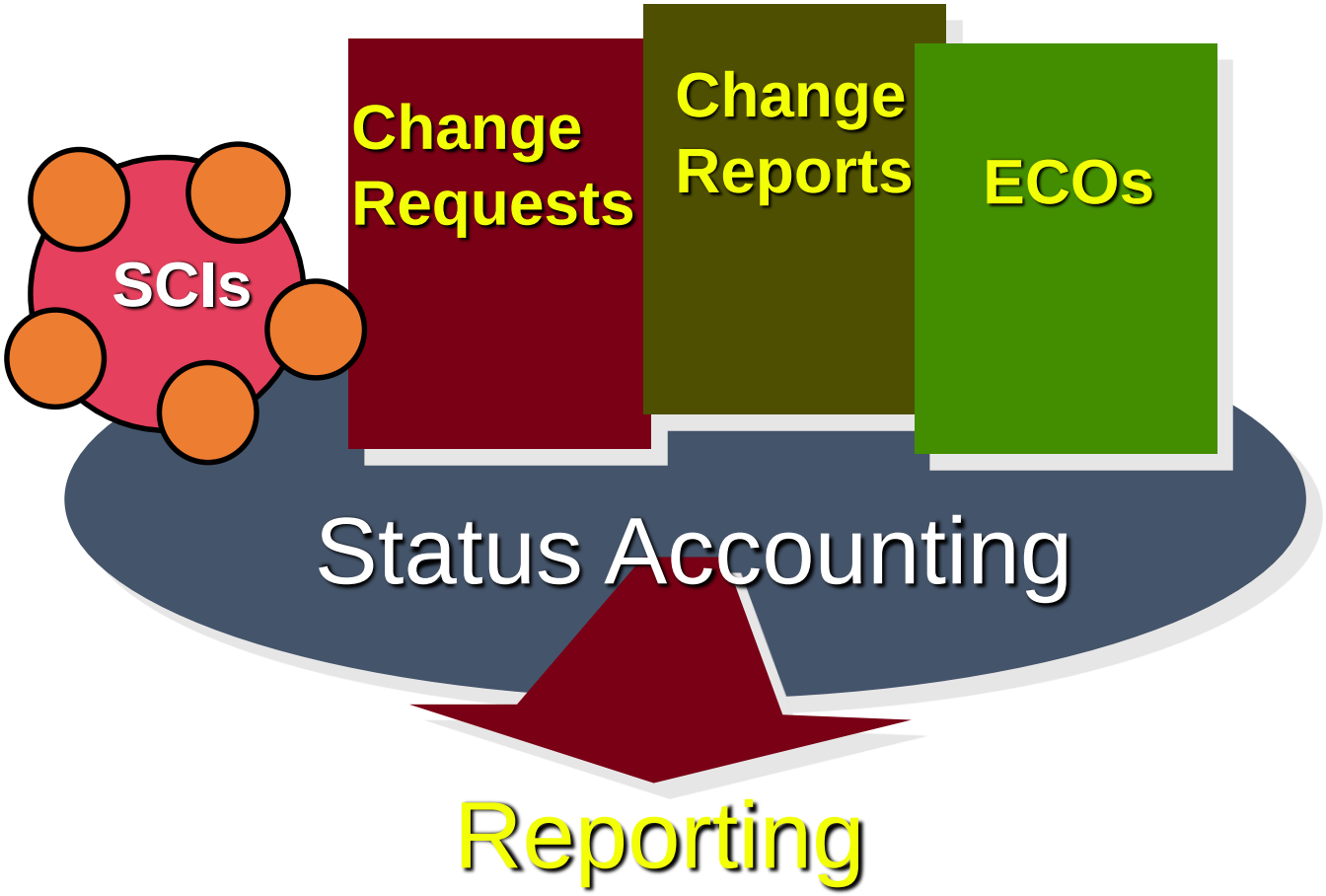
Was the software process followed and software engineering standards been properly applied?

Do the attributes of the configuration object reflect the change?

Have the SCM standards for recording and reporting the change been followed?

Were all related SCI's properly updated?

Status Accounting



What happened?

Who did it?

When did it happen?

What else will be affected by the change?

SCM for Web Engineering-I

- **Content.**
  - A typical WebApp contains a vast array of content—text, graphics, applets, scripts, audio/video files, forms, active page elements, tables, streaming data, and many others.
  - The challenge is to organize this sea of content into a rational set of configuration objects (Section 27.1.4) and then establish appropriate configuration control mechanisms for these objects.
- **People.**
  - Because a significant percentage of WebApp development continues to be conducted in an ad hoc manner, any person involved in the WebApp can (and often does) create content.

SCM for Web Engineering-II

- **Scalability.**
  - As size and complexity grow, small changes can have far-reaching and unintended affects that can be problematic. Therefore, the rigor of configuration control mechanisms should be directly proportional to application scale.
- **Politics.**
  - Who 'owns' a WebApp?
  - Who assumes responsibility for the accuracy of the information on the Web site?
  - Who assures that quality control processes have been followed before information is published to the site?
  - Who is responsible for making changes?
  - Who assumes the cost of change?

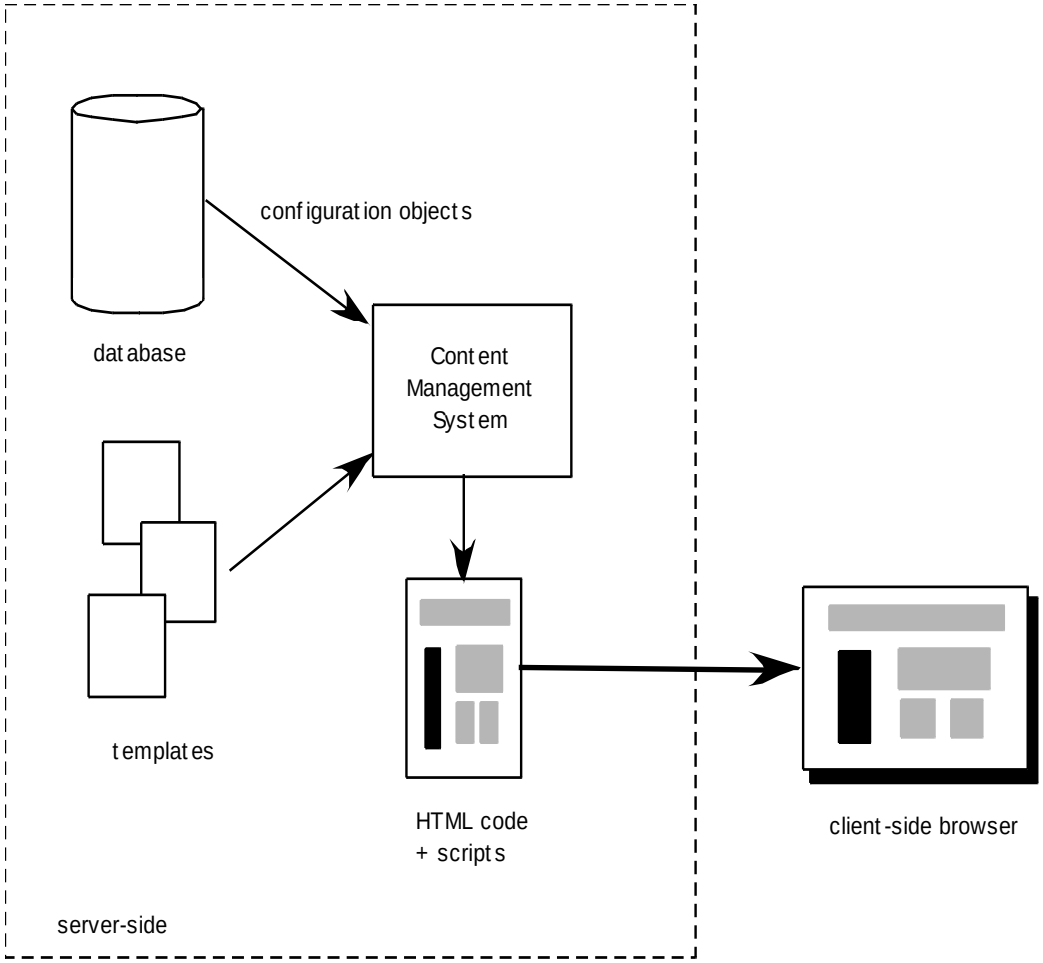
Content Management-I

- **The collection subsystem** encompasses all actions required to create and/or acquire content, and the technical functions that are necessary to
  - convert content into a form that can be represented by a mark-up language (e.g., HTML, XML
  - organize content into packets that can be displayed effectively on the client-side.
- **The management subsystem** implements a repository that encompasses the following elements:
  - *Content database*—the information structure that has been established to store all content objects
  - *Database capabilities*—functions that enable the CMS to search for specific content objects (or categories of objects), store and retrieve objects, and manage the file structure that has been established for the content
  - *Configuration management functions*—the functional elements and associated workflow that support content object identification, version control, change management, change auditing, and reporting.

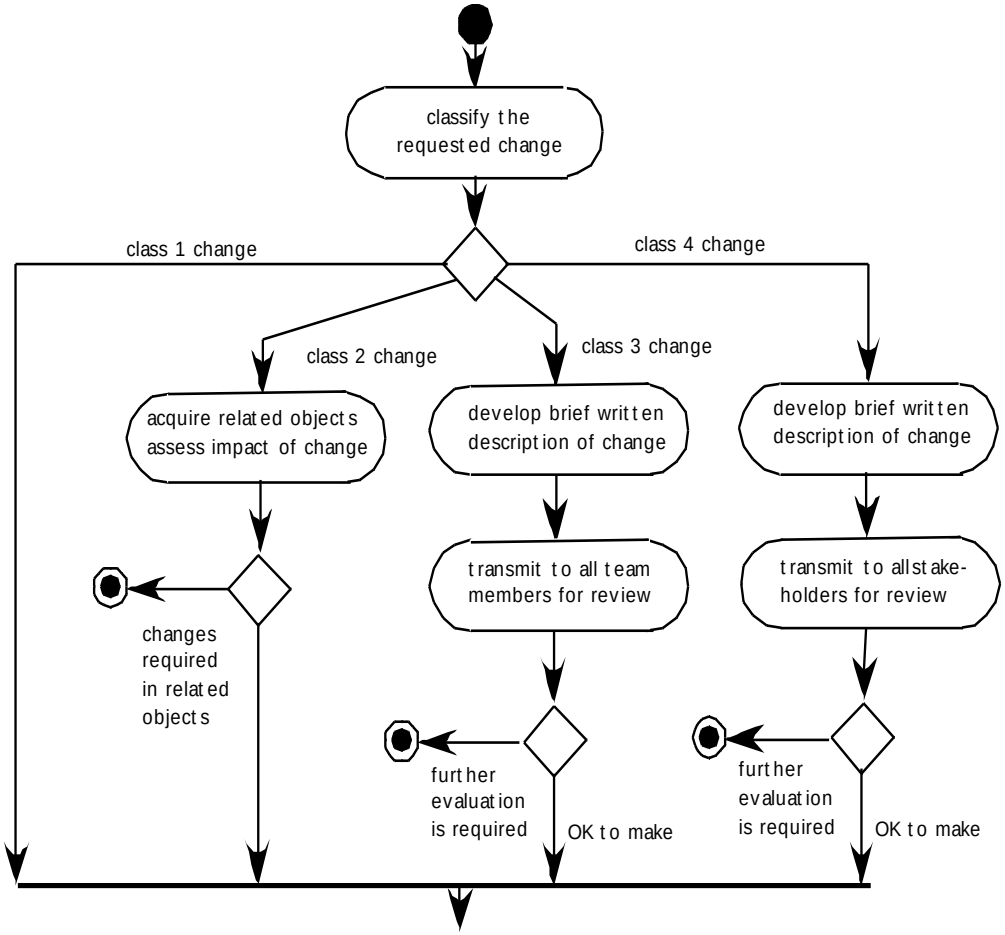
Content Management-II

- The **publishing subsystem** extracts from the repository, converts it to a form that is amenable to publication, and formats it so that it can be transmitted to client-side browsers. The publishing subsystem accomplishes these tasks using a series of templates.
- Each *template* is a function that builds a publication using one of three different components [BOI02]:
  - *Static elements*—text, graphics, media, and scripts that require no further processing are transmitted directly to the client-side
  - *Publication services*—function calls to specific retrieval and formatting services that personalize content (using predefined rules), perform data conversion, and build appropriate navigation links.
  - *External services*—provide access to external corporate information infrastructure such as enterprise data or “back-room” applications.

Content Management



Change Management for WebApps-I



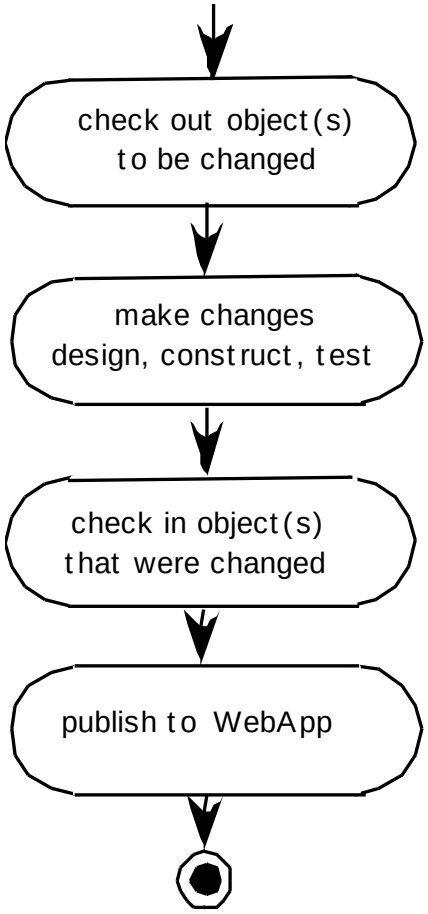
Class 1 – content or function change to correct error or enhance local content or functionality

Class 2 – content or function change that has impact on other content objects or functional components

Class 3 – content or function change that has broad impact across WebApp

Class 4 – major design change that will be noticeable to one or more categories of user

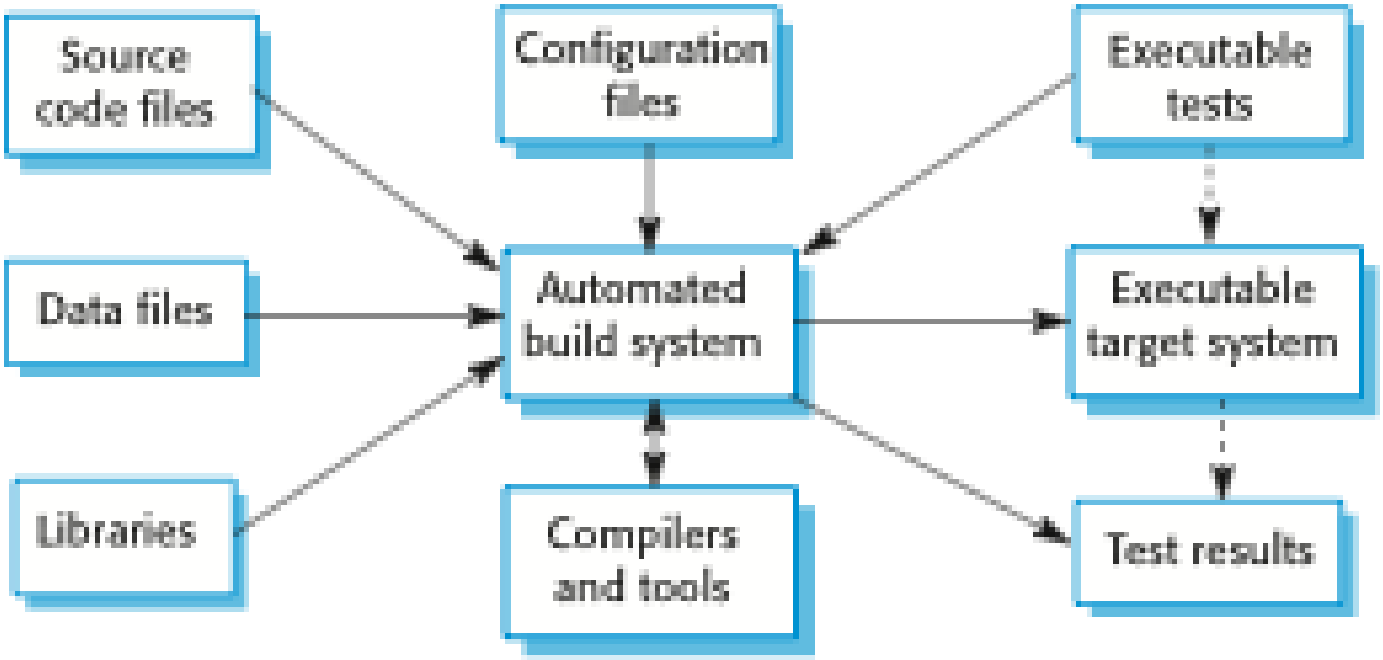
Change Management for WebApps-II



Change management with agile methods

- In agile methods, customers are generally directly involved in change management.
- The propose a change to the requirements and work with the team to assess its impact and decide whether the change should take priority over the features planned for the next increment of the system.
- Changes to improve the software improvement are decided by the programmers working on the system.
- Refactoring, where the software is continually improved, is not seen as an overhead but as a necessary part of the development process.

System building



Build system functionality

- Build script generation
- Version management system integration
- Minimal re-compilation
- Executable system creation
- Test automation
- Reporting
- Documentation generation

Minimizing recompilation

- Tools to support system building are usually designed to minimize the amount of compilation that is required.
- They do this by checking if a compiled version of a component is available. If so, there is no need to recompile that component.
- A unique signature identifies each source and object code version and is changed when the source code is edited.
- By comparing the signatures on the source and object code files, it is possible to decide if the source code was used to generate the object code component.

File identification

- Modification timestamps
  - The signature on the source code file is the time and date when that file was modified. If the source code file of a component has been modified after the related object code file, then the system assumes that recompilation to create a new object code file is necessary.
- Source code checksums
  - The signature on the source code file is a checksum calculated from data in the file. A checksum function calculates a unique number using the source text as input. If you change the source code (even by 1 character), this will generate a different checksum. You can therefore be confident that source code files with different checksums are actually different.

Timestamps vs checksums

- Timestamps
  - Because source and object files are linked by name rather than an explicit source file signature, it is not usually possible to build different versions of a source code component into the same directory at the same time, as these would generate object files with the same name.
- Checksums
  - When you recompile a component, it does not overwrite the object code, as would normally be the case when the timestamp is used. Rather, it generates a new object code file and tags it with the source code signature. Parallel compilation is possible and different versions of a component may be compiled at the same time.

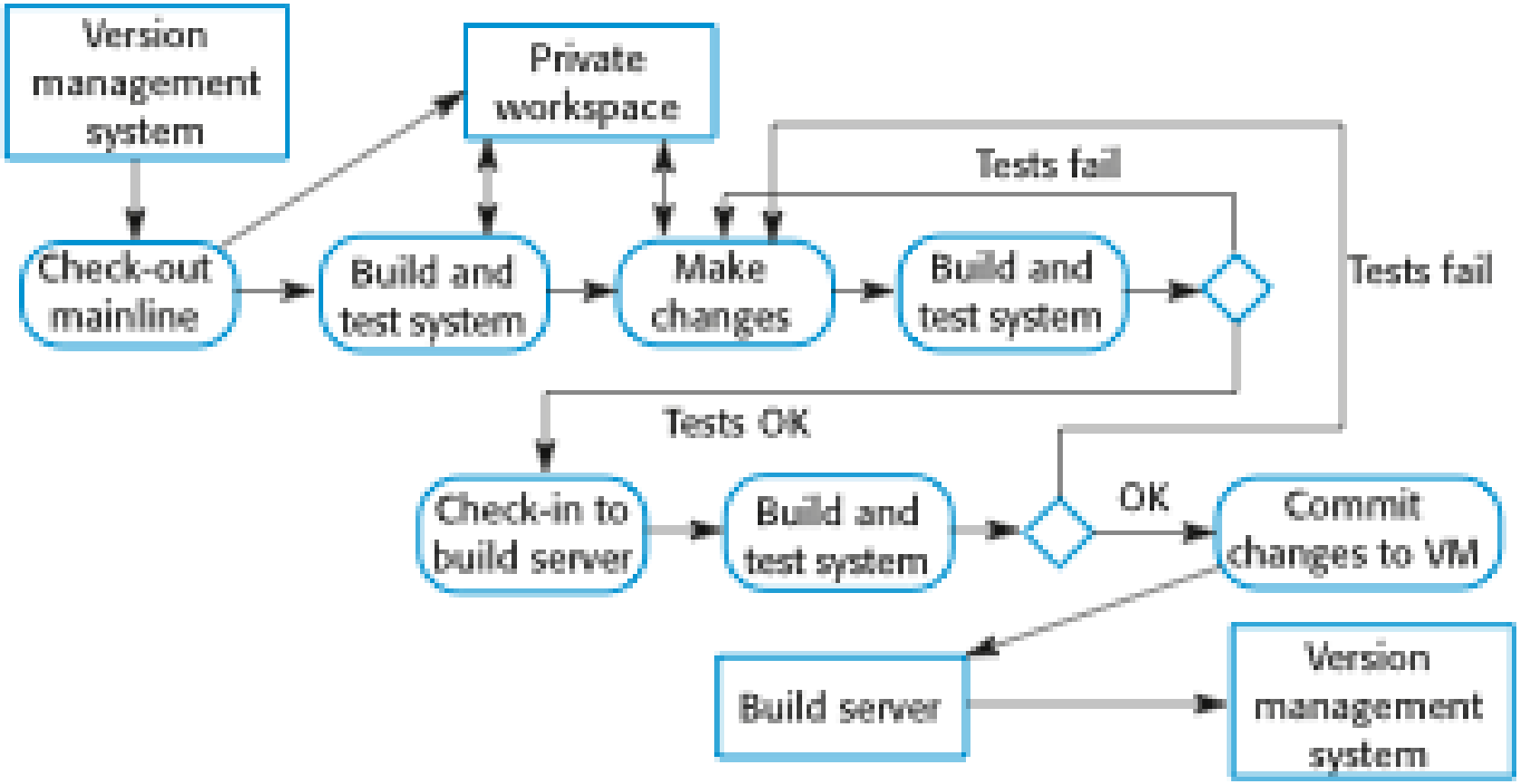
Agile building

- Check out the mainline system from the version management system into the developer's private workspace.
- Build the system and run automated tests to ensure that the built system passes all tests. If not, the build is broken and you should inform whoever checked in the last baseline system. They are responsible for repairing the problem.
- Make the changes to the system components.
- Build the system in the private workspace and rerun system tests. If the tests fail, continue editing.

Agile building

- Once the system has passed its tests, check it into the build system but do not commit it as a new system baseline.
- Build the system on the build server and run the tests. You need to do this in case others have modified components since you checked out the system. If this is the case, check out the components that have failed and edit these so that tests pass on your private workspace.
- If the system passes its tests on the build system, then commit the changes you have made as a new baseline in the system mainline.

Continuous integration



Continuous integration

Check out the mainline system from the version management system into the developer's private workspace

Build the system and run automated tests to ensure that the built system passes. If not, the build is broken – Alert!

Make changes to the system components

Build the system in the private workspace and rerun tests. Continue changes until tests are passed.

Once tests are passed, check the system into build system, but do not commit it as new system baseline

Rerun tests on the build server, to ensure that your changes work well the any changes that might have happened since you checked out

Commit changes into new baseline, upon passing tests on the build server

Daily building

- The development organization sets a delivery time (say 2 p.m.) for system components.
  - If developers have new versions of the components that they are writing, they must deliver them by that time.
  - A new version of the system is built from these components by compiling and linking them to form a complete system.
  - This system is then delivered to the testing team, which carries out a set of predefined system tests
  - Faults that are discovered during system testing are documented and returned to the system developers. They repair these faults in a subsequent version of the component.

Release management

- A system release is a version of a software system that is distributed to customers.
- For mass market software, it is usually possible to identify two types of release: major releases which deliver significant new functionality, and minor releases, which repair bugs and fix customer problems that have been reported.
- For custom software or software product lines, releases of the system may have to be produced for each customer and individual customers may be running several different releases of the system at the same time.

Release tracking

- In the event of a problem, it may be necessary to reproduce exactly the software that has been delivered to a particular customer.
- When a system release is produced, it must be documented to ensure that it can be re-created exactly in the future.
- This is particularly important for customized, long-lifetime embedded systems, such as those that control complex machines.
  - Customers may use a single release of these systems for many years and may require specific changes to a particular software system long after its original release date.

Release reproduction

- To document a release, you have to record the specific versions of the source code components that were used to create the executable code.
- You must keep copies of the source code files, corresponding executables and all data and configuration files.
- You should also record the versions of the operating system, libraries, compilers and other tools used to build the software.

Release planning

- As well as the technical work involved in creating a release distribution, advertising and publicity material have to be prepared and marketing strategies put in place to convince customers to buy the new release of the system.
- Release timing
  - If releases are too frequent or require hardware upgrades, customers may not move to the new release, especially if they have to pay for it.
  - If system releases are too infrequent, market share may be lost as customers move to alternative systems.

Release components

- As well as the the executable code of the system, a release may also include:
  - configuration files defining how the release should be configured for particular installations;
  - data files, such as files of error messages, that are needed for successful system operation;
  - an installation program that is used to help install the system on target hardware;
  - electronic and paper documentation describing the system;
  - packaging and associated publicity that have been designed for that release.

Factors influencing system release planning

Factor	Description
Technical quality of the system	If serious system faults are reported which affect the way in which many customers use the system, it may be necessary to issue a fault repair release. Minor system faults may be repaired by issuing patches (usually distributed over the Internet) that can be applied to the current release of the system.
Platform changes	You may have to create a new release of a software application when a new version of the operating system platform is released.
Lehman's fifth law	This 'law' suggests that if you add a lot of new functionality to a system; you will also introduce bugs that will limit the amount of functionality that may be included in the next release. Therefore, a system release with significant new functionality may have to be followed by a release that focuses on repairing problems and improving performance.

Factors influencing system release planning

Factor	Description
Competition	For mass-market software, a new system release may be necessary because a competing product has introduced new features and market share may be lost if these are not provided to existing customers.
Marketing requirements	The marketing department of an organization may have made a commitment for releases to be available at a particular date.
Customer change proposals	For custom systems, customers may have made and paid for a specific set of system change proposals, and they expect a system release as soon as these have been implemented.

Thank You

Any Questions?