Course Lecture Schedule

Maria
Susan
Sami Hyrynsalmi

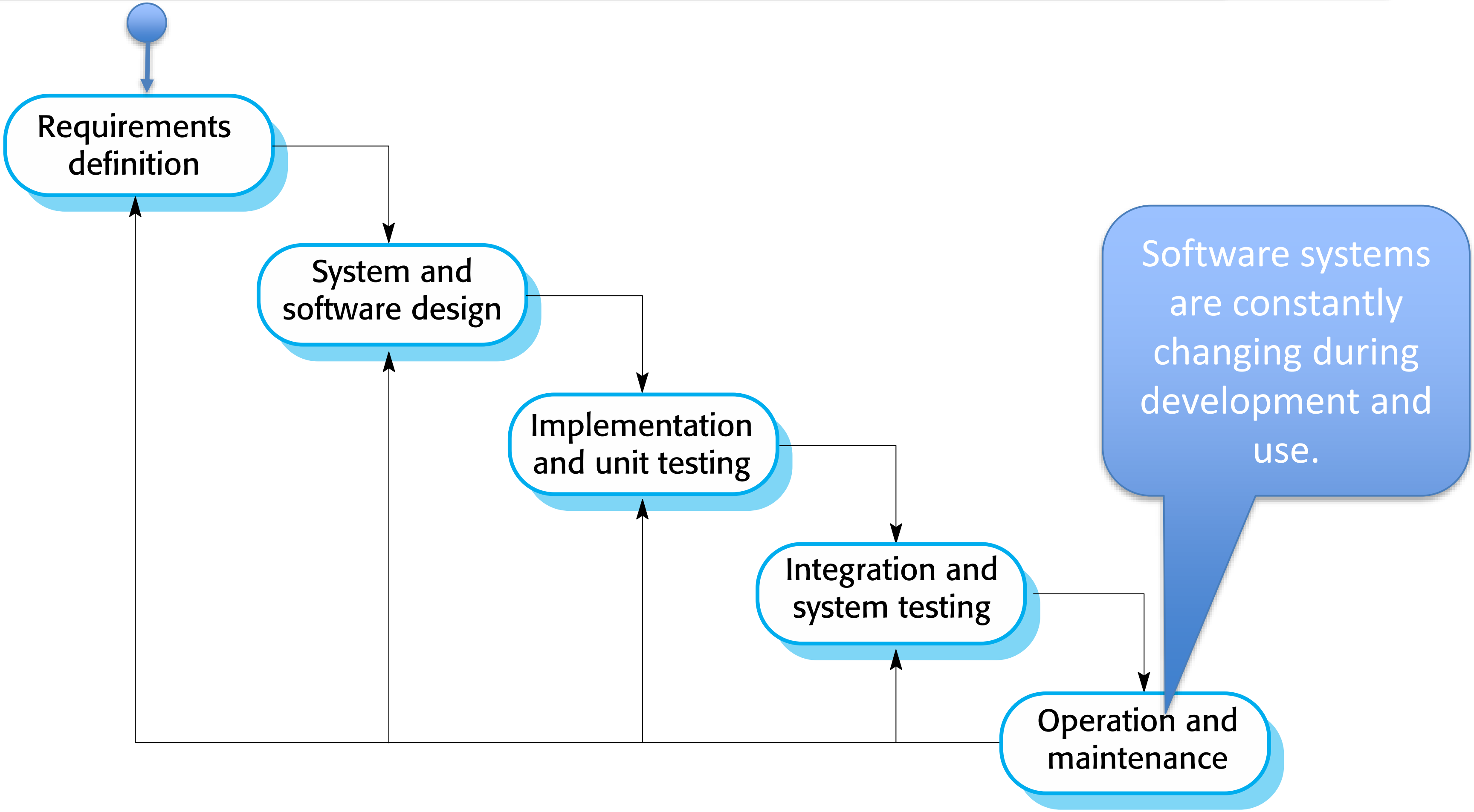
Date	Topic	Book Chapter(s)
Wed 8.9.	Course introduction	
Tue 14.9.	Introduction to Software Engineering	Chapter 1
Tue 21.9.	Software Processes	Chapter 2
Mon 27.9	Agile Software Engineering	Chapter 3
Tue 5.10.	Requirements Engineering	Chapter 4
Mon 11.10.	Architectural Design	Chapter 6
Wed 20.10.	Modeling and implementation	Chapters 5 & 7
Mon 1.11.	Testing & Quality	Chapters 8 & 24
Mon 8.11.	**Software Evolution & Configuration Management**	**Chapters 9 & 25**
Mon 15.11.	Software Project Management	Chapter 22
Mon 22.11.	Software Project Planning	Chapter 23
Mon 29.11.	Global Software Engineering	
Wed 8.12.	Software Business	
Mon 13.12.	Last topics	

Lecture 9
Software Evolution & Configuration Management

Topics covered

- Evolution processes
- Legacy systems
- Software maintenance
- Configuration management
  - Version management
  - System building
  - Change management
  - Release management

Testing stage in waterfall model


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Software systems are constantly changing during development and use.

Software change（evolution）

✧ 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.

✧ Proposals for change are the driver for system evolution.

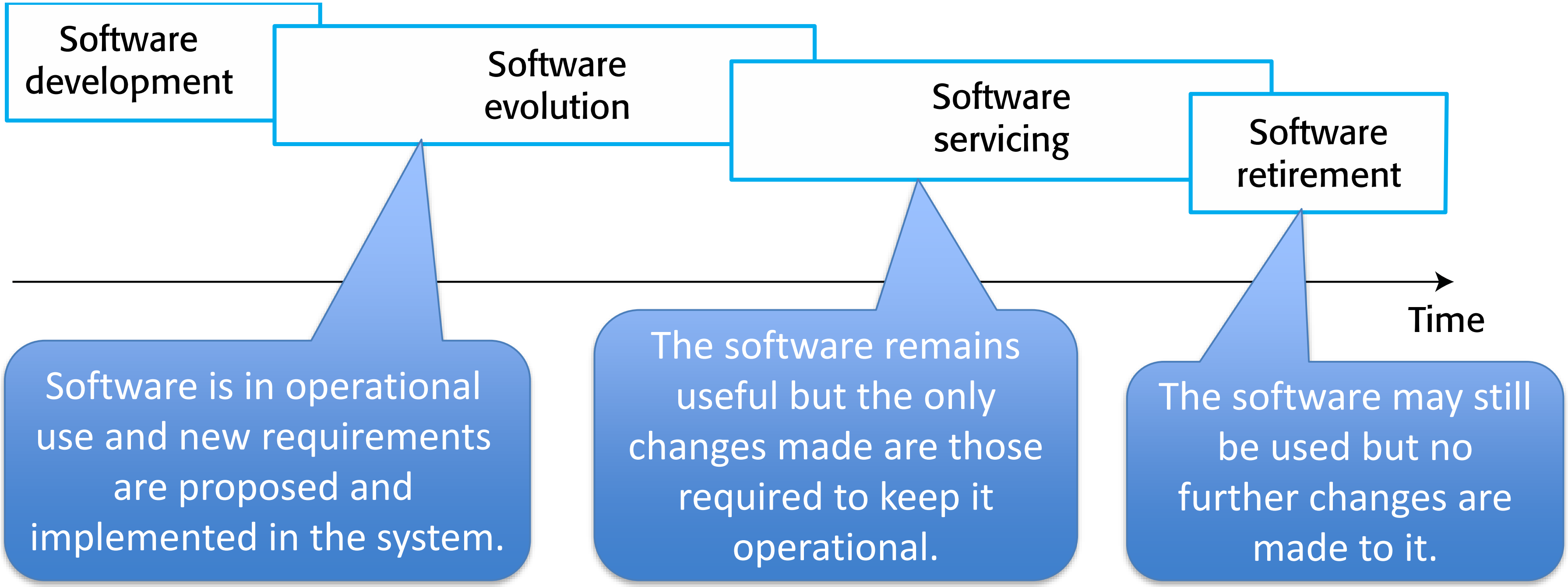
Configuration management

- ✧ Software systems are constantly changing during development and use.
- ✧ Configuration management (CM) is concerned with the policies, processes and tools for managing changing software systems.

Evolution processes

Evolution and servicing

- ✧ Organizations have huge investments in their software systems - they must maintain the value of these assets.
- ✧ The majority of the software budget in large companies is devoted to changing and evolving existing software rather than developing new software.



Software development

Software evolution

Software servicing

Software retirement

Time

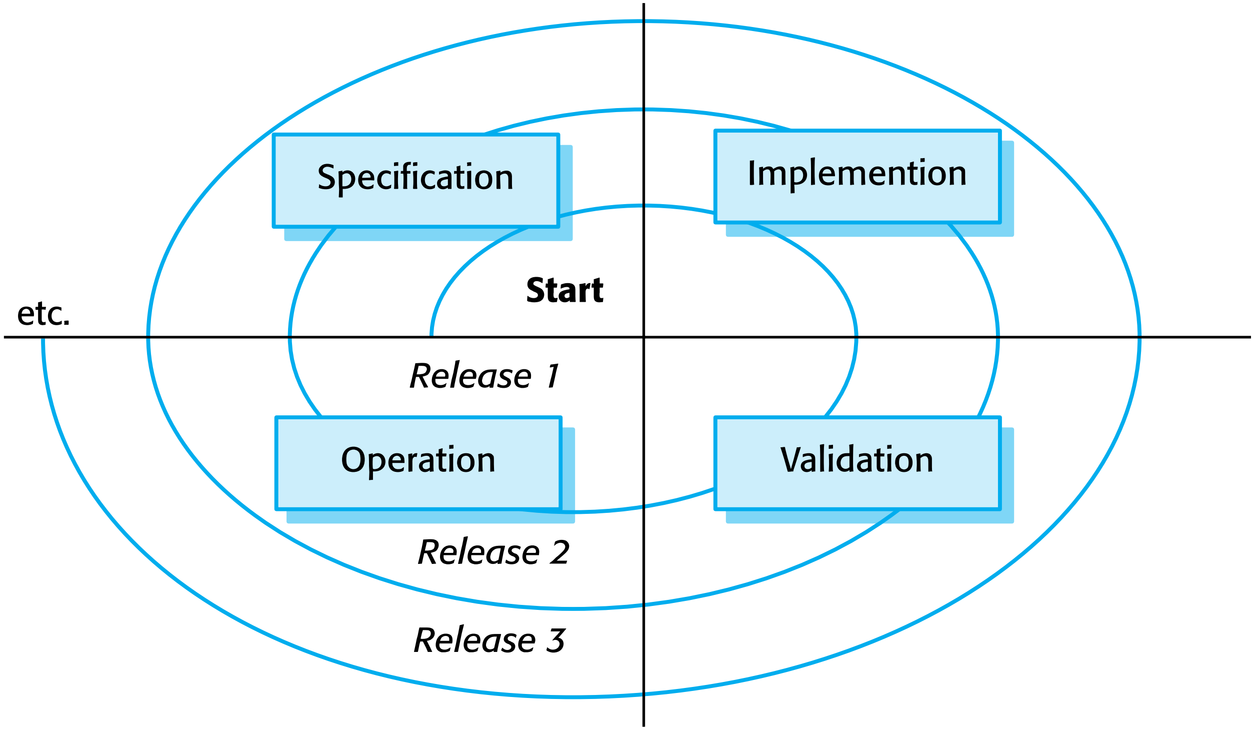
Software is in operational use and new requirements are proposed and implemented in the system.

The software remains useful but the only changes made are those required to keep it operational.

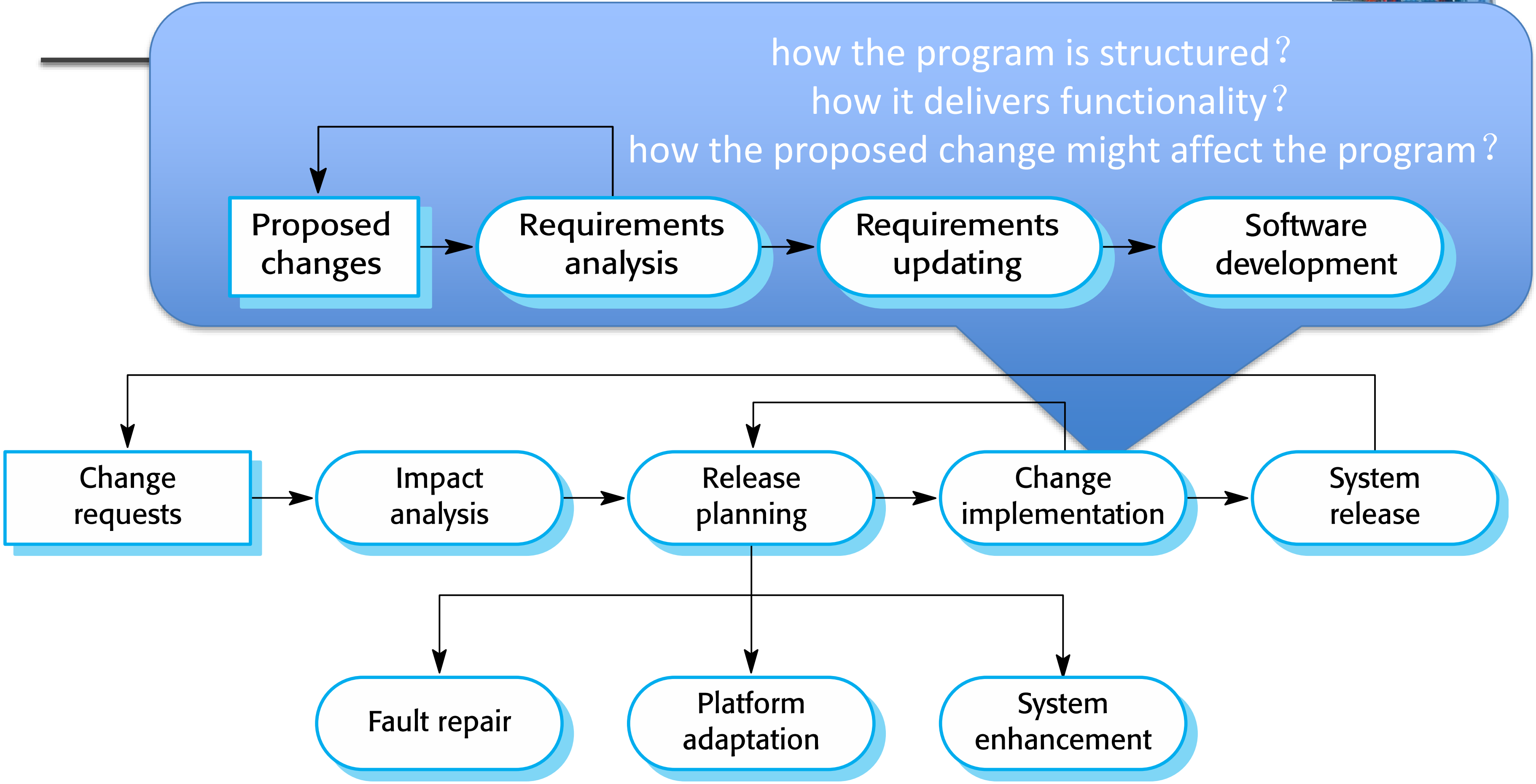
The software may still be used but no further changes are made to it.

A spiral model of development and evolution

- ✧ Change identification and evolution continues throughout the system lifetime.



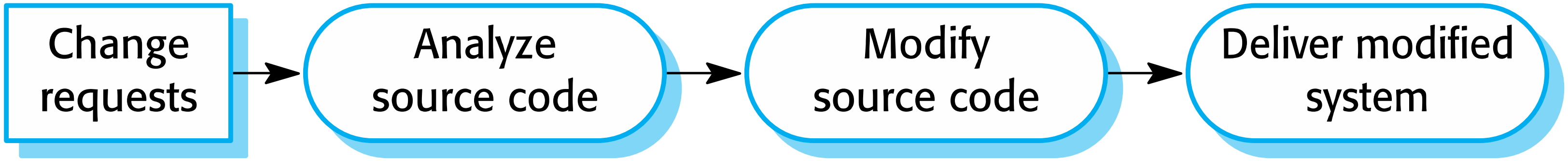
The software evolution process



how the program is structured？
how it delivers functionality？
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
    - Serious system fault has to be repaired to allow normal operation to continue;
    - Changes to the system’s environment (e.g. an OS upgrade) have unexpected effects;
    - Business changes that require a very rapid response (e.g. the release of a competing product).

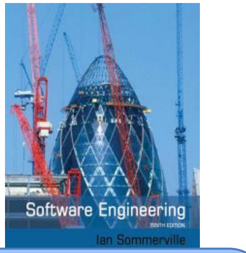


Legacy systems

Legacy systems

- ✧ Legacy systems are older systems that rely on languages and technology that are no longer used for new systems development.
- ✧ Legacy software may be dependent on older hardware, such as mainframe computers and may have associated legacy processes and procedures.
- ✧ Legacy systems are not just software systems but are broader socio-technical systems that include hardware, software, libraries and other supporting software and business processes.
- ✧ Legacy system replacement is risky and expensive so businesses continue to use these systems.

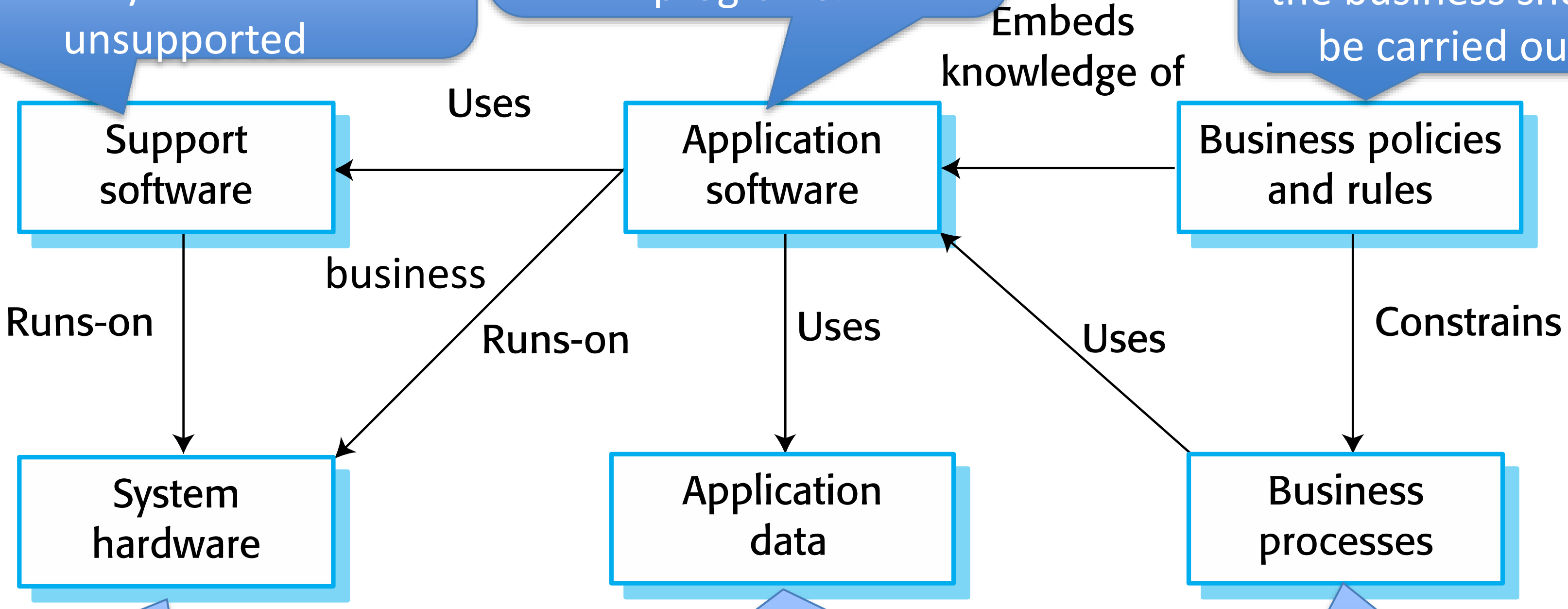
The elements of a legacy system



legacy system may rely on a range of support software, which may be obsolete or unsupported

Business services are usually made up of a number of application programs.

Definitions and constraints of how the business should be carried out.



Legacy systems may have been written for hardware that is no longer available.

Application data processed by the system may be inconsistent, duplicated or held in different databases.

Business processes are used to achieve some business objective.

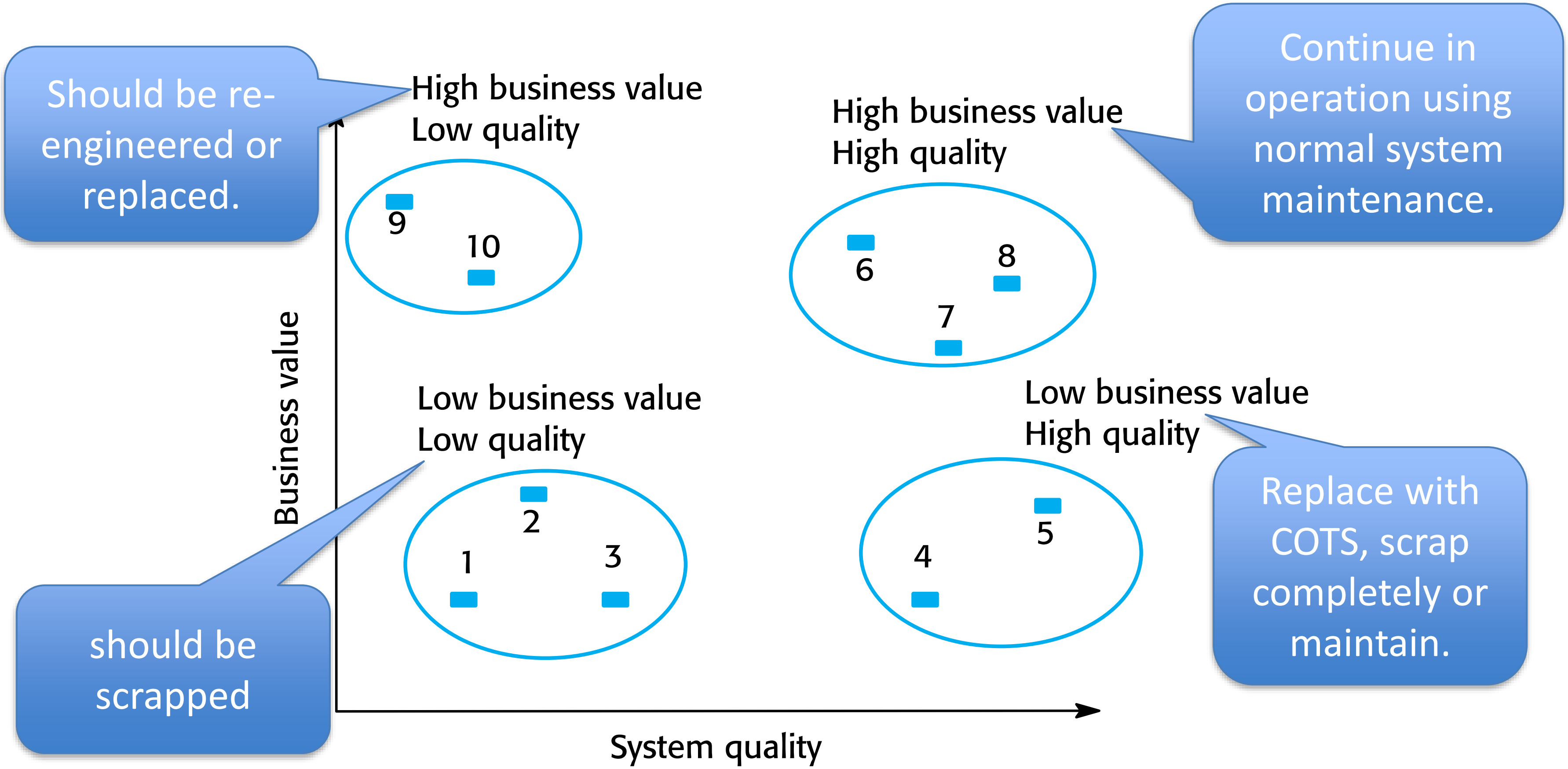
Legacy system replacement or change

- ✧ System replacement is risky for a number of reasons :
  - Lack of complete system specification
  - Tight integration of system and business processes
  - Undocumented business rules embedded in the legacy system
  - New software development may be late and/or over budget
- ✧ Legacy systems are expensive to change for a number of reasons:
  - No consistent programming style
  - Use of obsolete programming languages
  - Inadequate system documentation
  - System structure degradation
  - Program optimizations may make them hard to understand
  - Data errors, duplication and inconsistency

Legacy system management

- ✧ Organizations 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.

Figure 9.13 An example of a legacy system assessment



Should be re-engineered or replaced.

High business value
Low quality

9
10

High business value
High quality

6
8
7

Continue in operation using normal system maintenance.

Business value

Low business value
Low quality

2
1
3

Low business value
High quality

4
5

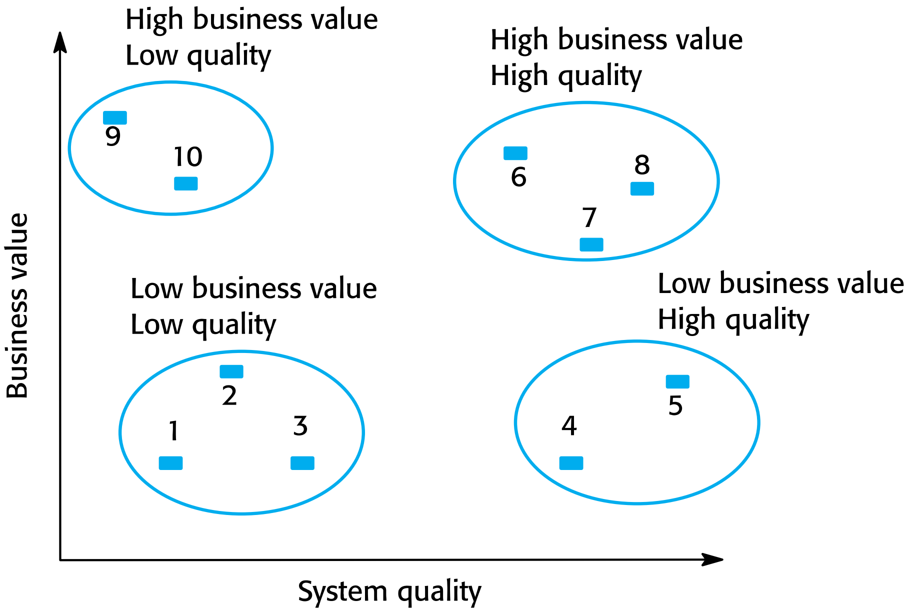
Replace with COTS, scrap completely or maintain.

should be scrapped

System quality

Business value & system quality assessment

- ✧ The use of the system
- ✧ supported business processes
- ✧ System dependability
- ✧ The system outputs



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

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?

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?

System measurement

- ✧ You may collect quantitative data to make an assessment of the quality of the application system
  - The number of system change requests-The higher this accumulated value, the lower the quality of the system.
  - The number of different user interfaces used by the system-The more interfaces, the more likely it is that there will be inconsistencies and redundancies in these interfaces.
  - The volume of data used by the system-The volume of data processed by the system increases, inconsistencies and errors in that data increase. Cleaning up old data is a very expensive and time consuming process.

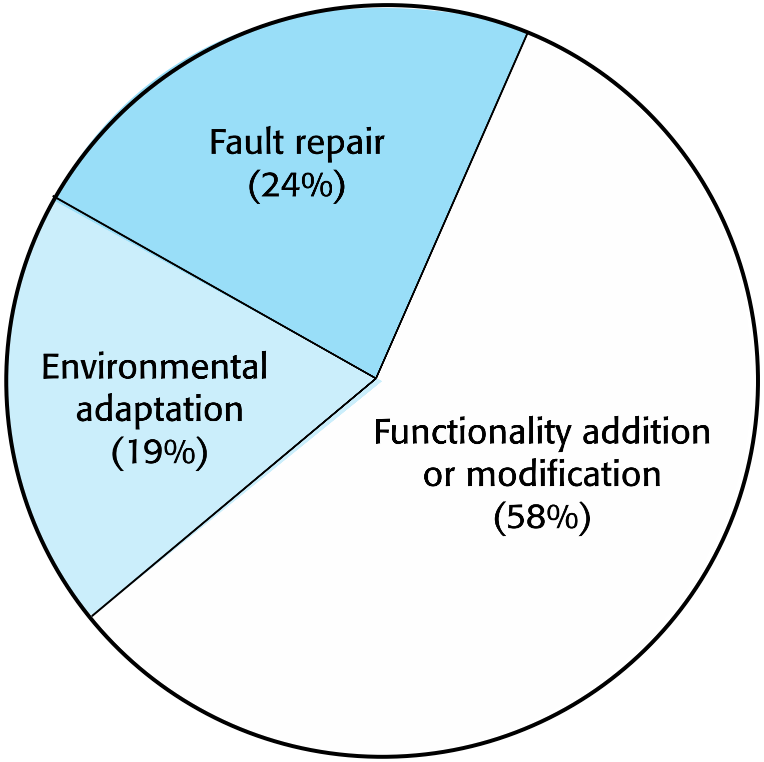
Software maintenance

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

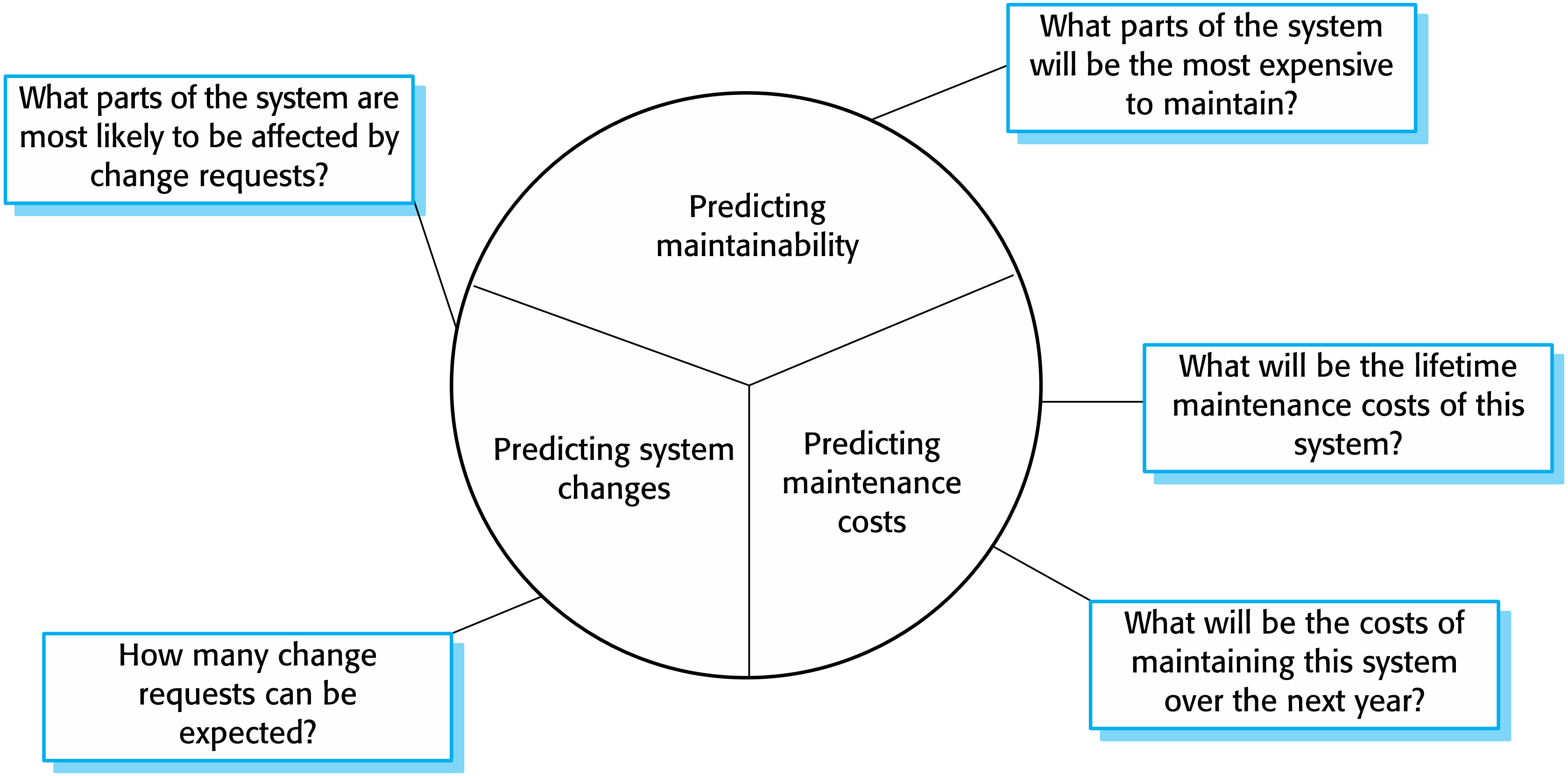
- ✧ Fault repairs
  - fix bugs/vulnerabilities and correct
- ✧ Environmental adaptation
  - adapt software to a different operating environment
  - changing a system so that it operates in a different environment (computer, OS, etc.)
- ✧ Functionality addition and modification
  - modifying the system to satisfy new requirements



Maintenance costs

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

Maintenance prediction


Predicting maintainability
Predicting system changes
Predicting maintenance costs
What parts of the system are most likely to be affected by change requests?
What parts of the system will be the most expensive to maintain?
What will be the lifetime maintenance costs of this system?
What will be the costs of maintaining this system over the next year?
How many change requests can be expected?

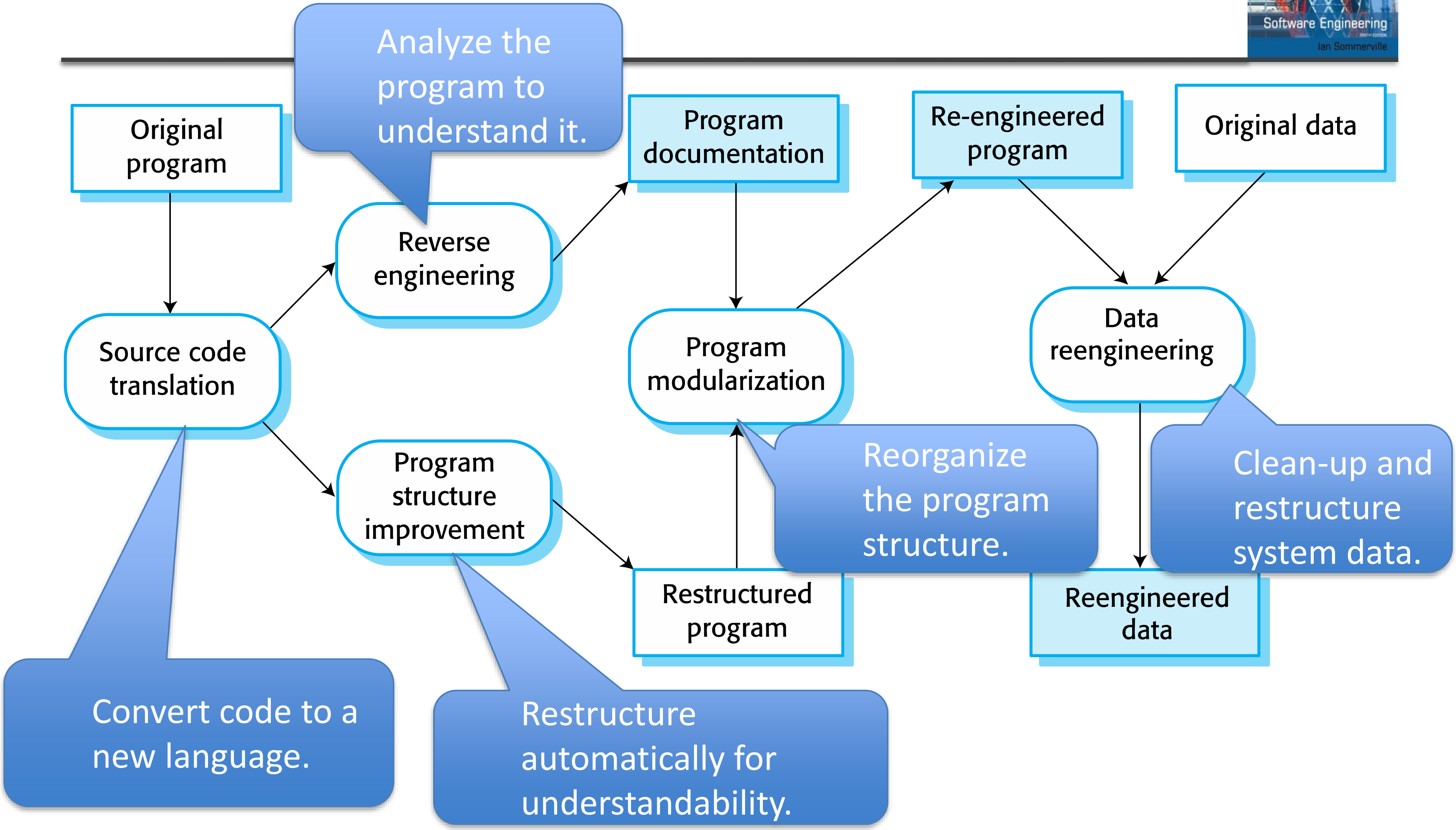
Software reengineering

- Restructuring or rewriting part or all of a legacy system without changing its functionality.
- Applicable where some but not all sub-systems of a larger system require frequent maintenance.
- Reengineering involves adding effort to make them easier to maintain. The system may be re-structured and re-documented.

Advantages of reengineering

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

The reengineering process



Analyze the program to understand it.

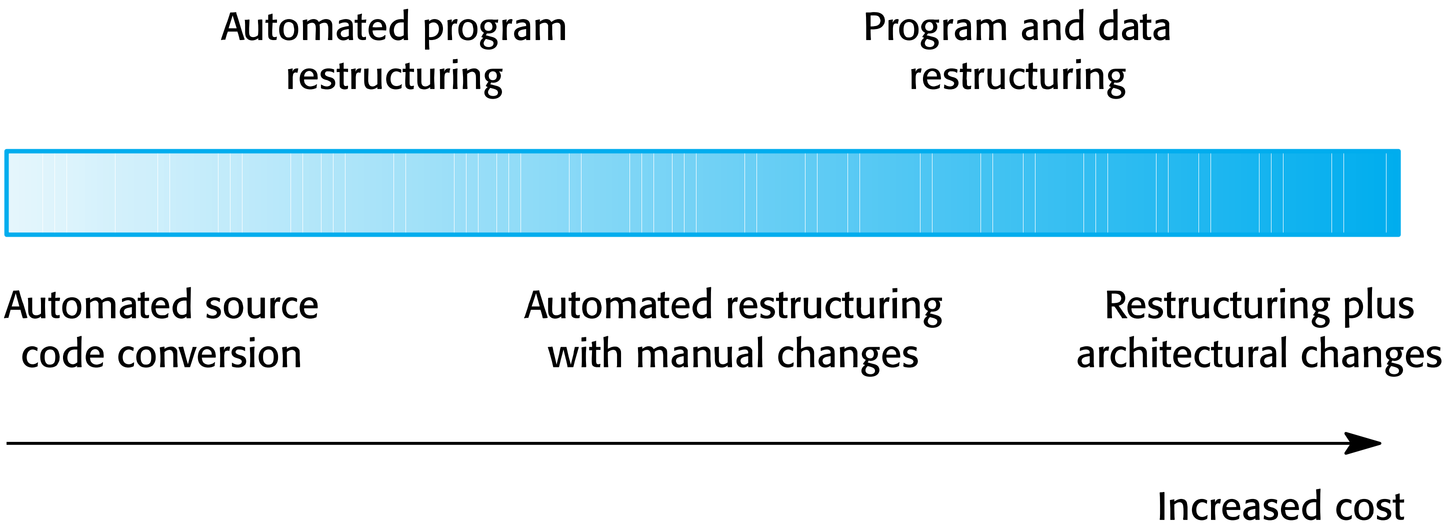
Reorganize the program structure.

Clean-up and restructure system data.

Convert code to a new language.

Restructure automatically for understandability.

Reengineering approaches & cost



Automated program restructuring

Program and data restructuring

Automated source code conversion

Automated restructuring with manual changes

Restructuring plus architectural changes

Increased cost

- 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.

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 refactoring 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.

‘Bad smells’ in program code

- Duplicate code-The same or very similar code can be removed and implemented as a single method or function.
- Long methods- If a method is too long, it should be redesigned as a number of shorter methods.
- Switch (case) statements -In object-oriented languages, you can often use polymorphism to achieve the same thing.
- Data clumping-Data clumps re-occur in several places, can often be replaced with an object that encapsulates all of the data.

Configuration Management

Configuration management

Configuration management (CM) is a systems engineering process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design, and operational information throughout its life:

I. Version management

II. System building

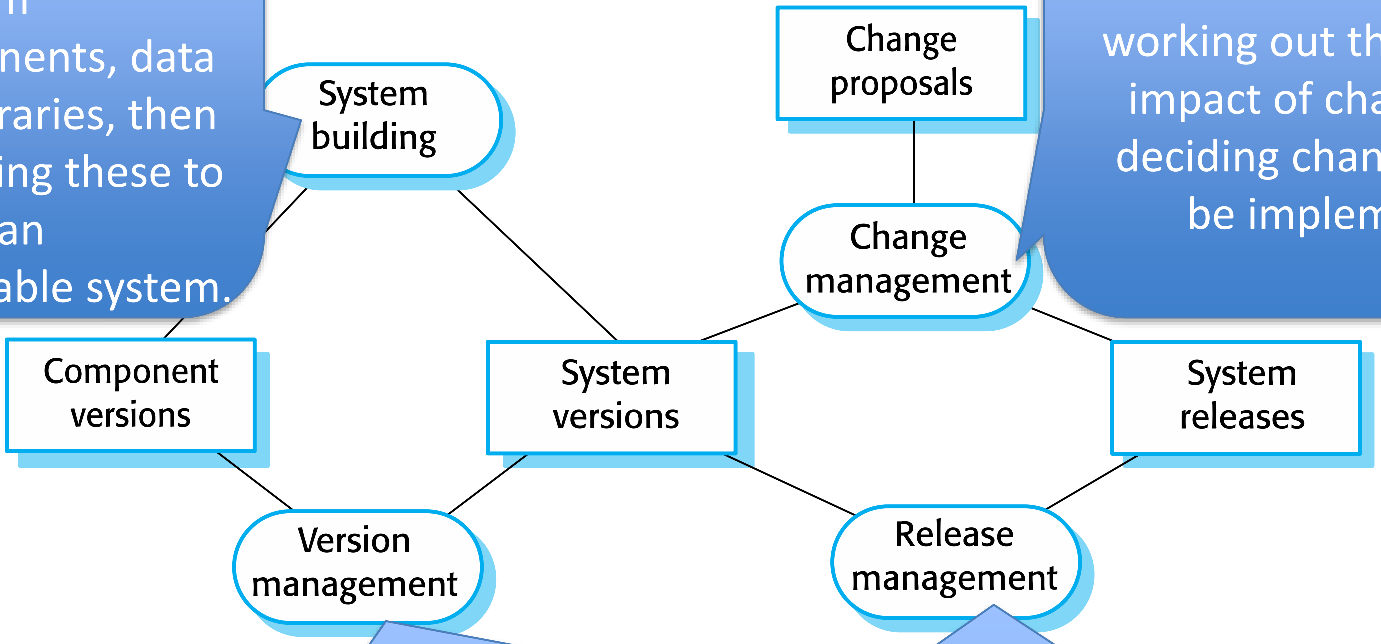
III. Change management

IV. Release management

Configuration management activities

The process of assembling program components, data and libraries, then compiling these to create an executable system.

Keeping track of requests for changes from customers and developers, working out the costs and impact of changes, and deciding changes should be implemented.

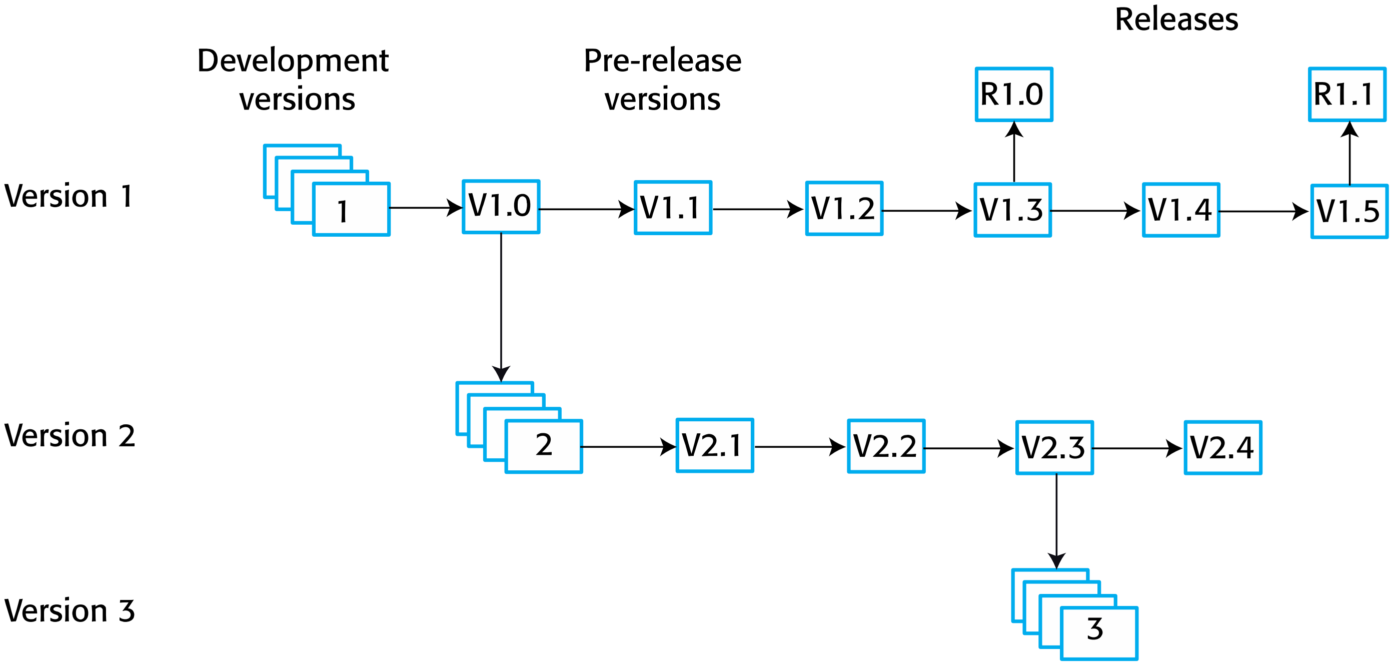


Keeping track of the multiple versions of system components and ensuring that changes made to components by different developers do not interfere with each other.

Preparing software for external release and keeping track of the system versions that have been released for customer use.

Multi-version system development

- For large systems, there are always several 'working' version of a system.
- There may be several teams involved in the development of different system versions.

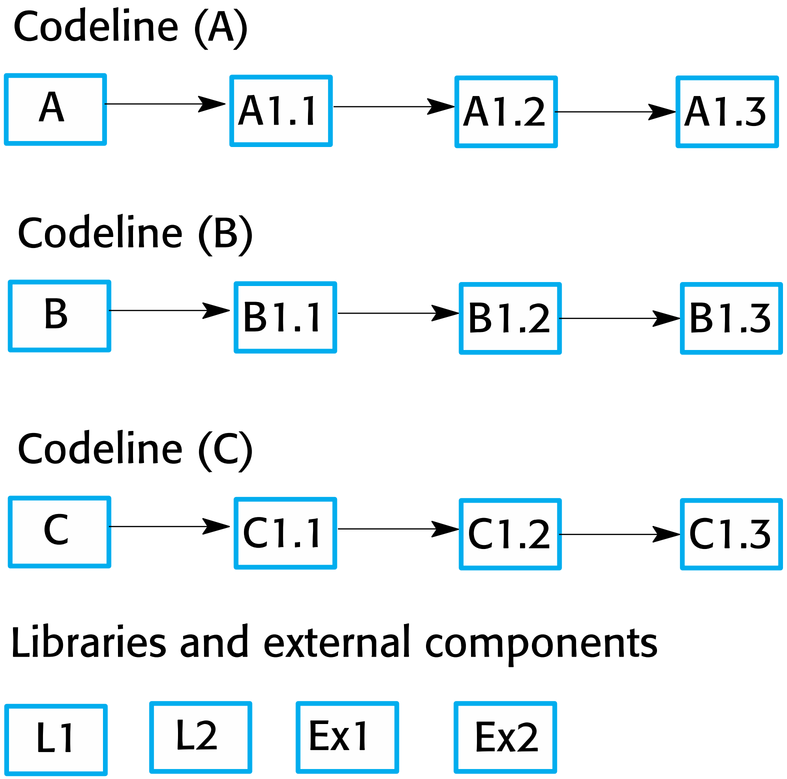


I. Version management

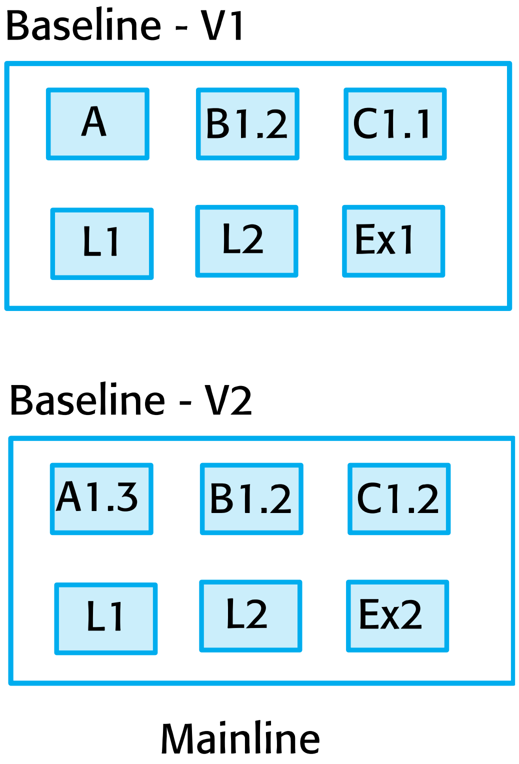
- Version management (VM) is the process of keeping track of different versions of software components or configuration items and the systems in which these components are used.
- It also involves ensuring that changes made by different developers to these versions do not interfere with each other.
- Therefore version management can be thought of as the process of managing codelines and baselines.

Codelines and baselines

Codeline is a sequence of versions of source code with later versions in the sequence derived from earlier versions.



Baseline specifies the component versions that are included in the system plus the libraries used, configuration files, etc.



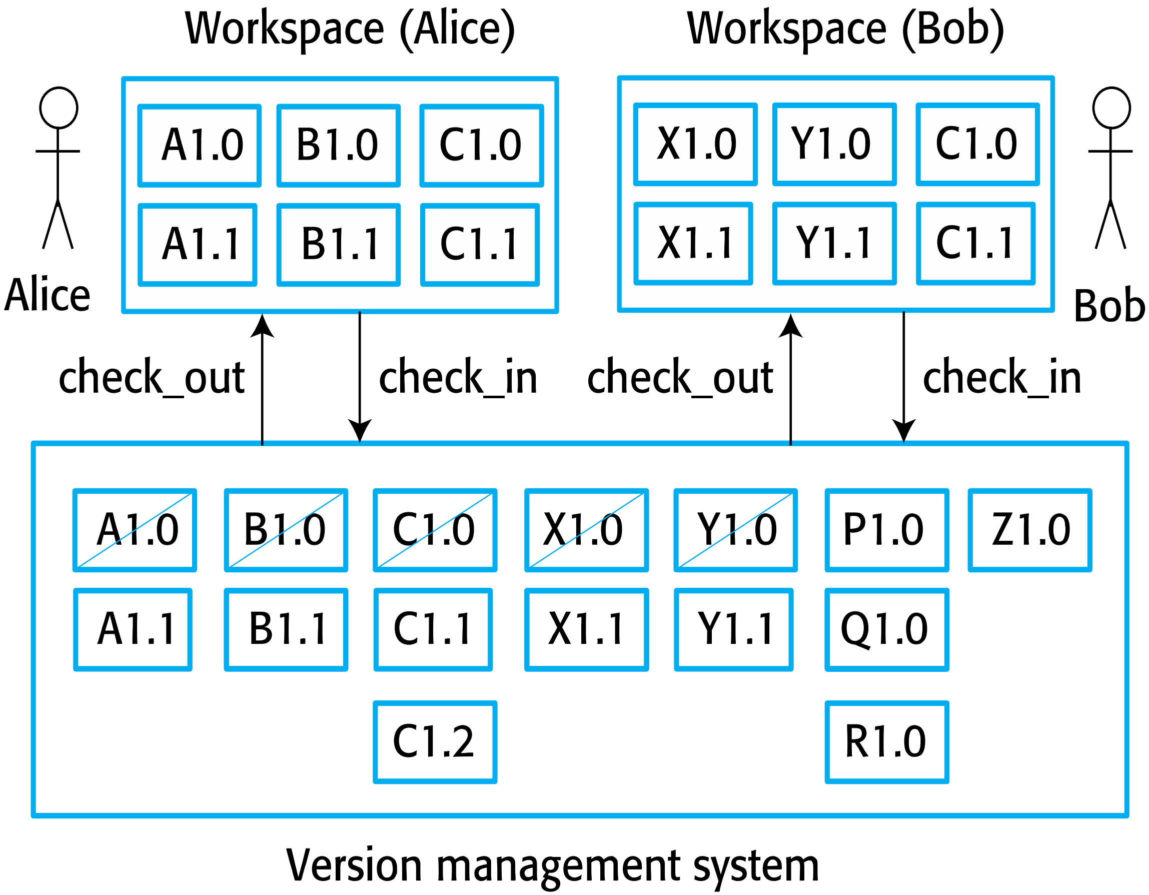
Version control systems

- Version control (VC) systems identify, store and control access to the different versions of components.
- Key features of version management systems：
  - Version and release identification
  - Change history recording
  - Support for independent development
  - Project support
  - Storage management
- There are two types of modern version management system：
  - Centralized systems
  - Distributed systems

Repository Check-in/Check-out（centralized）

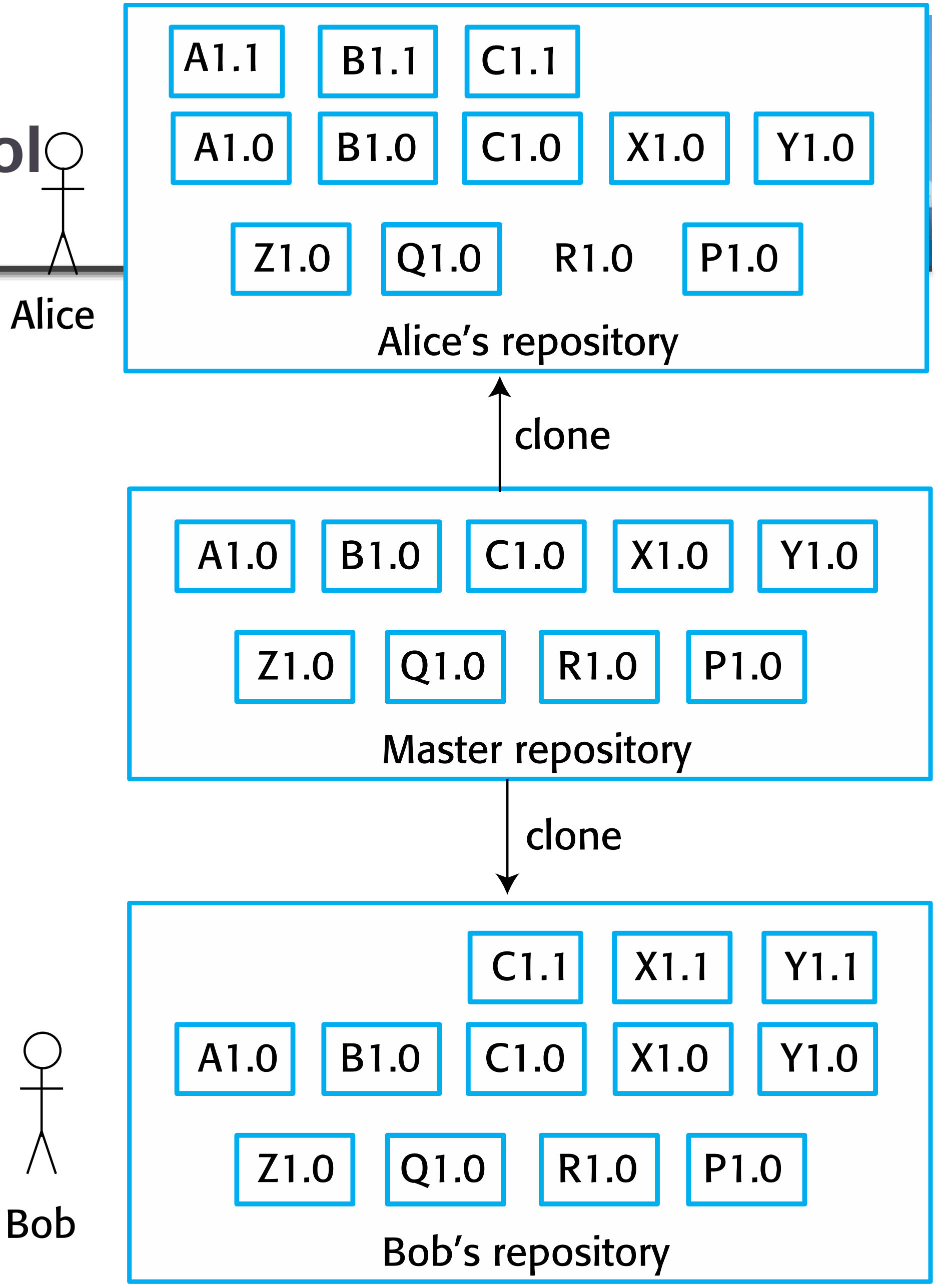
- Developers check out components into their private workspace and work on these copies in their private workspace, and check-in the components back to the repository after complete.

- The project repository maintains the ‘master’ version of all components.



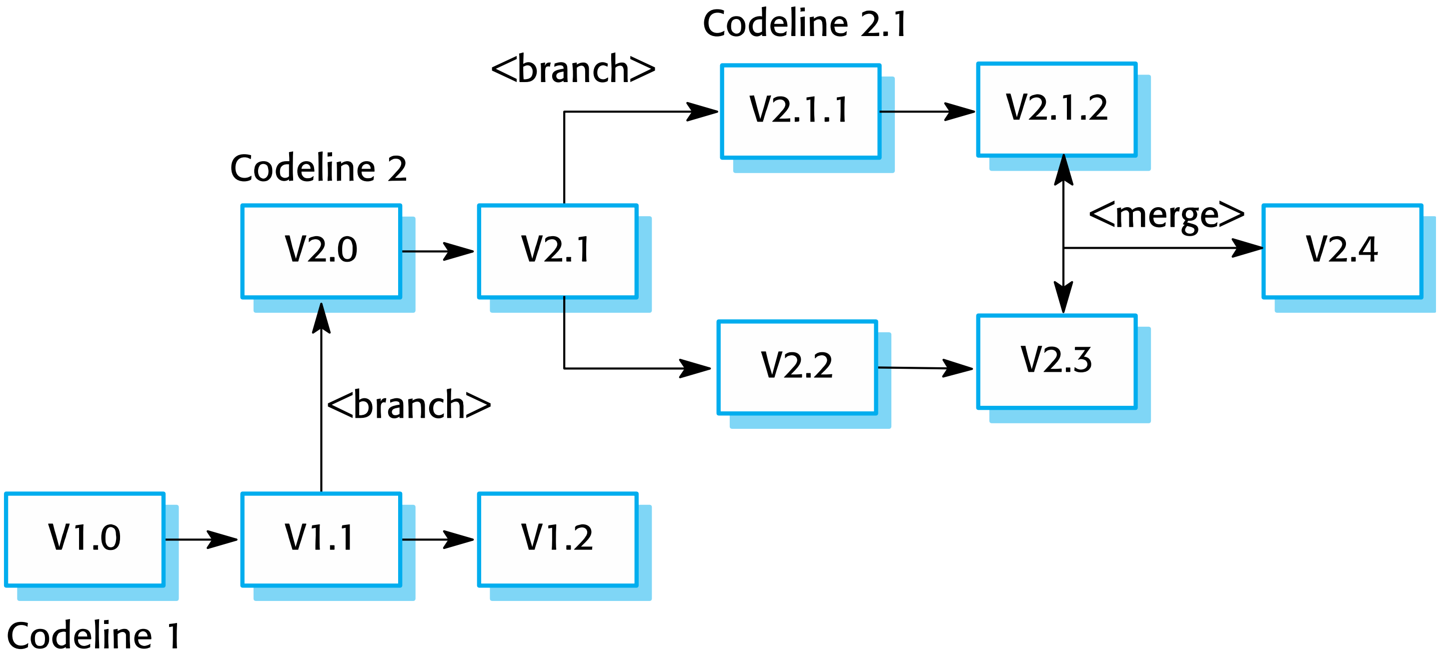
Distributed version control

- A ‘master’ repository is created on a server that maintains the code produced by the development team.
- Developers create a clone of the project repository and work on it.
- When changes are done, they ‘commit’ these changes and update their private server repository, and may ‘push’ these changes to the project repository.



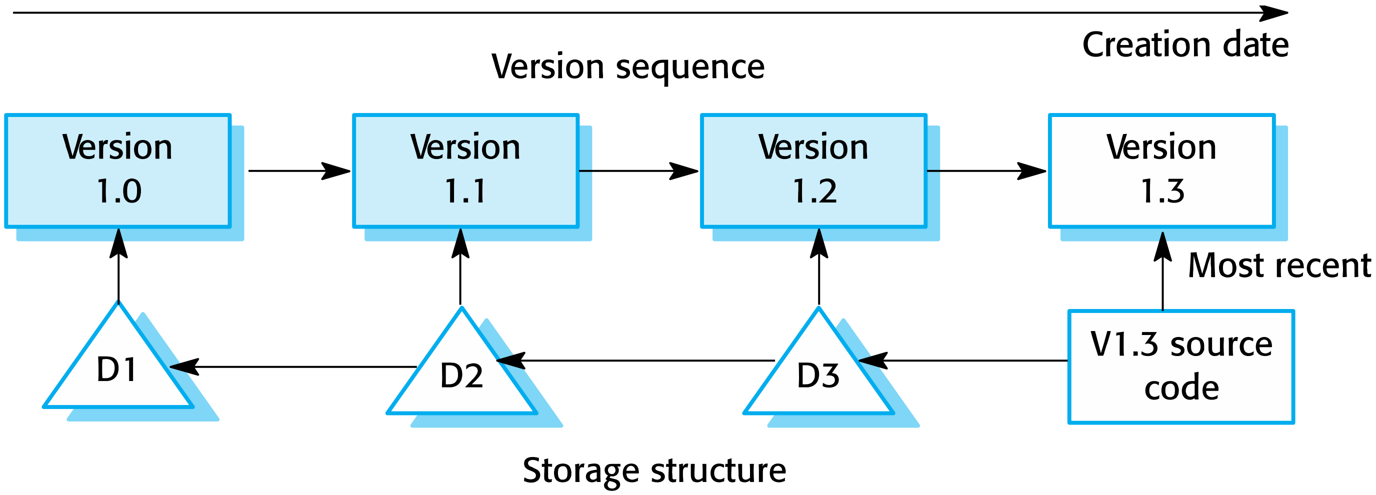
Branching and merging

✧ At some stage, it may be necessary to merge codeline branches to create a new version of a component.



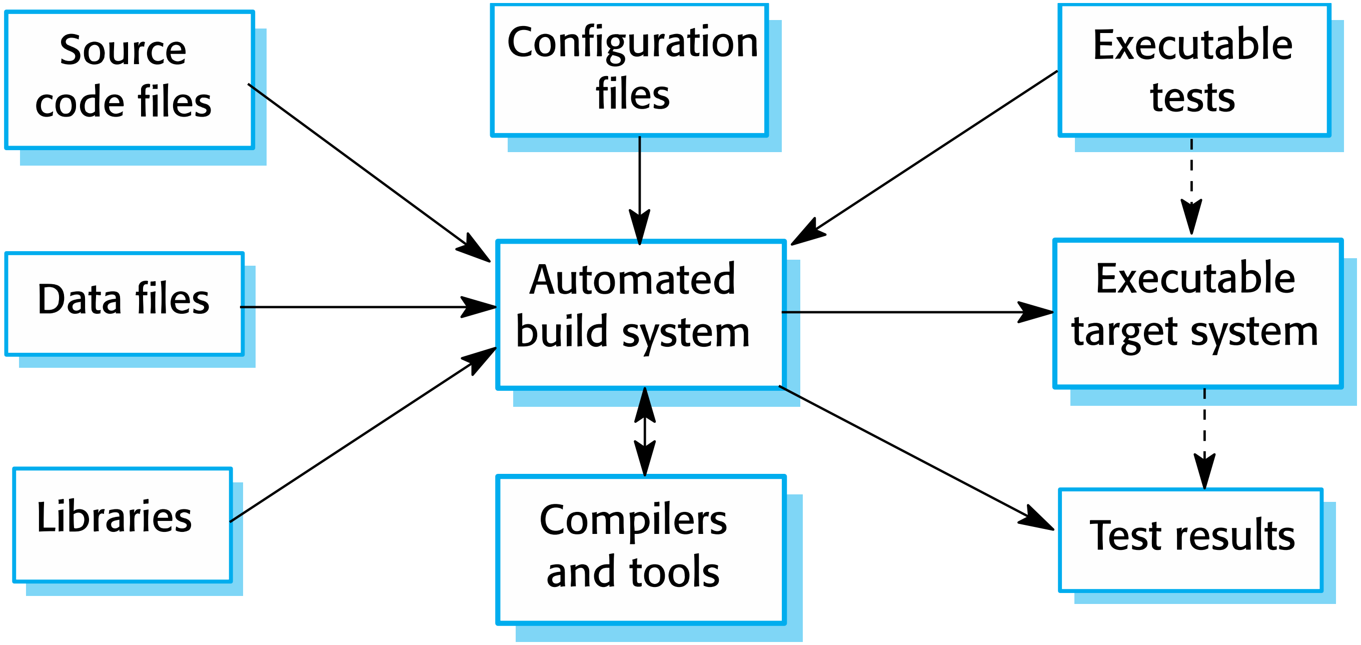
Storage management using deltas

- Instead of keeping a complete copy of each version, the system stores a list of differences (deltas) between one version and another.



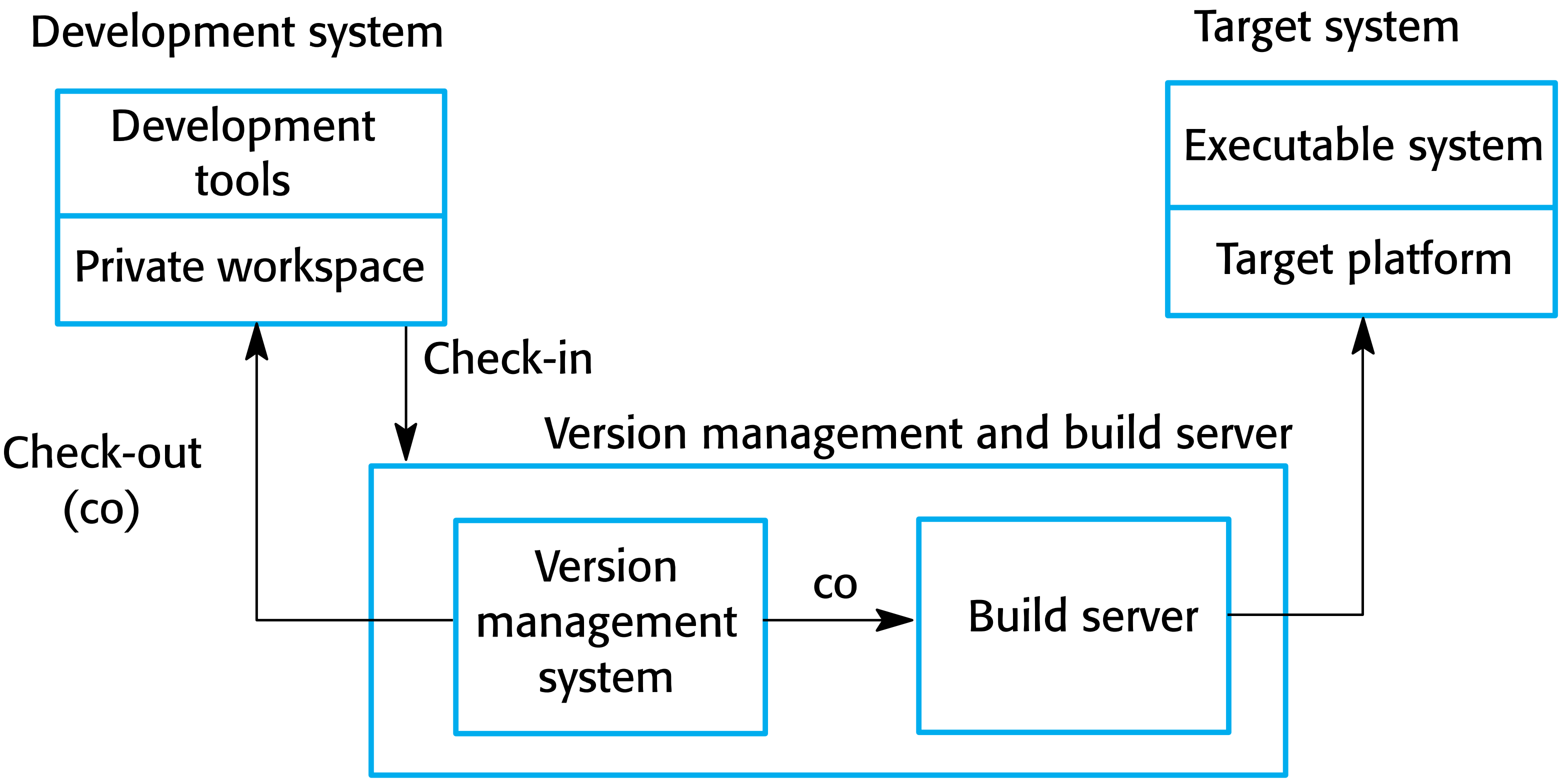
- Git does not use deltas but applies a standard compression algorithm.

II. System building

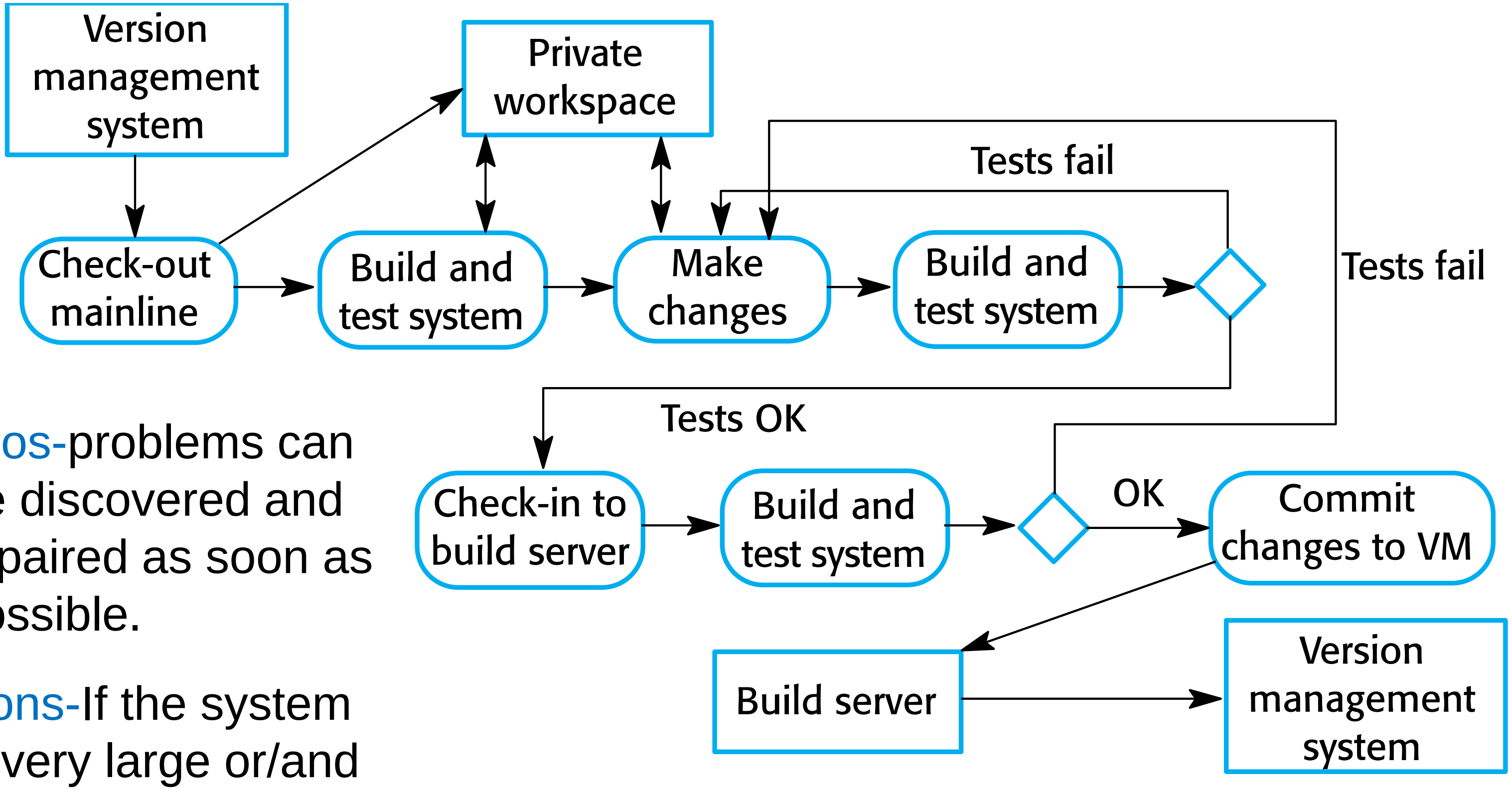


✧ System building is the process of creating a complete, executable system by compiling and linking the system components, external libraries, configuration files, etc.

Development, build, and target platforms



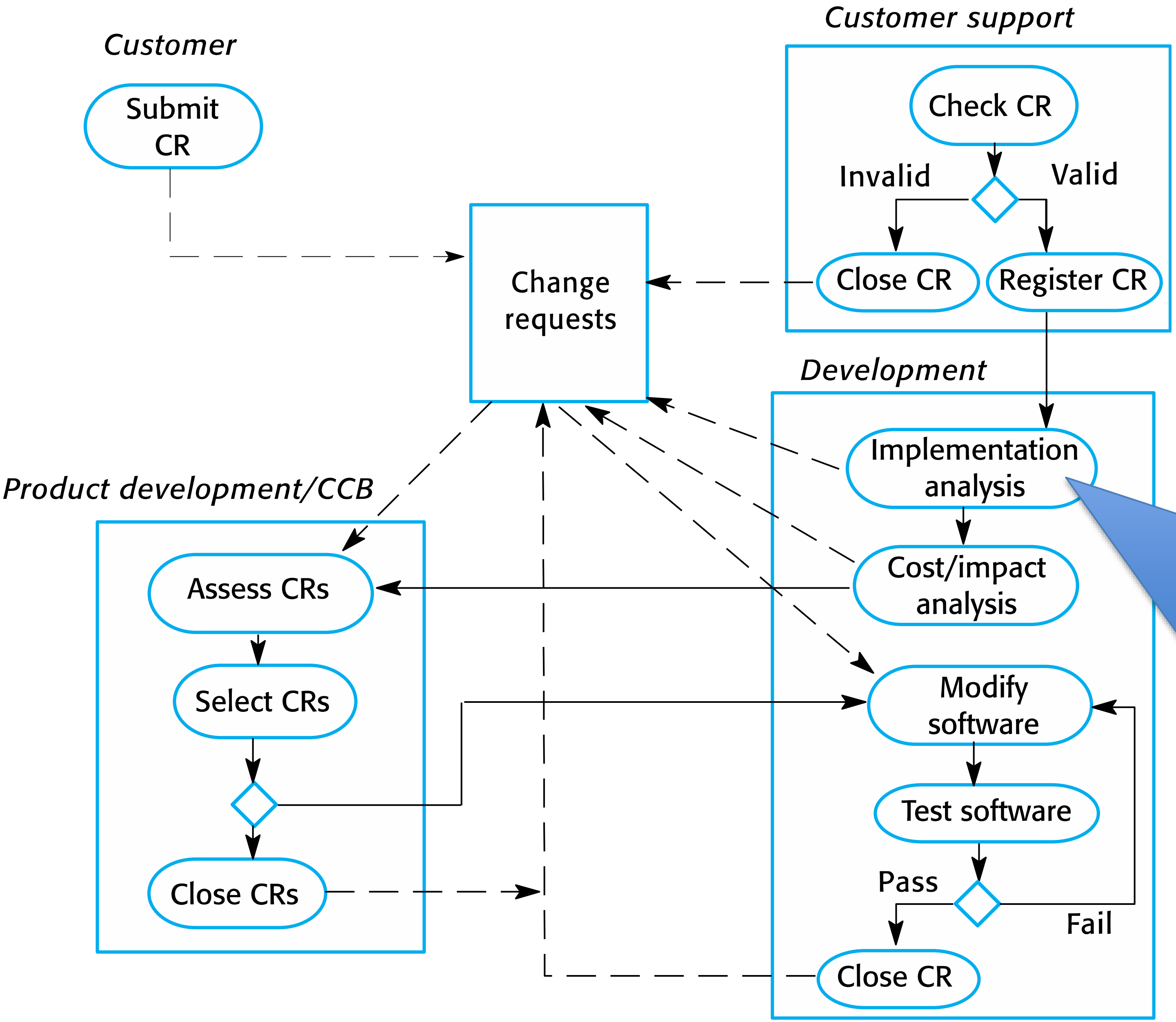
Agile Continuous integration



- Pros-problems can be discovered and repaired as soon as possible.

- Cons-If the system is very large or/and complex, it may take a long time to build and test.

III. Change management



- ☐ 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

A partially completed change request form (a)

Change Request Form

Project: SICSA/AppProcessing **Number:** 23/02
Change requester: I. Sommerville **Date:** 20/07/12
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:** 25/07/12
Components affected: ApplicantListDisplay, StatusUpdater

Associated components: StudentDatabase

A partially completed change request form (b)

Change Request Form

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 SGA app. team: 28/07/12 **CCB decision date:** 30/07/12
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:

Derivation history

```
// SICSA project (XEP 6087)
//
// APP-SYSTEM/AUTH/RBAC/USER_ROLE
//
// Object: currentRole
// Author: R. Looek
// Creation date: 13/11/2012
//
// © St Andrews University 2012
//
// Modification history
// Version Modifier   Date          Change        Reason
// 1.0     J. Jones   11/11/2009    Add header    Submitted to CM
// 1.1     R. Looek   13/11/2012    New field     Change req. R07/02
```

IV. 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 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.

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.
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.
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.

Release creation

- 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.
- All documents must be recorded in case the event of a problem, it may be necessary to reproduce exactly the software.

Software as a service

- ✧ Delivering software as a service (SaaS) reduces the problems of release management.
- ✧ It simplifies both release management and system installation for customers.
- ✧ The software developer is responsible for replacing the existing release of a system with a new release and this is made available to all customers at the same time.

lecture9 : Software Evolution & Configuration Management

- **Reading for this week:**
  - ' Enabling Agile Testing through Continuous Integration ' by Stolberg, S
  - Chapter 9: Software Evolution from the coursebook (Software Engineering by Ian Sommerville)
  - Chapter 25: Configuration Management from the coursebook (Software Engineering by Ian Sommerville)
- **Assignments for this week:**
  - Quiz 9: 2 attempts, 30 minutes time
  - Essay7: Software evolution and configuration management