Change Control

SWEN 256 – Software Process & Project Management

Software Change

- Software change is inevitable
  - New requirements emerge when the software is under development or being used
  - The business environment changes
  - Errors must be repaired, Risks mitigated
  - New equipment must be accommodated
  - The performance or reliability may have to be improved
- A key problem for organisations is implementing and managing change to their current projects and legacy systems

Change During Development

- Sometimes change occurs during development that necessitates changes in scope
  - Approval of CCB (Change Control Board) and
  - Requires extensive planning
  - May require more time/resources (project triangle)
- Plan-driven methodologies may or may not have this built in (i.e. Spiral) or may be specifically built to resist change (i.e. Waterfall)
- Agile Methodologies embrace change
  - Scrum allows for change to the Product Backlog at any time, but manages risk by freezing the current Sprint Backlog
- Stakeholder Communication IS KEY

Software Change Strategies

- Software maintenance
  - Changes are made in response to changed requirements but the fundamental software **structure is stable**
- Architectural transformation
  - The architecture of the system is modified generally from a **centralised** architecture to a **distributed** architecture
- Software re-engineering
  - No new functionality is added to the system but it is restructured and **reorganised** to facilitate future changes
- These strategies may be applied separately or together

Lehman's Laws

Law	**Description**
Continuing change	A program that is used in a real-world environment necessarily must change or become progressively less useful in that environment.
Increasing complexity	As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplifying the structure.
Large program evolution	Program evolution is a self-regulating process. System attributes such as size, time between releases and the number of reported errors are approximately invariant for each system release.
Organisational stability	Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.
Conservation of familiarity	Over the lifetime of a system, the incremental change in each release is approximately constant.

Applicability of Lehman's Laws

- This has not yet been established
- They are **generally applicable** to large, tailored systems developed by large organisations
- It is not clear how they should be modified for
  - Shrink-wrapped software products
  - Systems that incorporate a significant number of COTS components
  - Small organisations
  - Medium sized systems

Software Maintenance

Software Maintenance

- Modifying a program after it has been put into use
- Maintenance does not normally involve major changes to the system's architecture
- Changes are implemented by modifying existing components and adding new components to the system

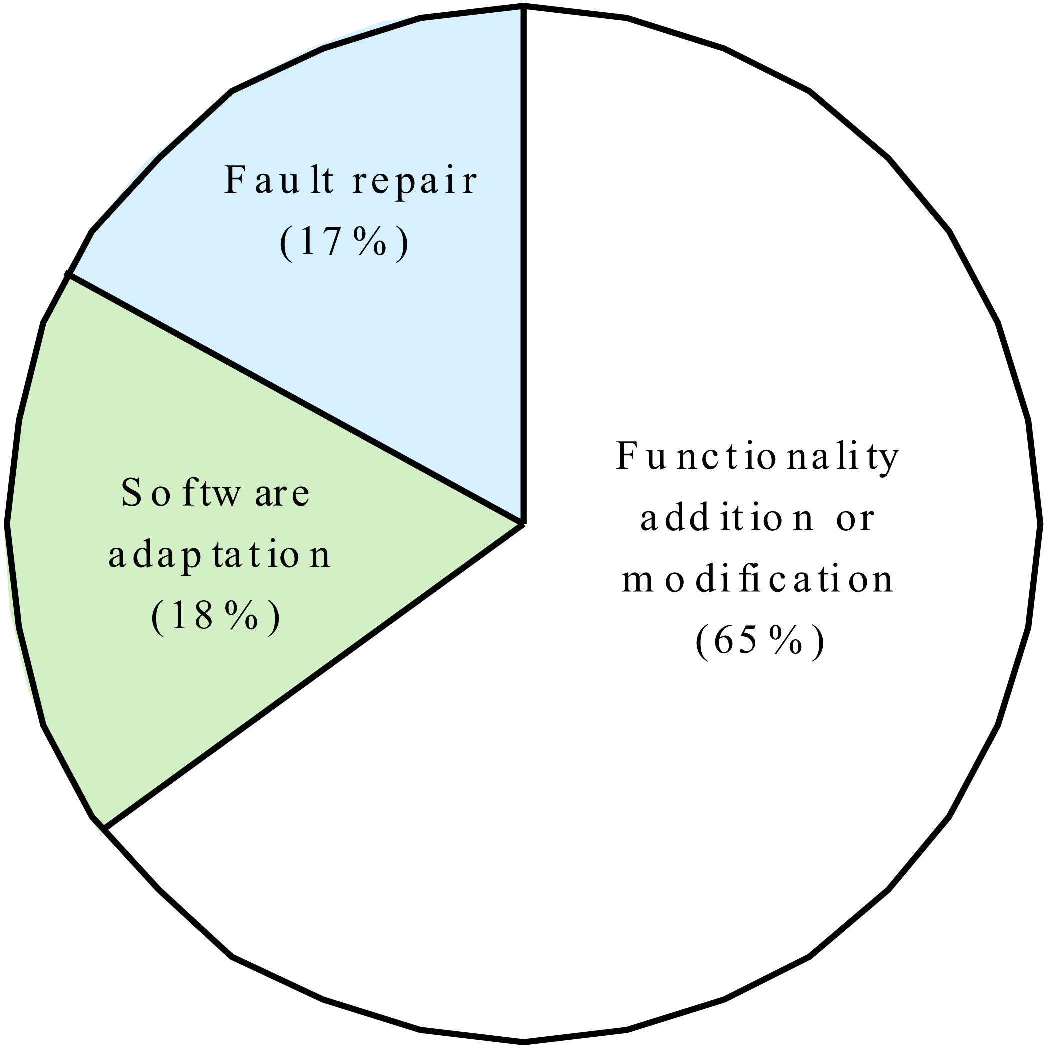
Maintenance is Inevitable

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

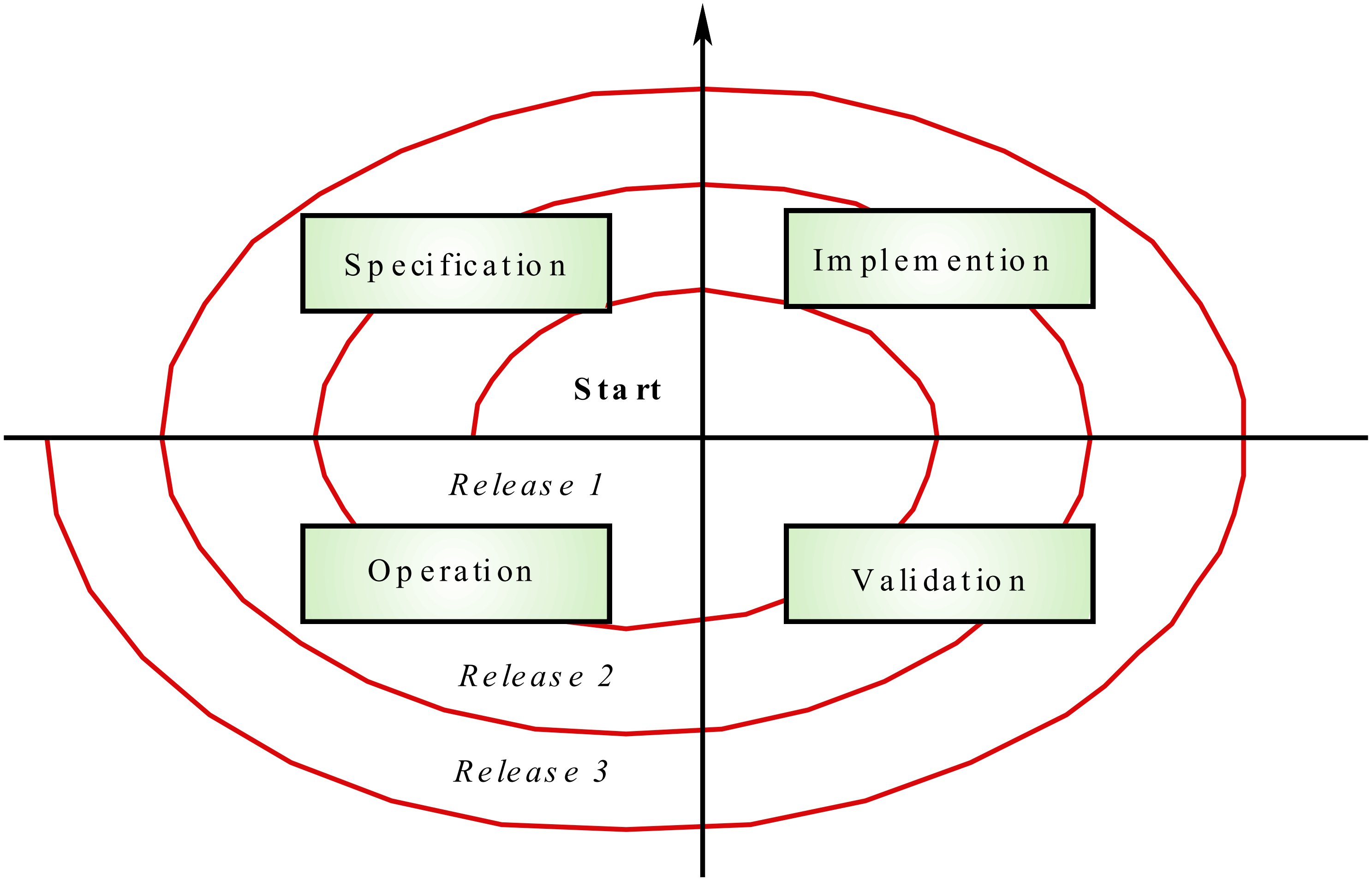
Types of Maintenance

- Maintenance to **repair** software faults
  - Changing a system to correct deficiencies in the way meets its requirements (**Corrective** Maintenance)
- 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 (**Adaptive** Maintenance)
- Maintenance to **add** to or **modify** the system's functionality
  - Modifying the system to satisfy new requirements (**Perfective** Maintenance)

Distribution of Maintenance Effort



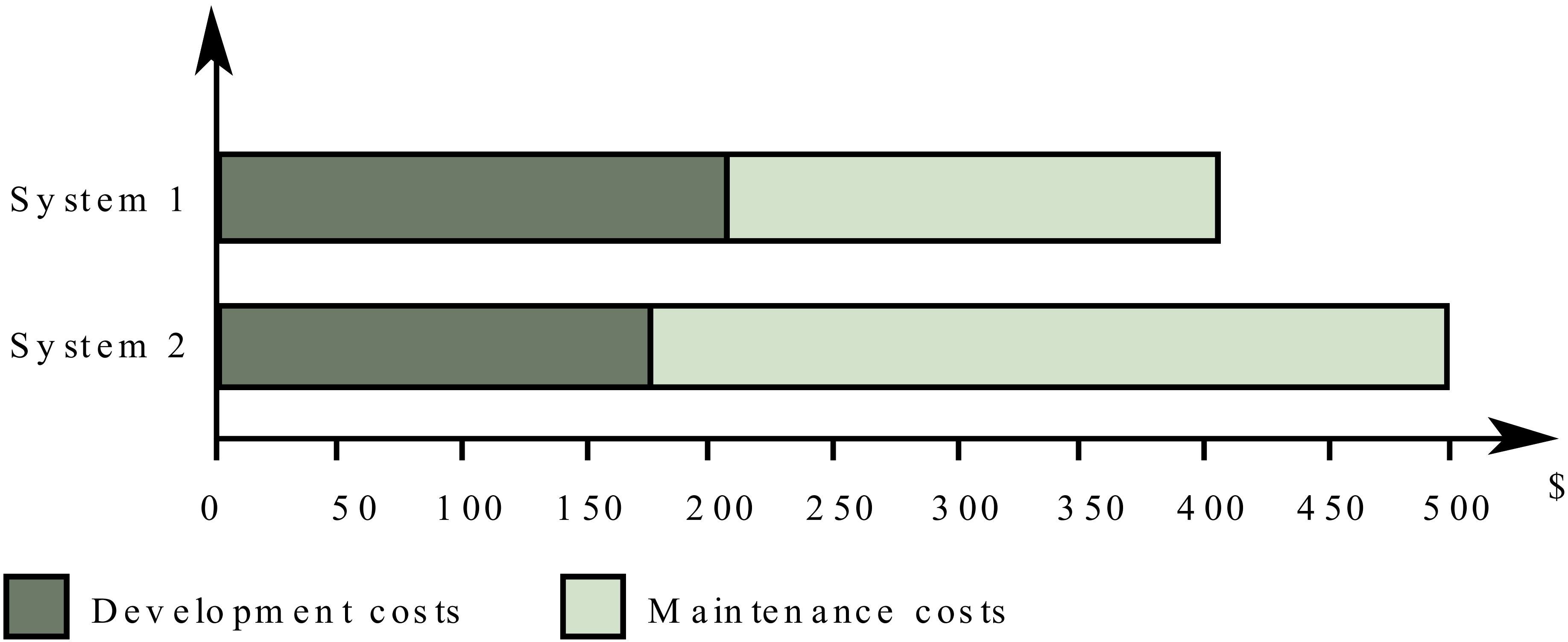
Spiral Maintenance Model



Maintenance Costs

- Usually greater than development costs (2* to 100* depending on the application)
- Affected by both technical and non-technical factors
- Increases as software is maintained. **Maintenance corrupts** the software structure so makes further maintenance more difficult.
- Ageing software can have high support costs (e.g. old languages, compilers etc.)

Development/Maintenance Costs



Maintenance Co$t 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

Evolutionary Software

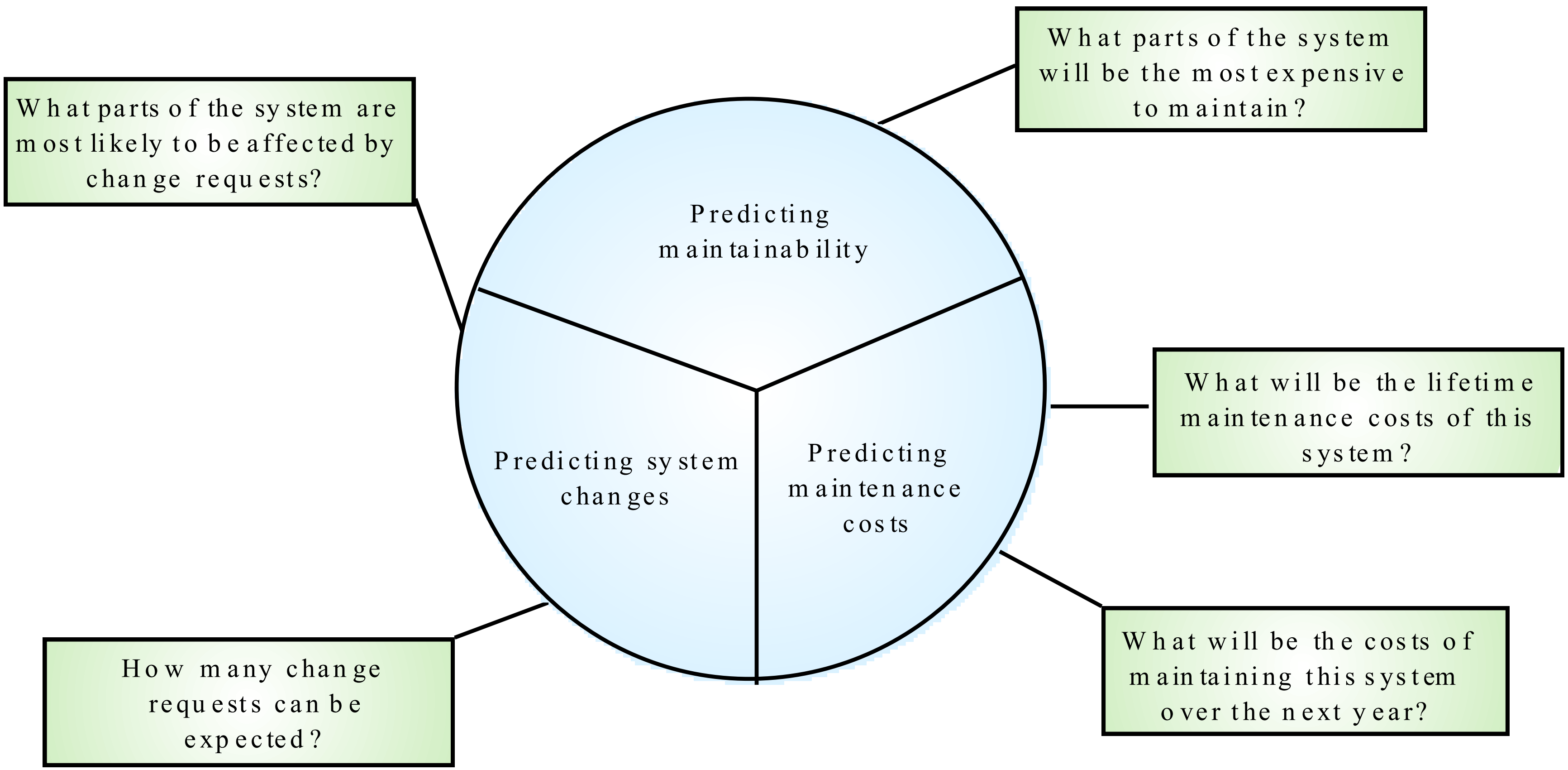
- Rather than think of separate development and maintenance phases, evolutionary software is software that is **designed** so that it can continuously **evolve** throughout its lifetime

YES, but **how/much**?

Maintenance Prediction

- Maintenance prediction is concerned with assessing **which parts** of the system may cause problems and have high maintenance costs
  - **Change acceptance** depends on the maintainability of the components affected by the change
  - Implementing **changes degrades** the system and reduces its maintainability
  - Maintenance costs depend on the **number of changes <u>and</u> costs of change** depend on maintainability

Maintenance Prediction



Change Prediction

- Predicting the number of changes requires an understanding of the **relationships** between a system and its environment
- Tightly **coupled** systems require changes whenever the environment is changed
- Factors influencing this relationship are
  - Number and *complexity* of system **interfaces**
  - Number of inherently *volatile* system **requirements**
  - The **business processes** where the system is used

Complexity Metrics

- Predictions of maintainability can be made by assessing the complexity of system components
- Studies have shown that most **maintenance** effort is spent on a relatively **small number** of system components
- Complexity depends on
  - Complexity of **control** structures
  - Complexity of **data** structures
  - Procedure and module **size**

Process Metrics

- Process measurements may be used to assess maintainability
  - Number of requests for corrective maintenance
  - Average time required for impact analysis
  - Average time taken to implement a change request
  - Number of outstanding change requests
- If any or all of these is increasing, this may indicate a **decline in maintainability**

Architectural Evolution

- There is a need to convert many legacy systems from a centralised architecture to a client-server architecture
- Change drivers
  - **Hardware costs.** Servers are cheaper than mainframes
  - User interface expectations. Users expect graphical user interfaces (**CLI→GUI**)
  - **Distributed** access to systems. Users wish to access the system from different, geographically separated, computers

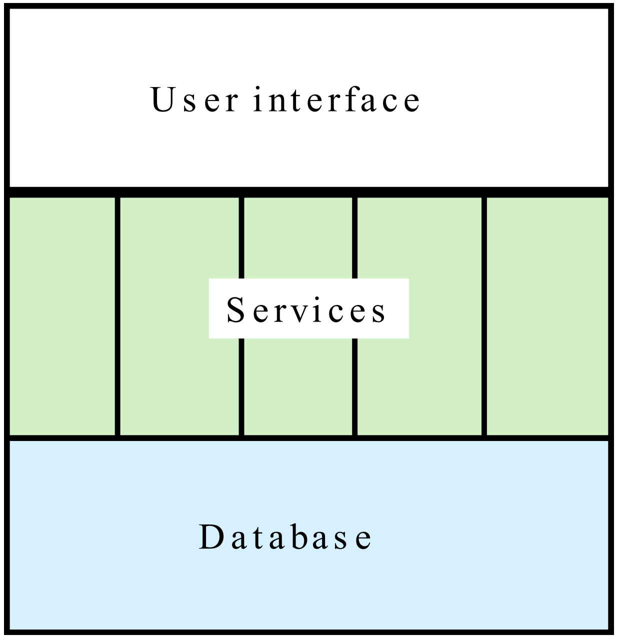
Distribution Factors

Factor	Description
Business importance	Returns on the investment of distributing a legacy system depend on its importance to the business and how long it will remain important. If distribution provides more efficient support for **stable business processes** then it is more likely to be a cost-effective evolution strategy.
System age	The older the system the more difficult it will be to modify its architecture because previous changes will have degraded the structure of the system.
System structure	The more modular the system, the easier it will be to change the architecture. If the application logic, the data management and the user interface of the system are closely intertwined, it will be difficult to separate functions for migration.
Hardware procurement policies	Application distribution may be necessary if there is company policy to replace expensive mainframe computers with cheaper servers. .

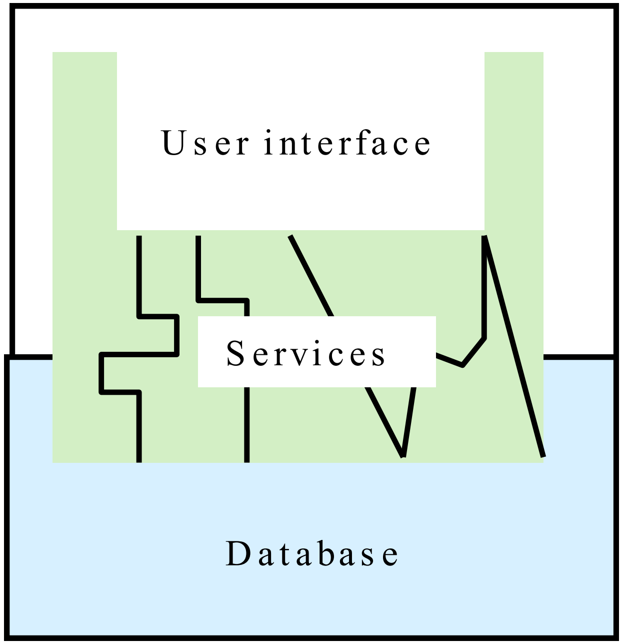
Legacy System Structure

- Ideally, for distribution, there should be a clear separation between the user interface, the system services and the system data management
- In practice, these are usually intermingled in older legacy systems

Legacy System Structures



Ideal model for distribution



Real legacy systems

UI Migration Strategies

Strategy	Advantages	Disadvantages
Implementation using the window management system	Access to all UI functions so no real restrictions on UI design Better UI performance	Platform dependent May be more difficult to achieve interface consistency
Implementation using a web browser	Platform independent Lower training costs due to user familiarity with the WWW Easier to achieve interface consistency	Potentially poorer UI performance Interface design is constrained by the facilities provided by web browsers

Key Points

- The costs of software change usually exceed the costs of software development
- Factors influencing maintenance costs include staff stability, the nature of the development contract, skill shortages and degraded system structure
- Architectural evolution is concerned with evolving centralised to distributed architectures
- A distributed user interface can be supported using screen management middleware