UNIT -5
Product Release and Software Maintenance

- Product Release Management
- Product Implementation
- User Training
- Software Maintenance
- Software Maintenance Types
- Software Maintenance Cost
- Software Maintenance Process
- Software Maintenance Life Cycle
- Software Maintenance Techniques
- Software Release

Product Release

- The software product which you have been building for so long is now complete.
- You need to take it to the customer's site and get it implemented so that the end users can start using it.
- However, do not run fast in anticipation of wrapping things as early as possible. After all this is your magnum opus and you need to be careful.
- You need to make sure that all your tasks are completed
- In software engineering, a release is a new or modified software and the process of its creation.
- A release constitutes a fully functional version of the software, and it is the climax of the software development and engineering processes.
- Alpha and beta versions of the software typically precede its release.
- Releases may also referred to as launches or increments.

- A product release is the process of launching a new product or feature set into the market.
- Releases can happen frequently (daily or weekly) or just occasionally (quarterly or annually).
- Your release rhythm will depend on a few factors, including where your product business is in its market lifecycle and your available resources.
- There are many reasons for releasing product. Here’s just a few:
- You’re releasing an existing product onto a new platform (such as Android)
- You’re releasing a new idea into market for the first time
- You’re releasing an existing product onto a new platform (such as Android)
- You’re introducing new features to current customers
- You’re introducing new features to new or different customer segments
- You’re re-releasing your current product using different technology (often called “re-platforming”)

Release Planning

A release planning session is a descendent of the product roadmap. That means that the company should first decide what initiatives it wants to deliver on (and when) and then set aside time to plan each (major) product release.

Start your release planning session by revisiting or discussing why the release matters.

What do you hope will happen for the business once the product has been released?

By when?

How will you measure that success?

Take care to assess team alignment at this stage. Is everybody in agreement about why we're doing this?

The release planning session is an opportunity to get buy-in from your team. This is especially important given that most of the attendees will lead the efforts of actually building the product.

Product Release Management

Release management process may likely mean different things to a product management team and a development team.

Both perspectives are strongly incorporated into a dependable release management process.

A release management process incorporates all of the following:

1)Planning

An initial release plan takes into account the team's velocity on the previous release (or general capacity to deliver) and the feature prioritization to create a general scope, sequencing, and timeline for the release.

During release planning, a general expectation on the number of sprints or iterations to deliver the scope is achieved.

The accuracy of this expectation and plan depends on whether the team's capacity is well-known as well as the level of detail (or grooming) the scope has been through during estimation.

This general plan will also provide expectations of major product changes (or dependencies) for products that may depend on your roadmap or platform.

Plans will be revisited after each iteration. For this reason, a tracking of an external release target (with quarterly, monthly, or other precision) can be helpful.

It will still set the expectation and trust with your customers, but can be refined by you as the plan progresses.

2) Phased communication and supporting team engagement

Product managers know best how to deliver their product successfully — and that includes a series of non-development tasks of engagement with supporting teams to complete items like documentation, sales training, or marketing campaigns.

As a product manager, establishing a launch or release template will enable you to create a "gold standard" for major delivery.

Use this template to engage your greater team, who may be supporting multiple products in the portfolio only when needed.

A standard for launch also sets expectation for when these teams will be needed internally.

3) Repeatable stages to readiness

Establish standardized status at both the release and feature level to indicate overall health of the plan.

Status provides an "Are we good?" pulse point that can be key to seeing around corners and proactive risk mitigation.

A release status will enable communication to your internal stakeholders, while feature status workflows enable granular visibility into the readiness of the feature and its current status with respect to development, staging, or QA environments

4)Forward adjustment on plan

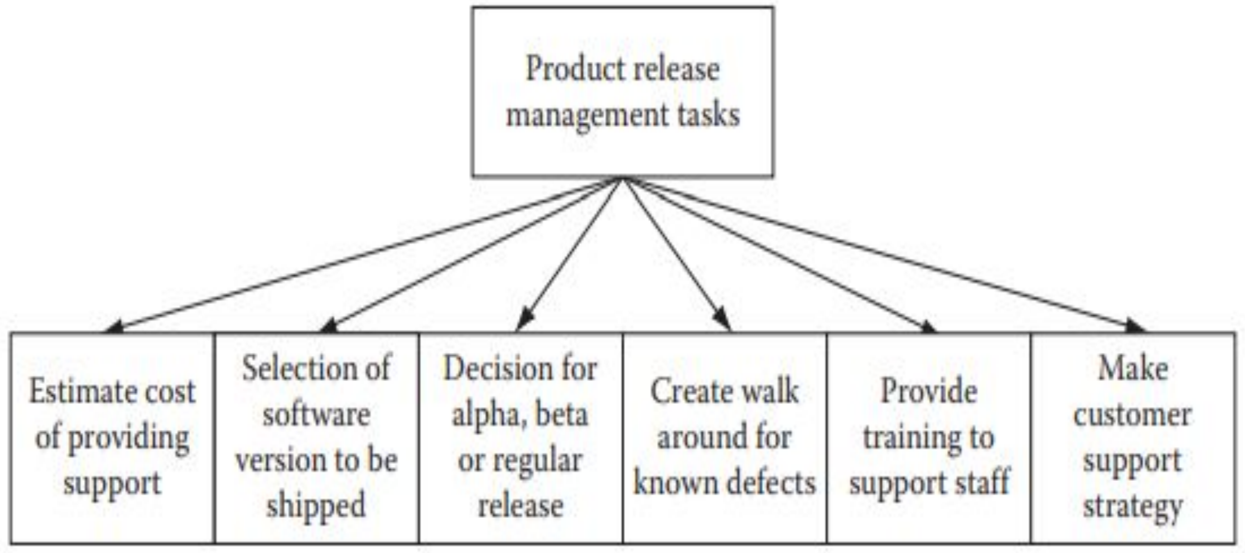
Regardless of whether your release plan is executed in sprints or via more waterfall methodologies, regular check-ins and adjustments to plan are necessary.

Use your sprint closure to adjust plans as needed, or schedule regular reviews to ensure plans are on track.

Product Release Management

- Project teams working for software product vendors struggle to keep pace with release of the software product.
- There is pressure from the market to launch new versions by certain dates.
- New features are to be added, porting the product to new platforms, old features are to be enhanced, existing bugs are to be removed, and yet it has to meet the deadline.
- It is a constant struggle that calls for good product release strategies.
- Depending on the situation, the project manager may need to convince the management to cut short some of the product features to meet the deadline as well as meet quality standards.
- A half-baked product will never have any takers; instead the project manager may be blamed for its poor quality issues.
- Bargaining also has to be done for other requirements of bug fixes, feature enhancements, etc. If quality concerns are paramount, then moving some of the tasks of new features to a future release may be the best solution for meeting quality standards.
- If the software vendor is not too sure about product quality, then he may opt for an alpha or beta release of the product.
- In that case, the product will be released only among a few selected groups and not in the market as a whole. The controlled product release is the best option in these conditions

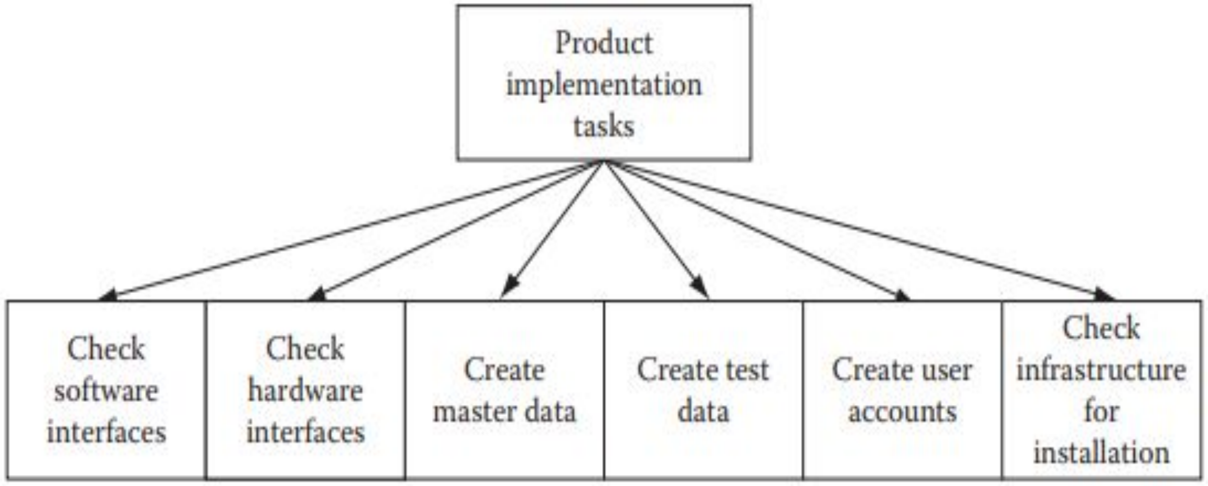
- Product release management is such a dynamic environment that if proper planning is not done at a minute level and constant vigilance is not applied over project activities, then a huge mess can be created and there will be no time to clear it.
- So the project manager must be vigilant all the time
- Walk around for known issues, estimated number of critical bugs still remaining in the product, training for the support staff, etc., should be done.
- The cost of support, depending on the number of estimated users, walk around, and remaining bugs should be figured out.
- These measures will ensure that the product is transitioned into market without facing major difficulties.


Product release management tasks
Estimate cost of providing support
Selection of software version to be shipped
Decision for alpha, beta or regular release
Create walk around for known defects
Provide training to support staff
Make customer support strategy

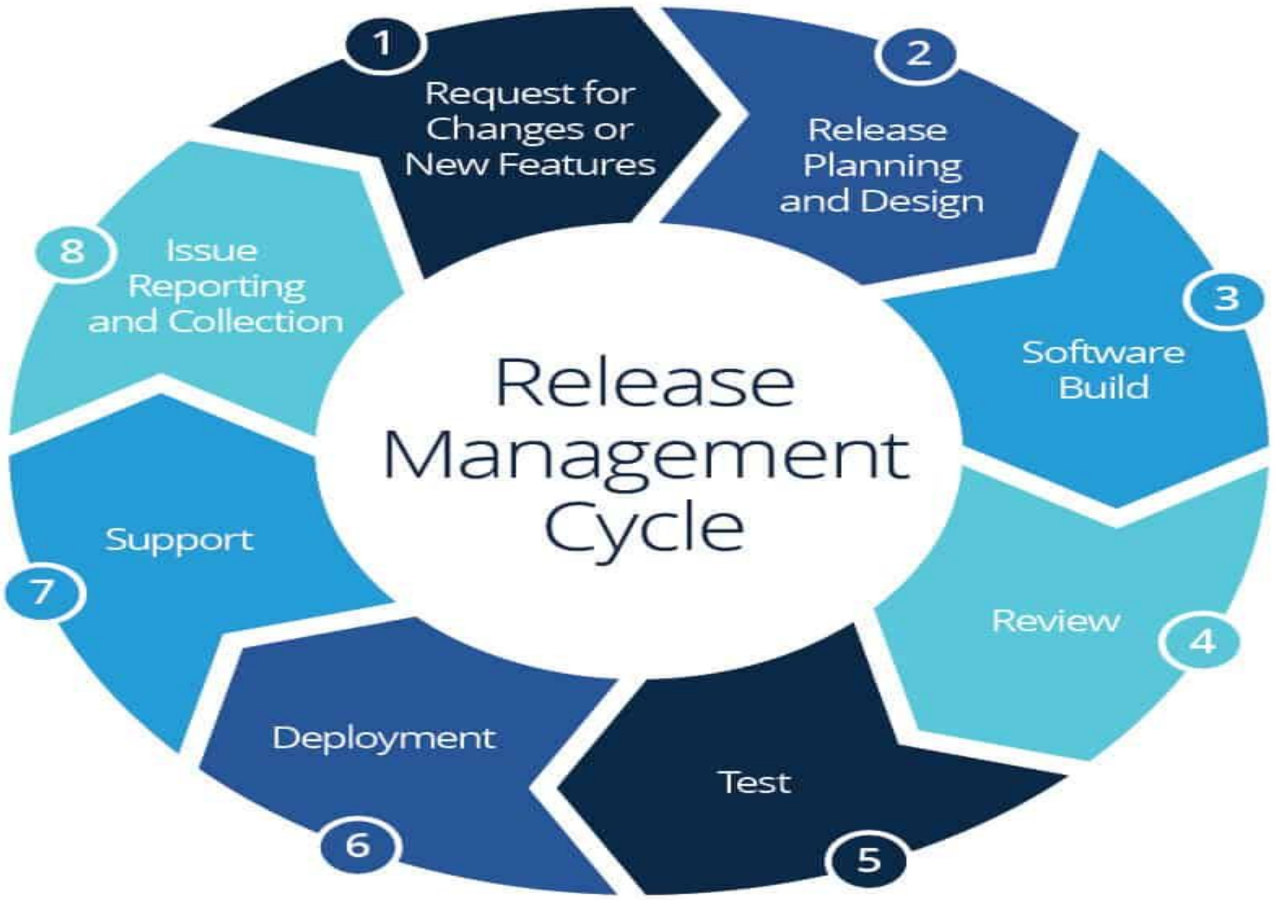
Product Implementation

- The product that has been developed and thoroughly tested now needs to be implemented at a customer site.
- You need to prepare all master data and test transaction data for testing the implemented product.
- You need to get all required hardware and software that need to be there for installing your software product.
- You need to make sure that you have developed and tested all the hardware and software interfaces for integrating your product, with existing legacy systems and infrastructure.
- You also need to make sure that your product will run smoothly on customer premises without any interference with their existing applications.
- Often project teams run into problems during implementation, due to unforeseen circumstances or negligence on part of the production team or customer's team.
- Therefore, prepare a list of your own requirements and hand it over to your customer's support team so that they are prepared when you arrive for implementation.

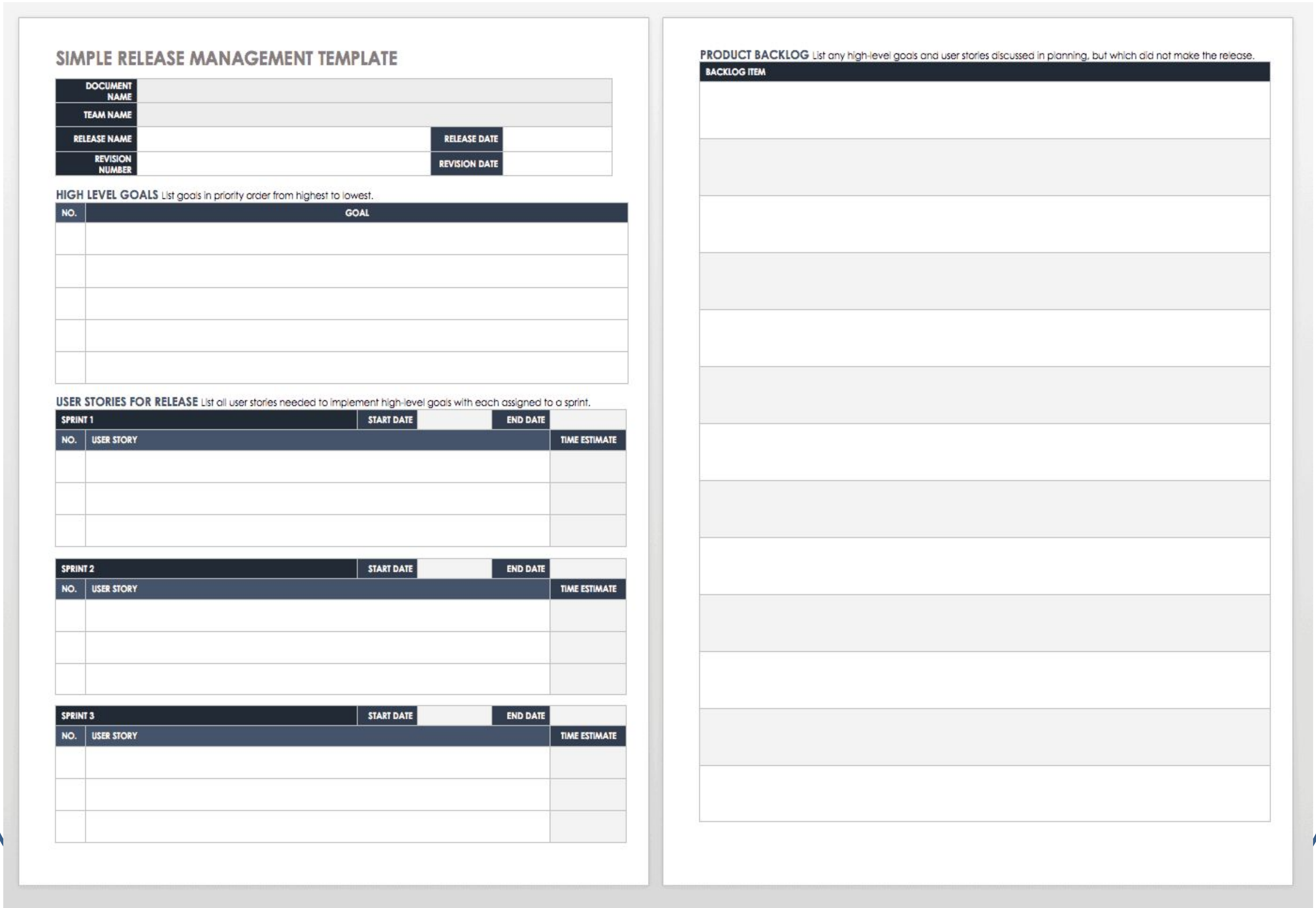
- A product software implementation method is a blueprint to get users and/or organizations running with a specific software product.
- The method is a set of rules and views to cope with the most common issues that occur when implementing a software product: business alignment from the organizational view and acceptance from human view.
- The implementation of product software, as the final link in the deployment chain of software production, is in a financial perspective of a major issue.
- It is stated that the implementation of (product) software consumes up to 1/3 of the budget of a software purchase (more than hardware and software requirements together).


Product implementation tasks
Check software interfaces
Check hardware interfaces
Create master data
Create test data
Create user accounts
Check infrastructure for installation

Release Management cycle


Release Management Cycle
1
Request for Changes or New Features
2
Release Planning and Design
3
Software Build
4
Review
5
Test
6
Deployment
7
Support
8
Issue Reporting and Collection

Simple Release Management Template



SIMPLE RELEASE MANAGEMENT TEMPLATE

DOCUMENT NAME			
TEAM NAME			
RELEASE NAME		RELEASE DATE	
REVISION NUMBER		REVISION DATE	

HIGH LEVEL GOALS List goals in priority order from highest to lowest.

NO.	GOAL

USER STORIES FOR RELEASE List all user stories needed to implement high-level goals with each assigned to a sprint.

SPRINT 1	START DATE		END DATE	

NO.	USER STORY	TIME ESTIMATE

SPRINT 2	START DATE		END DATE	

NO.	USER STORY	TIME ESTIMATE

SPRINT 3	START DATE		END DATE	

NO.	USER STORY	TIME ESTIMATE

PRODUCT BACKLOG List any high-level goals and user stories discussed in planning, but which did not make the release.

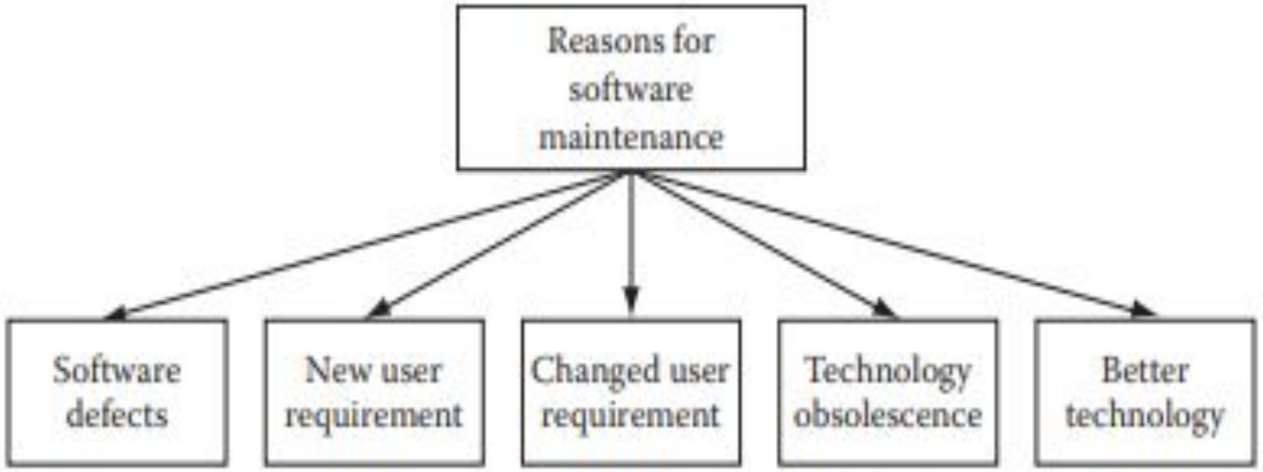
BACKLOG ITEM

User Training

- Make sure that the user manual prepared by your team is up to date and in synch with the version of your software product, which you will implement at the customer site.
- It is not possible to provide training to all users. So prepare a list of roles that are needed to operate the product.
- Give this list to the end users and ask them to select one user per role who will receive the training.
- Apart from the user manual, you also need to prepare a tutorial to include probable scenarios that may arise during operation of the product.
- The tutorial will provide a step-by-step guide for using the product under those scenarios.
- This will be a very important step in training, because if users do not learn it during training, then they will contact you later after implementation and ask you to provide information as to how to use the product in those circumstances.

User Training

- This will lead to a waste of your support team's time.
- It is lot better to train them now, during user training, rather than face user requests later



OBJECTIVES

- To appreciate need of Software maintenance performed.
- To understand reasons for change taking place in a software system.
- To appreciate the concept of legacy system.
- To know Software maintenance prediction.
- To list various factors which adversely affect software maintenance.
- To know types of software maintenance, viz. corrective, adaptive, perfective, and preventive maintenance.
- To understand the Software maintenance life cycle.

OBJECTIVES

- To know various software maintenance models, viz. quick-fix, iterative enhancement and reuse model.
- To understand various techniques for maintenance, such as configuration management, impact analysis and software rejuvenation.
- To list various tools that assist in maintaining the software.
- To appreciate need for technology change management, which is a process of identifying, selecting and evaluating new technologies.
- To understand software maintenance documentation.

CONTENTS

BASICS OF SOFTWARE MAINTENANCE

IEEE defines maintenance as :

‘A process of modifying a software system or component after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment’.

- In software engineering a software needs to be ‘serviced’ so that it is able to meet the changing environment (such as business and user needs) where it functions this servicing of software.

- **Providing Continuity of Service**

  Fixing errors, recovering from failures, such as hardware failures or incompatibility of hardware with software, and accommodating changes in the operating system and the hardware.

- **Supporting Mandatory Upgrades**

  Due to changes in government regulations or students to maintain competition with other software that exist in the same category.

- **Improving the Software to Support User Requirements**

  To enhance functionality in a software, to improve performance .

- **Facilitating Future Maintenance Work**

  Include restructuring of the software code and database used in the software.

- **Improving the Software to Support User Requirements**

  To enhance functionality in a software, to improve performance .

- **Facilitating Future Maintenance Work**

  Include restructuring of the software code and database used in the software.

Changing a Software System

- Software maintenance and evolution of system was first introduced by Lehman.
- One of the key observations of the studies was that large system are never complete and continue to evolve and as these system evolve, they become more complex unless some actions are taken to reduce the complexity.
- Lehman stated five laws for software maintenance and evolution of large systems.

Lehman Laws for software maintenance and evolution of large systems

Law	Description
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.
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.
Law of conservation of familiarity	For E-Systems to efficiently continue to evolve a deep understanding of how the system functions, and why it has been developed to function in that manner, must be preserved at all costs. The incremental change in each release over the life time of the system is approximately constant.
Law of conservation of organizational stability	Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.

Lehman Laws for software maintenance and evolution of large systems

Law	Description
Law of self regulation	Global E-type system evolution processes are self-regulating, and distribution of product and process measures close to normal.
Law of continuing growth	E-type system's functional capability must be continually enhanced to maintain user satisfaction over system lifetime, BUT expansion of the system size can have negative affects to the ability to be comprehended along with its ability to evolve.
Law of declining quality	Poorly modified systems lead to introduction of defects; & The quality of E-type systems will appear to be declining as newer products emerge.
Law of feedback system	To sustain continuous change or evolution, & to minimize threats of software decay & loss of familiarity, feedback to monitor the performance is must. Feedback helps to collect metrics on the systems and maintenance efforts performance.

Legacy Systems

- Were developed before the introduction of structured programming .
- Process models and basic principles such as modularity coupling, cohesion, good programming practice emerged too late for them.
- Legacy system are generally associated with high maintenance costs.
- The root cause of this expense is the degraded structure that results from prolonged maintenance.

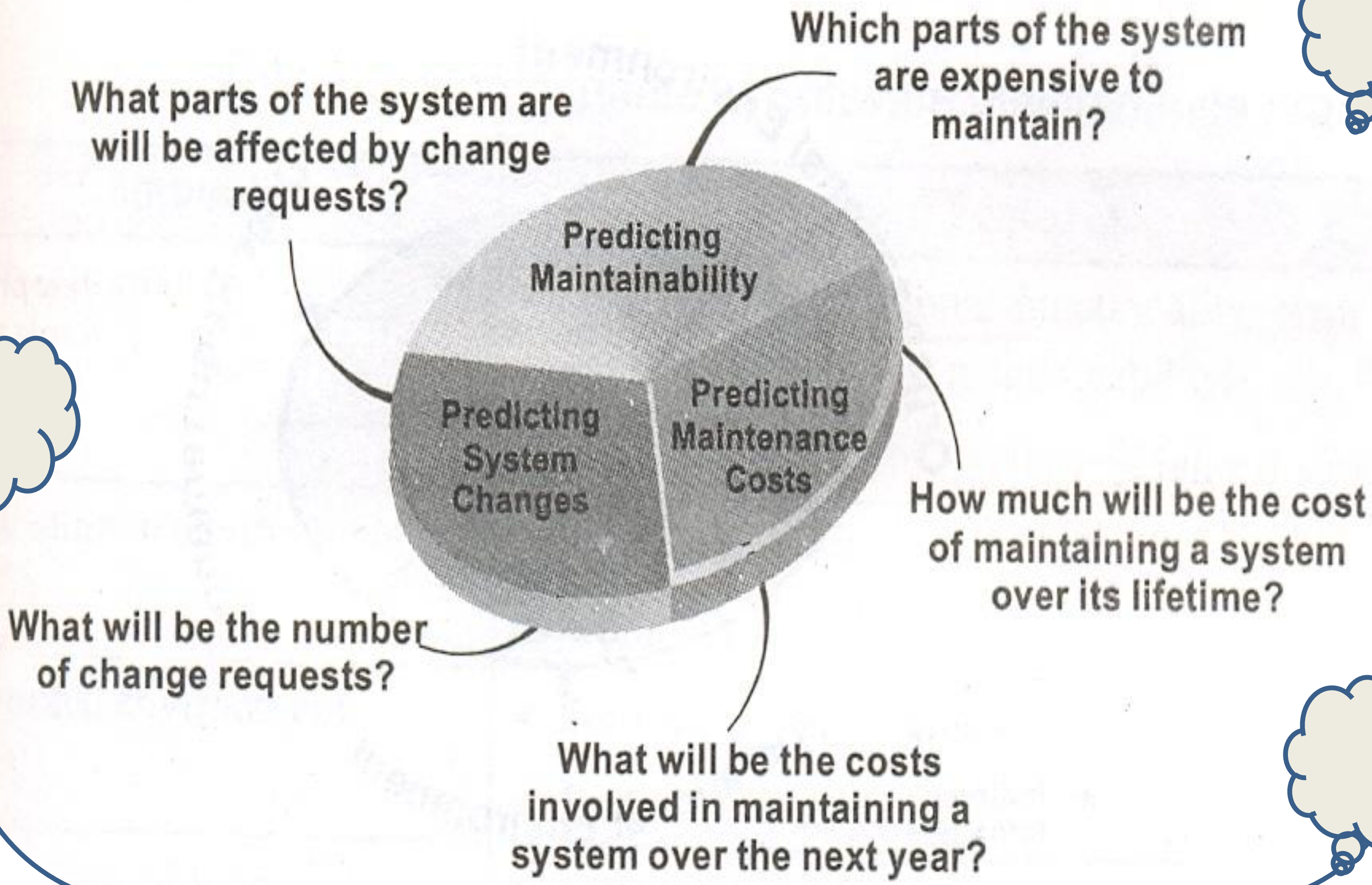
Legacy System Characteristics

Characteristics	Description
High maintenance cost	Results due to combination of other system factors, such as complexity, poor documentation and lack of inexperienced personnel.
Complex software	Results due to structural degradation, which must have occurred over a legacy system's lifetime of change.
Obsolete support software	Support software may not be available for a particular platform, or no longer be supported by its original vendor or any other organization.
Obsolete hardware	Legacy system's hardware may have been discontinued.

Legacy System Characteristics

Characteristics	Description
Lack of technical expertise	Original developers of a legacy system are unlikely to involved with its maintenance today.
Business critical	Many legacy system are essential for the proper working of the organizations which operate them.
Poorly documented	Documentation is often missing or inconsistent.
Poorly understood	As a consequence of system complexity and poor documentation, software maintainers often understand the legacy system poorly.

Software Maintenance Prediction


Which parts of the system are expensive to maintain?
What parts of the system are will be affected by change requests?
Predicting Maintainability
Predicting System Changes
Predicting Maintenance Costs
How much will be the cost of maintaining a system over its lifetime?
What will be the number of change requests?
What will be the costs involved in maintaining a system over the next year?

Software Maintenance Prediction

The various predictions shown in the figure are closely related and specify the following:

- The decision to accept a system change depends on the maintainability of the system components affected by that change up to a certain extent.

- Implementing change degrades the system structure and hence reduces its maintainability.

- Costs involved in implementing change depend on the maintainability of system components.

Components of Software Maintenance Framework

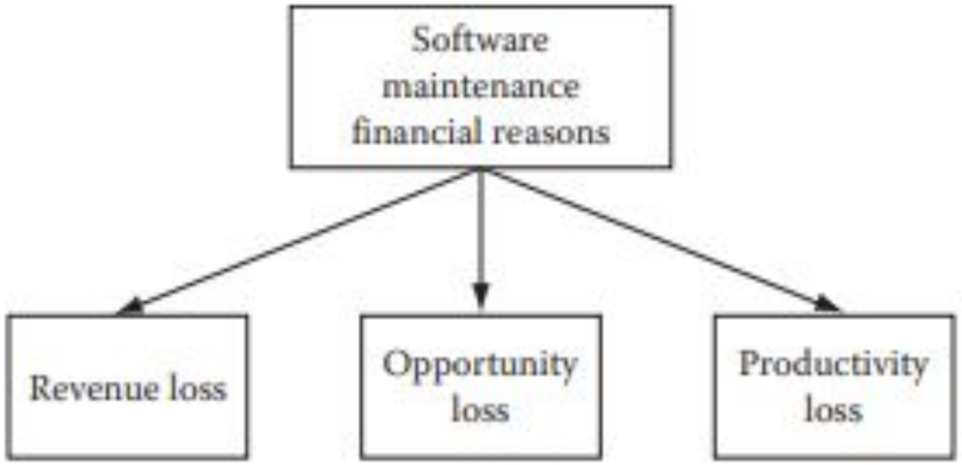
Components	Features
User requirements	•Request for additional functionality, error correction, capability and improvement in maintainability. • Request for non-programming related support.
Organizational Environment	• Change in business policies. • Competition in market.
Operational Environment	• Hardware platform. • Software specifications.

Components of Software Maintenance Framework

Components	Features
Maintenance Process	• Capturing requirements • Variation in programming and working practices. • Paradigm shift. • Error detection and correction.
Software Product	• Quality of documentation. • Complexity of programs. • Program Structure.
Software Maintenance team	• Staff turnover. • Domain expertise.

Maintenance Cost

A software product is generally very valuable to an organization if it is used for doing a large portion of their daily business. If for some reason the software product has become unusable, then the organization in fact will be making losses on their revenue. Moreover, large enterprise software products are that much crucial. When the organization faces such a case, it is left with no alternative but to either get an entirely different software product that will replace the existing one or do maintenance of an existing product to make it usable.



Maintenance Cost

Following are some financial reasons for which a maintenance may be needed:

1. Loss in business revenue: It may happen that business transactions are faulty and thus the business may lose revenue.
2. Opportunity loss: Sometimes there could be some business opportunity in the marketplace, but due to some software problems it could not be availed.
3. Productivity loss: If the software product becomes difficult to operate due to many walk around or lengthy processing then productivity will become lower for business personnel Maintenance of an existing software product has its own share of problems. The maintenance will incur costs. A profit/loss analysis can be done, to see if it is more profitable to conduct a maintenance program on the software or keep using it as it is.

The losses due to problems with the software can be compared to probable cost of maintenance and an ROI (return on investment) can be done.

If we get a desirable ROI then it is better to go for maintenance.

Factors Affecting Software Maintenance

- Relationship of Software product and Environment
- Relationship of Software product and User
- Relationship of Software product and Software Maintenance team

- **Software Maintenance Team:**
  - Various functions performed by the Software Maintenance team are:
  - Locating information in system documentation
  - Keeping system documentation up-to-date
  - Extending existing functions to accommodate new or changing requirements
  - Adding new functions to the system
  - Finding the source of system failures or problems
  - Managing change to the system as they are made.
- **The aspects of a maintenance team that lead to high maintenance costs are:**
  - Staff turnover
  - Domain expertise

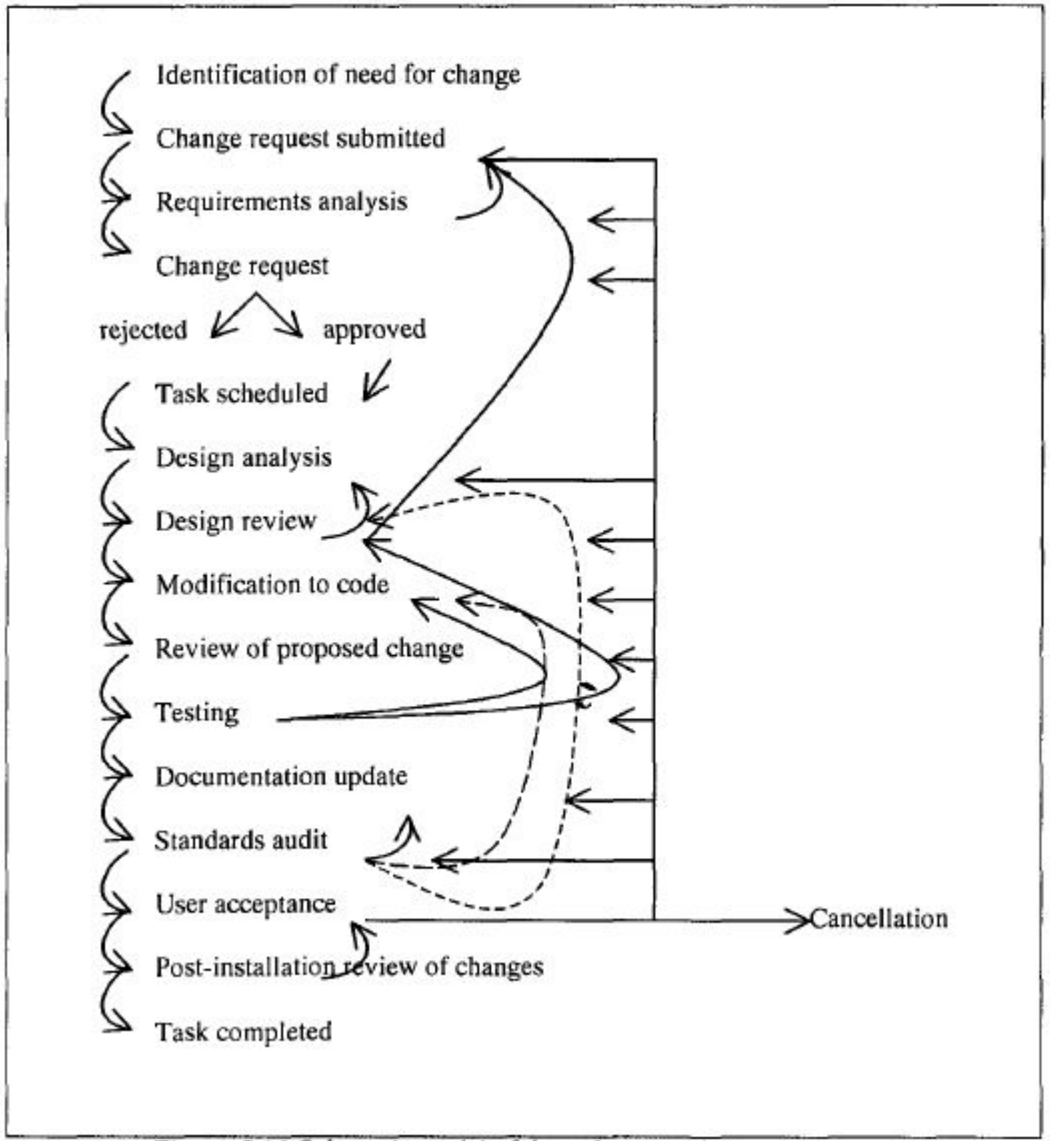
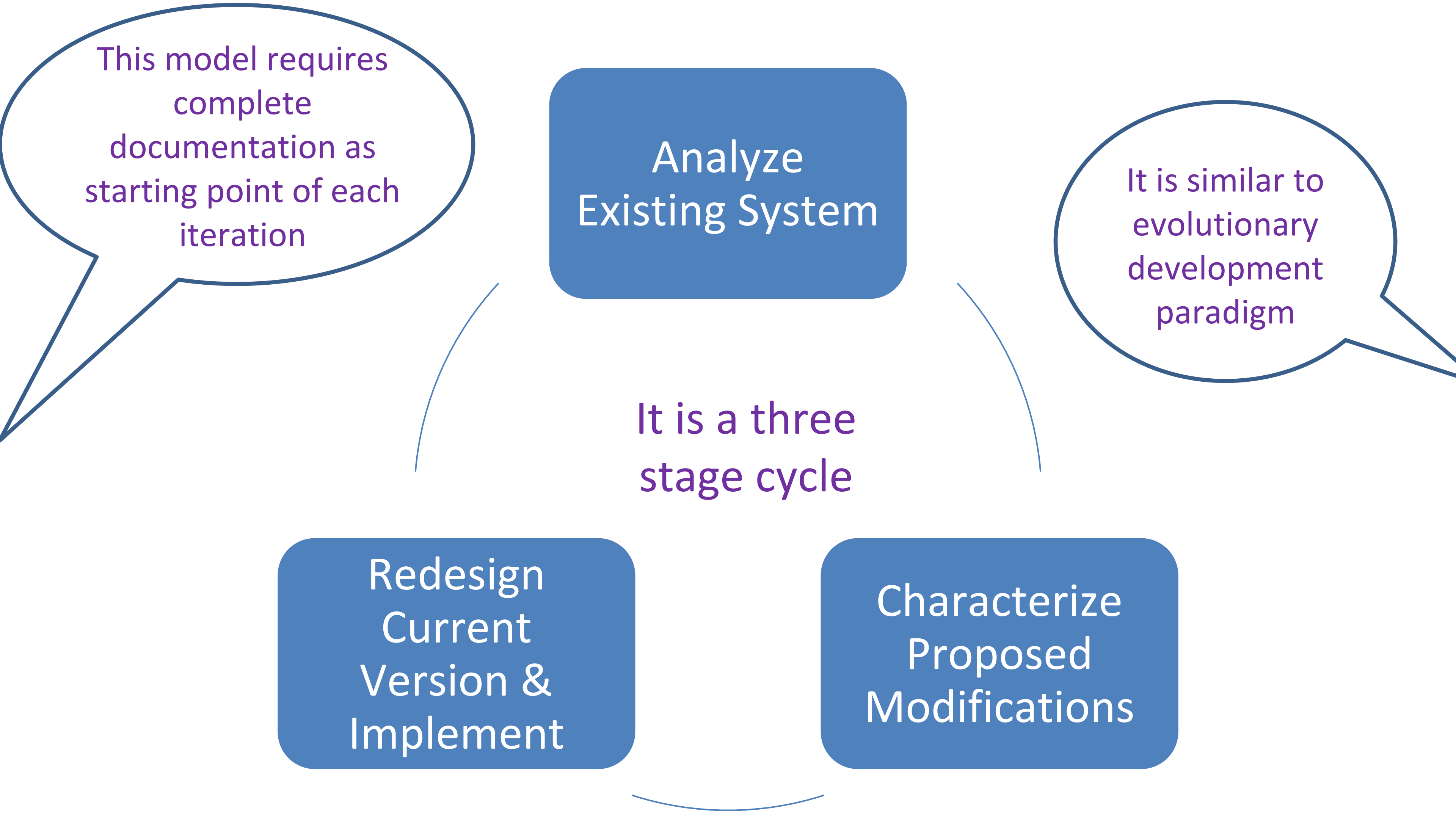


Figure 5.16 Osborne's model of the software maintenance process

Osborne hypothesizes that many technical problems which arise during maintenance are due to inadequate management communications and control, and recommends a strategy that includes:

1. the inclusion of maintenance requirements in the change specification;
2. a software quality assurance program which establishes quality assurance requirements;
3. a means of verifying that maintenance goals have been met; performance review to provide feedback to managers

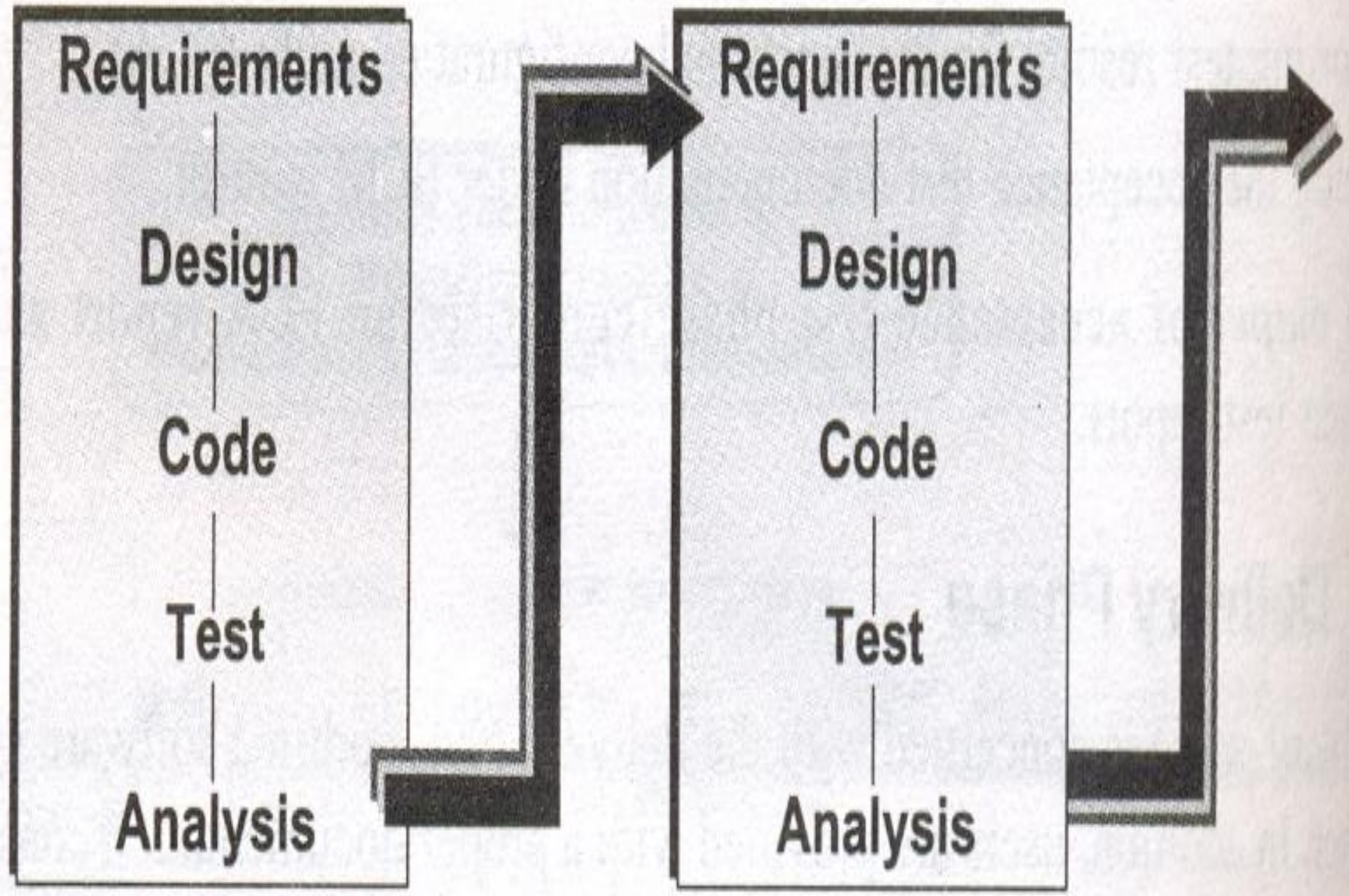
Iterative Enhancement Model


This model requires complete documentation as starting point of each iteration
Analyze Existing System
It is similar to evolutionary development paradigm
It is a three stage cycle
Redesign Current Version & Implement
Characterize Proposed Modifications

Existing documentation for each stage is:

- Requirement documentation, Design Documentation, Source code documentation, & Test documentation

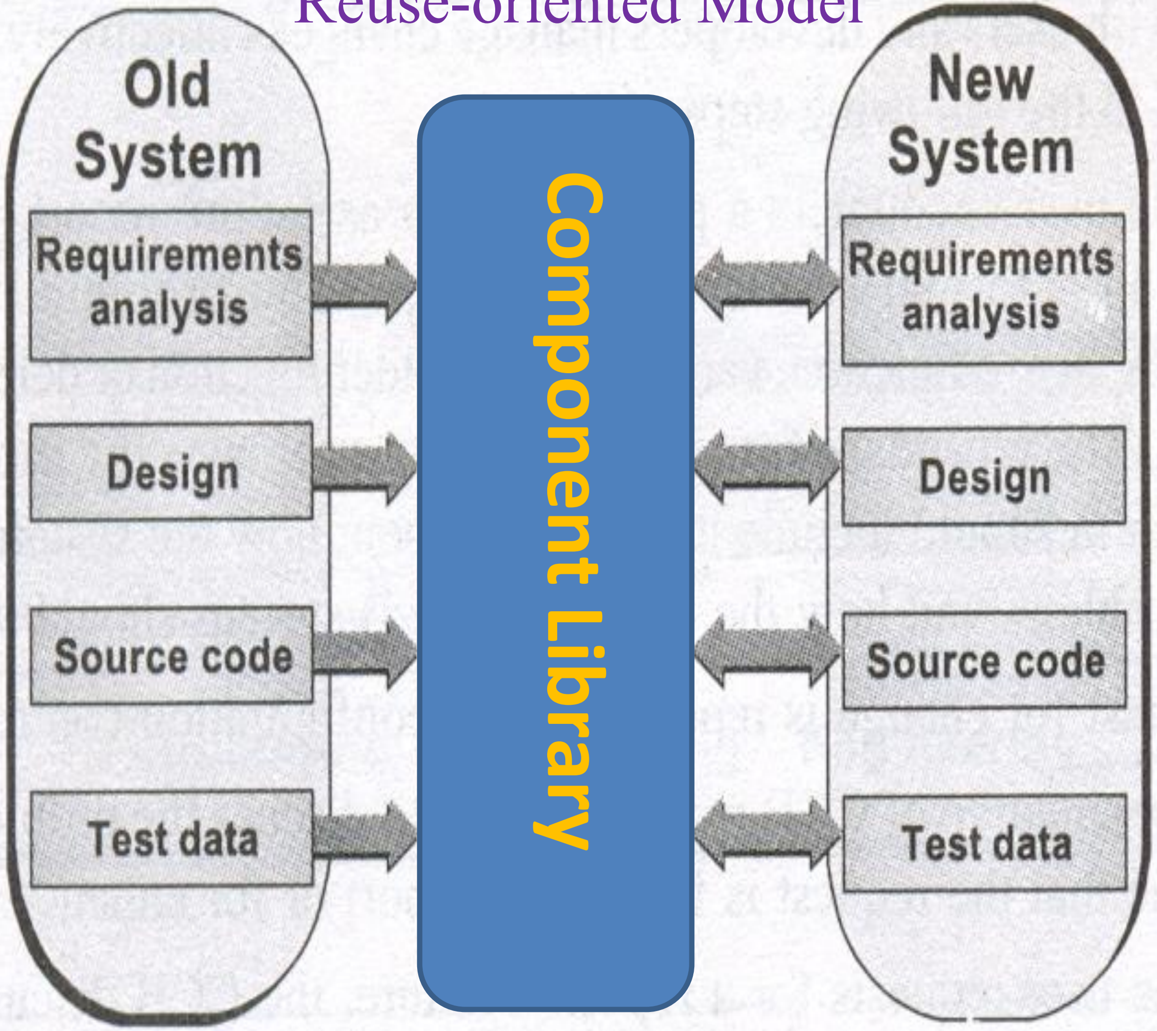
Iterative Enhancement Model



The Reuse-Oriented Model

- Maintenance is viewed as the activity involving the reuse of existing program components.
- It builds a component library for reusing requirements, design, source code, and test-data.
- A detailed framework is required for classification of components for reuse.

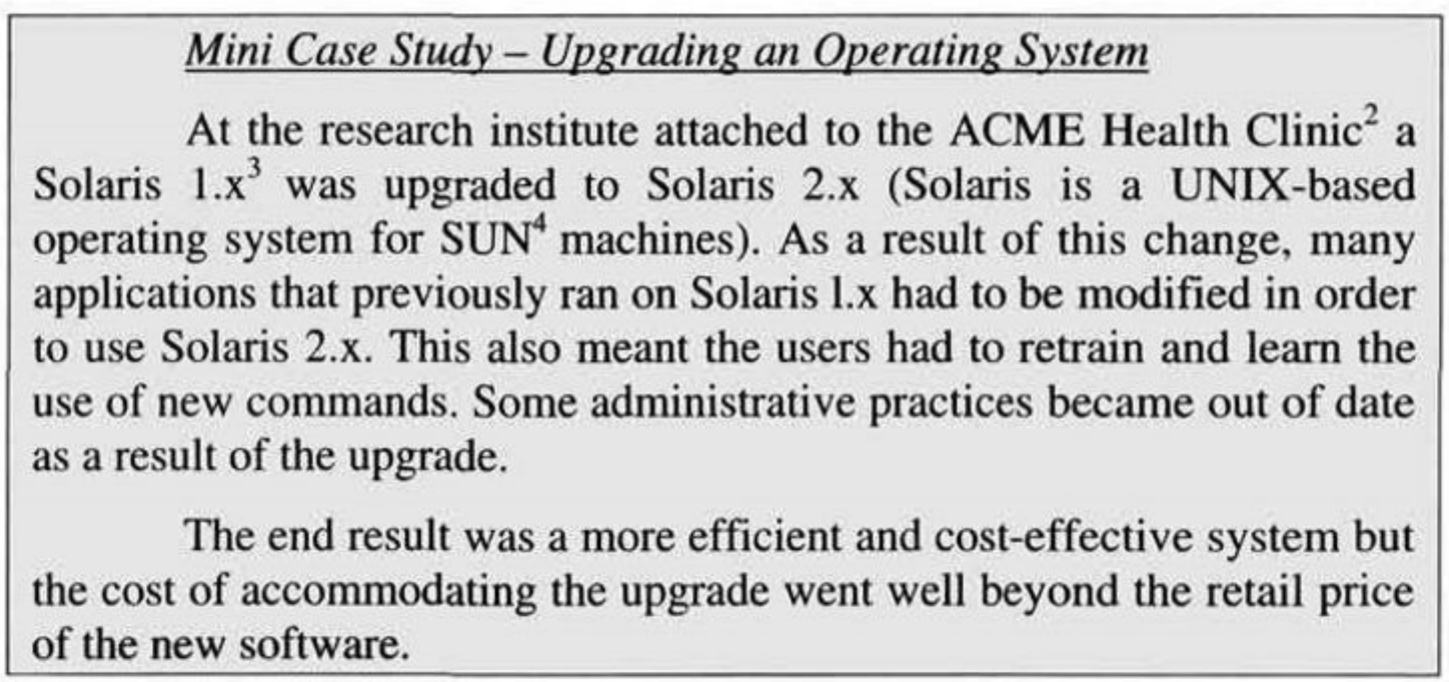
Reuse-oriented Model



The Reuse-oriented Model has 4 main steps:

- Identifying the parts of the old system which have the potential for reuse,
- Fully understanding the system parts,
- Modifying the old system parts according to the new requirements, and
- Integrating the modified parts into the new system.

Case study

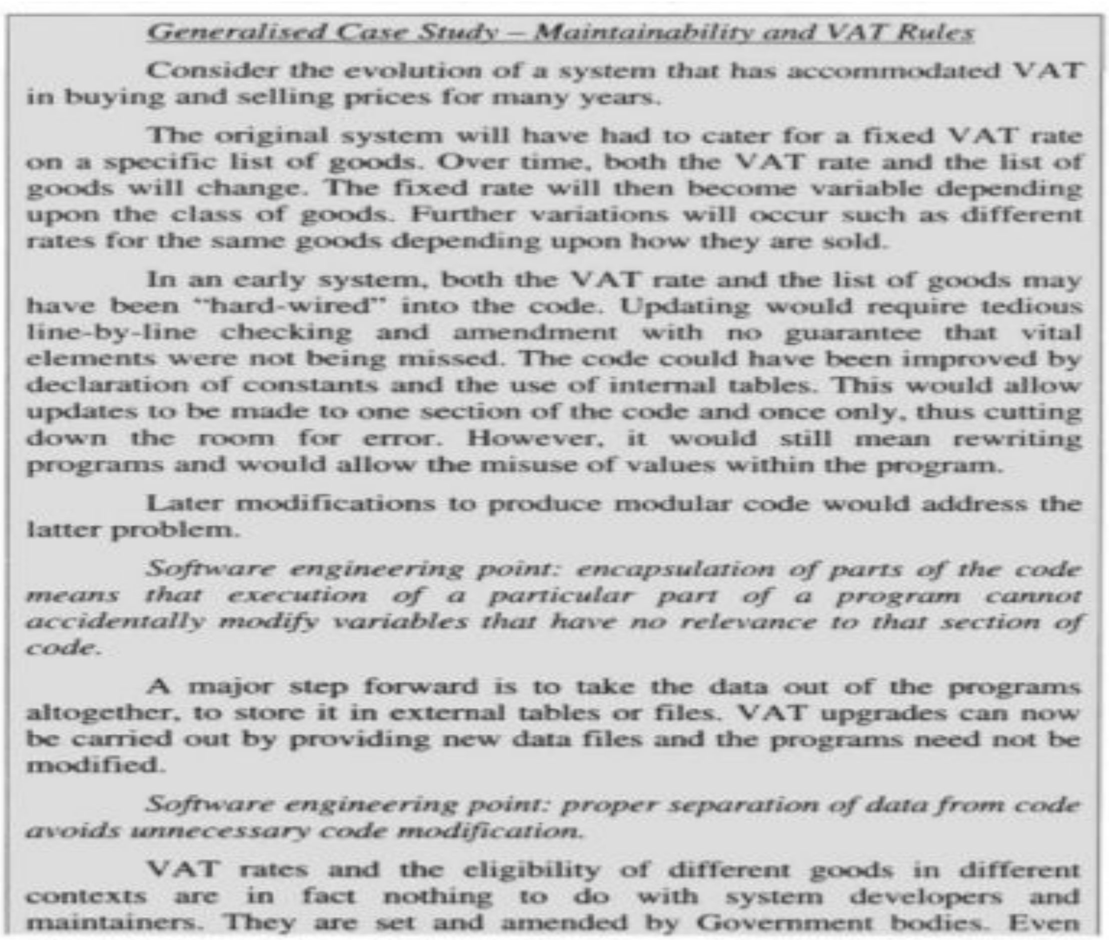


Mini Case Study – Upgrading an Operating System

At the research institute attached to the ACME Health Clinic[2] a Solaris 1.x[3] was upgraded to Solaris 2.x (Solaris is a UNIX-based operating system for SUN[4] machines). As a result of this change, many applications that previously ran on Solaris l.x had to be modified in order to use Solaris 2.x. This also meant the users had to retrain and learn the use of new commands. Some administrative practices became out of date as a result of the upgrade.

The end result was a more efficient and cost-effective system but the cost of accommodating the upgrade went well beyond the retail price of the new software.

Case study2



Generalised Case Study – Maintainability and VAT Rules

Consider the evolution of a system that has accommodated VAT in buying and selling prices for many years.

The original system will have had to cater for a fixed VAT rate on a specific list of goods. Over time, both the VAT rate and the list of goods will change. The fixed rate will then become variable depending upon the class of goods. Further variations will occur such as different rates for the same goods depending upon how they are sold.

In an early system, both the VAT rate and the list of goods may have been "hard-wired" into the code. Updating would require tedious line-by-line checking and amendment with no guarantee that vital elements were not being missed. The code could have been improved by declaration of constants and the use of internal tables. This would allow updates to be made to one section of the code and once only, thus cutting down the room for error. However, it would still mean rewriting programs and would allow the misuse of values within the program.

Later modifications to produce modular code would address the latter problem.

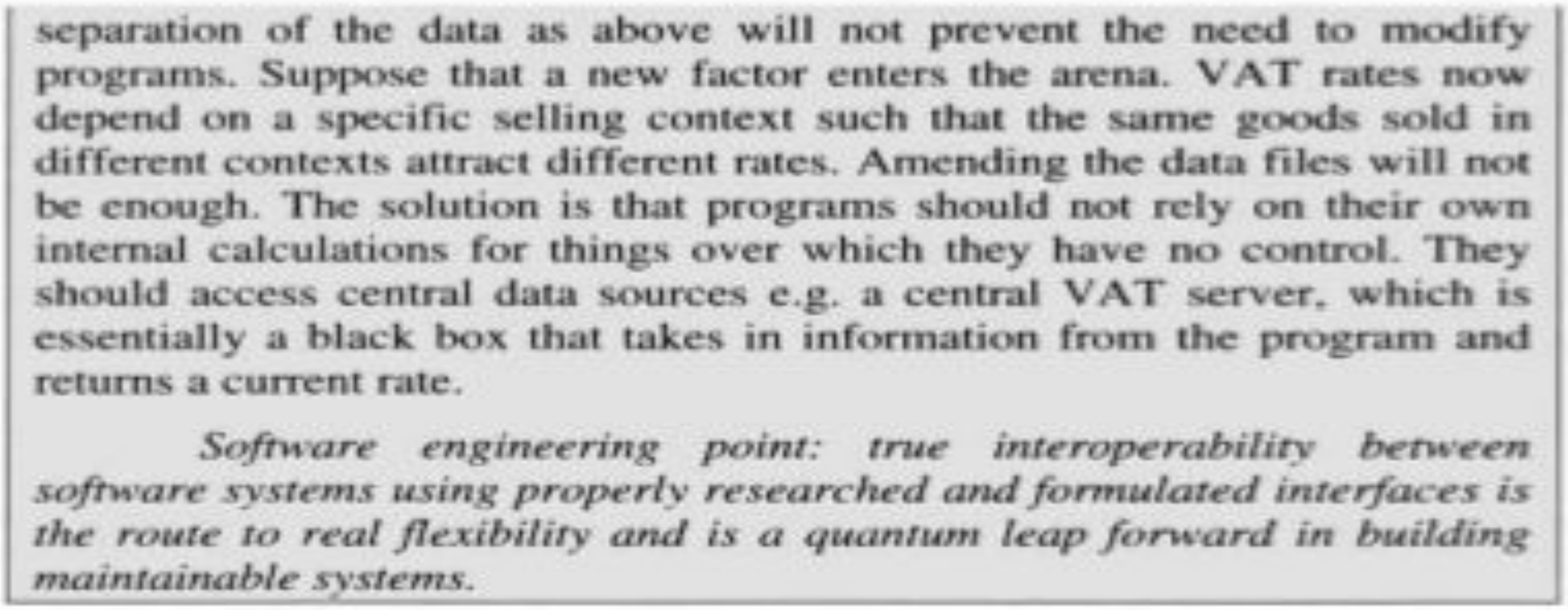
Software engineering point: encapsulation of parts of the code means that execution of a particular part of a program cannot accidentally modify variables that have no relevance to that section of code.

A major step forward is to take the data out of the programs altogether, to store it in external tables or files. VAT upgrades can now be carried out by providing new data files and the programs need not be modified.

Software engineering point: proper separation of data from code avoids unnecessary code modification.

VAT rates and the eligibility of different goods in different contexts are in fact nothing to do with system developers and maintainers. They are set and amended by Government bodies. Even

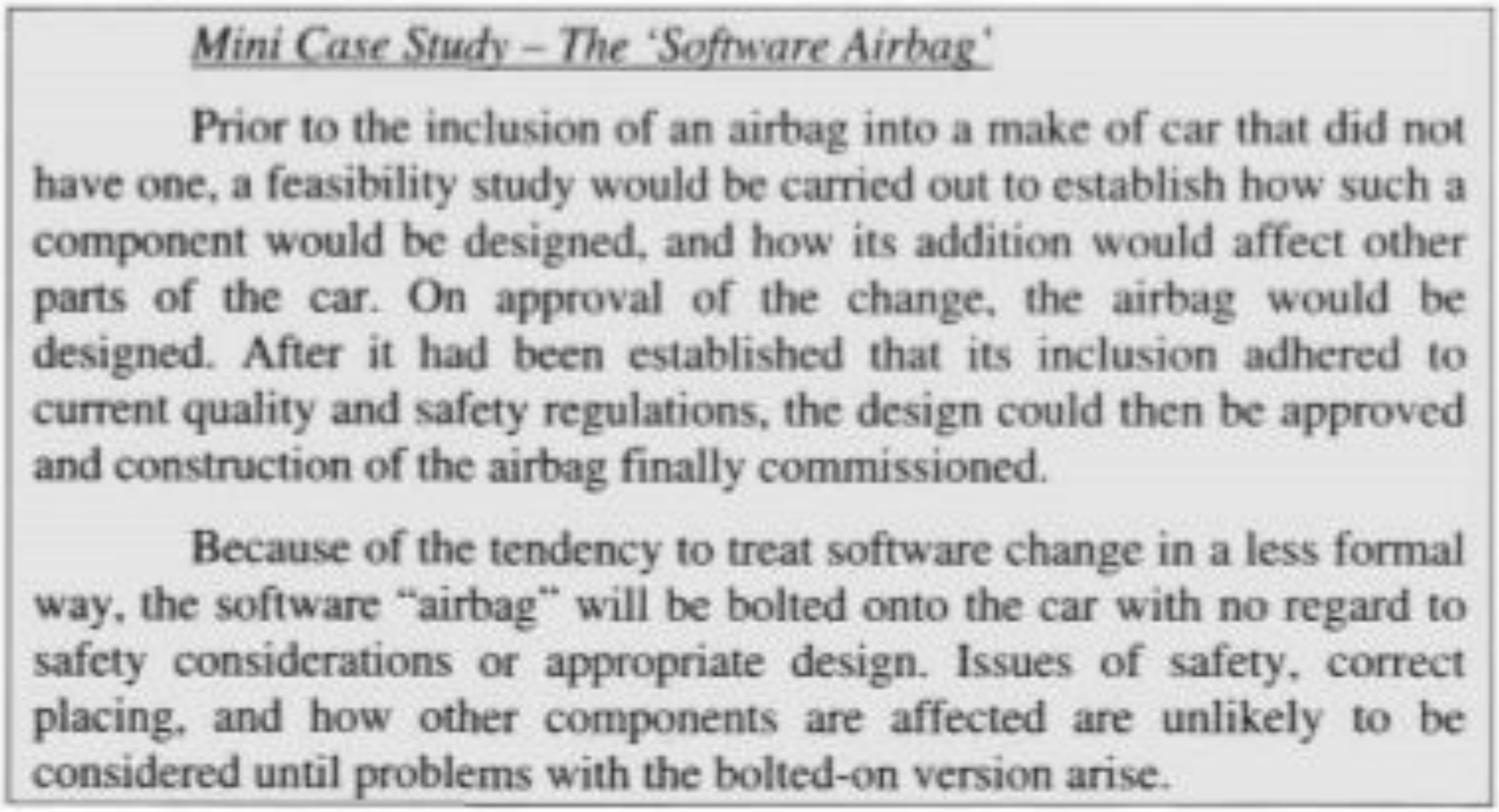
Case study 2 continuation



separation of the data as above will not prevent the need to modify programs. Suppose that a new factor enters the arena. VAT rates now depend on a specific selling context such that the same goods sold in different contexts attract different rates. Amending the data files will not be enough. The solution is that programs should not rely on their own internal calculations for things over which they have no control. They should access central data sources e.g. a central VAT server, which is essentially a black box that takes in information from the program and returns a current rate.

Software engineering point: true interoperability between software systems using properly researched and formulated interfaces is the route to real flexibility and is a quantum leap forward in building maintainable systems.

Casestudy 3

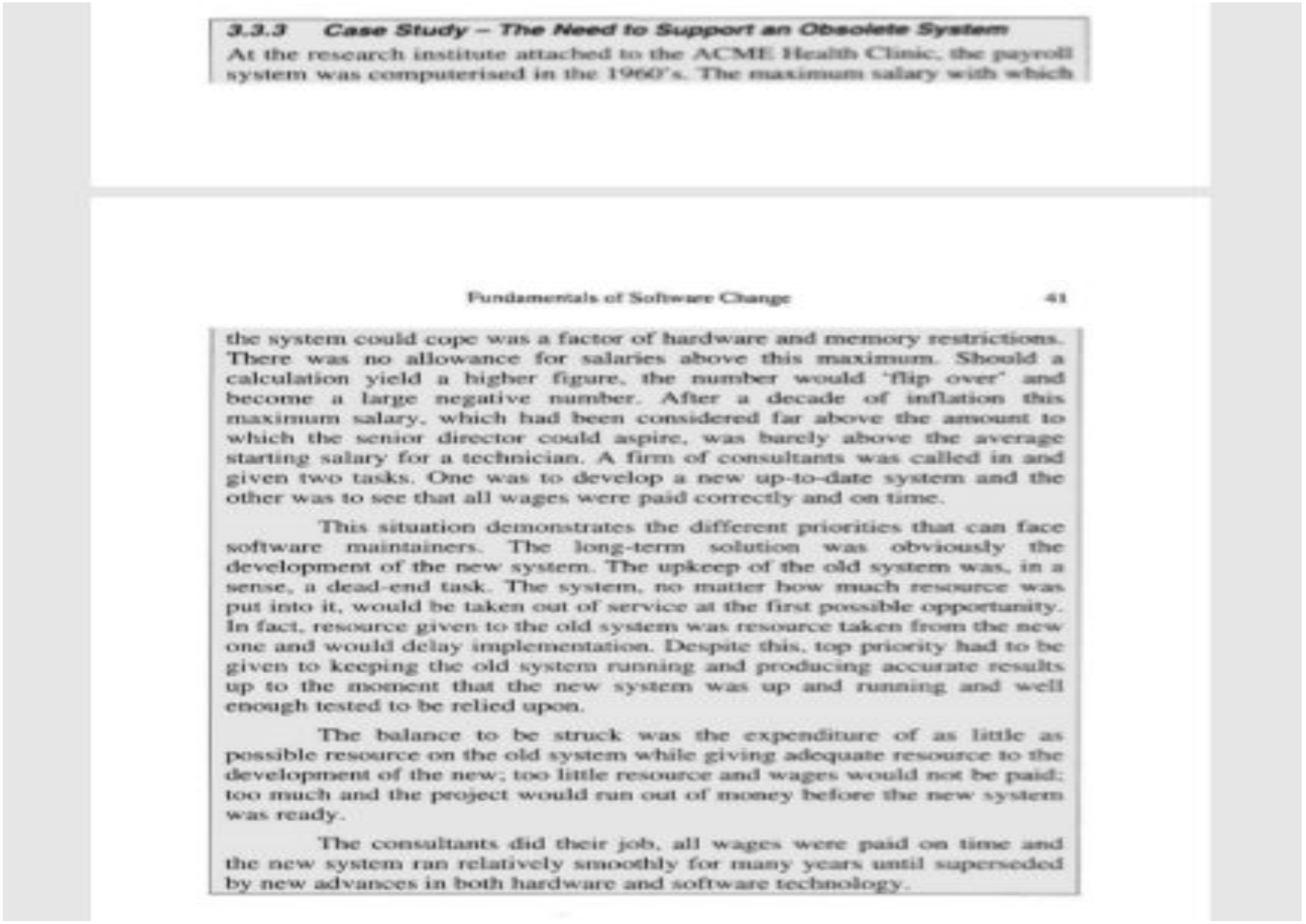


Mini Case Study – The 'Software Airbag'

Prior to the inclusion of an airbag into a make of car that did not have one, a feasibility study would be carried out to establish how such a component would be designed, and how its addition would affect other parts of the car. On approval of the change, the airbag would be designed. After it had been established that its inclusion adhered to current quality and safety regulations, the design could then be approved and construction of the airbag finally commissioned.

Because of the tendency to treat software change in a less formal way, the software "airbag" will be bolted onto the car with no regard to safety considerations or appropriate design. Issues of safety, correct placing, and how other components are affected are unlikely to be considered until problems with the bolted-on version arise.

Casestudy 4



3.3.3 Case Study – The Need to Support an Obsolete System

At the research institute attached to the ACME Health Clinic, the payroll system was computerised in the 1960's. The maximum salary with which

the system could cope was a factor of hardware and memory restrictions. There was no allowance for salaries above this maximum. Should a calculation yield a higher figure, the number would 'flip over' and become a large negative number. After a decade of inflation this maximum salary, which had been considered far above the amount to which the senior director could aspire, was barely above the average starting salary for a technician. A firm of consultants was called in and given two tasks. One was to develop a new up-to-date system and the other was to see that all wages were paid correctly and on time.

This situation demonstrates the different priorities that can face software maintainers. The long-term solution was obviously the development of the new system. The upkeep of the old system was, in a sense, a dead-end task. The system, no matter how much resource was put into it, would be taken out of service at the first possible opportunity. In fact, resource given to the old system was resource taken from the new one and would delay implementation. Despite this, top priority had to be given to keeping the old system running and producing accurate results up to the moment that the new system was up and running and well enough tested to be relied upon.

The balance to be struck was the expenditure of as little as possible resource on the old system while giving adequate resource to the development of the new; too little resource and wages would not be paid; too much and the project would run out of money before the new system was ready.

The consultants did their job, all wages were paid on time and the new system ran relatively smoothly for many years until superseded by new advances in both hardware and software technology.

THANK YOU