Hyperiondev

TASK

Deployment and Maintenance Best Practice

Visit our website

Introduction

Welcome to the Deployment and Maintenance Best Practice Task!

In this task , the processes of getting software out of the hands of the developers into the hands of the end-users and changing a system after it has been delivered.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

Deployment

A software system must be placed into operation as soon as it has been developed and tested. There are many activities involved with deployment of a system. These activities include:

- Converting existing data
- Building training materials and conducting training
- Configuring and setting up the production environment

These activities have many conflicting constraints, including cost, maintaining positive customer relations, supporting employees, logistical complexity, and overall risk to the company. We will now take a closer look at these activities.

Converting and Initialising Data

In order to support processing that is ongoing, an operational system requires an already fully populated database. A system with an online order-entry function, for example, needs information about products, promotions, customers, and previous orders. Developers must therefore ensure that such information is available to the system as soon as it becomes operational.

Data that is needed when the system is initially deployed are obtained from:

- Files or databases from the system that is being replaced
- Manual records
- Files or databases from other systems within the organisation
- User feedback from when the system is operating normally

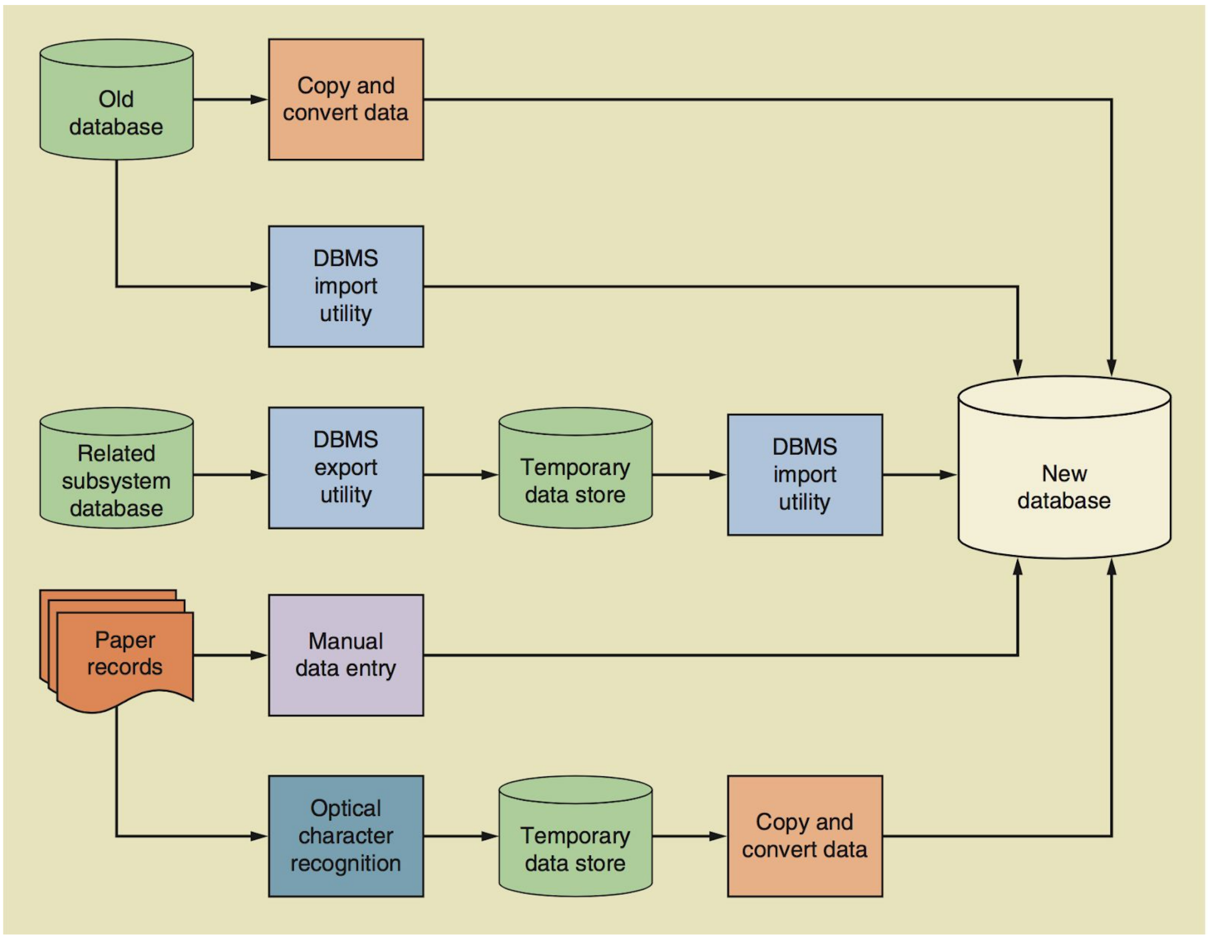
Reusing an Existing Database

Most new software systems are developed to replace older, existing systems. Data conversion, in its simplest form, involves using the old system's database directly in the new system with little or no change to the structure of the database. It is fairly common to reuse a database that already exists because of the difficulty and expense of creating a whole new database from scratch. This is especially true when a single database supports multiple information systems such as in **enterprise resource planning** (ERP) systems.

Although old databases are commonly reused in new systems, some changes to the database content are usually required. New tables and new attributes are usually added and existing tables and attributes are modified. DBMSs usually allow database administrators to modify the structure of a fully populated database. DBMSs can perform simple changes such as adding new attributes or changing attribute types.

Reloading Databases

Complex changes to database structure might require creating an entirely new database and copying and converting data from the old database to the new database. Utility programs supplied with the DBMS are used to copy and convert the data whenever possible. In more complex conversions, programs must be developed to perform the conversion and transfer data. The diagram below shows both approaches, either copying and converting data or using the DBMS import utility. In both cases, once the conversion and transfer process is complete, the old database can be discarded.



Complex data conversion (John W. Satzinger et al., 2000)

In the diagram below, the middle section shows an approach that uses an export utility, an import utility, and a temporary data store. This more complex approach might be used when the old and new databases use different database technologies.

Using the same programs being developed for the operational system, data from paper records can be entered. Initial data entry can be used as a user training exercise. Data obtained from paper records can also be scanned into an optical character recognition program. The data can then be entered into the database by using conversion programs that have been custom-developed or a DBMS import utility.

It is also possible to start operating the system without a database that is either empty or partially empty. For the order-entry system, for example, existing customer information is not needed to be initially loaded onto the database. The customer's information could be added to the database as soon as they place their first order. Adding data as they are encountered reduces the complexity of data conversion but slows down processing of initial transactions.

Training Users

There are two types of users that need to be trained; end-users and the operators of the system. End users people who use the system regularly while system operators are people who perform administrative functions and routine maintenance to keep the system operating.

End-user typically:

- Create records or transactions
- Modify the database contents
- Generate reports
- Query the database
- Import or export data

System operators on the other hand typically:

- Start or stop the system
- Query the system status
- Backup data to archive
- Recover data from archive
- Install or upgrade software

Depending on whether the target audience is end-users or system operators, the nature of the training varies. For end users, training emphasises hands-on use for specific business functions. Examples of business functions are; order entry, inventory control, and accounting. Training must include those procedures if users are not familiar with them already. End-users typically have a variety of skill and experience levels, therefore it is advisable to use hands-on training. This can include practice exercises, questions and answers, and one-on-one tutorials. Face-to-face training is preferable but self-paced training materials can also be used to an extent. Group training sessions can be used for large groups of end users and experienced end-users that are well versed in the system can be trained to train other end-users.

When the system operators aren't also end users, their train can be much less formal. Operators and administrators can learn most or all they need to know by self-study. Formal training sessions may therefore not be required. Since there are also a relatively small number of system operators, it is feasible to provided one-on-one training if it is needed.

The best time to begin formal training can be difficult to determine. Users can be trained as parts of the system are developed and tested, which ensures that they start using the system as soon as possible. This, however, can lead to users and trainers becoming frustrated by using a system that may not be stable or complete. A buggy, crash-prone system with constantly changing features and interfaces can cause end-users to become increasingly discouraged.

Ideally, training should not begin until the interfaces are finalised and a test version has been installed and fully debugged. However, this approach is not often used due to time constraints. Training materials are therefore normally developed as soon as the interfaces are reasonably stable, and training begins as soon after.

Before any formal training begins, documentation and other training materials are developed. There are two types of documentation:

1. System documentation: This provides descriptions of system requirements, architecture, and construction details
2. User documentation: This provides descriptions of how to interact with and use the system

Each type of documentation is targeted at a different audience and has a different purpose. System documentation is used to support present and future development activities. User documentation is only created during implementation as many details of the user interface and system operation haven't yet been determined or may change earlier on during development.

System Documentation

System documentation is used to provide information to developers and other technical personnel who will build, maintain, and upgrade the system. System documentation is developed throughout the development of the system. System documentation that is generated early on in the project is used to guides activities later on and documentation developed throughout the lifecycle of the system guides future system maintenance and upgrades.

User Documentation

User documentation provides ongoing support for the systems end-users. It describes the routine operation of the system, including functions such as data entry, output generation, and periodic maintenance. The topics that are covered in user documentation include:

- The startup and shutdown of software
- Keystroke, mouse, or command sequences required to perform certain functions
- Program functions required to implement specific business procedures. For example, what are the steps that you need to follow to enter a new customer order?
- Common errors and how to correct them

User documentation normally includes a table of contents, a general description of the purpose and function of the system, a glossary, and an index in order to make using it easier. User documentation is typically electronic and is an extremely important part of the system. Operating systems usually provide facilities to support embedded documentation.

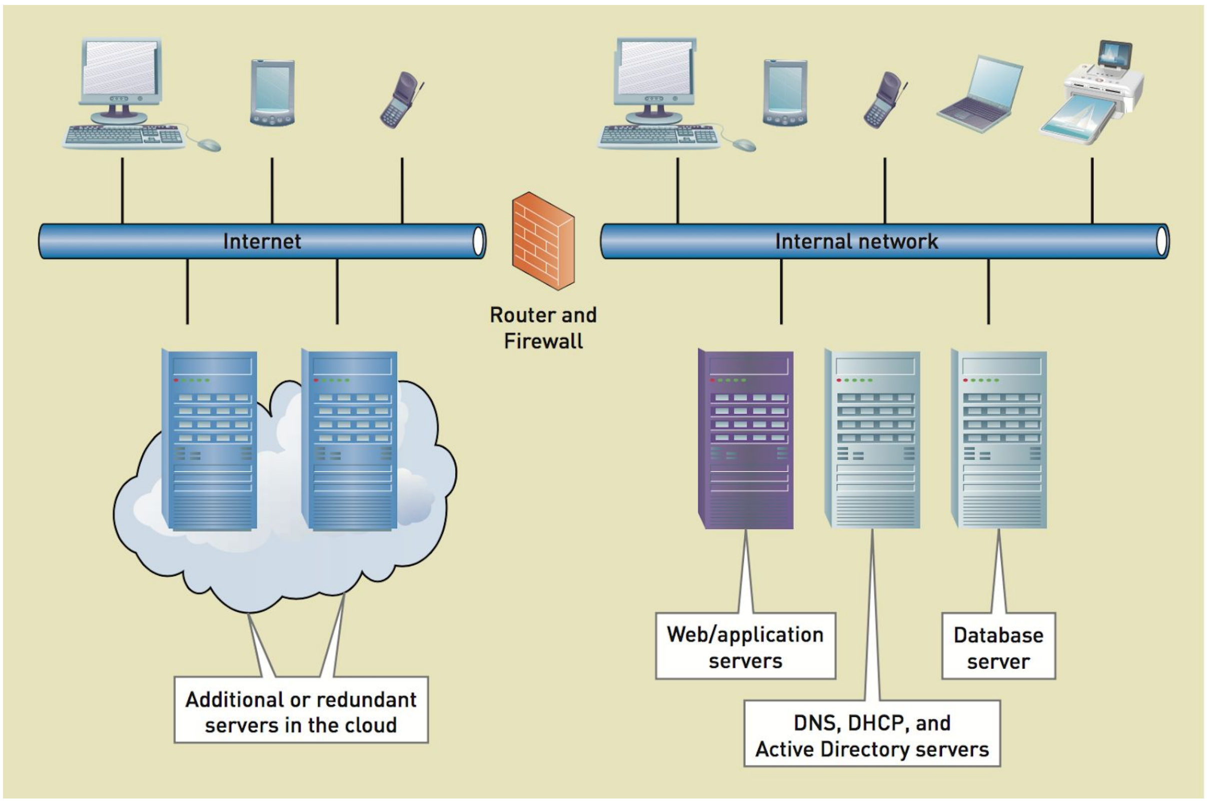
Knowing how to use a system is as important as the system itself. Training provides end-users with practical knowledge however this knowledge is often difficult to maintain or transfer to other users. It is often difficult for direct person-to-person transfer of operational knowledge to take place due to many factors such as employee turnover and reassignment. Written or electronic documentation is easier to access and far more permanent.

Developing good user documentation requires skill, time and resources. In order to develop good user documentation, one needs to be able to write clearly and concisely, develop effective presentation graphics, organise information for easy learning and access, and communicate effectively with a nontechnical audience. Good quality user documentation takes time to produce and are achieved only with thorough review and testing.

Configuring the Production Environment

Applications are built from software components which are based on interaction standards. Examples of interaction standards are, Common Object Request Broker Architecture (CORBA), Simple Object Access Protocol (SOAP), and Java Platform Enterprise Edition (Java EE). Each standard defines ways in which components locate and communicate with one another. They also define a set of supporting system software that provides services, such as maintaining component directories, enforcing security requirements, and encoding and decoding messages across networks and other transport protocols. The exact system software, hardware, and its configuration requirements vary substantially among the component interaction standards.

A support infrastructure for an application that has been deployed using Microsoft .NET is shown below. Microsoft .NET is a variant of SOAP.


Internet
Internal network
Router and Firewall
Additional or redundant servers in the cloud
Web/application servers
Database server
DNS, DHCP, and Active Directory servers

Support infrastructure for Microsoft .NET application (John W. Satzinger et al., 2000)

Components of the application are stored on one or more application servers. Other services that are required include a Web server for browser-based interfaces, a database server to manage the database, an Active Directory server to authenticate users and authorise access to information and software resources, a router and firewall, and a server to operate low-level Internet services such as domain naming (DNS) and Internet address allocation (DHCP).

All hardware and software infrastructure must be acquired, installed, and configured before application software can be installed and tested. Some or all of the infrastructure will already exist to support existing information systems in most cases. If this is the case developers work closely with the administrators of the existing infrastructure to plan the support for the new system.

Packaging, Installing, and Deploying Components

You cannot deploy a system that has not been implemented and tested. Large and complex systems are typically deployed in multiple stages or versions. Some formal method of configuration and change management is therefore needed.

When planning deployment, you should consider the following important issues:

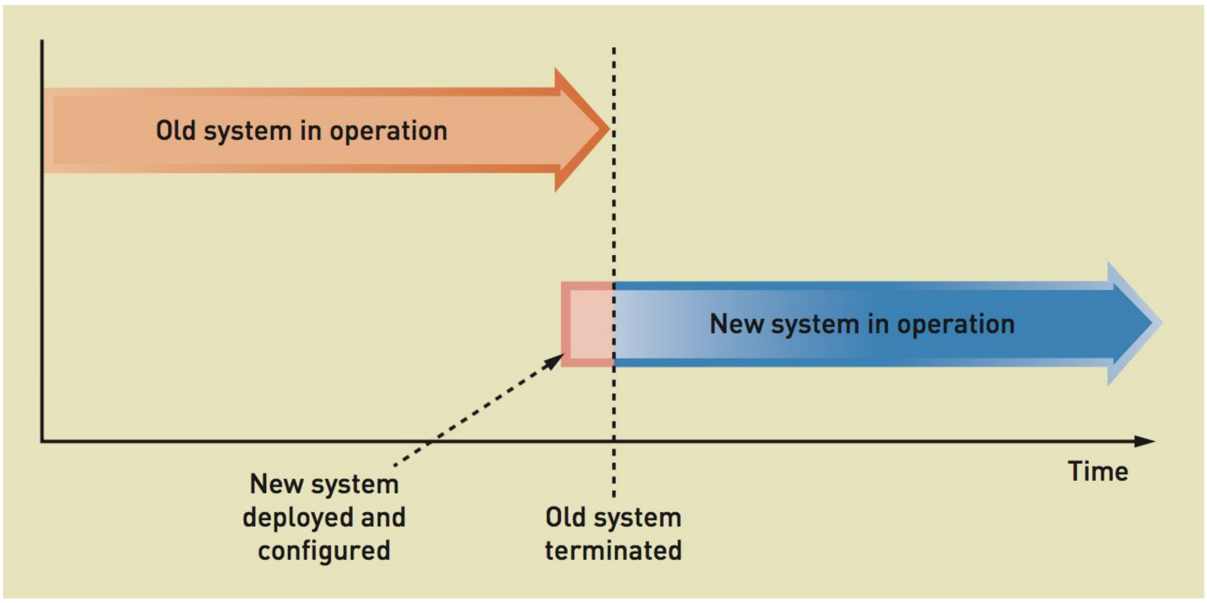
- The cost that might be incurred by operating both systems in parallel.
- Finding and correcting errors in the new system
- Disruptions that might be caused to the company
- Familiarising customers with the new procedures and training users.

There are three different approaches to deployment:

1. Direct deployment
2. Parallel deployment
3. Phased deployment

Direct Deployment

With the direct deployment approach, the new system is quickly installed and made operational and the old overlapping system is turned off. For a brief time, while the new system is being installed and tested, both systems operate concurrently. The advantage of using direct deployment is that it is simple. There are fewer logistical issues to manage and fewer resources required because the old and the new systems do not operate in parallel. The disadvantage of direct deployment however is the risk involved. There is no backup in the event that the new system fails because the old system is not operated in parallel. The amount of risk involved depends on the nature of the system, the cost of workarounds in the event of a system failure, and the cost of system unavailability or non-optimal system function.



Direct Deployment (John W. Satzinger et al., 2000)

Parallel Deployment

With parallel deployment both the old and new systems are operated in parallel. This usually done over an extended period of time. The old system should continue to operate until the new system has been tested thoroughly, is error-free and ready to operate on it own without the old system. The time allocated for both the systems to be operated in parallel should be determined in advance and limited in order to minimise the cost of operating two systems at the same time. The advantage of parallel deployment list the low operational risk. This is as the old system functions as a backup for the new system and failure in the new system can be reduced by using the old system.

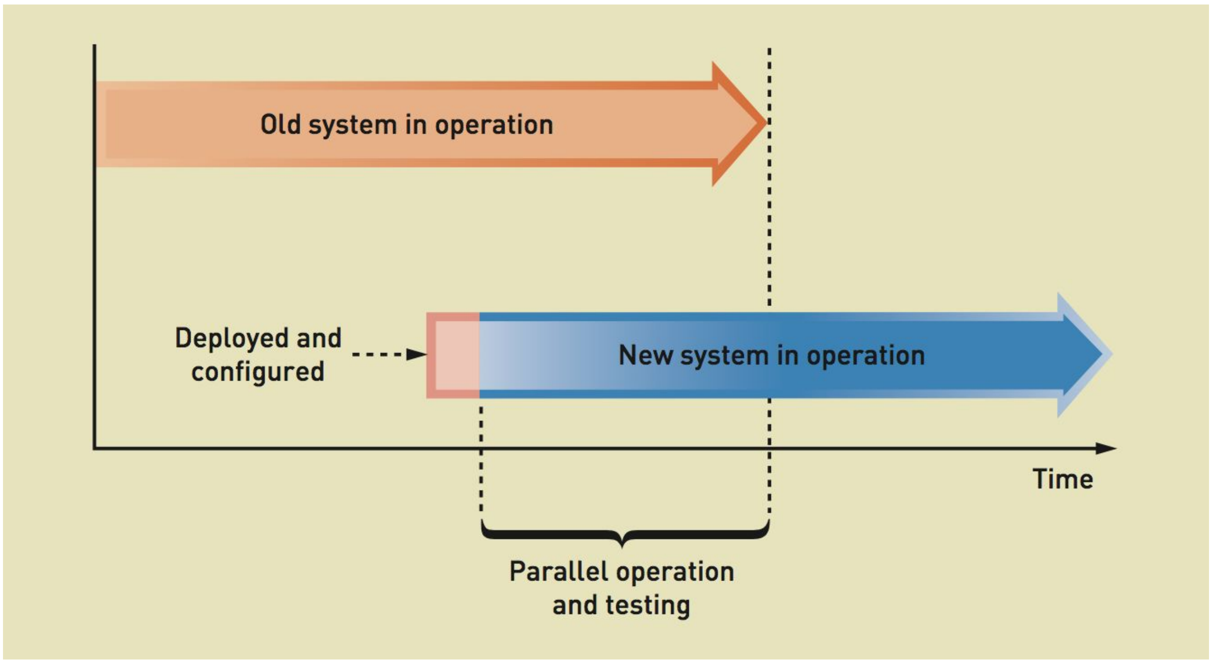
The cost of parallel deployment is the main disadvantage. The company has to pay to operate both the old and new systems during the period where they operate in parallel. Generally it is best to use parallel deployment consequences of a system failure are severe as, through redundant operation, it reduces the risk of a system failure significantly. Full parallel operation is impractical for a number of reasons however. Some of these reasons include:

- It may not be possible to use the inputs to one system in the other system and it may not be possible to use both types of inputs.

- The new system may use the same equipment as the old system and there may be insufficient capacity to operate both systems at the same time.
- There may not be enough staff to operate or manage both systems.

A partial parallel operation may be employed when full parallel operation is not possible. Possible forms of partial parallel operation are:

- Only process a subset of input data in one of the systems.
- Only perform a subset of processing functions
- Perform a combination of data and processing function subsets

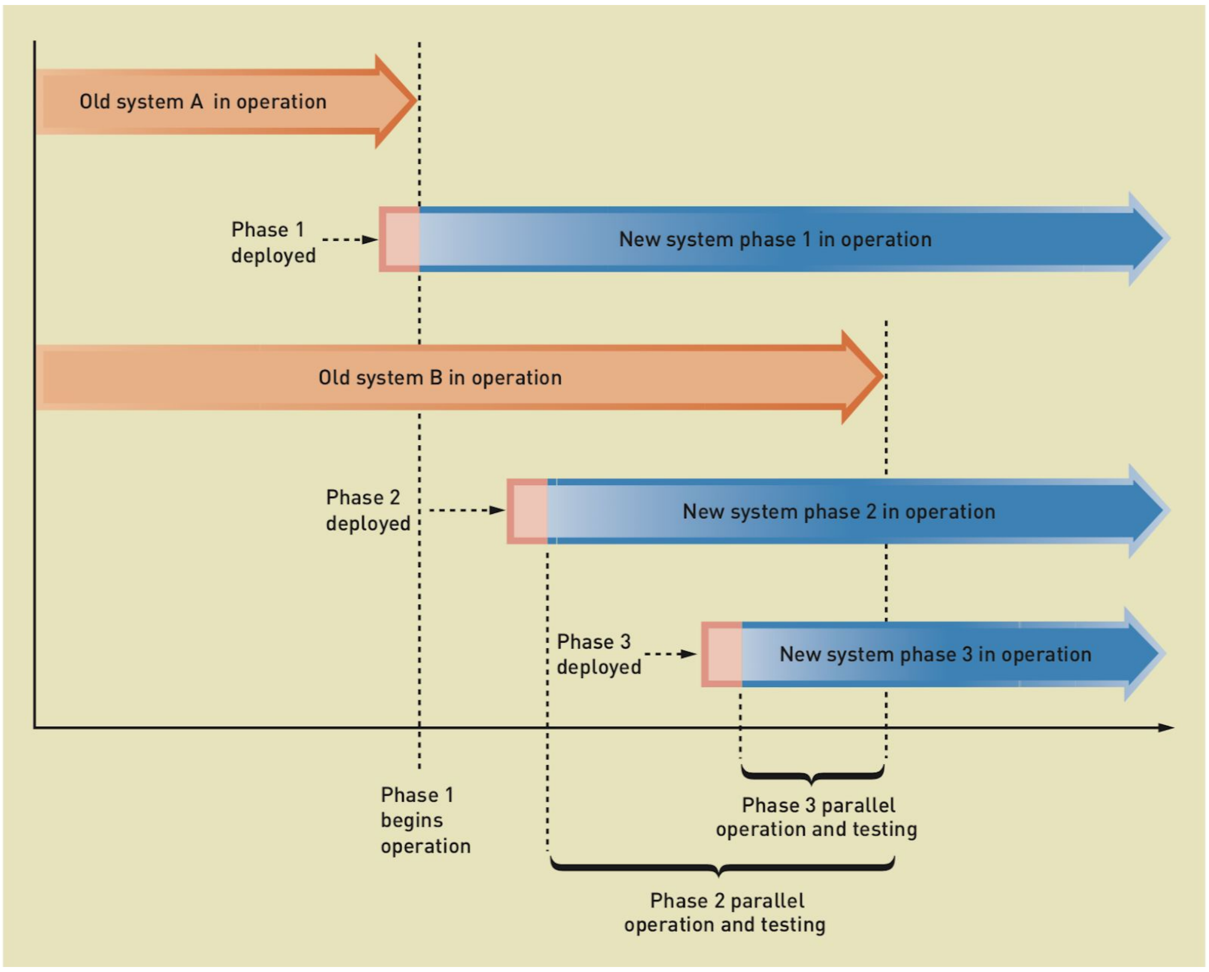


Parallel Deployment (John W. Satzinger et al., 2000)

Phased Deployment

With phased deployment, the system is deployed in a series of steps or phases. Each successive phase adds components or functions to the system. The system is tested during each phase to ensure that it is ready for the next phase. Phased deployment can be used in combination with parallel deployment. This is especially useful when the new system aims to take over the operation of multiple existing systems.

The advantage of using phased deployment is that there is a reduced risk because failure of a single phase is not as catastrophic as failure of an entire system. The disadvantage of this approach to deployment is increased complexity. By dividing the deployment into phases, more activities and milestones are created. This makes the entire process more complex. Each of the phases however, contains a smaller and more manageable set of activities. For extremely large or complex systems that cannot be installed at one time, the reduced risks of phased deployment outweigh the increased complexity.


Old system A in operation
Phase 1 deployed
New system phase 1 in operation
Old system B in operation
Phase 2 deployed
New system phase 2 in operation
Phase 3 deployed
New system phase 3 in operation
Phase 1 begins operation
Phase 3 parallel operation and testing
Phase 2 parallel operation and testing

Phased Deployment (John W. Satzinger et al., 2000)

Software Maintenance

Software will evolve over time regardless of its size, complexity or application domain. This is due to change that occurs when errors are corrected, the software is adapted to a new environment, a customer requests new functions, and when the application is reengineered to provide benefit in a modern context. Software maintenance is the process of changing a system after it has been delivered. It is an ongoing activity that occurs throughout an applications lifecycle.

There are four types of software maintenance:

1. Corrective Maintenance: This type of maintenance involves diagnosing and correcting errors in an operational system.

2. Adaptive Maintenance: This type of maintenance involves adding new features or functions to an operational system and making the system easier to use. The need for adaptive maintenance usually arises from changes in the business environment.

3. Perfective Maintenance: This type of maintenance involves changing an operational system to make it more efficient, reliable, or maintainable. Users normally request corrective and adaptive maintenance, while the IT department usually initiates perfective maintenance.

4. Preventive Maintenance: This type of maintenance aims to avoid future problems by analysing areas where trouble is likely to occur. Like perfective maintenance, the IT department normally initiates preventive maintenance.

Examples of maintenance tasks for each maintenance type:

Corrective Maintenance:

- Diagnose and fix logic errors
- Restore proper configuration settings
- Debug the program code

- Update drivers

Adaptive Maintenance:

- Add online capability
- Add mobile device support
- Add a new data entry field to an input screen
- Create a portal for employees

Perfective Maintenance:

- Replace outdated hardware
- Compress files in the system
- Install a more powerful network server
- Upgrade wireless network capability

Preventative Maintenance:

- Install antivirus software
- Develop a backup schedule
- Tighten cable connections
- Analyse problem reports for patterns

Cost of Maintenance

Software maintenance takes up a higher proportion of IT budgets than developing a new application. Roughly two-thirds of the budget goes to maintenance while the other third goes to development. Also, more of the maintenance budget is spent on implementing new requirements than on fixing bugs.

The relative costs of maintenance and new development vary from one application to another. The maintenance costs for business applications are compatible with system development costs, however maintenance costs can be up to four times more than development costs for embedded real-time systems.

Generally it is more cost-effective in the long run to invest a lot of effort into designing and implementing a system to reduce the cost of future changes. It is expensive to add new functionality to the system after it has been delivered because you have to spend time learning the system and analysing the impact the proposed changes will have on it. Developing software that is easy to understand and change therefore is likely to reduce evolution costs in the future. Good software engineering practices and techniques contribute to the reduction of maintenance costs.

The more effort that is expended during the development process to produce a maintainable system, then more the overall lifetime costs of the system will decrease. This is why refactoring your code is so important. Without refactoring, code becomes more and more difficult and expensive to change.

Software Reengineering

Many systems, are difficult to understand and change. This is especially true for older **legacy systems**. This can be because programs may have been optimised for performance or space utilization at the expense of understandability, or, the initial program structure may have been corrupted by a series of changes over time.

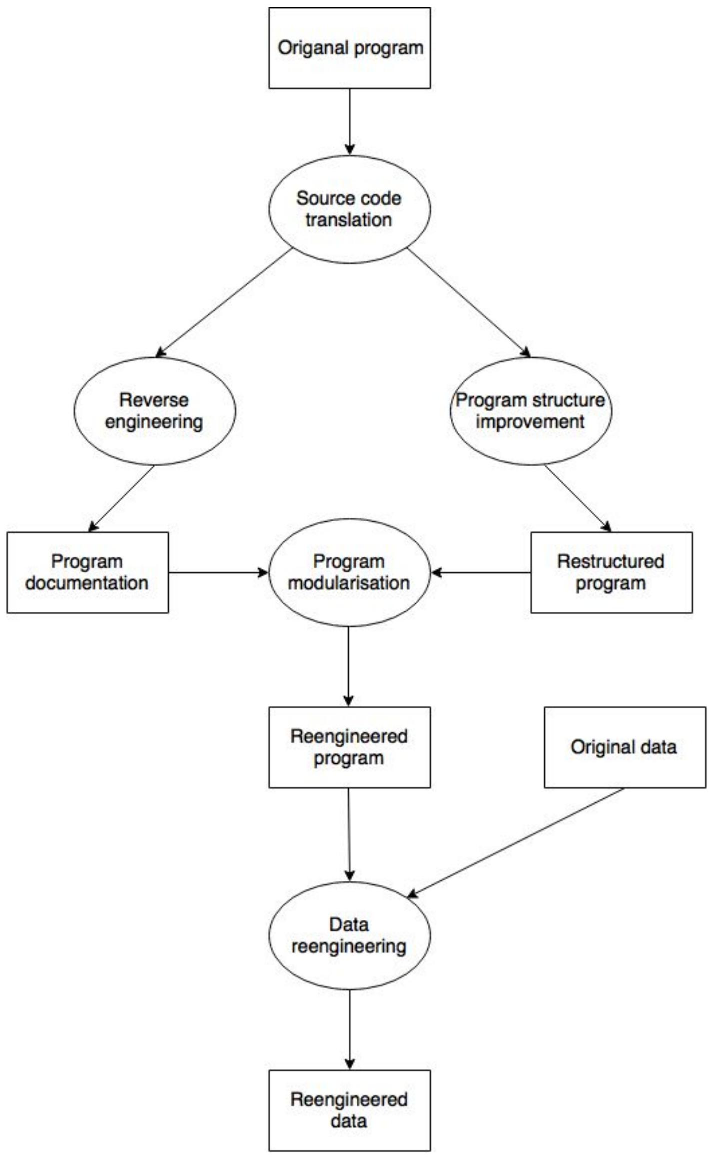
In order to make legacy software systems easier to maintain, you need to reengineer them to improve their structure and understandability. Reengineering can involve re-documenting the system, refactoring the system architecture, translating programs to a modern programming language, and modifying and updating the structure and values of the system's data. The overall functionality of the software should remain the same however and you should avoid making changes to the aritchure of the system.

The advantages of reengineering a system and not simply replacing it are:

1. Reduced risk: Developing software that is critical to a company is high risk. There may be errors in the system specification or they may development problems. There might also be delays in introducing new software that could result in extra cost being incurred or business being lost.

2. Reduced cost: It costs less to reengineer a system then to develop a new one.

The process of reengineering a system is shown in the diagram below:



The reengineering process

The activities in reengineering process are discussed in more detail below:

1. Source code translation: The program is converted from an old programming language to a more modern version of the same language or to a completely different language. This is done using a translation tool.

2. Reverse engineering: Information is extracted from the program by analysing it. This helps to document the programs organisation and functionality. This is completely automated.

3. Program structure improvement: In order to make the program easier to read and understand, the control structure of the program is analysed and modified. This can be partially automated.

4. Program modularization: Related parts of the program are grouped together and redundancy is removed where appropriate. This stage may involve architectural refactoring. Program modularization is a manual process.

5. Data reengineering: The data processed by the program is changed to reflect program changes. This can mean redefining database schemas and converting existing databases to the new structure. The data should also be cleaned up by finding and correcting mistakes, removing duplicate records, etc. There are tools that are available to support data reengineering.

Reengineering a program might not actually require all of the steps listed above. If you are still using the applications programming language, for example, you do not need source code translation. Recovering documentation through reverse engineering may also be unnecessary if you can do all reengineering automatically. Data reengineering is also only required if the data structures in the program change during system reengineering.

To help a reengineered system to work reliably with with a new software adapter services might have to be developed. Adaptor services hide the original interfaces of the software system and present new interfaces that are better structured which can be used by other components.

There are practical limits to how much you can improve a system by reengineering. For example, you can convert a system written using a functional approach to an object-oriented one. You cannot carry out major architectural changes or extreme reorganisation of the system data management automatically so they are therefore very expensive. Reengineering can improve maintainability however a reengineered system probably will not be as maintainable as a new system developed using modern software engineering methods.

Preventative Maintenance by Refactoring

Refactoring consists of modifying a program to improve its internal structure, to reduce its complexity or make it easier to understand while not changing its functionality. Some people consider refactoring to be a object-oriented development concept but the principles can be applied to any development approach. You should think of refactoring as preventive maintenance that can reduce problems caused by future change.

The purpose of reengineering and refactoring is the same; they are both intended to make software easier to understand and change. They are not the same thing however. Reengineering is done after a system has been maintained for some time and costs of maintenance are increasing. A system is reengineered using automated tools to create a new, more maintainable system. Refactoring is a continuous process that improves the system throughout the development and evolution process. It is used to avoid the degradation of structure and code which increases the costs and difficulties of maintaining a system.

Agile methods, such as extreme programming, are based around change. The program quality is therefore liable to degrade quickly. Refactoring is therefore an essential part of agile methods. Risk of introducing new errors through refactoring is reduced in agile methods because of their emphasis on regression testing. If a previously successful test that fails indicates that errors were introduced. Refactoring does not only have to be used with agile development however. It can be used with any approach to development.

We have discussed refactoring and “bad smells” which are the stereotypical situations in which the code can be improved in a previous task so we will not discuss them here.

Refactoring that is carried out during program development is a good way to reduce long term maintenance costs. If you have a program whose structure has been significantly degraded then it may not be possible to refactor the code alone.

Compulsory Task

Answer the following questions:

- List possible sources of data used to initialize a new system database. Briefly describe the tools and methods used to load initial data into the database.
- How do user documentation and training activities differ between end users and system operators?
- Briefly describe direct, parallel, and phased deployments. What are the advantages and disadvantages of each deployment approach?
- Describe four types of system maintenance and provide two examples of each type not listed in the task.
- What are the main factors that affect the costs of reengineering a system?

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.