TASK

Deployment and Maintenance Best Practice

Visit our website

Introduction

WELCOME TO THE DEPLOYMENT AND MAINTENANCE BEST PRACTICE TASK!

In this task, we cover the processes of getting software out of the hands of the developers into the hands of the end-users. We also look at 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 an expert code reviewer 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 a code reviewer. You can also schedule a call or get support via email.

Our expert code reviewers are 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 the 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

When you get a new cellphone, it's nice to have all your data transferred from your old one to help with a more seamless transition. Similarly, it can make for a much easier switch to a new system if the old data is available to access. To do this, before the new system is operational, existing data needs to be converted. This can be done either by reusing the current database or creating a new one where the old data is transferred.

- **Reusing an Existing Database:** This is the cheapest and most efficient choice, mostly because it is possible for a DBMS to make any necessary changes, such as adding new tables, or adding/modifying attributes.

- **Reloading Databases:** Where the changes needed are too complicated to manipulate the existing database, we can create a new one for the new system and convert and transfer data to it. Once this process is complete, the old database is then deleted.

Occasionally, it is not necessary to transfer data to a new system. In this case, we simply create an empty database for the system where future data will be stored.

TRAINING USERS

Ideally, training should take place once the test version of the system has been fully debugged. However, because of time constraints, this is usually not possible.

Therefore, training usually begins when the user interfaces are stable. There are two types of users that need to be trained: end-users and the system operators.

The end-users only work with the front end. They typically (Satzinger, Jackson, & Burd, 2010, p. 650):

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

System operators, on the other hand, need to know how to run the system. They typically (Satzinger, *et al.*, 2010, p. 650):

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

End-User Training

If you think of your cellphone, you do not need to know the inner workings and mechanisms of it in order to operate it — you just need to know how to use the phone for its intended purposes. How would you teach someone to use a phone? The process can be tricky because people have different levels of experience and knowledge — teaching someone to use a phone after an upgrade is different from teaching someone who has never used one before. The same applies when teaching end-users how to use a new system. The most effective form of teaching is practical lessons to teach the most useful aspects of the system for each person. This can be done in groups or one on one, but should ideally be face-to-face. Manuals also need to be comprehensive enough for the end-user to grasp the concept, but not so in-depth that it becomes confusing.

System Operator Training

When the system operators aren't also end-users, training can be much less formal because they will already have an in-depth understanding of the system. This means that most of the training can be conducted through self-study. There are also usually fewer system operators than end-users, which means training is much easier to conduct.

Before any formal training begins, documentation and other training materials are developed. As you learned in Introduction to Software Engineering, 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

Due to the information they contain, these are important materials that are used in the training process. If you need a refresher, go back to your *Software Documentation* task in level 2.

CONFIGURING THE PRODUCTION ENVIRONMENT

In order to configure your environment, you need to set up software and hardware dependencies. Hardware dependencies will include minimum system requirements such as CPU and hard drive space. Software requirements will include drivers, Java runtime environment (JRE) or certain server software (e.g. Apache or MySQL). You will also need to ensure that the file structure is correct; if your program expects certain data files to be stored in a certain location, they need to be there.

PACKAGING, INSTALLING AND DEPLOYING COMPONENTS

Packaging includes compressing the components to make them easily downloadable, while installing deals with extracting the components from the packaging.

When you install the software, it needs to run checks to see everything is configured correctly e.g. adding to path variable, creating directories, downloading additional software, checking for and uninstalling previously installed versions.
When planning deployment, there are factors that need to be considered, such as the cost of deployment, making sure that the system has been fully debugged, the potential disruptions to the company when the new system is put in place, and getting users comfortable with new procedures as a result of the new system.

There are three approaches to deployment, which are each discussed below:

- **Direct Deployment:** In this approach, the old system is turned off and the new system is quickly installed and made operational.
- **Parallel Deployment:** In this approach, the old and new systems operate in parallel when the new system is first introduced. After a while, once the new system is stable, the old system is switched off.
- **Phased Deployment:** In this approach, the new system is deployed in phases while the old system is still running. Once the new system is fully integrated and stable, the old system is switched off.

SOFTWARE MAINTENANCE

Software will evolve over time regardless of its size, complexity or application domain. This is due to the 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 re-engineered 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 application's lifecycle.

There are four types of software maintenance (Quezada, 2017):

1. **Corrective Maintenance:** This type of maintenance involves diagnosing and correcting errors in an operational system. For example:
   a. Diagnose and fix logic errors
   b. Restore proper configuration settings
   c. Debug the program code
   d. Update drivers

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. For example:
   a. Add online capability
   b. Add mobile device support
   c. Add a new data entry field to an input screen
   d. Create a portal for employees

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. For example:

   a. Replace outdated hardware
   b. Compress files in the system
   c. Install a more powerful network server
   d. Upgrade wireless network capability

4. **Preventive Maintenance:** This type of maintenance aims to avoid future problems by analysing areas where the trouble is likely to occur. Like perfective maintenance, the IT department normally initiates preventive maintenance. For example:
   a. Install antivirus software
   b. Develop a backup schedule
   c. Tighten cable connections
   d. 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. Therefore, developing software that is easy to understand and change is likely to reduce evolution costs in the future. Good software engineering practices and techniques contribute to the reduction of maintenance costs.

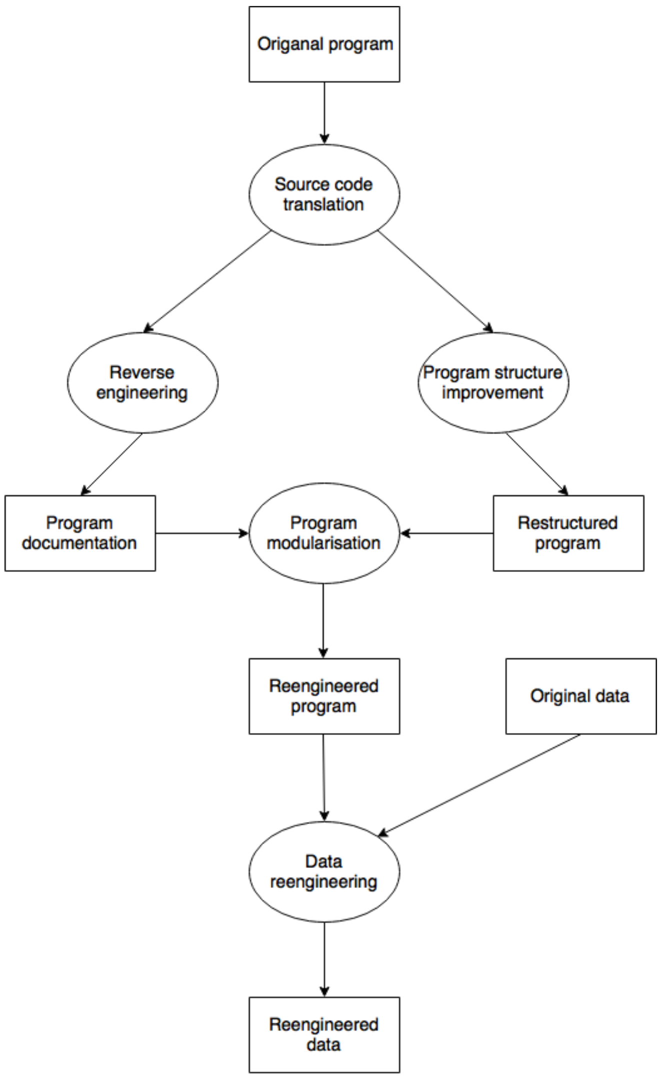
The more effort that is expended during the development process to produce a maintainable system, the more the overall lifetime costs of the system will decrease. This is why refactoring your code is so important. Without refactoring, code becomes more and more difficult and expensive to change.

SOFTWARE REENGINEERING

Many systems are difficult to understand and change because programs may have been optimised for performance or space utilisation 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 older software systems easier to maintain, you need to re-engineer them to improve their structure and understandability. According to Sommerville (2016), re-engineering 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. As you can imagine, re-engineering an existing system is far more cost-effective for a company as time and money are not spent on creating an entirely new system and training people on how to use it.

Sommerville's diagram of the process of reengineering a system is shown in the diagram below:



The reengineering process (Sommerville, 2016, p. 277)

Bear in mind that because the re-engineering process will differ depending on the system itself (Byrne, & Gustafson, 1992), not all of the above steps need to be followed. Nevertheless, let us take a look at each segment of the diagram in more detail below:

1. **Source code translation:** The program is converted to a modern version of the same language or a different language altogether.

2. **Reverse engineering:** Information from the old system is analysed so that its functionality can be fully understood. This will then guide the re-engineering process to make sure no functionality is lost.

3. **Program structure improvement:** The structure of the program is modified to improve readability and understandability.

4. **Program modularisation:** The system is broken down into segments and related segments are grouped together. This helps to check for any redundancies in the system to increase efficiency.
5. **Data re-engineering:** The data processed by the program is neatened up and modified so that it will still be processed smoothly with the new system changes. Databases could also be modified to reflect program changes.

Unfortunately, there are practical limits to how much you can improve a system by re-engineering. This is because there isn't the freedom to create an entirely new product — you are confined by the existing system. If you want to make bigger changes, this could cost extra time and money. Re-engineering can improve maintainability, however, a re-engineered system will likely not be as maintainable as a system created anew with 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 an 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. However, they are not the same thing. 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 so refactoring is an essential part of agile methods. The risk of introducing new errors through refactoring is reduced in agile methods because of their emphasis on regression testing. If a previously successful test fails, it 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.

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

Create a text file called **deploymentMaintenance.txt** and answer the following questions:

- Do some research about the different approaches to deployment. What are the pros and cons of each? When would you use each approach?
- Answer the following scenarios:
    - A second-hand book shop is converting from a paper system to digital, and they've asked you to develop their catalogue software. After finishing, how do you deploy it and why?
    - A large retail chain has asked you to develop a more modern set of tools for their use, including point-of-sale systems, stock records, and customer service systems. After finishing, how do you deploy it and why?
    - You have developed an alternate fire alarm system for a large shopping centre. This system's reliability is of utmost importance. After finishing, how do you deploy it and why?

Completed the task(s)?

Ask an expert to review your work!

Review work

Rate us

Share your thoughts

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

Think that 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.

References:

Byrne, E., & Gustafson, D. (1992). *A Software Re-engineering Process Model* [PDF]. Manhattan: Department of Computing and Information Sciences. Retrieved from https://ieeexplore.ieee.org/document/217608/metrics

Quezada, N. (2017). The 4 Types of Software Maintenance - Endertech. Retrieved 26 February 2020, from https://endertech.com/blog/maintenance-bug-fixing-4-types-maintenance

Satzinger, J., Jackson, R., & Burd, S. (2009). *Systems analysis and design in a changing world* (5th ed., pp. 616-659). Boston: Cengage Learning.

Sommerville, I. (2016). *Software Engineering* (10th ed., pp. 255-282). Essex: Pearson Education Limited.