



NATIONAL OPEN UNIVERSITY OF NIGERIA

FACULTY OF SCIENCE

COURSE CODE: CIT831

Module Introduction

This module introduces you to the basic principles and notions of software engineering. We will discuss topics like software engineering methodology, including software process and software process models. You will also get to know the professional and ethical demands of a software engineer.

- Unit 1: Software Engineering Methodology – Basic Notions

Unit 2: Professional and Ethical Responsibilities

Unit 3: Software Process

Software Process Models

Unit 5: Evolutionary and Incremental Development
- Unit 4:

Unit 1: Software Engineering Methodology – Basic Notions

Contents

1.0 Introduction	
2.0 Intended Learning Outcome (ILOs)	
3.0 Main Content	
3.1 Software Engineering Methodology	
3.2 Software Engineering	
3.3 Software	
3.4 Software Costs	
3.5 Software Engineering vs. popular computing disciplines	
3.6 Attributes of good software	4.0
Self-Assessment Exercise	5.0
Conclusion	6.0
Summary	7.0
Further Readings	



1.0 Introduction

This unit introduces students to the basic concepts of software engineering. It describes software and software engineering methodology, explaining the attributes of good software. The unit discusses the fundamental notions of software engineering methodology, thus guiding and facilitating your understanding of the subsequent units.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- define software engineering methodology
- describe software engineering
- explain what a software is
- list the attributes of a good software.



3.0 Main Content

3.1 Software Engineering Methodology

Software engineering methodology describes a framework for planning, executing, and managing the process of developing software systems.

3.2 Software Engineering

Software engineering is the application of systematic and quantifiable approach in the development, operation, maintenance; and retirement of software. As a discipline, it is an engineering discipline and a branch of computer science and engineering that is concerned with all aspects of software production. Software engineers adopt a systematic and organised approach to solving a software engineering problem. They also use appropriate tools and techniques in delivering software jobs.

3.3 Software

Software simply refers to computer programs including its associated libraries, data and documentation such as requirements, design models and user manuals. Software products may be developed for a particular customer or may be developed for a general market. It basically may be

1. Generic: developed to be sold to a range of different customers e.g. PC software such as Excel or Word.
2. Bespoke (custom): developed for a single customer according to their specification.

New software can be created by developing new programs, configuring generic software systems or reusing existing software.

3.4 Software Costs

Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost. Also software costs more to maintain than it does to develop. Thus software engineering is always concerned with cost-effective software development.

3.5 Software Engineering vs. popular computing disciplines

While computer science is concerned with theory and fundamentals; computer engineering deals more with hardware architecture and hardware-software integration. software engineering on the other hand is concerned with the practicalities of developing and delivering useful software. Figures 1.1, 1.2 and 1.3 show how each of software engineering, computer engineering and computer science occupy the problem space of computing. The horizontal range runs from Theory, Principles, Innovation on the left, to Application, Deployment, Configuration on the right. The vertical range runs from Computer Hardware and Architecture at the bottom, to Organizational Issues and Information Systems at the top.

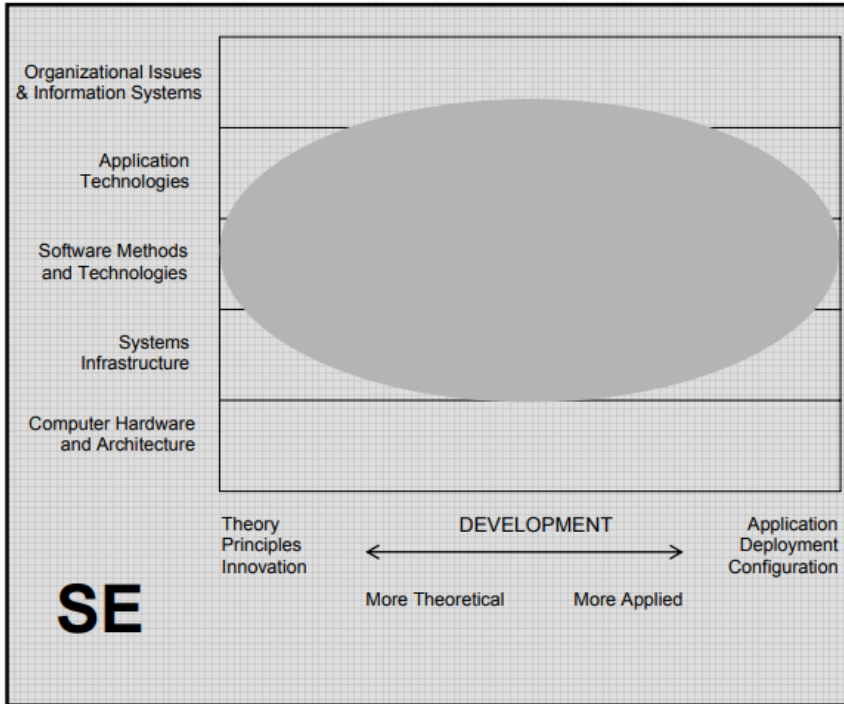


Figure 1.1: the problem space of software engineering

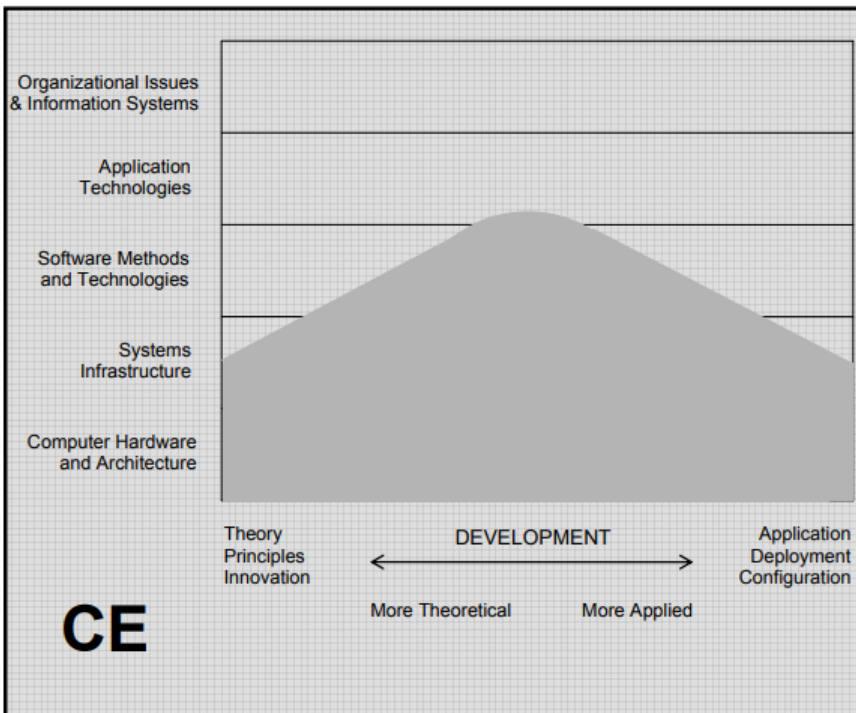


Figure 1.2: the problem space of computer engineering

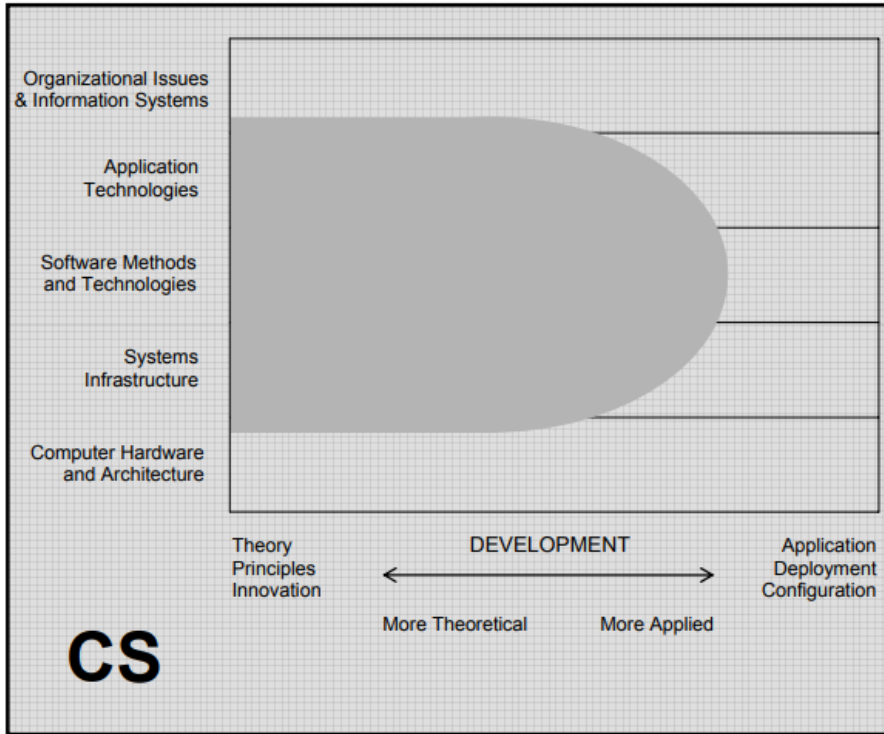


Figure 1.3: the problem space of computer science

3.6 Attributes of good software

A software product should deliver the required functionality and performance. Usually, the attributes would fall into the following categories;

a. Operational attributes

Budget: software must be cost-effective

Usability: software must be usable

Efficiency: software must be efficient and not waste system resources

Functionality: software must be functional and serve the purpose for its creation

Dependability: software must be trustworthy

Security: software must be secure

Safety: software must be safe to use

b. Transitional attributes

Portability: should be possible to move from one environment to another

Interoperability: should exchange and make use of information from other software systems

Reusability: should be possible to deploy software assets on different projects
Adaptability: should be adaptable to change in its environment and circumstances

c. Maintenance attributes

Modularity: should be divided into components

Maintainability: software system or component should be easy to correct faults

Scalability: should be upgradeable in order to meet new user requirements

Discussion

Briefly outline three ways a generic software might be different from a bespoke software.

Scenario

After using a certain desktop application on his friend's computer, a user ordered online for the same application for his computer, to enable him complete his word editing assignment. Would you classify this application as a generic or bespoke software?



4.0 Self-Assessment Exercise(s)

1. One of these is not an attribute of a good software
 - a. Dependability
 - b. Adaptability
 - c. Expensive
 - d. Usability
2. What describes a framework for planning, executing, and managing the process of developing software systems?
 - a. Software engineering
 - b. Software lifecycle
 - c. Software process
 - b. Software engineering methodology



5.0 Conclusion

In this unit, you have learnt about software engineering methodology. You have also been able to understand the difference between software engineering, computer engineering and computer science. Finally, you have also understood what software is and the attributes of a good software.



6.0 Summary

What you have learnt borders on the basic idea of software engineering. The subsequent units shall build upon these fundamentals.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component?* *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications,*” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability,*”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jermaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A, (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Vijayasarathy, L. R. and Butler, C. W. (2016). *Focus: The Software Engineering Process*. Colorado state university.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-thewatersluice-t573.html#754>

http://stg-tud.github.io/eise/WS11-EiSE-03-What_is_Software_Engineering.pdf

https://web.archive.org/web/20141021153204/http://www.acm.org/education/curric_vols/CC2005-March06Final.pdf

https://ftms.edu.my/v2/wp-content/uploads/2019/02/csca0101_ch07.pdf

Unit 2: Professional and Ethical Responsibilities

Contents

1.0 Introduction	
2.0 Intended Learning Outcome (ILOs)	
3.0 Main Content	
3.1 Professional and Ethical Responsibilities	
3.2 Areas of Professional Responsibility	
3.3 ACM/IEEE Code of Ethics	
4.0 Self-Assessment Exercise	
5.0 Conclusion	
6.0 Summary	7.0
Further Readings	



1.0 Introduction

In this unit, you will understand your professional and ethical responsibilities as a software engineer. You will also understand the IEEE-CS/ACM code of ethics.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you should be able to:

Describe the professional and ethical responsibility of a software engineer

Outline and explain the eight principles of IEEE-CS/ACM code of ethics



3.0 Main Content

3.1 Professional and Ethical Responsibilities

Software engineering involves greater responsibilities than simply the application of technical skills. They must be honest and ethically responsible to be respected as professionals. Similarly, ethical behaviour is more than simply upholding the law.

3.2 Areas of Professional Responsibility

Confidentiality: software Engineers should normally respect the confidentiality of their employers or clients, irrespective of whether or not a formal confidentiality agreement exists.

Competence: software Engineers should not misrepresent their level of competence. They should not intentionally accept jobs outside their areas of competence.

Intellectual property rights: software Engineers should be aware of local laws governing the use of intellectual property like patents, copyright, etc. They should always ensure that the intellectual property of employers and clients is protected.

Computer misuse: Software engineers should not use their technical skills to misuse other people's computers. Computer misuse ranges from simple misuse such as game playing on an employer's machine, to very serious misuse like spread of computer viruses.

3.3 ACM/IEEE Code of Ethics

The Association of Computing Machinery(ACM) and the Institute of Electrical and Electronics Engineering (IEEE) have jointly produced a code of ethical practice for software engineers. The IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practices was responsible for developing these set of expected ethical behaviours from software engineers. The expected code of ethical behaviours were listed under the following eight principles:

1. **Public** - Software engineers shall act consistently with the public interest.
2. **Client and Employer** - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. **Product** - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. **Judgment** - Software engineers shall maintain integrity and independence in their professional judgment.
5. **Management** - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. **Profession** - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. **Colleagues** - Software engineers shall be fair to and supportive of their colleagues.
8. **Self** - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Discussion

In your understanding, discuss the benefits of code of ethics to the software engineer.

Scenario

As the chief executive officer of a large software company, your client's competitor has approached you with good offer to sell a custom made software product you developed for your client to them. What would you do?

Is there any area of the code ethics you would contravene if you do the bidding of your client's competitor?



4.0 Self-Assessment Exercise(s)

1. one of these is not one of the eight principles of IEEE-CS/ACM code of ethics

- a. public
- b. commercial
- c. product
- d. profession

2. what area of responsibility would require a software engineer not divulge very important information about their client?

- a. Confidentiality
- b. secrecy
- c. repudiation
- d. intellectual property rights



5.0 Conclusion

In this unit you have learnt about the issues of professional responsibility.



6.0 Summary

What you have learnt in this unit concerns issues of professional responsibility.



7.0 Further Readings

ACM/IEEE (2001). *Computing curricula 2001, computer science, final report*. The Joint Task Force on Computing Curricula, IEEE Computer Society and Association for Computing Machinery.

ACM/IEEE (1999). *Software engineering code of ethics and professional practice*, (v5.2). Retrieved 28th June, 2020 from <https://ethics.acm.org/code-of-ethics/software-engineering-code/>

Bott, F. (2005). *Professional Issues in Information Technology*. Swindon, UK: British Computer Society.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*, Morgan Kaufmann Publishers: San Francisco, CA.

Duquenoy, P. (2007). *Ethical, Legal and Professional Issues in Computing*. London. Thomson Learning.

Génova, G., González, M. R. and Fraga, A. (2007). Ethical Education in Software Engineering: Responsibility in the Production of Complex Systems. *Science and Engineering* **13**, pp 505–522

Génova, Gonzalo; González , M. R. and Fraga, Anabel (2014). Ethical Responsibility of the Software Engineer

Johnson, D. G. (2001). *Computer Ethics*. Englewood Cliffs, NJ: Prentice-Hall.

Laudon, K. (1995). Ethical Concepts and Information Technology. *Comm. ACM* 38 (12): 33–39. doi:10.1145/219663.219677.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology*, (5th Edition). McGraw-Hill

Unit 3: Software Process

Contents

1.0 Introduction	
2.0 Intended Learning Outcome (ILOs)	
3.0 Main Content	
3.1 Software Process	
3.2 Process Activities	
3.3 Software Specification	
3.4 Software Development (Design and Implementation)	
3.5 Software Validation	
3.6 Software evolution	
4.0 Self-Assessment Exercise	
5.0 Conclusion	
6.0 Summary	7.0
Further Readings	



1.0 Introduction

In this unit, you will learn about software process and software process activities.



2.0 Intended Learning Outcomes (ILOs)

At the end of this unit, you will be able to:

describe a software process
outline and describe the software process activities.



3.0 Main Content

3.1 Software Process

Software process, also known as Software Development Life Cycle refers to well-defined and structured sequence of stages and activities in software engineering to develop the intended software product.

3.2 Process Activities

The stages (activities) in a software process include the following:

Software Specification: what the system should do and its development and operational constraints

Software Development: production of the software system according to specification

Software Validation: checking that the software product has the customer specified functionalities

Software Evolution: changing the software in response to changing user and work demands.

3.3 Software Specification

This is the process of establishing what services are required and the constraints on the system's operation and development.

3.4 Software Development (Design and Implementation)

This is the process of converting the system specification into an executable system. Software design involves designing a software structure that realises the specification; while implementation involves translating this structure into an executable program.

3.5 Software Validation

Software validation is concerned with showing that a software product conforms to its specification and that the product meets the expectations of the customer. It involves checking the processes at each stage of the software process. The majority of validation costs are incurred after implementation when the operation of system is tested.

The software may be tested in three stages namely: the system component, where each component of the system is tested; the integrated system, some groups of components of the system is tested as a unit; and the whole system, where the overall functionality of the system is tested.

3.6 Software evolution

The aim of software evolution is to implement the possible major changes to the system. The existing larger system is never complete and continues to evolve, thereby making the system more complex. The objectives of software evolution are to ensure the reliability and flexibility of the system.

Discussion

Clearly distinguish between software process and software engineering methodology.

Scenario

A client has approached your software company to develop a certain software for them. Clearly state how you would begin this assignment in accordance with the software process activities.



4.0 Self-Assessment Exercise(s)

1. Software process is also known as
 - a. Software development lifecycle
 - b. Software engineering
 - c. Software engineering methodology
 - d. Software evolution
2. One of these is not part of the activities of a software process
 - a. Software specification
 - b. Software evolution
 - c. Software maintenance
 - d. Software validation



5.0 Conclusion

In this unit you have learnt about the software process and software process activities.



6.0 Summary

The software process activities include; Software Specification, Software Development, Software Validation and Software Evolution.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component? Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A., (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.fretechbooks.com/software-engineering-methodology-thewatersluice-t573.html#754>

http://moodle.autolab.uni-pannon.hu/Mecha_tananyag/szoftverfejlesztési_folyamatok_angol/ch03.html#d0e550

https://www.tutorialspoint.com/software_engineering/software_engineering_tutorial.pdf

Unit 4: Software Process Models

Contents

1.0 Introduction	
2.0 Intended Learning Outcome (ILOs)	
3.0 Main Content	
3.1 Software Process Models	
3.2 The Waterfall Model	
3.3 Problems of the waterfall model	
3.4 The V-Model	
3.5 application of v- model	
3.6 Disadvantages of the V-Model	
4.0 Self-Assessment Exercise	
5.0 Conclusion	
6.0 Summary	7.0
Further Readings	



1.0 Introduction

In this unit, you will understand what a software process model is. You will also understand the waterfall model and V-Model.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you should be able to:

Define software process model

Explain the waterfall model

Distinguish between waterfall model and V-Model



3.0 Main Content

3.1 Software Process Models

A software process model is a simplified representation of a software process or software development lifecycle, presented from a specific perspective.

Examples of generic process models include:

Waterfall
Evolutionary development
Formal transformation
Integration from reusable components.

3.2 The Waterfall Model

The waterfall model, also referred to as a linear-sequential life cycle model was the first software process model to be introduced and is. It consists of separate and distinct phases of specification and development.

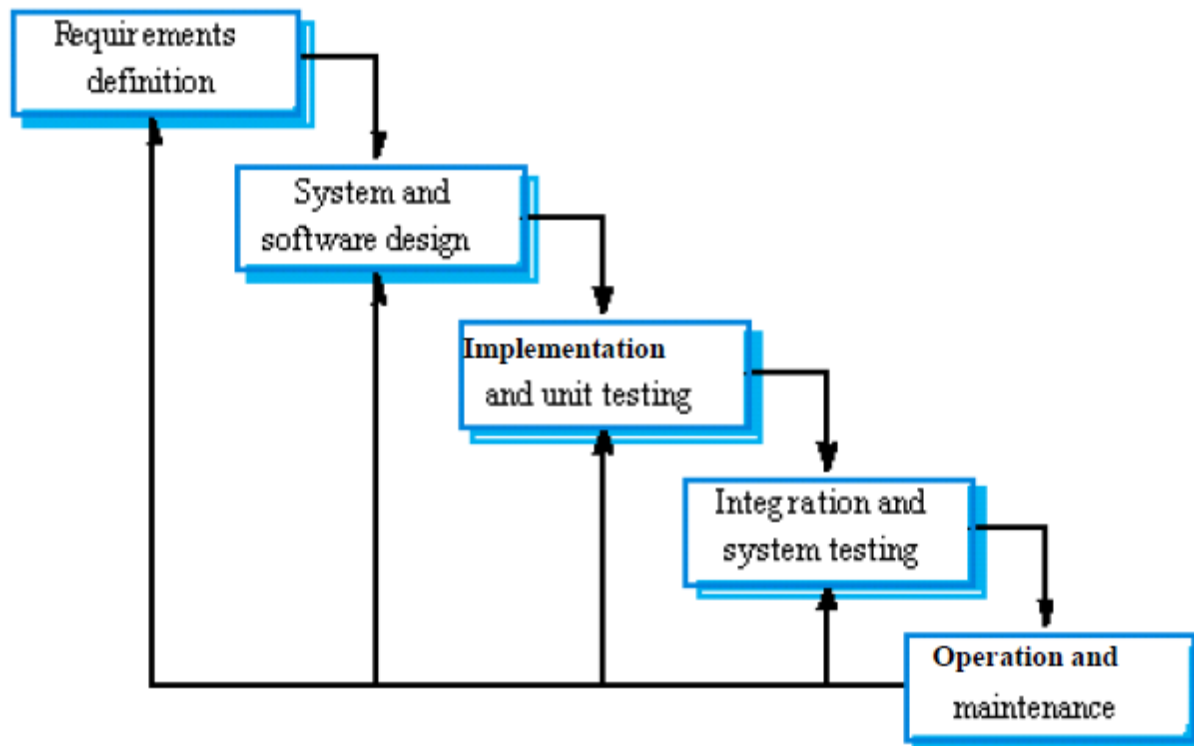


Figure 1.4: the waterfall model

The phases of waterfall model include

- a. Requirements analysis and definition
- b System and software design
- c. Implementation and unit testing
- d. Integration and system testing
- e. Operation and maintenance.

3.3 Problems of the waterfall model

1. Difficult to accommodate change after the process is underway. One phase has to be complete before moving onto the next phase.
2. Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
3. This model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
4. Few business systems have stable requirements.
5. The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.

3.4 The V-Model

The V-model is an improvement of the waterfall model. It provides means of testing of software at each stage in reverse manner. This is an improvement to the waterfall model which only allows you to move down in a linear way. Just like the waterfall model the process steps follow each other in a sequential order but V-model allows the parallel execution of activities. On each stage of the software development process, testing is carried out to validate and verify each stage of the process. This model is often referred to as verification and validation model

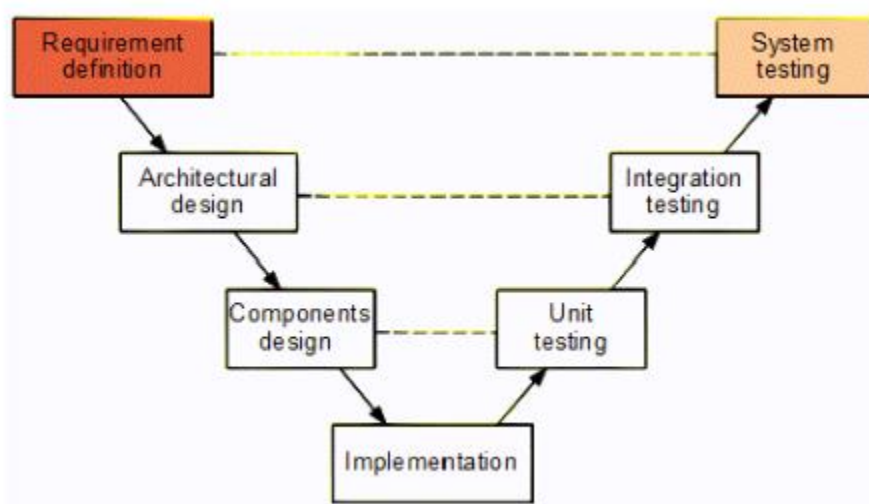


Figure 1.5: the V-Model

3.5 application of v- model

V-Model usually can be applied when;

- a. Requirements are well defined, clearly documented and fixed
- b. Product definition is stable.
- c. Technology is not dynamic and is well understood by the project team.

- d. There are no ambiguous or undefined requirements.
- e. The project is short.

3.6 Disadvantages of the V-Model

- a. High uncertainty and risk.
- b. Not a good model for complex and object-oriented projects.
- c. unsuitable for long and ongoing projects.
- d. unsuitable for the projects where requirements are at a moderate to high risk of changing.
- e. Once an application is in the testing stage, it is difficult to go back and change a functionality.

Discussion

Why is V-Model considered an improvement over waterfall model?

Scenario

You are the Chief Operating Officer of a software development company. A client has approached your team to develop a new software for them. On discussing with their team, you observe they do not have all the system requirements well documented and would rather prefer an arrangement where they keep adjusting system requirement until the software product is fully developed and functional. Why will either waterfall model or V-model be unsuitable for this job?



4.0 Self-Assessment Exercise(s)

1. which of these was the first software process model to be introduced
 - a. spiral development
 - b. Evolutionary development
 - c. waterfall model
 - d. V - model
2. Which of these is not a phase of the waterfall model
 - a. Requirements analysis and definition
 - b Reverse engineering
 - c. Implementation and unit testing

d. Integration and system testing



5.0 Conclusion

In this unit you have learnt about software process model. You have also learnt some software process models which include the waterfall model and V-Model.



6.0 Summary

waterfall model and V-model are software process models that require system requirements to be specified and fully documented before design and implementation can commence.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component?* *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A., (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freotechbooks.com/software-engineering-methodology-the-watersluice-t573.html#754>

https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm

<http://tryqa.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/>

Unit 5: Evolutionary and Incremental Development

Contents

1.0 Introduction	
2.0 Intended Learning Outcome (ILOs)	
3.0 Main Content	
3.1 Evolutionary Development	
3.2 Problems of Evolutionary Development	
3.3 Application of evolutionary development	
3.4 Incremental Development	
3.5 Advantages of Incremental Development	
4.0 Self-Assessment Exercise	
5.0 Conclusion	
6.0 Summary	7.0
Further Readings	



1.0 Introduction

In this unit, we will look at the evolutionary development and incremental development.



2.0 Intended Learning Outcomes (ILOs)

At the end of this unit, you will be able to:

- describe the objective of evolutionary development
- problems of evolutionary development
- outline the applications of evolutionary development
- point out the advantages of incremental development.



3.0 Main Content

3.1 Evolutionary Development

The main objective of the evolutionary development is to work with customers and to evolve a final system from an initial outline specification. This model starts with well-understood requirements and adds new features as proposed by the customer. The Evolutionary development model divides the development cycle into smaller, incremental waterfall models in which users are able to get access to the product at the end of

each cycle. Feedback is provided by the users on the product for the planning of the next cycle with the development team responding, often by changing the product, plan or process.

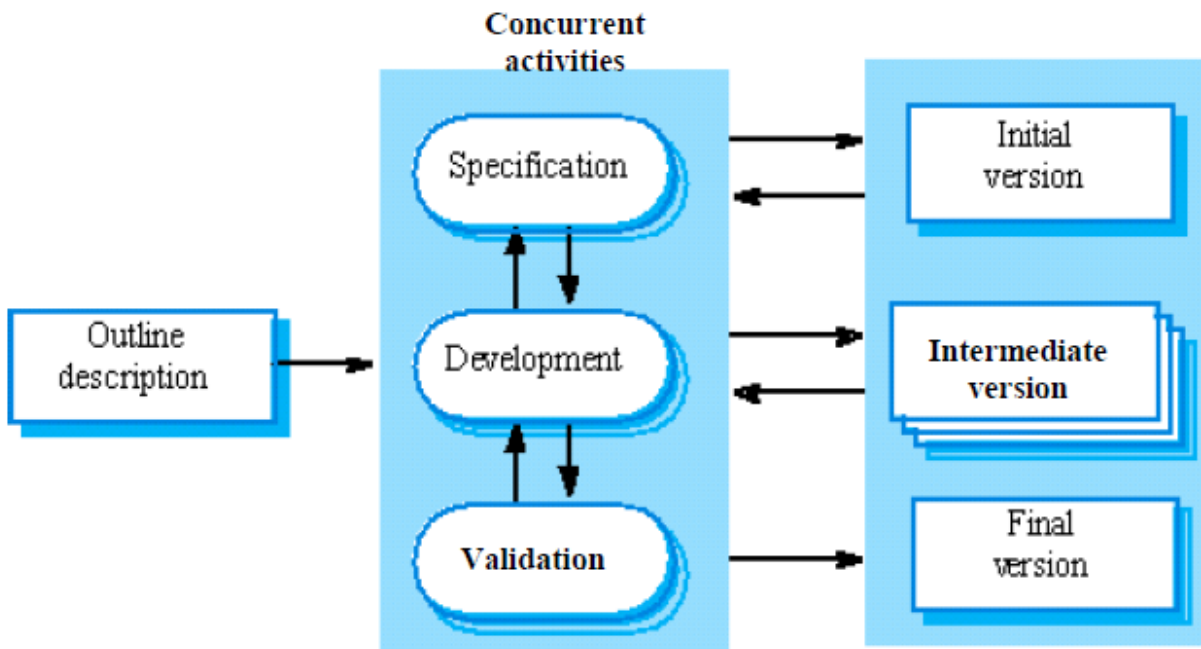


Figure 1.6: Evolutionary development

3.2 Problems of Evolutionary Development

- a. Lack of process visibility;
- b. Systems are often poorly structured
- c. Special skills (e.g. in languages for rapid prototyping) may be required.

3.3 Application of Evolutionary Development

- a. In small or medium-size interactive systems
- b. In parts of large systems (e.g. the user interface)
- c. In short-lifetime systems.

3.4 Incremental Development

In the incremental development rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.

User requirements are prioritised and the highest priority requirements are included in early increments. Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

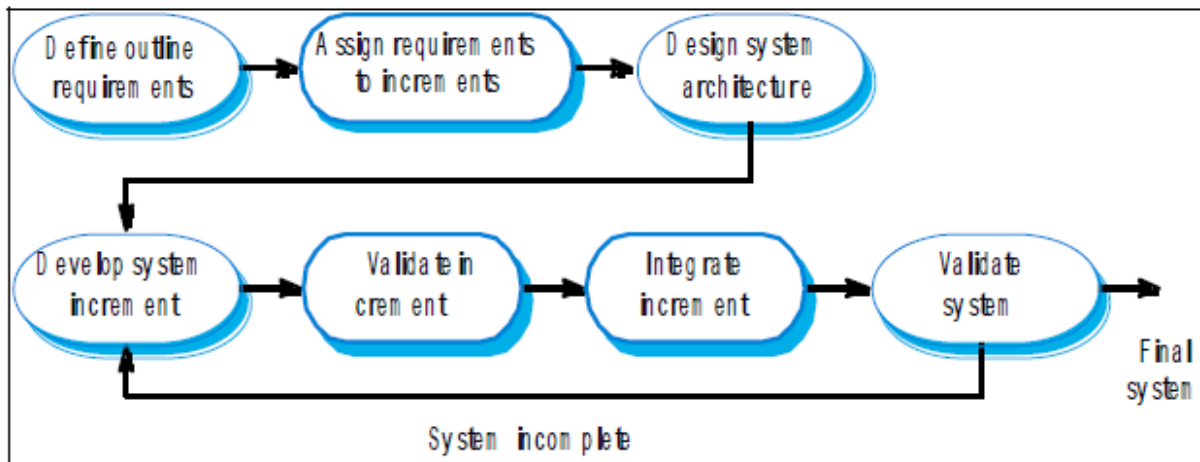


Figure 1.7 Incremental development

3.5 Advantages of Incremental Development

1. Customer value can be delivered with each increment so system functionality is available earlier
2. Early increments act as a prototype to help elicit requirements for later increments
3. Lower risk of overall project failure
4. The highest priority system services tend to receive the most testing.

Discussion

Why is evolutionary development unsuitable for big software development projects

Scenario

A small size company with twenty staff has requested for a Human Resource Management system. They have forecasted that staff strength will not grow beyond thirty persons, and require just a basic product that can handle the most simple human resource management activities.

If they are unwilling to provide a complete detail of their system requirement, what software methodology would you recommend for this type of project?



4.0 Self-Assessment Exercise(s)

1. which of the following process models has the objective of working with customers to evolve a final system from an initial outline specification?
 - a. evolutionary development
 - b. waterfall model
 - c. V-model
 - d. spiral development
2. Evolutionary development can be applied in all but one of these projects
 - a. In small or medium-size interactive systems
 - b. In parts of large systems (e.g. the user interface)
 - c. In short-lifetime systems
 - d. Large long-lifetime system



5.0 Conclusion

In this unit you have learnt about evolutionary and incremental development. You have also learnt the advantages of incremental development and the problems of evolutionary development.



6.0 Summary

Evolutionary and Incremental development are models where the customer and the team developing the software collaborate throughout the developmental stage of the software product; with the customer providing the specification throughout this stage until the final product is delivered.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component?* *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jermaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A., (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-thewatersluice-t573.html#754>

<https://www.geeksforgeeks.org/software-engineering-incremental-process-model/>

Module 2: Introduction To Software Engineering Methodology

Module Introduc

tion

This module continues discussion on the different traditional software engineering process models. The module also discussed agile development methods in software engineering.

Unit 1: Prototyping

Unit 2: Spiral Development

Unit 3: Agile Development Methods

Unit 4: Extreme Programming

Unit 5:

The Dynamic System Development Method (DSDM)

Unit 1: Prototyping

Contents

1.0 Introduction

2.0 Intended Learning Outcome (ILOs)

3.0 Main Content

3.1 Prototyping Model

3.2 Types of Prototyping

3.3 Benefits of Prototyping

3.4 Problems of Prototyping

4.0 Self-Assessment Exercise

5.0 Conclusion

6.0 Summary

7.0

Further Readings



1.0 Introduction

In this unit, you will understand the prototyping model, the benefits of prototyping and the problems of prototyping.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:
describe the prototyping model
understand the benefits of prototyping
understand the problems of prototyping.



3.0 Main Content

3.1 Prototyping Model

A prototyping moves the developer and customer toward a "quick" implementation. It involves the creating of incomplete versions of the software program being developed. The prototype typically simulates a few aspects of the proposed system, and may be completely different from, the final product. Prototyping begins with requirements gathering. Meetings between developer and customer are conducted to determine overall system objectives and functional and performance requirements. The developer then applies a set of tools to develop a quick design and build a working model (the "prototype") of some element(s) of the system. The customer or user "test drives" the prototype, evaluating its function and recommending changes to better meet customer needs. Iteration occurs as this process is repeated, and an acceptable model is derived. The developer then moves to "productize" the prototype by applying many of the steps described for the classic life cycle.

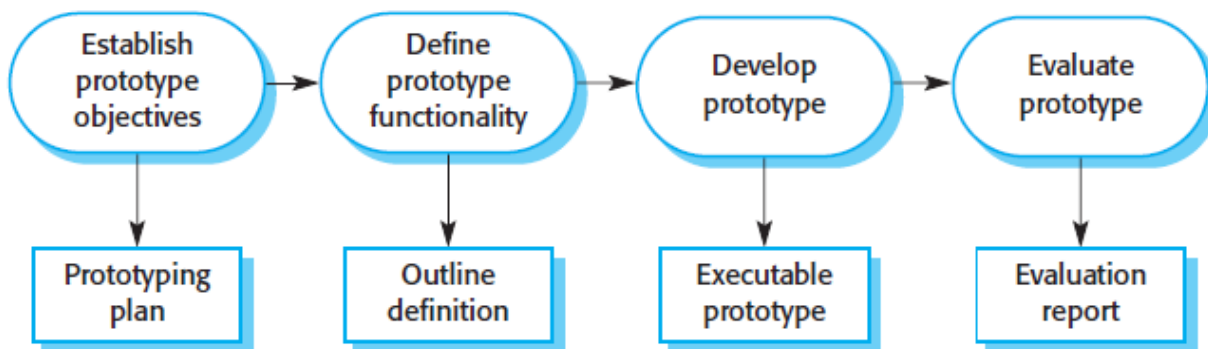


Figure 2.1: The prototyping approach

3.2 Types of Prototyping

1. Throwaway prototyping: this is also known as rapid prototyping, and involves creation of a model that will eventually be discarded rather than becoming part of the final delivered software system.

2. Evolutionary prototyping: involves the creation of a very robust prototype in a structured manner. The developed prototype eventually becomes part of the new software system.

3. Incremental prototyping: here, the final product is built as separate prototypes. The separate prototypes are merged later in an overall design.

3.3 Benefits of Prototyping

The benefits of prototyping include the following;

1. a working model is provided to the customer/user early in the process, enabling early assessment and bolstering confidence,
 2. the developer gains experience and insight by building the model, thereby resulting in a more solid implementation of "the real thing"
 3. the prototype serves to clarify otherwise vague requirements, reducing ambiguity and improving communication between developer and user.
4. Increased chances of having high quality functionality with low risks of failure.

3.4 Problems of Prototyping

1. The user sees what appears to be a fully working system (in actuality, it is a partially working model) and believes that the prototype (a model) can be easily transformed into a production system. This is rarely the case. Yet many users have pressured developers into releasing prototypes for production use that have been unreliable, and worse, virtually unmaintainable.
2. The developer often makes technical compromises to build a "quick and dirty" model. Sometimes these compromises are propagated into the production system, resulting in implementation and maintenance problems.
3. Budget may increase, as the management cost may go beyond the specified limit.
4. Prototyping is applicable only to a limited class of problems. In general, a prototype is valuable when heavy human-machine interaction occurs, when complex output is to be produced or when new or untested algorithms are to be applied. It is far less beneficial for large, batch-oriented processing or embedded process control applications.

Discussion

Describe the prototyping model

Scenario

As the leader of a team of software engineers for a new software project, you observed that the new system would involve heavy human-machine interaction. Would you consider prototyping model for this project? State the reasons for your answer.



4.0 Self-Assessment Exercise(s)

1. one of this is applicable to the prototyping model
 - a. Projects developed with the prototyping model seldom succeeds
 - b. There is increased chances of having high quality functionality with low risks of failure.
 - c. Prototyping model reduces the development cost immensely
 - d. Projects developed with prototyping model have high risk of failure
2. one disadvantage of prototyping is that
 - a. most projects developed with prototyping model never succeeds
 - b. the customers/users run a risk of never understanding how the system operates
 - c. Budget may increase, as the management cost may go beyond the specified limit.
 - d. it is often difficult to achieve any meaning design using prototyping model



5.0 Conclusion

In this unit, you have learned about prototyping, the benefits and the challenges.



6.0 Summary

Prototyping is important because it moves the developers and the customers towards a quick implementation of the desired software project.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component?* *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jermaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A., (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-thewatersluice-t573.html#754>.

Unit 2: Spiral Development

Contents

1.0 Introduction	
2.0 Intended Learning Outcome (ILOs)	
3.0 Main Content	
3.1 Overview of Spiral Development	
3.2 The Spiral Process	
3.3 The Advantages of the Spiral Model	
3.4 Problems of the Spiral Model	
4.0 Self-Assessment Exercise	
5.0 Conclusion	
6.0 Summary	7.0
Further Readings	



1.0 Introduction

This unit discusses spiral development, including the advantages and the problems of spiral development model.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:
describe the spiral development process
discuss the advantages of the spiral model



3.0 Main Content

3.1 Overview of Spiral Development

The idea of minimising risks via the use of prototypes and other means is the underlying concept of the spiral model. A simplistic way of looking at the spiral model is as a series of waterfall models, each preceded by a risk analysis.

Before commencing each phase, an attempt is made to control (or resolve) the risks. If it is impossible to adequately resolve all the significant risks at a given stage, the project is immediately terminated.

In the spiral development, the process is represented as a spiral rather than a sequence of activities with backtracking. Each loop in the spiral represents a phase in the process. There are no fixed phases; while radial dimension as shown in figure 2.2 represents cumulative cost to date. The angular dimension represents progress through the spiral. Each cycle of the spiral corresponds to a development phase.

3.2 The Spiral Process

A phase begins (in the top left quadrant) by determining objectives of that phase, alternatives for achieving those objectives, and constraints imposed on those alternatives. Next, that strategy is analyzed from the viewpoint of risk. Attempts are made to resolve every potential risk, in some cases by building a prototype. If certain risks cannot be resolved, the project may be terminated or scaled down. If all risks are resolved, the next development step is started. This quadrant of the spiral model corresponds to the pure waterfall model. Finally, the results of that phase are evaluated and the next phase is planned.

3.3 The Advantages of the Spiral Model

1. The primary advantage of the spiral model is that it has a wide range of options to accommodate the good features of other life-cycle models. It becomes equivalent to another lifecycle model in appropriate situations.
2. Also the risk-avoidance approach keeps from having additional difficulties.
3. The spiral model focuses its early attention on the option of reusing existing software.
4. It prepares for life-cycle evolution, growth, and changes of the software product. Major sources of this change are included in the product objectives.
5. It incorporates software quality objectives into software product development.
6. Emphasis is placed on identifying all objectives and constraints during each round.
7. The risk analysis and validation steps eliminate errors early on.
8. Maintenance is included as another cycle of the spiral; there is essentially no distinction between maintenance and development. This helps to avoid underestimation of resources needed for maintenance.

3.4 Problems of the Spiral Model

1. The risk-driven model is dependent on the developers' ability to identify project risk.
2. The entire product depends on the risk assessment skills of the developer. If those skills are weak then the product could be a disaster.

3. A design produced by an expert may be implemented by non-experts. In a case such as this, the expert does not need a great deal of detailed documentation, but must provide enough additional documentation to keep the non-experts from going astray.
4. The process steps need to be further elaborated to make sure that the software developers are consistent in their production.
5. It is still fairly new compared to other models, so it has not been used significantly and therefore the problems associated with it haven't been widely tested and solved.

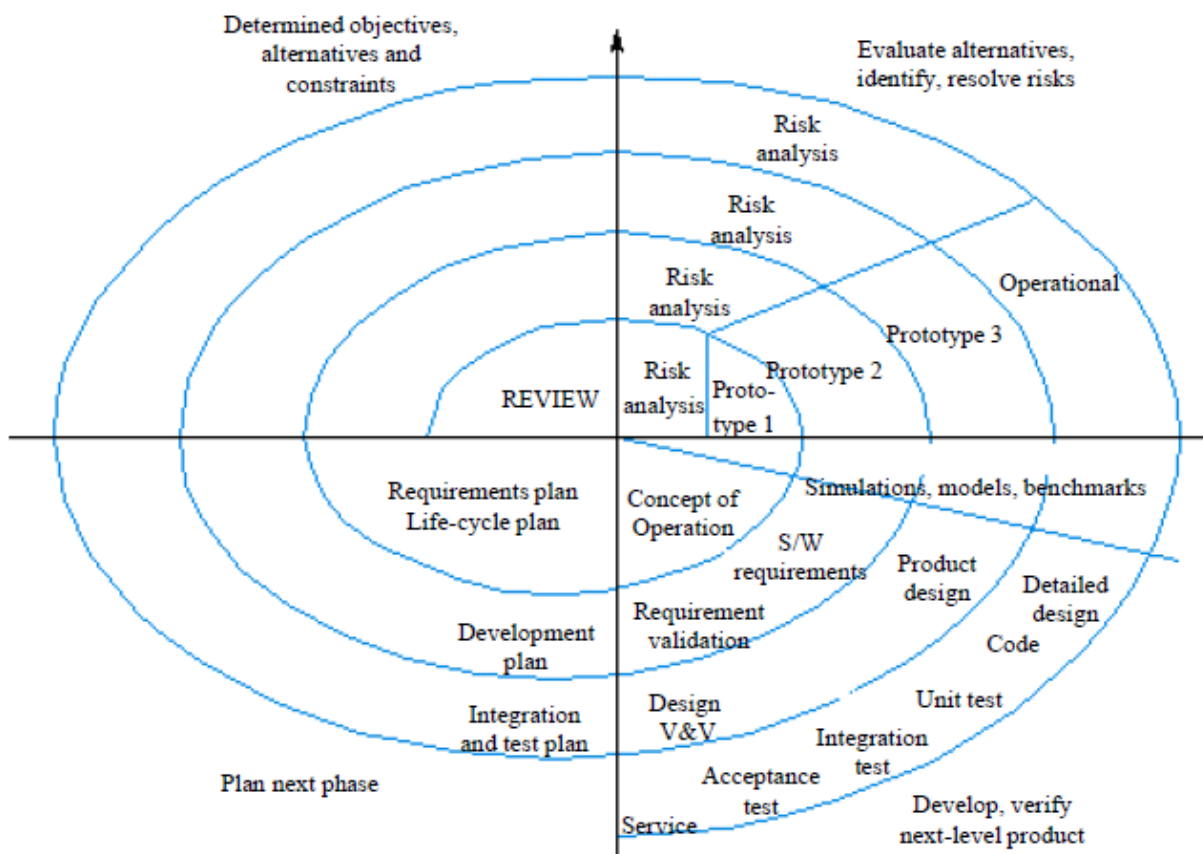


Figure 2.2: Spiral development

Discussion

Describe the spiral process

Scenario

If your new software project may involve reusing an existing software, would you consider the spiral model for the development of this software project?



4.0 Self-Assessment Exercise(s)

1. in spiral development, one the following is applicable.
 - a. The spiral model does not put maintenance into consideration
 - b. Maintenance is included as a cycle in the spiral model
 - c. Maintenance is not part of the spiral model
 - d. Spiral development is a very old and ineffective model
2. one good feature of the spiral development model is that _____ is constantly done
 - a. risk analysis
 - b. cost analysis
 - c. reverse engineering
 - d. model analysis



5.0 Conclusion

In this unit, you have learned about spiral development, the advantages and the problems.



6.0 Summary

Spiral development is important, due to its ability to accommodate other models. There are no fixed phase in spiral development. Also, maintenance is part of the spiral development process.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component? Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). "A Security Framework for Reflective Java Applications," *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). "Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability,". *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A., (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-thewatersluice-t573.html#754>.

Unit 3: Agile Development Methods

Contents

1.0 Introduction	
2.0 Intended Learning Outcome (ILOs)	
3.0 Main Content	
3.1 Introduction to Agile Software development	
3.2 Characteristics of Agile Software development method	
3.2.1 The Agile Manifesto	
3.2.2 Agile software development principles	
3.3 Applications of Agile Methods	
3.4 Problems of Agile Method	
3.5 Some Agile Software Development Models	
4.0 Self-Assessment Exercise	
5.0 Conclusion	
6.0 Summary	7.0
Further Readings	



1.0 Introduction

This unit introduces you to Agile development methods.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- define Agile software development method
- understand the differences between Agile and plan-based software development methods
- list the characteristics of Agile software development methods
- understand the applications of agile methods
- list some agile development models.



3.0 Main Content

3.1 Introduction to Agile Software Development

Today's businesses operate in a rapidly changing environment, making them to contend with new market opportunities, changing economic situations and fast evolving customer and product requirements. This has made software delivery and products to try to keep up with changes in economic situations. Software has to be developed and delivered rapidly to satisfy growing user demands and requirements.

Similarly, the speed with which software change has made it impossible for it to have stable requirements. Consequently, plan-based software development process like waterfall and V-model that completely specify the system requirements before such system is designed, built, and tested make it impossible for developers to satisfy the growing and changing user requirements for most business software projects.

faster software development took off in the late 1990s with the development of the idea of "agile methods", which is also known as rapid software development. These agile methods are designed to produce useful software quickly.

In figure 2.3, we can see that while Agile development has the system alternate from requirement engineering to design and implementation, plan-based models have the system flow from one stage to another in the model.

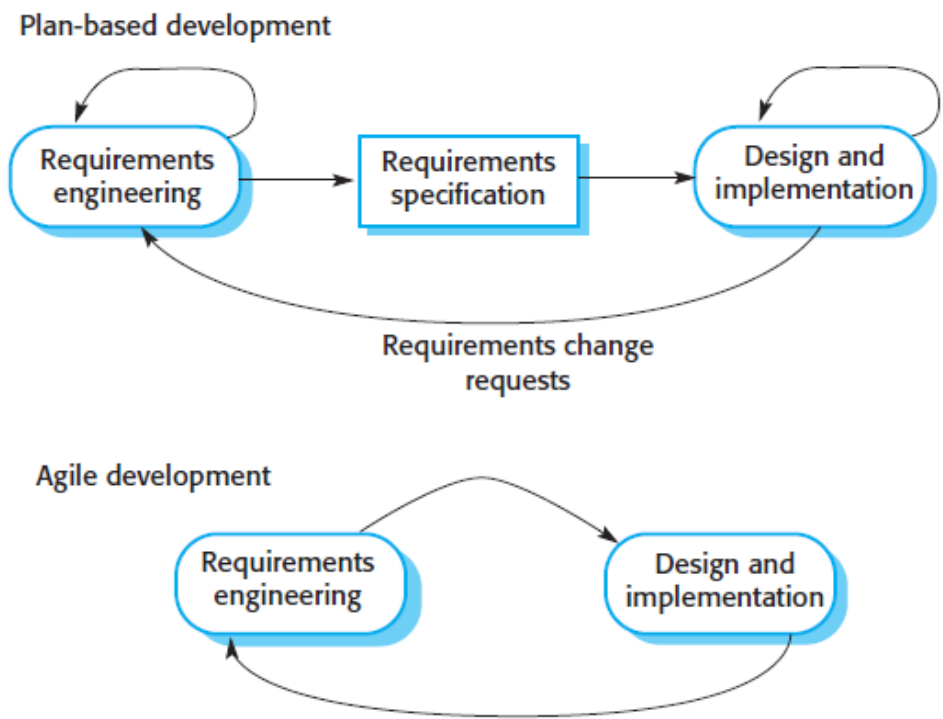


Figure 2.3: Plan-based development compared against agile development.

3.2 Characteristics of Agile Software development method

- a. The processes of specification, design and implementation are interleaved; No detailed system specification, design documentation is minimized or generated automatically by the programming environment used for system implementation. The user requirements document only contains an outline of the most important system characteristics.
- b. The system is developed in a series of increments, with the end-users and other system stakeholders involved in specifying and evaluating each increment.
- c. Supports extensive tool which is used in the development process.

3.2.1 The Agile Manifesto

The agile manifesto was developed and signed by seventeen software engineers. It is a four-element philosophy which states the values of agile software development. The signatories based this manifesto on their combined experience of developing software and helping others do that. It includes the following;

- *Individuals and interactions* over processes and tools
- *Working software* over comprehensive documentation
- *Customer collaboration* over contract negotiation
- *Responding to change* over following a plan

This is to say that, the items on the left are more valued than the items on the right.

3.2.2 Agile Software Development Principles

The agile manifesto is built on the following twelve principles of agile software development;

1. Customer satisfaction by early and continuous delivery of valuable software.
2. Welcome changing requirements, even in late development.
3. Deliver working software frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the primary measure of progress
8. Sustainable development, able to maintain a constant pace

9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Best architectures, requirements, and designs emerge from self-organizing teams
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Embrace change	Expect the system requirements to change, and so design the system to accommodate these changes.
Incremental delivery	The software is developed in increments, with the customer specifying the requirements to be included in each increment.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.
People, not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.

Figure 2.4: summary of the agile principles

3.3 Applications of Agile Methods

Agile methods have largely been successfully in the following systems;

- a. Small or medium-sized product development where the product is developed for sale. Most software products and apps are now developed using an agile approach.
- b. Custom product development in an organization, where there is a commitment from the customer to get involved in the development process; with few external stakeholders and regulations affecting the software.

3.4 Problems of Agile Method

Though Agile method has been successful in some software products, especially apps, it still won't be suitable for complex systems and embedded systems. Also, for large, long-lifetime systems that are developed by a software company for an external client, an agile approach would present a number of problems, such as:

- a. The informality of agile development is incompatible with the legal approach to contract definition that is commonly used in large companies

- b. Agile methods are most appropriate for new software development rather than for software maintenance. However, the majority of software costs in large companies come from maintaining their existing software systems
- c. Agile methods are designed for small co-located teams, yet much software development now involves worldwide distributed teams.

3.5 Some Agile Software Development Models

The following are some agile development methodologies;

Dynamic systems development method (DSDM)

Extreme programming (XP)

Scrum

Rapid application development (RAD)

Discussion

List the manifesto of agile methods

Scenario

As a member of a software development team, your boss has asked that you lead the team that would engage your client's team in an ongoing software project. On your first meeting, you discover that your client's team is unwilling to release a detailed document containing the software requirement and specification; and would rather prefer producing the specifications throughout the period of the software development. List two types of software development models that could be used for this project. Give reasons for your choice.



4.0 Self-Assessment Exercise(s)

1. One of these is not a principle of Agile development
 - a. Customer satisfaction by early and continuous delivery of valuable software
 - b. Deliver working software frequently (weeks rather than months)
 - c. time is spent on specification and documentation
 - d. Continuous attention to technical excellence and good design
2. one of these is not an agile development model
 - a. Rapid application development
 - b. Extreme programming
 - c. V-model

d. Dynamic systems development method (DSDM)



5.0 Conclusion

In this unit, you have learnt about Agile development methods. You have also understood the difference between traditional plan-based methods and Agile methods.



6.0 Summary

In agile methods, the processes of specification, design and implementation are interleaved. There is also no detailed system specification. Similarly, design documentation is minimized or generated automatically by the programming environment used for system implementation.



7.0 Further Readings

Arisholm, E., Gallis, H., Dyba, T. and Sjoberg, D. I. K. (2007). Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. *IEEE Trans. on Software Eng.* 33 (2): 65–86. doi:10.1109/TSE.2007.17.

Beck, K. (1998). Chrysler Goes to ‘Extremes. *Distributed Computing* (10): 24–28.

Fowler, M., Beck, K., Brant, J., Opdyke, W. and Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code*. Boston: Addison-Wesley.

Jeffries, R. and Melnik, G. (2007). TDD: The Art of Fearless Programming. *IEEE Software* 24: 24–30. doi:10.1109/MS.2007.75.

Larman, C., and Basili, V. R. (2003). Iterative and Incremental Development: A Brief History. *IEEE Computer* 36 (6): 47–56. doi:10.1109/MC.2003.1204375.

Leffingwell, D. (2011). *Agile Software Requirements: Lean Requirements Practices for Teams, Programs and the Enterprise*. Boston: Addison-Wesley.

Williams, L., Kessler, R. R., Cunningham, W., and Jeffries, R. (2000). Strengthening the Case for Pair Programming. *IEEE Software* 17 (4): 19–25. doi:10.1109/52.854064.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Online Resources

https://en.wikipedia.org/wiki/Agile_software_development

<https://www.scrumalliance.org/resources/agile-manifesto>

<https://agilemanifesto.org/>

Unit 4: Extreme Programming

Contents

1.0 Introduction	
2.0 Intended Learning Outcome (ILOs)	
3.0 Main Content	
3.1 Overview of Extreme Programming	
3.2 Extreme Programming Techniques	
3.2.1 User Stories	
3.2.2 Refactoring	
3.2.3 Test-first Development	
3.2.4 Pair Programming	
3.3 Criticism of Extreme Programming	
4.0 Self-Assessment Exercise	
5.0 Conclusion	
6.0 Summary	7.0
Further Readings	



1.0 Introduction

This unit discusses extreme programming as an agile method. It also looks at the techniques of extreme programming.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

Discuss extreme programming

List and explain some of the extreme programming techniques



3.0 Main Content

3.1 Overview of Extreme Programming

Extreme Programming (XP) created by Kent Beck is an agile software development methodology which is intended to address the issue of responsiveness to changing customer requirements and also

improve software quality. It is built on the idea of frequent releases in short development cycles, which is intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted.

Extreme Programming (XP), meant pushing recognized good practices such as iterative development to the extreme. With XP, requirements are expressed as scenarios called user stories, and implemented as a series of tasks.

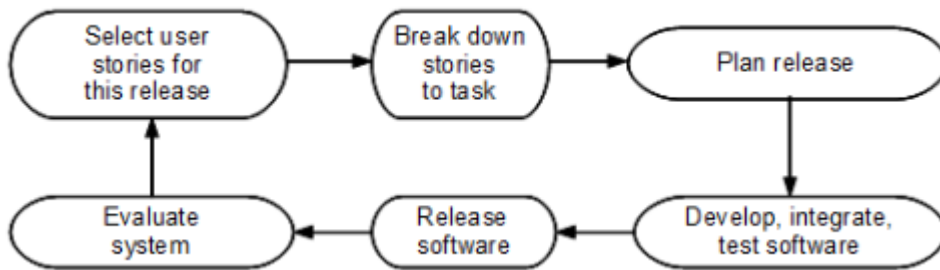


Figure 2.5: Extreme programming release cycle

3.2 Extreme Programming Techniques

Some of the techniques used in extreme programming include;

3.2.1 User Stories

Agile methods do not have a requirement engineering activity to take care of user and system requirements. Instead, it integrates requirement elicitation with the development. This is done using "User stories", which is a scenario of use that might be experienced by a system user. With the user stories, a story card is developed with the team working to implement these new scenarios in future releases of the software.

3.2.2 Refactoring

In traditional software engineering, teams typically design with anticipation of a change in the software design. In XP, this is often regarded as a waste of time. To take care of these changes, XP uses the idea of refactoring, where the team looks for improvements to the software and implements such immediately.

3.2.3 Test-first Development

As a result of lack of formal specification in XP, testing a developed software product becomes a challenge. XP implements a new approach to program testing, where testing is automated and is

central to the development process; and development cannot proceed until all tests have been successfully executed. The key features of testing in XP are:

- a. test-first development
- b. incremental test development from scenarios
- c. user involvement in the test development and validation
- d. the use of automated testing frameworks.

3.2.4 Pair Programming

XP introduced the idea of pair programming, where programmers work in pairs to develop the software. Each pair sits at the same computer to develop the software. Also pairs are created dynamically, making them not to always work together. Some well established software companies that have adopted Agile method do not use pair programming, as they doubt its efficiency in terms of work output. However, many proponents of Agile method admit it can produce as much codes as the pair would have produced if they were working individually.

3.3 Criticism of Extreme Programming

Extreme programming has faced so much criticism, especially as it concerns pair programming. However, many of the criticisms faced by extreme programming are believed by Agile practitioners to be lack of understanding of the agile development method. Some of the criticisms include;

- a. Insufficient software design
- b. difficult in contractual negotiations
- c. can be inefficient, especially for large software systems.
- d. lacks structure and necessary documentation.

Discussion

One of the techniques of Agile method is pair programming, in your understanding, briefly discuss with your team why you think large software companies would want to avoid pair programming.

Scenario

Briefly list and explain three projects that could be developed with agile methods.



4.0 Self-Assessment Exercise(s)

1. One of these is not an extreme programming technique

- a. Pair programming

- b. User stories
- c. Refactoring
- d. Specification

2.0 in extreme programming, requirements are captured as _____

- a. User stories
- b. Pair requirements
- c. User elicitation
- d. Specification



5.0 Conclusion

In this unit, you have learnt about the extreme programming development method.



6.0 Summary

Extreme programming is an agile methods. It may be suitable for the production of apps and software produced by a company for sale to a client. It is highly unsuitable for complex and embedded systems.



7.0 Further Readings

Arisholm, E., Gallis, H., Dyba, T. and Sjoberg, D. I. K. (2007). Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. *IEEE Trans. on Software Eng.* 33 (2): 65–86. doi:10.1109/TSE.2007.17.

Beck K (2000). *Extreme Programming Explained*. Addison-Wesley

Beck, K. (1998). Chrysler Goes to ‘Extremes. *Distributed Computing* (10): 24–28.

Fowler, M., Beck, K., Brant, J., Opdyke, W. and Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code*. Boston: Addison-Wesley.

Jeffries, R. and Melnik, G. (2007). TDD: The Art of Fearless Programming. *IEEE Software* 24: 24–30. doi:10.1109/MS.2007.75.

Larman, C., and Basili, V. R. (2003). Iterative and Incremental Development: A Brief History. *IEEE Computer* 36 (6): 47–56. doi:10.1109/MC.2003.1204375.

Leffingwell, D. (2011). *Agile Software Requirements: Lean Requirements Practices for Teams, Programs and the Enterprise*. Boston: Addison-Wesley.

Williams, L., Kessler, R. R., Cunningham, W., and Jeffries, R. (2000). Strengthening the Case for Pair Programming. *IEEE Software* 17 (4): 19–25. doi:10.1109/52.854064.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Online Resources

https://en.wikipedia.org/wiki/Agile_software_development

<https://www.scrumalliance.org/resources/agile-manifesto>

http://moodle.autolab.uni-pannon.hu/Mecha_tananyag/szoftverfejlesztési_folyamatok_angol/ch04.html

Unit 5: The Dynamic System Development Method (DSDM)

Contents

1.0 Introduction	
2.0 Intended Learning Outcome (ILOs)	
3.0 Main Content	
3.1 Overview of Dynamic Systems Development Method (DSDM)	
3.2 The Phases of the DSDM Lifecycle	
3.3 The Principles of the DSDM	
3.4 The Techniques of DSDM	
3.5 Roles in DSDM	
4.0 Self-Assessment Exercise	
5.0 Conclusion	
6.0 Summary	7.0
Further Readings	



1.0 Introduction

This unit discusses the Dynamic System Development Method (DSDM) as an agile method.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- Discuss the Dynamic System Development Method (DSDM)
- List the phases of the DSDM lifecycle
- Explain the DSDM principles
- Explain the techniques of the DSDM
- Identify the individual roles in the DSDM



3.0 Main Content

3.1 Overview of Dynamic Systems Development Method (DSDM)

The Dynamic Systems Development Method (DSDM) is an agile development method developed in a bid to bring discipline in rapid application development. It was first created and released in 1994 and focuses on the full project lifecycle. Later versions of the DSDM Agile Project Framework has been revised and used as a generic approach to project management and solution; and has been used in non-Information Technology fields as well.

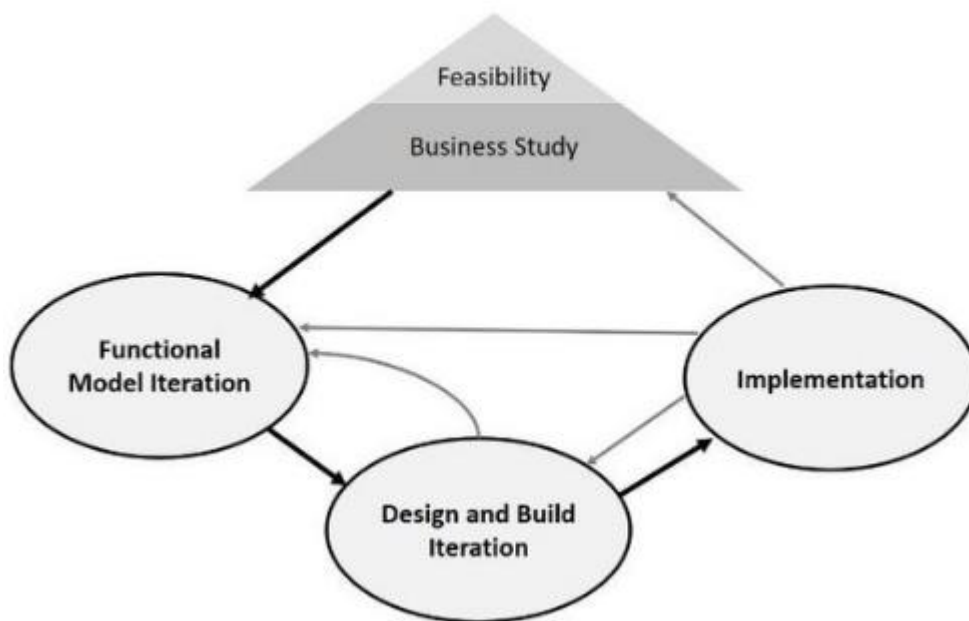


Figure 2.6: The Dynamic Systems Development Method lifecycle

3.2 The Phases of the DSDM Lifecycle

The phases of the DSDM lifecycle include the following;

Feasibility Study
Business Study
Functional Model Iteration
Design and Build Iteration
Implementation

3.3 The Principles of the DSDM

The DSDM has eight principles. These principles guide any DSDM team in the right attitude and mindset to deliver in every software project. These principles include;

1. Focus on the business need

2. Deliver on time
3. Collaborate
4. Never compromise quality
5. Build incrementally from firm foundations
6. Develop iteratively
7. Communicate continuously and clearly
8. Demonstrate control

3.4 The Techniques of DSDM

The techniques adopted in DSDM include the following;

1. **Prototyping:** This involves the creation of prototypes of the system at an early stage of the project; making it possible to discover the shortcomings in the system early-on and allow future users to ‘test-drive’ the system.
2. **Timeboxing:** This is an approach for completing the project incrementally by breaking it down into project portions, each having a fixed budget and delivery date.
3. **Testing:** This helps to ensure a good quality solution. DSDM advocates that, testing should be done throughout each iteration.
4. **Workshop:** This brings the project stakeholders together to discuss requirements, functionalities and mutual understanding.
5. **Modeling:** Modeling helps one to visualise a business domain in order to have a better understanding of the system. It involves a diagrammatic representation of specific aspects of the system or business area that is being developed.
6. **Configuration management:** Since the system development involves multiple deliverables under development at the same time which are delivered incrementally at the end of each time-box, the deliverables must be well managed throughout this process, until completion.

3.5 Roles in DSDM

with the use of DSDM, some roles were introduced to facilitate the software project development. These roles are important for the software project to succeed. Each role also has responsibilities assigned to the team members in them. Some of the roles include the following;

1. **Executive Sponsor:** The executive sponsor is a very important person who commits funds and resources to the project. This role has an ultimate power to make decisions.

2. **Visionary:** This person initialises the project by ensuring that essential requirements are found early on. They have the most accurate perception of the business objectives of the system and the project and ensures the project is on the right track.
3. **Project Manager:** This is the main manager of the whole software development process.
4. **Facilitator:** Manages the workshops' progress, and motivates team members.
5. **Technical Co-ordinator:** This individual designs the software system architecture. They also control the technical quality of the project.
6. **Ambassador User:** This person brings the knowledge of the user community into the project, and ensures that the developers receive adequate user feedback during the software development process.
7. **Advisor User:** This is a user that represents an important viewpoint; and usually brings daily knowledge of the project.
8. **Team Leader:** The leader of the team, who ensures that the team works effectively.
9. **Solution Developer** Interpret the system requirements and model it; including developing the deliverable codes.
10. **Solution Tester:** This individual tests the technical correctness of the system by performing some specified testing.
11. **Scribe:** Records the requirements, decisions and agreements made in every workshop.

Discussion

List and discuss three principles of DSDM

Scenario

In a Software project in which DSDM is being used. The team ran out of project cash, as they exhausted their original budget. Whose role is it to approve a new budget and make cash available for the project to continue?



4.0 Self-Assessment Exercise(s)

1. One of these is not a Dynamic System Development Method (DSDM) technique.
 - a. Prototyping
 - b. Timeboxing
 - c. Requirements engineering
 - d. Testing

2.0 In Dynamic System Development Method (DSDM) , which roles has the ultimate power to commit funds to the projects_____?

- a. Executive sponsor
- b. Project manager
- c. Visionary
- d. Solution developer



5.0 Conclusion

In this unit, you have learnt about Dynamic System Development Method (DSDM).



6.0 Summary

Dynamic System Development Method (DSDM) is an agile methods. It seeks to bring discipline into rapid application development as it focuses on the full project lifecycle.



7.0 Further Readings

Arisholm, E., Gallis, H., Dyba, T. and Sjoberg, D. I. K. (2007). Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. *IEEE Trans. on Software Eng.* 33 (2): 65–86.

doi:10.1109/TSE.2007.17.

Delima, R., Santoso, H. B., and Wibowo, A. (2018). Development of Purchasing Module for Agriculture E-Commerce using Dynamic System Development Model. *International Journal of Advanced Computer Science and Applications.* 9(10)

Fahad, M., Qadri, S., Ullah, S., Husnain, M., Qaiser, R., Qureshi, S. A., Ahmed, W., and Muhammad, S. S. (2017). A Comparative Analysis of DXPRUM and DSDM. *International Journal of Computer Science and Network Security.* 17(5)

Jeffries, R. and Melnik, G. (2007). TDD: The Art of Fearless Programming. *IEEE Software* 24: 24–30. doi:10.1109/MS.2007.75.

Larman, C., and Basili, V. R. (2003). Iterative and Incremental Development: A Brief History. *IEEE Computer* 36 (6): 47–56. doi:10.1109/MC.2003.1204375.

Leffingwell, D. (2011). *Agile Software Requirements: Lean Requirements Practices for Teams, Programs and the Enterprise*. Boston: Addison-Wesley.

Mekni, M., Buddhavarapu, G., Chinthapatla, S., and Gangula, M. (2018). Software Architectural Design in Agile Environments. *Journal of Computer and Communications*. 6, pp 171-189

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Online Resources

https://en.wikipedia.org/wiki/Agile_software_development

<https://www.agilebusiness.org/page/whatisdsdm>

http://moodle.autolab.uni-pannon.hu/Mecha_tananyag/szoftverfejlesztési_folyamatok_angol/ch04.html

<https://www.geeksforgeeks.org/dynamic-systems-development-method-dsdm/>

Module 3: Requirements Engineering Processes

Module Introduction

This module introduced the basic principles of requirements engineering and requirements engineering processes.

Unit 1: Requirements	Unit
2: Functional and Non-Functional Requirements	Unit
3: Domain Requirements	Unit 4:
Requirements Design and Writing	Unit 5:
Requirement Elicitation and Analysis	

Unit 1: Requirements

Contents

1.0 Introduction	
2.0 Intended Learning Outcome (ILOs)	
3.0 Main Content	
3.1 Requirements and Requirements Engineering	
3.2 Types of Requirements	
3.3 Requirements Completeness and Consistency	
3.4 Requirement Engineering	
4.0 Self-Assessment Exercise	
5.0 Conclusion	
6.0 Summary	7.0
Further Readings	



1.0 Introduction

In this unit we would consider requirements engineering as well as requirements. In addition, we would consider requirements imprecision, completeness and consistency.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

give a basic definition of a requirements engineering

describe a requirement
explain requirements imprecision
list the types of requirements.



3.0 Main Content

3.1 Requirements and Requirements Engineering

Requirements are the descriptions of the services that will be provided by a system and the constraints on its operation. They are basically a representation of the needs of customers in a particular software product. A requirement may range from a high-level abstract statement of a service or a system constraint to a detailed mathematical functional specification. This is inevitable as requirements may serve a dual function. They may be the basis for a bid for a contract, and therefore must be open to Interpretation. It may also be the basis for the contract itself, and therefore should be defined in detail.

In the context of software engineering, requirements would have different types or levels.

3.2 Types of Requirements

1. User Requirements

This simply means high-level abstract requirements. Usually, this is provided in a natural language with diagrams of the services the system is expected to provide to system users and the constraints under which it must operate. Also, the user requirements may range from general statements of the system features to detailed, precise descriptions of the system functionality.

2. System Requirements

This is a more structured and detailed descriptions of the software system's functions, services, and operational constraints. The system requirements document will define exactly what is to be implemented. It may be part of the contract between the client and the software developer.

3. Functional Requirements

Contains statements of services the system should provide how the system should react to particular inputs and how the system should behave in particular situations. Functional requirements may also explicitly state what the system should not do.

4. Non-Functional Requirements

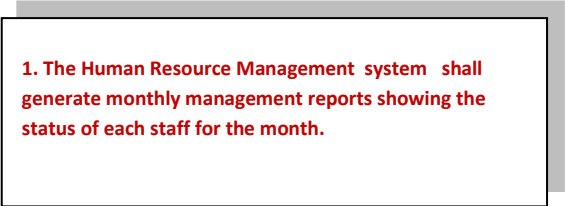
These are the constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc. Non-functional requirements often apply to the system as a whole rather than individual system features or services.

5. Domain Requirements

These are requirements that come from the application domain of the

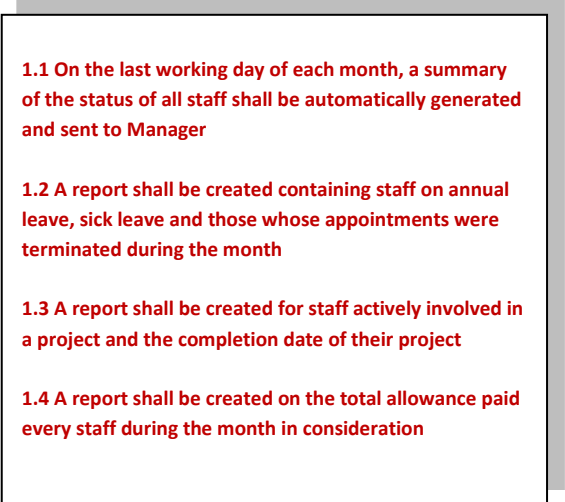
system and that reflect characteristics of that domain.

Figures 3.1 and 3.2 give an example of user requirements and system requirements for a Human Resource Management system.



1. The Human Resource Management system shall generate monthly management reports showing the status of each staff for the month.

Figure 3.1: an example of user requirement for a Human Resource Management system



1.1 On the last working day of each month, a summary of the status of all staff shall be automatically generated and sent to Manager

1.2 A report shall be created containing staff on annual leave, sick leave and those whose appointments were terminated during the month

1.3 A report shall be created for staff actively involved in a project and the completion date of their project

1.4 A report shall be created on the total allowance paid every staff during the month in consideration

Figure 3.2: an example of system requirement for a Human Resource Management system

3.3 Requirements Completeness and Consistency

In principle, requirements should be both complete and consistent.

Complete: Requirements should include descriptions of all facilities required.

Consistent: Requirements should not contain conflicts or contradictions in the descriptions of the system facilities.

Generally, in practice, it is impossible to produce a wholly complete and consistent requirements document.

3.4 Requirement Engineering

Requirements engineering refers to the process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed. In traditional software development methodology, Requirements engineering is usually presented as the first stage of the software engineering process.

Discussion

In a new software engineering project, your client has informed you that their overall user requirement is a system that is able record the volume of sales each day. With provided user requirement, develop four likely system requirements, the system could have.

Scenario

Between user requirements and system requirements, which is more detailed and which of them uses more natural language?



4.0 Self-Assessment Exercise(s)

1. What is a descriptions of the services that will be provided by a system and the constraints on its operation?
 - a. Requirements engineering
 - b. Requirements
 - c. Reverse engineering
 - d. Requirements specification
2. One of these is not a type of requirements
 - a. Functional
 - b. Non-functional
 - c. Operational
 - d. System



5.0 Conclusion

In this unit you have learned about requirements engineering. You have also learned about requirements imprecision. Finally, you have been able to learn about types of requirements.



6.0 Summary

Before a system can be developed, a form of requirements have to be provided by the clients or software users. Requirements are very important in the development of a system, as they will guide the software developers in their task.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component? Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jermaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A., (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Unit 2: Functional and Non-functional Requirements

Contents

- 1.0 Introduction
- 2.0 Intended Learning Outcome (ILOs)
- 3.0 Main Content
 - 3.1 Functional Requirements
 - 3.2 Levels of Functional Requirements
 - 3.2.1 Functional User Requirements
 - 3.2.2 Functional System Requirements
 - 3.3 Non-Functional Requirements
 - 3.4 Non-Functional Requirements Classifications
 - 3.5 Problems With Non-Functional Requirements
- 4.0 Self-Assessment Exercise
- 5.0 Conclusion
- 6.0 Summary
- Further Readings

7.0



1.0 Introduction

This unit discusses functional and non-functional requirements.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- describe functional requirements
- give Examples of functional requirements
- describe non-functional requirements
- describe the classes and types of non-functional requirements.



3.0 Main Content

3.1 Functional Requirements

Functional requirements describe the functionality; which is system services, how the system should react to particular inputs, and how it should behave in particular situations. They depend on the type of software, expected users and the type of system where the software would be used.

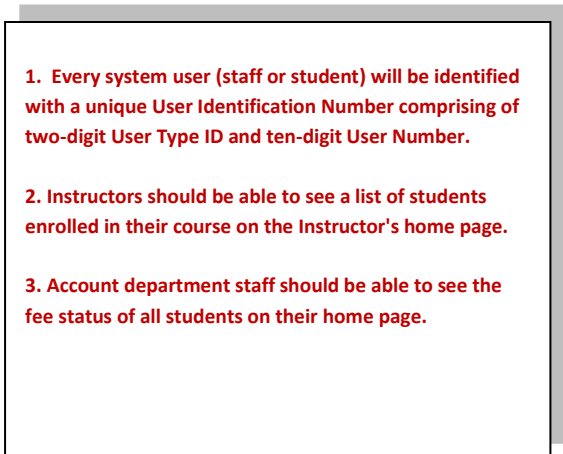


Figure 3.3: an example of functional requirements for a Learning Management System

3.2 Levels of Functional Requirements

Functional requirements are of two levels, namely; functional use requirements and functional system requirements.

3.2.1 Functional User Requirements

This is usually a high-level statement of what the system should do, normally written in natural language. It should be very clear for managers and system users to understand them.

3.2.2 Functional System Requirements

Functional system requirements describe the system services in detail by expanding the user requirements. They should describe the system functions, their inputs and outputs, and exceptions in detail.

Functional system requirements vary from general requirements that cover what the system should do to very specific requirements that reflects local ways of working or an organization's existing systems. They are written for system developers and may not always make sense to system users.

3.3 Non-Functional Requirements

Non-functional requirements are requirements that are not directly concerned with the specific services delivered by the system to the system users. They usually specify or constrain characteristics of the system as a whole. These characteristics may include reliability, response time and storage requirements. Also, constraints on system implementation will include I/O device capability, system representations, etc.

Process requirements may also be specified mandating a particular CASE system, programming language or development method. Non-functional requirements may be more critical than functional requirements; and if they are not met, the system will be unusable.

3.4 Non-Functional Requirements Classifications

Figure 2.6 shows the classifications and types of non-functional requirements for a Learning Management system. The non-functional requirements are classified as product requirements, organisational requirements and external requirements. Each of the classes is further broken down into types.

1. Product Requirements

Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.

2. Organisational Requirements

Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.

3. External Requirements

Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

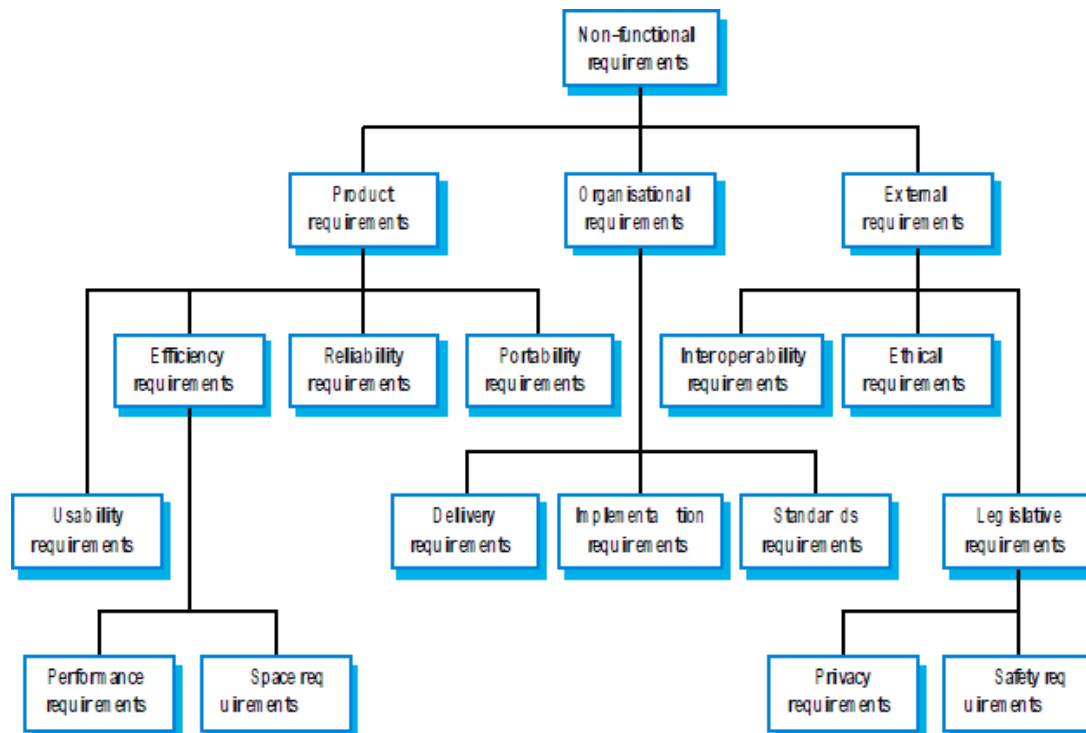


Figure 3.4: Classifications and types of non-functional requirements for a Learning Management system

3.5 Problems With Non-Functional Requirements

Usually, stakeholders may propose requirements as general goals; such as ability of the system to recover from failure or ease of use. This causes problems for the system developers as it may result in disputes of whether the needs of the users were met after system delivery. This is usually as a result of the difficulty customers often face in stating measurable goals, as there may not be standard or simple metric that can be used to measure such goals. At other times, even when there exists a standard metric for such measurement, the customer may still not be able to relate their goals to these metrics. It is usually important that the software developers decide and agree with the customers before time on a standard metric that could be used in measuring such non-functional requirements.

Table 3.1 shows typical scenarios where managers and users give the software developers immeasurable non-functional requirements and possible metrics that could be used to measure such.

Table 3.1: Possible metrics for measuring user non-functional requirements

Property	Possible metrics for measurement
System should be fast	No of completed transactions/time/system user
System should be easy to learn and use	Time taken to train a staff and number of helps given to them during practice
System should have less error	No of errors made by a user/time

Discussion

List out three typical immeasurable non-functional requirements you could face in a software development project.

How do you intend to measure them?

Scenario

You have recently been selected by your manager to lead a team of software developers to document the requirements of a new online market software projects.

Briefly list and discuss two possible functional system requirement that may emanate from the system.



4.0 Self-Assessment Exercise(s)

1. What type of functional requirement is expressed in high-level statement using natural language?
 - a. functional user requirements
 - b. functional system requirements
 - c. non-functional requirements
 - d. domain requirements
2. Which type of requirements are not directly concerned with the specific services delivered by the system to the system users.
 - a. Non-functional
 - b. Functional
 - c. System
 - d. Operational



5.0 Conclusion

In this unit you have learned about functional requirements, non-functional requirements, and the classes and types of functional requirement types.



6.0 Summary

Functional requirements set out services the system should provide, while Non-functional requirements constrain the system being developed or the development process.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component?* *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A., (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Unit 3: Domain Requirements

Contents

1.0 Introduction	
2.0 Intended Learning Outcome (ILOs)	
3.0 Main Content	
3.1 Domain Requirements	
3.2 Problems of Domain Requirements	
4.0 Self-Assessment Exercise	
5.0 Conclusion	
6.0 Summary	7.0
Further Readings	



1.0 Introduction

This unit describes domain requirements. It also discusses the problems of domain requirements.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- explain domain requirements
- give example of a particular domain's requirements
- state domain requirements problems.



3.0 Main Content

3.1 Domain Requirements

Domain requirements come from the application domain of the system, and describe system characteristics and features that reflect the domain. Domain requirements can be new functional requirements, constraints on existing requirements, define specific computations or set out rules on how particular operations will be performed. If domain requirements are not satisfied, the system may be unworkable. Figure 3.5 gives an example of a domain requirement for the Library System domain.

- 1 There shall be a standard user interface to all databases which shall be based on the Z39.50 standard.

2. Because of copyright restrictions, some documents must be deleted immediately on arrival. Depending on the user's requirements, these documents will either be printed locally on the system server for manually forwarding to the user or routed to a network printer.

Figure 3.5: Domain requirements for the library system domain

3.2 Problems of Domain Requirements

Because domain requirements are functions of the application domain of a system which may be peculiar to such domain; software engineers may not always understand these peculiarities. This may result in software engineers not knowing whether or not a domain requirement has been missed out or conflicts with other requirements of the system.

1. Understandability

- a. Requirements are expressed in the language of the application domain
- b. It is often not understood by software engineers developing the system.

2. Implicitness

- a. Domain specialists understand the area so well and do not always make the domain requirements explicit.

Discussion

List out possible domain requirements for a Learning Management system.

Scenario

Why is it sometimes difficult for software engineers to capture all domain requirements?



4.0 Self-Assessment Exercise(s)

1. What type of requirement comes from the application domain of the system, and describes system characteristics and features that reflect the domain.
 - a. domain requirement
 - b. system requirement
 - c. functional requirement

d. non-functional requirement

2. one of these is not true about domain requirements

- a. without fully specifying the domain requirements, a system becomes unusable
- b. domain requirements are not easily understood by software engineers
- c. domain requirements are sometimes unique to a domain
- d. domain requirements are not as important as functional requirement



5.0 Conclusion

In this unit you have learned domain requirements. You have also been able to learn about domain requirement problems.



6.0 Summary

Domain requirements are important; and if not captured explicitly the system may be unusable.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component?* *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A, (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Unit 4: Requirements Design and Writing

1.0 Introduction

2.0 Intended Learning Outcome (ILOs)

3.0 Main Content

3.1 Guidelines for Writing Requirements

3.2 System Requirements

3.3 Requirements and Design

3.4 The Requirements Document

3.5 Users of a Requirement Document

3.6 User Requirements

3.7 IEEE Requirements Standards

4.0 Self-Assessment Exercise

5.0 Conclusion

6.0 Summary

7.0

Further Readings



1.0 Introduction

This unit describes requirements. It states the guidelines for writing requirements. You will learn about requirements document and the IEEE requirements standards.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

list guidelines for writing requirements

explain system requirements

describe the requirement document

list user requirements

understand the IEEE requirement standard.



3.0 Main Content

3.1 Guidelines for Writing Requirements

i. Invent a standard format or use an existing standard template for all requirements.

ii. Be consistent in your language. Use "shall" for mandatory requirements and "should" for desirable requirements.

iii. Use text highlighting to identify key parts of the requirement.

iv. Avoid the use of computer jargon.

- v. Avoid ambiguity.
- vi. Use simple and direct sentences.
- Vii. Avoid mixing requirements and design.

3.2 System Requirements

System requirements are more detailed specifications of system functions, services and constraints than user requirements. They are intended to be a basis for designing the system. They may be incorporated into the system contract.

3.3 Requirements and Design

In principle, requirements should state what the system should do and the design should describe how it does this. In practice, requirements and design are inseparable.

A system architecture may be designed to structure the requirements. The system may inter-operate with other systems that generate design requirements. The use of a specific design may be a domain requirement.

3.4 The Requirements Document

The requirements document is the official statement of what is required of the system developers. It should include both a definition of user requirements and a specification of the system requirements. It is NOT a design document. As far as possible, it should set up WHAT the system should do rather than HOW it should do it.

3.5 Users of a Requirement Document

A requirements document would be used by different types of persons such as Managers, Software Developers and System Users for different purposes. Table 3.2 shows the users of a requirements document and what they would likely use the requirement document for.

Table 3.2: Different users of a requirement document

Requirements Document User	Possible tasks
System Customers	Specify the requirements and read them to confirm they meet their needs and expectations. They also specify changes to the requirements
Managers	Use the requirements document to plan a bid for the system and to plan the system development process

System engineers	Use the requirements to understand the system to be developed
System test engineers	Use the requirements to develop and implement validation tests for the developed system
System maintenance engineers	Use the requirements to help understand the system and how the different parts and modules of the system interact

3.6 User Requirements

- i. Should describe functional and non-functional requirements in such a way that they are understandable by system users who don't have detailed technical knowledge.
- ii. User requirements are defined using natural language, tables and diagrams as these can be understood by all users.

3.7 IEEE Requirements Standards

In 1998, IEEE published a standard for the organisation of requirements documents. This standard is generic and can be adapted to specific uses. A sample standard is as follows;

- Introduction.
- General description
- Specific requirements
- Appendices
- Index.

Discussion

List ten guidelines for writing requirements

Scenario

As the leader of a team to develop a new proposed system for a client, you received a note of requirements presented by the users of the system.
Can this requirement be enough to embark on technical design of the system?
Give reasons for your answer.



4.0 Self-Assessment Exercise(s)

1. one of these is not a guideline for writing requirement

- a. use a standard template
- b. use simple and direct sentences
- c. use professional jargons
- d. avoid ambiguity

2. Which type of requirements are concerned with the specific services delivered by the system to the system users.

- a. Non-functional
- b. Functional
- c. System
- d. Operational



5.0 Conclusion

This unit has exposed us to the guidelines and standards for writing requirements . It has also given us an idea of what different types of stakeholders in a software development project would use the requirement document for.



6.0 Summary

Requirements set out what the system should do and define constraints on its operation and implementation. User requirements are high-level statements of what the system should do. User requirements should be written using natural language, tables and diagrams. System requirements are intended to communicate the functions that the system should provide.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component?* *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A., (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Unit 5: Requirement Elicitation and Analysis Contents

Contents

- 1.0 Introduction
- 2.0 Intended Learning Outcome (ILOs)
- 3.0 Main Content
 - 3.1 Requirements Engineering Processes
 - 3.2 Requirements Elicitation and Analysis
 - 3.3 Requirements Elicitation and Analysis Process
 - 3.4 Challenges of Requirements Elicitation
- 4.0 Self-Assessment Exercise
- 5.0 Conclusion
- 6.0 Summary
- Further Readings

7.0



1.0 Introduction

In this unit we will learn a few basic concepts of requirements engineering processes.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:
describe requirements engineering processes
explain requirements elicitation and its processes
discuss the challenges of requirements elicitation.



3.0 Main Content

3.1 Requirements Engineering Processes

The processes used for requirements engineering (RE) vary widely depending on the application domain, the people involved and the organisation developing the requirements. The output of the RE process is a system requirements document.

There are a number of generic activities common to all processes. These activities are actually interleaved and iterative. They include the following:

- a. Requirements elicitation and analysis
- b. Requirements specification
- d. Requirements validation

figure 3.6 shows the requirement engineering process flow. Similarly, figure 3.7 shows the interleaved nature of the requirements engineering process with a spiral. Attention is paid to business requirements early in the spiral with system requirements coming later in the process.

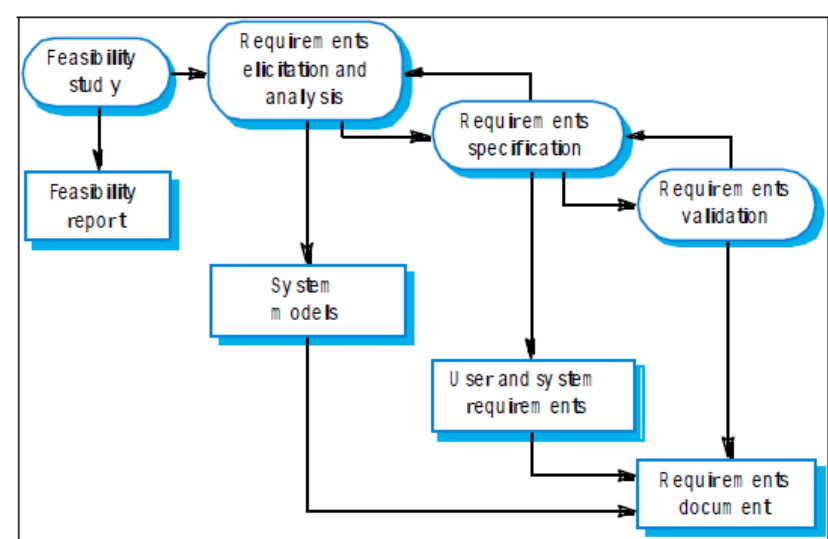


Figure 3.6: The requirements engineering process

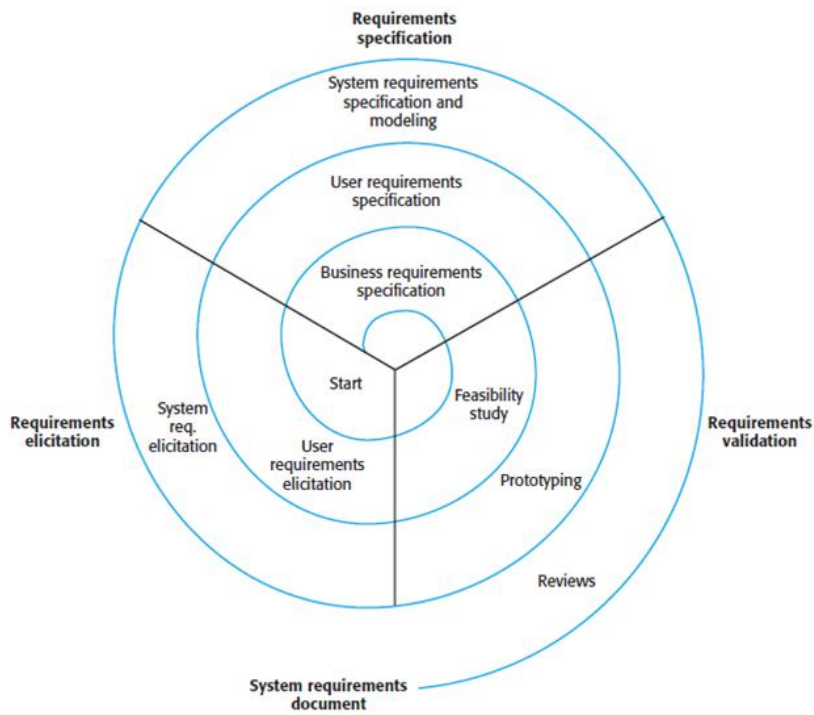


Figure 3.7: The spiral view of the requirements engineering process

3.2 Requirements Elicitation and Analysis

This is a way of trying to understand the job roles of the stakeholders in an organisation and how a new system could support their work. Software engineers work with stakeholders to find out more about the application domain, services and system properties, work activities of stakeholders, required performance of the system and the hardware constraints of the system.

During requirements elicitation, software engineers work with stakeholders to find out about the application domain, work activities, the services and system features that stakeholders want, the required performance of the system, hardware constraints, and etc.

3.3 Requirements Elicitation and Analysis Process

The process activities of the requirements elicitation process involves the following;

- a. **Requirements discovery and understanding:** this is also called requirements gathering. It involves interacting with stakeholders (clients and system users) to know their expectations from the software.
- b. **Requirements classification and organization:** this involves organising the unstructured collected requirements into groups of related.
- c. **Requirements prioritization and negotiation:** this involves prioritizing requirements by finding and resolving requirements conflicts through negotiations. This activity is made possible because many stakeholders are involved in a system. This often leads to conflicts in requirements.
- d. **Requirements documentation:** requirements are documented and input into the next phase of the requirement engineering process.

3.4 Challenges of Requirements Elicitation

- a. Generally, stakeholders don't often know what they want from a system; and most times, they may find it difficult to articulating what they want.
- b. Stakeholders often express requirements in their language, making it difficult for requirements engineers without experience in the customer's domain to understand the requirements.
- c. Different stakeholders, with diverse requirements, may express their requirements in different ways. Requirements engineers have to discover all potential sources of requirements and discover commonalities and conflict.
- d. Because requirements are always changing, the importance of particular requirements may change. New requirements may emerge from stakeholders who were not originally consulted.

Discussion

Why are the requirements engineering processes interleaved?

Scenario

As the leader of a team of requirements engineers for a new software project, you recently observed there were many disagreeing stakeholders, who have produced several conflicting requirements. Explain how you would resolve this challenge.



4.0 Self-Assessment Exercise(s)

1. Which of these is not a requirements elicitation and analysis process activity?

- e. Requirements validation
- f. Requirements discovery and understanding
- g. Requirements documentation
- h. Requirements discovery and understanding

2. Which of these is not a requirements engineering process activity?

- a. Requirements elicitation and analysis
- b. Requirements specification
- c. Requirements documentation
- d. Requirements validation



5.0 Conclusion

In this unit, you have learned about requirements engineering processes. You have also been able to understand requirements elicitation and analysis, together with the processes involved in requirements elicitation.



6.0 Summary

Requirements engineering processes are usually an iterative process and mostly interleaved.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component? Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A, (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Online Resources

https://www.tutorialspoint.com/software_engineering/software_engineering_tutorial.pdf

Module 4: Requirements Engineering Processes

Module Introduction

This module discussed the topic of requirements engineering and requirements engineering processes further.

Unit 1: Requirements elicitation - Observation

Unit 2: Requirements Elicitation - Interviewing

Unit 3: Requirements Analysis - Viewpoints

Unit 4: Requirements Specification

Unit 5: Requirements Validation

Unit 1: Requirements elicitation - Observation

Contents

1.0 Introduction

2.0 Intended Learning Outcome (ILOs)

3.0 Main Content

3.1 Requirements Elicitation Techniques

3.2 Observation

3.3 Effectiveness of Observation

3.4 Shortcomings of Observation

4.0 Self-Assessment Exercise

5.0 Conclusion

6.0 Summary

7.0

Further Readings



1.0 Introduction

In this unit, we will discuss requirements elicitation techniques. We will also discuss observation as a requirement elicitation technique.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

describe requirements elicitation techniques
explain the concept of observation and its use
know the shortcomings of observation



3.0 Main Content

3.1 Requirements Elicitation Techniques

Requirements elicitation involves meeting with different stakeholders to discover information about the proposed system. You may also need to consult different documents to get information about the existing system and its usage. This could supplement the information from the stakeholders. To achieve this, you will need to spend time in order to understand how people work, including how they use the existing system and how they would accommodate the new system.

The two basic approaches to collecting information from stakeholders are;

1. **Observation:** where the requirements engineer watch people doing their job to see what they do, what they use, and how they use them etc.
2. **Interviewing:** where the requirements engineer talks to people, to know what they do.

3.2 Observation

Observation is a technique that can be used to understand operational processes and help derive software requirements that support these processes. In observation, analysts or requirements engineers immerse themselves in the work environment where the system will be used, thereby observing the day-to-day work and actual tasks which participants get involved in. This helps them discover implicit system requirements that reflect the actual ways people work, rather than the formal processes defined by the organization. Observation is an important tool because people always find it difficult articulating what they do.

3.3 Effectiveness of Observation

Observation is highly effective in discovering these two types of requirements:

1. Requirements derived from the way in which people work, rather than the way in which business process definitions say they should work. This is because in practice, people do not follow formal processes. They follow processes that help them compete their tasks easily.
2. Requirements derived from cooperation and awareness of other people's activities.

3.4 Shortcomings of Observation

Does not support innovation: though, observation is helpful, especially when trying to understand existing systems, it does not support innovation.

Not effective for discovering broader organizational or domain requirements: this is because of its focus on end users. Consequently, observation should be used as one of a number of techniques for requirements elicitation.

Discussion

Why is it necessary to meet with various stakeholders during requirement elicitation process?

Scenario

After interviewing various stakeholders, you realise the information gathered from the stakeholders does not give detail of how the stakeholders actually use the existing system. What other approach would you likely use to acquire more information about how the end users use the system?



4.0 Self-Assessment Exercise(s)

1. the basic ways of gathering information during requirements elicitation is observation and _____

- a. analysis
- b. interviewing
- c. specification
- d. documentation

2. Which of these activity is highly helpful in understanding how the end users actually do their work, rather than how the system model says they should do the work?

- a. interviewing
- b. observation
- c. system analysis
- d. specification



5.0 Conclusion

In this unit, you have learned about requirements elicitation techniques. You have also understood the concept of observation and when it should be used.



6.0 Summary

The two basic approaches of gathering information from stakeholders are observation and interviewing. Observation is very helpful in understanding how the end users actually complete their work, rather than how they should complete it.



7.0 Further Readings

Cheng, B. H. C., and Atlee, J. M. (2007). *Research Directions in Requirements Engineering*. Proc. Conf. on Future of Software Engineering, IEEE Computer <http://dx.doi.org/10.1109/FOSE.2007.17>.

Robertson, S., and Robertson, J. (2013). *Mastering the Requirements Process* (3rd ed) Addison-Wesley

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Unit 2: Requirements Elicitation - Interviewing

Contents

1.0 Introduction	
2.0 Intended Learning Outcome (ILOs)	
3.0 Main Content	
3.1 Interviewing	
3.2 Types of Interview	
3.3 Interviews in Practice	
3.4 Effective Interviewers	
4.0 Self-Assessment Exercise	
5.0 Conclusion	
6.0 Summary	7.0
Further Readings	



1.0 Introduction

What you will learn in this unit concerns interviewing. You will also learn the types of interview and the attributes of an effective interviewer.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- explain the term ‘interviewing’
- describe the 2 types of interview
- explain interviews in practice
- give the attributes of an effective interviewer.



3.0 Main Content

3.1 Interviewing

Interviewing is one the tools requirements elicitation. In formal or informal interviewing, the requirements engineering team puts questions to stakeholders about the system that they use and the system to be developed. This is done to get information about the current system and the system to be developed.

3.2 Types of Interview

There are basically two types of interview:

Closed interviews; where a pre-defined set of questions are answered.

Open interviews; where there is no pre-defined agenda and a range of issues are explored with stakeholders.

3.3 Interviews in Practice

In practice interviews are normally a mix of closed and open-ended interviewing. Indeed, interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system.

Interviews are not very good for understanding domain requirements.

Requirements engineers cannot understand specific domain terminology.

Some domain knowledge is so familiar that people find it hard to articulate or sometimes think that it isn't worth articulating.

3.4 Effective Interviewers

- i. Interviewers should be open-minded, willing to listen to stakeholders and should not have pre-conceived ideas about the requirements.
- ii. They should prompt the interviewee with a question or a proposal and should not simply expect them to respond to a question such as 'what do you want'.

Discussion

List the shortcomings of interviewing in requirements elicitation process.

Scenario

Which type of requirements would be better acquired by observation, rather than interviewing?



4.0 Self-Assessment Exercise(s)

1. the basic ways of gathering information during requirements elicitation is interviewing and
 - a. analysis
 - b. observation

- c. specification
- d. documentation

2. the types of interviews include closed and _____ interviews?

- a. open
- b. long
- c. timed
- d. specification



5.0 Conclusion

In this unit you have learned about interviewing. You have also been able to identify the types of interview and the attributes of an effective interviewer.



6.0 Summary

Interviewing is a requirements elicitation tool. It basically has two types, namely closed and open interviews.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component?* *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A, (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Unit 3: Requirements Analysis - Viewpoints

Contents

1.0 Introduction	
2.0 Intended Learning Outcome (ILOs)	
3.0 Main Content	
3.1 Viewpoints	
3.2 Types of Viewpoints	
3.3 Viewpoint Identification	
3.4 LIBSYS Viewpoint Hierarchy	
4.0 Self-Assessment Exercise	
5.0 Conclusion	
6.0 Summary	7.0
Further Readings	



1.0 Introduction

In this unit, you will learn about viewpoints, together with viewpoints identification and types.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:
explain the concept of viewpoints
list the types of viewpoints
identify viewpoints.



3.0 Main Content

3.1 Viewpoints

Viewpoints are used in the simplifying and the analysing of requirements. They are a way of structuring the requirements to represent the perspectives of different stakeholders. Stakeholders are classified into different viewpoints.

3.2 Types of Viewpoints

i. Interactor Viewpoints

People or other systems that interact directly with the system. In an ATM, the customer and the account database are interactor viewpoints.

ii. Indirect Viewpoints

Stakeholders who do not use the system themselves but who influence the requirements. In an Automated Teller Machine, bank management and security staff are indirect viewpoints.

iii. Domain Viewpoints

Domain characteristics and constraints that influence the requirements. In an Automated Teller Machine, an example would be standards for inter-bank communications.

3.3 Viewpoint Identification

Identify viewpoints using

- Providers and receivers of system services;
- Systems that interact directly with the system being specified;
- Regulations and standards;
- Sources of business and non-functional requirements.
- Engineers who have to develop and maintain the system;
- Marketing and other business viewpoints.

3.4 LIBSYS Viewpoint Hierarchy

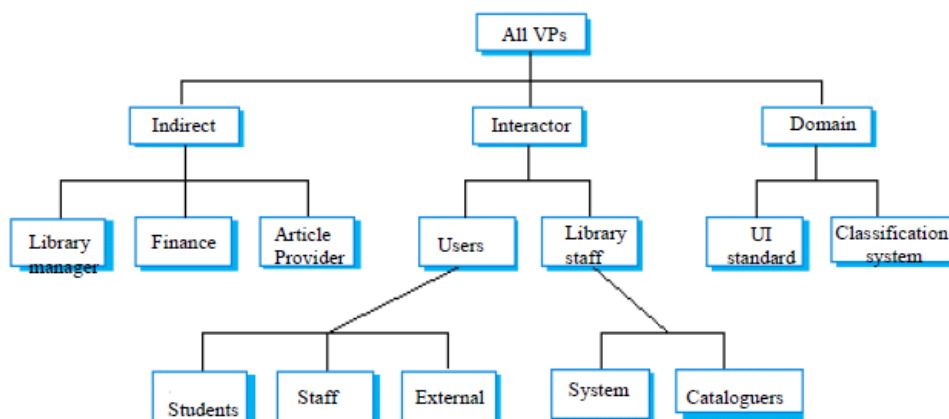


Figure 4.1: LIBSYS viewpoint hierarchy

Discussion

What would you use viewpoints for?

Scenario

Identify and discuss possible viewpoints for a health management system deployed in a general hospital?



4.0 Self-Assessment Exercise(s)

1. one of these is a type of a viewpoint

- a. Domain
- b. application
- c. direct
- d. analysis

2. one of these is not a way of identifying viewpoints

- a. knowing the sources of business and non-functional requirements
- b. knowing providers and receivers of system services
- c. knowing the data gathering approach to use
- d. knowing the systems that interact directly with the system being specified



5.0 Conclusion

In this unit you have learned about viewpoints, types of viewpoints and viewpoints identification.



6.0 Summary

Viewpoints are great tools for analysing and structuring requirements to represent the perspectives of different stakeholders.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component?* *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A., (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Unit 4: Requirements Specification

Contents

- 1.0 Introduction
- 2.0 Intended Learning Outcome (ILOs)
- 3.0 Main Content
 - 3.1 Requirements Specification
 - 3.2 Features of a Specification Document
 - 3.3 Ways of Writing System Specification
 - 3.4 Software Requirements Specification
- 4.0 Self-Assessment Exercise
- 5.0 Conclusion
- 6.0 Summary
- Further Readings

7.0



1.0 Introduction

This unit discusses specification as a requirements engineering process activity. We also look at the various ways of writing specifications.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:
describe software specification
describe the properties of a specification documents
describe the various ways of writing a software specification document



3.0 Main Content

3.1 Requirements Specification

Specification is the second process activity in traditional requirements engineering process. it is the process of writing the user and system requirements in a requirements document.

3.2 Features of a Specification Document

System specification should describe the external behavior of the system only, and its constraints.

The user and system requirements written should be clear, consistent, and complete. This is to avoid conflicts, as stakeholders rarely understand and interpret requirements in a similar way.

Requirements document should not contain details of the system architecture or design. It should also not be concerned with how the system should be designed or implemented.

It is most beneficial, not to use software jargon and structured notations when writing specifications.

Natural language, tables, forms, and intuitive diagrams should be used when writing specification.

3.3 Ways of Writing System Specification

Table 4.1 shows the various ways of writing system specifications and their descriptions.

Table 4.1: ways of writing system specification

Specification	Specification description
Natural language	Requirements are written using numbered sentences in natural language. Each sentence contains one requirement. Vague and ambiguous.
Structured natural language	Requirements are written on a template or standard form in natural language. Each field contains information about an aspect of the requirement.
Graphical notations	Use of graphical models, and text annotations to define functional requirements for the system. Unified Modeling Language (sequence diagram and use case diagram) is used.
Mathematical specifications	Use of mathematical concepts such as finite-state machines or sets. reduce ambiguity in a requirements document. Customers don't understand mathematical specification, and

	cannot confirm whether it represents what they want.
--	--

3.4 Software Requirements Specification

The software requirements specification also called the software requirements document is a document created after the specification activity of the requirements engineering process. It is an official document containing what the system developers should implement. Guidelines for developing the software requirements document had already been discussed in unit 4 of module two.

Discussion

Why is it important not to use software jargons during the requirements specification activity of the requirements engineering process?

Scenario

As the leader of a requirements engineering team for a new software project, would you use natural language or structured natural language to develop your team's software requirements specification document?

Give reasons for your choice.



4.0 Self-Assessment Exercise(s)

1. one of these is not a way of writing software requirements specification.

- a. Structured natural language
- b. Graphical notations
- c. Low-level language
- d. Natural language

2. which of these reduces ambiguity in a requirements document?

- a. Graphical notations
- b. Structured natural language
- c. Natural language
- d. Mathematical specifications



5.0 Conclusion

In this unit you have learned about specification, and the ways of writing software specifications.



6.0 Summary

Requirements specification follows requirements elicitation and analysis. It is normally done to produce a statement of what the system developers should implement.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component?* *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssi re, J., (July 2003). “*A Security Framework for Reflective Java Applications,*” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability,*”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A, (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Unit 5: Requirements Validation

Contents

1.0 Introduction	
2.0 Intended Learning Outcome (ILOs)	
3.0 Main Content	
3.1 Requirements Validation	
3.2 Requirements Check	
3.3 Requirements Validation Techniques	
3.4 Requirements Change and Reviews	
3.5 Requirements Management Planning	
3.6 Requirements Change Management	
4.0 Self-Assessment Exercise	
5.0 Conclusion	
6.0 Summary	7.0
Further Readings	



1.0 Introduction

This unit is focused on requirements validation, validation techniques and requirements management.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:
explain requirements validation
describe requirements checking
discuss requirements management.



3.0 Main Content

3.1 Requirements Validation

Requirements validation is concerned with demonstrating that the requirements define the system that the customer really wants.

Requirements error costs are high, so validation is very important. Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

3.2 Requirements Check

During a requirements validation process, various checks should be carried out on the requirements contained in the requirements document. These checks include the following;

- i. Validity:** checks whether the system provides functions which best supports the customer's needs.
- ii. Consistency:** checks for requirements conflicts.
- iii. Completeness:** checks whether all functions required by the customer are included.
- iv. Realism:** checks whether the requirements can be implemented given the available budget and technology.
- v. Verifiability:** checks whether the requirements can be tested.

3.3 Requirements Validation Techniques

The following are requirements validation techniques. These techniques can either be combined or used individually.

- i. Requirements reviews:** this involves a systematic analysis of the requirements by a review team, in order to identify errors and inconsistencies.
- ii. Prototyping:** This involves using an executable model of the system to check requirements. Customers and end-users make use of the model to see if it meets their expectations.
- iii. Test-case generation:** Developing tests for requirements to check testability. If the requirements are testable, then the system is implementable. Otherwise, it may be difficult to implement the system.

3.4 Requirements Change and Reviews

Requirements change continually due to changing stakeholders' understanding of the problem and other errors like omissions in the original requirements document. This means regular reviews should be held while the requirements definition is being formulated.

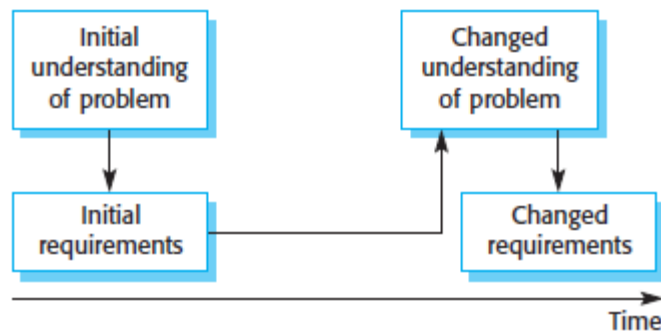


Figure 4.2: Requirements evolution

Usually, changes on system and system requirements arise due to the following;

1. Changing business and technical environment of the system after installation.
2. The people who initiate and pay for a system are rarely the end-users of the system. Consequently, the initiators' requirements may conflict with end user requirements. New features may have to be added to support the end-user goals.
3. Large systems usually have varying stakeholder groups, with varying requirements which often conflict. The final requirements must be a compromise of these various requirements. This results in changing stakeholder requirements.

3.5 Requirements Management Planning

Requirements management planning is the process of managing changing requirements during the requirements engineering process and system development. To effectively plan your requirement management, such issues as the following need to be considered;

1. **Requirements identification:** Each requirement must be uniquely identified and cross-referenced with other requirements.
2. **A change management process:** there should be a defined set of activities that assess the impact and cost of changes.

3. **Traceability policies:** Recorded policies that define the relationships between each requirements and between the requirements and the system design. This policy also defines how records should be maintained.

4. **Tool support:** Tools that should be used for the requirements management. These tools may include special requirements management systems and spreadsheet. This is necessary because requirements management involves the processing of large amount of data.

3.6 Requirements Change Management

Every proposed system and requirements change after the approval of the requirements document must go through requirements change management. This is important to make sure the requirements change is worth carrying out. The benefit of this is that all changes to the requirements documents are made in a controlled way.

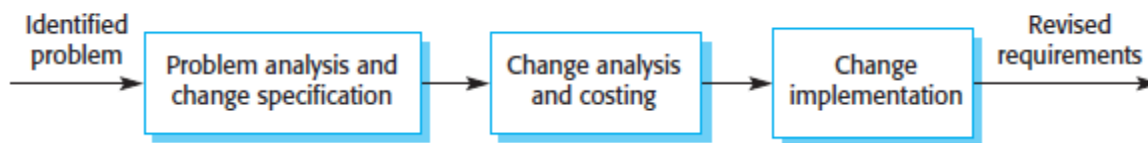


Figure 4.3: Requirements change management

The three stages of a change management process are as follow:

1. **Problem analysis and change specification:** after identifying a requirements problem, the change proposal is analyzed and checked for validity. This analysis is communicated to the change initiator who may now give a detailed requirements proposal or withdraw the request.

2. **Change analysis and costing:** Here, the effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. The cost of this proposed change is also estimated. if this is worthwhile, the requirements change is proceeded with, else it is dropped.

3. **Change implementation:** Here, the requirements document is updated with the change, so that it can contain the new requirements.

Discussion

Why is it important to apply requirements change management on every proposed requirements or system change?

Scenario

While developing requirements for a new software project, you noticed with your team and stakeholders that a certain requirement may need to be changed. On applying requirements change management processes, you realised that the cost of implementing this change far outweighs the benefit of this change. What will be your decision and recommendation on the proposed requirements change?



4.0 Self-Assessment Exercise(s)

1. which of these is not a requirements validation technique?
 - a. Requirements reviews
 - b. Prototyping
 - c. Automatic review
 - d. Test-case generation

2. _____ is the process of managing changing requirements during the requirements engineering process and system development.
 - a. Requirements engineering
 - b. Requirements change
 - c. Requirements management planning
 - d. Requirement engineering process



5.0 Conclusion

In this unit you have learnt about the requirements validation, requirements change, and requirement management.



6.0 Summary

Requirements validation is the last stage of the requirements engineering processes. It involves checking that the produced requirements document is valid.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component? Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A., (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Module 5: Software Engineering

Module Introduction

This module introduces you to the topic of system modeling and design.

- Unit 1: System modeling
- Unit 2: Formal Methods
- Unit 3: Introduction to Architectural Design and Models
- Unit 4: Sub-Systems and Modules
- Unit 5: Software Design

Unit 1: System modeling

Contents

- 1.0 Introduction
 - 2.0 Intended Learning Outcome (ILOs)
 - 3.0 Main Content
 - 3.1 System Modeling
 - 3.2 Relevance of System Modelling
 - 3.3 Model Types
 - 3.4 Context Models
 - 3.5 Process Models
 - 3.6 Behavioural Models
 - 3.7 Data Processing Models
 - 3.8 Application of Data Flow Diagrams
 - 4.0 Self-Assessment Exercise
 - 5.0 Conclusion
 - 6.0 Summary
 - Further Readings
- 7.0



1.0 Introduction

In this unit, the student will gain knowledge of the relevance of system modeling. The unit describes the context, process, behavioural and data processing models.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- Describe system modeling
- state the relevance of system modeling
- list types of models
- explain the context, process, behavioural and data processing models.



3.0 Main Content

3.1 System Modeling

System modeling is basically the process of developing abstract models of a system; where each model presents a perspective of the system. Usually, some kind of graphical notations like the Unified Modeling Language (UML) are used. Other ways of modeling a system include the use of formal models (mathematical expressions), in this case formal methods.

3.2 Relevance of System Modelling

System modelling helps the analyst to understand the functionality of the system.

Models are used to communicate with customers.

Different models present the system from different perspectives.

External perspective shows the system's context or environment.

Behavioural perspective shows the behaviour of the system.

Structural perspective shows the system or data architecture.

3.3 Model Types

- a. Data processing model: shows how the data is processed at different stages.
- b. Composition model: shows how entities are composed of other entities.
- c. Architectural model: shows principal sub-systems.
- d. Classification model: shows how entities have common characteristics.
- e. Stimulus/response model: shows the system's reaction to events.

3.4 Context Models

Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries.

Social and organisational concerns may affect the decision on where to position system boundaries. Architectural models show the system and its relationship with other systems.

3.5 Process Models

Process models show the overall process and the processes that are supported by the system.

Data flow models may be used to show the processes and the flow of information from one process to another.

3.6 Behavioural Models

Behavioural models are used to describe the overall behaviour of a system.

Two types of behavioural model are:

- Data processing models that show how data is processed as it moves through the system;
- State machine models that show the system's response to events.

These models show different perspectives so both of them are required to describe the system's behaviour

3.7 Data Processing Models

Data flow diagrams (DFDs) may be used to model the system's data processing. These show the processing steps as data flows through a system.

Data flow diagrams are an intrinsic part of many analysis methods. DFDs have simple and intuitive notation that customers can understand. They show end-to-end processing of data.

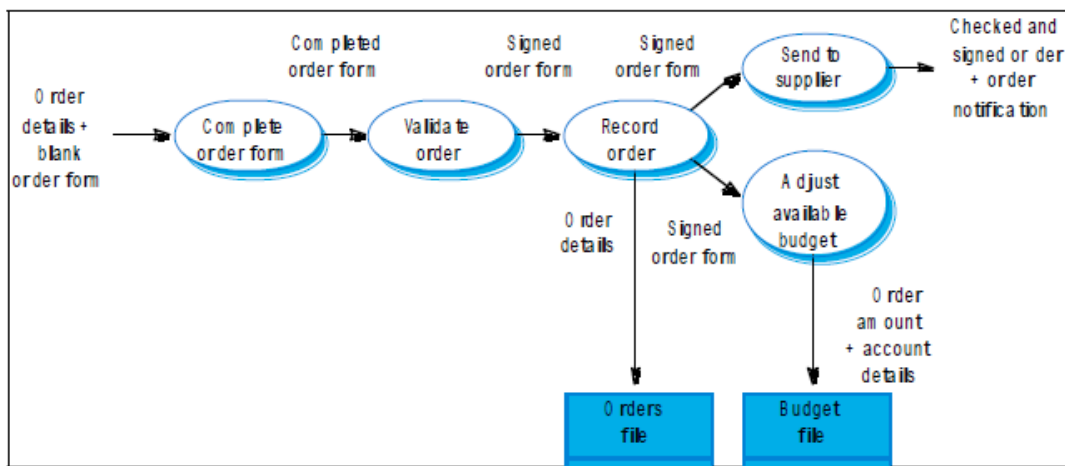


Figure 5.1: Order processing data flow diagram

3.8 Application of Data Flow Diagrams

DFDs model the system from a functional perspective.

Tracking and documenting how the data associated with a process is helpful to develop an overall understanding of the system.

Data flow diagrams may also be used in showing the data exchange between a system and other systems in its environment.

Discussion

Why are system models important in trying to demonstrate a proposed system to the customers and users?

Scenario

List and describe some of the diagram types in the Unified Modeling Language (UML).



4.0 Self-Assessment Exercise(s)

1. One of these is not a model type?
 - a. Data processing model
 - b. Architectural model
 - c. Implementation model
 - d. Composition model

2. System modeling can be done using any of these approaches
 - a. graphical notation and formal model
 - b. software application and hardware
 - c. Requirements engineering and programming
 - d. validation and testing



5.0 Conclusion

In this unit, we have given a basic understanding of system modeling, relevance of system modeling and the types of models



6.0 Summary

A model is an abstract system view. Complementary types of model provide different system information. Context models show the position of a system in its environment with other systems and processes. Data flow models may be used to model the data processing in a system.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component? Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jermaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A., (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Unit 2: Formal Methods

Contents

1.0 Introduction	
2.0 Intended Learning Outcome (ILOs)	
3.0 Main Content	
3.1 Formal Methods	
3.2 Formal Specification	
3.3 Formal Specification Languages	
3.4 Advantages of Formal Specification	
3.5 Acceptance of Formal Methods	
3.6 Use of Formal Methods	
4.0 Self-Assessment Exercise	
5.0 Conclusion	
6.0 Summary	7.0
Further Readings	



1.0 Introduction

This unit discusses formal methods. It also covers acceptance and use of formal methods.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

Describe formal methods

enumerate formal methods

discuss the acceptance of formal methods

explain the use of formal methods.



3.0 Main Content

3.1 Formal Methods

Formal methods are mathematical approaches to software development where a formal model of the software is defined. The model is then analysed formally to search for errors and inconsistencies. Formal methods start with a mathematical system model; which acts as a system specification. In creating this model, the user requirements which are expressed in natural language, tables, and diagrams are translated into mathematical expressions with defined semantics.

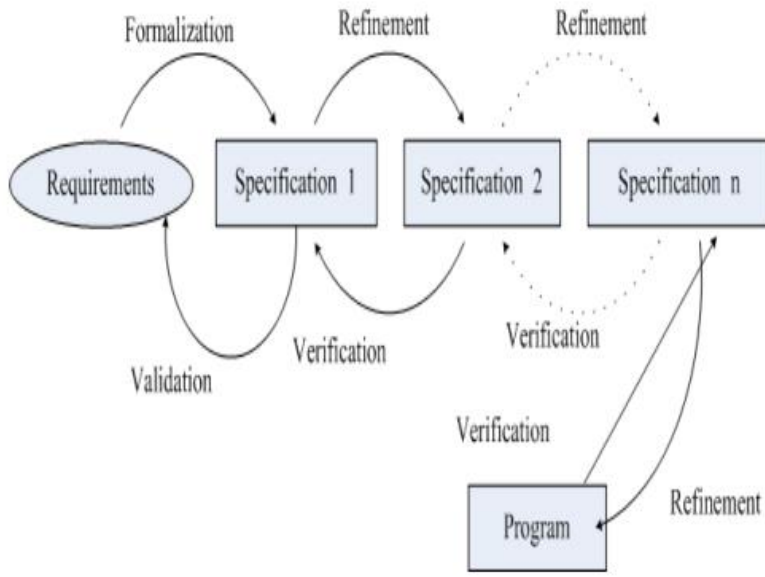


Figure 5.2: Software design using formal methods

Formal methods include:

Formal specification;
 Specification analysis and proof;
 Transformational development;
 Program verification.

3.2 Formal Specification

Formal specification is part of a more general collection of techniques that are known as ‘formal methods’. These are all based on mathematical representation and analysis of software. It is an unambiguous description of what the system should do.

3.3 Formal Specification Languages

The following are examples formal specification languages:

1. Knowledge Based Software Assistant (KBSA)
2. Java Modeling Language (JML)
3. Common Algebraic Specification Language (CASL)
4. Autonomic System Specification Language (ASSL)

3.4 Advantages of Formal Specification

1. by developing a formal specification in detail, one develops a deep and detailed

understanding of the system requirements.

2. easy analysis to discover inconsistencies and incompleteness of specification, since the specification is expressed in a language with formally defined semantics
3. Program testing costs can be reduced since it has been verified against its specification.
4. removes inconsistencies between a program and its specifications.

3.5 Acceptance of Formal Methods

Formal methods have not become mainstream software development techniques as was once predicted. Other software engineering techniques have been successful at increasing system quality. Hence the need for formal methods has been reduced. Market changes have made time-to-market rather than software with a low error count the key factor. Formal methods do not reduce time to market and the scope of formal methods is limited. They are not well-suited to specifying and analysing user interfaces and user interaction. Formal methods are still hard to scale up to large systems.

3.6 Use of Formal Methods

The principal benefits of formal methods are in reducing the number of faults in systems. Consequently, their main area of applicability is in critical systems engineering. There have been several successful projects where formal methods have been used in this area. In this area, the use of formal methods is most likely to be cost-effective because high system failure costs must be avoided.

Discussion

Why has formal methods not gained the desired acceptability in software engineering?

Scenario

List and describe three benefits of formal methods for an account management software.



4.0 Self-Assessment Exercise(s)

1. which of these is not a formal method?
 - a. Formal specification;
 - b. Specification analysis and proof;
 - c. agile development;
 - d. Program verification.
2. what is the principal benefits of formal methods?

- a. reduced the number faults
- b. supports faster system implementation
- c. suitable for large software projects
- d. reduced software delivery time



5.0 Conclusion

In this unit you have learned about formal methods and their use.



6.0 Summary

Though formal methods were proposed to dominate software engineering practice; they have not really enjoyed acceptability as forecasted.



7.0 Further Readings

Abrial, J. R. (2009). Faultless Systems: Yes We Can. *IEEE Computer* 42 (9) pp 30–36

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component?* *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications,*” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A, (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Sher, A. K., and Nazir, A. Z. (2007). Promotion of Local to Global Operation in Train Control System. *Journal of Digital Information Management*

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Unit 3: Introduction to Architectural Design and Models

Contents

1.0 Introduction

2.0 Intended Learning Outcome (ILOs)

3.0 Main Content

 3.1 Architectural Design

 3.2 Software Architecture

 3.3 Architectural Models

 3.4 Domain-Specific Model

 3.5 Architectural Design Decisions

4.0 Self-Assessment Exercise

5.0 Conclusion

6.0 Summary

Further Readings

7.0



1.0 Introduction

This unit describes the notion of architectural design. You will learn about software architecture and the common architectural models. Also, we will cover architectural design decisions.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- explain the notion of architectural design
- define software architecture
- describe the common architectural models
- list the architectural design decisions.



3.0 Main Content

3.1 Architectural Design

Architectural design is the process of designing a system organization that satisfies the functional and non-functional requirements of a software system. It involves identifying the sub-systems and major

system components making up the software system and the framework for sub-system control and communication.

Architectural design is an early stage of the software system design process. It represents the link between specification and design processes; and is often carried out in parallel with some specification activities.

3.2 Software Architecture

The software architecture is the fundamental framework for structuring the system. It is the output of the architectural design process

3.3 Architectural Models

Architectural models are used to document an architectural design. They include:

- a. Static structural model which shows the major system components.
- b. Dynamic process model which shows the process structure of the system.
- c. Interface model which defines sub-system interfaces.
- d. Relationships model such as a data-flow model that shows sub-system relationships.
- e. Distribution model that shows how sub-systems are distributed across computers.

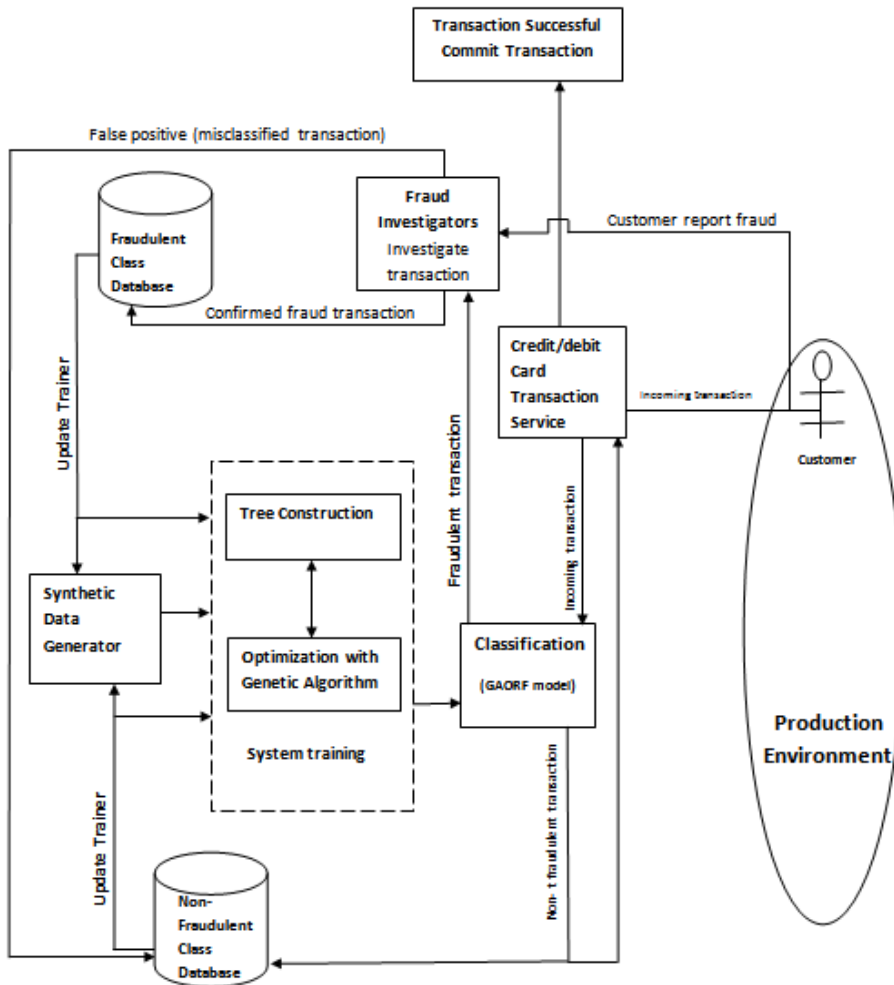


Figure 5.3: Sample architecture for a card fraud detection system

3.4 Domain-Specific Model

These are architectural models which are specific to some application domains.

Two types of domain-specific models are:

1. Generic models: Abstractions from a number of real systems that encapsulate the principal characteristics of these systems.
2. Generic models: Top-down or bottom-up models that are more abstract idealised. They provide a means of information about the class of the system and of comparing different architectures.

3.5 Architectural Design Decisions

Architectural design is a creative process so the process differs depending on the type of system being developed. Architectural design decisions include decisions on the application architecture, the distribution and the architectural styles to be used.

A number of common decisions span all design processes. These include:

1. Is there a generic application architecture that can be used?
2. How will the system be distributed?
3. What architectural styles are appropriate?
4. What approach will be used to structure the system?
5. How will the system be decomposed into modules?
6. What control strategy should be used?
7. How will the architectural design be evaluated?
8. How should the architecture be documented?

Discussion

What is the main use of architectural design

Scenario

List and describe any two architectural models



4.0 Self-Assessment Exercise(s)

1. _____ is the process of designing a system organization that satisfies the functional and non-functional requirements of the system.?
 - a. Data processing model
 - b. Architectural model
 - c. Architectural design
 - d. Composition model
2. _____ is the fundamental framework for structuring a system; and also, the output of the architectural design process.

- a. Architectural design
- b. software architecture
- c. hardware architecture
- d. structural architecture



5.0 Conclusion

In this unit you have learned about architectural design. You have also learned about architectural models and design decisions.



6.0 Summary

Architectural design is important for one to understand the different components and modules that make up a software system and how data flow across the different components and modules.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component?* *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications,*” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability,*”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A, (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Nwogu, E. R. (2014). Improving the Security of the Internet Banking System Using Three-Level Security Implementation. *International Journal of Computer Science and Information Technology & Security* 4(6)

Nwogu, E. R., Nwachukwu, E. O., Ejiofor, V. E. (2019). An Improved Hybrid System for The Prediction of Debit and Credit Card Fraud. *Computing, Information Systems & Development Informatics Journal* 10(3)

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Unit 4: Sub-Systems and Modules

Contents

- 1.0 Introduction
- 2.0 Intended Learning Outcome (ILOs)
- 3.0 Main Content
 - 3.1 What is a Sub-System?
 - 3.2 What is a Module?
 - 3.3 Modular Decomposition
 - 3.4 Modular Decomposition Models
- 4.0 Self-Assessment Exercise
- 5.0 Conclusion
- 6.0 Summary
- Further Readings

7.0



1.0 Introduction

This unit will focus on sub-systems and modules. You will also learn about modular decomposition.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- a sub-system
- a module
- modular decomposition
- modular decomposition models.



3.0 Main Content

3.1 What is a Sub-System?

A sub-system is a system in its own right whose operation is independent of the services provided by other sub-systems.

3.2 What is a Module?

A module is a system component that provides services to other components but would not normally be considered as a separate system.

3.3 Modular Decomposition

Modular decomposition refers to the process whereby sub-systems are decomposed into modules.

3.4 Modular Decomposition Models

There are two modular decomposition models covered in this unit:

An object model where the system is decomposed into interacting objects;

A pipeline or data-flow model where the system is decomposed into functional modules which transform inputs to outputs.

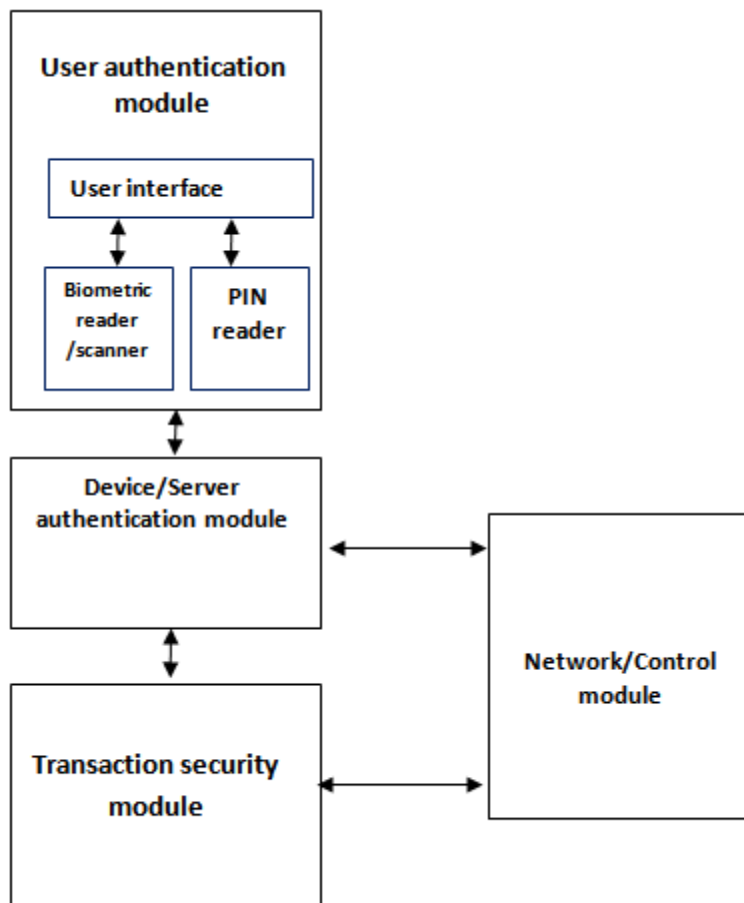


Figure 4.6: sample module interaction in a software system

Discussion

List and describe the two modular decomposition models covered in this unit.

Scenario

How could sub-systems possibly interact in a typical system?



4.0 Self-Assessment Exercise(s)

1. _____ is a system component that provides services to other components but would not normally be considered as a separate system.
 - a. A module
 - b. A system
 - c. An architecture
 - d. A model

2. _____ refers to the process whereby sub-systems are decomposed into modules.
 - a. modular decomposition
 - b. modular architecture
 - c. modular disintegration
 - d. structural decomposition



5.0 Conclusion

Specifically, you learned about sub – systems and modules. You also learned about modular decomposition.



6.0 Summary

What you have learned in this unit concerns sub-systems and modules.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component?* *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A., (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Nwogu, E. R. (2014). Improving the Security of the Internet Banking System Using Three-Level Security Implementation. *International Journal of Computer Science and Information Technology & Security* 4(6)

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Unit 5: Software Design

Contents

1.0 Introduction	
2.0 Intended Learning Outcome (ILOs)	
3.0 Main Content	
3.1 Software Design	
3.2 Stages of Design	
3.3 Design Phases	
3.4 Top-Down Design Technique	
4.0 Self-Assessment Exercise	
5.0 Conclusion	
6.0 Summary	7.0
Further Readings	



1.0 Introduction

This unit introduces you to software design. It explains the design stages and phases and describes the top-down design technique.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- describe software design
- identify the design stages
- state the design phases
- explain the concept of top-down design technique
- identify the design strategies.



3.0 Main Content

3.1 Software Design

Software design is a process of problem-solving and planning for a software solution. It entails deriving a solution which satisfies software requirements.

3.2 Stages of Design

A. Problem understanding

Look at the problem from different angles to discover the design requirements.

B. Identify one or more solutions

Evaluate possible solutions and choose the most appropriate depending on the designer's experience and available resources.

C. Describe solution abstractions

Use graphical, formal or other descriptive notations to describe the components of the design.

D. Repeat the process for each identified abstraction

3.3 Design Phases

- i. The Architectural design: Identify sub-systems.
- ii. Abstract specification: Specify sub-systems.
- iii. Interface design: Describe sub-system interfaces.
- iv. Component design: Decompose sub-systems into components.
- v. Data structure design: Design data structures to hold problem data.
- vi. Algorithm design: Design algorithms for problem functions.

3.4 Top-Down Design Technique

In principle, top-down design involves starting at the uppermost components in the hierarchy and working down the hierarchy level by level.

In practice, large systems design is never truly top-down. Some branches are designed before others. Designers reuse experience (and sometimes components) during the design process.

3.5 Design Strategies

i. Functional Design

The system is designed from a functional viewpoint. The system state is centralized and shared between the functions operating on that state.

ii. Object-Oriented Design

The system is viewed as a collection of interacting objects. The system state is decentralized and each object manages its own state. Objects may be instances of an object class and communicate by exchanging methods.

Discussion

Describe-object oriented design

Scenario

As the leader of a team of software engineers working on a new software project, would you consider top-down or bottom-up design?

State reasons for your answer.



4.0 Self-Assessment Exercise(s)

1. which design approach involves starting at the uppermost components in the hierarchy and working down the hierarchy level by level?

- a. functional design
- b. Architectural design
- c. top-down design
- d. bottom-up design

2. in _____ design, the system is viewed as a collection of interacting objects.

- a. Object-Oriented
- b. interaction-based
- c. functional
- d. architectural



5.0 Conclusion

In this unit you have learned about software design. You have also gained insight of design phases as well as top-down design technique.



6.0 Summary

What you have learned in this unit concerns software design and design strategies.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component? Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “A Security Framework for Reflective Java Applications,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A., (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetchbooks.com/software-engineering-methodology-thewatersluice-t573.html#754>.

Module 6: Software Engineering

Module Introduction

This module discussed the topics of software testing, software evolution and software maintenance.

Unit 1: Software Testing

Unit 2: Software Inspection

Unit 3: Software Reliability

Unit 4: Software Re-use

Unit 5: Software Evolution and Maintenance

Unit 1: Software Testing

Contents

1.0 Introduction

2.0 Intended Learning Outcome (ILOs)

3.0 Main Content

3.1 Verification and Validation

3.2 Verification and Validation Process

3.3 Software Inspections

3.4 Testing and Debugging

3.5 Testing Stages

4.0 Self-Assessment Exercise

5.0 Conclusion

6.0 Summary

Further Readings

7.0



1.0 Introduction

In this unit the student will be equipped with the knowledge of verification and validation and its principal objectives. The unit describes testing and debugging as well.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:
describe the terms verification and validation
understand the principal objective of verification and validation
distinguish between verification and validation
distinguish between testing and debugging
list the testing stages.



3.0 Main Content

3.1 Verification and Validation

In software testing, and software engineering, **Verification and Validation (V&V)** entail the process of checking that a software system meets specifications and fulfils its intended purpose.

Verification checks whether the final software product meets the stated functional and non-functional requirements (low-level checking) – i.e., you built the product right. This is done through static testing.

Validation ensures that the final product meets user expectations (high-level checking) – i.e., you built the right product. This is done through dynamic testing and other forms of review.

3.2 Verification and Validation Process

Verification and validation process is a whole life-cycle process which must be applied at each stage in the software process.

Verification and validation has two principal objectives:

- i. The discovery of defects in a system.
- ii. The assessment of whether or not the system is usable in an operational situation.

3.3 Software Inspections

Software inspections involve people examining the source representation with the aim of discovering anomalies and defects.

Software inspections do not require execution of a system so may be used before implementation. They may be applied to any representation of the system (requirements, design, test data, etc.). It is a very effective technique for discovering errors.

3.4 Testing and Debugging

Defect testing and debugging are distinct processes. Defect testing is concerned with confirming the presence of errors, while debugging is concerned with locating and repairing these errors. Debugging involves formulating hypotheses about program behaviour then testing these hypotheses to find the system error.

3.5 Testing Stages

- i. Unit testing:** testing of individual components
- ii. Module testing:** testing of collections of dependent components
- iii. Sub-system testing:** testing collections of modules integrated into sub-systems
- iv. System testing:** testing the complete system prior to delivery
- v. Acceptance testing:** testing by users to check that the system satisfies requirements.

Discussion

Why is it necessary to do software verification and validation?

Scenario

After confirming that a software system was built according to the stated functional and non-functional requirements, the users of the system still claimed the software does not serve their expectations. Has this software failed verification or validation test?



4.0 Self-Assessment Exercise(s)

1. _____ checks whether the final software product meets the stated functional and non-functional requirements?
 - a. Software Inspection
 - b. software examination
 - c. verification
 - d. Software analysis

2. _____ is concerned with locating and repairing errors in software products?
 - a. testing
 - b. debugging

- c. inspection
- d. validation



5.0 Conclusion

Verification and validation were considered in this unit. You have also learned about testing and debugging.



6.0 Summary

Verification and validation are important processes in software engineering. They ensure that the software product meets the stated specifications.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component? Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “A Security Framework for Reflective Java Applications,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A, (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Sher, A. K., and Nazir, A. Z. (2007). Promotion of Local to Global Operation in Train Control System. *Journal of Digital Information Management*

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetchbooks.com/software-engineering-methodology-thewatersluice-t573.html#754>.

Unit 2: Software Inspection

Contents

1.0 Introduction	
2.0 Intended Learning Outcome (ILOs)	
3.0 Main Content	
3.1 Software Inspection	
3.2 Inspection and Testing	
3.3 Inspection Pre-conditions	
3.4 Inspection Procedure	
3.5 Inspection Checklists	
4.0 Self-Assessment Exercise	
5.0 Conclusion	
6.0 Summary	7.0
Further Readings	



1.0 Introduction

This unit considers software inspections. It delves into the inspection pre-conditions and inspection checklists.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:
explain the aim of software inspections
identify inspection pre-conditions
enumerate inspection checklists.



3.0 Main Content

3.1 Software Inspection

Software inspection involves people examining the source representation with the aim of discovering anomalies and defects. It does not require execution of a system; and so, may be used before implementation.

Inspection may be applied to requirements, design, test data and etc. It is a very effective technique for discovering errors.

3.2 Inspection and Testing

Inspection and testing are complementary. Both should be used during the verification and validation process.

Inspections will check conformance with a specification and not conformance with the customer's real requirements. It cannot check non-functional characteristics such as performance, usability etc.

3.3 Inspection Pre-conditions

A precise specification must be available

Team members must be familiar with the organisation's standards

Syntactically correct code must be available

An error checklist should be prepared

Management must accept that inspection will increase costs early in the software process

Management must not use inspections for staff appraisal.

3.4 Inspection Procedure

System overview presented to inspection team

Code and associated documents are distributed to inspection team in advance

Inspection takes place and discovered errors are noted

Modifications are made to repair discovered errors

Re-inspection may or may not be required.

3.5 Inspection Checklists

Checklist of common errors should be used to drive the inspection

Error checklist is programming language dependent

The 'weaker' the type of checking, the larger the checklist

Examples: Initialisation, Constant naming, loop termination, array bounds, etc.

Discussion

What type of software error does inspection unravel?

Scenario

Why is error checklist programming language dependent?



4.0 Self-Assessment Exercise(s)

1. which of these statements is correct about inspection?

- a. Software inspection does not require execution of a system
- b. Software inspection requires execution of a system
- c. Software inspection requires high-level execution of a system

d. Software inspection requires low-level execution of a system

2. _____ involve people examining the source representation with the aim of discovering anomalies and defects?

- a. Software Inspection
- b. Software analysis
- c. Software review
- d. software examination



5.0 Conclusion

In this unit you have learned about software inspection. You have also been able to identify inspection pre-conditions and checklists.



6.0 Summary

What you have learned borders on software inspection.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component?* *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssi re, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A, (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Sher, A. K., and Nazir, A. Z. (2007). Promotion of Local to Global Operation in Train Control System. *Journal of Digital Information Management*

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freotechbooks.com/software-engineering-methodology-thewatersluice-t573.html#754>.

Unit 3: Software Reliability

Contents

1.0 Introduction	
2.0 Intended Learning Outcome (ILOs)	
3.0 Main Content	
3.1 Software Dependability	
3.2 Fault Minimisation	
3.3 Fault-Free Software Development	
3.4 Reliable Software Processes	
4.0 Self-Assessment Exercise	
5.0 Conclusion	
6.0 Summary	7.0
Further Readings	



1.0 Introduction

This unit covers software dependability and fault minimisation.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

explain the notion of software dependability
describe fault minimisation
discuss the reliable software processes.



3.0 Main Content

3.1 Software Dependability

In general, software customers expect all software to be dependable. However, for non-critical applications, they may be willing to accept some system failures. Some applications, however, have very high dependability requirements and special programming techniques must be used to achieve this.

3.2 Fault Minimisation

Current methods of software engineering now allow for the production of fault-free software. Fault-free software means software which conforms to its specification. It does NOT mean software which will always perform correctly as there may be specification errors. The cost

of producing fault-free software is very high. It is only cost-effective in exceptional situations.

3.3 Fault-Free Software Development

- i. Fault-free software development needs a precise (preferably formal) specification.
- ii. It requires an organisational commitment to quality.
- iii. Information hiding and encapsulation in software design is essential
- iv. A programming language with strict typing and run-time checking should be used
- v. Error-prone constructs should be avoided
- vi. Dependable and repeatable development process

3.4 Reliable Software Processes

To ensure a minimal number of software faults, it is important to have a well-defined, repeatable software process. A well-defined repeatable process is one that does not depend entirely on individual skills; rather can be enacted by different people. For fault minimisation, it is clear that the process should include significant verification and validation.

Discussion

What do you understand by dependability in software products?

Scenario

Why would a fault free software still not be correct?



4.0 Self-Assessment Exercise(s)

1. which of these statements is correct about inspection?

- a. all software applications have very high dependability requirements
- b. all software applications have low dependability requirements
- c. non-critical applications have very high dependability requirements
- d. critical applications have very high dependability requirements

2. _____ are software which conforms to its specification?

- a. fault-free software
- b. good software

c. highly specified

d. valid software



5.0 Conclusion

In this unit you have learned about software dependability. You have also learned about fault minimisation and fault-free software development.



6.0 Summary

What you have learned in this unit borders on software reliability and fault-free software development.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component? Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “A Security Framework for Reflective Java Applications,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A., (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Sher, A. K., and Nazir, A. Z. (2007). Promotion of Local to Global Operation in Train Control System. *Journal of Digital Information Management*

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freotechbooks.com/software-engineering-methodology-thewatersluice-t573.html#754>.

Unit 4: Software Re-use

Contents

1.0 Introduction

2.0 Intended Learning Outcome (ILOs)

3.0 Main Content

3.1 Software Re-use

3.2 Benefits of Re-use

3.3 Software Development with Re-use

3.4 Requirements for Re-use

3.5 Software Development for Re-use

4.0 Self-Assessment Exercise

5.0 Conclusion

6.0 Summary

Further Readings

7.0



1.0 Introduction

This unit considers the concept of software re-use. You will equally learn about the benefits and the requirements for re-use.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

describe the concept of software re-use

enumerate the benefits of re-use

explain software development with re-use

list the requirements for re-use.



3.0 Main Content

3.1 Software Re-use

In most engineering disciplines, systems are designed by coupling existing components that have been used in other systems. Software engineering has been more focused on original development but it is now recognised that to achieve better software, more quickly and at lower cost, we need to adopt a design process that is based on systematic re-use.

3.2 Benefits of Re-use

- i. Increased reliability:** components exercised in working systems
- ii. Reduced process risk:** less uncertainty in development costs
- iii. Effective use of specialists:** re-use components instead of people
- iv. Standards compliance:** embed standards in reusable components
- v. Accelerated development:** avoid original development and hence speed-up production

3.3 Software Development with Re-use

Software development with re-use attempts to maximise the use of existing components. These components may have to be adapted in a new application.

Fewer components need be specified, designed and coded. Overall development costs should therefore be reduced.

3.4 Requirements for Re-use

- i. It must be possible to find appropriate reusable components in a component data base.
- ii. Component re-users must be able to understand components and must have confidence that they will meet their needs.
- iii. The components must have associated documentation discussing how they can be re-used and the potential costs of re-use.

3.5 Software Development for Re-use

Software components are not automatically reusable. They must be modified to make them usable across a range of applications. Software development for re-use is a development process which takes existing components and aims to generalize and document them for re-use.

Discussion

Describe the concept of software re-use

Scenario

Why is software re-use gaining popularity in the software engineering field?



4.0 Self-Assessment Exercise(s)

1. which of these is not a benefit of re-use?

- a. Increased reliability
- b. Accelerated development

- c. Saves the developers the need for documentation
- d. Reduced process risk

2. which of these statements is true?

- a. Software components are not automatically reusable
- b. Software components are automatically reusable
- c. all software are reusable
- d. good software are reusable



5.0 Conclusion

In this unit you have learned about software re-use. You have also learned about the benefits of re-use and the requirements for re-use.



6.0 Summary

What you have learned in this unit borders on software re-use, its benefits and requirements for re-use.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component?* *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A., (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Sher, A. K., and Nazir, A. Z. (2007). Promotion of Local to Global Operation in Train Control System. *Journal of Digital Information Management*

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.fretechbooks.com/software-engineering-methodology-thewatersluice-t573.html#754>.

Unit 5: Software Evolution and Maintenance

Contents

1.0 Introduction

2.0 Intended Learning Outcome (ILOs)

3.0 Main Content

3.1 Software Evolution

3.2 Software Evolution Activities

3.3 Software Maintenance

3.4 Types of Software Maintenance

4.0 Self-Assessment Exercise

5.0 Conclusion

6.0 Summary

7.0

Further Readings



1.0 Introduction

This unit discussed the concept of software evolution and maintenance.



2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

describe the concept of software evolution and maintenance

list some types of software maintenance

list and explain the software evolution activities.



3.0 Main Content

3.1 Software Evolution

Large software products, especially products developed for large business use usually have a long lifetime of many years. Software must evolve in order to remain useful. Business changes and user requirements changes lead to new system requirements. Also changing technology and technological requirements lead to new system requirements as well. Software must be able to accommodate such changes during its lifetime. There must be new releases of the software to take care of the new user and business requirements.

Software evolution begins with the identification of a new requirement to be implemented.

3.2 Software Evolution Activities

The activities involved in software evolution include:

1. **change analysis:** analysis of the proposed change
2. **release planning:** planning on how to release the update
3. **system implementation:** the upgrade or change is implemented on the current version of the software
4. **releasing a system to customers:** the upgraded or changed software is now released to the customers.

3.3 Software Maintenance

Software maintenance is basically the process of changing a system after it has been delivered. The term is usually applied to custom software, with separate development groups before and after delivery. During software maintenance, changes may be made to a system to correct coding error, design error, and specification error. Changes are also implemented by adding new components to the system or modifying existing system components.

3.4 Types of Software Maintenance

1. Environmental adaptation to adapt the software to new platforms and environments
2. Fault repairs to fix bugs and vulnerabilities
3. Functionality addition to add new features and to support new requirements.

Discussion

Describe the concept of software evolution

Scenario

Why does it cost more to maintain a software than it costs to develop it?



4.0 Self-Assessment Exercise(s)

1. which of these is not a type of software maintenance?
 - a. Environmental adaptation to adapt the software to new platforms and environments
 - b. Fault repairs to fix bugs and vulnerabilities
 - c. Functionality addition to add new features and to support new requirements.
 - d. hardware maintenance

2. which of these is the first step in software evolution process?

- a. system implementation
- b. release planning
- c. change analysis
- d. releasing a system to customers



5.0 Conclusion

In this unit you have learned about software evolution and maintenance. You have also learned about the types of software maintenance



6.0 Summary

Software evolution and maintenance are important parts of the software lifecycle. It involves making sure a software system continues to deliver the desired expectations to the system users and customers.



7.0 Further Readings

Brooks, F. P., (1995). *The Mythical Man-Month: Essays on Software Engineering* (20th Anniversary Edition). Addison-Wesley Inc.: Reading, MA.

Broy, M., Deimel, A., Henn, J., Koskimies, K., Plášil, F., Pomberger, G., Pree, W., Stal, M. and Szyperski, C., (1998). *What Characterizes a (Software) component?* *Software – Concepts & Tools*, vol. 19, pp. 49-56.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc.: New York.

Calvert, K. L. and Donahoo, M. J., (2002). *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Publishers: San Francisco, CA.

Caromel, D. and Vayssière, J., (July 2003). “*A Security Framework for Reflective Java Applications*,” *Software – Practice & Experience*. John Wiley & Sons, Inc., vol. 33, no. 9, 821-846.

Chellappa, R., Phillips, P. J. and Reynolds, D., (Editors), (November 2006). *Special Issue on Biometrics: Algorithms and Applications of Fingerprint, Iris, Face, Gait, and Multimodal Recognition. Proceedings of the IEEE*, vol. 94, no. 11.

Chen, G., and Szymanski, B. K., (December 2001). “*Object-oriented Paradigm: Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability*,”. *Proceedings of the 2001 Winter Simulation Conference (WSC 2001)*, pp. 495-501, Arlington, VA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., and Jeremaes, P., (1994). *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Constantine, L. L. and Lockwood, L. A., (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*, Addison-Wesley Professional/ACM Press: Reading, MA.

Grogono, P. (1999). *Software Engineering* (2nd Edition). New Jersey: Prentice Hall.

Sher, A. K., and Nazir, A. Z. (2007). Promotion of Local to Global Operation in Train Control System. *Journal of Digital Information Management*

Somerville, I. (2016). *Software Engineering*, (10th Edition). Pearson Education Limited, Edinburgh Gate Harlow Essex CM20 2JE England

Somerville, I. (2002). *Software Engineering Methodology* (5th Edition). McGraw-Hill

Kozlov, D., Koskinen, J. Sakkinen, M. and Markkula, J. (2008). Assessing Maintainability Change over Multiple Software Releases. *Journal of Software Maintenance and Evolution* 20 (1) pp 31–58.

Online Resources

<http://www.cs.nmsu.edu/~jeffery/courses/371/lecture.html>

<http://mail.svce.ac.in/~uvarajan/cn.html>

<http://www.freetechbooks.com/software-engineering-methodology-thewatersluice-t573.html#754>.

