African Virtual University

Applied Computer Science: CSI 5102

SOFTWARE QUALTY ASSURANCE

Dr. Godfrey Justo

Foreword

The African Virtual University (AVU) is proud to participate in increasing access to education in African countries through the production of quality learning materials. We are also proud to contribute to global knowledge as our Open Educational Resources are mostly accessed from outside the African continent.

This module was developed as part of a diploma and degree program in Applied Computer Science, in collaboration with 18 African partner institutions from 16 countries. A total of 156 modules were developed or translated to ensure availability in English, French and Portuguese. These modules have also been made available as open education resources (OER) on oer.avu.org.

On behalf of the African Virtual University and our patron, our partner institutions, the African Development Bank, I invite you to use this module in your institution, for your own education, to share it as widely as possible and to participate actively in the AVU communities of practice of your interest. We are committed to be on the frontline of developing and sharing Open Educational Resources.

The African Virtual University (AVU) is a Pan African Intergovernmental Organization established by charter with the mandate of significantly increasing access to quality higher education and training through the innovative use of information communication technologies. A Charter, establishing the AVU as an Intergovernmental Organization, has been signed so far by nineteen (19) African Governments - Kenya, Senegal, Mauritania, Mali, Cote d'Ivoire, Tanzania, Mozambique, Democratic Republic of Congo, Benin, Ghana, Republic of Guinea, Burkina Faso, Niger, South Sudan, Sudan, The Gambia, Guinea-Bissau, Ethiopia and Cape Verde.

The following institutions participated in the Applied Computer Science Program: (1) Université d'Abomey Calavi in Benin; (2) Université de Ougagadougou in Burkina Faso; (3) Université Lumière de Bujumbura in Burundi; (4) Université de Douala in Cameroon; (5) Université de Nouakchott in Mauritania; (6) Université Gaston Berger in Senegal; (7) Université des Sciences, des Techniques et Technologies de Bamako in Mali (8) Ghana Institute of Management and Public Administration; (9) Kwame Nkrumah University of Science and Technology in Ghana; (10) Kenyatta University in Kenya; (11) Egerton University in Kenya; (12) Addis Ababa University in Ethiopia (13) University of Rwanda; (14) University of Dar es Salaam in Tanzania; (15) Universite Abdou Moumouni de Niamey in Niger; (16) Université Cheikh Anta Diop in Senegal; (17) Universidade Pedagógica in Mozambique; and (18) The University of the Gambia in The Gambia.

Bakary Diallo

The Rector

African Virtual University

Production Credits

Author

Dr. Godfrey Justo

Peer Reviewer

Victor Odumuyiwa

AVU - Academic Coordination

Dr. Marilena Cabral

Overall Coordinator Applied Computer Science Program

Prof Tim Mwololo Waema

Module Coordinator

Jules Degila

Instructional Designers

Elizabeth Mbasu

Diana Tuel

Benta Ochola

Media Team

Sidney McGregor

Barry Savala

Edwin Kiprono

Kelvin Muriithi

Victor Oluoch Otieno

Michal Abigael Koyier

Mercy Tabi Ojwang

Josiah Mutsogu

Kefa Murimi

Gerisson Mulongo

Copyright Notice

Supported By

AVU Multinational Project II funded by the African Development Bank.

Table of Contents

Course Overview

Welcome to Software Quality Assurance.

This course presents software quality assurance. Software quality assurance (SQA) is the concern of every software engineer to reduce costs and improve product time-to-market. A Software Quality Assurance Plan is not merely another name for a test plan, though test plans are included in an SQA plan. SQA activities are performed on every software project. Use of metrics is an important part of developing a strategy to improve the quality of both software processes and work products.

Building on preceding exposure on fundamentals of the software process, this course focuses on techniques for ensuring software quality. Here, quality assurance is viewed as an activity that runs through the entire development process: understanding the needs of clients and users; analyzing and documenting requirements; verifying and validating solutions through testing.

Prerequisites

Software Engineering

Object-Oriented Analysis and Design

Principles of programming

Materials

The materials required to complete this course are:

Computers with internet access

Course Goals

Upon completion of this course the learner should be able to

to understand the nature of software defects and the different approaches that lead to the development of quality software

to equip learners with the knowledge as well as skills necessary for producing quality software

to experience the advantages and challenges of using Software Quality Assurance standardsUnits

Unit 0: Pre-Assessment

This unit introduces the course and provide a roadmap on the organization of course materials.

Unit 1:Overview of Software Quality Assurance

This unit presents the notion of Software quality assurance as the activity applied throughout the software process, on aspects of;

whether the software adequately meet its quality factors, the software development been conducted according to pre-established standards and the technical disciplines performed their SQA roles properly. The unit thus highlights the quality assurance elements, the SQA tasks, the SQA goals and the SQA plan, amongst others.

Unit 2: Requirements analysis: Requirements elicitation and user-centered development

This unit presents the requirements process with emphasis to communicating with stakeholders as a philosophy of user-centered development. The unit highlights the process of requirements elicitation and analysis to derive system specification and models. It also discusses the requirements analysis through problem frames.

Unit 3: Verification and validation

This unit explain the verification and validation concepts, and presents the creation process of a test mission with a highlight of the commonly applied test strategies. Further it describes the concept of conformance testing and provide detailed insights on prototyping as a methodology with inherent mechanism for validation of software designs.

Unit 4: Quality management

This unit highlights the intertwine between the development process quality and product quality and advocates the creation of process standards, Monitoring of the development process and reporting the software as important mechanism of process quality management. The unit further discusses the important functions of quality management including quality planning, quality control and software measurement and metrics.

Assessment

Formative assessments, used to check learner progress, are included in each unit.

Summative assessments, such as final tests and assignments, are provided at the end of each module and cover knowledge and skills from the entire module.

Summative assessments are administered at the discretion of the institution offering the course. The suggested assessment plan is as follows:

1	Unit 1: Unit Assessments	50 Points
	Unit 2: Unit Assessments	
2	Unit 3: Unit Assessments	30 Points
3	Unit 4: Unit Assessments	20 Points

Schedule

Unit	Activities	Estimated time
Unit 0 & Unit 1	Independent study, Tutorials, Assignments	20 Hours
Unit 2	Independent study, Tutorials, Assignments	45 Hours
Unit 3	Independent study, Tutorials, Assignments	35 Hours
Unit 4	Independent study, Tutorials, Assignments	20 Hours

Readings and Other Resources

The readings and other resources in this course are:

Unit 0

Required readings and other resources:

Gerard O'Regan, Introduction to Software Quality, Springer, 2014

Optional readings and other resources:

Jeff Tian, Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement IEEE Computer Society, 2005

Unit 1

Required readings and other resources:

Gerard O'Regan, Introduction to Software Quality, Springer, 2014

Optional readings and other resources:

Darrel Ince, An Introduction to Software Quality Assurance and Its Implementation,

McGraw-Hill, 1994

Jeff Tian, Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement , IEEE Computer Society, 2005

Unit 2

Required readings and other resources:

Gerard O'Regan, Introduction to Software Quality, Springer, 2014

https://en.wikipedia.org/wiki/Problem_frames_approach[Accessed 23/02/2016]

Optional readings and other resources:

Alistair Sutcliffe, User-Centred Requirements Engineering, Springer-Verlag London Limited 2002

https://en.wikipedia.org/wiki/Requirements_analysis [Accessed 23/02/2016]

David J. Gilmore, Russel L. Winder and Francoise Detienne, User-Centred Requirements for Software Engineering Environments, Springer-Verlag Berlin Heidelberg GmbH, 1994

Unit 3

Required readings and other resources:

Gerard O'Regan, Introduction to Software Quality, Springer, 2014

http://www.rbcs-us.com/documents/Defining-Testing-Article.pdf [accessed 24/02/2016]

http://www.sqa.org.uk/e-learning/SDPL03CD/page_16.htm[accessed 24/02/206]

http://searchsoftwarequality.techtarget.com/definition/conformance-testing [accessed 24/02/2016]

Optional readings and other resources:

Paul Ammann and Jeff Offutt, Introduction to Software Testing, Cambridge University Press, 2008

William E. Perry, Effective Methods for Software Testing, Third Edition, Wiley Publishing,

Inc., Indianapolis, Indiana, 2006

http://www.softwaretestinghelp.com/what-is-conformance-testing/[accessed 24/02/2016]

Unit 4

Required readings and other resources:

Gerard O'Regan, Introduction to Software Quality, Springer, 2014

https://www.smashingmagazine.com/2010/06/design-better-faster-with-rapid-prototyping/ [accessed 24/02/2016]

Optional readings and other resources:

Jeff Tian, Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement IEEE Computer Society, 2005

Darrel Ince, An Introduction to Software Quality Assurance and Its Implementation,

McGraw-Hill, 1994

Unit 0. Pre-Assessment

Unit Introduction

The purpose of this unit is to determine your grasp of knowledge related to this course .

Unit Objectives

Unit Readings and Other Resources

The readings in this unit are to be found at the course-level section "Readings and Other Resources".

Unit 1: Overview of Software Quality Assurance

Unit Introduction

The mission of a software company is to develop high-quality innovative products and services at a competitive price to its customers, and to do so ahead of its competitors. This requires a clear vision of the business, a culture of innovation, an emphasis on quality, detailed knowledge of the business domain, and a sound product development strategy.

It also requires a focus on customer satisfaction and software quality to ensure that the desired quality is built into the software product, and that customers remain loyal to the company. Customers today have very high expectations on quality, and expect high-quality software products to be consistently delivered on time. The focus on quality requires that the organization define a sound software development infrastructure to enable quality software to be consistently produced.

Unit Objectives

Upon completion of this unit you should be able to:

Describe the quality assurance elements, SQA tasks, and SQA goals

Understand the quality factors and quality models as basis for quality measurements

Underscore the role of software reliability

KEY TERMS

SQA: Software quality assurance

Project manager: In charge for management of project activities

System Engineer: Expert in system development activities

SQA team: Group of individuals tasked with the SQA tasks

Learning Activities

Quality Assurance, Quality Elements and SQA Tasks and Goals

Introduction

Quality can not be added to a piece of software after you have built it, it must be built into the software from the beginning. Software Quality Assurance is an umbrella activity applied throughout the software process which encompasses; planned and systematic pattern of actions required to ensure high quality in software and responsibility of many stakeholders (software engineers, project managers, customers, salespeople, SQA team).

Quality is a universal concern in software engineering. —Achieving quality encompasses ensuring quality in software engineering processes, methods, and tools; which turn out to be one of the key aims of project management.

Quality products can assure increase market shares, provide long term profitability for companies; while poor quality can certainly have the opposite effect. —Yet, in spite of the focus on quality it can be remarkably elusive to attain in actual software engineering projects.

Activity Details

A reason for the concern with quality is the sheer complexity of many modern software systems.

For example, there is approximately 10 Mbytes of object code in an A320 airbus flight control system. —Complexity in software systems arises in a number of ways: The large number of modules, concurrent processes, and diversity of platforms that must interact to achieve the aims and requirements of the system; and the development of the product itself which is subject to changes in requirements, changes in personal and the ever present constantly changing technology used in implementing and engineering products.

From an end-user's perspective software quality is judged based their interaction with it. For users a system has quality if it is fit for purpose, reliable and has reasonable performance, easy to learn and use. Sometime, if the functionality is hard to learn but is extremely important, then users will still judge the system to have high quality. —The Developer's Perspective, software quality is typically perceived by the number of faults; ease of changing the system, ease of testing the system, the nature of the design, conformance to requirements, resource usage and performance. The maintainer's perspective is similar to developer's perspective; but also includes the simplicity and modularity of the system, the documentation produced by the developers; and the ease of understanding the implementation.

Software engineers need to choose processes, tools and techniques to monitor and control the quality of the software as it is being developed. —If we can not directly monitor the attributes, we often need to monitor the quality of the processes, under the assumption that the quality of the process influences the quality of the product.

The key SQA questions that any software engineers need to constantly ask are:

Does the software adequately meet its quality factors?

Has software development been conducted according to pre-established standards?

Have technical disciplines performed their SQA roles properly?

The key quality assurance elements that system engineers need to monitor are summarised in Table 1.

Table 1: Quality Elements

Element	Description
Standards	Ensure that standards are adopted and followed
Reviews and audits	Audits are reviews performed by SQA personnel to ensure that quality guidelines are followed for all software engineering work
Testing	Ensure that testing is properly planned and conducted
Error/defect collection and analysis	Collects and analyses error and defect data to better understand how errors are introduced and can be eliminated
Changes management	Ensures that adequate change management practices have been instituted
Education	Takes lead in software process improvement and educational program
Vendor management	Suggests specific quality practices vendor should follow and incorporates quality mandates in vendor contracts
Security management	Ensures use of appropriate process and technology to achieve desired security level
Safety	Responsible for assessing impact of software failure and initiating steps to reduce risk

Risk management	Ensures risk management activities are properly conducted and that contingency plans have been established

Software quality assurance (SQA) encompasses a quality management approach, an effective software engineering technology (methods and tools), a formal technical reviews that are applied throughout the software process, a multi-tiered testing strategy, a control of software documentation and the changes made to it, a procedure to ensure compliance with software development standards (when applicable), a measurement and reporting mechanisms. Typically in an project the project manager need to a set up an SQA team whose tasks include:

- Prepare SQA plan for the project.
- Participate in the development of the project's software process description.
- Review software engineering activities to verify compliance with the defined software process.
- Audit designated software work products to verify compliance with those defined as part of the software process.
- Ensure that any deviations in software or work products are documented and handled according to a documented procedure.
- Record any evidence of noncompliance and reports them to management.

The SQA team would operate under the guide of the SQA goals as listed in table 2.

Table 2: SQA Goals

Requirements Quality	Design Quality	Code Quality	Quality Control Effectiveness
Ambiguity	Architectural integrity	Complexity	Resource allocation
Completeness	Component completeness	Maintainability	Completion rate
Volatility	Interface complexity	Understandability	Review effectiveness
Traceability	Patterns	Reusability	Testing effectiveness
Model clarity		Documentation	

Conclusion

This activity described the quality assurance concepts and how the development team and stakeholders work together to assure product quality. The SQA elements, tasks and goal were also identified.

Assessment

i. Explain the concept of software quality with respect to different stakeholders

ii. Identify the role of the different stakeholders in ensuring software quality

iii. Describe the SQA team with respect to the tasks and guiding goals.

Quality factors and Quality models

Introduction

Software quality refers to conformance to explicitly stated requirements and standards. —Quality assurance is the activity that leads to "fitness of purpose". —Quality product is the one that does what the customer expects it to do.

User satisfaction = compliant product + good quality + delivery within budget and schedule.

Quality control is a series of inspections, reviews, and tests to ensure a product meets the requirements placed upon it, including a feedback loop to the process that created the work product. Quality control activities may be fully automated, entirely manual, or a combination of automated tools and human interaction. Quality assurance refers to the analysis, auditing and reporting activities. It provide management with the data necessary to be informed about product quality

Activity Details

Total Quality Management (TQM) is a popular approach for management practice.

TQM stresses continuous process improvement and can be applied to software development. TQM is a holistic business management methodology that aligns the activities of all employees in an organization with the common focus of customer satisfaction through continuous improvement of all activities, goods and services Not much can be done to improve quality until a visible, repeatable, and measurable process is created. The Quality Movement refers to a system of continuous process improvement, revolving around four main steps: develop a process that is visible, repeatable, and measurable.

examines intangibles that affect the process and works to optimize their impact on the process.

Concentrates on the user of the product in which examine the way the user applies the product. This step leads to improvement in the product itself and, potentially, to the process that created it.

A business-oriented step that looks for opportunity in related areas identified by observing the use of the product in the marketplace.

The SQA team must look at software from the customer's perspective, as well as assessing its technical merits. —The activities performed by the SQA team involve quality planning, oversight, record keeping, analysis and reporting, as depicted in Figure 1.

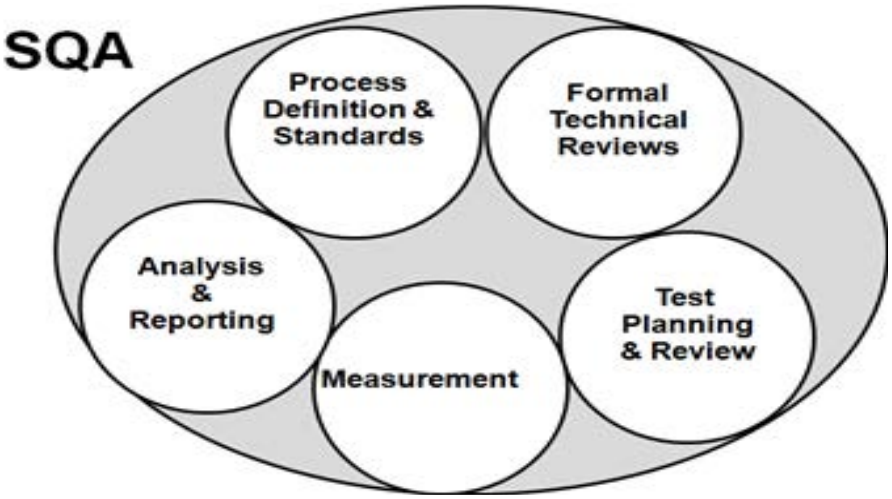

Figure 1: SQA Team Activities

At the beginning of the project, project manager will consider quality factors and decide which ones are important for the system. The fundamental quality factors entail the Functionality, Usability, Reliability, Performance, and Supportability requirements of the software as described by Table 3. Subsequently, the project manager will decide on what validation and verification activities will be carried out to check that the required quality factors are present in the product. —During the project validation and verification of quality standards and procedures are conducted and at the end of the project the expected quality achieved and to what extent can be determined.

Table 3: Fundamental Quality Factor

Factor	Description
Functionality	Refers to feature set, capabilities, generality of functions, and security
Usability	Refers to human factors like consistency, and documentation

Reliability	Frequency and severity of failures, accuracy of outputs, mean time between failures, ability to recover, predictability
Performance	Processing speed, response time, resource consumption, throughput, and efficiency
Supportability	Extensibility, adaptability, maintainability, testability, compatibility, configurability

Quality can not be measured or assessed directly, except perhaps by having someone inspect artifacts like designs and software modules for quality. It relies heavily on an individual's understanding and interpretation of quality. can make use of 'quality models' that decomposes the concept of quality Into number of attributes; each of which can be measured more objectively.

Under quality models, for a system to have quality, we usually require that it:

Satisfy explicit functional and non-functional requirements, that is, be correct, complete and consistent with respect to the requirements.

Adhere to internal (organizational or project) and external standards imposed on the project. For example, IEC 61508 (safety), Rainbow standard (US security), IEEE standards, and internal company standards.

Conform to implicit quality requirements; which are requirements for performance, reliability, usability, safety, security (Typically, what we think of as the attributes of quality)

Table 4 and Table 5 presents McCall's quality model and ISO-9126 Quality Models respectively.

Table 4: McCall's quality model and its interpretation of external system attributes

Quality Attribute	Description according to McCall et.al
Correctness	The extent to which a program satisfies it specifications and fulfils the user's mission objectives
Reliability	The extent to which a program can be expected to perform its intended function with required precision
Efficiency	The amount of computing resources and code required by a program to perform a given function

Integrity	The extent to which access to software or data by unauthorised persons can be controlled
Usability	The effort required to learn, operate, prepare input, and interpret output of a program
Maintainability	The effort required to locate and fix an error in an operational program
Testability	The effort required to test a program to ensure that it performs its intended function
Flexibility	The effort required to modify an operational program
Portability	The effort required to transfer a program from hardware and/or software environment to another
Reusability	The extent to which a program (or parts thereof) can be reused in other applications.
Interoperability	The effort required to couple one system with another

Table 5: ISO-9126 Quality Models

Characteristics	Sub-Characteristics
Functionality	Suitability
	Interoperability
	Security
Reliability	Maturity
	Fault Tolerance
	Recoverability

Usability	Understandability
	Operability
	Learnability
	Attractiveness
Efficiency	Time Behaviour
	Resource Utilization
Maintainability	Analysability
	Changeability
	Stability
	Testability
Portability	Adaptability
	Installability
	Co-existance
	Replaceability

Assessing attributes in quality standards using McCall's or ISO9126 is not so easy in practice:

As some interpretation of whether or not an attribute meets its targets is still required.

In some cases, such as usability, it is difficult to even specify the targets.

It is always necessary to consider the attributes of quality for any system. Software quality assurance methods can be chosen to assure that you are building the right quality targets into your system.

The notion of quality is now being comprehended, but the assurance counterpart can be just as difficult. It is difficult, if not impossible, to guarantee absolutely that all of the requirements, including the quality requirements are satisfied. We aim to provide a high level of assurance, that program will meet the needs for which it was written.

The problem is that programs and systems can be highly complex. A typical program of moderate size (30K lines of code) can exhibit an exponentially large number of interconnections, large numbers of paths etc. For the level of assurance expected to be achieved in projects, in practice we need to:

1. understand the quality attributes that we are interested in for our project
2. have methods to build these attributes into our systems

Conclusion

In this activity the concept of quality was further describe along with the related notions of total quality management and quality movement. This also lead to a description of quality factor and quality model which provide the basis for quality control.

Introduction to Software Reliability

Introduction

Software testing aimed at finding the faults in programs. —Software is part of a much larger system consisting of hardware and communications infrastructure as well as programs. —One is required to build a system that is far more reliable than any one of its component. —For example:

How reliable /available must a web server be to handle the number of transactions the site receives?

In process control industries, such as food and beverage industry or chemical plants automation is playing a larger role thus robots and computer systems need to be operating for long length of time without maintenance.

Activity Details

Software Reliability can be used in number of ways:

Evaluation of new system: To evaluate systems by testing the system and gathering reliability measures. The measures can be useful in deciding what third party software, such as libraries, to use in a project.

Evaluation of development status: Comparing current failure rates with desired failure rates. Large discrepancies may indicate the need for action.

Monitor the operational profile: The effect of changes or new features as they are added to the system.

Reliability is one of a number of dependability attributes of a program. —Dependability is a measure of a system's availability, reliability, and its maintenance support. —Dependability is analyzed in terms of attributes of dependability. Unlike quality attributes, dependability is a collective name for a set of attributes as shown in Figure 5.

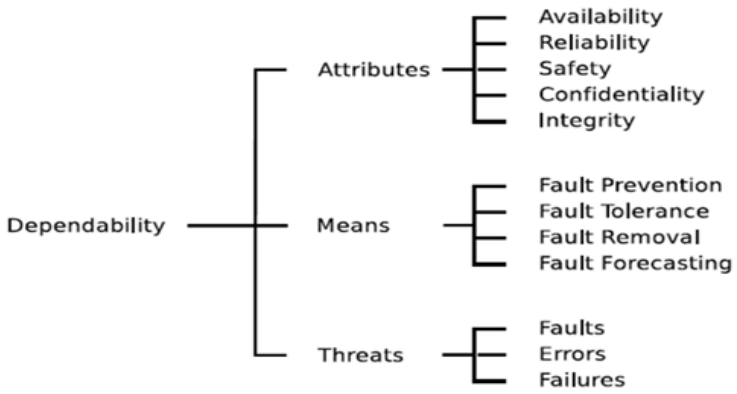

Figure 5: Attributes of Dependability

Reliability is a means for achieving dependability, just as for example, fault tolerance is the means for improving reliability, as seen in Figure 5. Reliability is the probability that a system will operate without failure for a specified time in a specified environment. —Consider two simple systems consisting of two writers and a communicating via a shared memory on a bus, as shown in Figure 6:

Suppose probability of failure of each channel of the bus is 10-5 per operational hour.

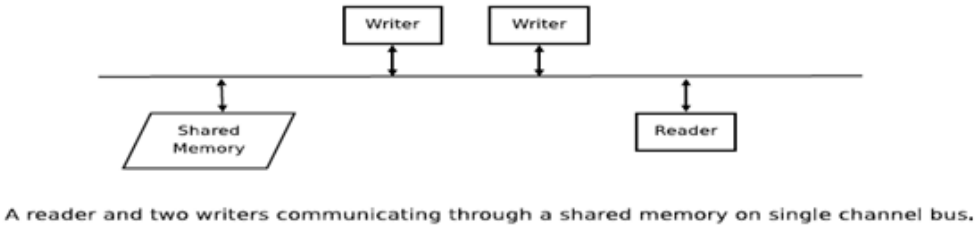

Figure 6: A reader and two writers communicating through a shared memory on single channel bus.

For a single channel bus the probability of a bus failure, such as message becoming corrupt or the bus failing to transmit a message is 'one failure every 11.4 years'. If we add the second channel then the failure probability decreases to one failure every 1,114,552 years, because both channels must fail at the same time for this to occur. —We have to be careful in not over interpreting these numbers: While improvement is significant, after all, it is only a probability. The definition of reliability depends heavily on determining, and in practice detecting, failures.

Examples of failures include:

- Functional failures – where the actual behavior of the system deviates from the specified functional behavior.

- Timing failures – where the program may deliver correct results but the program fails to meet its timing requirements.
- Safety failures – where an accident resulting in harm or injury is deemed to be a failure of the system.
- In practice, we need a way of observing failures and logging each failure as it arises. —We also need a way of determining the time at which a failure occurs – called failure time; Or the rate at which failure occur – called failure rate (with assurance that the estimates are accurate!).

Failures can be classified into:

- Transient Failure - Program gives incorrect result, but the program continues to execute.
- Hard Failure - Program crashes (stack overrun….)
- Cascaded Failure - Program crashes and takes down other programs
- Catastrophic Failure - Program crashes and takes down the operating system or the entire system;

A total system failure.

Software reliability enginering (SRE) encomapesses activities for improving the reliability attribute a a software. —Activities of SRE span the entire software development process as shown by Table 4, but the activities themselves do not constitute a development process. The activities themselves need to be supported by:

- Analytical tools – reliability models
- Evaluation methods – Markov Chains.

Table 4: SRE Activities

Phase	SRE Activities
Requirements	Determine the functional profile of the system
	Determine and classify important failures
	Identify client reliability needs
	Conduct trade-off studies.
	Set reliability objectives.

Design	Design and evaluate systems to meet reliability goals.
	Allocate reliability targets to components
Implementation	Measure and monitor the reliability during implementation.
	Manage fault introduction and propagation.
System and Field Testing	Measure reliability growth (using reliability growth models).
	Track testing progress against reliability growth.
Post-delivery & Maintenance	Monitor field reliability against reliability objectives

Conclusion

This activity presented the reliability quality factor and its relation to software failure. Also the activity highlighted how software engineers can improve reliability by incorporating SRE activities within the software development process.

Assessment

1. Is reliability the most critical quality factor? Discuss.
2. Discuss the key skills and tools that might be needed to perform SRE activities.

UNIT SUMMARY

This unit provided a thorough overview of software quality assurance (SQA). Concepts pertaining to quality assurance were defined including quality Assurance and SQA elements, tasks and Goals respectively.

In turn the quality factors and quality models were presented which provides a head start for quality measurement. Finally, the reliability quality factor was discussed and methods for improving software reliability outlined. The focus on quality requires that the organization define a sound software development infrastructure to enable quality software to be consistently produced.

Unit Assessment

Check your understanding!

Miscellaneous questions

Instructions

1. Discuss the implication of quality to products in general
2. Describe techniques/methods/tools which could be used by Software engineers to help answer the SQA questions:
3. Does the software adequately meet its quality factors?
4. Has software development been conducted according to prestablished standards?
5. Have technical disciplines performed their SQA roles properly?
6. Explain one example of software and its application domain where reliability is:

   i. very important

   ii. not very important

Grading Scheme

As per the offering University

Answers

mailto:njulumi@gmail.com

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources.

Unit 2: Requirements Analysis, Elicitation and User-Centered Development

Unit Introduction

Requirements analysis encompasses tasks for determining the needs or conditions to meet for a new or altered product or project, taking account of the possibly conflicting requirements of the various stakeholders, analyzing, documenting, validating and managing software or system requirements. Requirements analysis is critical to the failure of a systems or software project. The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

Requirements analysis generally includes three types of activities:

Eliciting requirements (also requirements gathering) which entails, for example, inspecting existing business process documentation and stakeholder interviews.

Analyzing requirements which entails determining whether the stated requirements are clear, complete, consistent and unambiguous, and resolving any apparent conflicts.

Recording requirement which entails documenting requirements in various forms, usually including a summary list and may include natural-language documents, use cases, user stories, or process specifications.

For a successful requirement analysis it is important to identify all the stakeholders, take into account all their needs and ensure they understand the implications of the new systems. Software engineers can employ several techniques to elicit the requirements from the customer. These may include the development of scenarios (represented as user stories in agile methods), the identification of use cases, the use of workplace observation, holding interviews, or focus groups (formally through requirements workshops, or requirements review sessions) and creating requirements lists.

Prototyping can also be used to develop an example system that can be demonstrated to stakeholders. Whenever needed, the Software engineers can also employ a combination of these methods to establish the exact requirements of the stakeholders, so that a system that meets the business needs is produced.

Requirements quality can be improved by additionally employing the following and other related methods:

Visualization - employ tools that promote better understanding of the desired end-product such as visualization and simulation.

Consistent use of templates - develop a set of models and templates to document the requirements and apply them consistently.

Documenting dependencies - document dependencies and interrelationships among requirements, as well as any assumptions and congregations.

Unit Objectives

Upon completion of this unit you should be able to:

- develop skills for requirement elicitation with user focus
- develop knowledge on the requirements analysis core activities of elicitation, analysis and negotiation
- describe a new analysis method of problem frames approach

KEY TERMS

1. **Requirement elicitation**: An activity of user requirement gathering
2. **User centered development:** A development process driven by involvement of stakeholders
3. **Requirement Analysis**: An activity that determine requirement necessity, consistency, completeness and feasibility with the business goals
4. **Requirements negotiation**: An activity for determining requirement problems, priority and change from stakeholders perspective.
5. **Problem statement:** An initial developed by client which give concise description of the requirements that should be addressed by the prospective System.
6. **Problem frames approach**: Also called Problem analysis is an approach to software requirements analysis, which encompasses a set of concepts to be used when gathering requirements and creating associated software specifications.

Learning Activities

Requirements Elicitation and User-Centered Development

Introduction

Requirement elicitation encompasses all activities involved in discovering the requirements of a system. System developers and engineers work in close relationship with customer and end-users (stakeholders) to find out more about the problem to be solved, to describe the functionality of the system, performance of the system, hardware constraints and so on.

Activity Details

Requirement elicitation is not just a simple process of gathering for requirements, but a highly complex process due to following inevitable field environments:

Customer rarely have a clear picture of their requirements

Different people have conflicting requirements

Business environment in which the elicitation process takes place is constantly changing, hence may trigger some requirements to change or new requirements resulting from new stakeholders.

Many different elicitation processes does exist which a requirements elicitor has to choose from.

Requirements elicitation is the earliest activity of the software development process and it is an integro process within as depicted in Figure 1. Also note that each successor development phase activity systematically transforms the requirement elicitation outcome of the successor development phase.

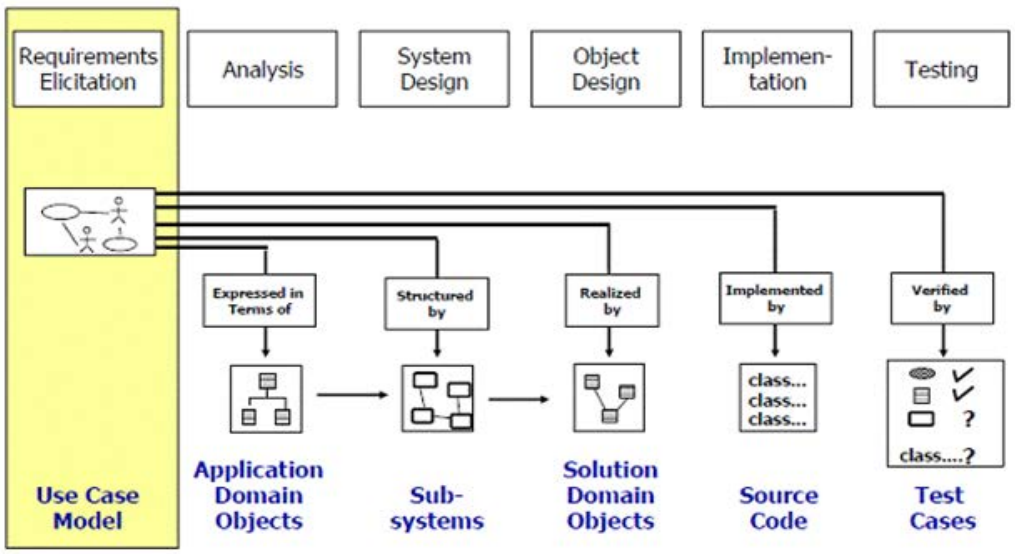

Figure 1: Requirement Elicitation in a Software Development Process

The requirement elicitation process revolves around four basic activities perspectives shown Figure 2; the application domain, problem, business and stakeholder perspective. The application domain perspective entails an understanding of knowledge of the general area where the system is applied. The problem perspective entail an understanding of details of the specific customer problem where the system will be applied. The business perspective entail understanding the how systems interact and contribute to overall business goals. The stakeholders perspective entails understanding the needs and constraints of system stakeholders, that is, aim to come up with detailed specific needs of people who require system support in their work.

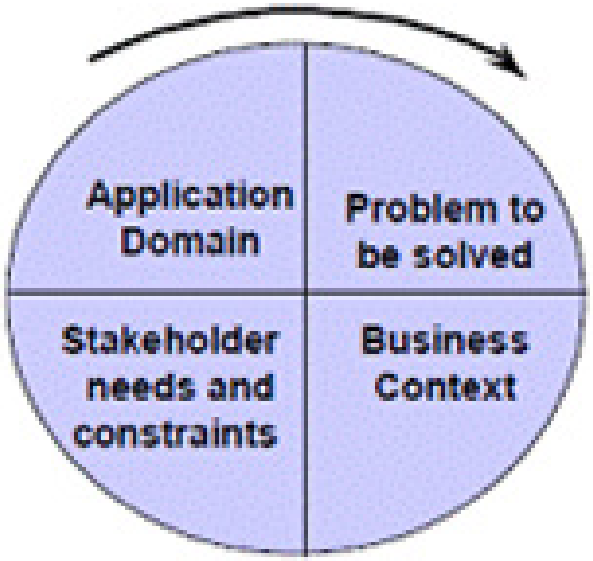

Figure 2: Requirement Elicitation Activities

Viewed from the process perspective, requirements elicitation encompasses four basic elicitation stages as shown in Figure 3 which includes: objective setting, background knowledge acquisition, knowledge organization and stakeholder requirement collection. In the objective setting the concern is to establish the overall organizational objectives, and answer why is the system necessary. The background knowledge acquisition concern with gathering and understanding more background information about the system. Knowledge organization stage involves with organizing, prioritizing and structuring the large amount of data collected in the preceding stages. The stakeholder requirements collection stage aims to involve system stakeholders to discover their requirements.

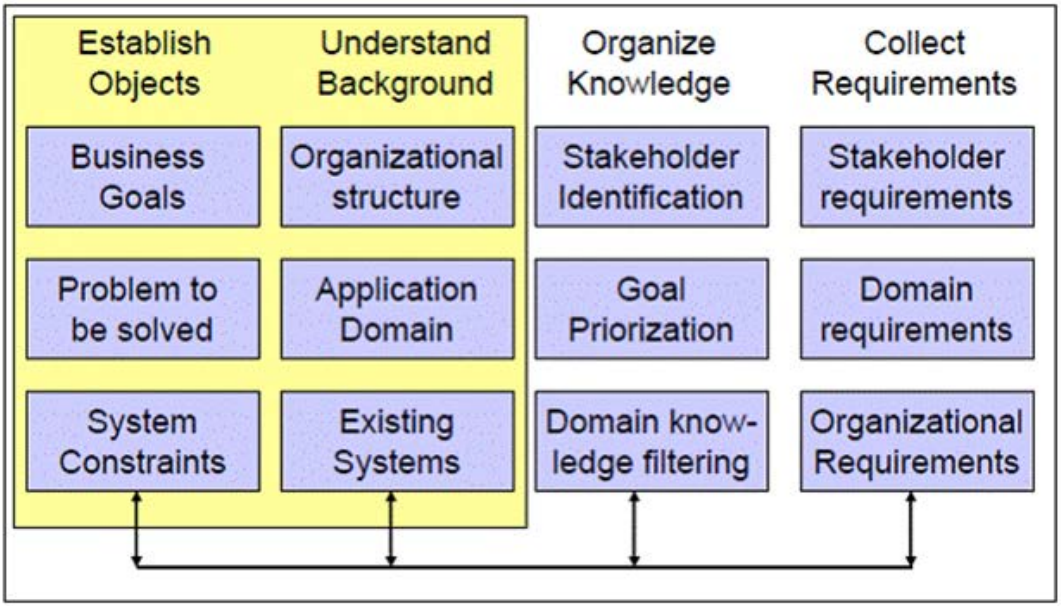

Figure 3: Requirement Elicitation Stages

Conclusion

In this activity we describe the requirement elicitation as an integral part of the software development process. We highlighted its complexity owing to inevitable occurring incidences in the field, it logical underlying activities and the stages involved when requirement elicitation is viewed as a process.

Assessment

Using knowledge acquired from Software engineering course choose a simple example of software application use case model and systematically develop the associated transformations into:

1. Application domain objects
2. Subsystems
3. Solution domain objects
4. Source code skeleton
5. Test cases

Elicitation, Analysis and Negotiation

Introduction

The requirement elicitation and analysis activities are fundamentally intertwined and in some literatures the span of the two activities is considered as analysis phase. In this activity we describe in detail the tasks involved to capture their relationships.

Activity Details

Requirement elicitation and analysis activities are typically intertwined as represented by a spiral in Figure 4, showing yet another view in requirements elicitation.

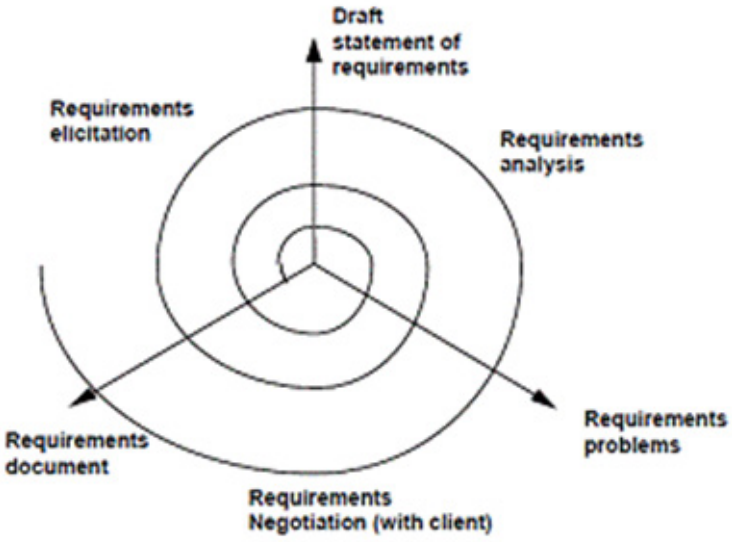

Figure 4: A Requirements Solicitation Spiral

As part of requirements elicitation process Software engineers typically are engaged in checking for the following:

Requirement Analysis sector checks:

Necessity - whether each requirement contribute to the business goals of the organization or to the specific problem to be addressed by the system.

Consistency and completeness - whether requirements are consistency and complete (i.e, no contradictions, no services or constraints are missed out)

Feasibility - the context of the budget and schedule.

Requirement negotiation sector engage with stakeholders for:

Requirements discussion in which requirements highlighted as problematical are discussed and the stakeholders involved present their views about the requirements.

Requirements prioritization in which critical requirements are identified

Requirements agreement in which a compromised set of requirements are agreed and any proposed necessary changes to requirements are affected

Being an integral part of requirement solicitation the the requirement analysis and requirement negotiation activities can be further be described as processes shown in Figure 5. Note how the deliverables from the requirement analysis process feeds into the requirement negotiation process, and the interactions of activities with each process.

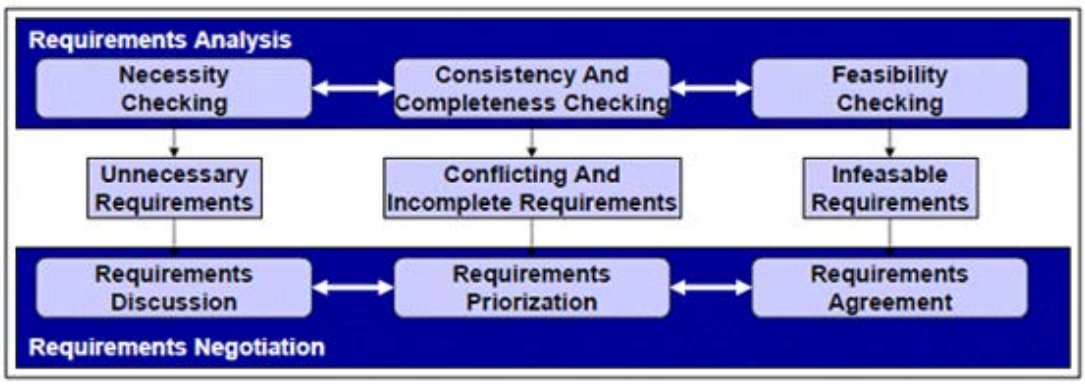

Figure 5: Requirement Analysis and Negotiation Process

The requirement elicitation sector and the requirement analysis sector shown in the spiral also interacts. Figure 6 presents the requirement elicitation and analysis process in which the respective deliverables shown, namely, the requirements specification and analysis model.

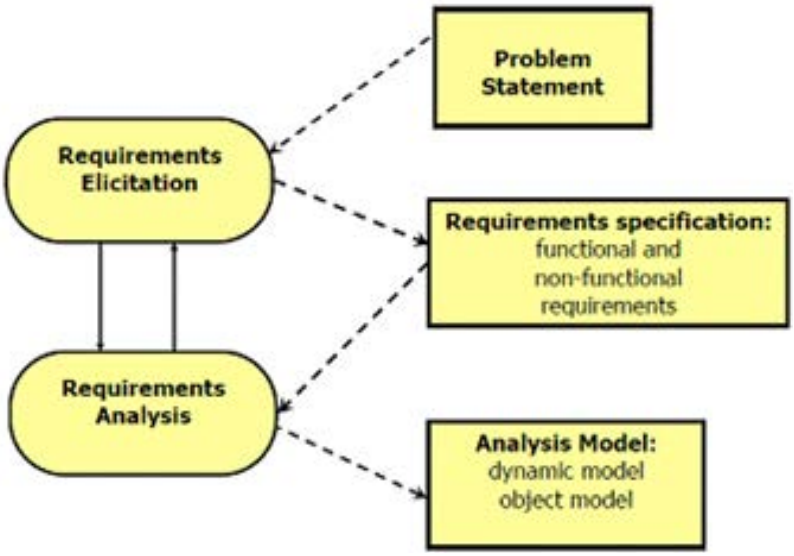

Figure 6: Requirement Elicitation and Analysis Process

The Analysis Model and Requirements Specification have fundamental difference along with some commonality. Both models focus on the requirements from the user's view of the system. However, the requirements specification uses natural language (derived from the problem statement), while the analysis model uses formal (Z, pi-calculus) or semi-formal notation (for example, a graphical language like UML). Note that the formal notations encompass an exact mathematical syntax and semantic.

The starting point for both is the problem statement. The problem statement is usually developed by the client as a concise description of the requirements that should be addressed by the System. The typical elements of the problem statement document includes:

Description of the problem that should be solved along with the current situation - It describes "what" is needed, not "how" it should be reached

Description of one or more scenarios

Brief description of initial requirements (Functional and non-functional requirements)– typically with no complete description.

Description of project schedule (Major milestones that involve interaction with the client including deadline for delivery of the system)

Description of target environment (The environment in which the delivered system has to perform a specified set of system tests)

Description of client Acceptance Criteria (Criteria for the system tests)

The description of the current Situation (Problem To Be Solved) may be stated to indicate for example:

The problem in the current situation - such as the response time in a travel booking system is far too slow or there have been illegal attacks to the system.

A change either in the application domain or in the solution domain has appeared, for example:

Change in the application domain - A new function (business process) is introduced into the business (e.g. A function is provided for credit payment with fingerprint as authorization)

Change in the solution domain - A new solution (technology enabler) has appeared (e.g. New standards / implementation for secure network communication)

Figure 7 shows an example of a problem statement in the Library System domain.

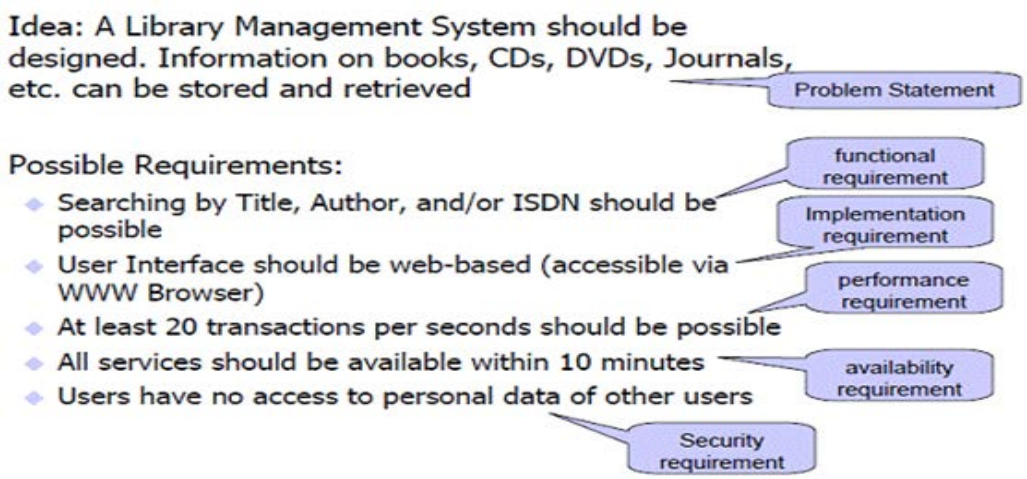

Figure 7: Problem Statement Example

The process of Requirements Elicitation will typically involve identifying and developing some of the following listed artifacts:

- Actors - Types of users, roles, external systems, etc.
- Scenarios - Interactions between users and the systems (one possible case)
- Use Cases - Abstractions of Scenarios (Many possible cases)
- Refining Use Cases - Refinements, adding exceptions, etc.

Relationships among Use Cases

Non-Functional Requirements (Security issues, Performance, etc.)

To achieve the above tasks one need to apply specific techniques which may be used to collect knowledge about system requirements. The information gathering techniques include:

Solicit sources of information - Documents about the application domain including manuals and technical documents of legacy systems.

Cultivating user participation through:

Interviews (Closed Interviews where users answer a predefined set of questions, or open Interviews where there is no predefined agenda)

User Observation

The collected knowledge must be structured via:

- Partitioning - aggregating related knowledge

- Abstraction - recognizing generalities
- Projection – organizing knowledge from several perspectives

Figure 8 depicts a requirements elicitation cycle. Note that requirements elicitation is cooperative process involving requirements engineers and system stakeholders. Also while in the field problems may arise including:

Not enough time for elicitation

Inadequate preparation by engineers

Stakeholders seem not convinced of the need for a new system

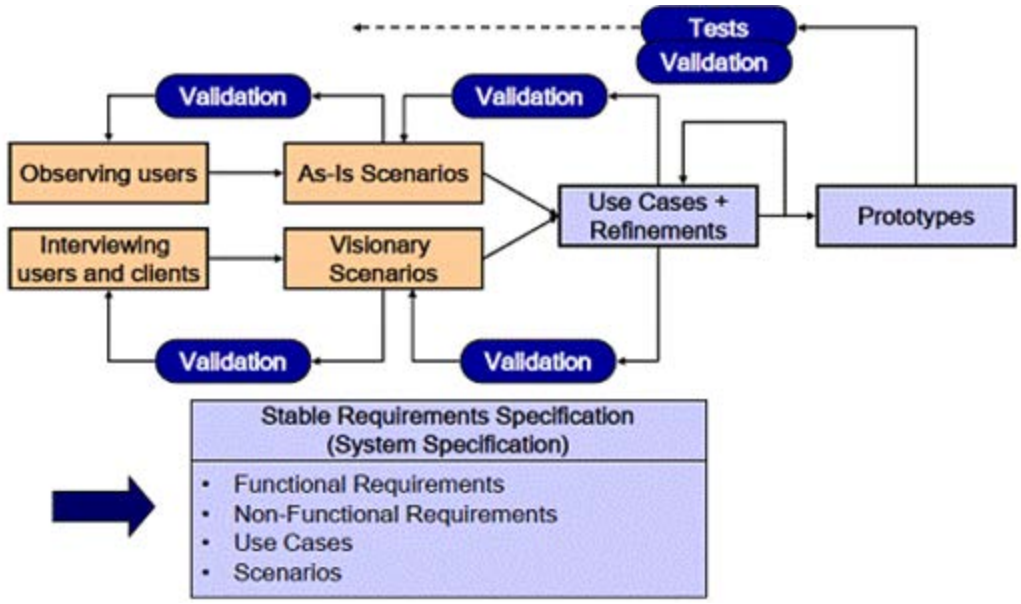

Figure 8: Requirements Elicitation Cycle

Conclusion

This activity presented the requirements elicitation analysis and negotiation process. It highlighted the natural instinct of these activities intertwinement. It discussed the process of developing a problem statement, which feeds into requirement elicitation, requirement analysis and negotiation activities. Also the steps and methods of requirements elicitation were highlighted.

Assessment

1. Using an application domain of your choice create a problem statement document.
2. In the problem statement you have created develop an initial set of requirements outline

Problem Frames Approach

Introduction

Problem analysis or the problem frames approach is an approach to software requirements analysis. It encompasses a set of concepts to be used when gathering requirements and creating associated software specifications. It differs from other software requirements methods owing to its philosophy that:

1. Requirements analysis is approached through a process of parallel (not the traditional hierarchical) decomposition of user requirements.
2. Perceives user requirements from corresponding relationship in the real world (the application domain) not from the software system or even the interface with the software system.
3. The problem frames approach uses sets of conceptual tools. Concepts used for describing specific problems including: phenomena (of various kinds, including events), problem context, problem domain, solution domain (i.e. the machine), shared phenomena (which exist in domain interfaces), domain requirements (which exist in the problem domains) and specifications (which exist at the problem domain:machine interface).
4. The graphical tools for describing problems are the context diagram and the problem diagram.

Activity Details

The problem frame approach provides tools for describing problems. The problem context is described as follows. Consider a software application to be a kind of software machine. A software development project aims to change the problem context by creating a software machine and adding it to the problem context, where it will bring about certain desired effects.

The particular portion of the problem context that is of interest in connection with a particular problem (the particular portion of the problem context that forms the context of the problem) is called the application domain.

After the software development project has been finished, and the software machine has been inserted into the problem context, the problem context will contain both the application domain and the machine. At that point, the situation will look like in Figure 9.

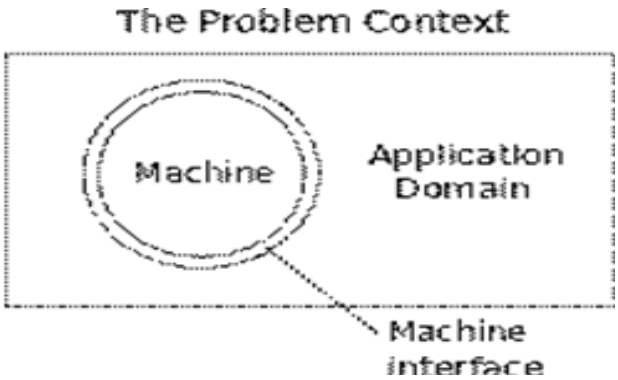

Figure 9: Modeling Problem Context

The problem context contains the machine and the application domain. The machine interface is where the Machine and the application domain meet and interact. The same situation can be shown in a different kind of diagram, called a context diagram, as in Figure 10.

Figure 10: Modeling Problem context as Context Diagram

The context diagram provide a tool for understanding the context in which the problem is set.

A domain is a part of the world that we are interested in. It consists of phenomena (individuals, events, states of affairs, relationships, and behaviors). A domain interface is an area where domains connect and communicate. Domain interfaces are not data flows or messages. An interface is a place where domains partially overlap, so that the phenomena in the interface are shared phenomena (they exist in both of the overlapping domains). In Figure 11, X is the interface between domains A and B. Individuals that exist or events that occur in X, exist or occur in both A and B.

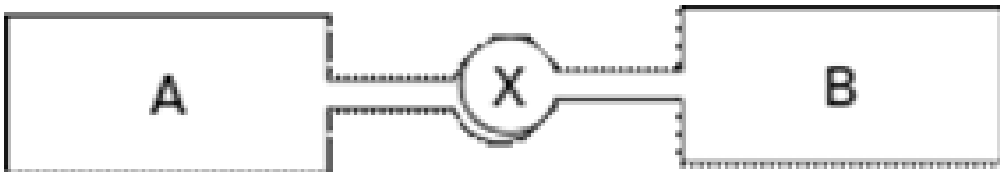

Figure 11: Describing and Interface

Shared individuals, states and events may look differently to the domains that share them. Consider for example an interface between a computer and a keyboard. When the keyboard domain sees an event Keyboard operator presses the spacebar the computer will see the same event as Byte hex("20") appears in the input buffer.

Problem diagrams provide a tool for describing a problem. Figure 12 shows a generic problem diagram.

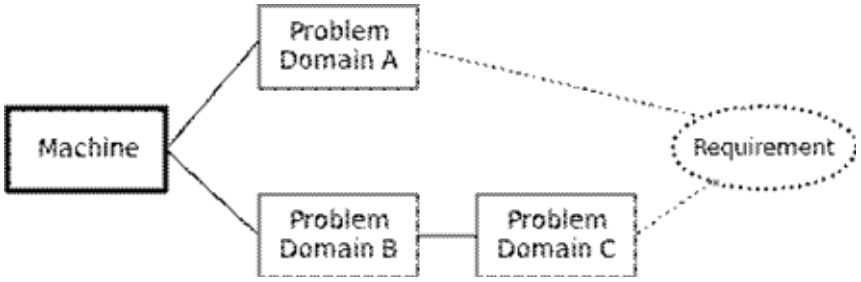

Figure 12: A Problem Diagram

In addition to the aspects shown on a context diagram, a problem diagram also shows:

a dotted oval representing the requirement to bring about certain effects in the problem domains.

dotted lines representing requirement references — references in the requirement to phenomena in the problem domains.

An interface that connects a problem domain to the machine is called a specification interface and the phenomena in the specification interface are called specification phenomena. The goal of the requirements engineer is to develop a specification for the behavior that the Machine must exhibit at the Machine interface in order to satisfy the requirement.

Figure 13 shows an example of a problem diagram which model a hospital system described as follow:.

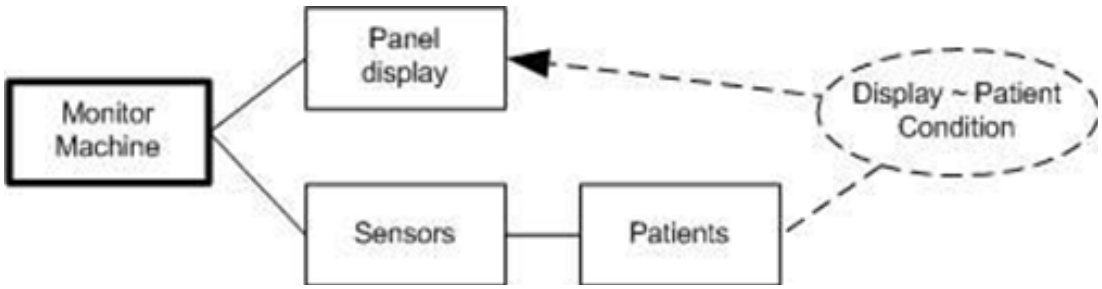

Figure 13: A hospital System Problem Diagram

Consider the following part of a computer system in a hospital. In the hospital, patients are connected to sensors that can detect and measure their temperature and blood pressure. The requirement is to construct a Machine that can display information about patient conditions on a panel in the nurses station.

The name of the requirement is "Display ~ Patient Condition". The tilde (~) indicates that the requirement is about a relationship or correspondence between the panel display and patient conditions.

The arrowhead indicates that the requirement reference connected to the Panel Display domain is also a requirement constraint. That means that the requirement contains some kind of stipulation that the Panel display must meet. In short, the requirement is that The panel display must display information that matches and accurately reports the condition of the patients.

The problem frame approach also provides tools for describing classes of problems (problem frames). A recognized class of problems is called a problem frame (roughly analogous to a design pattern). In a problem frame, domains are given general names and described in terms of their important characteristics.

A domain, for example, may be classified as causal (reacts in a deterministic, predictable way to events) or biddable (can be bid, or asked, to respond to events, but cannot be expected always to react to events in any predictable, deterministic way). (A biddable domain usually consists of people.)

The graphical tool for representing a problem frame is a frame diagram. A frame diagram looks generally like a problem diagram except for a few minor differences: domains have general, rather than specific, names; and rectangles representing domains are annotated to indicate the type (causal or biddable) of the domain.

Conclusion

This activity introduced a problem frames approach to software requirements analysis. It encompasses a set of concepts to be used when gathering requirements and creating associated software specifications. The activity highlight two tools provided by the problem frames approach: tools for describing problems and tools for describing classes of problems

Assessment

1. Compare and contrast the analysis tools proposed in the conventional requirement analysis with those in the problem frames approach.
2. How are the analysis tools in question 1 align with tools used during subsequent phases of software development?

UNIT SUMMARY

This unit presented the requirement analysis process looking on aspects of requirements elicitation. Requirements elicitation, analysis and negotiation activities were described. The unit also described the recently introduced analysis method (largely still being developed) namely the problem frames approach.

Unit Assessment

Check your understanding!

Miscellaneous Questions

Instructions

Using an application problem domain of your choice perform requirement analysis:

1. by conventional analysis methods
2. by problem frames approach
3. Discuss any challenges encountered in the process.

Grading Scheme

As per the offering Institution

Feedback/ Answers

mailto:njulumi@gmail.com

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources.

Unit 3. Verification and Validation

Unit Introduction

Before delving into strategies of testing we must understand the process of verifying and validating the software code. Verification and validation refers to checking processes which ensure that the software conforms to its specification and meets the needs of the customer.

The system should be verified and validated at each stage of the software development process using documents produced in earlier stages. Verification and validation thus starts with requirements reviews and continues through design and code reviews to product testing.

Verification and validation are sometimes confused, but they are different activities.

Validation involves checking that the program as implemented meets the expectations of the customer, that is, whether we building the right product. Verification involves checking that the program conforms to its specification, that is whether we are building the product right. Requirements validation techniques, such as prototyping, help in this respect. However, flaws and deficiencies in the requirements can sometimes be discovered only when the system implementation is complete.

To satisfy the objectives of the verification and validation process, both static and dynamic techniques of system checking and analysis should be used. Static techniques are concerned with the analysis and checking of system representations such as the requirements document, design diagrams and the program source code. Dynamic techniques or tests involve exercising an implementation.

Static techniques include program inspections, analysis and formal verification. Some theorists have suggested these techniques should completely replace dynamic techniques in the verification and validation process and that testing is not necessary, this is not a useful point of view and could be 'considered harmful'. Static techniques can only check the correspondence between a program and its specification (verification). They cannot demonstrate that the software is operationally useful.

Although static verification techniques are becoming more widely used, program testing is still the predominant verification and validation technique. Testing involves exercising the program using data like the real data processed by the program. The existence of program defects or inadequacies is inferred from unexpected system outputs. Testing may be carried out during the implementation phase to verify that the software behaves as intended by its designer. This later testing phase checks conformance with the requirements and assesses the reliability of the system.

Unit Objectives

Upon completion of this unit you should be able to:

- prepare an achievable test mission that engages the stakeholders
- Select appropriate testing strategy and apply conformance testing to chosen product standards
- Use prototyping techniques for validating software designs

KEY TERMS

1. **Validation**: checking that the program as implemented meets the
2. expectations of the customer
3. **verification:** checking that the program conforms to its specification
4. **Test mission:** Defines testing expectations and objectives from stakeholders perspective.
5. **Test strategy:** Defines a general approach to the testing process
6. **Compliance testing:** a methodology used to ensure that a product, process, computer program or system meets a defined set of standards.
7. **Prototyping:** A methodology for representing a design visually

Learning Activities

Defining the Testing Mission

Introduction

The test process is defined to include all necessary activities for these types of testing; planning; analysis, design, and implementation; execution, monitoring, results reporting and closure. We could do all of these activities in a fashion that is technically correct but still fail to deliver any value to the business.

The managers of these test groups may typically fail to identify the stakeholders or, worse yet, in spite of being aware of the stakeholders, fail to work collaboratively with them to determine how to connect the testing process to something valued by each of them.

Activity Details

Stakeholders have a stake in the testing and quality of a system. Can be broadly classify as being either technical stakeholders or business stakeholders. A technical stakeholder is someone who understands and participates in the design and implementation of the system. In the following the technical stakeholders and their role are identified:

Programmers, lead programmers, and development managers:

They are directly involved in implementing the test items, and so testers and programmers must have a shared understanding of what will be built and delivered. These stakeholders receive defect reports and other test results. In many cases, they need to act on these results (for example, when fixing a defect reported by testers).

Database administrators and architects, network architects and administrators, system architects and administrators, and software designers:

These stakeholders are involved in designing the software and the hardware, the database, and network environments in which the software will operate. As with programmers, they receive defect reports and other test results, sometimes based on early static tests such as design reviews. These stakeholders also may need to act on these results.

Technical support, customer support, system operators, and help desk staff:

These stakeholders support the software after it goes into production, including working with users and customers to help them realize the benefits of the software. Those benefits come from the features and quality of the software but can be compromised by defects in the software that remained when it was released.

A business stakeholder on the other hand is someone who is not so concerned with the design and implementation details but does have specific needs that the system must fulfil. In the following the business stakeholders and their role are identified:

Marketing and sales personnel, business analysts, user experience professionals, and requirements engineers:

These stakeholders work with actual or potential users or customers to understand their needs. They then translate that understanding into specifications of the features and quality attributes required in the software.

Project managers, product managers, and program managers:

These stakeholders have responsibilities to the organization for the overall success of their projects, products, or programs, respectively. Success involves achieving the right balance of quality, schedule, feature, and budget priorities. They often have some level of budget authority or approval responsibility, which means that their support is essential to obtaining the resources required by the test team. They will also often work directly with the test manager throughout the testing activities, reviewing and approving test plans and test control actions.

Users (direct and indirect) and customers:

Direct users are those who employ the software for work, play, convenience, transportation, entertainment, or some other immediate need, while indirect users are those who enjoy some benefit produced or supported by the software without actually employing it themselves. Customers, who might also be direct or indirect users, are those who pay for the software. They are the ones whose needs, ultimately, are either satisfied or not by the software (though satisfaction is typically more of a spectrum than a binary condition).

The above identified list of stakeholder is by no means exhaustive but help to get one started thinking about who are stakeholders. In addition, in some cases a stakeholder may be simultaneously a business stakeholder and a technical stakeholder. So, don't worry too much about trying to place a particular stakeholder clearly into one category or another.

Having identified the stakeholders, the next step is to work directly with each stakeholder, or a qualified representative of that stakeholder group, to understand their testing expectations and objectives. That might sound easy enough, but when you start that process, you'll find that most stakeholders cannot answer a straightforward question such as, "What are your testing expectations?" They simply will not have thought about it before, at least not with any real clarity, so the initial answers will often be something like, "Just make sure the system works properly before we ship it."

This, of course, is an unachievable objective, due to two fundamental principles of testing:

the impossibility of exhaustive testing and

testing's ability to find defects but not to demonstrate their absence.

You'll need to discuss the individual stakeholder's needs and expectations in terms of quality and then the extent to which testing can realistically help the organization achieve their objectives. These discussions may initially be hazy and often unrealistic, but ultimately they should coalesce into a crisp, achievable mission and set of objectives. The specific mission and objectives that you and your stakeholders arrive at will vary, so we can't provide a one-size-fits-all list here. However, a typical mission statement might be something along the following lines:

To help the software development and maintenance teams deliver software that meets or exceeds our customers' and users' expectations of quality."

Notice the word help here, which in this context means to assist (i.e., cooperate effectively with) others involved in the software process to achieve some goal. In this case, the goal is the delivery of quality software. You want to avoid mission statements like this one:

To ensure that our customers and users receive software that meets or exceeds their expectations of quality."

There are two problems with this mission statement. First, it leaves out the software development and maintenance team, making software quality a testing problem entirely. Second, it uses the word ensure, which in this context means to guarantee that customers and users are satisfied or even delighted with the quality of the software. Due to the two principles mentioned earlier, both of these flaws are fatal. Testing is an essential part of a broader strategy for quality, but it is only a part of that strategy and can never guarantee quality by itself.

Given a realistic and achievable mission for testing, what objectives might you establish that would support achieving that mission? While your objectives will vary, typically we see one or more of the following:

Find critical defects. Critical defects are those that would significantly compromise the customers' or users' perception that their expectations of quality had been met; that is, defects that could affect customer or user satisfaction. Finding defects also implies gathering the information needed by the people who will fix those defects prior to release. The person assigned to fix a defect can be a programmer (when the defect is in software), a business analyst (when the defect is in a requirements specification), a system architect (when the defect is in design), or some other technical or business stakeholder.

Find noncritical defects. While the main focus of testing should be on critical defects, there is also value in finding defects that are merely annoyances, inconveniences, or efficiency sappers. If not all noncritical defects are fixed, which is often the case, at least many of them can be known. With such defects known—especially if the test team can discover and document workarounds—technical support, customer support, system operators, or help desk staff can more effectively resolve production failures and reduce user frustrations.

Manage risks to the quality of the system. While testing cannot reduce the risk of quality problems to zero, anytime a test is run, the risk of remaining, undiscovered defects is reduced. If we select the tests that relate to the most critical quality risks, we can ensure that the maximum degree of quality risk management is achieved. If those tests are run in risk order, the greatest degree of risk reduction is achieved early in the test execution period. Such a risk-based testing strategy allows the project team to achieve a known and acceptable level of quality risk before the software is put into production.

Build confidence. Organizations, especially senior managers in organizations, don't like surprises. At the conclusion of a software development or maintenance project, they want to be confident that the testing has been sufficient, the quality will satisfy customers and users, and the software is ready to release. While it cannot provide 100 percent surety, testing can produce, and test results reporting can deliver, important information that allows the team to have an accurate sense of how confident they should be.

Ensure alignment with other project stakeholders. Testing operates in the context of a larger development and maintenance project and must serve the needs of the project stakeholders. This means working collaboratively with those stakeholders to establish effective and efficient procedures and service-level agreements, especially for processes that involve handoffs between two or more stakeholder teams. When work products are to be delivered to or from testing, then test managers must work with the relevant stakeholders to establish reasonable entry criteria and quality gates for those work products.

At this point in the process, you will have established a clearly defined mission and objectives for testing. What should be done with this important information? The best practice is to document them in a test policy. The test policy can be a standalone document, or it can be a page on a company intranet or wiki.

In addition to the testing mission and objectives, the test policy should talk about how testing fits into the organization's quality policy, if such a policy exists. If no such quality policy exists, it's important to avoid the trap of calling this document or page the "quality assurance policy" because testing by itself cannot assure quality.

If your group has the phrase quality assurance in its title, and therefore you must call this page the "quality assurance policy," be careful to explain in the policy that testing is not a complete solution to the challenge of delivering high-quality software.

Conclusion

This activity presented the process of defining test mission. Defining such a document is a first step towards bringing real value to software quality. The preparation steps and elements of test mission were also discussed.

Assessment

1. Identify the stakeholders and develop a sketch test mission for an AVU web portal.

Testing Strategies

Introduction

A testing strategy is a general approach to the testing process rather than a method of devising particular system or component tests.

Different testing strategies may be adopted depending on the type of system to be tested and the development process used.

Activity Details

Generic Test Strategies

There are two different strategies available: Top-Down Testing and Bottom-Up Testing.

In Top-Down Testing, high levels of a system are tested before testing the detailed components. The application is represented as a single abstract component with sub-components represented by stubs. Stubs have the same interface as the component but very limited functionality.

After the top-level component has been tested, its sub-components are implemented and tested in the same way. This process continues recursively until the bottom - level components are implemented. The whole system may then be completely tested. Top-down testing should be used with top-down program development so that a system component is tested as soon as it is coded. Coding and testing are a single activity with no separate component or module testing phase.

If top-down testing is used, unnoticed design errors may be detected at an early stage in the testing process. As these errors are usually structural errors, early detection means that extensive re-design re-implementation may be avoided. Top-down testing has the further advantage that we could have a prototype system available at a very early stage, which itself is a psychological boost. Validation can begin early in the testing process as a demonstrable system can be made available to the users.

Bottom-Up Testing is the opposite of Top-Down. It involves testing the modules at the lower levels in the hierarchy, and then working up the hierarchy of modules until the final module is tested. This type of testing is appropriate for object-oriented systems in that individual objects may be tested using their own test drivers. They are then integrated and the object collection is tested.

Conformance Testing

·Conformance testing, also known as compliance testing, is a methodology used in engineering to ensure that a product, process, computer program or system meets a defined set of standards. These standards are commonly defined by large, independent entities such as the International Standard Organization (ISO), the Institute of Electrical and Electronics Engineers (IEEE), the World Wide Web Consortium (W3C) or the European Telecommunications Standards Institute (ETSI).

Conformance testing can be carried out by private companies that specialize in that service.

In some instances the vendor maintains an in-house department for conducting conformance tests prior to the initial release of a product or upgrade. In the software industry, once the set of tests has been completed and a program has been found to comply with all the applicable standards, that program can be advertised as having been certified by the organization that defined the standards and the corporation or organization that conducted the tests.

Doing Compliance check is quite straightforward. A set of standards and procedures are developed and documented for each phase of the development lifecycle. Deliverables of each phase needs to compare against the standards and find out the gaps. This can be done by the team through the inspection process, but highly recommended for an independent team to do it. After the end of the inspection process, the author of each phase should be given a list of non compliant areas that needs to be corrected.

The inspection process should again be done after the action items are worked upon, to make sure that the non conformance items are validated and closed.

Compliance testing is performed to ensure the compliance of the deliverables of each phase of the development lifecycle. These standards should be well understood and documented by the management. If required training sessions should be arranged for the team. Compliance testing is basically done through the inspection process and the outcome of the review process should be well documented.

Conclusion

This activity presented the concepts of testing strategies and compliance testing. Two main testing strategies were described, the top down and bottom up test strategies. The process of compliance testing was also highlighted.

Assessment

1. State the factors that influences use of top down or bottom up testing strategies
2. What are the key aspects that drives compliance testing?

Validating Preliminary Designs Through Prototyping

Introduction

The old saying, "a picture speaks a thousand words" captures what prototyping is all about. Prototyping use visuals to describe thousands of words' worth of design and development specifications for a system's look and behavior.

In an iterative approach to system design, prototyping is the process of quickly building up the future state of a system, be it a website or application, and validating it with a broader audience of users, stakeholders, developers and designers. Doing this rapidly and iteratively generates feedback early and often in the process, improving the final design and reducing the need for changes during development. Prototypes range from rough paper sketches to interactive simulations that look and function like the final product. The key to successful prototyping are revising quickly based on feedback and using the appropriate prototyping approach. Prototyping helps developers experiment with multiple approaches and ideas. It facilitates discussion through visuals instead of words, it ensures that everyone shares a common understanding, and it reduces risk and avoids missed requirements, leading to better design rapidly.

Activity Details

Prototyping involves multiple iterations of a three-step process:

Prototype - Convert the users' description of solution into a build up, factoring in user experience standards and best practices.

Review - Share the prototype with users and evaluate whether it meets their needs and expectations.

Refine - Based on feedback, identify areas that need to be refined or further defined and clarified.

The prototype usually starts small, with a few key areas built up, and grows in breadth and depth over multiple iterations as required areas are built out, until the prototype is finalized and handed off for development of the final product. The rapidness of the process is most evident in the iterations, which range from real-time changes to iteration cycles of a few days, depending on the scope of the prototype.

Scoping a Prototype

The word prototype often captures images of a coded, fully functioning version of an application or interface. Prototypes are not intended to evolve into fully functional solutions, but are meant to help users visualize and craft the user experience of the final product. With that in mind, when scoping a prototype, decide on a few key issues before beginning any prototyping work.

What Needs to Be Prototyped?

Good candidates for prototyping include complex interactions, new functionality and changes in workflow, technology or design.

For example, prototyping search results is useful when you want to depart significantly from the standard search experience; say, to introduce faceted search or the ability to preview a document without leaving the search results.

How Much Should Be Prototyped?

A good rule of thumb is to focus on the 20% of the functionality that will be used 80% of the time; i.e. key functionality that will be used most often. Remember, the point of prototyping is to showcase how something will work or, in later stages, what the design will look like, without prototyping the entire product.

Find the Story

After identifying the areas to be prototyped, merge them together into one or more scenarios: identify the coherent paths through the user experience that the prototype simulates. For example, a website that sells shoes, one scenario could be "Boring Joe" buying the exact same Nike running shoes that he bought six months ago, while another scenario could be "Exploring Sam" browsing through size 10s to find a pair of Oxfords and pair of loafers that interest him.

Plan Your Iterations

The entire prototype is usually not built in a single iteration but rather piece by piece. A good approach is to start prototyping broadly and widely and then put focus into selected areas of the solution. For a website, this would mean building out the "home page" and landing pages for the main sections in the first iteration (sometimes referred to as a horizontal prototype) and then reviewing and revising that framework. Subsequent iterations could narrow down into one or more sections of the website (a vertical prototype); for a media download website, this could be the steps a user would take to find a video and to download it, or how they would manage the media in their online library.

Choose the Appropriate Fidelity

Fidelity refers to how closely a prototype resembles the final solution. There are multiple dimensions of fidelity, and prototypes can lie anywhere on the spectrum for each of these dimensions. Figure 1, demonstrate different fidelity level.

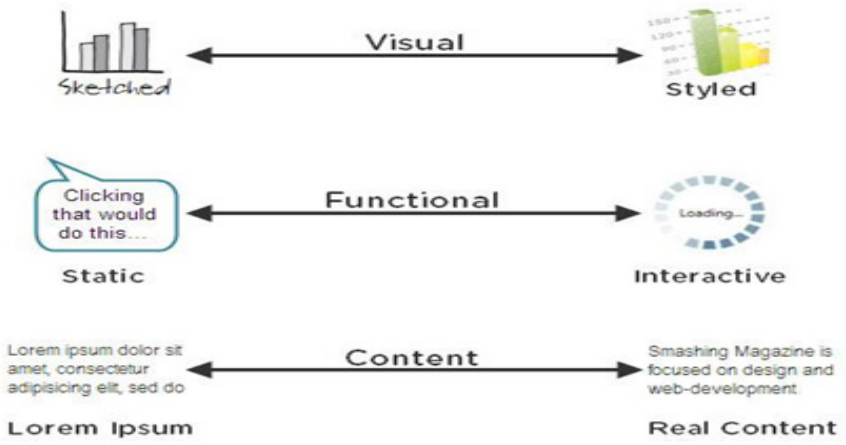

Figure 1: Fidelity Levels

Depending on the stage of the design process and the goals of the prototype, select the appropriate fidelity as shown in Figure 1 for each of the following dimensions:

Visual fidelity (sketched styled)

Look and feel are the most noticeable dimension of a prototype's fidelity and, if not properly selected, can sidetrack prototype reviews. For example, going "hi-fi" too soon and users will focus on visual design, which is not appropriate in early stages. From a visual standpoint, prototypes do not have to be pixel perfect but should be proportional. For example, if the left navigation area has to occupy one-fifth of a 1024-pixel screen, it does not need to be exactly 1024 pixels wide, as long as it is proportionally depicted in the prototype. As prototyping progresses through the design cycle, increase visual fidelity as needed by introducing elements of style, color, branding and graphics.

Functional fidelity (static interactive)

Does the prototype reveal how the solution will work (static) or does it appear to be fully functional and respond to user input (interactive)? This dimension is less of a distraction to users, but adding interactivity in subsequent iterations increases functional fidelity and allows the prototype to be used for usability testing and training and communications.

Content fidelity (lorem ipsum real content)

Another dimension that often distracts users is the content that is displayed in the prototype. Twisting lines and dummy text like lorem ipsum are useful to avoid in early stages of prototyping. But as the prototype is refined, evaluate the need to replace dummy text with real content to get a feel for how it affects the overall design.

The Prototyping Spectrum

Low Fidelity

The quickest way to start prototyping is also the easiest: putting pen/pencil to paper. Sketching on paper is a low-fidelity approach that anyone can do; no special tools or experience required. Most often used during the early stages of a design cycle, sketching is a quick way to create rough builds of design approaches and concepts and to get feedback from users. Paper prototyping is ideal during brainstorming and conceptualization and can be done alone in a workspace with a sketchbook or in a group with a flip chart (or whiteboard) and markers. Figure 2, presents paper sketches to illustrate low fidelity.

Figure 2: Low Fidelity Prototypes

Lying at the low-fidelity end of the prototyping spectrum, paper prototypes are static and usually have low visual and content fidelity. This forces users to focus on how they will use the system instead of what it will look like, and it makes designers more open to changes based on user feedback. Low-fidelity prototyping is easier to learn and lets you make changes easily and quickly.

Medium Fidelity

As we start using computer-based tools such as Visio and Omnigraffle to prototype, the fidelity increases on most fronts, yielding medium-fidelity prototypes. Wireframes, task flows and scenarios that are created with these tools take more time and effort but look more formal and refined.

While visual elements of branding, colors and style can be introduced, prototypers often stay away from them, focusing instead on demonstrating the behavior of the application. Interactivity can be simulated by linking pages or screens, but functional fidelity here is medium at best. These prototypes are best suited to determining whether user needs are met and whether the user experience is optimal. A screenshot in Figure 3, illustrates a medium fidelity prototype.

Figure 3: Medium Fidelity Prototype

There are two reasons why one might intentionally make a medium-fidelity prototype not look like a medium-fidelity prototype:

The first is that, by using Balsamiq or sketchy Visio stencils to make the prototype look low fidelity, you force users to view it as a draft or work in progress, rather than a polished and finished product.

The second is that, by giving the prototype a high visual fidelity (for instance, in a comprehensive layout done in Photoshop), you get the user to focus on the visual design and look and feel, including colors, fonts, layout, logo and images.

The speed of medium-fidelity prototyping is achieved with templates, stencils and reusable widgets and elements. It gets faster as you become more proficient with your tools of choice.

High Fidelity

High-fidelity prototypes are the most realistic and are often mistaken for the final product, but they are usually time-intensive. A screenshot in Figure 4, illustrates a high-fidelity prototype.

A few years ago, the only way to create high-fidelity prototypes was to actually code using a programming language, which often required the designer and developer to work together.

These days, however, application-simulation tools allow non-technical users to drag and drop UI widgets to create high-fidelity prototypes that simulate the functionality of the final product, even for business logic and database interactions. Axure and iRise are some examples of application-simulation tools that can be used to create high-fidelity prototypes.

Figure 4: High-Fidelity Prototype

These prototypes are appropriate when high visual and functional fidelity is required; for example, when introducing a new technology (say, when moving from a mainframe application (yes, they still exist!) to a desktop solution. Most of these prototypes cannot be converted to usable code, but they serve as an excellent reference for developers. These are also useful for conducting usability testing and training users.

High-fidelity prototyping is relatively rapid, considering the level of interactivity and fidelity involved, and it can be accelerated by using drag-and-drop simulation tools. In addition, some of these tools facilitate the gathering of user feedback and documenting requirements, which further speed up the design process.

Even though you do not need to learn a new programming language, these tools do require a learning effort.

Selecting a Fidelity Level

In choosing the prototype fidelity, there is no one correct approach. Most designs of new products are best started with sketches, then moving to either medium- or high-fidelity prototypes, depending on the complexity of the system and the requirements of the dimensions of fidelity. For example, in a pharmaceutical industry application development project, starting from whiteboards to interactive prototypes that have high functional and content fidelity but low visual fidelity, the client may care less about the look and feel than about adhering to corporate guidelines. On the other hand, in a retail application development project, an interactive prototype may need to have high visual and functional fidelity. The content fidelity does not matter because the clients would be reusing content and possibly already familiar with it. To them, the look and feel and interactive experience matter more, especially, being their first implementation of the application.

Selecting Tools

Depending on your approach, you have a wide variety of tools to choose from. Each tool has its own feature set and strengths. Based on your needs and the requirements of the projects you work on, evaluate which tool would be most appropriate. Here are some questions to consider when evaluating tools:

1. How easy is it to learn and use the tool?
2. Is it flexible to support prototypes for Web, packaged and custom software applications, as well as desktop and mobile applications?
3. Is there a repository of reusable stencils, templates or widgets available?
4. How easy is it to share the prototype with others for review? Can their feedback be captured using the tool?
5. How easy is it to make changes on the fly or to incorporate feedback?
6. Does it have any collaboration features, such as allowing multiple people to work on it at the same time?
7. What are the licensing terms and costs?

Conclusion

In this activity the prototyping method was present. Prototyping provide a quick way of building product design due to its visual way of presenting preliminary designs for user reviews.

This in-build validation mechanism by stakeholders involvement make prototyping deliver better design rapidly.

Assessment

1. Explain the concepts of low fidelity, medium fidelity and high fidelity prototypes
2. List any five prototyping tools and for each briefly evaluate its capability

UNIT SUMMARY

This unit presented the concepts of verification and validation and associated methods for improving software quality. The unit highlighted the process of test mission creation along with a discussion on testing strategies. The prototyping method was thoroughly discussed as a mechanism for rapid design development.

Unit Assessment

Check your understanding!

Miscellaneous Questions

Instructions

a. Using examples distinguish between validation and verification

b. Considering the main testing strategies described using examples the environment that suite each for use.

c. Using an AVU portal as a reference. Which aspects when developing such a portal might be suitable for prototyping?

Grading Scheme

As per the offering Institution

Feedback

mailto:njulumi@gmail.com

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources.

Unit 4. Quality Management

Unit Introduction

The quality of software has improved significantly over the past two decades. One reason for this is that companies have used new technologies in their software development process such as object-oriented development, CASE tools, etc. In addition, a growing importance of software quality management and the adoption of quality management techniques from manufacturing can be observed. However, software quality significantly differs from the concept of quality generally used in manufacturing mainly for the next reasons:

1. The software specification should reflect the characteristics of the product that the customer wants. However, the development organization may also have requirements such as maintainability that are not included in the specification.
2. Certain software quality attributes such as maintainability, usability, reliability cannot be exactly specified and measured.
3. At the early stages of software process it is very difficult to define a complete software specification. Therefore, although software may conform to its specification, users don't meet their quality expectations.

The software quality management is split into three main activities:

1. Quality assurance - The development of a framework of organizational procedures and standards that lead to high quality software.
2. Quality planning - The selection of appropriate procedures and standards from this framework and adapt for a specific software project.
3. Quality control - Definition of processes ensuring that software development follows the quality procedures and standards.

Quality management provides an independent check on the software and software development process. It ensures that project deliverables are consistent with organizational standards and goals. It is general, that the quality of the development process directly affects the quality of delivered products. The quality of the product can be measured and the process is improved until the proper quality level is achieved. Figure 1, illustrates the process of quality assessment based on this approach.

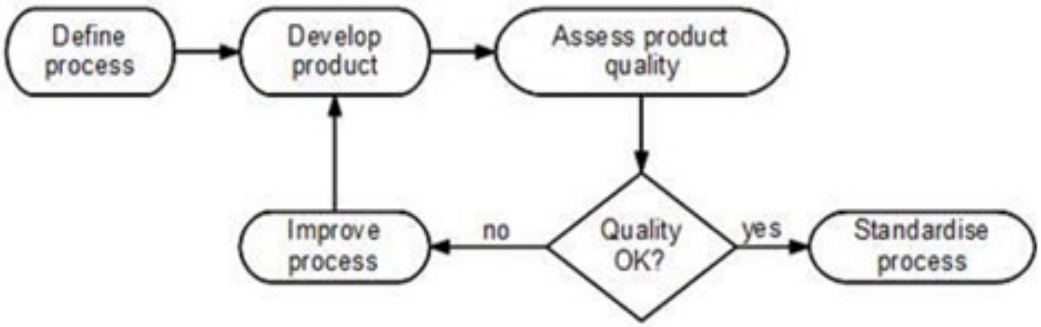

Figure 1: Process based quality assessment

In manufacturing systems there is a clear relationship between production process and product quality. However, quality of software is highly influenced by the experience of software engineers. In addition, it is difficult to measure software quality attributes, such as maintainability, reliability, usability, etc., and to tell how process characteristics influence these attributes. However, experience has shown that process quality has a significant influence on the quality of the software. Process quality management includes the following activities:

1. Defining process standards.
2. Monitoring the development process.
3. Reporting the software.

Unit Objectives

- Upon completion of this unit you should be able to:
- Describe the quality management concept
- Develop a framework of organizational procedures and standards that lead to high
- quality software.
- Select appropriate procedures and standards and adapt for a specific software project.
- Apply the quality procedures and standards.

KEY TERMS

- **Quality management:** Independent check on the software and software development process to promote quality
- **Quality standards**: a framework for implementing the quality assurance process
- **Quality plan:** Defines the quality requirements of software and describes how these are to be assessed.
- **Quality control:** monitoring the software development process to ensure that quality assurance procedures and standards are being followed.

Learning Activities

Quality Assurance and Standards

Introduction

Quality assurance is the process of defining how software quality can be achieved and how the development organization knows that the software has the required level of quality. The main activity of the quality assurance process is the selection and definition of standards that are applied to the software development process or software product.

Activity Details

There are two main types of standards, the product standards and the process standards. The product standards are applied to the software product, i.e. output of the software process. The process standards define the processes that should be followed during software development. The software product standards are based on best practices and they provide a framework for implementing the quality assurance process.

The development of software engineering project standards is a difficult and time consuming process. National and international bodies such as ANSI and the IEEE develop standards that can be applied to software development projects. Organizational standards, developed by quality assurance teams, should be based on these national and international standards. Table1 shows examples of product and process standards.

Table 1: Examples of product and process standards

Product standards	Process standards
Requirements document structure	Project plan approval process
Method header format	Version release process
Java programming style	Change control process
Change request form	Test recording process

ISO Quality Standards

ISO 9000 is an international set of standards that can be used in the development of a quality management system in all industries. ISO 9000 standards can be applied to a range of organizations from manufacturing to service industries. ISO 9001 is the most general of these standards. It can be applied to organizations that design, develop and maintain products and develop their own quality processes. A supporting document (ISO 9000-3) interprets ISO 9001 for software development. The ISO 9001 standard isn't specific to software development but includes general principles that can be applied to software development projects. The ISO 9001 standard describes various aspects of the quality process and defines the organizational standards and procedures that a company should define and follow during product development. These standards and procedures are documented in an organizational quality manual. The ISO 9001 standard does not define the quality processes that should be used in the development process. Organizations can develop own quality processes and they can still be ISO 9000 compliant companies. The ISO 9000 standard only requires the definition of processes to be used in a company and it is not concerned with ensuring that these processes provide best practices and high quality of products. Therefore, the ISO 9000 certification doesn't means exactly that the quality of the software produced by an ISO 9000 certified companies will be better than that software from an uncertified company.

Documentation standards

Documentation standards in a software project are important because documents can represent the software and the software process. Standardized documents have a consistent appearance, structure and quality, and should therefore be easier to read and understand. There are three types of documentation standards:

1. Documentation process standards - These standards define the process that should be followed for document production.
2. Document standards - These standards describe the structure and presentation of documents.
3. Documents interchange standards - These standards ensure that all electronic copies of documents are compatible.

Conclusion

The main activity of the quality assurance process is the selection and definition of standards that are applied to the software development process or software product. This activity presented the two types of standards applied in software quality assurance process. The ISO and documentation standards were also discussed.

Assessment

What are the two type of standards

1. Explain the role of standard in software quality assurance
2. In which category of standard does each of the following falls?
3. ISO 9000 standards

Documentation standards

Quality Planning

Introduction

Quality planning is the process of developing a quality plan for a project.

Activity Details

The quality plan defines the quality requirements of software and describes how these are to be assessed. The quality plan selects those organizational standards that are appropriate to a particular product and development process. Quality plan has the following parts:

1. Introduction of product.
2. Product plans.
3. Process descriptions.
4. Quality goals.

5. Risks and risk management.

The quality plan defines the most important quality attributes for the software and includes a definition of the quality assessment process. Table 2, shows generally used software quality attributes that can be considered during the quality planning process.

Table 2: Software quality attributes

Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

This activity described the process of quality planning.

Assessment

1. What is quality planning?
2. Identify the main parts of a quality plan

Quality Control

Introduction

Quality control provides monitoring the software development process to ensure that quality assurance procedures and standards are being followed. The deliverables from the software development process are checked against the defined project standards in the quality control process. The quality of software project deliverables can be checked by regular quality reviews and/or automated software assessment. Quality reviews are performed by a group of people. They review the software and software process in order to check that the project standards have been followed and that software and documents conform to these standards. Automated software assessment processes the software by a program that compares it to the standards applied to the development project.

Activity Details

Quality reviews

Quality reviews are the most widely used method of validating the quality of a process or product. They involve a group of people examining part or all of a software process, system, or its associated documentation to discover potential problems. The conclusions of the review are formally recorded and passed to the author for correcting the discovered problems. Table 3, describes several types of review, including quality reviews.

Table 3: Types of review

Review type	Principal purpose
Design or program inspections	To detect detailed errors in the requirements, design or code.
Progress reviews	To provide information for management about the overall progress of the project.
Quality reviews	To carry out a technical analysis of product components or documentation to find mismatches between the specification and the component design, code or documentation and to ensure that defined quality standards of the organization have been followed.

Software measurement and metrics

Software measurement provides a numeric value for some quality attribute of a software product or a software process. Comparison of these numerical values to each other or to standards draws conclusions about the quality of software or software processes. Software product measurements can be used to make general predictions about a software system and identify anomalous software components.

Software metric is a measurement that relates to any quality attributes of the software system or process. It is often impossible to measure the external software quality attributes, such as maintainability, understandability, etc., directly. In such cases, the external attribute is related to some internal attribute assuming a relationship between them and the internal attribute is measured to predict the external software characteristic. Three conditions must be hold in this case:

1. The internal attribute must be measured accurately.

2. A relationship must exist between what we can measure and the external behavioural attribute.

3. This relationship has to be well understood, has been validated and can be expressed in terms of a mathematical formula.

The measurement process

A software measurement process as a part of the quality control process is shown in Figure 2. The steps of measurement process are the followings:

4. Select measurements to be made - Selection of measurements that are relevant to answer the questions to quality assessment.

5. Select components to be assessed - Selection of software components to be measured.

6. Measure component characteristics -The selected components are measured and the associated software metric values computed.

7. Identify anomalous measurements - If any metric exhibit high or low values it means that component has problems.

8. Analyze anomalous components - If anomalous values for particular metrics have been identified these components have to be examined to decide whether the anomalous metric values mean that the quality of the component is compromised. Generally each of the components of the system is analyzed separately. Anomalous measurements identify components that may have quality problems.

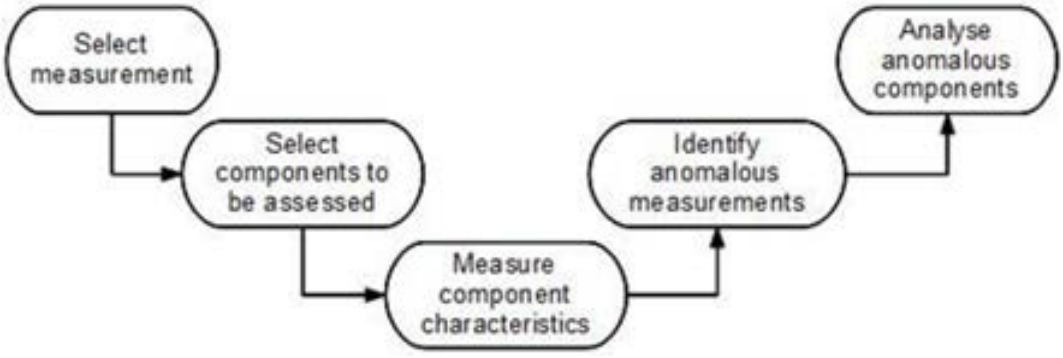

Figure 2: The software measurement process

Product metrics

The software characteristics that can be easily measured such as size do not have a clear and consistent relationship with quality attributes such as understandability and maintainability. Product metrics has two classes:

Dynamic metrics - These metrics (for example execution time) are measured during the execution of a program.

Static metrics - Static metrics are based on measurements made of representations of the system such as the design, program or documentation

Dynamic metrics can be related to the efficiency and the reliability of a program. Static metrics such as code size are related to software quality attributes such as complexity, understandability, maintainability, etc.

Conclusion

This activity presented quality control a mechanism for monitoring the software development process to ensure that quality assurance procedures and standards are being followed. The quality reviews method for validating the quality of a process or product was highlighted. Software measurement and metrics were discussed as instruments for assessing quality attributes of the software system or process.

Assessment

1. What is quality reviews?
2. Explain the steps for software measurement process?

UNIT SUMMARY

This unit presented quality management and quality management techniques. Quality management provides an independent check on the software and software development process. It ensures that project deliverables are consistent with organizational standards and goals. The activities of quality management were discussed including quality assurance, quality planning and quality control.

Unit Assessment

Check your understanding!

Miscellaneous Questions

Instructions

- Using examples distinguish between static and dynamic metrics
- Discuss the interdependence between the quality management activities.

Grading Scheme

As per the offering Institution

Feedback/ Answers

mailto:njulumi@gmail.com

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources.

Course Assessment

Comprehensive Exercise

Instructions

Assume that you have been tasked with the development of the AVU portal:

- Develop a sketch SQA plan
- Define the quality factors that may apply and identify any reliability critical aspects of the portal
- Draw a problem diagram that model the AVU portal
- Develop a sketch definition of standards and procedures which should be applied during each phase of development
- Develop a draft scenario of portal measurement process

Grading Scheme

As per the offering Institution grading policy

Answers

mailto:njulumi@gmail.com

Course References

Gerard O'Regan, Introduction to Software Quality, Springer, 2014

Jeff Tian, Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement , IEEE Computer Society, 2005

Darrel Ince, An Introduction to Software Quality Assurance and Its Implementation, McGraw-Hill, 1994

https://en.wikipedia.org/wiki/Problem_frames_approach[Accessed 23/02/2016]

Alistair Sutcliffe, User-Centred Requirements Engineering, Springer-Verlag London Limited 2002

https://en.wikipedia.org/wiki/Requirements_analysis [Accessed 23/02/2016]

David J. Gilmore, Russel L. Winder and Francoise Detienne, User-Centred Requirements for Software Engineering Environments, Springer-Verlag Berlin Heidelberg GmbH, 1994

http://www.rbcs-us.com/documents/Defining-Testing-Article.pdf [accessed 24/02/2016]

http://www.sqa.org.uk/e-learning/SDPL03CD/page_16.htm[accessed 24/02/206]

http://searchsoftwarequality.techtarget.com/definition/conformance-testing[accessed [24/02/2016]

Paul Ammann and Jeff Offutt, INTRODUCTION TO SOFTWARE TESTING, CAMBRIDGE UNIVERSITY PRESS, 2008

William E. Perry, Effective Methods for Software Testing, Third Edition, Wiley Publishing, Inc., Indianapolis, Indiana, 2006

http://www.softwaretestinghelp.com/what-is-conformance-testing/[accessed 24/02/2016]

https://www.smashingmagazine.com/2010/06/design-better-faster-with-rapid-prototyping/ [accessed 24/02/2016]

To contact parents and sister

The African Virtual University Headquarters

Cape Office Park

Ring Road Kilimani

PO Box 25405-00603

Nairobi, Kenya

Tel: +254 20 25283333

contact@avu.org

oer@avu.org

The African Virtual University Regional Office in Dakar

Université Virtuelle Africaine

Bureau Régional de l'Afrique de l'Ouest

Sicap Liberté VI Extension

Villa No.8 VDN

B.P. 50609 Dakar, Sénégal

Tel: +221 338670324

bureauregional@avu.org