TCC235-05-Software-Engineering-TMA1-LaiYeongWei-031150110-v0.8

by YeongWei Lai

FILE	235-05-SOFTWARE-ENGINEERING-TMA1-LAIYEONGWEI-031150110-V0.8.DOCX (797.39K)		
TIME SUBMITTED	31-AUG-2016 11:16AM	WORD COUNT	9244
SUBMISSION ID	699684883	CHARACTER COUNT	55087

WAWASAN OPEN UNIVERSITY

TCC235/05 Software Engineering

TMA 1

Yeong Wei Lai (031150110)

8/22/2016

yeongwei2004@yahoo.com

This document is a production outcome for the Tutor Marked Assignments distributed by Wawasan Open University for Online Assignment Submission (OAS).

Table of Contents

Overview

The writing for this Tutor Marked Assignment (TMA) is divided into 4 main sections with names starting with the word "Question" followed by a numeric number to indicate the index of the question according to the TMA questionnaire provided. Subsections are created depending on the necessity for answering the questions. As example, any texts that fall under the heading "Question N" can be assumed as the rightful answer for Question N. There is dedicated Reference section for supplementary purpose.

Question 1

The writing below consist of the eight software system product quality characteristics according to the ISO 25010 System and Software Quality model.

a. **Functional Suitability**

The functional suitability characteristic describes the degree which the developed software system meets the stated requirements gathered during the analysis and or design phase of the software development process. There are three associated sub characteristics; Functional completeness is concerned with the coverage of features delivered compared against the stated requirements; Functional correctness is concerned with the precision of results generated by the delivered software system; Functional appropriateness is concerned with the suitability of the software system used under certain circumstances to achieve some objectives.

b. **Performance Efficiency**

The performance efficiency characteristic describes the resource utilizations under some specific operational conditions. There are three associated sub characteristics; Time behavior is concerned with the response and processing time and the rates of certain measurements around the software system when loaded; Resource utilization is concerned with the amount and types of hardware resources required to execute functions normally; Capacity is concerned with the hard and soft limits of the software system measured in numeric values, usually related to the appetite of data consumption.

c. **Compatibility**

The compatibility characteristic describes the ease of information exchange between software systems under the same operational environment. There are two associated sub characteristics; Co-existence is concerned with multiple software systems deployed into the same operational environment and still able to perform individual functions normally; Interoperability is concerned with the interfaces for communications between software products.

d. Usability

The usability characteristic describes the ease-of-use for the software system to achieve some objectives. There are six associated sub characteristics; Appropriateness *recognizability* is concerned with the ease-of-identifiable product features for user needs and intuitions; Learnability is concerned with the amount of training efforts required to be able to operate the software system independently; Operability is concerned with the amount of effort needed to operate and maintain the software system for the long run; User error protection is concerned with the degree of fool proved implementations for minimizing user errors; User interface aesthetics is concerned with the pleasantness and satisfaction of user interaction with the software system; Accessibility is concerned with the need to support users at various circumstances or situations.

e. Reliability

The reliability characteristic describes the consistency in operational behavior under some operational conditions over some period of time. There are four associated sub characteristics; Maturity is concerned with the reliability of the software system under recommended operational environment to perform functions normally; Availability is concerned with the degree of usage availability whenever required; Fault tolerance is concerned with the software system response to either hardware or software faults; Recoverability is concerned with the effectiveness and efficiency of the software system recovery process after some outage occurred.

f. Security

The security characteristic describes the amount of protections and the level of authorizations for data access of the software system. There are five associated sub characteristics; Confidentiality is concerned with the data only being accessible to the rightful party with appropriate access rights; Integrity is concerned with data modification bounded by rules and approvals; Non-repudiation is concerned with evidence of events that occurred on the software system can be proved valid; Accountability is concerned with traceability of events up to some entities; Authenticity is concerned with resource to be proved as claimed.

g. **Maintainability**

The maintainability characteristic describes the flexibility for modifications of software system for improvements. There are five associated sub characteristics; Modularity is concerned with software system that is build based on the concept of discrete software components that each component has minimal impact with changes of other components; Reusability is concerned with the reuse of existing software assets for building other software systems; *Analysability* is concerned with ease to perform evaluation, impact analysis or diagnosis on the software system; Modifiability is concerned with the ease of modifications with minimal impact or degradation; Testability is concerned with the ease of identifiable test criteria to be executed on the software system.

h. **Portability**

The portability characteristic describes the ease of transferring software systems to other operational hardware or software environments. There are three associated sub characteristics; Adaptability is concerned with the ability for software system to adapt to changing or different hardware or software; *Installability* is concerned with the effort required to perform the installation and uninstallation of the software system; *Replaceability* is concerned with the degree of replacement for any existing software systems.

It is very important for software systems to be compliant to the ISO 25010 quality characteristic as described above for high quality assurance. The following describes a use case of software system that is compliant with the ISO 25010 quality standard.

The enterprise grade Operational Support System (OSS) for telecom players such as Maxis, Digi, Celcom, True Move Corporation or Telkomsel is chosen for illustration purpose here.

For functional suitability the OSS needs to be able to collect performance and configuration data from various domains, such as the Radio Access Network (RAN) for wireless domain or the Internet Protocol Core Network (IPCore) for wireline domain as part of the performance management module supporting various protocols such as Simple Network Management Protocol (SNMP) or File Transfer Protocol (FTP), these protocols should be completely compliant with industrial standards in order to deliver high functional suitability. The data is

processed then fed into the fault management module that accurately computes violations as alarms based on the metric values and threshold configurations. These alarms are then translated into the support tickets for the respective support service desk teams. Up to this point, the illustration is about the software system capable of delivering a complete closed looped solution with high accuracy about the operational insights for relevant stakeholder to make informed decisions. There should also be various user interfaces to cater for different user group needs such as dashboards for the Network Operation Center (NOC) team or the C-Level personnel.

For performance efficiency the OSS should support real-time or near-real-time or even streaming type of data collections for minimal latency. The processing model should handle scale-up and scale-out computational model for efficient use of for all computation stages such as aggregations or baselines. If machine resources are low, the processing model should be able to handle the processing requests with some pipelining semantics. Long term storage needs retention profiles to achieve optimal capacity utilization. Modern OSS delivers more business values through real-time processing and batch processing for various business needs, such system favors for either Lambda or Kappa Architecture.

For compatibility the OSS should be flexible and intelligent enough to alter necessary configuration in order to retain optimal functional health given that other software products may also exist in the same environment, such as dynamically changing ports or kernel configurations. The OSS should equipped with interfaces for integration with other software products such as Java Database Connection (JDBC) Driver for moving data to data ware house or reporting systems and Representational State Transfer Application Interface (RESTful API) to allow standardized way of inter-communications between software products.

For usability the OSS should come with out-of-the-box features that are friendly to end users, such as configuration templates and reporting templates that are easily recognized by industry users for immediate needs. With templates in place the learning curve will be smoother for end users. The OSS should also come with automation tools such as report scheduling or automatic fault propagation to reduce user attention. The user interface should have logic that prevents user from making input mistakes such as inserting negative values for durations. The user interface should use pleasant and industrial standard presentation schemes to present the

data such as green for normal, amber for required attention and red as critical. The application should be accessible by various means not only restricted to personal computers but also mobile devices.

For reliability the OSS should be able to run normally over a long period of time given the same operational environment. The OSS should behave in high availability mode regardless of the processing stage; the system will still be accessible by end users when required. Upon hardware or software failures, the application should be able to fail over to standby instances or secondary system to not disrupt business operations. After some planned outage the application should be able to recover itself in a seamless fashion, such as catching up with data processing to current time.

For security the OSS should have sufficient protections for data access since collected data are all confidential with customer information. There should be tight security and access policy that manage the data accessibility for rightful owners only. The OSS also needs to make sure that users are not able to alter data at will without any system level approvals, this is to ensure data integrity not compromised. Changes done to the system is to be tracked as historical data that cannot be changed. Any actions taken places on the system must be traceable either through access and transaction logs or audit tracks for responsible entity to be accountable. The login and credential modules should be sophisticated enough to be able to identify the identity of end user claimed to be.

For maintainability the OSS should be built with software as service model with components almost independent from one another to reduce down time and impact during maintenance activity. With the software as service model the components intuitively becomes reusable for different deployment scenarios or to produce new products since the components are not tightly stitched together. Each service component is independent to allows smoother troubleshooting and issues can easily be reproduced because it only requires minimal software components. Each service each should have its own configuration objects; therefore changing one will not affect the others but at the same time user should be able to push common configurations to all services. The steps needed to alter configurations should be simple with minimum clicks away. Since the entire OSS has been broken down into services with concise operational functionalities, this allows better test scenarios to be carried out because the operational behaviors are clearer and well defined.

For portability the OSS should be able to be transferred entirely from one operational environment to the other without losing data, there should be graceful switch over mechanisms in place to perform migration from one logical hardware node to the other. With such feature in place the installation and uninstallation process should require minimum amount of user inputs and the rest of the processes are handled automatically by the program. With software as service model, customer should be able to install the software into an existing production environment that is able to offer similar functionality and even cover new functionalities that was not covered by the existing software system.

Question 2

Based on the reading material provided, http://www.devtopics.com/20-famous-software-disasters/. The 2 software disasters selected are as follow.

1. **Patriot Fails Soldier (1991)**

   This incident was about an American defense missile system failed to intercept an incoming enemy missile hitting an army barracks that resulted 28 soldiers killed and about 100 people left injured.

   This American defense missile system is known as the Patriot Missile System. Its initial purpose was to protect against Soviet planes and missiles. However, twenty years later it got rebooted to be used against Iraqi missiles.

   When did it take place?

   February 25, 1991.

   Where did it take place?

   It was during the Gulf War era in Dhahran, Saudi Arabia.

   What was the impact?

   28 soldiers killed and about 100 people left injured.

   What was the main cause of the disaster?

   The entire disaster started when the Patriot Missile System was being assigned to an objective that it was not originally designed for. The initial objective of the Patriot Missile project was be used as a defense system to protect local airspace against intrusion of enemy aircrafts, in particular the Soviets. About 15 years later the system was brought back to be used against a specific variant of Ballistic Missiles known as the SCUD missile, which could travel twice the speed compared to the possible intrusion objects earlier.

Although the software of the system was modified to handle against these Ballistic Missiles but it contained bugs that required patching. Unfortunately the patch did not reach all intended destinations in time.

The bug caused accuracy discrepancies that resulted the system to miss tracking the incoming Ballistic Missiles therefore failed to intercept it before impact. This bug was due to some new functions that were not being propagated to all necessary parts of the program, to make matter worst it was written in Assembly Language which was hard to maintain.

Since the software was written about 15 years ago in Assembly Language, even the actual developers had hard time remembering the important details of the software, this were added challenges for understanding and maintaining the software.

Time was a great factor because the Americans needed defense system that was able to offer protections against these Ballistic Missiles, although the bug was announced to the respective military regions but the message was not clear about how to handle the system to avoid the bug being trigged.

Resources for testing was another constraint factor because the testing environment required actual running planes and missiles which was huge in expenses.

How could it be avoided?

It is inappropriate to reuse an existing system for new objectives. Although software can be patched properly and be tested thoroughly, the idea itself of reusing existing system for new objectives should be regarded as a final and risky option. Especially if the software is concerned with the safety of the American soldiers, such proposal should be regarded as highly risky and extreme alternatives should be aggressively and exhaustively considered instead.

The software was written using a Low Level Language, Assembly Language that is usually not easily understood by most. Therefore aggressive unit and integration testing should be carried out with highest priority during the software development or

enhancement cycles. Depending on the scale of the project, the required resources must be readily available to carry out these testing.

Better software development process should be used, since the Assembly Language is hard to understand, it is beneficial to have cycles of code reviews to make sure the amendments were made correctly and accurately and reviewed by other more experienced developers.

Use of heavy documenting strategy either in the code or separate documentation will improve the understanding of the software.

The strategies described above could be used to avoid such software disaster.

2. **British Passport to Nowhere (1999)**

This incident was about the South Wales Passport Agency not able to deliver the number of passports required during a summer holiday season. Given that a new system was installed and a law changed related children requiring passport to travel aboard.

The software system did not perform as expected and staffs were under trained. On top of that, since it was summer holiday and children were now required to have passport to travel. This situation has caused a spike in demand for passports but the department was unable to deliver due to software system faults and had no fall back plans.

The situation has caused loss of money, time and reputation of a governmental department.

When did it take place?

Summer. October, 1999.

Where did it take place?

United Kingdom.

What was the impact?

The impact was at least 500 people missed holiday trips that resulted in compensation of about £161,000. Extra spending of £6 million for staff overtime and £16,000 for umbrellas for applicants that were forced to queue under the rain. The Passport Agency also lost its Charter Mark which was an award for excellence in public service.

What was the main cause of the disaster?

The main cause was due to wrong time to market strategy, which was related to the planning phase of the software process cycle. Stake holders appeared to have neglected the market demand and market transition factors that resulted in inadequate risk analysis and mitigation plans. Therefore the Passport Agency could not recover in time when the system fault occurred.

The system has not undergone adequate testing for special events. Since summer holidays are seasonal, these could be considered as special events throughout the year and the system requirements should have anticipated the need for better reliability during these periods.

Staffs were not adequately trained therefore were not able to react effectively when the software system failed.

The software system had been deployed into production use in a Big Bang fashion because when fault presented the Agency was not able to react to the faults.

How could it be avoided?

During the requirement planning phase, market readiness must be taken into consideration. Rolling out a new software system during peak season puts software project at risk since these periods requires higher reliability of the software system. Therefore new software should be rolled out during low peak seasons or by stages.

To be ready with well-defined mitigation plans in the event of software failure. Software could fail for many reasons, either software itself, hardware failure or both.

Stakeholders got to be prepared if such incident occurred to have minimal impact to business operations.

Adequate quality assurance testing is not only required but also BETA rollout too. There are only that much of testing could be carried out according to factory testing specifications but not site testing specifications, and site testing only can be carried out if only the system is deployed into production environment. Therefore before a full system rollout some BETA rollout would discover unexpected production faults.

All stakeholders must be well trained for not only for operating the system but also for events if the system fails. The entire software system is not only made up of the executable software and operational hardware delivered, it also includes the administration and management protocol which are executed by human activities.

With the above, this software disaster could be avoided.

Question 3

Generally software process models are frameworks that guide software development team through the software development activities for greater possibility of success for the software project.

a. **Waterfall Model**

The waterfall software process model is the earliest software process model that has the attribute of behaving linear-sequentially with each phase executed one after another, each with distinct functions that does not overlap. It is usually used as the overall product life cycle for modern software development models. It was derived from the Mechanical Engineering domain of manufacturing business. Although not popular in modern times, however it had influence other modern software process models.

Below is a simplified pipeline of phases used in the waterfall model,

Requirement → *Design* → *Implementation* → *Testing* → *Maintenance*

The *Requirement* phase is about getting the software specifications from customers. During the *Design* phase, the customer requirements are translated into implementation details that include system architectures and functional requirements, it may also include any software and hardware dependencies. The *Implementation* phase is when the coding work are carried out based on the designs. In *Testing* phase, various test activities are carried out ensure the working software met stated requirements. If the testing went well, the working software will be delivered to customer. In the *Maintenance* phase, any errors found are corrected and new requirements as enhancements are being addressed. Each phase produces some outputs that are used as input for the next phase, the process direction moves from left to right as indicated by the arrows used above.

The advantages of using the waterfall model is that each phase is well defined, there is minimal ambiguity for scope of work. Since scope of work is well defined, departmentalization is possible for fixed resource allocations. This would result in a

very predictable schedule with clear milestones. Since most project variables are fixed, the project management activities can be carried out easily with reuse of project management templates.

The disadvantage of the waterfall model includes late evaluation of complete product, since working software is only delivered after the *Implementation* phase, there is very less visibility about the working software for stakeholders, not at least until the *Testing* phase, this puts the entire project on a higher risk and customer might be left confused because of the long wait time. Therefore this software process model is not suitable for projects with unclear requirements. Progress of project may be consistently reflected but might not be relevant to customer. Since integration testing in only carried out during the *Testing* phase , this possible "Big Bang" approach may cause more failures.

Therefore the waterfall development model is more suitable for projects with fixed, clear and concise requirements at the very beginning that usually happens for smaller software projects. It is also applicable for projects, if the technology used for building the end product is well understood and supported by an ample pool of resources. In the real world, software problems are only to become more complex, since waterfall model takes a rigid position; it is not suitable for most real world problems .

b. Spiral Model

The Spiral software process model is a combination of sequential and prototype model that fits project with high risk profile or requires continuous enhancement. It can be visually imagined as a spiral that circles out from some point of a Cartesian axis that is close to the origin. Each complete spiral or circle represents a software development cycle with outcome of prototypes or working software, known as Builds. Each cycle consists of four phases.

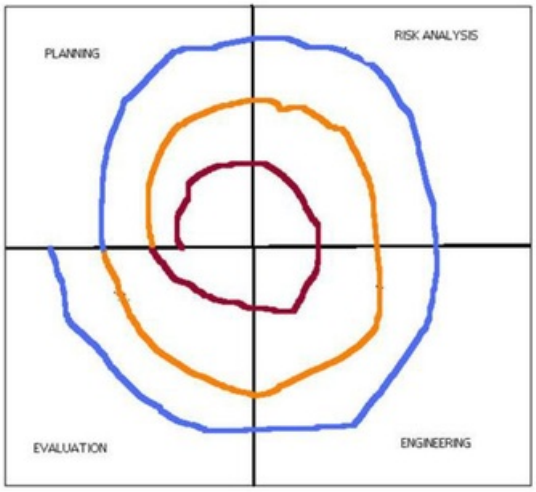

Figure 3.1 (http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/04/SDLC-Spiral-Model-1.jpg)

The first spiral is known as the baseline spiral represented in *Figure 3.1* with a maroon colored spiral. As development progresses the graph spirals out as represented by the amber and bluish color lines. There is not any fixed number of cycles, it all depends on the requirements of the software project.

As depicted in *Figure 3.1* there are four phases involved for each cycle. The *Planning* phase is used to determine the project objectives, alternative and constraints by gathering information from customer and other sources. By carefully studying the information for project feasibility along with review and walkthrough with stakeholders. The outcome of this phase would be a finalized business and system requirement specifications that will be used for the upcoming phases. Next is the *Risk Analysis* phase, the activity here includes identifying the risks then brain storming for alternatives or possible resolutions as finalized risk mitigation strategy. The outcome of this phase would be some artifacts that focus on risk profiles and possible resolutions. The third is *Engineering* phase which includes the development and testing activity of the software product. The outcome of this phase would be either prototype or working software. The last phase is the *Evaluation* phase that deals with planning for the following cycle for addressing any issues identified based on customer feedback and approval. Usually more features are needed for the upcoming

software version. The outcome of this phase would be some design enhancement documents for the next cycle to begin with.

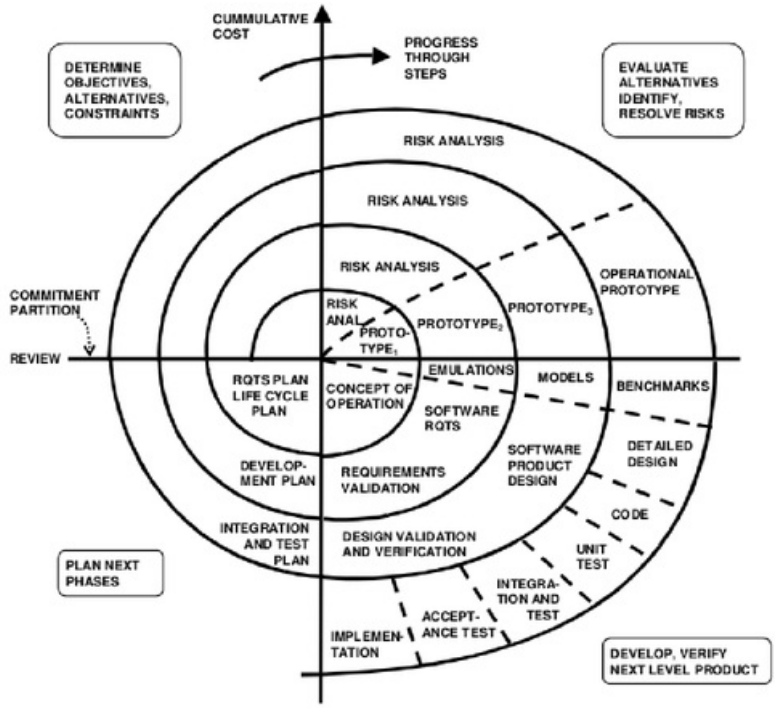

Figure 3.2 (http://xbsoftware.com/wp-content/uploads/2015/10/spiral-model-sdlc.png)

Figure 3.2 shows a more practical real world use of the Spiral process model with annotated details for each possible spiral segments. As shown, the Spiral model gives focus to risk analysis that each cycle has dedicated phase for risk analysis activities and as the progress spirals out, the prototype becomes more suitable as operational candidate and eventually goes through the required development activities in order to be production ready.

The advantages of are Spiral model are; it is flexible to accommodate new requirements, as these new requirements can be easily put into plan for the following cycle. Requirements are refined for accuracy, since each cycle produces some prototype, from these demonstration the requirements would become clearer after getting customers to evaluate. Since every cycle involved end users for evaluations, stakeholders get to witness tangible product as milestones and get a clearer picture of the final product. It offers better risk management with important and high risk features being delivered first and parts of system are being broken down to be

iteratively developed over various cycles. Features are added systematically as iterations spirals out, this offers better source code control for features to be incrementally introduced as opposed to "Big Bang" ideology. Every cycle has room for customer feedback, therefore the product will be more well aligned to customers' expectation. Since stakeholders get to see tangible outcome, the entire project becomes more transparent.

The disadvantages of Spiral model are; the need for more complex project management because to accommodate the strategy for mitigating risk and prototypes were developed at the beginning of the cycles, then followed by the actual product development cycles. These technicality might be harder to be communicated to all stakeholders, especially to the non IT-literate stakeholders. Since cycles are driven by customers' evaluation and feedback it is hard to estimate the end schedule and milestones of the software project and if expectations are not manage well, the spiral might go indefinitely. This model might overkill small scale projects, since small scale projects have low risk factors. In the risk analysis phase, there requires more experienced staffs for accurate risk analysis, each phase may involve different kind of risk therefore requires different expertise. The documentations might be too complicated as there are amendments for each iterations to address different issues.

Nevertheless each spiral iteration has both analysis and engineering elements which usually improves project success probability.

c. Iterative and Incremental Development

The general idea of using iterative and incremental development model is to avoid the inefficiencies and problems found with the waterfall model. This model often starts with a simple implementation, a subset of the entire software requirements and iteratively add small portion of enhancements to it. At the end of every cycle, the software is being reviewed for further requirements then the process is repeated. This model often starts even if the full requirements are not know because at each incremental delivery more requirements can be learnt.

The software model is suitable for situation where major requirements are already known with other details evolve over time. There could be a need for faster to market

to capture the market share. There would be situation for software products to use new technology that the development team is not familiar with, by using this approach as the product is being development, developers would learn more about the technology and subsequently add more features to the product, this effectively smoothens the learning curve. On the other hand if some required are not available the project still could start with development for domains that has resources available and subsequently introduce those other features at later iterations.

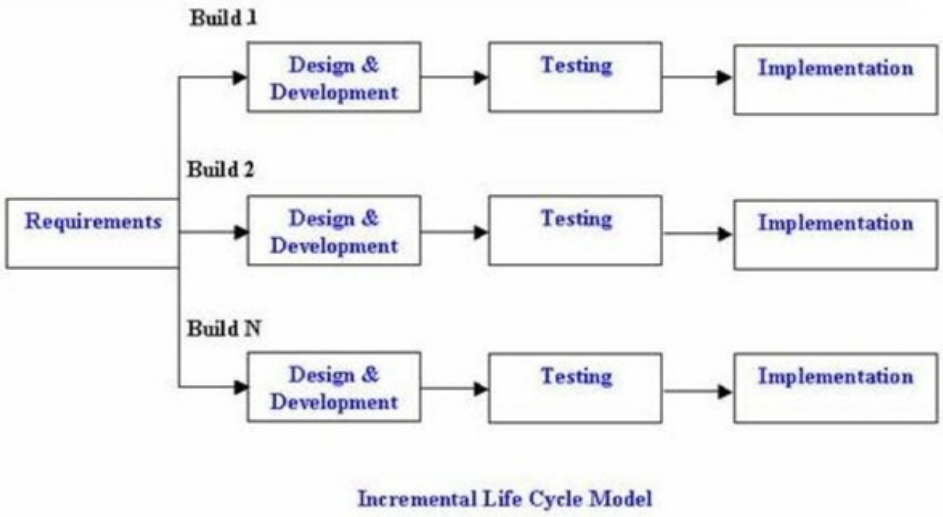

Figure 3.3 (http://istqbexamcertification.com/wp-content/uploads/2012/01/Incremental_model.jpg)

Figure 3.3 shows that each horizontal pipeline represents an iteration and its' vertical placement represent the incremental aspect of the software process model. Each iteration is carried out with similar fashion with same development activities and as each incremental of interactions are carried out more features are being introduced into the software product.

The advantages of this software process model is that there are opportunities for early view of some working software. As each iteration goes by, some periodic results are being captured that would give insights to project management. If planned properly, these iteration may happen in parallel. In general, it is less costly to accommodate requirement change because each iteration is expected to have some changes to

accommodate the incremental features. The testing activities might be easier because incremental functions are delivered and if planned correctly, only the delta features needed test. Through each iteration, stakeholders get the chance for learning mistakes and do better for the following iteration.

The disadvantages of this software process model is that each iteration might need different skillsets causing more resources to be used over the entire project. Since each iteration requires testing, more efforts might be needed for environment preparations to accommodate the next incremental Build. Since each iteration might have different objectives, project management might need more effort to track progress. Since each iteration is expected to accommodate changes, the system architecture and design might be at risk in the long run.

d. Prototyping Model

The Prototyping software process model uses the strategy of early approximation of a final software system known as prototype then presents it to customer for feedback then followed by refinements until an acceptable prototype model is produced and use it as software specification reference then the final software system can be built on top of this foundation.

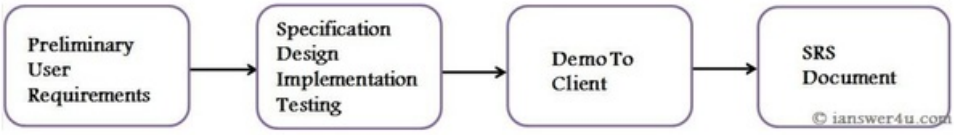

Figure 3.4 (http://1.bp.blogspot.com/-cvgwChf635A/TsFhfEawicI/AAAAAAAAAHs/f1gb92TdQ74/s1600/Prototyping_model_SDLC.jpg)

Figure 3.4 illustrates the core activities associate to Prototyping model. The *Preliminary User Requirement* activity is used to gather information from users at an incomplete high level specifications which would then be used to start building the prototype in the *Specification Design Implementation Testing* activity. During the *Demo To Client* activity, the prototype is show cased for feedback to identify potential problems and other requirements. The second and third activity might be repeated until an acceptable prototype is produced. Only then the Software Requirement

Specification is produced based on the latest version of prototype, only then the actual software development process happens.

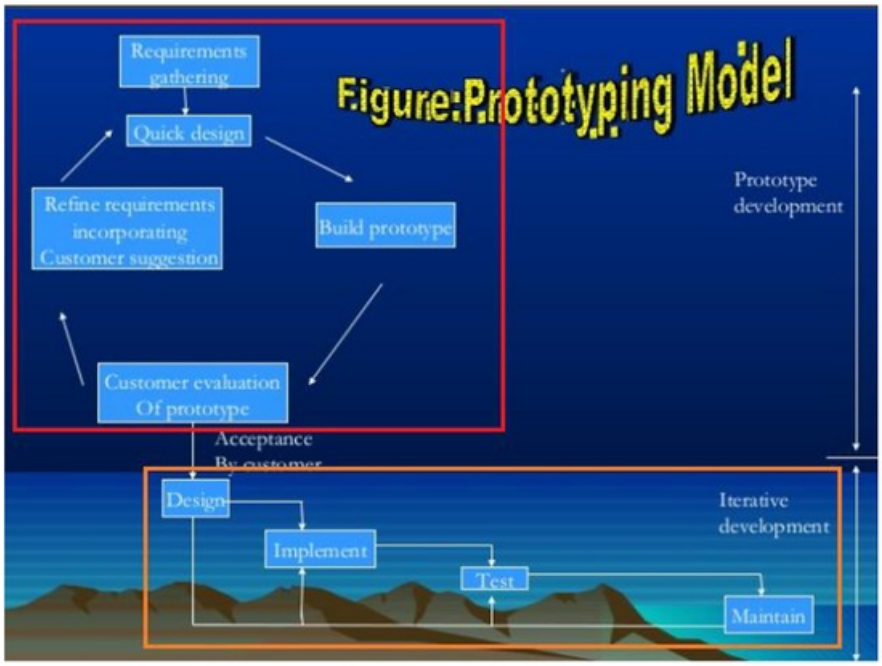

Figure 3.5 (http://www.slideshare.net/shuisharma/prototype-model)

Figure 3.5 shows a more complete cycle of the Prototype model. The activities that are highlighted by the red colored box is the core activities that was discussed earlier. Once customer has agreed an acceptable prototype, the actual software development activities begin as highlighted by the amber colored box, assuming the Waterfall Model would be used.

This software model is useful for scenario when requirements are not clear at the beginning and difficult to obtain then with the iterations of trial-and-error process between developers and end users is carried out to finalized the requirements

The advantages of Prototyping model are, it improves communications between end users and developer since the problem domain expert are the users, this reduces ambiguous requirements. With improved clarity of requirements, this results in reduced time and cost for modification since these cost increases exponentially for modifications as project proceeds. Tangible milestones increase end user confidence which might be easier candidate to source for investment funding. Reduced overall

project risk of failure since requirements are clearer. If prototypes were done correctly it might also reduce final product development time.

The disadvantages of Spiral model are; the prototyping activities might incur more development cost because usually the first few prototypes would be far off from user requirements, these startup cost would be counted as loss of investment. Too much customer involvement reduces developers' efficiency and affects team rhythms to progress. This model would produce confusing documentations since focused is more on building prototypes, best practices might be neglected. Sub-optimal final solution because prototype code based might not be in its' best form, since the focus for building prototypes is to get end user to buy-in, this situation might neglect the holistic analysis needed for complete solution which results in prototype code base being badly written and could steer down quality of final product. False impression to non IT-literate end users, these users might not be aware of the nonvisible functions such as security modules, load balancing, redundancy setups, exception handling, this might further cause end user to misunderstood final product deployment requirements.

e. Rational Unified Process

The Rational Unified Process (RUP) model is developed by the Rational Corporation, a division of IBM. The RUP is delivered not only as a software process model but it also comes with artifacts and software tools that support this software process. The RUP is essentially delivered as a product to customer that intends to implement the RUP in their software development process. It includes guidelines, templates and tool mentors, the unified knowledgebase is accessible through browsers and Rational software is used to execute the software development process. It is so flexible that this product comes with a Development Kit that enables users to customize software processes to suit the organization needs. At the primitive level, RUP provides a set of guidelines to software process via templates and tool mentors.

The RUP has to be illustrated first with the various concepts before the idea of an RUP can be delivered.

The RUP consist of three *Perspectives*; The *Dynamic* perspective is the moving elements of RUP that describes the phases of the software process model, essentially a

temporal related element. The *Static* perspective is the non-moving elements of RUP that describes the software process activities for any possible *Dynamic* perspective, essentially spatial related elements that moves along and dependent with the temporal elements. The *Practice* perspective is the recommended guide lines that can be used together with any of the *Dynamic* and *Static* elements.

The RUP consist of 6 *Workflows*. The *Business Modelling* workflow is concerned with the communication between the software and business community. The aim is to produce well documented business use cases to reflect business needs. The *Requirements* workflow is concerned with what the software system should do by using Vision document to illustrate personas and interactions with the software system components. The *Analysis and Design* workflow is concerned with the implementation details of the software system including the architecture, technology used, source code structure and any other information needed to kick start the actual software coding process. The *Implementation* workflow is concerned with the implementation of the software system in terms of coding activities. The *Test* workflow is concerned with verifying the developed software system against the agreed requirements. The *Deployment* workflow is concerned with the success of software product released to customer.

The RUP consist of 4 *phases*. The *Inception* phase is concerned with team to buy-in on the worthiness of the software project. The expected outcome would be the cycle objective. The *Elaboration* phase is concerned with the project architecture, required resources, use cases and most importantly the decision to commit to the cycle objective. It also includes the activity of prototyping for assessment. The expected outcome would be overall architecture of the project. The *Construction* phase is concerned with the software coding cycles until a deployable candidate is produced. The outcome of this phase is usually a BETA version of the final product. The *Transition* phase is concerned with development work based on the feedback from BETA release until a final General Availability product candidate is produced.

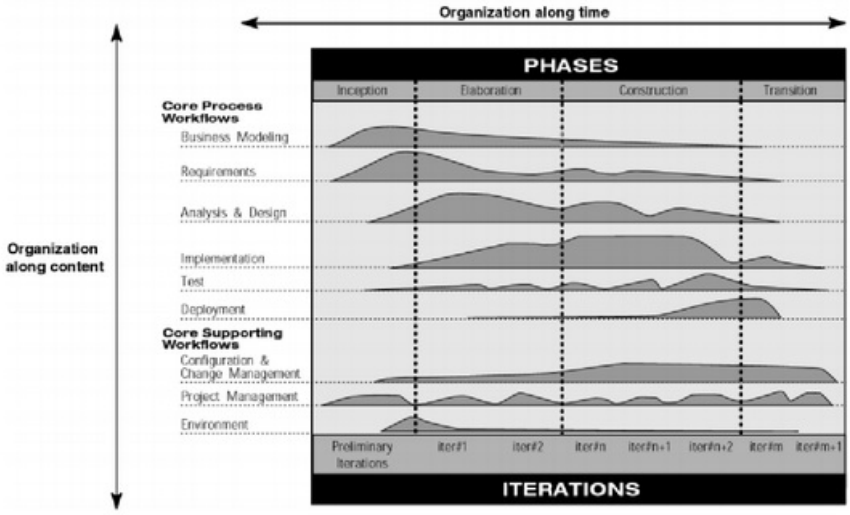

Figure 3.6

(https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf)

Figure 3.6 establishes the relationships for *Static* perspective, *Dynamic* perspective, *Workflows* and *Phases*. The *Dynamic* perspective is actually the *Phases* arranged along with time with on the horizontal axis that always move from in the direction from left to right. The *Static* perspective is actually the *Workflows* arranged along the vertical axis from top to bottom. Notice that at each crossed section there exist some bolded areas and depending on the location of the chart, the dimension of the bolded area changes, this indicates that as the software process progresses in time, the degree of *Workflow* relevance changes and usually in a reducting pattern.

Figure 3.6 effectively establishes the activity needed to be carried out as time progresses and it covers both the *Static* and *Dynamic* perspective. The *Practice* perspective covers the guidelines for the activities illustrated using *Static* and *Dynamic* perspective. Therefore to ensure successful software projects, RUP offers the following guidelines when carrying out software development activities,

1. *Develop software iteratively*. Waterfall model is too rigid for the real world software problems. Software development teams are encouraged to use iterative

approach with each iteration concludes with executable software release and be able accommodate uncertainties tactically.

2. *Manage requirements*. Through the use of business use cases approach for capturing functional requirements. This is to ensure the information is used to drive the software development activities that yields higher values to end user needs.
3. *Use component-based architectures*. This encourage the use of more resilient architecture that is flexible to accommodate changes, intuitively understandable and highly reusable. Builds subsystem that fulfill clear and concise function that is capable of being assembled into other products.
4. *Visually model software*. Use visuals methodologies to capture the structure and behavior of the software architectures and components. Visual abstraction helps in team communications and capable for envisioning the entire software system easily. System requirements are reflected consistently from designs to implementations with the use of UML.
5. *Verify software quality*. Quality must not be treated as afterthought or performed by separate group of people. It should always be ongoing process throughout the entire development process covering all activities and participants, measuring quantifiable metrics. It should be the subset of the entire requirements that is based on reliability, application performance and system performance.
6. *Control changes to software*. Software changes must be controlled by some decision framework that is traceable and monitored. There is need to establish secure workspace for each developer by having isolation to changes made in other workspace that eventually are consolidated into some common automation build ecosystem.

The advantages of RUP are; since RUP is a kind of iterative process this naturally improves communication between stakeholders with regular feedback, especially from customer. This results in deliveries that customer wants. There is a focus for resource planning during the *Inception* and *Elaboration* phase, therefore resources are much more effectively used.

The disadvantages of RUP are, the process might be too complicated to implemented because of RUP is not only delivered as knowledgebase but also associated software

for execution, this may increase overhead for some organizations. The complexity of the implementation might get the project out of hands because of focus needed for RUP project management aspects. Since additional software is part of the process, more expertise might be required.

f. Agile Development

Agile development is a software process framework that is based on the iterative and incremental development framework which promotes the practice for continuous delivery of software systems through cycles of iterations for refinements based on feedbacks from delivered software system; such practice encourages the use of sustainable software development resources and infrastructures for accommodating continuous evolution of the software product. This process model is highly adaptive to changing requirements or unexpected responses; such circumstances place high empowerment onto the collaborating parties for optimal agility and efficiency.

Agile development does not restrict the software development onto some fixed process semantics but rather promotes a sets of guidelines and methodologies for development team to adopt when required.

The Agile software development process was founded by a group of industry experts who outlined a set of high level driving policy to express the values brought by Agile software development process, these high level policy are known as the Agile Manifesto. The first manifesto is written as *Individuals and Interactions over Process and Tools*. The need for self-motivated individuals is far more superior than any advanced processes and technology because software engineers are the people driving these advanced static elements to produce top quality software system. The second manifesto is written as *Working Software over Comprehensive Documentation*. Ultimately it is the software product that gets delivered to the customer which creates business values for the customer, therefore good documentation can be a supplementary factor for customer satisfactions. The third manifesto is written as *Customer Collaboration over Contract Negotiation*. This means getting customer to commit into stages of software development cycles to get their early buy in of the product features in order to produce software that is more relevant to the market needs as this supersedes the closing sales of any potential customers. The fourth manifesto is

written as *Responding to Change over Following Plan*. This suggest that the survival of the fittest is not which is the strongest but the one which is adaptive to inevitable changes. The only constant is the real world is the attribute of ever-changing. Therefore the only way for software systems to stay relevant for a long time is to be able to adapt to the ever-changing real world needs. The four manifesto listed above, each can be broken down into two parts as delimited by the word "over"; the left and the right. The notion here is that while there is value for the item mentioned on the right, but the item on the left is being valued more. For example, with *Responding to Change over Following Plan*. The attribute of responding to change is more important than the ability of following a plan.

The Agile Manifesto described above lays the foundation policy for the twelve Agile principles for software development teams to follow as rules or guidelines governing the software development process.

Our highest priority is to satisfy the customer through early and continuous delivery - Continuous delivery yields more satisfying customer because each iterative release consist of new functions that add values to customer business based on customer feedback.

Welcome changing requirements, even late in development. Agile process harness change for the customer's competitive advantage - Embrace change since it is not evitable, more over the ability to change sharpens the competitive edge of the software product.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale - Shorter release cycles for each iteration, get earlier customer feedback and factor into next iteration planning and this keeps the attention of all stake holder at a healthier pace.

Business people and development must work together daily throughout the project - More horizontal collaboration means members from various business units and organizations come work together and contribute to the software product, this

effectively breaks down communication barriers and increase the accuracy of information.

1

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done - Empowerment seeks for self-motivated individuals who genuinely wants the software system to be successful, this eventually leads to a job getting done better and a more conducive working environment.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation - Use face-to-face conversation as the primary mode of communications, this mode of communication is more effective and keeps urgent issues at good grip.

Working software is the primary measure of progress - Progressive delivery is focused on working software as a measurement of progress, which each working software version delivered indicates a step closer to the project objective.

Agile processes promote sustainable development. The sponsors, developers and users should be able to maintain constant pace indefinitely - Use of sustainable development strategies to maintain a constant pace for all stakeholders, this is to maintain interest for all stakeholders and align to velocity of market requirements.

Continuous attention to technical excellence and good design enhances agility - Technical excellence focuses on good software development practice and high quality deliverables that would improve team agility.

Simplicity – the art of maximizing amount of work not done – is essential - Managing priority suggest that not to over commit to all requirements but tactically address them according to priorities, this will reduce the risk of getting into over-engineering for a problem.

The best architectures, requirements, and design emerge from self-organizing teams - Self-organizing team works better and produce higher quality deliverables because of self-commitments.

At regular intervals, the team reflect on how to become more effective, then tunes and adjust its behavior accordingly - Team retrospective is important to consolidate the strengths and weaknesses for improvements during the following iterations. Therefore for any software development teams that wishes to move into Agile, the above twelve principles can be used a guidance in carrying out the software development activities.

There are existing software development methodologies that are based on the Agile Manifesto and principles described above. Each of the methodologies are different such that each focuses on different aspects of Agile that are applicable depending on situations.

1. *Small, close-knit team of peers* and *Pair programming and no-owner code* emphasis on interactions between members.
2. *Periodic customer requirements meetings* and *High-level requirements statements* emphasis on collaboration between developments and customers to be more adaptive to changes.
3. *Code-centric* emphasizes on the delivery of a working software and puts documentation as secondary concern since modern technology allows documentations to be automated. For example, the use of Java Doc Annotations.
4. *High-level requirements statements only* and *Document as needed* emphasis on collaboration for the need of change.
5. *Refactor* emphasis on changing the code structure but maintaining functionality. This usually is done to accommodate modifications. As example, changing from component based architecture to service based architecture for more horizontal scalability.
6. *Pair-wise programming* and *Test-driven development* emphasis on collaboration for developers to have product code and test code produced in parallel fashion. This is to automate testing effort to ensure consistencies in functionality for every iteration without additional effort for rerunning test.

Extreme Programming is known to be a radical agile methodology that consist of three main activities that could run in somewhat parallel fashion. The first activity is to create a software product vision, this is usually customer requirements related that express the business values or features needed. The second activity is to create use

stories that are usually related to the software product design that includes architectures, designs and implementation details. This phase would include the development work required to produce some version of working software. The third activity is about producing automated acceptance test that could be ran over and over again for every iterations. This Agile method encourages pair programming and test driven development.

Scrum is another Agile method that is skewed more towards a project management methodology. In a scrum scenario, there are three main roles namely the product owner, scrum master and the development team. The purpose of having scrums is to track progression and unblock development barriers. It is usually executed with meetings for various discussions. The backlog grooming meeting is use to prioritize product features to be released for each iterations. The sprint planning meeting is use to get consensus from all stakeholders about the following iteration delivery objective. The daily scrum meeting is for development team to update progress, report problems and find solutions to progress. The sprint review is for team to demonstrate the delivered working software for feedbacks. The sprint retrospective is for team to reflect both strength and weaknesses to be more prepared for the upcoming iterations.

The apparent advantages from using Agile software development model are; greater customer satisfaction, because the fact that there is constant communication between developers and customers along with iterative and incremental working software, customer would get a better idea of the final product and requirements are more relevant. Much focus are concentrated for getting the working software out to end users, this indirectly means there are more effort spent on the software code that would yield better code quality.

Some of the disadvantages from using Agile development model are lack of documentation that would make handover or knowledge transfer difficult, especially within development team and support teams. Since requirements are defined in high level fashion, development team would require more experienced developers to be able to manifest those high level requirements into tangible software products. Inexperience members may suffer from the steep learning curve.

The details above described the various software process models from traditional to modern process model, each with its own unique features together with its own fair share of advantages and disadvantages. Therefore each process models find its way to a suitable software project that has could deliver the biggest impact. As examples, assuming that a small company wants a company website which comprises of just three static pages in order to make a small presence in the World Wide Web, such software project would fit nicely for the Waterfall model because the customer requirements are straight forward and the Web technologies needed are primitive and the expected outcome from all stakeholders are more consistent. Assuming the idea of coach sharing application like *Uber*. The idea of the software project was a new business concept in the market. Since the software product nature was new and not deterministic. This project ought to be more suited with Prototyping model because ideas were still high level and there was no existing business use cases as references. By using the prototyping approach, it would help the stakeholders to gather more insights into the additional requirements needed for the software application to eventually drive the business model and the prototype is also useful to be used as a show case evidence for investor funding. Given a project like the enterprise grade telecom OSS described in *Question 1*, such software system would be used by customer for a very long time to support their business operations and some even relied on these OSS for business success. Therefore these kind of projects are more suitable to adopt the iterative and incremental models, particularly RUP or Agile models because being an enterprise grade software requires high commitment from all stakeholders from development to customers and end users of customers. These big organizations often have survived in the market for a very long time and requires software that is able to be adaptive to market changes and at the same time still preserve its niche position in the market. Therefore features and requirements are often needed to be enhanced to stay relevant in the market.

Question 4

Below is the Activity Table provided from the TMA questionnaire,

Activity	Duration (Days)	Dependencies
A	20	None
B	13	A
C	25	None
D	31	B
E	18	C, D
F	10	E
G	16	None
H	18	F, G
I	23	D, H

Table 4.1

The details from *Table 4.1* are translated into a Precedence Network Diagram in this following section. The Precedence Network Diagram is assumed to have an arbitrary start and end point for completeness sake. Each activity is represented by a circular object with text indicating the respective activity followed by a number indicating the number of day(s) and delimited by a comma. As an example,

Figure 4.1

Figure 4.1 indicates an activity named X with a duration of 20 days.

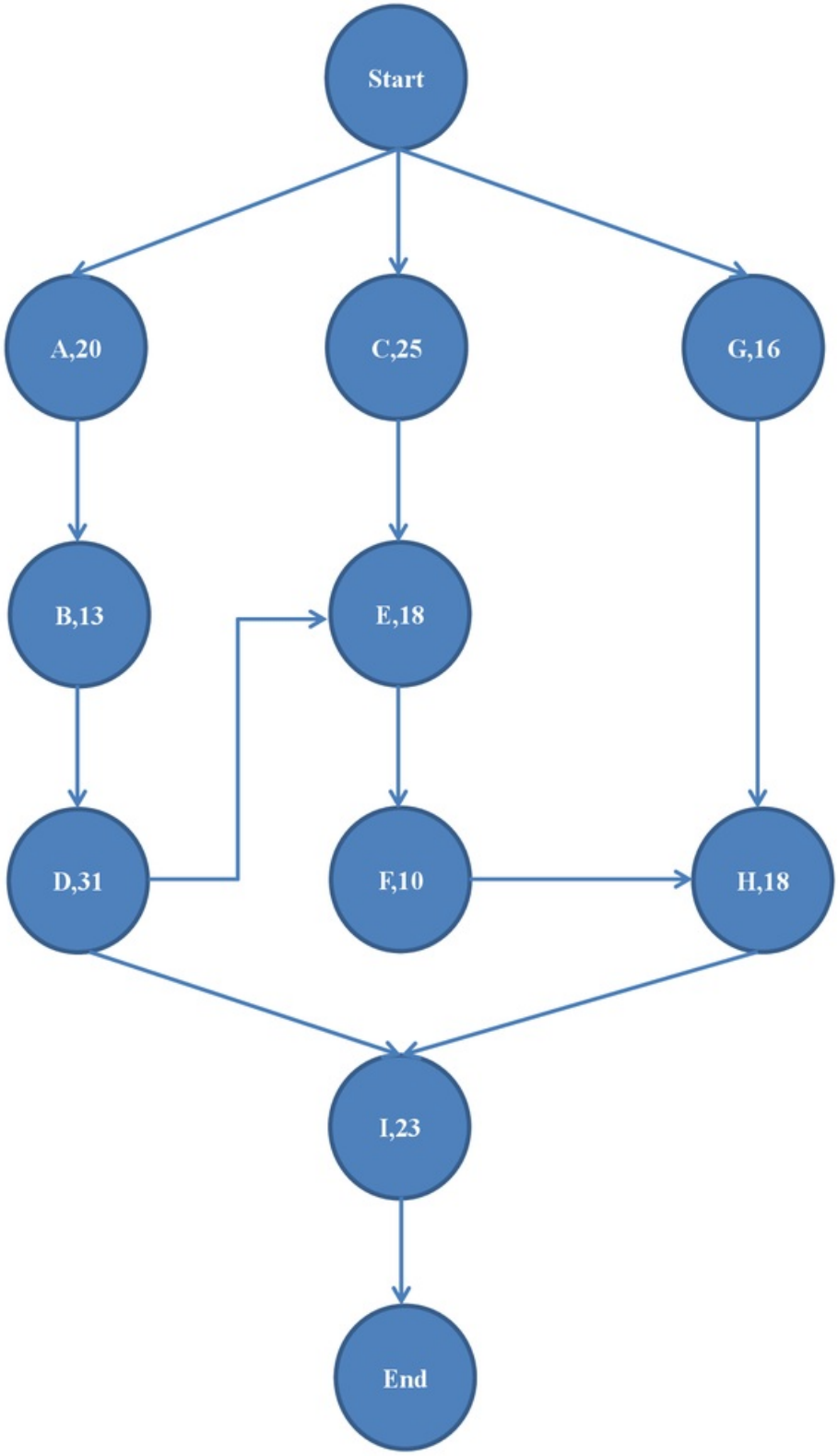

Figure 4.2

Figure 4.2 shows the Precedence Network Diagram based on the information found in *Table 4.1*. The diagram is presented in vertical instead of horizontal fashion due to the page orientation constraint. There are 4 paths identified to project completion as indicated below,

a. Path 1: Start → A → B → D → I → End
b. Path 2: Start → A → B → D → E → F → H → I → End
c. Path 3: Start → C → E → F → H → I → End
d. Path 4: Start → G → H → I → End

Accumulate the number of days according the respective activities on each path,

a. Path 1: 20 + 13 + 31 + 23 = 87
b. Path 2: 20 + 13 + 31 + 18 + 10 + 18 + 23 = 133
c. Path 3: 25 + 18 + 10 + 18 + 23 = 94
d. Path 4: 16 + 18 + 23 = 57

The results above show that Path 2 is the critical path because it uses the most duration. Therefore the estimated duration for project completion is 133 days.

A cross validation with an Activity Bar Chart,

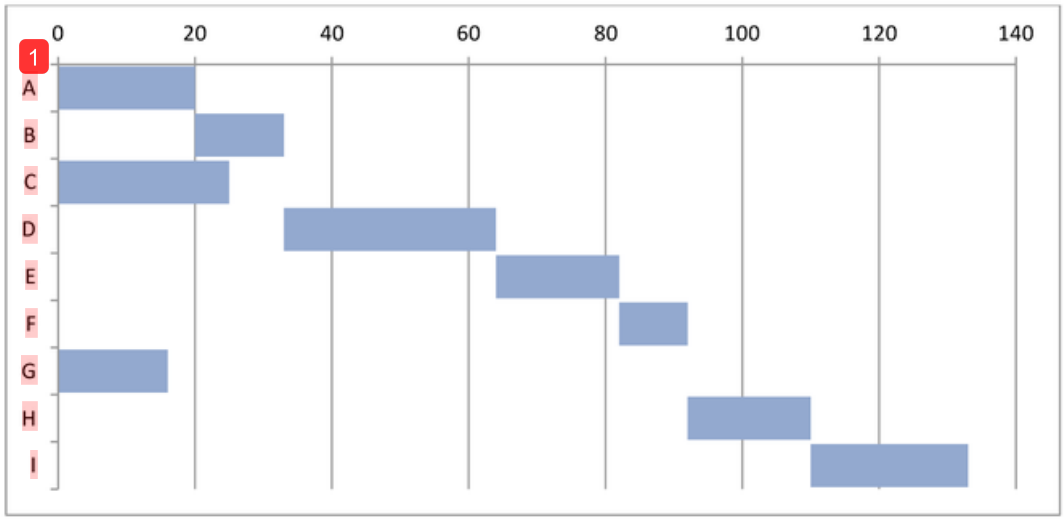

Chart 4.1

Chart 4.1 is an Activity Bar Chart based on the information from *Table 4.1*. The horizontal axis represents the duration with 20 days interval and the vertical axis represents the activity from *A* to *I*.

It is observed that the longest duration is hitting the section between 120 and 140 days and the bar situated at the same level as row *Activity I* indicates a fair approximate value of 133 days.

Reference

ISO 25000 Software Product Quality. 2015. ISO/IEC 25010.

http://iso25000.com/index.php/en/iso-25000-standards/iso-25010. (1 August 2016)

Jose P. Miguel, David Mauricio and Glen Rodriguez. November 2014. A Review of Software Quality Models for the Evaluation of Software Products. https://arxiv.org/ftp/arxiv/papers/1412/1412.2977.pdf. (2 August 2016)

Product Quality – ISO/IEC 25010. http://disciplinas.stoa.usp.br/pluginfile.php/294901/mod_resource/content/1/ISO%2025010%20-%20Quality%20Model.pdf. (3 August 2016)

Requirements Engineering and Quality Attributes. http://portal.ou.nl/documents/114964/2986739/IM0203_02.pdf. (4 August 2016)

Roelof Reitsma. 1 July 2011. Software has a new quality Standard: ISO 25010. http://ryreitsma.blogspot.my/2011/07/software-has-new-quality-model-iso.html. (5 August 2016)

Ajitesh Kumar. 25 March 2014. What's Needed to Get Your Code Quality Match ISO Standard 25010. http://vitalflux.com/checklist-code-quality-matches-iso-quality-standards-square/. (6 August 2016)

Ministry of Economy, Trade and Industry, Japan. March 2011. Investigative Report on Measure for System/Software Product Quality Requirement Definition and Evaluation. http://www.meti.go.jp/policy/it_policy/softseibi/metrics/20110324product_metrics2010_en.pdf. (7 August 2016)

BBC News. 27 October 1999. UK: Wales Holidaymakers may Foot Bill for Passport Chaos. http://news.bbc.co.uk/2/hi/uk_news/wales/487351.stm. (7 August 2016)

Douglas N. Arnold. 23 August 2000. The Patriot Missile Failure. https://www.ima.umn.edu/~arnold/disasters/patriot.html. (8 August 2016)

Robert Skeel. 18 January 2011. Roundoff Error and The Patriot Missile. https://www.ima.umn.edu/~arnold/disasters/Patriot-dharan-skeel-siam.pdf. (8 August 2016)

Tom Morgan and Jason Roberts. 2002. An Analysis of The Patriot Missile System. http://seeri.etsu.edu/SECodeCases/ethicsC/PatriotMissile.htm. (8 August 2016)

SDLC – Waterfall Model. http://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm. (8 August 2016)

The Software Experts. 29 May 2014. The Waterfall Model. https://www.youtube.com/watch?v=0NAsk0noT_U. (9 August 2016)

ISTQB Exam Certification. What is Waterfall model – advantages, disadvantages and when to use it? http://istqbexamcertification.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/. (10 August 2016)

SDLC – Spiral Model. http://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm. (11 August 2016)

Software Testing Help. Spiral Model – What is SDLC Spiral Model? http://www.softwaretestinghelp.com/spiral-model-what-is-sdlc-spiral-model/. (12 August 2016)

Naveen. What is Spiral Model in Software Testing and What are advantages and disadvantages of Spiral Model. http://testingfreak.com/spiral-model-software-testing-advantages-disadvantages-spiral-model/ (13 August 2016)

Dimitry Gurendo. XB Software Blog. 26 October 2015. http://xbsoftware.com/blog/software-development-life-cycle-spiral-model/. (14 August 2016)

ISTQB Exam Certification. What is Incremental Model – Advantages, Disadvantages and When to Use it? http://istqbexamcertification.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/ (15 August 2016)

SDLC – Iterative Model. http://www.tutorialspoint.com/sdlc/sdlc_iterative_model.htm (16 August 2016)

Technology Conversations. Software Development Models: Iterative and Incremental Development. https://technologyconversations.com/2014/01/21/software-development-models-iterative-and-incremental-development/ (17 August 2016)

Javed Nehal. Intelligence on Tap. 24 November 2009. http://www.iotap.com/blog/entryid/124/advantages-disadvantage-of-prototyping-process-model. (18 August 2016)

Penna Sparrow. November 2011. Prototype Model: Advantages and Disadvantages. http://www.ianswer4u.com/2011/11/prototype-model-advantages-and.html#axzz4IS9N1Ane. (19 August 2016)

Shuisharma. Prototype Model. 22 May 2012. http://www.slideshare.net/shuisharma/prototype-model. (20 August 2016)

Christensson, Per. RUP Definition. 2006. http://techterms.com/definition/rup. (21 August 2016)

Rational. 1998. Rational Unified Process – Best Practices for Software Development Teams. https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf. (22 August 2016)

Omkar Dash. 5 September 2009. Rational Unified Process. http://www.slideshare.net/OmkarDash/rational-unified-process (23 August 2016)

Kelly Waters. All About Agile. 10 February 2007. What is Agile? (10 Key Principles of Agile) http://www.allaboutagile.com/what-is-agile-10-key-principles/. (24 August 2016)

2008. The Agile Movement. http://agilemethodology.org/. (25 August 2016)

ISTQB Exam Certification. What is Agile Model – Advantages, Disadvantages and When to Use It? http://istqbexamcertification.com/what-is-agile-model-advantages-disadvantages-and-when-to-use-it/ (26 August 2016)

Scott W. Ambler. Ambysoft. The Agile System Development Life Cycle(SDLC). http://www.ambysoft.com/essays/agileLifecycle.html. (27 August 2016)

StackExchange. December 2010. What is Agile different from XP? http://programmers.stackexchange.com/questions/17843/how-is-agile-different-from-xp. (28 August 2016)

James D. McCaffrey. 11 April 2006. Agile Programming vs. Extreme Programming. https://jamesmccaffrey.wordpress.com/2006/04/11/agile-programming-vs-extreme-programming/. (29 August 2016)

Agile Alliance. 12 Principles Behind the Agile Manifesto. https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/. (29 August 2016)

Jamsheer K. 19 August 2016. 12 Best Software Development Methodologies with Pros and Cons. http://acodez.in/12-best-software-development-methodologies-pros-cons/ (30 August 2016)

Zoya Abbas. 7 October 2012. RUP Model. http://www.slideshare.net/zoimustafa/rup-model. (31 August 2016)

TCC235-05-Software-Engineering-TMA1-LaiYeongWei-031150110-v0.8

ORIGINALITY REPORT

%1	%1	%2	%1
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	de.scribd.com Internet Source	%1

EXCLUDE QUOTES	ON	EXCLUDE MATCHES	< 1%
EXCLUDE BIBLIOGRAPHY	ON		