ISBN 978-81-936279-4-5
9th International Conference on Science, Engineering, Technology and Healthcare (SETH-18)
Antalya (Turkey) May 3-4, 2018

Review of Software Engineering Principles and Practices in Modern Day Development

Amjed Abbas Ahmed

[1] Imam Al-Kadhum (a) University College for Islamic Sciences, Baghdad, Iraq

***Abstract*:** *Software Engineering principles and practices includes requirements generation, the logical design of a proposed system, its physical design as well as testing, deployment and management. Core principles and practices over time have been tested especially with new and emerging technologies of design and development to further see if it can still be applicable. Recent surveys and trends shows that software practices and principles are changing and overtime may lose its original core which is seen as basic building principles which has kept it and made it remain relevant across ages. New emerging practices does not only make software principles easy but also make it more effective especially with regards to huge projects that requirement fast track evaluation in areas considered delicate to the system to be developed. This paper critically examines the core of software engineering principles and practices as a comparison to new emerging practices to further evaluate its relevance and effectiveness toward system development which is considered the heart of information system development. This forms the basis for which most organisations strategically integrate their business to meet competitive market needs.*

Keywords: *Software Engineering, Modern Day Development, Review.*

1. Introduction

Software Engineering, as described by Ali (2014) can be said to be "is concerned with developing and maintaining software systems that behave reliably and efficiently, are affordable to develop and maintain, and satisfy all the requirements that customers have defined for them". Critically, this explains different things but emphasis is laid on specific processes such as development, design, maintenance, reliability, efficiency, affordability as well as user satisfaction which is considered a derivative from the use of the software that has been developed and implemented. Surveys have showed over time that for a software to be successful, some basic software engineering practices must be put into play to act as a guide or pathway toward successful development and integration. In reality, this is categorised into two parts: Individual Practices and Team Practices. The individual practices refers to the practices of an individual develop while the team practices refers to the practices performed or carried out by a project or developmental team. The need to critically analyse software engineering practices is evident and to further compare with emerging technologies to see if these practices are still relevant or have changed due to evolution. The essence of these practices comes into play.

2. Essence Of Software Engineering Practices

The essence of software engineering practices cannot be over emphasized but more important it drives the efficiency and effectives of the implementation of the practices. These essence are mainly four (4) which are:

2.1. Understanding the Problem

Understanding the problem is very important as it helps communication as well as analysis. It also gives the developer room to ask questions such as: who has the stake in the solution of the problem that must have been identified? Are there unknowns and what are they? What is required to solve the identified problem? Can the

problem be broken into smaller bits for further understanding? Can the problem be represented graphically? Can analysis model be created for the problem? Providing answer to all these numerous questions would have help the developer in understanding what the problem is and can proceed to into planning a solution.

2.2. Planning a Solution

These can be done or achieved in various forms. It also involves activities such as planning, system modeling also known as the logical design as well as software design which is the physical design of the software. It also gives the developer chance to ask critical question that would help in planning. Questions such as: have you seen similar problems before and are there solutions to them? Are the solution elements reusable? Can the identified problems be broken into small parts and find solutions to each small part? This helps a lot in planning as the planning process must encompass all aspect involved in software development.

2.3. Carry out the Plan

To carry out a plan, a developer must definitely have a pathway to follow. Carrying out the plan may involve physical development, code generation and simultaneous debugging. It also gives the developer to analyse and evaluate if the solution in form of software conforms to the plan? Is the source code traceable back to the design? Is each component of the solution correct?

2.4. Examine the Results

This explains the reason why test are carried out on developed software and at same time gives room for the developer or developing team to ask as lot of question that would help in the software evaluation. These might include asking questions such as: what kind of test plan is most suitable to test the system? How will the test plan be implemented? Has the software system developed been able to validate stakeholder's requirements? Answering these questions would help the developer or developing team to focus more on the results as a standard for evaluation at this stage.

The essence is as important as the practices itself. This leads to where a critical examination of software practices needs to be evaluated. For that to be done, there is a need to examine the types of practices.

3. Types Of Practices

There are basically 5 main types of practices as far as software engineering is concerned. These are

- **Communications Practices**

Effective communication among developmental team, customers and clients as well as stakeholders and project managers can be challenging but helpful in bring developmental team on same page. The more the developmental team is engaged in communication, the better it is to understand team goals as well as ways to achieve it. Principles of good communication practices in software engineering includes: listening, preparation before communication, facilitating communication, notes taking and documentations.

- **Planning Practices**

The communication activity helps a software team to define its overall goals and objectives. Understanding goals and objectives is one thing, defining the plans to get there is another thing that should be given high consideration. Planning practices is made up of a list of both management as well as technical practices which helps developmental team to define pathway. Planning practices includes:

- Understanding project scope
- Customer involvement in planning
- Iterative planning
- Estimation based on what is known
- Risk consideration during planning

- Adjust and define plan
- Define quality assurance
- Define change accommodation
- Track plan frequently

- **Modelling Practices**

This practices focuses more on models as models are created to gain better understanding of actual entity to be built. It is easier to build an identical model especially when the entity is a physical thing. This can be smaller in shape. In case of software engineering, model must be capable of representing information the software transforms, the architecture and functions that enable the transformation to occur, the features that user's desire, and the behaviour of the system as the transformation is taking place. Two different class models are created: Analysis and Design. Analysis model represents customer requirements which is shown by depicting a software as information, functional and behavioural domain. The design model represents the characteristics that helps effective development. Practices here includes:

- Representing and understanding problem
- Defining software functions
- Representing software behaviour
- Migration from requirements to implementation
- Design should be traceable to analysis model
- Architecture consideration
- Data design
- Interface design
- Components should be functionally independent
- Design representation should be easy to understand
- Iterative design

- **Construction Practices**

This is made up of or encompasses a set of coding and testing tasks that also leads operational software ready to be delivered to clients and end users. Here emphasis is laid on coding and maybe: direct creation of programming language source code, automatic generation of source code using intermediate design-like representation and automatic generation of executable code using fourth generation programming languages. Construction practices are also defined in 4 main principles which are preparation, coding, validation and testing. Preparation may include: understanding problem trying to be solved, understanding basic design principles as well as concepts, selecting a suitable programming language and creating a set of unit test that will be applied. Coding principles may include: use of algorithm, selecting data structure, understanding software architecture, creating nested loos, selecting meaningful variables names, writing codes that is self-documenting and creating visual layout that helps understanding. Validation principles may include: building architectural infrastructure, building software components, unit testing for individual components of the system and integrating completed components into architectural infrastructure. Testing principles may include: testing should be traceable to customer requirements, tests should be planned long before testing and test should begin in small and progress towards large

- **Deployment Practices**

Deployment practices is made of three main actions: delivery, support and feedback. Deployment is a process that doesn't happen once but a couple of time especially as the software is nearing completion. Each deployment provides end-users with operational software increment. Deployment principles includes

- Managing customer expectations
- Testing assembled package before delivery
- Establishing support before software is delivered
- Having end-user instructional materials

The types of practices further gives an insight into what software engineering practices are and how it affect software development generally.

4. Software Engineering Practices

There are a lot of software development practices which could be individually motivated or motivated by a developmental team. In all, there are basics things that make up the fundamentals of software practices. Since software practices involves development of software for private and public use, some considerations forms the basis for practices. Six considerations forms core software practices which are: development, requirements, architecture, modelling, quality and change. The core software practices are:

4.1. Iterative Development

Iterative development is one that allows back and forth in between development. It also allows successive series of releases of increasing completeness. Each iteration is focused on the following: identifying, defining and analysing some set of requirements, and designing, building and testing software based on the understanding of those requirements. Iterative develop is achieved using some specific system methodologies. System methodologies that support iterative development are Rational Unified Processing, Water Development

4.2. Requirement Management

Managing requirements can have a devastating effect on a project if not properly handled. Most project problems occurs from poor requirements management, incorrect definition of requirement from the start of the project and poor requirements management throughout the development lifecycle. The requirements for the software are key input to testing. Proper management of relationship between requirements and the test delivered from the requirements. Establishment of traceable relationship between those elements. This helps to be able to do the following

- Assess the project impact of a change in a requirement
- Assess the impact of a failure of a test on requirements
- Manage the scope of the project
- Verify that all requirements of the system are fulfilled by the implementation
- Verify that the application does only what it was intended to do
- Manage change

User requirements can be managed properly especially during design process using specific design processes. These include:

- USE Case Diagram
- Sequence Diagram
- Context Diagram
- Data Flow Diagram

4.3. Architecture Component Usage

Component architecture usually requires development of product that give highest return on investment. This also goes hand in hand with considerations such as quality, cost and schedule. Focus is on the architecture and most importantly, the software industry has to be really understood in and out. Architecture is an aspect of

design that involve making decisions on how a system will be developed but not all about design. The main property of an architecture is resilient which is described as flexibility in the face of change. It also meets current and future requirements, improves extensibility, enable reuse and encapsulates system dependencies. Its purpose is

- Basis for reuse: this involves both component and architecture reuse
- Basis for project management: this involves planning, staffing as well as delivery
- Intellectual control: this also involves managing complexity in the development as well as maintaining integrity.

4.4. Visual Modelling

Visual modelling involves developing a model and a model, as described by Rational (2002) says "*A model is a simplification of reality that provides a complete description of a system from a particular perspective*". Models are built in order to better understand the system to be modelled, this became very important and handily when dealing with very complex system because such system cannot be comprehended entirely. Among the reasons why visual modelling is important are:

- To help manage complexity which does the following
  - To capture both structure and behaviour
  - To show how system elements fit together
  - To hide or expose details as appropriate
- To keep design and implementation consistent
- To promote unambiguous communication

Modelling is important because it serves as help to the development team to visualize, specify, construct, and document the structure and behaviour of a system's architecture. As described by Rational (2002) saying "Using a standard modeling language such as the UML (the Unified Modeling Language), different members of the development team can communicate their decisions unambiguously to one another"

Visual modelling using UML diagrams can be done using

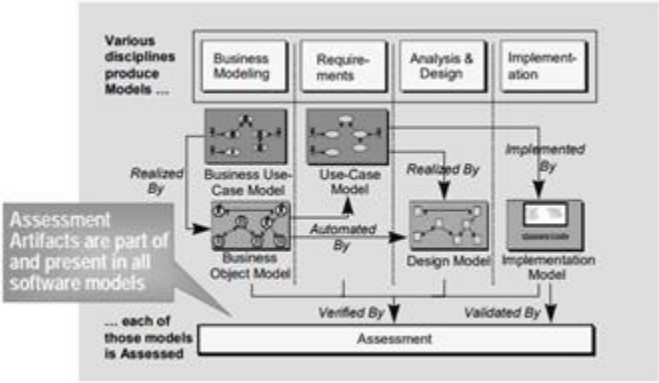
- Use-case diagram
- Class diagram
- Collaborations diagram
- Sequence diagram
- Component diagram
- Statechart diagram

4.5. Quality Verification

This involves verifying the quality of the software that has been developed by testing every unit of the software, an important aspect of Software Quality Process. Though testing can be difficult because of the numerous processes involves, it is an efficient way in quality verification. Continuous quality verification can be achieved by

- Building
- Test and Evaluate
- Achieve mission

In iterative development, assessment activities are an integral part of the effort in each iteration: they are needed to provide objective proof that the goals of the iteration have been met. Continuous verification quality is given below



The UML can be used to produce a number of models that represent various perspectives which are:

- The Business Model is a model of what the business processes are and of the business environment. It is primarily used to gain a better understanding of the software requirements in the business context.
- The Use-Case Model is a model of the value the system represents to the external users of the system environment. It describes the "external services" that the system provides.
- The Design Model is a model that describes how the software will "realize" the services described in the use cases. It serves as a conceptual model (or abstraction) of the implementation model and its source code.
- The Implementation Model represents the physical software elements and the implementation subsystems that contain them

4.6. Change Management

Change management involves a lot managing change at every step of the software development but this depends on what you want to control which can be

- Changes to enable iterative development
- Automated integration/build management

Aspect of Content Management System

- **Change Request Management (CRM)**: addresses the organizational infrastructure required to assess the cost and schedule impacts of a requested change to the existing product
- **Configuration Status Reporting**: is used to describe the "state" of the product based on the type, number, rate and severity of defects found, and fixed, during the course of product development
- **Configuration Management (CM)**: describes the product structure and identifies its constituent configuration items that are treated as single versionable entities in the configuration management process
- **Change Tracking**: describes what is done to components for what reason and at what time
- **Version Selection**: ensures that the right versions of configuration items are selected for change or implementation
- **Software Manufacture**: covers the need to automate the steps to compile, test and package software for distribution

5. Modern Day Development

Software engineering in recent times have moved to the state where a lot of tools, platforms and frameworks have been developed in recent times that makes software development easy, flexible and fast. Frameworks such as Bootstrap, and Mobirise have made it easy to even develop mobile responsive software and system that responds to devices that is been used to access it. Also in recent times, Artificial Intelligence capabilities has also introduced into software engineering that makes system development automation and simulation fast.

6. Conclusion

Software practices has a pattern which makes practices constant over a long time. However, software development has evolved over time with the integration of frameworks, tools, platforms and also Artificial Intelligence capability, software development has improved over time but still has its core practices intact. An examination and evaluation shows that more evolutions will be made in software engineering that will enhance software development. Such evolutions is Mixed and Augmented reality but core software practices will remain same for a very long time

7. References

[1] Adams. D (2016) Designing Software for Mixed Reality Requires a Massive Shift in Thinking. Available at: https://mixed.reality.news/news/designing-software-for-mixed-reality-requires-massive-shift-thinking-0171663/. Accessed on the 19th of April, 2018

[2] Adams. D (2016) If You're Curious About Creating Software for Augmented & Mixed Reality, Start Here. Available at: https://next.reality.news/news/if-youre-curious-about-creating-software-for-augmented-mixed-reality-start-here-0172153/. Accessed on the 22nd of April, 2018

[3] Brian. P (2015) Virtual reality in software engineering: affordances, applications, and challenges. Available at: https://dl.acm.org/citation.cfm?id=2819098. Accessed on the 19th of April, 2018

[4] Ciklum (2016) THE EFFECTS OF VIRTUAL REALITY ON SOFTWARE DEVELOPMENT. Available at: https://www.ciklum.com/white-papers/the-effects-of-virtual-reality-on-software-development/. Accessed on the 22nd of April, 2018

[5] David. P, Jared. B (1999) An Introduction to Software Engineering Practices Using Model-Based Verification. Available at: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=13371. Accessed on the 23rd of April, 2018

[6] Michael. F (2017) 30 best practices for software development and testing. Available at: https://opensource.com/article/17/5/30-best-practices-software-development-and-testing. Accessed on the 22nd of April, 2018

[7] Nawab. S (2014) Software Engineering Practices. Available at: https://www.slideshare.net/AkbarAli45/software-engineering-practice. Accessed on the 15th of April, 2018

[8] Odeh. L (2016) Software Engineering in Practice. Available at: https://www.dcs.bbk.ac.uk/study/modules/software-engineering-in-practice-compulsory-unless-foc-is-taken/. Accessed on the 17th of April, 2018

[9] Rational (2002) Software Engineering Practices: Principles of Software Testing for Testers. Available at: http://sceweb.sce.uhcl.edu/helm/ROLE-Tester/myfiles/Module2/03_TST170_S01_Engineering.pdf. Accessed on the 16th of April, 2018