About Software Engineering Frameworks and Methodologies

Ernest Mnkandla, *Member, IEEE*

***Abstract*—Defining software engineering can be a rather challenging task depending on the purpose of the definition and the intended beneficiaries. Talking about frameworks and methodologies in software engineering can be even more complex under whatever circumstances due to the many different ways in which the issues concerning frameworks, processes, approaches and methodologies have been handled. This paper explores software engineering frameworks and methodologies and related topics. The purpose of this study is to add further understanding in the area of software engineering frameworks and methodologies. Data from published literature is analysed to establish a position on the status of software engineering frameworks and methodologies. The fundamental questions to consider here are; what kind of benefits can be derived from a better understanding of these issues? Would a better understanding of frameworks and methodologies improve the way we design, develop, implement, maintain and use Information Systems?**

***Index Terms*—Software engineering, frameworks, methodologies, models, software development.**

I. Introduction

THE first born of computing technology was computer hardware, which was initially intended to automate a few routine human tasks in technical environments. Computing hardware multiplied the processing speeds to levels that were virtually impossible for humans. Following hardware development the tasks for humans became softer; for example sorting job cards, assigning jobs, collecting the data for processing, scheduling the jobs for the machine to process etcetera. It soon became clear that these machines would need some logic in order to increase their level of automation. The scientists and mathematicians were key to the new developments that followed this era. A new field of computing software was born. Writing programs for computers was so soft that it was initially assumed to be a job for women. However, the initial demand for software increased at such a rate that it became obvious that a more formalized approach to the development of software had become inevitable. Surprisingly even in the twenty–first century the demand for more software still outstrips the supply. During this era in the history of computing, software development problems led to what became known as the software crisis. The following references give more detail on the software crisis [1, 2, 3, 4, 5, 6, 7, 8, and 9]. The software crisis not only refers to problems associated with approaches to developing software but also includes the broader aspects of how to maintain software.

During this era of rapid growth in software demand the pioneers began to consider similarities between designing software systems and designing engineering systems. Based on this line of thought ideas were born that led to what is called software engineering today. So the fundamental concepts of software engineering were based on the original engineering paradigm which assumes that design of any engineering system follows sequential thinking from identifying a problem that requires an engineering solution to developing the specification which then informs the design leading to the execution of the design into a full product etcetera. However, as the software engineering profession grew it became clear that developing software is a craft involving a lot of abstraction, lack of visibility and does not lead to a physical product. All these attributes of software engineering make it a very complex field that requires more human input than other engineering fields.

The limitations of the classical software engineering paradigm are rooted in a focus on sequential thinking which focuses on the activities presumed to be necessary to produce a result and assumes that one has nothing to learn so that each activity can be completed in sequence [10 and 11]. To try and solve some of these problems many methodologies emerged from the early 1970s and continue to emerge up to the present moment.

The proliferation of systems development methodologies led to confusion and difficulty in objectively selecting specific methodologies for specific uses, [12, 13, 14, and 15]. Cockburn [16] classifies software developers into three levels of practice. The three levels describe the stages of learning for a software development practitioner. ""A level one" person is just starting and would not be able to learn many methodologies at once. This is a stage where people look for one methodology that works and they learn it and follow it and expect it to work always. "A level two" practitioner is at a stage where he or she realizes that the technique will not serve in all situations so he or she looks for rules about when the procedure breaks. At level three, developers do not pay much attention to the methodology formula. They pay attention to what is happening on the project and make process adjustments as they work. They understand the desired end effect and apply whatever works to get to that end" [16]. The three levels of practice

.Ernest Mnkandla is with the Department of Business Information Technology, University of Johannesburg, Auckland Park 2006 Johannesburg, 2006, South Africa, phone. +27 11 5591217, fax: +27 11 559 1239. Email: emnkandla@uj.ac.za

Manuscript received April. This work is supported in part by the Faculty of Management, University of Johannesburg.

978-1-4244-3919-5/09/$25.00 ©2009 IEEE

therefore add to the subjectivity of methodology selection rather than providing a solution. The complexities surrounding methodology selection led some to call it a 'methodology jungle'.

It is useful to clarify the meaning of 'methodology jungle' in this paper. In the systems development methodologies context the term methodology jungle refers to the difficulty of selecting the appropriate methodology for a given project.

With so many methodologies to choose from abstracting to a higher level in order to simplify these methodology issues became inevitable, hence frameworks were developed to wrap-up the methodologies and give a more generic understanding of systems design issues. At an even more abstract level we find approaches which spell out the general directions or flow of concepts. While the frameworks give the skeleton of systems design, the approaches provide more of a conceptual image or architectural bias.

The rest of this paper starts by providing some background to approaches, frameworks and methodologies, and then goes into details of some of the frameworks and methodologies found in literature. These frameworks are then analyzed in a unique way that reveals all components to improve understanding of approaches, frameworks and methodologies. A discussion then follows which gives some recommendations on the implications of frameworks, methodologies and models to the future of systems design and maintenance.

II. BACKGROUND

This section provides a detailed background to the concepts and issues surrounding software engineering approaches, frameworks and methodologies. A scan through software engineering literature shows that new approaches, frameworks and methodologies continue to emerge as new software technologies unfold. An analysis of the frameworks and methodologies reveals some similar patterns that originate from historical Information Systems themes. In order to provide a comprehensive understanding of the relationships and differences between approaches, frameworks and methodologies it is important to start by defining them in accordance with the most common perspectives. For those who think there is no operational difference among these concepts the definitions provided here will reveal new levels of understanding. To those who know there is a difference but are not clear of the major differences new knowledge will be acquired.

A. Defining software engineering frameworks

A framework generally provides a skeletal abstraction of a solution to a number of problems that have some similarities. A framework will generally outline the steps or phases that must be followed in implementing a solution without getting into the details of what activities are done in each phase.

Where there is confusion you may find a framework and a methodology combined into a single thing. In such cases the architectural overview to designing solutions suffers especially when dealing complex problems. The practical realities of implementing software development projects demand use of methodologies that can be easily tuned to the different and changing business environments without affecting the architecture of the system. Hence if the methodology is entangled into the framework you end up with a monolithic structure instead of a more desirable modular or layered structure.

B. Defining software engineering methodologies

In general software engineering involves the design, development, maintenance and documentation of software systems. Since the establishment of software engineering as a discipline more than thirty–five years ago, there have been different opinions about the exact definition of software engineering. These are some of the definitions found in various literature:

1) The Software Engineering Institute at Carnegie Mellon University in their report on Undergraduate Software Engineering Education gives the following definition:
Software engineering is that form of engineering that applies the principles of computer science and mathematics to achieving cost–effective solutions to software problems, [17].

2) The IEEE Computer Society in a joint effort with the ACM agreed on the following definition:
The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software, [18].

3) A definition that combines these two definitions is given by Pfleeger and Atlee [19] as follows:
Software engineering can be viewed as an approach that combines computer science (which is made up of theories and computer functions) and the customer (who provides the problem to be solved) to develop a set of tools and techniques to solve problems.

There are a few other views on what the definition for software engineering should or should not include. The definitions given here provide a more balanced view in accordance with the purposes of this research.

The introduction of software engineering as an engineering discipline has with certainty led to formal methodologies for developing software applications and has encouraged logical thinking in programming. However, software engineering has not met levels of predictability that classical engineering disciplines have achieved. This challenge contributes to the continued criticism about calling software engineering an engineering discipline instead of a branch of computer science or a craft [20].

Software engineering methodologies can therefore be defined a group of methodologies used in the development of applications. Methodologies will give details of what should be done in each phase of the software development process. You will notice that the methodologies do not necessarily specify how things should be done. That level of detail is usually left to the organization to tune the methodology to its environment by for example developing templates, and other documents that spell out how things should be done.

Sometimes people mix the meaning of methodology and framework to an extent where you may find a methodology being defined as a framework for implementing certain things. The major difference between a methodology and a framework is that a framework should be used at a more abstract level which means you will need a methodology or more in order to implement a framework. An example that comes to mind is a project management framework such as PMBOK which allows users to apply their own methodologies to implement the process in each knowledge area.

C. *Defining software engineering approaches*

You will often find the word 'approach' used interchangeably with methodology, framework or even process. An approach is simply a way things are done, in this context it is a culture that is followed in order to developing software systems. The word 'approach' can therefore be used to refer to any of the more technical systems design concepts. Its use should remain general without replacing the more technical terms.

D. *Defining software engineering processes*

Processes refer to the sequence of activities that must be followed usually in a given order in order to attain certain results. We usually find or define processes at methodology level where phases specifying tasks and their corresponding outputs are specified. There is therefore no need to give detailed processes at a framework level. In some software development literature the words 'processes' and 'life-cycles' are used interchangeably [21].

E. *Defining software engineering models*

A model is a design tool that is used to simulate a proposed solution or design. A process model would therefore be simulating the software engineering process for developing software. Modeling entails determining how the process will transform a given input into an intended output. The simulation of a process according to Pfleeger and Atlee [19] can be done in two ways; static –meaning that the inputs are changed to outputs, and dynamic – meaning that the movement from input to output passes through some intermediate stages.

Since a model can be used at the framework or methodology level it is possible to have a model as a framework for example the CMMI is a framework for measuring the maturity and/or capability level of an organization's software development work. It is also possible to have models that are more technical, looking at each phase of software development for example at the requirements phase there are models for requirements elicitation etcetera. Hence the methodology need not specify a model at that level so that organizations can apply the models that may be suitable to their situations. In the context of the software development processes you will notice that there is a general tendency to call methodologies models. However looking at the definition of a methodology and comparing it to the definition of a model given here you will find that what are normally called process models in most software engineering literature such as in [4, 5, 19, 22] are in fact software engineering methodologies. The problem with mixing these guiding principles for software design is that we tend to complicate design unnecessarily.

III. TYPES OF FRAMEWORKS AND METHODOLOGIES

A summary of some of the most commonly used frameworks, methodologies and models as found in published literature is given in this section. The lists shown in tables 1 to 3 are not exhaustive but have considered what appears to be the most commonly used software engineering frameworks, methodologies and models. There are however a few cases that may be viewed by others differently as they may be classified in a different way. This classification is based on an analysis of how the frameworks, methodologies and models are actually used not on the way they are represented in literature. This perspective of viewing frameworks, methodologies and models in software engineering should improve our understanding of the way software systems are designed implemented and maintained.

TABLE 1
MOST COMMONLY USED SOFTWARE ENGINEERING FRAMEWORKS

Framework	**Full / Partial**	**Focus area**	**Year of origin**
Unified Process	Full	Development Life cycle	1988 [24]
Agile	Full	Development Life cycle	2001
Lean Software Development	Partial	Development Life cycle management	2001
Adaptive Systems Development	Partial	Development Life cycle management	1st in 1974 [25], Mid 90s
Rational Unified Process	Full	Development Life cycle	1998 [24]
*PMBOK	Full	Project Life cycle	1987
*PRINCE2	Full	Project Life cycle	1989
Microsoft Solutions Framework	Full	Development Life cycle	1993

- Full means that it is a full framework covering most of the software engineering activities.
- Partial means that the framework only covers part of the software engineering activities.
- Year of origin means when the framework is believed to have come into use.
- * Means these are considered frameworks when used in software development, but in fact are project management methodologies.

A list of less popularly used frameworks, methodologies and models such as: Open Unified Process, Essential Unified Process, Agile Unified Process, Spiral method, V Model etcetera has been omitted as it would not add much value this work.

TABLE 2
MOST COMMONLY USED SOFTWARE ENGINEERING METHODOLOGIES

Methodology	Full / Partial	Focus area	Year of origin
Waterfall	Full	Development Life cycle	1970 [5]
Rapid Application Development	Full	Development Life cycle	1991
Incremental	Full	Development Life cycle	1971
Extreme Programming	Partial	Development activities	Mid 90s
Dynamic Systems Development Method	Full	Development Life cycle	1990s
Scrum	Full	Development Life cycle management	1993
Enterprise Unified Process	Full	Development Life cycle	2004 [24]
Crystal	Full	Development Life cycle	Early 1990s
Catalyst	Partial	Development Life cycle	2003

- See legend in Table 1.

TABLE 3
MOST COMMONLY USED SOFTWARE ENGINEERING MODELS

Model	Full / Partial	Focus area	Year of origin
CMMI	Full	Development Life cycle & process management	2002
Agile Modeling	Full	Development Life cycle	2003
Feature Driven Development	Partial	Parts of the Development Life cycle	1997
Component Based Model	Partial	Development Life cycle	1986, or 1990

- See legend in Table 1.

IV. DISCUSSION

Design, implementation and maintenance of software systems is usually complicated by the ways that we follow in trying to reach the bottom line which is delivering quality products on time, at reasonable cost, etcetera. Extremism in following methodologies and other formalized systems of developing software has also led to reduced innovation and killing of the love for simple engineering of software systems. It is these kinds of issues that have led to the emergence of new ideas and the new ideas have furthermore led to the birth of new frameworks, methodologies and models. We continue to have more and more new brands of frameworks, methodologies and models because software development is a thing of the mind and people will always find comfort in different environments. It is therefore the intention of this paper to bring understanding of software engineering frameworks, methodologies and models so that as we continue our voyage of developing more and more complex systems in these complex business environments we may be effective and efficient in the use of our knowledge.

V. CONCLUSION

This paper discussed software engineering frameworks, methodologies and models. The analysis followed here revealed the similarities and differences between these ways of designing software solutions. A framework at an abstract level simply says that the work or project should be done in stages or phases but will usually not specify what those phases are. We can therefore say that the framework deals with providing guidelines of the way things should be done. While methodologies on the other hand give more specific guidelines of what should be done. Methodologies go into the phases and provide detail of what those phases are and corresponding activities in each phase. Frameworks, methodologies and models are not the same thing to those who are interested in maximizing simplicity in system design, implementation and maintenance.

The analysis carried out in this paper reveals two most important concepts about frameworks, methodologies and models; the focus area and the full/partial parameter. Sound knowledge of these two concepts helps practitioners to know how the framework, methodology or model should be applied. For example if it is a full framework, methodology, or model it should have the ability to support the complete engineering process. While a partial framework, methodology or model would need to be used in conjunction with other complementary frameworks, methodologies and models. The knowledge of the focus area helps the practitioner to know which parts of the software engineering process the framework, methodology or model is strong in. it is therefore to conclude that a detailed understanding of frameworks, methodologies and models will help us to develop better systems.

REFERENCES

[1] Olerup, A., Design Approaches: A comprehensive study of information system design and architectural design, *The Computer Journal Vol. 34*, 1991, pp. 215–224.

[2] Yeh, R., System Development as a Wicked Problem, *International Journal of Software Engineering and Knowledge Engineering*, Vol. 1, No. 2, 1991, pp.117–130.

[3] Feldmann, R.L., and Pizka, M., An Online SE Repository for Germany's SME: An Experience Report, *Proceedings of the 4th International Workshop on Advances in Learning Software Organization, LSO 2002,* 2002, pp. 34 – 43.

[4] van Vliet, H., Software Engineering: Principles and Practice, John Wiley and Sons, 2003.

[5] Pressman, R.S., Software Engineering: A Practitioners Approach, McGraw Hill, 2005, pp.77-99.

[6] Brooks, F., No Silver bullet: essence and accidents of software engineering, *IEEE computer Magazine*, Vol. 21,No.4, 1987, pp. 10–19.

[7] Naur, P., Randell, B. and Buxton, J., *Software Engineering: Concepts and Techniques,* Charter Publisher, in: Fitzgerald, B., Formalized

Systems Development Methodologies: a critical perspective, *Information Systems Journal*, Vol. 6, No. 1, 1996, pp.3–23.
[8] Glass, R.L., Is there really a Software Crisis? *IEEE Software*, vol. 15, no. 1, 1998, pp. 104–105.
[9] Shemer, I, Systems Analysis: a systematic analysis of a conceptual model, *Communications of the ACM*, Vol. 30, No. 6, 1987, pp. 506–512.
[10] Larman, C., A*gile and Iterative Development: A Manger's Guide,* Addison–Wesley, USA, 2004, pp. 9–39.
[11] Hoffer, J.A., George, J.F. and Valacich, J.S., *Essentials of Systems Analysis and Design*, Prentice–Hall 2004.
[12] Avison, D.E. and Fitzgerald, G., Information Systems Development: Methodologies Techniques and Tools. McGraw–Hill, 2002
[13] Glass, R.L. and Vessey, I., (1994), Rethinking the Nature Application System Development, *working paper, Pennsylvania State University.*
[14] Siau, K. and Rossi, M., Evaluation of Information Modeling Methods: A Review, *IEEE Proceedings of the 31st Annual Hawaii International Conference on Systems Sciences*, 1998, pp. 314 –322.
[15] Hofstede, A.H.M. and van der Weide, Th.P., Formalisation of Techniques: chopping down the methodology jungle, *Information and Software Technology*, Vol.34, No.1, 1993, pp. 57–65.
[16] Cockburn, A., Crystal Clear: A Human–Powered Methodology for Small Teams, Addison–Wesley, 2004.
[17] Ford, G., 1990 SEI Report on Undergraduate Software Engineering Education, *CMU Technical Report no. CMU/SEI–90–TR–003,* 1990.
[18] IEEE. IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12, 1990.
[19] Pfleeger, S. L., and Atlee, J.M., *Software Engineering: Theory and Practice*, Prentice Hall, 2005.
[20] McBreen, P. *Software craftsmanship: The New Imperative*, Addison–Wesley Professional, 2001, pp.25–30.
[21] Abrahamsson, P., Salo, O., Ronkainen, J. and Warsta, J., Agile Software Development Methods: Review and Analysis, *VVT Publications,* No. 478, 2002, pp. 7–94.
[22] Sommerville, I., *Software Engineering*. Addison–Wesley, 2004, pp.63-84.
[23] Appelo, J., Software Development Methodologies: the definitive list, [online]: Available from: http://www.noop.nl/2008/07/the-definitive-list-of-software-development-methodologies.html, [Accessed 30 April 2009].
[24] Ambler, S.W., Nalbone, J. and Vizdos, M.J., *The Enterprise Unified Process: Extending the Rational Unified Process*. Prentice Hall PTR, 2005.
[25] Edmonds, E. A. A process for the development of software for non-technical users as an adaptive system, *General Systems* XIX: 1974, pp. 215–218

Ernest Mnkandla is a senior lecturer and a Deputy Head of Department in the Department of Business Information Technology in the Faculty of Management at the University of Johannesburg, South Africa. He has lectured in IT Project Management and Software Engineering and agile methodologies, and has presented several papers on agile methodologies and project management within Africa, Europe and the Pacific Islands. Doctor Ernest Mnkandla completed a BTech (honours) in Electrical Engineering at the University of Zimbabwe in 1992, completed an Msc(Comp. Sc) at the National University Science and Technology in Zimbabwe in 1997, and a PhD in Software Engineering obtained from the School of Electrical & Information Engineering at the University of the Witwatersrand, Johannesburg, South Africa. His current research is in the relationship between agile project management and the Project Management Institute's PMBoK approach, the adoption of agile methodologies, and Project Management Maturity Models.