A Rational Perspective on Software Modeling

Tony Spiteri Staines
University of Malta, Malta MSD 2080, Malta

Keywords: Software Engineering, Requirements Engineering, Modelling, Systems Modeling, Software Development Methods, Lightweight Methods, Quality Assurance, Best Practice Approaches.

Abstract: This work introduces the concept of rational software modeling from a practical perspective. Valid arguments about the importance of modeling in modern software engineering and requirements engineering are presented. The different stakeholder's perspectives or views on modeling are analysed and a soft uniform approach is presented. The uniform approach or rational perspective to modeling is based on the main fundamental concepts of requirements engineering. This approach presents the basic ground for more elaborate work in the future. The universal approach is based on i) usability, ii) universality, iii) uniqueness and iv) uniformity. The concepts presented can be combined with any particular approach or method. The ideas could prove to be useful for quality assurance and best practice approaches in the real world.

1 INTRODUCTION

Modeling is the very act of creating representational artifacts of a system or its parts, for the specific purpose of explaining it to different stakeholders. It is possible to represent the software development process and the product itself using appropriate techniques. Modeling is useful for deriving the fundamental properties of software.

In the modern world the ever increasing complexities of applications and software systems, ranging from distributed systems to hybrid computing, require even more robust forms of modelling. As a matter of fact, requirements elicitation, need new ways of expression and representation. New concepts in system design are constantly being introduced as is the concept of design for service presented in (Aleksy, 2012).

In spite of the increasing importance of modelling, many software development companies and even students do not seem to comprehend the importance of this topic. Modeling remains a vague abstract concept. Another problem lies in the fact that some notations, like the UML tend to over model a system, whilst other extreme approaches like XP tend to focus more on coding rather than modeling. This work focuses on finding a rational solution. The idea is to understand the rationality of modeling and possibly find a middle path solution.

2 WHY MODEL

Modeling exists for various reasons. In theory modeling, apart from clarifying requirements, is intended to i) reduce project delivery times ii) promote reuse and iii) serve as a basis for contracts between different groups. In the RUP (rational unified process) business modeling is a core workflow process. This process is important in all the stages of the development starting off from the inception stage to construction and testing stages. Even in light methods, like Agile ones, modeling practices cannot be omitted. The fundamental question to ask is why do we have to model in the first place. Perhaps some software aspects need better comprehension or some ideas need to be communicated to different stakeholders. It could be simply that the model has to be created for the persons who will develop the system, test it or implement it. Some steps, in modeling, are i) identifying the target audience, ii) develop the model and iii) check if the model is suitable. One needs to know the audience of the model. Referring to the RUP the underlying rationale is to create suitable models for every stage of the process of application building. The key concepts and philosophy behind the RUP are applicable to different types of problems and enhance one's own learning experience.

3 SOME IDEAS

Traditionally, models were mainly used at the initial stages of analysis and development. As opposed to this in modern development strategies and methods models are not necessarily just developed at a single stage; but a refactoring process takes place with modeling, which is seen as a continuous process. Modern processes are focused on concepts like best practices, case tool support and the idea of having several processes, rather than steps or stages as traditionally presented.

Information modeling deals with presenting real world views to different stakeholders. Models can be diagrammatic or formal. Models can represent conceptual, logical or physical views (Moody, 2009). Many new different modeling notations exist because of new technologies and distributed systems (Jeusfeld et al., 1998), (Jeusfeld et al., 2009). As a case in point, the UML (Collier, O'Hare, and Rooney, 2004) contains several notations and hundreds of elements are specified in the superstructure documentation. But are these really relevant to proper design?

If the view of system architectural complexity is considered, the count of things like entities, connections and patterns are factors that have to be considered for modern systems which are obviously software intensive systems (Booch, 2008). Here the concept of relative vs absolute complexity is considered and it is only possible to focus on the essence, using principles of abstraction. States in the real world are extremely important and sometimes unorganized complexity exists (Cao et al., 2009). The solution proposed by (Booch, 2008) is i) focus on fundamental problems, ii) define abstractions, iii) employ good separation of concerns, iv) proper responsibilities. I.e. it is possible to deduce that the general idea would be to simplify and focus on the essence of modeling.

Complexity can obscure or hide the essential elements of a system. The uniqueness, elegance and aesthetical value of a solution, imply that simple architectures have more value than complicated ones for representation and modeling purposes. Simplicity is easier to comprehend and measure rather than complexity.

In fundamental modeling concepts (FMC, 2013) it is also suggested that diagrammatic notations should follow certain aesthetic principles. The main principles behind the FMC idea are i) abstraction, ii) simplicity, iii) universality, iv) separation of concerns and v) aesthetics and secondary notation. The key principles behind FMC have been successfully applied at SAP in the form of TAM (technical architectural modeling) (FMC, 2013) which combines the salient principles of FMC with those of some UML modeling notations, creating better diagrammatic notations for analysing and representing customer requests (Knopfel et al., 2006).

Even the lines, harmonization and aesthetical way of drawing artifacts all sum up to a rational way of modeling systems (Knopfel et al., 2006).

Systems can fail because the systems architect selects a fundamentally wrong architecture or something that is too complex to properly implement. Sometimes systems can fail because of the continuous adding of parts and little bits. A system or software development that started off as a simple task sums up into a cumbersome ordeal that lacks structure or proper representation. Software engineering is a discipline that requires the reasonable combination of dynamic and static forces.

Architectural refactoring implies the use of patterns that help manage complex system design. Patterns help with the observation and classification in a natural ordered way. I.e. the mental concept that is observed is easily repeated and represented for future work and support.

MDE (model driven engineering) has kicked off research on domain specific languages (OMG, 2013). The idea of MDE rests on the construction of proper software models that are used to create PIMs (Platform independent models). MDD (model driven development) is based on models and architectures that have to be properly represented for successful implementation. The idea of PIMs signifies that a model should be conform to certain rules but simultaneously express universality for different platforms.

In light methods such as Agile, Scrum or the UP (unified process) and even XP (extreme programming) proper modeling is a must, even though the models must be kept concise and simple for quick use.

The principles that can be learned from different processes, methods and ideas are applicable independently of the process itself. These can be easily combined with different methods or techniques without being limited.

Universality is a form of understanding the diverse types of modeling notations that can be confusing and frustrating. Software engineering is a complex discipline, depending on many different types of constructs and their proper selection.

4 DIFFERENT PERSPECTIVES

Given the focus of modern systems that are mainly directed towards the users and customers of a system (Aleksy, 2012), it can be observed that software development is based on a model centric approach. The perspectives can be classified into the i) organizational perspective or ii) stakeholder perspective. Here the stakeholder perspective is considered. In (FMC, 2013), (Knopfel et al., 2006) and (Booch, 2008) it is obvious, but not stated directly that models have certain basic properties that can be classified here as: i) usability, ii) universality, iii) uniqueness and iv) uniformity

4.1 Developers

Developers can be satisfied by using proper models that explain properly what needs to be done. A balance has to be sought between complexity and compactness. Developers tend to be unsatisfied if too many models are used and a lot of consistency checking has to be done to verify the models. I.e. consistency checking can become very tedious and time consuming. These principles are observable from agile methods or other light methods.

Normally the developers would require to have refined models and also lower level representations of the operations of the artefact they have to develop. These are constructable in conjunction with the system analysts and customers.

4.2 Project Manager

The project manager is more concerned with the technicalities or overall architecture of the system rather than the details. This is similar to a coarse grained approach. From this perspective the models would represent the architecture or top level structure of the system. Proper notations are imperative, but also the models must not be too complex. Obviously visual modelling should express ideas in simplicity and an aesthetically pleasant form. If the system being designed is not straightforward it might be necessary to use the experience of an expert systems architect.

4.3 Systems Analyst

For the systems analyst or business analyst, the role of model creation would be an important part of their job. It is important that this group is substantially trained in the use of correct specification modelling. This implies that their education would have to go beyond just using some notations. I.e. the systems analyst must be aware of the different notations, models and methods and must have the ability to select the best notation for a particular task or job. This depends on the environment in which the systems or business analyst works. I.e. if future systems will be similar to those being created now, then in the future similar modeling notations can be employed.

However, if the nature of the problem changes, this might imply that better notations have to be sought. Proper training in modeling can help greatly. The experienced analyst has to develop his knowledge gradually after being exposed to the modeling notations and techniques. There are so many notations, that selecting the best ones is not a simple task. A problem will arise if too many or too few model representations are used. A simple solution is normally the best, but it does not imply that important details are ignored. Later work depends on the initial artefacts, yet still there is the possibility for refactoring at later stages.

4.4 Tester

The tester or testing team does not normally construct models. However the testing relies on previous models for comparison. The development artefact is compared with the initial proposed model. For testing, models must not be too complex otherwise it will be a much more time consuming process.

4.5 Customers or Users

The customer or users play an important role. The focus in many types of development is on the end users of customers. This is evident in SOA’s, CMS and CRM systems.

Communication with the clients, customers is a must. Customers are not interested in the detail but in the essential properties of the system. The models used should reflect this.

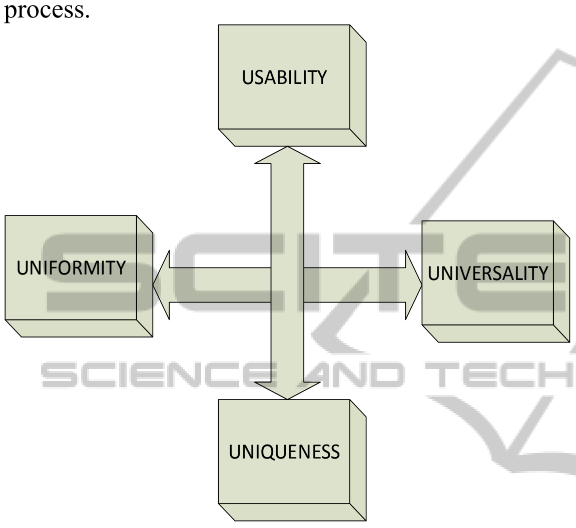
4.6 Other Stakeholders

Obviously system construction is normally not just limited to the primary stakeholders already mentioned. Managers, CEOs, CIOs and IT auditors might be interested in the system. Other roles like communications engineering or network engineering need to be give due consideration.

All these different stakeholders can exert pressure on the system and it is important to communicate with them using appropriate models for requirements and design decisions.

5 A PROPOSED UNIFORM APPROACH

The approach here is called the four U's. This can be included in any type of development method as required. The four U's are: i) usability, ii) universality, iii) uniqueness and iv) uniformity. These can be used to select the best notations or models for a particular task or complement a process.



5.1 Usability

Preferably diagrammatic notations should be used in combination with other forms of representation. Every artefact that is created will require some form of modification or updating over time. The models should be simple to update and correct without considerable effort. This is the Agile philosophy. Too much detail renders the model unusable for development teams working in environments that have a quick turnaround time. But for complex systems usability might imply a different idea.

Usability also implies the usefulness of the models from the customer or user viewpoint. In order to communicate the requirements the modelling approach being used should use effective structures that are simple enough for proper comprehension, but at the same time consider sufficient detail. From a certain perspective the concept of usability might be in direct conflict with complexity. For difficult systems or artefacts it can be difficult to explain the entire functions at a low level using simple representations.

Usability requires asking some fundamental questions about the notations being used. I.e. i) is the notation simple enough to understand, ii) do case tools that support the notation exist, iii) does the notation have a significant learning curve to get to use it, iv) is the notation suitable for sharing it in a team.

Depending on different stakeholders there is the possibility of different interpretations of usability. What is usable for one stakeholder group is not necessarily usable by another.

5.2 Universality

Universality implies that the modelling approach being used is based on effective structures for communicating the requirements. The quality of a software engineering method depends on the accuracy it has for representation. I.e. the method should focus on empowering and explaining the facts to the customers or the users of the method. This implies that there is some form of technique or method that is widely acceptable or can be comprehended simply by different groups of persons. Certain block diagrams and representational notations do not require special learning to understand. Examples of these are class diagrams, flow diagrams, DFDs, component and network diagrams, rich pictures, pictorial models, etc. Hence the modelling approach used needs to consider a wide target audience. Obviously the concept of universality merits more attention and study to comprehend the perception of different users having different abilities and learning skills. This is considered in the CMM (capability maturity model), where working knowledge is acquired after a number of years working experience.

From experience it is possible to identify notations and structures that have more significance of universality than other structures. This is possible because of the nature of the structures. I.e. diagrammatic representations and artefacts being visual, can have a more universal and wider appeal than structures that are represented using letters or sets of equations (Knopfel et al., 2006). Examples of structures that can communicate universality are i) use case diagrams, ii) class/object diagrams, iii) flowcharts, iv) structured block diagrams like deployment diagrams, network diagrams, etc.

5.3 Uniqueness

Uniqueness has several meanings and implications in software development which do not necessary agree with what is presented here. i) The solution or the artefact developed is unique, ii) the design is unique, iii) the problem is unique, etc. Here uniqueness mainly refers to the modelling approach. Obviously the model is part of the uniqueness of the solution. The model should be unique in the sense that it provides sufficient detail for a proper or special solution. This means that there is no need to repeat the obvious or to use several models that show the same entities or processes in different ways unless this is really required.

Uniqueness also states that the solution or the artefact is of a certain standard and quality. The approach to systems development is not just a hit and miss approach, but a scientific problem solving approach that is based on structure and proper comprehension about the problem domain. Any solution will not just do, it is required to have a set of solutions and to select the best possible one. Uniqueness means that the best proper notations for the problem domain are to be used. The idea of uniqueness is not learned in a short span of time but it can only come when one has acquired proper problem solving skills after practical work in the field combined with thinking skills. Group sessions, discussions, brain storming, teamwork and on the job training can contribute in this respect.

Uniqueness also implies that the structures have value or add value. The structures or notations must add some knowledge, they should not state what is obvious or already known but explain correctly what is unclear or obscure about what has to be developed.

5.4 Uniformity

In object oriented software engineering, uniformity represents the data integration that is the result of linking several abstract classes from several systems or sub systems to produce a shared model. Uniformity can be used at different levels of abstraction: i) conceptual, ii) logical and iii) physical. Uniformity refers to the proper integration of the modelling notations being used. Linking together the models and sharing the information from different models implies that the models cohesively fit in with each other, i.e. the models must support each other properly, with no overlapping. Uniformity becomes a big problem when too many diagrammatic and other notations are used to support software development. Consistency checking between the models becomes a time consuming complex process that wastes a lot of resources. The fewer the models the easier it is to have proper uniformity in place. The more complex a system is the more difficult is uniformity mantained. If FMC (fundamental modeling concepts) and TAM (technical architectural modeling) used at SAP are examined, the concept of uniformity becomes clearly visible. The notations used in FMC for architectural modeling, combine i)compositional structures, ii) behaviour and iii) data/value structures. I.e. there is uniformity between the three types of structures. On the other hand, if notations from UML are taken and combined with other notations quite often it is possible that the concept of uniformity is ignored.

6 BEST PRACTICE APPROACH

The ideas of the four U's presented can be implemented as part of a best practice approach, or in an appropriate framework. The concept of best practice is based on work related excellence. The idea is for continuous quality improvement and the learning process that requires due consideration. There is the concept that one size does not fit all. This means that there is no identical solution for different problems. For modeling this is more of an issue because it is intrinsically very difficult to compare system requirements. A certain degree of flexibility in modeling is a must, but the flexibility must be properly balanced against complexity. The experience required for proper modeling cannot be achieved only by formal training, but requires on the job training and perhaps a number of years of practical experience. But obviously formal training is imperative to get started on the concepts related to modelling (Stone and Madigan, 2007). Perhaps, in certain computer courses and training, there is still a lack or proper problem solving formulation which is propagated further up. A best practice approach means that one learns from mistakes and applies expertise that can only be developed with time. Sometimes it might look unpleasant to try to enforce a culture where design notations and modeling are given a great amount of importance. Because of excessive rigor, certain individuals might prefer to keep concepts and system details in their mind creating problems. Proper judgement and intuition provide for proper reasoning to problem solving. In a best practice approach notations, modeling and documentation are a must.

Elaborate solutions are not always desirable and elegant approaches do not mean that elaborate features and complexities are dealt with. Under-engineering might be a problem just as over engineering. Certain challenges implicate that a best practice culture is developed and instilled in the organization that is using modelling techniques, so that a rational perspective on modeling is taken.

Unfortunately the philosophy of a best practice approach has to pervade the entire culture of the organization and this might imply that it will take a long time to develop it properly.

7 CONCLUSIONS

The importance of proper modelling in the field of software and systems engineering cannot be over emphasized. In modern environments where complex holistic solutions are always increasingly demanding, models must prevail at every stage of proper project management. Considering the difficulties for selecting proper models, a rational perspective for software modelling has been presented. The concepts can be applied to any type of project ranging from small to large size and independently of whichever method is used.

Obviously users of Agile methods like Scrum and XP might find these ideas of interest to them. Even users of the UP (unified process) might consider implementing the concepts for further quality improvement in the design of software products.

It is obvious that commonsense should prevail when using these ideas and that the concept of one solution fits all one can never fulfill the complete needs of different problem domains. Hence the concept of adaptability that is fundamental in principle to agile methods must be given due consideration.

Best practice approaches can only be learned through acute observation and mistakes that can happen in time. This is similar to the idea of the capability maturity model where the experience gained over a number of years and projects in the field of software engineering are quantifiable. Obviously the approach presented here needs to be validated in future work.

REFERENCES

Aleksy, M., 2012. Coverage of Design for Service Principles in Software Engineering*, 6th Int. Conf. on Complex, Intelligent, and Software Intensive Systems (CISIS)*, 100-105.

Booch, G., 2009. The Defenestration of Superfluous Architectural Accoutrements, *IEEE Software Domain Specific Languages and Modeling*, vo.l 26, no. 4., pp. 7-8.

Cao, L., Ramesh, B., Rossi, M., 2009. Are Domain-Specific Models Easier to Maintain than UML Models?, *IEEE Software Domain Specific Languages and Modeling*, vo.l 26., no. 4., pp. 19-21.

Collier, R., O'Hare, G., Rooney, C., 2004. A UML-based Software Engineering Methodology for Agent Factory, Int. Conf. on Software Eng. And Knowledge Eng.

FMC., 2013, TAM - The SAP way combining FMC and UML, Technical report and documentation: http://www.fmc-modeling.org/fmc-and-tam

Jeusfeld, M.A.., Jarke, M., Mylopoulos, J., 2009. *Metamodeling for Method Engineering*, The MIT press 1st edition.

Jeusfeld, M.A.., Jarke, M., Nissen, H.W., Staudt, M., 1998. *ConceptBase Managing Conceptual Models about Information Systems*, Handbook on Architectures of Information Systems, Springer, ch. 12, pp. 265-285.

Knopfel. A., Grone, B.,Tabeling, P., 2006. Fundamental Modeling Concepts, *Wiley* 1st edition.

Moody, D.L., 2009, "The Physics of Notations: Towards a Scientific Basis for Constructing Visuals in Software Engineering", IEEE trans. On Software Eng. 35(6):756-779.

Stone, J.A., Madigan E., 2007. Inconsistencies and Disconnects, *Communications of the ACM*, vol.5., no.4., pp. 76-79.

OMG., 2013, MDA - The Architecture of Choice for a Changing World, OMG Documentation Website: http://www.omg.org/mda.