ISSN (e): 2250 – 3005 || Volume, 09 || Issue, 1 || January – 2019 ||
International Journal of Computational Engineering Research (IJCER)

Basic Concepts for Creating Visual Models

Anthony Spiteri Staines
Associate Academic
Department of Computer Information Systems University of Malta, Msida MSD 2080, Malta

ABSTRACT
This work explains the fundamental principles for simplifying and improving visual modeling. It builds on previous work presented in [1]. Sections I to II discuss software and system development, different types of techniques and modeling approaches are used. Visual modeling is suited to developing systems because these notations are understandable by different stakeholders. Visual modeling can make use of graphs. The paper structured as described: I. Introduction: Information and software systems require the use of methods and notations for proper representation. Unfortunately many users do not appreciate the need for creating suitable and aesthetically good diagrammatic notations II. Motivation and Some Background: Different modeling notations can be used to model systems. The key principles behind these notations are based on principles of tidiness, neatness, constructability and the level of detail. Different works and findings are presented. III. Problem Statement: This section presents various problems with visual modeling. In reality the use of good principles for creating the models are not necessarily identified, considered and adhered to when diagrammatic notations are used. IV. Some Possible Solutions: several key principles are used as solutions. These are:i) abstraction, ii) universality, iii) aesthetics, iv)correct sequence and v) patterns. Their importance is explained and it is indicated how these can improve and solve the overall diagrammatic modeling approach. V. A Toy Example Using A Control Flow Diagram: Models can have several characteristics and still be useful and offer good representation. VI. Discussion and Fundamental Issues: Explains the validity of the toy examples and how this can be extended to other models. Several problems still remain and these are not straightforward to solve. These issues are explained in this part. VII. Concluding Remarks: Summarizes the paper and explains other issues that can be tackled in the future. Many of the principles and ideas presented in this work can be used to improve other domains of computing and even other fields of computer engineering.
KEYWORDS:*Diagrammatic notations, software engineering, requirements engineering, visual modeling*

Date of Submission: 03-02-2019 Date of acceptance: 20-02-2019

I. INTRODUCTION

By using the term visual modeling it is implied that this is a key principle for information systems analysis, software, hardware design and computer engineering research. Modeling makes use of several forms of system representation which can be grouped into i) visual modeling and ii) formal modeling.

Several formal and semi-formal methods, notations and design methods have been created. Normally traditional formal methods are based on expressions, logics, set theory etc. Many of these approaches are quite rigorous and they are based on sound mathematical principles. Semi-formal methods exist and can be combined with formal methods. Some examples of semi-formal methods are: the UML, software design methods and techniques. These semi-formal methods cam make use of graph based diagrams as in [1]-[11], [13]-[16]. In a sense semi-formal methods can combine principles from formal modeling with visual modeling.

Extensive literature and courses have been developed to support knowledge related to modeling in [1]-[8]. This topic has no end in sight and is continually evolving. Modern system development is not just responsible for the creation of software but even other parts like the hardware interface design, the design of the gui (graphical user interface), integration, testing etc.

The design prior to coding principle is a mainstream approach that is adopted in software development. This fundamental principle is used in many other areas of system design and construction. Modeling is not only useful and used for good system design. It can be applied to many other computer related areas.
Several reasons exist for modeling. These are: i) understanding a system, ii) simplifying the representation of a system, iii) component identification, iv) scalability, v) representing the architecture, vi) communicating to stakeholders, vii) verification, viii) analysis and design, ix) creating architectural specifications etc.as in [1]-[4], [13]-[16], [27]. This list is by no means exhaustive, according to the problem scenario new uses for representation can be found.
Modeling has become a topic of fundamental importance for the design and analysis of computer systems. It is increasing complex to select the right model for the job as in [2],[3]. Some models are never used because of their complexity or oversimplification. Models provide and give insight to different scenarios. There are several cases where models are not used for various reasons.
Models used in computing range from formal models and mathematical models to different forms of visual models [1]-[13].
In this work the focus is on visual modeling. Visual models are easier to understand at a glance and appeal to a wide audience [4], [6]-[8]. This paper is concerned with visual models mainly based on graphs and their representation.
Modeling principles and skills are often left to the user to develop and are not taught as a separate and important topic. Abstraction and representation lie at the very core of system development.
Many existing notations used in literature are modification of block diagram notations and graphs [3]-[4],[6]. These are simple and straightforward. Because of their effectiveness they play a crucial role in software modeling. Currently many compositional structures that are used for representation are based on modifications of these models [3],[4],[6]. This is evident in the UML and in Fundamental modeling concepts (FMCs) [3].
It is imperative to educate students and IT professionals from the onset about key modeling principles for good system representation [10],[13]. Good notations and designs should support the process of creating better and more robust systemsand they support the intuitive creative thinking process which is fundamental in requirements engineering and software engineering [3], [13]-[16]. Creating models also has an artistic and expressive side to it which is very often ignored [23],[24]. How has the balance between the artistic side and the expressive or relation side have to be maintained? In many instances it can be seen from practical work and in many applications that the fundamental and universal principles for modeling are not well known or are ignored. Many diagrams are too complex to read at a glance and understand [25]. As an analogy a similar problem seems to manifest in computer programs. Certain programs are unnatural to use. They are counter intuitive or badly designed. This implies that users will reject using them for several reasons. Could this happen because of over complex design? Visual modeling is not only related to design but to the approach adopted by the user. The level of problem solving skills acquired can be seen from the approach that is used. Visual models of quality will add value to the system.
Modeling can be symbolic in nature [2], [14]-[15]. In literature there are several approaches that have been created for visual modeling. The mainstream approaches can be shown as follows: the i) UML notation [7], ii) Fundamental modeling concepts (FMCs) [3], iii) Technical architectural modeling (TAM) [16],[21], and different approaches that are graph based, formal or semi-formal [16],[19]. Some graph based approaches could be different classes of Petri nets or plain graphs like those used in [4],[6],[9],[12],[12],[21],[22]. The key principles presented here are an integral part of FMCs and TAM. The FMC approach places a lot of emphasis on aesthetics, node harmonization and layout of the visual models. The same ideas apply for TAM. This can be considered to be a systematic artistic part. Unfortunately when using UML and graph notations these principles might not be considered. The emphasis in the UML is to provide a large unified repository of notations for system representation. Other works focus mainly on formal representation, which is a very important aspect. The ideas in the problem solution section related to good diagrammatic notation can be extended to all visual modeling domains [26],[27] used in various computer related fields.

II. MOTIVATION AND SOME BACKGROUND

In this work modeling can be defined as the activity of creating valuable structures that capture the behaviour or composition of a system. This is an intellectual activity that adds value and involvesstructures that either i) exist or are ii) planned to create more knowledge and value about them [1]-[16]. Many different methods and notations are presented in literature and the motivation of this work is to improve the selection and use of these notations. The ideas will help to improve representation.
The main requirements of systems must be precisely understood by designers and other specialists [2], [3]. Information and knowledge about a system might have to be exchanged between different persons. Models serve as exchange mechanisms for knowledge that could otherwise go missing. Drawing models and their interpretation are two separate activities. Models can be refined and transformed into new models or notations.

Diversity in notations used for representing modern software development happens because of the different types of requirements and complexity issues. New problems and challenges are being created all the time. Certain modeling structures that are universally accepted and understood are normally used for normal stakeholder comprehension. Class diagrams are an example of this [7], [8]. New structures might have to be created for particular scenarios. Formal models that are non-visual are shunned by stakeholders being unfamiliar with the notations that are being used.

Just for the sake of comprehension of different requirements. E.g. the requirements of real time systems are very different from those of an e-commerce application. It is impossible to reconcile these because the nature of the end product is entirely different. Each of these systems requires the use of diverse modeling notations and techniques [23]. A delicate balance has to be maintained between using too few or too many models.

Modern systems could be diverse as distributed systems and web-based applications. At the extreme side, embedded applications could be stand-alone applications. Any system can contain a finite number of subsystems depending on classification. Sub-systems can exhibit diversity. These type of systems open up a myriad of diverse possibilities and unique needs.

A single simple system can have multiple configurations excluding those that not necessarily accounted for. Diversity, configurability, distributedness, new technologies and platforms etc. require proper management.

Creating good software does not only imply successful coding but involves several other structures and sound modeling principles [3]. These can be extended to the gui creation, interfacing, usability, safe levels of the system, etc. These factors must be considered for the success of the software artifact.Modeling issues tend to be neglected and are not properly understood by many stakeholders in development. This is obvious when good programs that are created require to be developed again from scratch because some important issues were not implemented or considered. Even in modern technologies it is possible to get some high end device that has a poor ui (user interface) layout or software that is too tedious to operate or gives unforeseen errors in some state that was unaccounted for.

Good models should promote reuse. They should be restrictive, descriptive, suggestive and self-explanatory. For comprehension it should contain a sufficient level of detail. Tidiness implies that it is neatly laid out and drawn up using appropriate scales. The model must be suitable for visualization. Principles of layout harmonization and proper aesthetics in its design must be self-evident. Artistic principles should also be included if possible. The topic of modeling is still evolving and new methodsand techniques are constantly being created as the need arises.

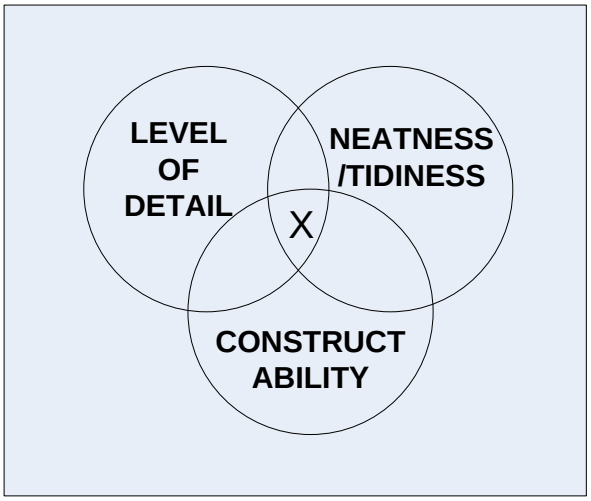

Figure: 1 Three Key Attributes of System Models [1]

III. PROBLEM STATEMENTS

Several reasons exist to explain why modeling is neglected: i) there is a lack of uniform modeling foundation, ii) so many diverse notations exist. Which should be chosen? iii) there is a lack of training in requirements engineering and modeling, iv) it is time consuming to apply notations. When should we stop applying them? How much is enough? Which should be selected? v) there is a lack of education and training about the importance of models. For more details refer to [1]-[22]. Normally a user gets an informal education in modeling. The user learns to use models indirectly in his education and work experience. The skill of modeling is not taught. The problem is that normally no systematic approach to modeling has been presented and the importance of modeling as a topic in its own right is ignored. Then with work experience the user stands a better chance of selecting the right model for the job and appreciate the use of robust modeling.

In many occasions the use of models is overlooked for various reasons like lack of time or prohibitive costs. There is a lack of instruction at different levels on how to produce suitable models. Analysts and developers are in a rush to get done with the application and solve problems i.e. the creation of models could be prohibitively time consuming. The result of having a quick implementation comes at the expense of proper comprehension of the problem domain. Another possible reason not to use notations could be information risk exposure as the notations would divulge too much information and could be used for reverse engineering by competitors to their advantage [24]-[27].

In many cases detailed diagrams of the system components and their functionalities are not provided for specific obvious reasons like size, complexity and readability issues. Hence it is preferred to use smaller diagrams.

Lack of proper representation is one reason of the large software costs. This leads to the problem of not doing something right at the first attempt so many other attempts need to follow. The following are mainstream problems that can be clearly identified when constructing system models in general: i) precise and fundamental terminologies need to be used. ii) the model should be comprehensive enough to reflect about the structures of the system and the subsystem. iii) add value and be understandable by different stakeholders, iv) concise or retain simplicity but not at the cost of omitting important details. v) be very presentable and simple to construct. vi) the models should promote reuse and transformation to other frameworks or applications [2],[3],[6],[7],[13]-[15].

The main issues presented here can be summarized or condensed into the following fundamental points:

i) the model is too detailed or contains too little information. If the model contains too much information for the user it becomes unreadable and some of the information is not taken in by the user. On the other hand if the model is too compact, then there is insufficient information and knowledge that can be derived from the model.

ii) the model is not neatly drawn. This is a big problem with many mainstream modeling approaches [6]. The models could contain overlapping edges and nodes. The size of the nodes and the node arraignment setup give the model a look of untidiness. It could also imply that excessive nodes and edges are used [6].

iii) the model should be easy to construct. This implies that there should not be a high level of difficulty to construct the model.

These three main points summarize many of the problems with modeling systems and software. These are shown infigure 1.

More problems directly and indirectly related to these three main issues are discussed in section VI.

IV. SOME POSSIBLE SOLUTIONS

For i) too much detail, the solution would be to simplify the structures and on the other hand for ii) too little detail more information could be added. The solutions presented are by no means absolute solutions. Several tradeoffs and dilemmas exist. There is no single perfect solution to these [1].

The solutions cannot be implemented without the following qualities. The users need to comprehend the value of modeling: i) the user needs to have some idea or a rough idea of how the system works and its possible main components. ii) the user needs to have the ability to formulate or give shape to different classes of problems. I.e. have some explicit form of representation to put the concept into visible form. iii) knowledge of basic computer related structures is a pre-requisite and iv) one has to identify alternative structures or ways of representing.

The key principles presented can be used to solve many of the problems with visual modeling. These form an integral part of creating good visual structures for system and software development. Some of these principles have been already presented to some extent in the MDA, MDE literature, FMCs and TAM and the UML [7],[8],[16],[3]. In this work more details have been added. These have been identified and put together from experience in this area over a long period of time.

These key principles are:

Abstraction: this is the ability to describe conceptual architectural structures at different levels. Abstraction implies simplicity. Simplicity happens because the representation is decomposed. Description techniques should be restricted to very few elements and notations.

Universality: a description albeit being simple requires to provide sufficient information that will cover a variety of situations and scenarios. Basic block diagram notations and certain graphs like undirected graphs or digraphs can be used to create useful models that are visually understandable. Many of the diagrams used in the UML like

activity diagrams, class diagrams, sequence diagrams are based on graphs. Petri nets and FMC or TAM diagrams are similarly based on graphs [3],[4],[9],[11],[12],[16],[21],[22].

Separation of concerns: there are important and trivial parts. Different parts of the system have different requirements hence the different parts (e.g. components or entities that collaborate together are fundamentally different.

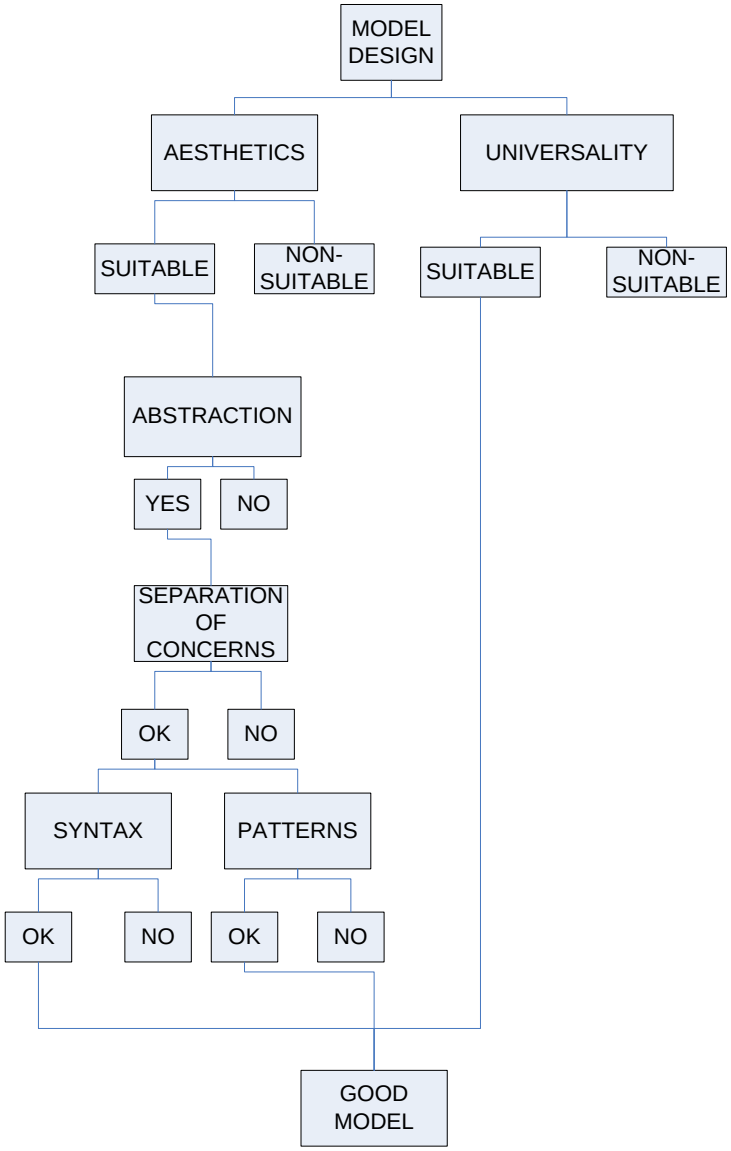

Figure: 2 Dynamic Vs Static Composition Model Attribute Classification [1]

Aesthetics: this deals with the neatness or tidiness of the diagram. This implies that a structure should support a proper layout and presentation. The graphical patterns should be well formed and follow certain key principles that make them presentable. The aesthetical value of the diagram is often overlooked or ignored. The principle of aesthetical value can be extended even to formal notations or representation that are non-graphical. The actual process of creating structures for computer based systems can be attractive not just from the scientific and economical point of view but also from the compositional point of view. Thus it is possible to acquire a skill where designing becomes a sort of craft or art. Aesthetical design would imply the correct layout of nodes, and harmonization of the layout. This will imply proper graphical connections and tidiness in the design. These principles are not easily defined and it is important to be tied down to a particular approach.

Correct Syntax: this property deals with the correctness and validity of the model being created. Sometimes the syntax used is full of errors. The tool used to create the model could use some form of syntax that is not the proper one. This is commonly seen with the UML where different case tools support different syntax. The model can normally be supported using formal methods to give it more rigor if this is required. Correct syntax is not necessary in the most important part of the model. Sometimes the aesthetical value of the model is more important.

Patterns: a pattern refers to something that can be repeated. In the case of modeling the patterns refer to the building style properties of the model. E.g. if a form of representation is similar, in this case the pattern will also be similar. In the case of modeling having a pattern, this implies that for a particular representation in a diagram a similar layout or form is used. E.g. if a decision node is being modeled twice it should look similar.

Simplification: simplification implies that the model is not too difficult to construct and comprehend. E.g. if a graph type notation is being used it should not have too much nodes and edges that are confusing to understand. This principle goes hand in hand with the principle of aesthetics.

V. A TOY EXAMPLE USING A CONTROL FLOW DIAGRAM

A simple control flow graph (CFG) is used to illustrate the ideas presented in the proposed solutions. These have already been given and are explained in [1] also. The CFG represents the basic processing of a typical bank ATM (automated teller machine) [1]. This is selected because the ideas can be easily understood. The processes or activities in the CFG are just used to explain the control flow and do not necessarily reflect correct ordering. This CFG can be refined and subgraphs can be added. There are several ways and many different notations used in software engineering that are used to represent CFGs. There are several techniques and notations that can be used to draw and illustrate CFGs.

The example of the CFG drawn using the principles presented here are constructed in fig. 3 and 4. These show exactly the same thing even though they look different. It is possible to find many other valid combinations for representing this example [1].

The focus is on generating models that are usableand understandable. As regards aesthetics some might prefer the diagram in fig. 4 to that in fig. 3. For others the opposite might hold. In fig. 4 there is more abstraction and separation of concerns than in fig. 3. This is because the ATM menu has been properly separated from the main functioning of the ATM [1].

VI. DISCUSSION AND FUNDAMENTAL ISSUES

In this section some key principles that are useful for creating good models are indicated. These would help software engineers to come up with better designs. The principles presented can be easily implemented with minimum effort. Ideally, they should be a crucial part of software engineering. However this is not normally the case.

In section V, CFGs have been used as an example but the key points presented here can be used with any type of diagrammatic and graphical notation like those found in MDA [8], UML [7], data flow diagrams DFDs, class diagrams [7], block diagram notations [3], etc. The key principles can be used elsewhere.

The principle of aesthetics is not often considered. This concept could create a new field of modeling notations and software development models. Other principles like universality, syntax, patterns and abstraction are closely linked with aesthetics.

The suitability and usefulness of models is not always in agreement with good construction principles. This is because for good construction principles the following apply i) representation and ii) understanding. Representation does not necessarily guarantee that a model is easy or simple to translate into a given specific language. Why? It could be that the model cannot be translated automatically but requires additional information or additional human information. The measures presented in section IV are quite subjective and might not be in agreement with different types of stakeholders. There are no universal rules that actually define what defines‘a good model’ and how to use it or interpret it.

System size and complexity are very difficult to estimate and describe. It is fundamental to know at what stage a model will be used. Will the model be used at the analysis stage, design stage, for system documentation or at any other stage?

A model to be used for input to a tool or another transformation needs to be complete and free from inconsistencies. There are several secondary issues for it to be communicable and usable at the stakeholder level. When teaching software modeling in topics like requirements engineering and software engineering the layout and visual impact of the model might be of greater significance than its actual completeness. This is because the functionality of the model should serve its purpose as an agent of i) representation ii) communication and iii) pattern capturing. Ambiguity will not allow proper comprehension.

Unfortunately if MDA [8] and UML [7] driven approaches are considered, architectural models can become surprisingly complex and unmanageable. The UML representation can become too rigid. Abstraction does serve to reduce complexity but detail can be lost. It cannot be expected for MDA and UML along with other mainstream approaches to fully replace the need for creating simple system diagrams using block diagram notations. These serve to enable better thinking and give a holistic picture of the system. It is observed that there does not exist a single notation that covers all the modeling needs for asystem. There is a great amount of fragmentation and diversity in requirements engineering thus complicating issues related to its presentation. This happens because different stakeholders require different viewpoints.

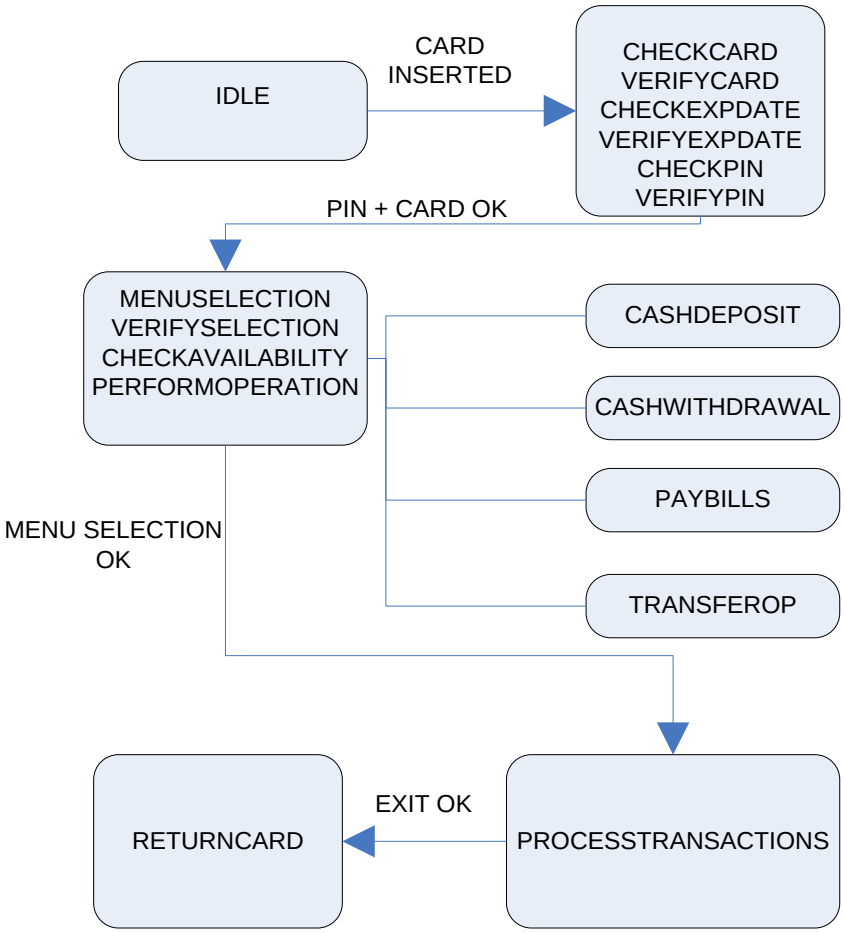

Figure: 3 A Bank ATM Simplified Control Graph [1]

Visual models created to represent information systems can be developed to create new forms of representation and understanding. The visual models can be combined with pictorial representation, other notations and mathematical expressions as required. The idea of reuse in a model driven engineering approach is presented in [25]. This work gives an indication of the complexity of modeling that remains unsolved and indicates that simplification is of great importance to creating usable models. The ever increasing diversity and heterogeneity of systems and their components complicates things.

The expression of form through geometry, graphs, interacting shapes and symbols provides one level of the expression of knowledge and understanding, depending on the transformations carried out.

Graph diagrams and geometric shapes can undergo several processes of dissolution, expansion or contraction of their components. Graph drawing for visualization is a vast area in its own right. Thereare various algorithms that can be used to optimize the layout of the shape and the particular drawing. Several different layouts can be found to depict the shape. The concept of aesthetics can be expanded to implement ideas related to symmetry, size, vertices layering etc.

There are software tools and several algorithms [6] that can be used to redraw graphs. The use of these is important because a poorly drawn graph can have its nodes and edges re arraigned in a much neater way that makes the diagram more readable. The way a graph or structure is drawn can make a big difference.

Selection of the appropriate notation for representation presents a significant challenge to inexperienced users. A user must be familiar with awhole set of notations and methods before there is the possibility of selecting the best notation and optimizing the drawing or layout of the notation. This knowledge will come with time and experience implying that sub-optimal solutions are normally used in a trial and error fashion. The design artifacts in some cases will reflect the actual implementation. Hence design representation is of fundamental importance at the initial stages. There is a clear distinction between hard physical reality which represents the implementation and the perception of it which is expressed via modeling approaches. Issues that are not normally expressed are related to the scale of the model. Should the model be a large scale or a condensed model? These questions pose open problems that are not easily solved or understood.

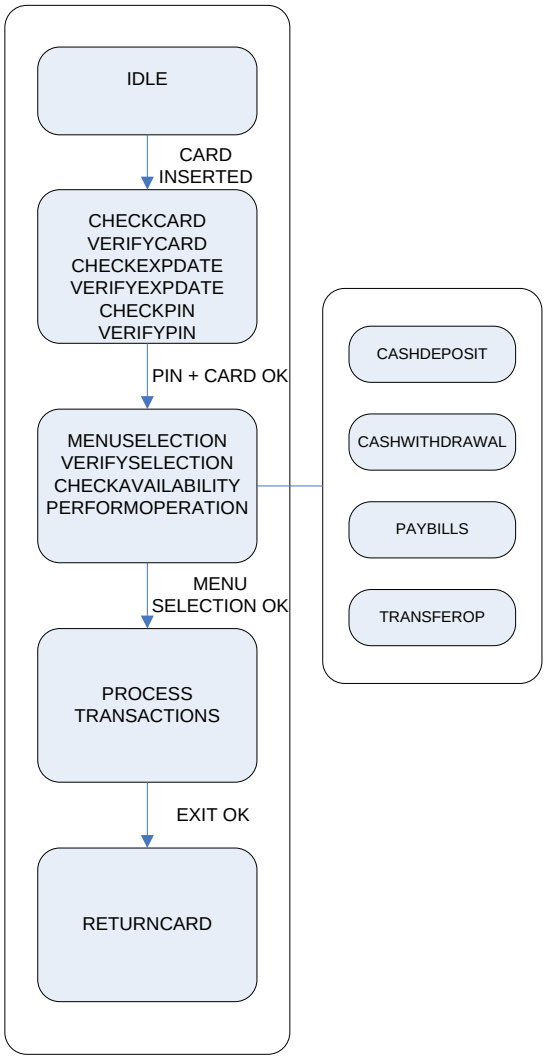

Figure: 4 An Alternative Representation Using Some Key Principles [1]

A model or a system can evolve dynamically in the real world. This is an additional problem that requires more work in this field.

Examining a diagram, its connections and shapes immediately indicates how its components do relate to each other. A user can discuss various relationships of patterns, connections, layout, form and aesthetics. Liking or disliking the object model is to a large extent dependent on how we perceive the model as a whole.

This is similar to when we see a well laid out model. We experience various qualities based on the relationships between the entities, the size of the nodes, communication, form, aesthetics, layout etc. In this respect modeling is not just a scientific approach but it also has its artistic part. This part is still not properly understood in software engineering.

Software engineering has a variety of issues related to modeling and quality that remain unresolved till this very day. Ideally one must not become attached to a particular approach or notation. This could imply that under certain circumstances some rules and key principles mentioned might have to be broken in favor of others. This choice is left up to the user to decide what is best with ample room for experimentation and creativity.

It could be of great benefit if computer specialists derive knowledge about modeling by studying or understanding other domains that are not directly related to computing.

E.g. Physics, mathematics, applied sciences. Modeling is practically used in every field and certain key principles are universal. The learning experience of modeling in different domains could help the user to better abstract the problem and see different viewpoints that previously would gounseen. Another area that requires investigation is that of information graphics. The ideas presented in information graphics could be combined with the models used for describing systems. There is much to be learned from this area.

Another issue that is imperative is that in computing the solution and the modeling process is done in a hurried way leaving little space for thinking and playing with the model for improvement. This problem is not simple to solve because the development environment used could favor a rapid application development approach leaving

little time for modeling at the expense of developing artifacts that are at a sub optimal level. If more space and time was available then this issue could be solved. Quick solutions are normally cheaper at the expense of omitting important facts and details.

A model can be made i) more complex, ii) simpler. A complex model is more faithful to reality whilst a simpler model is easier to handle. It is a dilemma whether a simpler model is to be preferred over a more detailed one or vice-versa. Reality shows that it is not possible to create a perfect model. This is because a model is always a restricted representation of a system. To implement more details entails a higher price and more effort that is not always requested. If the model is too large and detailed the model is like a large map which sometimes is useful and at other times might be too much. The larger the model the greater the drawbacks for creating and using it. This problem can be called over-modelling or better explained as overrepresentation where too much detail is given for its purpose. This problem can happen also in software designs where the application is overdesigned or over-engineered where a disproportionate effort is placed on the qualities of the application which are deemed unnecessary. It could also be that many features in the application are never used or really needed. This problem is inherent in many web applications, mobile phones etc. that are developed, where at least 10%-20% of the features might never be used. It seems that the problems of over-engineering and overrepresentation are related to one another. Excessive detail can imply that the system is too difficult to fix and comprehend. Even usability of the system will be problematic with over-engineering.

When creating a model at what level of detail should we stop refining the model? This process could be an ongoing process with no end in sight. A good model should be able to teach by example.

The key solution to these problems is simplification in the design and in the constructionstages. Simplification is separate topic that is based on principles of reduction and composition. Simplification of models used in computing is non-trivial and deserves a lot of attention. Simplification implies straightforward communication of salient aspects and attributes of the problem domain.

Table 1 below summarises the key issues explained in this part.

ISSUES	**DESCRIPTION**
Over Modeling	Complications
Over Engineering	Complications
Simplicity	Important attribute
Abstraction	Important attribute
Universality	Important attribute
Aesthetics	Model design
Node Layout	Graph design

Table: 1 Some Key Issues For Consideration

VII. CONCLUDING REMARKS

This work deals with basic key principles for creating good system and software models. But it is far off from finding a solution to this problematic area. Requirements engineering and software engineering have several problems that have withstood the test of time and that seem to remain unsolvable. The more complex and fragmented systems become, the greater is the significance and importance of sound modeling principles. A one size fits all solution does not work. Formal methods and formal notations could be combined with the visual diagrams for creating more robust models.

Simplification is a universal principle that needs to be considered when modeling systems. Even though simplification looks like something trivial in reality it is not an easy topic and requires a lot of work. New and more expressive modeling methods will have to be created. Teaching and instructing students and other professionals in this area is a great challenge as certain skills are acquired only through the painstaking learning experience of trial and error.

REFERENCES

[1]. Anthony Spiteri Staines. Some Key Principles for Creating Good Visual Models. International Journal of Computers (IARAS), 3, (2018) , 112-119.
[2]. H. Kaindl And J.M. Carroll, Symbolic Modeling in Practice, Communications of the ACM, vol. 42, No 1, (1999), pp. 28-37.
[3]. A. Knopfel, B. Grone, P. Tabeling, Fundamental Modeling Concepts, Uk: Wiley, (2006).
[4]. K. van Hee, Information Systems: A Formal Approach, Cambridge Univ. Press, (2009).
[5]. C.B. Jones, Systematic Software Development using VDM, Pretence Hall, (1990).
[6]. G. Di Battista, P. Eades, R. Tamassia, I.G. Tollis, Graph Drawing Algorithms for the Visualization of Graphs, New Jersey: Pretence Hall, (1999).
[7]. OMG, UML Super Structure Specification Documentation, 2018,https://www.omg.org/spec/UML/2.4.1/About-UML/
[8]. OMG, Model Driven Architecture, (2018), http://www.omg.org/mda/
[9]. A. Spiteri Staines, Some Fundamental Properties of Petri Nets, International Journal of Electronics Communication and Computer Engineering, IJECCE, vol.4, Issue 3, (2013), pp. 1103-1109.

[10]. T. Spiteri Staines, A Rational Perspective on Software Modeling, Software Engineering and Applications, 9th ICSOFT- EA, (2014), pp. 345-350.
[11]. A. Spiteri Staines, A Triple Graph Grammer (TGG) Approach for Mapping UML 2 Activities into Petri Nets, 9th SEPADS conf., WSEAS ,Cambridge UK, (2010), pp. 90-95.
[12]. A. Spiteri Staines, Rewriting Petri Nets as Directed Graphs, Int. Journal of Computers, NAUN, issue 2, vol. 5, (2011) ,pp. 289-297.
[13]. J. Osis, and E. Asnina, Topological Modeling for Model-Driven Domain Analysis and Software Development: Functions and Architectures, Model-Driven Domain Analysis and Software Development: Architectures and Functions, (2010), pp. 15-39.
[14]. T.D. Kelly, Symbolic and Sub-Symbolic Representations in Computational Models of Human Cognition, Theory& Psychology, Sage Publications: Vol. 13, No. 6, (2003), pp. 847-860.
[15]. D. Hofstadter, Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought, Great Britain: Penguin Books: (1995).
[16]. www.fmc-modeling.org, Standardize Technical Architectural Modeling Conceptual and Design Level, SAP, http://www.fmc-modeling.org/download/fmc-and-tam/SAP-TAM_Standard.pdf
[17]. S. Fleurke, Forecasting Automobile Sales using an Ensemble of Methods, WSEAS Transactions on Systems, WSEAS, Vol. 16, (2017), pp. 337- 345.
[18]. Agostino Poggi, Developing Scalable Applications with Actors, WSEAS Transactions on Computers, WSEAS, Vol. 13, (2014), pp. 660-669.
[19]. L. Pace, P. Maggiore, Model-Supported Verification of Space Systems, WSEAS Transactions on Systems, WSEAS, Vol. 16, (2017), pp. 64-68.
[20]. V. Kasyanov, T. Zolotuhin, A System for Structural Information Visualization Based on Attributed Hierarchical Graphs, WSEAS Transactions on Computers, WSEAS, Vol. 16, (2017), pp. 193-201.
[21]. SAP, How to communicate Architecture – Technical Architecture Modeling at SAP, (2015), https://blogs.sap.com/2015/02/11/how-to-communicate-architecture-technical-architecture-modeling-at-sap-part-4/
[22]. A. Spiteri Staines, Matrix Representations for Ordinary Restricted Place Transition Nets, WSEAS Transactions on Computers, WSEAS, Vol 16, (2017), pp. 23-9-29.
[23]. J. Gleick, Chaos, Making a New Science, Penguin, (2008).
[24]. M. Žáček, A. Lukasová , Knowing and/or believing a think: deriving knowledge using RDF CFL, Transactions on Computers, WSEAS, Vol 16, (2017), pp. 202-207.
[25]. N. EL Marzouki, Y. Lakhrissi, O. Nikiforova, M. EL Mohajir, K. Gusarovs, Behavioral and Structural Model Composition Techniques: State of Art and Research Directions, Transactions on Computers,WSEAS, Vol 16, (2017), pp. 39-50.
[26]. Oksana Nikiforova Two Hemisphere Model Driven Approach for Generation of UML Class Diagram in the Context of MDA, e-Informatica Software Engineering Journal, Volume 3, Issue 1, (2009).
[27]. P.L. Tarr, H. Ossher, W. Harrison, M. Stanley, Jr. Sutton. 'N Degrees of Separation: Multi Dimensional Separation of Concerns'. Proceedings of the 21st International Conference on Software Engineering, ICSE '99, LA USA, (1999), pp. 107-119.

Anthony Spiteri Staines" Basic Concepts for Creating Visual Models" International Journal of Computational Engineering Research (IJCER), vol. 09, no. 1, 2019, pp 55-64