EVALUATION OF UML IN PRACTICE
Experiences in a Traffic Management Systems Company

Michel dos Santos Soares and Jos Vrancken
Faculty of Technology, Policy and Management, Delft University of Technology
2600 GA Delft, The Netherlands

Keywords: UML, SysML, Petri nets, Empirical Software Engineering, Software Process Improvement, Action Research.

Abstract: This article is about research performed by the authors into improving the Software Engineering process at a company that develops software-intensive systems. The company develops road traffic management systems using the object-oriented paradigm, and UML as the visual modeling language. Our hypothesis is that UML has some difficulties/drawbacks in certain system development phases and activities. Many of these problems were reported in the literature normally after applying UML to one project and/or studying the language's formal specifications and comparing with other languages. Unfortunately, few publications are based on surveys and interviews with practitioners, i.e., the developers and project managers that are using UML in real projects and are frequently facing these problems. As a matter of fact, some relevant questions were not fully addressed in past research, mainly related to UML problems in practice. The purpose of this text is to report the main findings and the proposed improvements based on other methods/languages, or even considering UML diagrams that are not often used. The research methodology involved surveys, interviews and action research with a system developed in order to implement the recommendations and evaluate the proposed improvements. The recommendations were considered feasible, as they are not proposing to radically change the current situation, which would involve higher costs and risks.

1 INTRODUCTION

This article is about research done to improve Software Engineering processes in practice in a company that develops software-intensive systems. More specifically, the domain is dynamic road traffic management systems, such as road traffic monitoring and control systems. Software process improvement has been recognized in the literature as being capable of reducing costs and augmenting quality and productivity (Herbsleb and Goldenson, 1996), (Rainer and Hall, 2003), (Bannerman, 2008). The current advantages, practices and problems related to the company's processes in general, and in particular with UML, were evaluated.

UML (OMG, 2007) is currently the *de facto* standard object-oriented modeling language in the software industry. It offers good structural representation for the static part of objects and provides several types of dynamic diagrams for behavior specification. UML is used in projects varying from small to large software-intensive systems, in a variety of domains.

Many studies on the application of UML were performed in the past, and a large number of problems, limitations and drawbacks were identified. The language is considered too informal and ambiguous (Beneken et al., 2003), which makes it difficult to use in automatic code generation (Henderson-Sellers, 2005). There are too many diagrams, with some rarely used in practice (Dobing and Parsons, 2006), making it more difficult to chose which one should be used in a specific situation (Anda et al., 2006). In addition, diagrams may overlap in modeling power (Sequence and Communication diagrams, for example), diagrams for important activities in systems development are missing, such as designing graphical user interfaces (Blankenhorn and Jeckle, 2004) and representing non-functional requirements (Soares and Vrancken, 2008b). More specifically, the language diagrams also present problems. Behavior diagrams,

dos Santos Soares M. and Vrancken J. (2009).
EVALUATION OF UML IN PRACTICE - Experiences in a Traffic Management Systems Company.
In *Proceedings of the 11th International Conference on Enterprise Information Systems - Information Systems Analysis and Specification*, pages 313-319
DOI: 10.5220/0001850903130319

such as the Sequence diagram, cannot represent time constraints effectively, (Soares et al., 2008), as they are essentially untimed, expressing only chronological order (André et al., 2007). Use Cases are too informal, with many semantic problems (Simons, 1999). One clear semantic problem is that the *include* and *extend* relationships are considered similar or even the inverse of each other (Jacobson, 2004). In addition, Use Cases can be easily misused when too many details are modeled (Agarwal and Sinha, 2003).

Many of these problems were perceived due to experience, normally after applying UML to some projects and/or studying OMG Specifications and comparing with other languages. Unfortunately, few publications are based on surveys and interviews with practitioners, i.e., the developers and project managers that are using UML in real projects and are frequently facing these problems. As a matter of fact, some relevant questions related to UML problems in practice were not fully addressed in past research. For instance, what do developers think of UML? How do they use UML in their projects? What can be improved? These are among the questions answered in the research that resulted in this article. Our approach is based on surveys and interviews with practitioners that have used UML for several years in many projects.

Despite all the well-known problems, UML also presents many advantages, and is heavily used in the modeling of software-intensive systems both in academia and industry. The language is a combination and adaptation of well-known methods used in the past, which facilitates the learning process. For instance, Class diagrams are considered a superset of Entity-Relationship diagram (Booch et al., 2005), Sequence diagrams are based on SDL's Message Sequence Charts (Collins et al., 1996), and the State-Machine diagram is derived from Statecharts (Harel, 1987). As a generic modeling language, UML can be used in many different projects, domains and methodologies. There are many software tools to support diagrams drawing. By now, the language has been already used for more than 10 years. By no means it will be easy to simply stop using UML, and this article and research do not advocate that. This article proposes that UML can be better employed, adapted and used jointly with other languages and methods. This avoids radical changes, which have well-known drawbacks, such as the necessity of additional training for employees, buying and integrating new software tools in the process, and adapting documentation of legacy systems.

The purpose of this paper is to present the evaluation of how UML is used in a company that develops software-intensive systems, the main difficulties/disadvantages recognized by the company, and propose improvements using other methods/languages.

2 RESEARCH QUESTIONS AND METHODOLOGY

Our hypothesis is that UML is a useful software modeling language that has some difficulties/drawbacks in certain system development phases and activities. We want to investigate the following questions about UML and the development process used at the company.

- Which systems engineering phases and activities present UML-related problems?
- What drawbacks/problems are most commonly observed?
- What can be done to solve these difficulties and drawbacks?
- Which UML diagrams are most suitable for each development activity?

These questions are answered for a company developing systems for a specific domain: road traffic management systems. Nevertheless, general, similar results can be inferred, as the systems considered in the domain are software-intensive systems comparable, in terms of complexity, to systems in other domains.

Rather then dealing only with theory to answer the research questions, we have chosen to apply our own theories in practice. The research methodology was inspired by Action Research, which supports that researchers should test their theories with practitioners in real situations and real organizations (Avison et al., 1999). Action Research combines theory and practice through an iterative collaboration between practitioners and researchers. This collaboration is emphasized, as it will help researchers understand the ill-structured and complex environment of organizations. In each iteration, problems are identified, intervention is performed, and reflections on the results are presented.

The research approach was composed of five steps. First, a questionnaire was performed with some employees in order to understand the company's development context and processes. Then, data were tabulated to serve as input for interviewing employees. The first two steps were crucial to understand the current situation, otherwise every effort for process improvement would be useless. The third step was to present to the company managers a diagnosis of current problems. The diagnosis was based on data from

the survey and the interviews. The next step was to use the diagnosis as input for a series of recommendations for improvements. The final step was to actively apply in practice the recommendations in a project in order to be used as a model for further projects, and to demonstrate the usefulness of the recommendations.

3 CURRENT SITUATION

The company where this study was performed is a software development company with a long standing experience in the development of innovative traffic management software, with applications in every major traffic control center in the Netherlands. The company develops systems according to the 4+1 view model of software architecture (Kruchten, 1995), using UML as the modeling language for all the five views. Implementation is performed using proprietary publish-subscribe middleware and a domain specific language. The purpose of both is to raise the abstraction level when developing applications.

In order to understand the current situation and the difficulties faced in the development process, a survey was proposed for developers and project managers. The survey consisted of open questions and closed statements, in which the respondent could give one of the following answers: 0) Strongly disagree, 1) Disagree, 2) Neutral, 3) Agree or 4) Strongly agree.

Open questions were related to the employee's own career, such as number of years of experience with UML and road traffic systems. Closed statements were related to:

- UML in systems development activities. Example: "UML is useful in the requirements phase".
- Use Case diagrams. Example: "Use Cases are sufficient to be used as the only diagram for Requirements representation".
- UML problems in general. Example: "There are constructions/diagrams missing in UML".
- Development Process. Example: "I would like to change radically the development process in order to improve productivity/quality".

Interviews were performed with selected respondents in order to clarify some of their answers, receive more information about problems in general, and even comment any additional information added by the respondents in the survey. Both the interviews and the survey were used to give a diagnosis of the current situation of the company's development process.

4 DIAGNOSIS

The most important items of the produced diagnosis are presented as follows.

1. The employees suggested simplifications in the current process but also the inclusion of new diagrams to model specific problems. Among the suggested additions, better system overview, showing a complete context of the system but without many details, at least in the first phases of development, was mentioned. This overview should be related more to the customer's viewpoint and less to the software.
2. According to the employees, UML could be combined with other methods/languages in order to diminish potential drawbacks.
3. About Use Cases.
   - Use Case diagrams are used when talking to clients and during software development. This means that Use Cases are used for two completely different purposes in software design. One is to present to stakeholders a high-level model, delimiting the functional system context. The other one is during design and implementation.
   - Use Cases shouldn't be used as the only diagram for Requirements modeling. The employees think that there are some missing constructs that should be complemented by other diagrams.
   - For large, complex systems, it may be difficult to know to which level of detail Use Cases should be modeled.
   - Use Cases are often misused, with too much detail added, incorrectly transforming the diagrams into flowcharts or making them difficult to comprehend.
4. The employees recognize that there are some constructions/diagrams missing in UML, and that they can't model everything they need with the language. Examples cited were:
   - Difficulty of representing continuous processes.
   - Verifying important system properties, such as absence of deadlock in models of distributed systems, is not possible with UML diagrams.
   - Representing synchronous and asynchronous messages is not satisfactory, even with Sequence diagrams. Only changing the shape of an arrow can be confusing, and this is only a representation in a language that cannot be formally checked.

- Another problem with Sequence diagrams arrows is that they have been changed from previous versions to UML 2.0. This can be confusing because legacy systems were documented using older versions.
- Some employees agree with the fact that UML is not capable of representing time accurately. The main problem is that with UML, representing important time constraints, such as initial time and intervals of operations, is very difficult or even impossible.

5. Although problems and missing constructions that would allow more modeling power were recognized by the employees, they think that UML should not be adapted by creating a particular profile for modeling specific road traffic management systems, or they are neutral on this point. UML is already a complex language, and adding new specific constructs for traffic may not be so benefic.
6. Even considering the recognized problems with UML, the employees wouldn't like to radically change the current situation even if that would result in a gain in productivity and/or quality. This is an important issue when trying to improve current processes or including new languages/methods, because this would involve high costs with training and adapting legacy systems with new documentation.
7. It was argued that a serious problem with new standard languages/processes/tools/methods in Software Engineering in general is their high rate of change. Whether a new technology will still exist in the near future is hard to predict. That is one important reason why industry prefers to use proven concepts/technologies that may not be the most advanced, but that have already been used and tested in practice.

For each item, at least one recommendation is proposed in the next section.

5 RECOMMENDATIONS

5.1 Action Planning

Based on the items in the diagnosis section, using only UML for all the five views of the 4+1 model of architecture was recognized as insufficient. It was recommended that other languages/methods should be used, combined with UML, to model specific constructions that are difficult with UML only. We investigated possible languages/methods or approaches in the literature that already tries to solve UML problems. The following recommendations were presented to the company.

5.2 Action Taking

1. Use UML profiles, such as MARTE (OMG, 2008b) and SysML (OMG, 2008a), which are conformant to the same metametamodel – MOF (OMG, 2006), and wouldn't radically change the modeling language used in their development process. SysML is a profile for systems engineering and MARTE for real-time and embedded systems.
2. Include SysML Use Case diagrams to represent not only software, but also systems functionalities. Although having the same syntax, the semantics of UML Use Case models and SysML Use Case models are different. Within UML, normally one Use Case is specified by one or more Sequence diagrams, which represent object communication through time. Within SysML, Use Cases normally model higher levels of abstraction, modeling communication between all types of elements in a system, such as hardware, software and people. Thus, a SysML Use Case, relating to the system's view in general, may be refined into several UML Use Cases, relating only to the software view.
3. UML Use Cases are useful to model functional requirements, but are not able to model other types of requirements. SysML Requirements diagrams can model other types of requirements, including external and the many types of non-functional requirements. Tracing between requirements is also enhanced. Requirements can be related to each other in many different ways (Robinson et al., 2003), which can be modeled with SysML Requirements diagrams and tables. Requirements can be modeled in a tree structure, which is also useful to decompose large systems already in the early development phases. Whenever a requirement is changed or deleted, all related requirements are known, as well as the nature of each relationship, providing the important tracing activity to the designers. The other tracing mechanism is to use the "trace" relationship to cross-cut systems requirements through the whole system development, tracing requirements and more specific/detailed models.
4. Use Activity diagrams to model flow of events. Initially one high-level Activity diagram is used to model the general flow. This diagram can later be refined into more detailed specifications. At the

lowest level, Activity diagrams are used to model complete algorithms to be implemented.

5. Avoid modeling too complicated Use Cases, trying to model them at different levels of detail instead of modeling all functionalities at once in the same diagram. One Use Case can be refined, and another Use Case diagram created based on a higher level one.
6. Apply Petri nets (Murata, 1989) for detailed specification of the process view. The resulting models can be simulated and formally verified using a variety of computer-based tools. This is useful not only to model the behavior of a process, but also to formally verify this behavior. In addition, Sequence and Activity diagrams can be translated into Petri nets, as shown by (López-Grao et al., 2004), (Eichner et al., 2005) and (Soares and Vrancken, 2008a), offering better semantics and executable models.
7. Create small additions to UML and SysML instead of creating huge specific profiles for the traffic domain.

All these recommendations have as final result the introduction of Petri nets and SysML into the 4+1 views of software architecture, avoiding using only UML for all views when designing software-intensive systems, by including a systems language and a formal method.

6 EVALUATION AND FUTURE WORK

Evaluation was done by applying some of the recommendations in a project in the company. On purpose, the chosen case study is a small project. The idea was to produce current company's documents using the new diagrams and methods suggested in order to demonstrate how and where they fit better and how to use them. The development environment was exactly the same used by other developers in other applications, depicted in figure 1. As a matter of fact, the idea is to demonstrate that the introduced modifications are compatible with the current implementation architecture.

The implementation view is based on a layered software architecture, which provides improved modularity and abstraction. One important layer of this architecture is the publish-subscribe middleware (Fiege et al., 2006), on top of the basic software layer. Publish-subscribe middleware provides higher levels of abstraction, hiding the complexity of dealing with

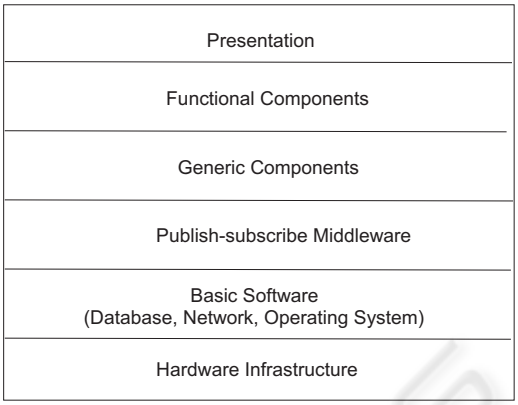

Figure 1: Layered implementation software architecture

a variety of platforms, networks and low-level process communication. Application developers may concentrate only on the current requirements of the software to be developed, and use lower-level services provided by the middleware when necessary. In addition, developers may use a list of existing components and combine them as much as necessary. The Graphical User Interface layer is used mainly only to present the application results.

After the design and implementation, the company's project managers gave comments on the results. According to them, it was clear that SysML Requirements diagrams are more suitable to represent all kinds of requirements than Use Cases, and can represent requirements in a nice structured manner that helps not only during development but also in system maintenance. In particular, the possibility of tracing requirements through the development phases was considered extremely useful. In addition, as the semantics of SysML Use Cases is considered broader than that of the UML Use Cases, improved system overview is achieved, and the detailed specification of software activities is considered separate from systems' activities.

Besides the introduction of SysML, the UML diagrams normally used at the company (Sequence, Class and Use Case diagrams) are still considered useful and will keep being used. In addition, Activity diagrams will be included to the company's development process in two tasks: i) for the high-level specification of processes, ii) which are later refined into more detailed specifications to be implemented. Another interesting comment was related to the company's model of software architecture. The possibility of including a new language in the 4+1 view model of software architecture, instead of changing the model of architecture already in use by the company for the past ten years, was highly appreciated.

Action Research is an iterative process for chang-

ing and further evaluation is necessary. The second evaluation step is to apply the recommendations to a series of projects and compare the results with previous projects. After experimenting the recommendations in larger projects, another survey will be performed with the same employees to assess the advantages perceived by them. In addition, although Petri nets were used for process modeling in the case study, their introduction as a regular method was shown to have more resistance from the company. Finally, SysML allocation tables will be used in larger projects to represent the many relationships that map one model element to another, as for instance, an Activity to a Block.

7 DISCUSSION

Software process improvement has been shown to deliver software products with reduced cost and higher quality. There are many proposed models to improve software processes, such as ISO/IEC 15504, PSP, CMM and CMMI. Normally they are based on changing the way software is developed, for instance, by introducing new methods, additional documentation, and measuring, in terms of costs, defects and development time, the deliverables of phases and activities. Nevertheless, there are many reasons why these software process improvement initiatives fail (Hardgrave and Armstrong, 2005). Perhaps the most common one occurs when large changes that are not compatible with the company or not possible to be implemented, at least at that moment, are proposed. As a result, there will be no support from managers or even from team members. Large changes are not welcome in industry for many reasons: high costs and risks involved, the need for additional training and the existence of legacy systems created using previous technologies. An approach with few changes, maintaining compatibility with the current situation, is more likely to succeed.

In general, using SysML is helpful to bring Systems and Software Engineering together. The language is a UML profile. SysML has been adopted by several different companies, and many vendors are already selling software tools, even with a variety of degrees of integration with UML. As a result, software engineers and systems engineers can communicate through the use of similar languages.

Although SysML and UML share the same name and even syntax for some diagrams, they are not semantically the same. For example, SysML Blocks can be mapped to UML Classes. SysML Blocks are used to model general elements in a software-intensive system, such as all types of hardware (sensors, actuators) and their operations, constraints, values and parts. When the hardware is included as a software object in the system, then it can be modeled as a UML Class.

There is still resistance to apply formal methods in general in industry, and Petri nets are no exception. These methods normally require highly-skilled personnel, which normally increases costs and time. Nevertheless, some research studies were published arguing that there are so many advantages that it is worth trying (Davis, 2005), (Bowen and Hinchey, 2005). In particular, an advantage of Petri nets is the wise combination of mathematical definitions and graphical representation, which may diminish opposition to its application in practice. Therefore, Petri nets provide many advantages, such as formally verifying models using many different techniques, improved semantics and computer-based tools that may be used to draw and execute models.

8 CONCLUSIONS

This article presented the research performed in order to improve Software Engineering processes in practice in a company that develops software-intensive systems. The main idea was to try to improve current processes without drastically changing them.

The company uses UML as the visual modeling language. Although recognized as a language with many drawbacks, UML is generally accepted as useful for many domains and companies. The drawbacks and problems were identified in practice, and solutions were proposed and implemented in one case study. Improvements were considered feasible, as they are not proposing to radically change the current situation, which would involve higher costs and risks. Specific measures, such as the difference in terms of project time and product quality are difficult to give. The improvements are based on the possibility of discovering errors during requirements and design phases, the execution of scenarios by Petri net simulation, and the improved requirements representation and tracing, through requirements modeling using SysML.

Future research will address the advantages of introducing SysML and Petri nets in the company by comparing with the former approach. In addition, creating a specific SysML profile for the company is in study.

REFERENCES

Agarwal, R. and Sinha, A. P. (2003). Object-oriented Modeling with UML: a Study of Developers' Perceptions. *Communications of the ACM*, 46(9):248–256.

Anda, B., Hansen, K., Gullesen, I., and Thorsen, H. K. (2006). Experiences from Introducing UML-based Development in a Large Safety-Critical Project. *Empirical Software Engineering.*, 11(4):555–581.

André, C., Mallet, F., and de Simone, R. (2007). Modeling Time(s). In ACM-IEEE, editor, *10th International Conference on Model Driven Engineering Languages and Systems (MODELS '07)*, pages 559–573, Nashville, TN, USA. Springer Verlag.

Avison, D. E., Lau, F., Myers, M. D., and Nielsen, P. A. (1999). Action research. *Communinations of the ACM*, 42(1):94–97.

Bannerman, P. L. (2008). Capturing business benefits from process improvement: four fallacies and what to do about them. In *BiPi '08: Proceedings of the 1st international workshop on Business impact of process improvements*, pages 1–8, New York, NY, USA. ACM.

Beneken, G., Hammerschall, U., Broy, M., Cengarle, M. V., Jürjens, J., Rumpe, B., and Schoenmakers, M. (2003). Componentware - State of the Art 2003. In *Proceedings of the CUE Workshop Venedig*.

Blankenhorn, K. and Jeckle, M. (2004). A UML Profile for GUI Layout. In Weske, M. and Liggesmeyer, P., editors, *Net.ObjectDays*, volume 3263 of *Lecture Notes in Computer Science*, pages 110–121. Springer.

Booch, G., Rumbaugh, J., and Jacobson, I. (2005). *Unified Modeling Language User Guide, The (2nd Edition) (The Addison-Wesley Object Technology Series)*. Addison-Wesley Professional.

Bowen, J. P. and Hinchey, M. G. (2005). Ten commandments revisited: a ten-year perspective on the industrial application of formal methods. In *FMICS '05: Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, pages 8–16, New York, NY, USA. ACM.

Collins, W. D., Rasch, P. J., Eaton, B. E., Fillmore, D. W., Kiehl, J. T., Beck, C. T., and Zender, C. S. (1996). Message Sequence Charts (MSC. In *ITU-TS Recommendation Z.120*, page 2002.

Davis, J. F. (2005). The affordable application of formal methods to software engineering. *Ada Lett.*, XXV(4):57–62.

Dobing, B. and Parsons, J. (2006). How UML is used. *Communications of the ACM*, 49(5):109–113.

Eichner, C., Fleischhack, H., Meyer, R., Schrimpf, U., and Stehno, C. (2005). Compositional Semantics for UML 2.0 Sequence Diagrams Using Petri Nets. In *SDL Forum*, pages 133–148.

Fiege, L., Cilia, M., Muhl, G., and Buchmann, A. (2006). Publish-Subscribe Grows Up: Support for Management, Visibility Control, and Heterogeneity. *IEEE Internet Computing*, 10(1):48–55.

Hardgrave, B. C. and Armstrong, D. J. (2005). Software process improvement: it's a journey, not a destination. *Communications of the ACM*, 48(11):93–96.

Harel, D. (1987). Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274.

Henderson-Sellers, B. (2005). UML - the Good, the Bad or the Ugly? Perspectives from a panel of experts. *Software and System Modeling*, 4(1):4–13.

Herbsleb, J. D. and Goldenson, D. R. (1996). A systematic survey of CMM experience and results. In *ICSE '96: Proceedings of the 18th International Conference on Software Engineering*, pages 323–330, Washington, DC, USA. IEEE Computer Society.

Jacobson, I. (2004). Use cases - Yesterday, today, and tomorrow. *Software and System Modeling.*, 3(3):210–220.

Kruchten, P. (1995). Architectural Blueprints—The "4+1" View Model of Software Architecture. *IEEE Software*, 12(6):42–50.

López-Grao, J. P., Merseguer, J., and Campos, J. (2004). From UML Activity diagrams to Stochastic Petri nets: application to software performance engineering. In *WOSP '04: Proceedings of the 4th international workshop on Software and performance*, pages 25–36, New York, NY, USA. ACM.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580.

OMG (2006). Meta-Object Facility (MOF) Core Specification Version 2.0.

OMG (2007). Unified Modeling Language (UML), Superstructure version 2.1.1.

OMG (2008a). Systems Modeling Language (SysML) Version 1.1.

OMG (2008b). UML Profile for MARTE, Beta 2.

Rainer, A. and Hall, T. (2003). A quantitative and qualitative analysis of factors affecting software processes. *Journal of Systems and Software*, 66(1):7–21.

Robinson, W. N., Pawlowski, S. D., and Volkov, V. (2003). Requirements Interaction Management. *ACM Computing Surveys*, 35(2):132–190.

Simons, A. J. H. (1999). Use Cases Considered Harmful. In *In 29th Conference on Technology of Object-Oriented Languages and Systems*, pages 194–203. IEEE Computer Society.

Soares, M. S., Julia, S., and Vrancken, J. (2008). Real-time Scheduling of Batch Systems using Petri Nets and Linear Logic. *Journal of Systems and Software*, 81(11):1983–1996.

Soares, M. S. and Vrancken, J. (2008a). *A Metamodeling Approach to Transform UML 2.0 Sequence Diagrams to Petri Nets*, volume 1, pages 159–164. ACTA Press.

Soares, M. S. and Vrancken, J. (2008b). Model-Driven User Requirements Specification using SysML. *Journal of Software*, 3(6):57–68.