STSM Report: Review and analysis of user profiles and cost factors of modelling languages for CPS development

Ankica Barišić
NOVA Laboratory for Computer Science and Informatics
Departamento de Informática, Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa, Portugal
Email: a.barisic@campus.fct.unl.pt

I. Purpose of the Visit

Short Term Scientific Mission (STSM), was carried out in period of 2nd till 8th of September 2018 at Faculty of Electrical Engineering and Computer Science in Maribor, Slovenia, together with dr. Tomaz Kosar, in the context of the MPM4CPS COST Action IC1404.

This report contributes to WG4, and WG1, by reviewing modelling languages and tools in the scope of modelling of CPS, identifying cost factors involved in their use and development, and presenting profile characteristics for users of this tools.

Faced with the challenges associated with costs associated to construction of modelling languages and tools, the difficulty arises in the decision when to use them. Typically, Software Engineering project managers need to take decisions of various kinds in relation to the technologies they will use, such as time spent, distribution of tasks, and human resources that are needed. One of the major problems in project management is the lack of information, or evidence, on the technology that confirms the disadvantages, advantages, costs and risks associated with its appropriate decision [7]. Due to the lack of evidence, software engineers can make wrong decisions with medium and long-term impact.

In the context of modelling languages and tools, a higher initial investment is required which includes domain analysis which helps to understand the context of the problem [10]. It includes consultation with experts in the field and access to individuals who are experts in language development.

II. Related work

A. Methods for Estimating Costs

Predicting the cost of a project development is a responsibility of project managers. They often have to make difficult decisions regarding to the effort and the time which are required to complete a project [20]. Without having a cost model this task becomes complicated, if not impossible to perform with the greatest precision possible. It is necessary to determine, among others, the time required and the number of people who are needed for project development [1], [11].

Cost estimation methods are used for [4]:

- *Budget:* the overall accuracy of the estimation process is the most desired quality;
- *Trade-off and Risk Analysis:* Determine project cost based on decisions (e.g. choice of tools, reuse approaches, among others);
- *Project planning:* An additional important capacity is to provide a the cost estimate by component, phase or activity;
- *Analysis of investment in software improvements:* estimate the cost, as well as the benefits of various strategies used such as the choice of tools, reuse and process maturity.

In the case of modelling languages, it is necessary, for example, to decide for how many people having a different profile are needed to form the development team and how many domain experts are required. These are the two factors, which can greatly influence the success or failure of the project.

There are different cost estimation methods and Boehm discusses each of them [5]. No method is considered to be the better, and it is recommended that several alternative methods be used simultaneously. It is very important to understand their advantages and disadvantages when the objective is to estimate the development effort of a project.

III. Description of work

We reviewed the initial deliverable from WG1, on "Framework to Relate / Combine Modeling Languages and Techniques", and identified the tools which should be added to the lists. Participants marked to which of existing languages team is familiar. Modelling tools and languages are suggested to be added to the list. The team selected the technologies from the list and are willing to help in providing descriptions.

Following a principle of underground theory approach [8], the STSM participant performed individual interviews with the members of the team. The information which was collected was the following:

1) What type of modelling languages or tools did team develop?: e.g. Internal/external; /textual/hybrid/other

2) What is development environment/process?: e.g. Planned/adhoc; Incremental/iterative; Domain analysis; Testing/evaluation; Maintenance.

3) What was application domain and who were the users?: e.g. what is profile of target users?

4) How do you calculate the cost?: e.g. in which phases; the metrics that are taken into concern.

IV. Result

A. Modelling Languages

We identify that the following modelling languages and tools could be added to the list of the tools provided by WG1:

1) MetaEdit+: is an environment for creating and using Domain-Specific Modeling languages [**?**]. The research behind the genesis of MetaEdit+ was carried out at the University of Jyväskylä, as part of the MetaPHOR project. A metamodeling and modeling tool, MetaEdit, had been created by the earlier SYTI project in the late 1980s and early 1990s, in co-operation with a company, MetaCase.

2) ANT-LR: (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. It's widely used to build languages, tools, and frameworks. From a grammar, ANTLR generates a parser that can build and walk parse trees [19].

3) Sequencer: The purpose of the Sequencer was to enable the easier construction of measurement procedures inside the measurement system DEWESoft. The main goal of the Sequencer is to push the development of the application from using DCOM objects to a specialized tool that enables domain experts to develop measurement sequences efficiently in a simple manner, without the need of support from programming engineers. Sequences was developed on top of GME and provide editors in a textual or visual mode, which are customized for application development in the measurement domain [13], [12].

4) Python: The Python (https://www.python.org/) is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

5) Lab View: LabVIEW is systems engineering software for applications that require test, measurement, and control with rapid access to hardware and data insights [3]. The programming paradigm used in LabVIEW, sometimes called G, is based on data availability. LabVIEW integrates the creation of user interfaces (termed front panels) into the development cycle. LabVIEW programs-subroutines are termed virtual instruments (VIs). Each VI has three components: a block diagram, a front panel, and a connector panel. The last is used to represent the VI in the block diagrams of other, calling VIs. The front panel is built using controls and indicators. Controls are inputs: they allow a user to supply information to the VI. Indicators are outputs: they indicate, or display, the results based on the inputs given to the VI. The back panel, which is a block diagram, contains the graphical source code. All of the objects placed on the front panel will appear on the back panel as terminals. The back panel also contains structures and functions which perform operations on controls and supply data to indicators. The structures and functions are found on the Functions palette and can be placed on the back panel. Collectively controls, indicators, structures, and functions are referred to as nodes. Nodes are connected to one another using wires, e.g., two controls and an indicator can be wired to the addition function so that the indicator displays the sum of the two controls. Thus a virtual instrument can be run as either a program, with the front panel serving as a user interface, or, when dropped as a node onto the block diagram, the front panel defines the inputs and outputs for the node through the connector panel. This implies each VI can be easily tested before being embedded as a subroutine into a larger program.

6) NOED-RED: is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways (https://nodered.org/docs/writing-functions) . It is a model that lends itself very well to a visual representation and makes it more accessible to a wider range of users. If someone can break down a problem into discrete steps they can look at a flow and get a sense of what it is doing; without having to understand the individual lines of code within each node. Node-RED consists of a Node.js-based runtime that you point a web browser at to access the flow editor. Within the browser you create your application by dragging nodes from your palette into a workspace and start to wire them together. With a single click, the application is deployed back to the runtime where it is run. The palette of nodes can be easily extended by installing new nodes created by the community and the flows you create can be easily shared as JSON files.

7) Hardware description Language (VHDL): [18] is a specialized computer language used to describe the structure and behavior of electronic circuits, and most commonly, digital logic circuits. A hardware description language enables a precise, formal description of an electronic circuit that allows for the automated analysis and simulation of an electronic circuit. It also allows for the synthesis of a HDL description into a netlist (a specification of physical electronic components and how they are connected together), which can then be placed and routed to produce the set of masks used to create an integrated circuit. A hardware description language looks much like a programming language such as C; it is a textual description consisting of expressions, statements and control structures. One important difference between most programming languages and HDLs is that HDLs explicitly include the notion of time.

B. Costs

During our interview sessions we identify the following costs as most influencing for development of modelling languages and tools:

1) Choice of Modelling environment: The proper choice of the tool to model a CPS can influence the time spent on completing a software project. This is due, essentially, the features that a tool can offer, but it also depends on the time it is necessary to learn how to model.

According to a comparison of the effort required to implement a given modelling language or tool [21], it is obvious that modelling environment may have an influence on the time taken to develop a modelling language. It is observed that the development of a modeling language using the GMF tool, can take up to twenty-four days more to finish in relation to the use of the MetaEdit+ tool.

2) Evolution of modelling languages and tools: The evolution of modelling languages and tools is an important problem, there are articles that discuss it [6], [23]. However, The evolution phase seems to be relatively neglected by the community, since in the mapping study [14], only 9 primary studies, in 361 possible studies, mentioned maintenance phase. In addition to these problems, poor development practices can also be a hindrance [10]. If the concepts of language are very specific, this makes it difficult for the learners to use it (either for use or for development) and, consequently, makes it difficult to maintain them. Given this, due to the several factors stated above, the maintenance of a modelling languages can become a serious and time-consuming problem [16].

3) Learning time: Learning time is dependent on several factors. These include the type of task to be performed and the difficulty experienced by the person moment of realization. In the case of language, there should be at least one person who knows the language in order to facilitate learning of individuals who later come to work with language.

The learning time is directly related to the learning capacity of a human being, which may differ from person to person. This cost can be reduced if language documentation is available, we can neglect the cost of producing *Documentation* for modelling languages and tools. What should be documented, for example, would be the structure and syntax of the language and how to use editors and transformers [22].

4) Validation: This cost reflects the validation of language, reading and analysis of the data model, as well as its learning. Typically, validation software is always a difficult and time-consuming process depending on the complexity of the software system.

There are several techniques that can be used to validate the software (eg validation of the model) [17]. In the context of CPS, for example, domain experts can validate the model, that is, whether it meets the needs identified by stakeholders. Usability refers to the ease of use and acceptability of stakeholders through specific tasks [9]. Usability assessment is a very important factor for a stakeholder. Typically, the usability assessment is done through empirical methods [2], [15].

V. Future collaboration

The list of suggested modelling languages and tools will be provided to the participants of WG1, and after revision added to the ontology provided by the group members.

Based on the provided model of the costs, we plan to obtain more interviews regarding the costs of development of modelling tools.

References

[1] Amit Agrawal, Vaibhav Jain, and Mohsin Sheikh. Quantitative estimation of cost drivers for intermediate COCOMO towards traditional and cloud based software development. In *Proceedings of the Ninth Annual ACM India Conference on - ACM COMPUTE 16*. Association for Computing Machinery (ACM), 2016.

[2] Ankica Barišić, Vasco Amaral, and Miguel Goulão. Usability Driven DSL development with USE-ME. *Computer Languages, Systems and Structures (ComLan)*, ISBN 1477-, 2017.

[3] Rick Bitter, Taqi Mohiuddin, and Matt Nawrocki. *LabVIEW: Advanced programming techniques*. Crc Press, 2006.

[4] Barry Boehm, Chris Abts, and Sunita Chulani. Software development cost estimation approaches — a survey. *Annals of Software Engineering*, 10(1):177–205, 2000.

[5] Barry W. Boehm. *Software Engineering Economics*. Prentice Hall, 1 edition, 11 1981.

[6] Arie Van Deursen and Paul Klint. Little languages: Little maintenance? *Journal of Software Maintenance: Research and Practice*, 10(2):75–92, 1998.

[7] T. Dyba, B.A. Kitchenham, and M. Jorgensen. Evidence-based software engineering for practitioners. *IEEE Softw.*, 22(1):58–65, jan 2005.

[8] Catarina Gralha, Daniela Damian, Anthony I. (Tony) Wasserman, Miguel Goulão, and João Araújo. The evolution of requirements practices in software startups. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 823–833, New York, NY, USA, 2018. ACM.

[9] Andreas Holzinger. Usability engineering methods for software developers. *Communications of the ACM*, 48(1):71–74, 2005.

[10] Steven Kelly and Risto Pohjonen. Worst practices for domain-specific modeling. *IEEE Software*, 26(4):22–29, 2009.

[11] B Kitchenham and NR Taylor. Software cost models. *ICL technical journal*, 4(1):73–102, 1984.

[12] Tomaz Kos, Tomaz Kosar, Jure Knez, and Marjan Mernik. Improving end-user productivity in measurement systems with a domain-specific (modeling) language sequencer. In *ADBIS (Local Proceedings)*, pages 61–76. Citeseer, 2010.

[13] Tomaž Kos, Tomaž Kosar, Jure Knez, and Marjan Mernik. From dcom interfaces to domain-specific modeling language: A case study on the sequencer. *Computer Science and Information Systems*, 8(2):361–378, 2011.

[14] Tomaž Kosar, Sudev Bohra, and Marjan Mernik. Domain-Specific Languages: A Systematic Mapping Study. *Information and Software Technology*, 71:77–91, March 2016.

[15] Deborah J Mayhew. The usability engineering lifecycle. In *CHI'99 Extended Abstracts on Human Factors in Computing Systems*, pages 147–148. ACM, 1999.

[16] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and How to Develop Domain-specific Languages. *ACM Comput. Surv.*, 37(4):316–344, December 2005.

[17] Glenford J. Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing*. Wiley, 3 edition, 11 2011.

[18] Zainalabedin Navabi. *VHDL: Analysis and modeling of digital systems*. McGraw-Hill, Inc., 1997.

[19] Terence J. Parr and Russell W. Quong. Antlr: A predicated-ll (k) parser generator. *Software: Practice and Experience*, 25(7):789–810, 1995.

[20] M. Shepperd and C. Schofield. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, 23(11):736–743, 1997.

[21] Juha-Pekka Tolvanen and Steven Kelly. Model-driven development challenges and solutions - experiences with domain-specific modelling in industry. In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development*. Scitepress, 2016.

[22] Markus Voelter, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart Kats, Eelco Visser, and Wachsmuth. *DSL Engineering: Designing, Implementing and Using Domain-Specific Languages*. CreateSpace Independent Publishing Platform, 2013.

[23] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.