Designing Functional Specifications for Complex Systems

Olga Goubali[1,2(✉)], Patrick Girard[1], Laurent Guittet[1], Alain Bignon[2], Djamal Kesraoui[2], Pascal Berruet[3], and Jean-Frédéric Bouillon[4]

[1] LIAS/ENSMA, 1 avenue Clément Ader, 86961 Chasseneuil, France
{Olga.goubali,girard,laurent.guittet}@ensma.fr
[2] SEGULA Technologies, BP 50256, 56602 Lanester cedex, France
{alain.bignon,djamal.kesraoui}@segula.fr
[3] Lab-STICC, BP 92116, 56321 Lorient cedex, France
pascal.berruet@univ-ubs.fr
[4] ENSM, 38 rue Gabriel Péri, 44100 Nantes, France
Jean-frederic.bouillon@supmaritime.fr

Abstract. For designing complex and sociotechnical (System that strongly interact with humans (e.g., a ship is a large sociotechnical system).) systems, designers are in charge of the functional specification because they have an operational expert knowledge. However, these experts do not usually master the programming knowledge of those who design supervision systems. Complex and sociotechnical systems include supervision systems which comprise monitoring interfaces and associated control codes. In this paper we propose an approach that facilitates functional specification of supervision systems. This approach aims at exploiting Example Based Programming (EBP) to propose a specification tool, which contains a generalization module and an interface generation module. Our tool allows experts who are acting as non-professional software developers to describe high level system functional services from elementary services. These functional services contain elementary interactions and configuration data. Thus, the expert, involved in coding, avoids a lot of errors related to the interpretation of the functional specifications. Our aim is to capture expert knowledge on the system being designed in order to have verified and validated functional specifications, without having to train experts in formal methods.

Keywords: Industrial supervision · Functional specification · Example based programming · Model driven engineering

1 Introduction

Since the 80's, the V-model [21] has become the industry standard designing model. However, it is sometimes difficult to strictly implement this model especially when there are significant changes in specifications, for example new features added by the customer in an advanced project phase. There is a high risk that these changes will affect the system so that it no longer matches the initial requirements as they progress over time. Moreover, it is during the coding phase one often realizes that initial specifications were incomplete, inconsistent, false or unfeasible. The late identification of these defects affects development costs and application evolution.

M. Kurosu (Ed.): HCI 2016, Part I, LNCS 9731, pp. 166–177, 2016.
DOI: 10.1007/978-3-319-39510-4_16

Other approaches such as agile software development methods are proposed to facilitate changes of the initial design. The word "agile" refers to the ability to adapt to changes in contexts and in specifications that might occur during the development process. However, agile software development is mostly used by small teams, adding value to direct communication in a changing environments. Thus these practices are not suitable for all contexts [20].

In the context of large-scale projects such as the design of sociotechnical and complex systems, the number of stakeholders and the diversity of required expertise lead to overall consistency problems and specification misinterpretation. Indeed, errors from misunderstandings are only detected during the system testing phase. In fact, 79 % of failures come from design and implementation services [28], while 72 % of failures are only detected during operational testing [26].

Mixing together Model Driven Engineering (MDE) and the component approach has helped to overcome these problems. This approach aims at raising programming activity abstraction level by using models at the very beginning of the software development process. It introduces models, metamodels and transformations notions as well as representation, "conformity" and "based on" relationships. These terms are detailed in [25]. As for the component approach, it improves the transmission of knowledge through libraries or collections [4]. These two approaches are supported by our Anaxagore tool [4, 12] that enables to automatically generate applications corresponding to a business model, from standard elements libraries and business models. To validate the combination of both approaches, Anaxagore was applied to a concrete case.

The chosen example is a system for the production, storage and distribution of fresh water, onboard a ship, called EdS (Eau douce Sanitaire *in English Sanitary freshwater*). To date, Anaxagore enables to generate a control code and a monitoring interface for this system. The generated monitoring interface only allows basic commands such as "Open" or "Close" a valve. Elaborating high-level functions from this interface leads to element-by-element interactions between the system and the user. Controlling such a system onboard requires triggers (widgets) for high-level control and monitoring, allowing functions to be run more easily. To implement these widgets, we need to describe functional specifications. These specifications are user's sequences of actions on the system, required for performing functions taking into account all possibilities (configurations). The task of the expert in system design is to define these functional specifications. S/he writes these latter in natural language, and then provides them to the designers of the supervision interface and the control-command code. The designers' job is to implement and integrate these specifications into the system. Specification interpretation errors come from the difference of technical knowledge between prescribers (mechanic engineers) and designers (computer system engineers and/or control-command engineers).

We propose in this article a specification tool that facilitates functional specification and automatically generates the widgets for high level functions of complex and sociotechnical systems. To validate our proposition, we evaluate the usability of the system with expert users. Results are analyzed and detailed in this paper.

2 State of the Art

Designing widgets for high-level control and monitoring leads to implement control code and function interfaces. These implementations are based on functional specifications and come to be tedious depending on the complexity of the system. Some works propose automatic generation of the control code, and others, automatic generation of the user interface. Various approaches used for generating control code are surveyed in [5]. Despite their many advantages, these approaches are inadequate for the control laws used to design industrial monitoring for embedded systems on ships. In fact, the generated control code needs to be transformed into normalized language before used in design phase.

In recent years, several works have explored the idea of automatically generating user interface from detailed models of program functionalities, user knowledge, presentation environment etc. This simplified the user interface implementation process by integrating it with the implementation of application logic. Systems proposed by these works came to be known as model-based systems and are surveyed in [23]. The common property of all these systems is that the user interface is automatically generated from a specification. In the literature, several works has been focused on automatic user interface generation from abstract specification. Some model-based systems such as ERGO-CONCEPTOR [22] enable graphical views specification based on detailed description of the appliance. However, this description is tedious and prone to interrogations and omissions. The performance of system degrades as the complexity of user interfaces increases. Other models such as Jade or PUC include no specific detail about graphical references, which keeps each specification small [23]. The generated user interface is rarely sufficient for the entire application because the works focusses on only one aspect of the system design. To address this, for example, ERGO-CONCEPTOR+ [22] combines the knowledge-based system with the formal specification of the application.

Most of the time, describing requirements in natural language to establish specifications is the first step in the development of embedded systems. Specification is a communication base between customers and design teams [7]. Specification documents should be as explicit as possible because they are the reference point for designers of system functional axis [13]. A good functional specification must be correct, unambiguous, complete and consistent. Some formal or semi-formal methods have been proposed to help with respect to these properties.

Formal methods enable expression of specifications with notations and semantics based on mathematical concepts. Baresi gives a survey of formal notations [2], models and techniques for specifying user interface and a lot of researches was published [9, 13]. Formal or semi-formal methods are known for analyzing and validating the specifications, so that the interpretation errors are limited. However, in some industrial sectors, such as shipbuilding, specifications are typically written in natural language [30] as designers of specifications do not have the required technical knowledge for using other languages. Although there are defects detection techniques in the specifications [10], they require a lot of efforts and do not necessarily detect all defects. Despite the use of some of these techniques, undetected errors are still found in specification documents [29].

Usually, the design of control and monitoring applications is difficult because many errors in applications can be traced to defects in their specifications. A main aspect of

user-interface development is the specification of exactly what the user interface has to do. The automatic interfaces generation is a difficult problem because it requires to determine what abstract application specification is needed, how to formalize specifications and to build an interface generator that can design usable interface from those abstract specifications.

In our case, the business model can be considered as low level formal specification because it is a standardized structuro-functional diagram which is used systematically and earlier in the project, and follows system designing lifecycle [4]. However, this specification is not enough to implement high-level application functions. To generate a high-level application, it is important to know what functional information about the application is necessary to create a usable interface. Business experts have this knowledge about applications but they do not have any training to express them in formal languages. They can only describe them in natural language by using business model. The issue is how can we help them to express the specification correctly?

Our goal is therefore to facilitate functional specification of complex systems by proposing a tool that enables business expert without training in programming to create program for system specification using a suitable interface.

3 Proposition

In interactive system design, the functional specification phase is preceded by task analysis. Functional specifications contain a list of the major system functions (high-level requirements) and scenarios of operations. They describe what the system needs to do, from all of its user's viewpoints. However, the clear expression of needs and requirements, and their translation into functional specifications are not easy tasks, despite the importance of the latter step in the design phase [17]. Task analysis enables to have a set of functional requirements expressed by the designer that can be used in design approaches but this model does not avoid errors related to the interpretation of the functional specifications. We define a process for obtaining specifications by using an original approach based on task models and the paradigm of Example Based Programming (EBP) [19].

Expression of functional requirements by the designer follows an iterative process which first step is the formalization of functional specifications. This formulation is based firstly on a review and analysis of the literature [1], and secondly on lessons learnt by analyzing specifications established in past real projects and by interviewing design experts. This formalization helps to identify the necessary information in specifications and which must therefore be expressed by the designer for designing task models.

EBP is an extension of the concept of macro recorders, which allows users to record and replay sequences of actions. However, replaying macros is reduced only to replaying the recorded actions, whereas in the case of EBP, the system provides a generalization of recorded actions to generate actual programs. Systems including EBP techniques record, generalize and replay user's actions through specific interfaces that keep the functional aim of the application. Therefore, EBP systems allow non-computer scientists/engineers to build programs. EBP techniques enable users to automate repetitive tasks [11], to adapt an application to their specific needs [8] and to integrate applications

for developing a solution made to measure [27]. Systems including EBP techniques record, generalize and replay user's actions. An EBP system must provide specific interfaces that can naturally integrate EBP techniques while maintaining the functional goal of the application. A detailed state of the art of this paradigm [19] shows that its implementation takes place in three phases. A first phase of recording enables the designer to specify the behavior expected by the system interactively by using an interface template of the monitoring system. Then, the system creates generalized programs and configurations from the recorded actions. The generalization remains a central and complex problem of EBP. Although various generalization techniques exist, they are not suited to the generalization of configurations in the designing of industrial complex monitoring systems. For example, designing the base of knowledge to use inference methods to generalize configurations can be a time consuming task. The generalization configuration application proposed here is based on graph theory. Firstly the use of a combination of algorithms enables to propose the user with all possible paths that a system can use to achieve the functional purpose requested. These possible paths are validated by business experts, then we combined them using a solver and a configuration algorithm. The goal is to provide an exhaustive list of possible configurations to run the specified functions. A last phase of replay enables the designer to validate the generalized functions.

The described process is then implemented through a proof of concept, which is a specification tool that consists of a specification interface, a generalization module based on graph theory and a module for interface functions generation that exploits the task model data. Figure 1 describes this process in detail with its phases. During the first phase, the expert uses the specification interface to demonstrate examples of sequences execution for all functions (examples of functional specifications). Two types of generalizations are made during the second phase: the generalization of programs and the generalization of configurations. The third phase enables the use of generalized configurations and tasks models to provide interface models for launching the system functions. The generated possible

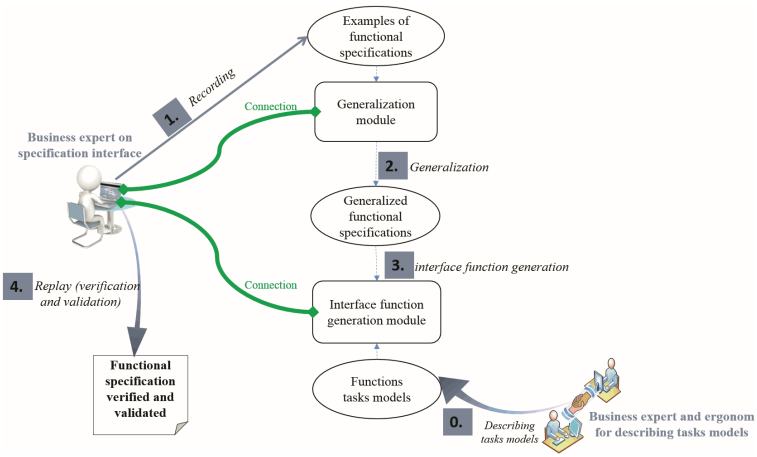

Fig. 1. Specification tool

configurations and functions interface models must then be verified and validated by the expert, in the last phase (4. Replay: verification and validation).

The specification interface is well-known by the expert and integrates the described techniques. This interface has been used for specifying functions of the EdS system taken as example here.

4 Case Study

For our case study, 7 functions need to be specified (transfer, treatment, embedded distribution, distribution from quay, production, loading and unloading) and this implies the description of a total of 73 unit configurations. In fact, given the architecture of the system, each function can be performed according to several configurations. Furthermore these configurations do not take into account the possibility of performing several functions simultaneously. The expert's task is to define these 73 functional specifications as well as simultaneous executions. S/He provides them to the designers of the supervision interface and the control-command code. The designer's job is then to implement and integrate these specifications into the system.

Figure 2 shows an extract of the business model (synoptic diagram) of the considered system. The described system is composed of several elements: tanks, 2-ways valves, water pumps, 3-ways valves, and a chlorination module for water treatment, respectively numbered from 1 to 5. For example, we can consider the function that can transfer water from a tank (St1) to another (St2) via one of the pumps (H1, H2 or H3). Each pump is isolated from the tank St1 by 2-ways valves (V2VM03, V2VM05 and V2VM07). The water transferred is sent to the tank St2 via 3-ways valves (V3VM01, V3VM02 and V3VM03), the chlorination module (TRCH) and V2VM02 valve. The implementation of this function requires the definition of sequences of actions made by the user on the circuit for achieving the function. These are the "functional specifications".

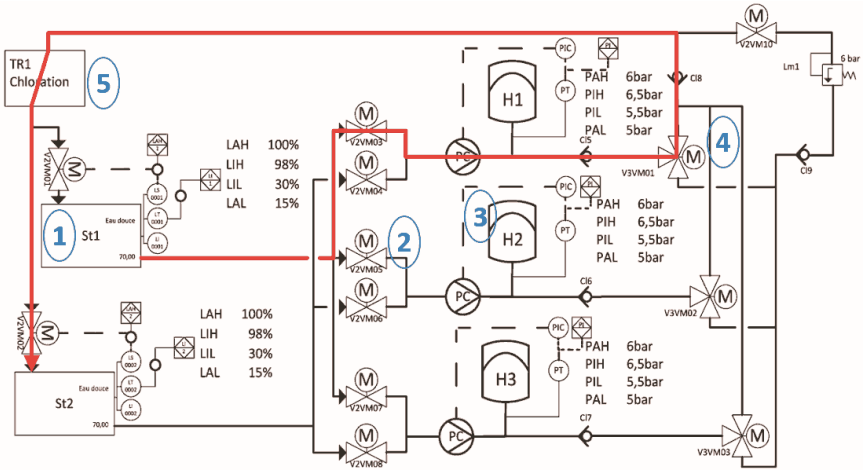

Fig. 2. A function on an extract of synoptic diagram of EdS system

The proposed solution aims at offering the opportunity to the expert to express the functional specifications through an interface that contains elements of the system and integrates EBP techniques.

To test the feasibility of our approach we applied it to the specification of EdS' system functions. The design of the specification interface (Fig. 3) is based on a recorder/replaying (RR) generic model adapted with information from task models, and on graphical views of system components to design, as stored in the standard elements library of Anaxagore.

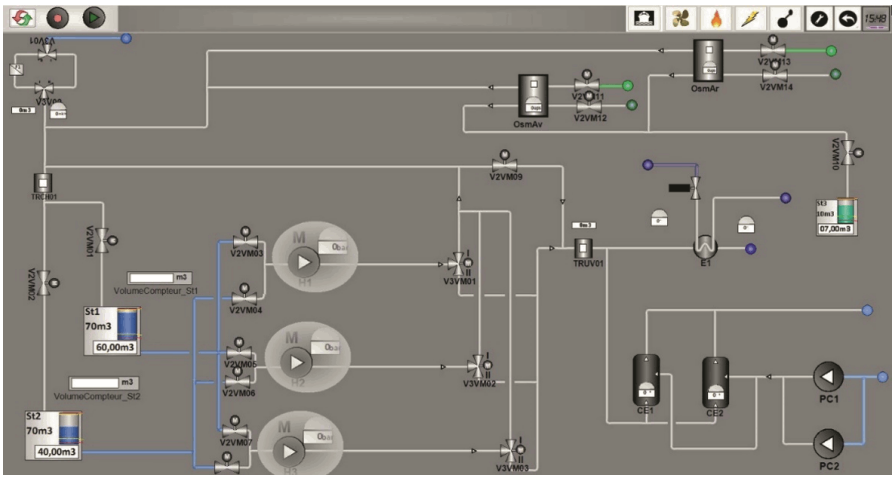

Fig. 3. Specification interface designed for EdS system

4.1 A Recording Model for Specification Interface

EBP techniques (recording, generalization and replay) are implemented through a RR (Recorder Replaying) model present on specification interface. This model is designed with Panorama E2 SCADA software [12] in the form of views. These views show the recorder and the replaying commands (at the top left of Fig. 3).

The recorder command put the system into recording mode and memorize all the expert's actions. It is used for recording examples of sequences of execution for all functions. The system generalizes the recorded examples to generate an executable program in a different context. The replaying command can then replay the recorded examples and their variants. The generic nature of this model allows its integration into any monitoring system. However, setting the adequate parameters is required with the information from the task models of the system to conceive.

4.2 Using the Specification Interface for Describing Functional Specifications

Obtaining functional specifications of a system from the implemented specification interface follows the three steps of EBP: recording, generalization and replay. During the recording step, the system records the whole sequence of actions made by the expert and

the state of the elements manipulated to perform the functions. For it to be generalized, two examples must be recorded. A generalization system, integrated to the specification interface, uses these two examples to produce a generic program and all other possible configurations for the function. Verification and validation of these configurations has to be performed during the replaying phase. Validated configurations and generic programs are then used to generate the monitoring interface and the control-command code that contain high-level system functions.

The implementation of the interface specification enabled us to validate our approach of using EBP to describe functional specifications of a system. To validate the proposed interface, we have evaluated the usability of the system with real users to identify required modifications, to verify our choice a posteriori, to analyze behavioral data, critics and feedbacks on the interface and to involve users (system designers) who have an expert knowledge of the system in designing the specification interface.

5 Evaluation

The evaluation method used is semi-structured interviews, aimed to collect specific and qualitative information. This technique is often used for conducting exploratory studies to improve knowledge of a field of study which main themes are familiar to experts but present aspects requiring in-depth study.

The semi-structured interview was centered on the theme of the functional specification of complex systems. The interview guide used for our experiment is articulated around questions related to three main themes. They have been defined on one hand with respect to the knowledge that we had and wanted to acquire about the subject and on the other hand with respect to the use and the improvement of our tool. The experimental protocol and results analysis are described in this section.

5.1 The Protocol

The experiment took place with 5 participants at ENSM[1], all with navigation experience and functional knowledge of the involved type of systems. The equipment used for the experiment included a 23-inch screen to display the specification interface, a mouse and a keyboard. First, the synoptic diagram of the EdS system was presented and explained to them. To verify that they have a good knowledge of this kind of system, it was asked them to verbally describe the diagram (what is this diagram, how it works…). They were also asked to describe the steps to achieve the Transfer from a tank to another, using the same diagram. Next, we simulated with them task models (two task models describe how to achieve the transfer with and without high-level command) through the Prototask tool [18]. The objective of this part of the test was to check whether the method used to define (specify) the transfer function, and then to launch (in use) is correct. Participants were trained for about 15 min with the use of the specification interface (the part of the interface they do not know); then they were required to use this interface to specify the

[1] ENSM: École Nationale Supérieure Maritime – French Maritime College.

"Transfer" function they had previously described verbally. Finally, when they had finished using the interface, they answered a questionnaire that enabled us to get their feedback in order to improve the interface.

5.2 Presentation and Interpretation of Results

Semi-structured interviews enabled participants to give their perceptions of functional specification process. Most often, their remarks were general and not necessarily specific to the application case; this allowed us to obtain a general feedback on the functional specification approach. Describing and analyzing of the transcripts of the voice recordings during the experiment gave us an initial response. The described tasks for performing transfer by using a command were all validated unanimously. We were able to verify compliance of task models which were used in specification interface designing. The various steps proposed in the interface to achieve the specification of a function have been validated. For each step, a widget is proposed to begin the actions. Each widget is named according to the expert tasks at the phase. The action intuitiveness related to the employed themes for the tasks of each phase is summarized in Fig. 4.

On a scale of 1 to 10, with 1 for "very easy" to 10 for "very complicated", the difficulty of using the specification interface is 4. All participants reported that the use of interface became much easier after specifying the first function.

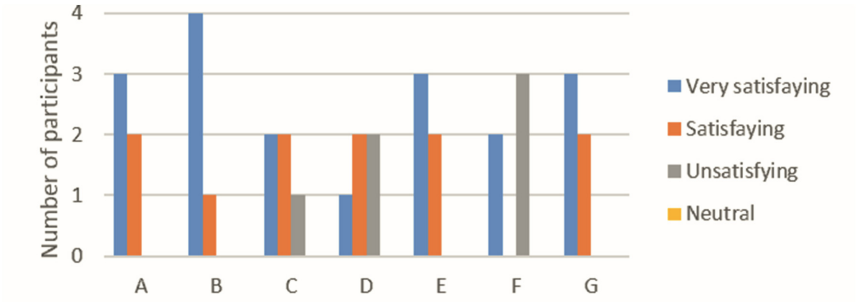

Fig. 4. Evaluation of widgets

The analysis of the experimental results enabled to confirm that our approach is an original solution welcomed by all participants. The use of the interface has been more or less intuitive for all. To complete our analysis, ergonomic expertise was carried out based on the Jacob Nielsen's ten heuristic principles [24] and the criteria of Bastien and Scapin [3]. The information from this analysis will be used to improve the interface and make it more intuitive for the user. Then, the specification interface will be submitted to more stringent tests based on a SUS (System Usability Scale) questionnaire.

6 Conclusion and Future Work

The presented work provides additional support in the process of a system functional specification. Our goal is to reduce specifications effort while getting verified and validated functional specifications through an interface familiar to designers. The introduction of EBP techniques in a specification tool empowers the expert to express functional specifications of the system s/he is designing by simple clicks on an interface; in the same way s/he would describe the system verbally.

The use of this interface by experts in design, but not in software development, solves communication and interpretation problems, which can significantly reduce functional specification design time of a system, hence the project timeline and costs. However, designing specification interface can be time consuming depending on the complexity of system. Current works aim at reducing this design effort by offering methods and tools to automatically generate the specification interface.

Although the specification interface has not yet been generated, its implementation has enabled us to validate our approach of using EBP for describing functional specification of a system. Usability (ISO 9241-11 and ISO-13407) of the specification tool was demonstrated during the semi-structured interviews with users.

In future work, the various defined models will be used to generate the specification interface. From the business model previously used, the library, tasks models and a RR model, information association methods and derivation models will enable to generate gradually, in addition to existing models, the specification interface. Specification interface generation will follow the same steps as those of the low-level interface [4]. We define a design flow based on the principles of MDE, which will enable us to implement our entire approach to generate operational models. The enhanced specification interface with tests results and ergonomic expertise, as well as high level interface generated from these specifications will be evaluated.

Acknowledgements. We would like to thank teachers of the ENSM for their participation. We are grateful to Eric Le Bris, Line Poinel and Davy Rodier for their help.

References

1. AFNOR NF X50-151, Management par la valeur et ses outils, analyse fonctionnelle, analyse de la valeur, conception a objectif designe. French National Standards
2. Baresi, L., Orso, A., Pezzè, M.: Introducing formal specification methods in industrial practice. In: Proceedings of the 19th International Conference on Software Engineering, pp. 56–66. ACM, May 1997
3. Bastien, J.M.C., Scapin, D.: Ergonomic Criteria for the Evaluation of Human-Computer interfaces. Institut National de recherche en informatique et en automatique, France (1993)
4. Bignon, A., Rossi, A., Berruet, P.: An integrated design flow for the joint generation of control and interfaces from a business model. Comput. Ind. **64**, 634–649 (2013)
5. Bévan, R.: Approche composant pour la commande multi-versions des systèmes transitiques reconfigurables (Doctoral dissertation, Lorient) (2013)

6. Bollin, A., Rauner-Reithmayer, D.: Formal specification comprehension: the art of reading and writing Z. In: Proceedings of the 2nd FME Workshop on Formal Methods in Software Engineering, pp. 3–9. ACM, June 2014
7. Clarke, E.M., Wing, J.M.: Formal methods: state of the art and future directions. ACM Comput. Surv. (CSUR) **28**(4), 626–643 (1996)
8. Coutaz, J., Calvary, G., Demeure, A., Balme, L.: Interactive systems and user-centered adaptation: the plasticity of user interfaces. In: Computer Science and Ambient Intelligence, pp. 147–202 (2012)
9. Dix, A.: Formal Methods for Interactive Systems. Academic Press, London (1991)
10. Fagan, M.E.: Design and code inspections to reduce errors in program development. IBM Syst. J. **15**(3), 182–211 (1976)
11. Girard, P., Patry, G., Pierra, G., Potier, J.-C.: Deux exemples d'utilisation de la programmation par démonstration en conception assistée par ordinateur. In: Revue Internationale de CFAO et D'informatique Graphique, vol. 12, no. 1-2, pp. 169–188 (1997)
12. Goubali, O., Bignon, A., Berruet, P., Girard, P., Guittet, L.: Anaxagore, an example of model-driven engineering for industrial supervision. In: Proceedings of the 2014 Ergonomie et Informatique Avancée Conference-Design, Ergonomie et IHM: quelle articulation pour la co-conception de l'interaction, pp. 58–65. ACM, October 2014
13. Harrison, M., Thimbleby, H.: Formal Methods in Human-Computer Interaction. Cambridge University Press, Cambridge (1990)
14. IEEE Guide for Information Technology - System Definition - Concept of Operations (ConOps) Document. IEEE Std 1362-1998. The Institute of Electrical and Electronics Engineers, New York, p. 21
15. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process, vol. 1. Addison Wesley, Reading (1999)
16. Khalil, C., Fernandez, V.: Agile management practices in a "lightweight" organization: a case study analysis. J. Modern Proj. Manage. 1(1) (2013)
17. Knight, J.C., Brilliant, S.S.: Preliminary evaluation of a formal approach to user interface specification. In: Till, D., Bowen, J.P., Hinchey, M.G. (eds.) ZUM 1997. LNCS, vol. 1212. Springer, Heidelberg (1997)
18. Lachaume, T., Patrick, G., Laurent, G., Allan, F.: ProtoTask, new task model simulator. In: Winckler, M., Forbrig, P., Bernhaupt, R. (eds.) HCSE 2012. LNCS, vol. 7623, pp. 323–330. Springer, Heidelberg (2012)
19. Lieberman, H., Paternò, F., Klann, M., Wulf, V.: End-User Development: An Emerging Paradigm, pp. 1–8. Springer, Netherlands (2006)
20. Martin, R.C.: Agile Software Development: Principles, Patterns, and Practices. Prentice Hall PTR, Upper Saddle River (2003)
21. Mathur, S., Malik, S.: Advancements in the V-model. Int. J. Comput. Appl. **1**(12), 30–35 (2010)
22. Moussa, F., Riahi, M., Kolski, C., Moalla, M.: Interpreted petri nets used for human-machine dialogue specification. Integr. Comput. Aided Eng. **9**(1), 87–98 (2002)
23. Nichols, J.: Automatically generating high-quality user interfaces for appliances (Doctoral dissertation, Hewlett-Packard) (2006)
24. Nielson, J.: Heuristic evaluation. In: Usability Inspection Methods, 17 (1), pp. 25–62 (1994)
25. OMG. MDA Guide version 1.0.1. OMG (2003)
26. Pham, H.: System Software Reality (Spring Series in Reality Engineering). Springer-Verlag, New York (2005)
27. Piernot, P.P., Yvon, M.P.: A model for incremental construction of command trees. In: HCI 1995, pp. 169–179. Huddersfield, England (1995)

28. Sourisse, C., Boudillon, L.: La sécurité des machines automatisées: Techniques et moyens de prévention opératifs, systèmes de commandes, utilisation des machines. Institut Schneider Formation (1997)
29. Trudel, S.: Using the COSMIC functional size measurement method (ISO 19761) as a software requierements improvement mechanism. Université du Québec (2012)
30. Weigers, K.E.: Software Requirements, 2nd edn, p. 350. Microsoft Press, Redmond (2003)