Contract-based design in controller development and its evaluation

Pontus Boström[1], Marta Pląska[1], Mikko Huova[2], Matti Linjama[2], Mikko Heikkilä[2], Kaisa Sere[1], Marina Waldén[1]

[1]Department of Information Technology
Åbo Akademi University
Joukahaisenkatu 3-5
FIN-20520 Turku
Fax: +358 2 215 4732
{pontus.bostrom, marta.plaska, kaisa.sere, marina.walden}@abo.fi

[2] Department of Intelligent Hydraulics and Automation (IHA)
P.O. Box 589
FIN-33101 Tampere
Fax: +358 3 3115 2240
{mikko.huova, matti.linjama, mikko.heikkila}@tut.fi

1. Introduction

Software is present in many application areas and is becoming an integral part of mechanical appliances. Therefore cost-effective development of reliable control software has become more critical than ever. In this paper we describe a lightweight formal approach for control systems development and an application of the method to the development of a part of a controller for a digital hydraulic system. We simultaneously evaluate the impact of the used methodology on the development and on the final system. Digital hydraulics involve advanced controllers to achieve high performance and to realize sophisticated characteristics, such as energy saving. This affects the software of the hydraulics controllers, which is therefore highly complex when compared to more traditional controllers. Therefore adequate software development techniques have to be used in order to manage the complexity and to ensure the reliability and functional correctness of the developed system.

Model-based design has become a popular method for development of embedded control software. In this design method, a simulation model of the controlled system is also constructed. This enables fast and cheap analysis of the performance and correctness of the controller. Simulink is one of the most popular tools for model-based design at the moment. This is due to its user-friendly graphical modelling language, simulation tools and its large library of ready-made components. However, Simulink lacks tools and concepts for stepwise design and modular techniques to analyse system correctness. These features would be desirable to better reason about the correctness of the constructed system in a more manageable way and thereby increase the reliability of the system.

To enable stepwise design and modular analysis of correctness, we have introduced contract-based design in Simulink [1] [2]. Contracts here refer to pre- and post-conditions for programs that describe the assumptions the program makes about its environment and the results it can compute. Contract-based design is a well-known technique for object-oriented software development [3]. Our aim has been to transfer those principles to Simulink. To validate that Simulink models satisfy their contracts, we have developed formal analysis techniques based on action systems [1] [4] [2]. Formal reasoning about Simulink models has also been investigated in several notations before. There is a translation to Lustre [5] that can handle a large subset of Simulink. Another formalisation is a translation to Circus [6]. This translation can also handle a relatively large subset of Simulink. The focus is there on analysis of models and refinement of models into possibly parallel code. Neither of those works considers contracts as an integrated part of the formalisation. However, contracts could be introduced there also.

This paper describes a practical application of contracts on the development of a part of a digital hydraulics controller using Simulink. Contracts are used as an aid for structuring the system and for analysis of the system in the form of testing and design reviews. The case study demonstrates positive impact of using contracts on the quality of the resulting control software. Only an overview of the development method and application of it to a case study is presented here. A complete paper that gives a more thorough presentation of the topic has been submitted to a journal [7].

To ensure the quality of the constructed system, quality measurements should be performed. Quality measurements should be, and often already are, a part of the development cycle [8]. Since control systems are becoming more software intensive nowadays, we study and evaluate the impact of lightweight formal methodology in perspective of quality of final product and its development process. We also research the system's complexity characteristic, as it influences the schedule, costs and risks of the development, as well as

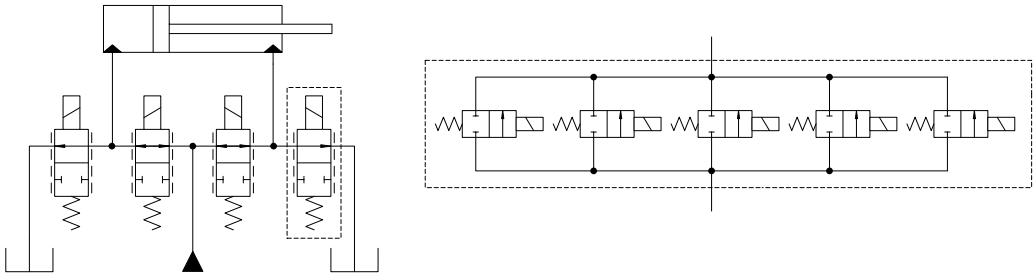
system's understandability [9]. Additionally, complexity has an impact on reliability and cost management of the software process [10].

The paper first shortly describes the application area of the case study, i.e. digital hydraulics. Then a presentation of Simulink and contracts is given. The development steps carried out to create the software for the system in the case study are then presented. Subsequently, evaluation of the development process and the software quality, with a special focus on the complexity of the system is given. Finally, we conclude and give directions for our future work.

2. Application Domain and Case Study Description

The evaluation of applicability of our development method was done using a case study from the area of digital hydraulics. Digital hydraulics is a rather new area of fluid power, which aims at reducing the complexity of the mechanical parts by improving control algorithms. One of the objectives of digital hydraulics is the cost-effectiveness and ease of manufacturing of these parts. However, a sophisticated controller is needed to obtain good performance of the system. Therefore, the need for high quality control software in digital hydraulic systems arose.

Hydraulic systems are widely used in industrial and mobile applications which require high forces or high power to weight ratio. One typical application of digital hydraulics is hydraulic cylinder control, which traditionally have been achieved using complex analogue proportional and servo valves. Digital hydraulics makes use of simple on/off-valves connected in parallel to achieve similar or better performance [11]. Figure 1 illustrates a digital hydraulic system. The figure (b) shows a Digital Flow Control Unit (DFCU) with five on/off-valves connected in parallel. The figure (a) shows digital hydraulic valve system with four DFCUs for controlling a hydraulic cylinder. Digital hydraulic valve systems can produce different control modes and thereby save energy, which is an important benefit as the efficiency of traditional hydraulic systems is usually considered to be low [12]. The digital hydraulic system also has high fault tolerance when compared to traditional systems, which use only one valve to control each cylinder [13]. To achieve fault tolerance, suitable control algorithm has to be used together with some method to detect faulty valves.



(a) (b)

Figure 1. Cylinder drive with a 4- DFCU digital hydraulic valve system (a), as well as a hydraulic diagram of a DFCU consisting of 5 on/off-valves (b).

A relatively complex model-based control algorithm is used in order to be able to utilize all features of the digital hydraulic system. The objective of the controller is to find the optimal configuration of opened and closed valves. The controller uses a steady-state model of the valves and cylinder to be able to estimate the behaviour that would eventually result from the different valve control combinations. There are usually a large number of possible combinations that should be analyzed to find the optimal control solution. For example, a valve system with twenty valves has over one million different control combinations. Several techniques are used to obtain an optimal control configuration using only relatively light-weight calculations that do not require high performance computing hardware.

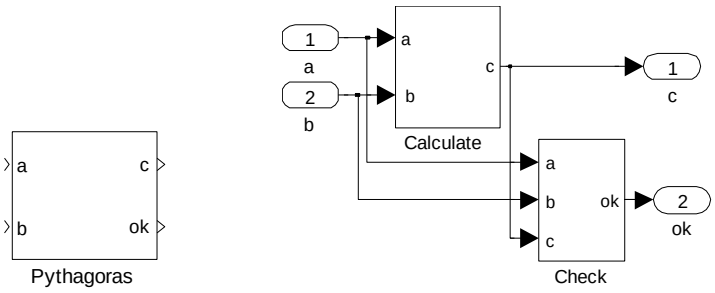
The correct operation of the controller is critical because hydraulic systems are usually heavily loaded and hazardous if malfunctioning. The control algorithms have previously been developed in Simulink without using a structured development method. However, there were problems with software quality and maintainability.

3. Overview of Simulink and contract-based design

Simulink is a part of the MATLAB environment developed by Mathworks Inc. It is a graphical language, where behaviour is described with dataflow diagrams. A diagram contains *functional blocks* that describe how data is manipulated. These blocks are also often *parameterised* to make them more flexible. Blocks can also contain memory, which means that the output of these blocks depends also on this internal state. The blocks are connected by *signals* that describe how data flows between them. The connection points for signals in the blocks are called *ports*. A diagram can be hierarchical, where the functional blocks are organised into *subsystem blocks*. A subsystem block can contain any number of ports for input and output of data to and from the subsystem. An example of a Simulink diagram is given in Figure 2 (b). This diagram contains two subsystem blocks: *Calculate* and *Check*. The blocks with rounded corners correspond to the ports of the subsystem they are in. They enable communication over the subsystem boundary.

The subsystem block in Figure 2 (a) contains a subsystem to calculate the length of the hypotenuse in a triangle according to Pythagoras theorem, $c = \sqrt{a^2 + b^2}$. In Figure 2 (b), the subsystem *Calculate* calculates the result *c*, while the subsystem *Check* checks if the calculated solution is accurate enough. If the solution is good enough then *ok* is set to *true* otherwise *ok* is set to *false*. The content of subsystems *Calculate* and *Check* is shown in Figure 3 (a) and (b), respectively.

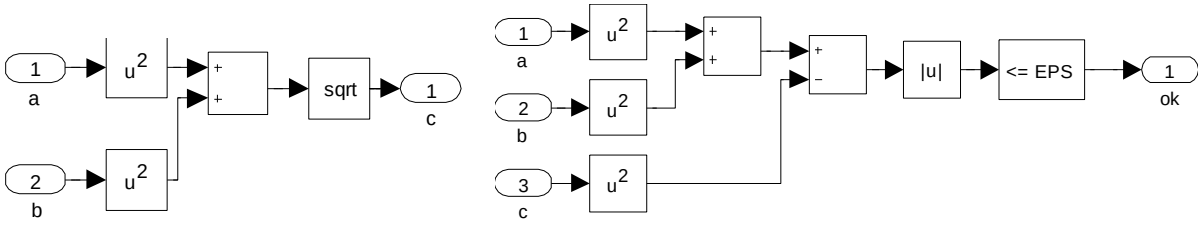
The idea of contracts from object-oriented systems can be applied to Simulink diagrams. The unit that is described by contracts is here the subsystem block. The idea with contracts is that they give a more high-level description of the subsystem block behaviour than the subsystem diagram. This enables reasoning about diagrams in a more modular fashion. The correctness of a diagram can be analysed in terms of the contracts of the subsystem blocks that it contains. The detailed content of the subsystems does not have to be known. The contracts also document the division of responsibility between subsystems.



(a) (b)

Figure 2. Subsystem that uses the Pythagoras theorem to compute the length of the hypotenuse (a), as well as the content of that subsystem (b)

Each subsystem can be described by a pre-condition that states what it assumes of the values on its in-ports and a post-condition that states which values can be produced on its out-ports. Assume we have a subsystem M_s with a list of in-ports p_i, list of out-ports p_o, as well as block parameters c. The pre-condition is a predicate of the form $Q^{pre}(p_i,c)$. The post-condition is of the form $Q^{post}(p_o,p_i,c)$. Often a subsystem or a diagram will not function correctly with arbitrary block parameters. To describe the valid values of block parameters c, a predicate $Q^{param}(c)$ is used.



(a) (b)

Figure 3. The content of subsystem *Calculate* (a) and subsystem *Check* (b) in Figure 2

Table 1. Contracts of the subsystems presented in Figure 2 and Figure 3

Subsystem	pre-condition	post-condition
Pythagoras	$a \geq 0 \wedge b \geq 0$	$ok \Rightarrow \mid a^2 + b^2 - c^2 \mid \leq EPS$
Calculate	*true*	*true*
Check	*true*	$ok \Rightarrow \mid a^2 + b^2 - c^2 \mid \leq EPS$

Consider the subsystem for calculating the length of the hypotenuse in a triangle from Figures 2 and 3. The contracts for the different subsystems are given in Table 1. The subsystem *Pythagoras* also has one block parameter *EPS* that gives the desired precision of the computation. The contract for this block parameter is that *EPS* should be greater than zero, $Q^{param} \triangleq EPS \geq 0$. The complete contract of the subsystem *Pythagoras* states that the inputs *a* and *b* should be greater than zero. If this condition is satisfied the subsystem will ensure that the result has been calculated with desired precision if *ok* is set to true. The post-condition subsystem *Calculate* states that it will compute any value, since the precision of the calculation is not known, which is only modelled coarsely here. The subsystem *Check* then checks that the solution is accurate enough and sets the port *ok* accordingly. Note that here it is assumed that all ports have the type double. The contracts encode design decisions about division of responsibility in a clear way. The contract clearly states what inputs can be given to a subsystem and what outputs the subsystem will produce in response to those inputs.

To fully benefit from contracts it should be possible to prove that a subsystem always satisfies its contract. Furthermore, it should be possible to analyse *complete Simulink models*, i.e. subsystems that do not have any in-ports. In this case, we need to show that all of the subsystems in the model are connected in such a way that every pre-condition and post-condition are satisfied. To achieve these goals, we have analysed [2] [4] the Simulink diagrams using refinement calculus and action systems [14] [15] [16]. In the case study discussed in this paper, the validation has been carried out using testing. However, the testing was based on the rules of correctness given by the theory.

4. Development process

In this section we present an outline of the development process for a part of a controller for a digital hydraulics system. The developed subsystem is an important part of the controller and is used to evaluate the application of contract-based design in the MATLAB/Simulink environment [17]. The developed subsystem is complex enough to generate realistic data about the development method. The subsystem presented here is a re-engineered version of an old subsystem with the same basic functionality, but it is also extended with several new features.

The design process started from gathering the requirements of the subsystem. First, the subsystem has to be able to choose the correct control solution according to several criteria that different applications of the valve system has, e.g., find balance between accuracy of velocity and pressure tracking, reduce unnecessary switching of the valves and to minimise the energy consumption. The re-engineering was also aiming at making the subsystem suitable for a broad range of applications by increasing the configurability, as well as making the user interface easy to use. Since the functionality of the controller was extended, the system became more complex.

After gathering the requirements, a first high-level specification of the subsystem was created. It consisted of parameters, inputs and outputs together with contracts that described their properties. Requirements for the parameters and inputs were defined using parameter- and pre-conditions. The functionality of the application was guaranteed using post-conditions for the outputs.

After the high-level specification of the subsystem was completed, it was then refined into a subsystem diagram where the functionality was decomposed into five subsystems. Each of these subsystems was also described by contracts, which were written following the same pattern as in the high-level specification. The contracts were used as a basis for allocating functionality to each subsystem and to describe the assumptions made for each of them. Here contracts were used to analyse that the diagram indeed satisfies the contract of the high-level specification via design reviews.

During the design process the design reviews were performed multiple times by a team of developers. At this point contracts gave a good basis for discussing and analysing the design at an early development stage. Moreover, they helped to gain confidence that the specification was correct and complete, as it was done accurately using contracts. This in turn influenced the quality of the final control algorithm, as shown in Section 5 in more detail.

After the refinement phase, each of the five subsystems was implemented according to the specification documents as Simulink diagrams. Afterwards, unit testing was performed for each subsystem, thus enabling testing activity before finalising the implementation phase. The unit tests checked that the subsystems conformed to their contracts. The pre-conditions described valid test-data and the post-conditions described the valid outputs of the subsystem. The outputs from a subsystem were checked after the execution of each test and the defects could be detected by checking if the post-conditions were violated. Simulation of the complete controller with a model of a mobile boom test system allowed evaluation of its performance. Testing and simulation were considered as sufficient for confirming the correctness of the subsystem. Proofs of correctness were omitted, as being too time and effort consuming considering the extra confidence in the system that would have been gained.

5. Evaluation of development

Quality measurements were done for the development process and for the constructed subsystem. The study of the quality of *final subsystem* was done with respect to defect-related metrics, size of the system and its structure, which all influence reliability. We also consider system's complexity, since it impacts its maintainability. Complexity is also a major factor when talking about testing coverage. The evaluation of the *development process* included the analysis of its distribution over time, as well as the examination of activities regarding the defect handling (defect removal). The aim is to investigate the impact and suitability of the applied methodology on system development. The data collection for our examination was done in parallel with the development, thus enabling thorough quality evaluation.

The number of defects in the system is considered as one of the most important aspects of quality. We have measured the number of defects and their distribution over the development phases with respect to their origin and removal phase, excluding the severity level classification. Here by defect (fault) we understand an anomaly in a product. In the developed subsystem eight defects in total were detected. The small defect density, equal to 0.55 defects per thousand of generated lines of code (kGLOC), was not only a result of methodology used, but additionally experienced developers, design reviews, code generation and relatively small system size (14428 GLOC in the C language, 846 corresponding Simulink blocks). For comparison, in an earlier development, where contract based design was only introduced as a new development approach, the defect density was equal to 1.38. A critical system, e.g. the air traffic control system investigated in [18], that uses formal methods and C language, has 1.25 defects per thousand of non-commented source code lines (NCkSLOC) after delivery.

The information about defect distribution is particularly interesting, as it shows the impact of the methodology on the development process and on the cost of handling defects. In our study the defects originated from the specification and refinement (3) phase and programming phase (5). The defect discovery was as follows, programming phase: 3, unit testing: 4, system testing: 1. It is worth noticing that only one defect was found at the last stage of the development. It is generally thought that the earlier a defect is found the cheaper it is to fix it [19]. Therefore the final result is very good, as there were no defects found after the system was deployed. However, the system has not been used extensively yet, therefore some defects might still be latent. The coverage of the tests was also not measured and therefore the effectiveness of the test cases in finding defects cannot be guaranteed.

The same part of the controller has been developed earlier using an unstructured development methods. The system developed in this paper has been compared to that system. We computed the *total complexity* as a sum of *structural* and *data complexity* for the developed subsystem [20]. We observed relatively high structural complexity (dependent on quantity of subsystems and connections between them). This was caused by structuring the system into numerous subsystems, thereby decomposing functionality, and simultaneously increasing the connections between subsystems.

There was also a minor increase in data complexity (dependent on number of input/output data and connections between the subsystems). This was caused by the fact that more functionality needed to be accomplished by the system. Moreover, there were more user selectable and fine-tuning related parameters, in order to make the system configurable, more usable and flexible. Additionally, there were many computational requirements due to relatively complex control algorithms.

More sophisticated control algorithm, additional parameters, structuring the system, increase of the system's size (54% in number of blocks compared to the previous version) and other system enhancements reflected in rather minor increase in total complexity of the subsystem. The developers' perception confirms the outcome of complexity analysis and attributes it to the use of contract-based design. In fact, although to some extent more complex, the system is more readable and understandable. It should also be stated that the developers were constantly extending their comprehension of the system. Therefore, the quality of the software should be considered as being influenced by increasing experience of the developers. The system is now more maintainable and reusable, as the lower layers can be modified without the necessity of altering the entire system and higher layers can be reused later on.

The contract-based design approach influences not only the quality of the final product but also the development process and its distribution over time. Figure 4 presents the distribution of development phases over the development cycle. We distinguish five development phases, i.e. specification, refinement(s), programming, unit and system testing. System design (specification and refinement phases) in the test application took majority of the development time (approximately 60% of the total development time of 94 man-hours). Since the focus was shifted to the proper design of the system, coding and testing phases were relatively short (35% in total). For comparison in COCOMO II effort model the average time provided for these phases is 58% [21]. Concentrating on the design enables early defect detection and at the same time prevents defect propagation to the later stages of the development.

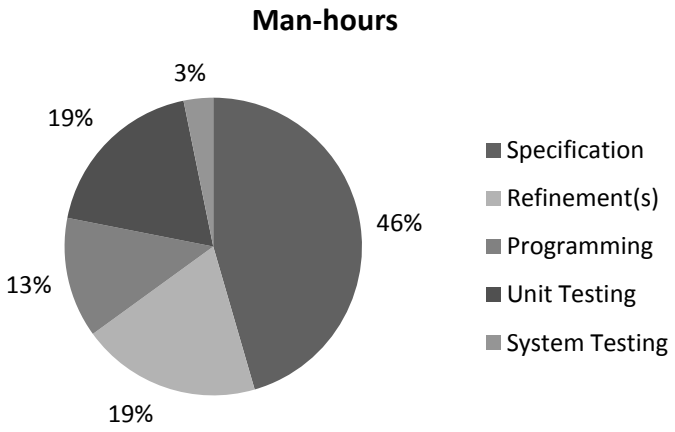


Figure 4. Distribution of time over the development cycle

Design reviews performed by development team were a part of the system design phase (about 10% of the time) and contributed to the high quality of the product. Concentrating on system design and thorough specification of the system benefited in better comprehension of the constructed system. Furthermore, system was well documented, thus facilitating later maintenance.

6. Conclusions and future work directions

Contracts support the decomposition of functionality into subsystems and allow modular analysis of the Simulink models. The analysis can be carried out with different levels of formality. In the case study in this paper the analysis was performed using design reviews and testing. Our experience with using contracts has been encouraging. This positive experience was supported by the evaluation of the impact of used methodology on development process, as well as quality of final product. The contracts facilitate the design process of the system. By documenting the decomposition of the system into subsystems in a legible way, they give explanation for the design decisions. Comprehensible and carefully prepared specification also benefits unit testing and shortens its time. The structure of the system improved when using contracts, as the functionality of the system is well decomposed. Moreover, understandability of the system increased, since the developers focused on the objectives of the created software. This in turn improved its reliability. Given that the subsystems have clear specifications of their functionality, as well as the assumptions they make about the environment where they are used, the system is also more maintainable and reusable. This will benefit in case of the future system adjustments or developments of similar type.

Further research on influence of lightweight formal methods on quality of the created system should be conducted. Measurements can be used for the evaluation and improvement of dependability attributes, as well as give feedback on methods applied in the development process. The complexity of the system is one of the characteristics that have a major impact on its development, usability, maintainability, risk analysis, as well as cost. We have worked on establishing complexity model, which would suit the specific MATLAB/Simulink environment. Our model enables the analysis of the structure and the data flow in the system. We have examined the complexity model for the higher level subsystems. The complexity was computed for each level of subsystems separately. We need to extend this model to cover lower level subsystems. This can be achieved by considering computational layers and ready made Simulink specific subsystems. Afterwards it would be possible

to create the complete complexity metric for the whole system. We would also like to compare our model with other complexity metrics. The aimed relative metric is McCabe complexity [22], adjusted to the Simulink development environment. This would enable the evaluation of the validity of our model.

Correctness of the developed system is important from a dependability point of view. More research on validation of Simulink diagrams with respect to contracts is needed. The contracts can be used as a basis for formal verification of Simulink models. More research is necessary to develop tools and techniques that would enable this to be carried out in an economical way. Validation by testing should also be investigated. When testing, there is a need to determine the appropriateness of test cases. Different types of coverage metrics for both Simulink diagrams and contracts should be investigated.

Bibliography

1. Boström P., Linjama M., Morel L., Siivonen L., Waldén M., *Design and Validation of Digital Controllers for Hydraulics Systems*, The 10th Scandinavian International Conference on Fluid Power. Tampere University of Technology, Tampere, Finland (2007)

2. Boström P., *Formal design and verification of systems using domain-specific languages. Ph D thesis*. Turku Centre for Computer Science, Turku, Finland (2008).

3. Meyer B., *Object-Oriented Software Construction*. 2 ed., Prentice-Hall (1997).

4. Boström Pontus, Waldén Marina, Morel Lionel, *Stepwise development of Simulink models using the refinement calculus framework*, 4th International Colloquium on Theoretical Aspects of Computing (ICTAC2007). Springer, Macao, China (2007)

5. Tripakis S., Sofronis C., Caspi P., Curic A., *Translating discrete-time Simulink to Lustre*, ACM Transactions on Embedded Computing Systems (TECS), 4 (2005), pp.779-818.

6. Cavalcanti A., Clayton P., O'Halloran C., *Control Law Diagrams in Circus*, Proceedings of FM 2005. Springer-Verlag (2005)

7. Pontus Boström, Mikko Huova, Marta Pląska, Matti Linjama, Mikko Heikkilä, Kaisa Sere, Marina Waldén, *development of controllers using Simulink and contract-based design*, International Journal of Critical Computer-Based Systems (IJCCBS), (2009) (Submitted February 2009).

8. Fenton N., Neil M., *Software metrics: roadmap*, Conference on the Future of Software Engineering. , Limerick, Ireland (2000)

9. *Software Technology Roadmap*. Carnegie Mellon Software Engineering Institute, Pittsburgh (2008)

10. Center Software, *Guidelines for Successful Acquisition and Management of Software Intensive Systems: Weapon Systems, Command and Control Systems, Management Information Systems*. (2003)

11. Linjama M., Koskinen K.T., Vilenius M., *Accurate tracking control of water hydraulic cylinder with non-ideal on/off valves*, International Journal of Fluid Power, 4 (2003), pp.7-16.

12. Linjama M., Huova M., Boström P., Laamanen A., Siivonen L., Morel L., Waldén M., Vilenius M., *Design and Implementation of Energy saving Digital Hydraulic Control System*, The 10th Scandinavian International Conference on Fluid Power. Tampere University of Technology, Tampere, Finland (2007)

13. Siivonen L., Linjama M., Vilenius M., *Analysis of Fault Tolerance of Digital Hydraulic Valve System*, Bath Workshop on Power Transimission and Motion Control (PTMC'05). , Bath, UK (2005)

14. Back R.-J., Kurki-Suonio R., *Decentralization of Process Nets with Centralized Control*, Proceedings of the 2nd ACM SIGACT-SIGOPS Symposium of Principles of Distributed Computing. ACM (1983)

15. Back R.-J., Sere K., *Stepwise Refinement of Action Systems*, Structured Programming, 12 (1991), pp.17-30.

16. Back R.-J., von Wright J., *Trace Refinement of Action Systems*, Proc. of the 5th International Conference on Concurrency Theory, CONCUR'94. Springer-Verlag, Uppsala, Sweden (1994)

17. Mathworks Inc., MATLAB/Simulink, http://www.mathworks.com

18. Hatton L., *What is a formal method, (and what is an informal method)*, COMPASS '97 - Are we making progress towards computer assurance?. IEEE, Gaithersburg, Maryland, USA (1997)

19. Cem K., Bach Kaner,, Pettichord B., *Lessons Learned in Software Testing: A Context-Driven Approach*. Wiley (2001).

20. Pląska M., Waldén M., *Quality Comparison and Evaluation of Digital Hydraulic Control Systems*. Turku Center for Computer Science (TUCS), Turku (2007)

21. Yang Y., He M., Li M., Wang Q., Boehm B., *Phase Distribution of Software Development Effort*, ESEM'08. ACM, Kaiserslautern (2008)

22. McCabe T.J., Butler C.W., *Design Complexity Measurement and Testing*, Communications of the ACM, 32 (1989), pp.1415-1425.