Quality Analysis of Simulink Models

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

[1]Department of Information Technology
Åbo Akademi University
Joukahaisenkatu 3-5
FIN-20520 Turku
{marta.plaska, marina.walden, kaisa.sere}@abo.fi

[2]Institute of Hydraulics and Automation
Tampere University of Technology
P.O. Box 589
FIN-33101 Tampere
{mikko.huova, matti.linjama}@tut.fi

Abstract

In this paper we present a measurement framework for complex control systems developed in Simulink environment. Quality analysis of a digital hydraulics control system is done for the purpose of its evaluation and improvement. We concentrate on the assessment of quality of the development process and the created software in perspective of lightweight formal methods.

1 Introduction

Continuous software improvement is observable everywhere nowadays. It is also a major goal for most software organizations. The evaluation of quality of produced software is considered as a main activity towards achieving high quality products. Our goal for performing a measurement program is to monitor and evaluate the quality of control system software and its development process.

Most of the developers have some intuition about the quality of software they produced [Jor99]. However, to genuinely determine the quality and obtain its complete picture, measurement activities and appropriate metrics are necessary. The analysis, which takes place after data collection and gathering measurement results, is the final outcome of the examination. It portrays the quality features of the system, by assigning values to the tested characteristics in question, thus giving them certain meaning. Measuring software quality is very complex, because it consists of evaluating software products, processes and resources, each of which is composite as well. Assembling these aspects of quality allows us to get a comprehensive view on the developed system.

Assessing the quality of control software systems, like the digital hydraulics ones, is challenging, as it is multidimensional and algorithmically complex. Digital hydraulic systems are an alternative in the fluid power technology, replacing high-cost analogue valves with simple and easy to manufacture on/off-type valves. Digital hydraulic valves

consist of digital flow control units (DFCU) composed of parallel connected on/off-valves [LKV03]. With this kind of systems it is possible to achieve more flexible functions than with traditional ones. These functions include e.g. energy saving capabilities due to distributed nature of the valve system [MLi07].

To realize these sophisticated functions with just on/off-valves the control algorithms have become more complex. The difficulty with complex controllers is that if they are not properly designed and produced the risk of malfunction becomes higher. To be able to handle the complexity and produce reliable software Boström et al. [Bos07] proposes the use of contract-based design method in producing control algorithms for modern hydraulic systems. In this paper we focus on the test application development, later called TC II, which is a part of controller of digital hydraulic valve system that controls one cylinder. TC II was developed with use of contract-based design methodology in the MATLAB/Simulink environment [Mat07], a high-level graphical design environment for modelling, simulation, implementation and testing of dynamic and embedded systems [Sim09]. The data for the quality measurements were gathered simultaneously within the development of TC II. Therefore we not only can examine the product quality including its structure, but also we are able to observe and analyse the process of the controller development. We also depict project quality including resources, tools and methodology used.

One should note that the term "measurements" can be used with different meaning both in hydraulics and software (system) field. Software related definition states about assigning a number or symbol to an entity in order to characterise software quality attribute [Fen97]. Since software for digital hydraulics is nowadays crucial for obtaining effective and efficient system, a need for software measurements arose. Numerous books, just to mention [Kan03] [KRK05], and publications, for instance [Fen00], [Gra94], were published on quality of software engineering. They discuss the relevance of quality measurement and its contribution towards improving quality of software, as well as introduce software metrics. Structural measurements have been used for managing quality in terms of complexity in [LMB09]. However we use a broader complexity model for the quality assessment in specific environment (Simulink).

In section 2 we describe the application domain with short history of digital hydraulics. Section 3 specifies the TC II that we later on examine. The contract-based design development methodology, an essential factor influencing the quality of the final system, is depicted in Section 4. The quality metrics and the measurement results of our study are given in Sections 5 and 6, respectively. In Section 7 we conclude and give directions for our future work.

2 Application Domain – Digital Hydraulics

The need for software development methods with digital hydraulics can be understood when considering the advances of the control algorithms. Many benefits can be gained using digital valves instead of analogue servo and proportional valves. If simple On/Off-

valves are manufactured in large series, the cost of the hydraulic valves may be reduced. Because the response time of the DFCU is the same as the response time of the single on/off-valve, the former is not dependent on the amplitude [LiV07]. Fault tolerance of a digital hydraulic system is much better than fault tolerance of a traditional hydraulic system, because fault in a single valve does not paralyze the system [Sii05]. Different ways to improve energy efficiency of the hydraulic system are also possible with digital hydraulics [Lin08].

Figure 1 presents cylinder control with four digital flow control units. A system of four DFCUs controlling cylinder is known as a digital valve system. This kind of distributed valve system is used to independently control flow rate from supply line *P* to cylinder chambers *A* and *B*, and from the cylinder chambers into the tank line *T*. A typical system consists of four DFCUs that have five parallel on/off-valves, resulting in 20 valves in total. Because each valve can be individually controlled and has two possible states, the number of different control solutions is 2^{20}, resulting in over one million possibilities to choose from. To be able to select the optimal valve control solution at each time step, a model based control algorithm is used.

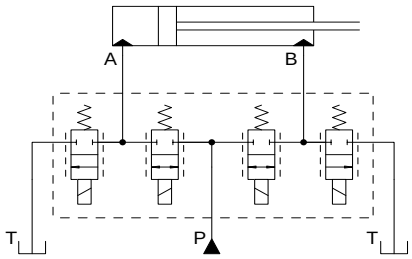

Fig. 1. Hydraulic diagram of digital hydraulic valve system.

The control algorithm consists of four main components. The first component chooses the right control mode according to the load force that is acting on the cylinder, velocity reference and the pressure of the supply line. The control mode describes which of the DFCUs should be used to direct the main part of the flow to achieve optimal efficiency. The second part of the control algorithm is used to select the most promising control solution candidates to more accurate and computationally demanding analysis. Selected candidates are fed to the steady-state calculation component, which calculates the pressures in cylinder chambers and piston velocity that would result from the use of each control candidate. The last component in the control algorithm uses the cost function to select the optimal control signals to the valves using the calculated pressure and velocity information.

Reliability of the control algorithm is very important in heavily loaded hydraulic systems. Incorrect control of the valves may seriously damage the system itself or cause injuries to people who use the system in the worst case. A proper design method has to be used, in order to be able to produce reliable control software. The use of design by contract in development of MATLAB/Simulink based controllers of digital hydraulic

systems has been suggested [Bos07]. In order to study the suitability of contract-based design approach in a development process of such software, TC II was designed.

3 TC II (Test Application)

The cost function of the digital hydraulic valve controller was chosen as TC II (subsystem *Cost Function and Optimal Control*). The task of the cost function is to choose the optimal control signals for the on/off-valves. The cost function is used to simultaneously enable the control of different features of the system. The cost function has to be able to find a compromise between pressure and velocity tracking. Other aspects that the cost function should be able to handle are minimisation of power losses and minimisation of unnecessary valve switching.

At the beginning of the design process the desired properties of TC II were considered at an abstract level. The cost function should be flexible enough to be used with different valve configurations and the system should be as easy to use as possible. The tuning parameters should be intuitive and the user interface well documented. The controller should also give a user an information about the chosen control signals. Other desired properties of the cost function were reliability and expandability to meet future requirements.

The Simulink diagram with the basic structure of the subsystem Cost Function and Optimal Control is shown in Figure 2. The algorithm is divided into five subsystems, all of which have clear area of responsibility. The subsystem *Velocity terms* is used to calculate the velocity error and cost term for each valve control combination in the search space. The subsystem *Pressure terms* calculates the pressure error for both cylinder chambers and the corresponding cost term. The *Switching terms* calculates three different types of cost terms about the switchings of the valves. The cost terms concerning energy consumption and the opening of the secondary DFCUs are calculated in the subsystem *Secondary DFCU terms*. The fifth subsystem uses calculated cost terms to sum up the value of cost function for each control signal candidate in the search space. The selection of the final valve control signal is also done in the subsystem *Finding the optimal u* by selecting a control candidate with the lowest cost function value.

4 Contract-based design

When considering large and complex control systems, formal methods are recognised to be the approach that effectively manages their reliability. Stepwise development based on *superposition refinement* [BKS83] [BaW98] [Kat93] [BaS96] allows the system to be modelled in a layered fashion at different levels of abstraction. The methodology is present from the very beginning of a development process. It not only assists when specifying the system from the requirements, but also facilitates the development in later stages - design and modelling phases in addition to testing.

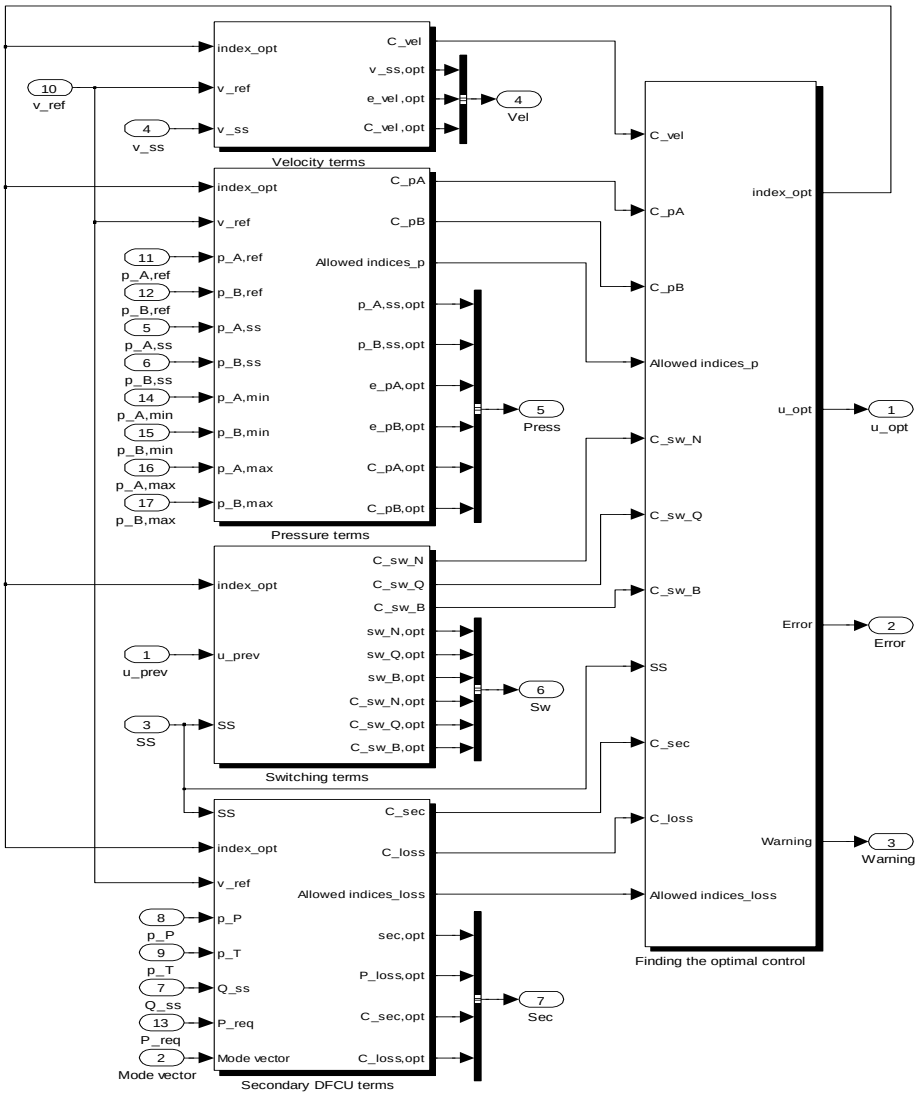

Fig. 2. Structure of the cost function (subsystem Cost Function and Optimal Control).

It is crucial to be able to reason about the elements of the model and their interaction. Therefore we benefit from *Contract-based design* method [Mey92] [Bos08], a systematic approach to specify and implement software. The developed system is seen as a set of communicating components, in our case subsystems. Connections between them are based on accurately defined specifications of mutual require-ensure contracts. Input condition is introduced in the *require clause* (pre-condition) and output condition is introduced by *ensure clause* (post-condition). The specification is associated with every software element. Specifications (or contracts) manage the interaction of a given

element with the real-life elements in terms of obligations and benefits. Every element in the contact is supposed to accept some obligations (pre-conditions), and produce some benefits in return (post-conditions). These characteristics apply to individual routines. For the properties which hold for all instances of classes, class invariants are used. They are assertions of special type and describe the constraints that apply to subsequent changes and cover mutual consistency for the elements. All these properties are examples of assertions or logical conditions related with elements in the contract. Contracts increase the reliability of the developed systems [Bos07].

In the development of TC II we used lightweight formal method, in which proofs were omitted in favour of testing and simulation, as main analysis techniques. The abstract specification was thoroughly prepared with the use of mathematical notation or detailed description. It explained the real world requirements and its dependencies on a high level and was refined into more detailed, lower level specification. Later stages of the development, design and coding, as well as testing, were performed with respect to this specification, ensuring the reliability of the system. This stepwise development guaranteed preserving system's functionality and behaviour given in the initial specification. The whole development process was thus made more manageable and the developers' decisions were more justifiable.

Contracts have been used when modelling TC II in *Simulink*, as they are helpful in reasoning about the development of the model [Bos08]. They ensure that the tasks are performed in a correct manner with respect to the specification and provide robustness, as they assist with handling abnormal situations delivering exception handling mechanisms (errors, warnings as Outport blocks). The notions of inheritance, polymorphism and redefinition are well supported in contract-based design theory. Therefore, it is suitable for stepwise system development, where in each step the system becomes more detailed and clearly defined. In Simulink environment this translates to layering of subsystem blocks and detailing their functionality. The principle of subcontracting ("require no more, provide no less") in the contact methodology is appropriate in this type of systems as well. The assertion-based method facilitates systems analysis due to its precise notation, which prevents risky and premature implementation commitment. In addition, it is mainly used for modelling, as the internal details are out of the scope. It supports software design and project management, as it focuses the developers' and managers' attention on the most significant issues of the development. Furthermore, it is useful in standard documentation of the software elements, as well as communication between developers. This in turn increases the understanding of the system being developed and results in better maintainability of the system later on. The controllers in digital hydraulic systems are becoming more complex and computationally extensive, due to, for instance, many tuning parameters. The similar tendency can also be observed in more traditional machine automation development. Therefore, systematic methods are essential when implementing this kind of complex control systems [MLi07].

5 Software Quality Metrics

The intuition of the researchers is not sufficient to state about the quality of produced software or the efficiency and adequacy of the development approach. *Quality metrics* are necessary in order to demonstrate that lightweight formal methodology is successful in the digital hydraulics area; they are the evidence for scientific progress. Some metrics for control systems exist, but are limited to performance and simulation results. Simulink itself offers simple and direct model measurements.

By *quality* we mean conformance to requirements and fitness for use, both of which are related and consistent [Kan03]. We focus more on the former, as we approach the software creation process from the developers' point of view. Any nonconformance is considered as defect, thereby decreasing the quality of system.

To assess the developed system we are examining three of its aspects: *project* characteristics and execution, the final *product* and *processes* governing its development. Each of these is in fact a composition of many other, more detailed characteristics.

5.1. Project and Process Metrics

When analysing *project* metrics we take under consideration the resources used in the development, such as number of developers in the team and their skill levels, the structure of staff assigned to different tasks, the schedule and division of project lifecycle, as well as the effort. When analysing the resources, one has to also consider tools and methodologies used as they are one of the factors heavily influencing the quality as a whole.

In the examination of the development *process* one has to study the development itself and the impact, which this has on overall product quality improvement. The observations that are made are essential for the enhancement of software development and its later maintenance. Process metrics include the effectiveness of defects removal and time needed to fix discovered problem. By *defect* we mean an anomaly in a product and this term is used interchangeably with *fault* [Kan03].

When the numbers of detected and removed defects are known, we can compute the defect removal effectiveness (DRE), which is a process measurement. DRE is defined as the percentage of defects removed during particular phase per defects latent in the product. The latter is estimated by sum of defects removed during the phase and defects found later. In our work we are calculating the *defect removal phase effectiveness*, which is the DRE for specific phase of the development. The higher the value of this metric's outcome, the more effective is the development process. At the same time the defect propagation to later phases is reduced. Since in our development we are aiming for high reliability, DRE is one of the metrics that well describe the impact of using lightweight formal methods and contract-based design on our system's development. It is inaccurate to assume for all the defects to be injected into the system only at the early stages of the development. Observing the state of defects during development is crucial for portraying the development process itself, as well as the quality of the product at

every development stage. Having defect origin (the phase of defect introduction) as well as defect discovery and removal per phase we are able to gather information about the distribution of defects. Subsequently, from obtained data we produce a matrix of defect origin and discovery per phase. This cross-classification enables us to straightforwardly calculate various defect removal effectiveness measures for the specification, design and code inspection, along with unit and system test DRE. Moreover, the overall defect removal effectiveness for the entire development cycle can be computed, in order to examine defect detection efforts before the product is released to the field.

5.2. Product Metrics

The *product* assessment involves direct measurements about those physical features of the system, like size and structure measurements, that influence the complexity measurements. Product metrics can be used to describe performance and quality level. The size of the developed system will be measured in two ways, with respect to the Simulink environment.

The structures that are examined first, blocks, are gathered directly by the Simulink environmental command *'sldiagnostics'*. Block diagrams are the representations of the system in Simulink. Models created from blocks represent both data and the control flow, and can be simulated using Simulink. Since the implementation is diagrammatic, it gives a graphical view of the system and the development. It might also be used for the documentation, which in turn enhances systems maintenance.

The second ancillary size measurement is the Generated Lines of Code (*GLOC*) metric, understood as the number of physical lines, including executable lines, data definitions and comments, as well as blank lines and program headers. GLOC is not a fully reliable metric, since the automation does not allow measuring the productivity of programmer. Furthermore, it is negatively correlated with design efficiency, in which we are interested in. In our research GLOC serves to determine the relation between the diagrammatic block structures and the executable code itself.

GLOC and block attributes are used for discussion in the context of *defect* rate calculation in particular development phases. Defect density is relative to the software size and directly influences the system quality. We classify the defects by the phase of their origin and the phase that they were found. This enables us to analyse the distribution of defects over entire development process.

The *complexity* metric for the Simulink models is one of the main outcomes of our research. It is based on the structural and relational attributes of the product and influences not only the schedule of the project, but also the productivity of the developers. The higher the complexity, the more effort the developers need to make in order to design a system. In our research we use the *Card and Glass complexity model* [Kan03], which we have adjusted to Simulink environment [PlW07]. This metric evaluates a system by analysing its structure and properties of its model. Our complexity model consists of two components, i.e. *structural* and *data complexity*. For calculating both of the elements the structure called *fan-out* is required. Since the

Simulink system has a graphic representation and its blocks can be interpreted as modules, fan-out stands for a count of subsystems that are called by a given subsystem. In our case it is the count of subsystem blocks that are connected with given subsystem block by input-output parameter. In other words, it is the number of subsystem blocks that can be reached by leaving a given one. Structural complexity is defined as a mean of squared values of fan-out per number of subsystem blocks. Data complexity is specified as a function dependent on the number of inport/outport variables and inversely dependent on the number of fan-out in the subsystem block. After calculating complexity and analysing the results of the computations, we shortly present statistics about *library usage*. This is another measurement related to physical features of the system. It can be obtained with '*sldiagnostics*' command.

6 Quality Measurements in TC II

Both project and process characteristics influence the quality of final product. Resources and methodology used, as well as the development itself affect the software features, which can be assessed with a certain measurement program. Presented framework can be used for determining the quality of Simulink systems also in domains other than the digital hydraulics.

6.1 Project and Process Quality Measurements in TC II

The project in focus was created by three system developers and three indirectly involved project staff members. The development team is well-experienced in the (digital) hydraulics field and averagely experienced in formal methodologies (almost two years of practice). The latter is caused by cooperation with the formal researchers on preceding project. The Simulink is used as the development environment, in which the team members have already been programming for approximately two and a half, five and fifteen years correspondingly.

The iterative development of TC II is combining the superposition refinement with the contract-based design approach in order to obtain reliable and efficient digital hydraulic system. Each of the iterations relies on detailing previous development step, and at the same time it acknowledges the rules of used methodologies. The development is carefully planned; from the comprehensively created system specification, through the deliberately prepared design and controlled coding, to the testing. This lightweight formal approach influences the architecture of the system by structuring it into layers. Each successive layer has more detailed and more extensively specified behaviour than the preceding layer.

The development of the project was divided into five phases: specification, refinement, programming, unit and system testing. The first two can be interpreted as the design or modelling phase. Development with methodologies such as the contract-based design and stepwise refinement devote attention to creation of the system in such a way that it fulfils the requirements and preserves given properties. This in turn directs

the focus to the modelling phase and at the same time reduces the time needed for testing. The unit testing is done in an iterative style, which follows the idea of stepwise development. Therefore, each component is tested after being coded. The system testing phase starts once all components are coded and individually tested.

In our research we analysed the effort made in each development phase in order to be able to examine the development process. The development took 94 man-hours in total. The specification and refinement phase used majority of the development time (65%), whereas the programming phase took 13%. The remaining part was spent on testing. The time needed for testing is relatively small in comparison with projects that are not using formal approaches [JAH90] [LFB96] [JPB06]. The overall system testing phase (3%) is significantly shorter than the unit testing phase (19%), which indicates that final assembly of the system from the already tested components is effective and cost-efficient. The development phases are slightly overlapping with neighbouring ones, as they influence each other.

The documentation of the system and justifications of the design decisions were created simultaneously with the development of the system in its initial two phases. This increased the effort in the first phase by about 25%. However, it eliminated the need of writing the documentation after the deployment of the project, which is proven to be laborious and deficient [PlW07]. The approach to the documenting process we used is beneficial, as the information being recorded is complete, consistent and thoroughly checked. Moreover, we obtain better management over project by relying on good quality documentation. Additionally, the maintenance of the system is facilitated.

6.2 Product Quality Measurements in TC II

The produced system is the main outcome of the project and is analysed as such. First, we focus on the physical measurable feature of the system, which is the system's size. The Simulink environmental command '*sldiagnostics*' provides us with the number of blocks. We have 854 blocks in total. TC II contains 54% more blocks than the previous development, which was less complex and used lightweight formal methodology to a very little extent. The increase in number of blocks is caused by the fact that more functionality needs to be accomplished by the system. The code that was automatically generated from the model consists of 14 428 lines of code (GLOC). From these size measurements we are particularly interested in blocks, as we can examine their relation with GLOCs. The ratio between the number of GLOC and blocks is 17, which is comparable with previously developed systems [PlW07].

In the assessment of software product measurements, gathering information about defects provides a broader view not only on the quality of the system, but also the process of quality assurance. It serves for depicting the process of handling defects – discovering and removing them. In our research we are interested in the defect density and the distribution of defects over the development phases, focusing mostly on classification of defects with respect to their origin and detection phases.

One of our main goals of using contract-based design methodology in combination with stepwise refinement was to obtain reliable and correct software. Therefore, we are concerned with the calculation of defect density. We define defect density as a measure of the total known defects divided by the size of the software entity being measured. There were eight defects found throughout the development cycle of the system until its deployment. Considering relatively small system's size (854 blocks, which corresponds to 14 428 GLOC), the defect density for the final product is very low (0.0095 defects per block or 0.554 defects per kGLOC). For comparison, in previous development there were 0.0221 defects per block or 1.38 defects per kGLOC. It is worth mentioning that in earlier development the team was already using refinement approach and gaining knowledge of how to apply the contract-based design methodology to their development process. Low defect density in TC II was influenced by the skilled development team, well-experienced in digital hydraulics field, as well as generation of code from the model. The equivalent data from projects before the application of lightweight formal methods were not gathered. Therefore we are not able to perform a baseline comparison. Nevertheless, presented numbers give a solid evidence of the accuracy and adequacy of the used approach in a perspective of system's quality.

In Figure 3 we present the classification of defects throughout the system development. The origin of defects is placed in the specification and programming phases, whereas their detection – in programming, unit and system testing. It is worth mentioning that there was very small amount of defects found and the majority of them were discovered by programming and unit testing. Only one defect was found in the last phase of the development, when the system was assembled and tested as a whole. There were no defects found after the deployment of TC II; however, it has not been extensively used yet.

During the system's development we mostly concentrated on the design phase in order to decrease the likelihood of introducing the defects in the early stage of the project, which in consequence would cost more to handle later on. By obtaining the desired behaviour of the system (contract-based design methodology) we prevent system's unplanned adjustments and augmentation. It would require significant amount of resources and might possibly lead to injecting new defects to the system. Since the produced system is correct by construction, the defect density is reduced and the possibility of defect propagation is minor.

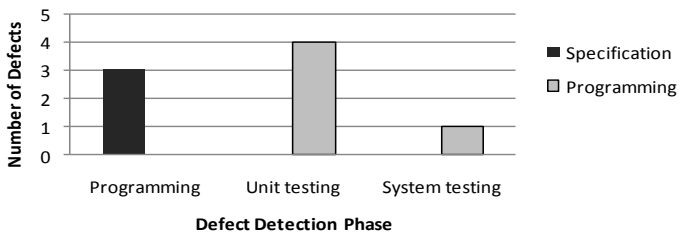

Fig.3. Defect classification

Another defect-related metric is defect removal effectiveness (DRE), which is a process measurement. The higher value of this metric translates to more effective development process. Moreover, this means that fewer defects propagate or emerge in the next phase. Given that the defects were found and fixed in the programming, unit and system testing phases, the effectiveness of defect removal considers exactly these phases. In our calculations we did not take into account estimations concerning the after-deployment phase, as the system was not used extensively enough. The results are presented in Figure 4. In programming phase there were three defects removed, out of eight known, therefore the effectiveness is 37.5%. At the same time five other defects were injected during programming. The defects from the programming phase were removed in sequence: four during unit testing and one – during system testing phase. The defect removal phase effectiveness for those stages of system development is equal to 80% and 100% respectively.

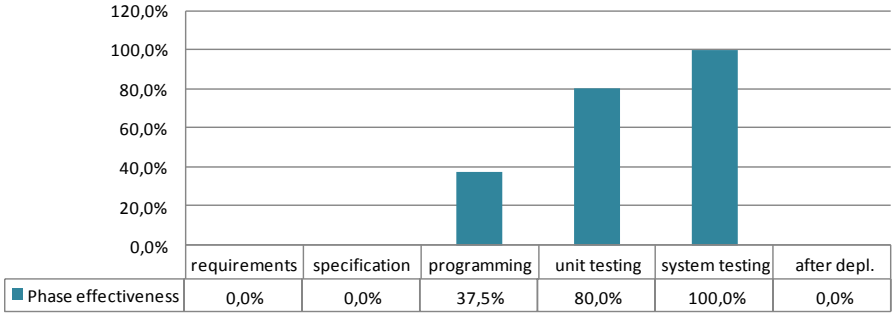

Fig.4. Defect Removal Phase Effectiveness

When considering the physical characteristics of the system we also need to focus on the structural measurements, since the Simulink models are represented in a graphical manner. The model of the system is structured in a *layered* style. This means that when "entering" one subsystem block, more detailed lower layer is exposed. Our system is organised in five layers, where the uppermost one represents the controller and bottom one implements low-level computations. We focus on the structure starting from the second top-most layer, the Cost Function and Optimal Control, and afterwards its subsystems. We are interested in the structure of the system, as it gives information needed to assess the complexity of each layer.

In order to be able to compute the *complexity* based on our complexity model [PlW07], we need to obtain the number of inport and outport variables in the subsystems, as well as compute the fan-out values and count the number of subsystem blocks in examined layer. It should be noted that inport and outport variables used in the complexity calculations are represented in the Simulink diagram by in-going and out-going arrows to and from the subsystems of a considered layer, respectively. By Simulink definition, Inport blocks are the links from outside of a system into it and correspond to inputs, whereas Outport blocks are the links in the opposite direction

representing outputs. Therefore one Inport or Outport block (represented in the diagram as an oval with an arrow) can represent several inport or outport variables of the same value with respect to several subsystems that are using it. In Table 1 we have gathered all the structural measurements for the second and third highest layers.

Tab. 1. *Structural measurements of the system – second and third layer*

Layer level	Subsystem layer	Subsystem blocks	Fan-out	Inport blocks	Outport blocks
2nd	Cost Function and Optimal Control	5	8	17	4
3rd	Finding Optimal *u*	2	1	3	4
3rd	Pressure Terms	4	3	8	8
3rd	Secondary DFCU Terms	5	4	8	7
3rd	Switching Terms	5	7	3	9
3rd	Velocity Terms	3	3	3	4

Cost Function and Optimal Control subsystem is the subsystem in second highest layer and consists of five lower level subsystems: Finding Optimal *u*, Pressure Terms, Secondary DFCU Terms, Switching Terms and Velocity Terms. Because of that structuring we observe the relatively high number of fan-outs, as the functionality is deferred to subsystems at lower layers. Very high value of Inport blocks in the upper layer is caused by the tasks that they are performing and additional configurable user parameters, as well as parameters for tuning. The Outport blocks represent the outcome of the algorithm in addition to the fault tolerance mechanisms.

We follow the model formula for the structural complexity, $S = \frac{\sum_{i=0}^{n} f^2(i)}{n}$, where *f(i)* is fan-out of subsystem block *i* and *n* is a number of subsystem blocks in the system. Furthermore, we compute data complexity according to the formula $D = \frac{V(i)}{n(f(i)+1)}$, where *V(i)* is the number of inport/outport variables in a subsystem block *i*, *f(i)* and *n* are as mentioned before. The structural measurements necessary for computing complexity are gathered in Table 2. It represents the Cost Function and Optimal Control layer and its subsystems with corresponding values of fan-out, inport and outport variables, as well as the sum of inport and outport variables. Values for lower layers are gathered in the same manner.

Tab. 2. Structural measurements of Cost Function and Optimal Control layer

Cost Function and Optimal Control	Fan-out	Inport variables	Outport variables	Overall variables
Velocity terms	1	3	4	7
Pressure Terms	1	10	9	19
Switching Terms	1	3	9	12
Secondary DFCU Terms	1	8	7	15
Finding the Optimal *u*	4	11	4	15

Table 3 presents values of structural and data complexity, along with total complexity for the Cost Function and Optimal Control layer, as well as its subsystems. From these results we deduce that the Cost Function and Optimal Control layer has the highest value of structural complexity. This is caused by the fact that it is the higher level layer, which represents the abstracted view of all the lower layers. Furthermore, there are more connections between the subsystems (high fan-out), which also increases the structural complexity. Moreover, the Cost Function and Optimal Control layer is structured in a recursive manner. This means that the subsystem block where the optimal *u* is being found is called by other subsystem blocks in the layer. This forms a structural loop and in consequence increases the value of fan-out, which in turn increases the value of computed structural complexity.

Tab. 3. Complexity measurements

Subsystem layer	Structural complexity (S)	Data Complexity (D)	Total Complexity (C = S + D)
Cost Function and Optimal Control	4	5.9	9.9
Finding Optimal *u*	0.5	7.5	8
Pressure Terms	1.25	4.96	6.21
Secondary DFCU Terms	2.75	3.5	6.25
Switching Terms	5	2.62	7.62
Velocity Terms	1.67	2.11	3.78

The recursive feature mentioned previously has the influence also on data complexity, but in an inverse manner, since the fan-out value is placed in the equation as denominator. Another factor increasing the data complexity is the number of inport/outport variables. The number of user configurable parameters is higher (comparing to previous developments), as the system can be used for both 4 and 5 Digital Flow Control Units systems. There are also more parameters used for fine-tuning the system's performance. This overall augmentation in inport/outport variables is an evidence of more functionality that needs to be accomplished by the system, and, as a consequence, increase of data complexity. There are many computational requirements, as a result of relatively complex control algorithms.

It is worth noticing that subsystem Finding Optimal *u* has high data complexity, which is caused by extensive algorithmic computations. Conversely, the structural complexity is relatively low, since the computational functionality of this subsystem is not deferred into lower subsystems.

The total complexity is the sum of structural and data ones. It is highest for Cost Function and Optimal Control layer due to high value of structural complexity. The values of total complexities for the lower layers are comparable, which is caused by equal decomposition of the functionalities for particular layers. The total measured complexity increase is relatively small (68%), considering the system's enhancements,

such as user and tuning parameters, more complex control algorithm and broader functionality. The developers' perception is consistent with this result.

Our system uses a set of *library blocks*, information about which can be obtained by triggering the '*sldiagnostics*' command. These are the *digital hydraulics*, *dspstat3* and *simulink* libraries. *Digital hydraulics* library is an in-house library and includes components that are commonly used in controllers of digital hydraulic systems. It includes functions that are used to calculate the force equation of a cylinder. They are very often used in the control algorithms therefore they are considered as quite reliable and complex. The *dspstat3* library is responsible for the display of mathematical and statistical functions such as correlation, standard deviation, minimum or maximum and sorting, among many others. The *simulink* library is built-in, generic type of library that can be extended according users' needs. In our system we are using seven links to the basic functions of the *digital hydraulics* library, one link to the *dspstat3* library to compute minimum and six links of the *simulink* library responsible of logic and bit operations.

An outcome of the development, the documentation, increases the quality level of maintenance process. Every design decision is documented and thereby justified. In addition, structuring the system into layers makes the system more modular and maintainable. Thus, the system is more modifiable, anticipating prospective changes, no matter if they are just adjustments or augmentation.

7 Conclusions

In our research we analysed the final software and its development process in order to be able to state about their quality. To our knowledge there are no metrics that could be used directly for control systems software. The existing ones concern system's physical performance and its simulation. The available in-house Simulink metrics concern mostly the straightforward statistics about the model, as well as design effort and time related metrics. Since software is becoming more composite nowadays, there was a need of more elaborate metrics, such as complexity or defect related metrics. We presented these metrics in a perspective of lightweight formal method development in the Simulink environment. We illustrated how the metrics can be applied to a digital hydraulics domain project, TC II. Our framework, however, is more flexible and we believe it can be used in other domains as well.

In our future work we would like to extend the proposed complexity metric to those low-level block structures that are not subsystems. A normalisation of the planned complexity metric with other model factors would give a comprehensive view on this characteristic. Additionally, we aim at continuing the research on the impact of lightweight formal methods on the quality of the development process and product.

References

[BaS96] Back R.J.R., Sere K., *From modular systems to action systems*, Software - Concepts and Tools 17. (1996)

[BaW98] Back R.J.R., von Wright J., *Refinement Calculus: A Systematic Introduction. Graduate Texts in Computer Science*. Springer, Heidelberg (1998).

[BKS83] Back R.J.R., Kurki-Suonio R., *Decentralization of process nets with centralized control*, 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing. (1983)

[Bos07] Boström P., Linjama M., Morel L., Siivonen L., Waldén M., *Design and Validation of Digital Controllers for Hydraulics Systems.*, 10th Scandinavian International Conference on Fluid Power (SICFP'07). (2007)

[Bos08] Boström P., *Formal Design and Verification of Systems Using Domain-Specific Languages, PhD Thesis.*. TUCS, Turku (2008).

[Fen00] Fenton N., Neil M., *Software metrics: roadmap.*, Conference on the Future of Software Engineering. , Limerick, Ireland (2000)

[Fen97] Fenton N., Pflegger S., *Software Metrics. A Rigorous and Practical Approach.*. PWS Publishing Company (1997).

[Gra94] Grady R., *Successfully applying software metrics*, Computer, 27 (1994), pp.18-25.

[JAH90] Hall J.A., *Seven Myths of Formal Methods*, IEEE Software, (1990), pp.11-19.

[Jor99] Jorgensen M., *Software Quality Measurement*, Advances in Engineering Software, 30 (1999), pp.907-912.

[JPB06] Bowen P., Hinchey G., *Ten Commandments of Formal Methods Ten Years Later*, IEEE Software, (2006).

[Kan03] Kan S., *Metrics and Models in Software Quality Engineering*. Addison-Wesley (2003).

[Kat93] Katz S.M., *A superimposition control construct for distributed systems*, ACM Transactions on Programming Languages and Systems, (1993), pp.337-356.

[KRK05] Kandt R., *Software Engineering Quality Practices*. Auerbach Publications (2005).

[LFB96] Larsen P., Fitzgerald J., Brookes T., *Lessons Learned from Applying Formal Specifications in Industry*, IEEE Software, (1996).

[Lin05] Linjama M., Vilenius M., *Digital Hydraulic Tracking Control of Mobile Machine Joint Actuator Mockup*, SICFP'05. , Linköping, Sweden (June 2005)

[Lin08] Linjama M., Huova M., Vilenius M., *Online Minimisation of Power Losses in Distributed Digital Hydraulic Valve System.*, The 6th International Fluid Power Conference. , Dresden (2008)

[LiV07] Linjama M., Vilenius M., *Digital hydraulics - towards perfect valve technology.*, The 10th Scandinavian International Conference on Fluid Power. (2007)

[LKV03] 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.

[LMB09] Lindemann U,, Maurer M., Braun T., *Structural Complexity Management. An approach for the Field of Product Design*. Spinger (2009).

[Mat07] (MAAB) MathWorks, *Control Algorithm Modeling Guidelines Using Matlab®, Simulink®, and Stateflow®, Version 2.0*. (2007).

[Mey92] Meyer B., *Applying Design by Contract*, In Computer (IEEE), vol 25, no. 10, (1992), pp.40-51.

[MLi07] 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*, SICFP'07. , Tampere (2007)

[PlW07] Pląska M., Waldén M., *Quality Comparison and Evaluation of Digital Hydraulic Control Systems*. Turku Center for Computer Science (TUCS), Turku (2007)

[Sii05] Siivonen L., Linjama M., Vilenius M., *Analysis of Fault Tolerance of Digital Hydraulic Valve System*, Power Transmission and Motion Control. (2005)

[Sim09] Simulink - Simulation and Model-Based Design, http://www.mathworks.com/products/simulink/