OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

EIT

FAKULTÄT FÜR ELEKTROTECHNIK UND INFORMATIONSTECHNIK

INSTITUT FÜR AUTOMATISIERUNGSTECHNIK

Complexity metrics for Engineering Models

Thomas Hadlich

Technical Report

IFAT-LIA 2/2012

13. September 2012

Universität Magdeburg
Fakultät für Elektrotechnik und Informationstechnik
Institut für Automatisierungstechnik
Postfach 4120, D-39016 Magdeburg
Germany

1 Why do we need complexity metrics for engineering models?

During engineering process different models for the engineered system are developed, starting from coarse granular models, stepwise providing more details, increasing the granularity of the models until reaching the point that the system may be implemented. At different steps of the development process the same type of model or different types of models may be used. It is common knowledge, that mistakes introduced in the early development stages will require higher effort the later they are fixed in the development process.

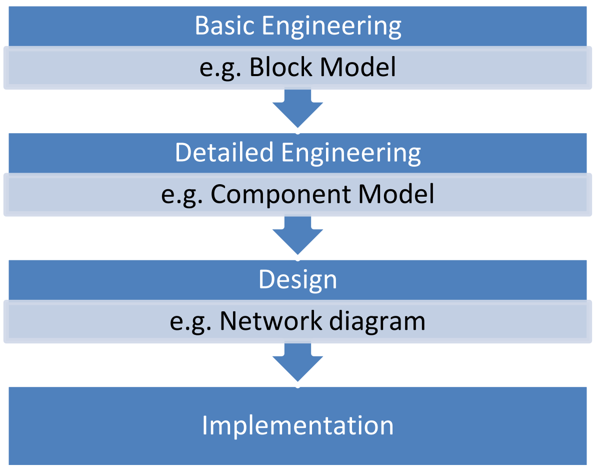


Figure 1 - Example for engineering process with models

Between different steps of the development, the models in each step inherit from earlier models. This inheritance often includes the general structure of the model, but also other characteristics of the designed system and its elements (and of course the mistakes).

Comparing models of different steps is difficult, because it is difficult to capture 'structure' as a characteristic of a model. That is why it is interesting to find characteristics which may be used to describe the structure.

Complexity metrics are well accepted within the area of software engineering for characterizing the structure of software. Such complexity metrics describe the structural complexity of software models. They are used to estimate how easy it is to create, to test and to maintain the modeled software.

This paper proposes a general approach for describing the complexity of engineering models, so that models on different stages of the development process may be evaluated and compared in regard to their structural complexity. Complexity within the early phases of design will result in increased complexity of the implemented system. That is why complexity should already be observed during the early phases of the development process.

Using a general approach will allow to collect empirical knowledge on these models, which then can be used to develop a methodology for handling complexity of engineering models within the development process.

2 Related Work

2.1 McCabe

The McCabe cyclomatic complexity is a generally accepted complexity metric. In [McC76] the cyclomatic number of a graph G with n nodes, e edges and p connected components was introduced as:

$$v(G) = e - n + 2p \tag{1}$$

The cyclomatic number measures the number of independent paths through the graph G. [FP97] shows that the number indicates the number of predicates (branches) in the flowgraph:

$$v(G) = 1 + d \tag{2}$$

(d is the number of predicates).

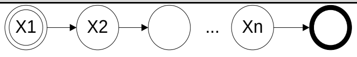
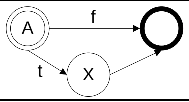
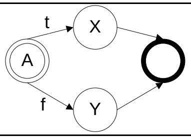
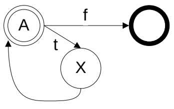
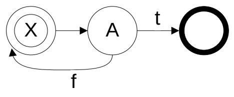
According to [FP97] „… the cyclomatic number is a useful indicator of how difficult a program ore module will be to test and maintain. In this context, *v* could be used for quality assurance. In particular, McCabe has suggested that, on the basis of empirical evidence, when *v* exceeds 10 in any one module, the module may be problematic“.

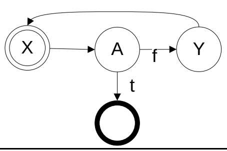
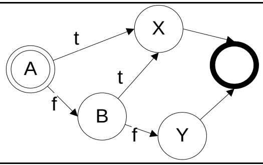
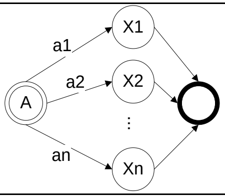
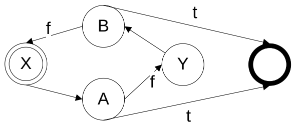
The McCabe complexity metric does not consider the complexity of the nodes (statements) in the flow model. Each node is considered equally complex, whether it is a simple assignment, a call to a complex function or an invocation of a function that may affect the execution of the control flow (e.g. access to unique resources, which might cause deadlocks). A sequence of statements is considered as simple, no matter how complicated the statements within that sequence are.

2.2 Fenton

In order to analyze complexity of a control flowgraph [FP97] introduces a set of common flowgraphs (‚primes‘) that can be used to create any structured flowgraph by composition. These primes should be considered as atomic components of flowgraphs which cannot be decomposed any further. The primes defined by Fenton are shown in Table 1.

Table 1 - Prime flowgraphs according to ([FP97]

Name	Term	Graph	McCabe cyclomatic complexity
Sequence	P_n $P_n(X1, X2, \ldots, Xn)$		$v = n - (n+1) + 2$ $v = 1$
Condition (selection)	D_0 $D_0(A, X)$		$v = 3 - 3 + 2$ $v = 2$
Branch	D_1 $D_1(A, X, Y)$		$v = 4 - 4 + 2$ $v = 2$
While-loop (iteration)	D_2 $D_2(A, X)$		$v = 3 - 3 + 2$ $v = 2$
Do-loop	D_3 $D_3(A, X)$		$v = 3 - 3 + 2$ $v = 2$

Name	Term	Graph	McCabe cyclomatic complexity
middle-exit loop	D_4 $D_4(A,X,Y)$		$v = 4 - 4 + 2$ $v = 2$
lazy Boolean evaluation OR	D_5 $D_5(A,B,X,Y)$		$v = 6 - 5 + 2$ $v = 3$
Multi-Branch	C_n $C_n(A, X_1, \dots, X_n)$		$v = (2 * n) - (n + 2) + 2$ $v = n$
two-exit loop	L_2 $L_2(A,B,X,Y)$		$v = 6 - 5 + 2$ $v = 3$

Fenton uses these primes as base for further analysis and evaluation of other existing metrics (for evaluation on McCabe see above). Since any structured flowgraph ('structured' in the sense of structured programming) can be composed by a subset of these primes, it is possible to decompose all such flowgraphs into a decomposition tree. The decomposition tree shows how the respective flowgraph is composed by primes.

The decomposition tree may serve as base for additional measures like

- Number of nodes
- Number of edges
- 'Largest prime'
- Cyclomatic number of largest prime
- Number of occurrences of specific primes
- Fulfillment of structuredness principle

Fenton points out, that a single metric is not sufficient to measure complexity, but that several measures should be used in conjunction in order to evaluate the complexity of a program.

2.3 NetComplex

In [CRK08] a complexity metric for networked systems has been introduced. This metric is unique in a sense, that it not only considers the influence of the node connections on complexity, but also regards the influence of the node behavior. The metric differentiates between 'value dependency' and 'transport dependency'. Value dependency derives from the fact, that a network node may use additional input or state information for influencing the transmitted information while transport dependency shows that information needs to be transmitted between nodes in a network and that the used transport mechanisms may have different complexities.

The calculation for the complexity of a state is defined as:

$$c_s = \sum_{x \in D_s} c_{s \leftarrow x} \qquad (3)$$

c_s – complexity of state s. The complexity is calculated from the sum of influences on state s. If a state does not depend on another state, then complexity of that state is considered to be zero ($c_s = 0$). The dependency on other states is defined as:

$$c_{s\leftarrow x} = \begin{cases} u_{s\leftarrow x} + \sum_{y \in T_{s\leftarrow x}} \max(c_y, \varepsilon) + c_x & \text{if } x \text{ is linked to } s \\ \varepsilon & \text{if } x \text{ is unlinked to } s. \end{cases} \tag{4}$$

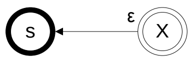
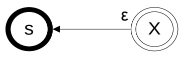
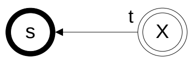
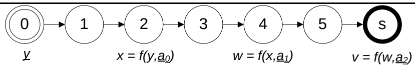
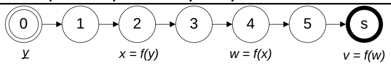
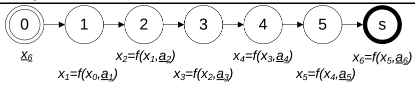
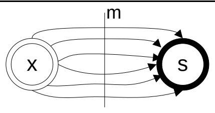
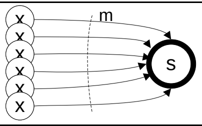
'linked' and 'unlinked' describe whether state *x* has continuous impact on *s* or whether state *x* has only once impact on *s* (e.g. at T_0). ε describes a very small, but not negligible value with $0 < \varepsilon \ll 1$.

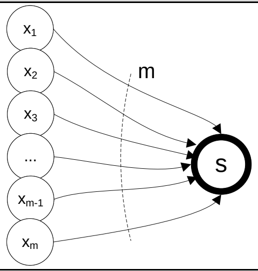
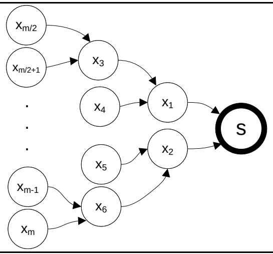
$u_{s\leftarrow x}$ describes the submetric 'direct value dependency impact' – basically the number of remote states from which state s is directly derived.

$$u_{s\leftarrow x} = \begin{cases} u_x & \text{if } x \text{ is linked to } s \text{ and } x \text{ is not dependent on local state} \\ u_x + 1 & \text{if } x \text{ is linked to } s \text{ and } x \text{ is dependent on local state} \\ \varepsilon & \text{if } x \text{ is unlinked to } s. \end{cases} \tag{5}$$

Table 2 shows the canonical scenarios provided by [CRK08] for comparison reasons provided together with McCabe cyclomatic complexity. It is interesting to observe how the transport mechanism for the information influences the complexity calculation (e.g. for examples 1 (broadcast) and 3 (unicast)). [CRK08] uses t as a factor > ε.

Table 2 - Canonical scenarios according to [CRK08]

#	Name	Graph / Description	NetComplex	McMcabe cyclomatic complexity
1	Single input (linked), 1-hop broadcast	 $s = f(x)$	$c_s = 1 + \varepsilon$	$v = 1 - 2 + 2$ $v = 1$
2	Single unlinked input, 1-hop broadcast	 $s = f(x)$ at T_0	$c_s = \varepsilon$	$v = 1 - 2 + 2$ $v = 1$
3	Single input (linked), 1-hop unicast	 $s = f(x)$	$c_s = 1 + t$ (t is a factor > ε)	$v = 1 - 2 + 2$ $v = 1$
4	m value dependencies, 1 direct, m-1 indirect	 0 1 2 3 4 5 s y x = f(y,a0) w = f(x,a1) v = f(w,a2) Nodes 4; 2; s have value dependency on local state and remote state; Nodes 5; 3; 1 have transport dependency only.	$c_v = u_v + 2t + c_w$ $c_w = (u_x + 1) + 2t + c_x$ $c_x = (u_y + 1) + 2t + c_y$ $c_y = 0$ $c_v = 6 + 6t$	$v = 6 - 7 + 2$ $v = 1$
5	m value dependencies, 1 direct, m-1 indirect	 0 1 2 3 4 5 s y x = f(y) w = f(x) v = f(w) Nodes 4,2,s have value dependency on remote state; Nodes 5; 3; 1 have transport dependency only.	$c_v = u_v + 2t + c_w$ $c_w = u_x + 2t + c_x$ $c_x = (u_y + 1) + 2t + c_y$ $c_v = 1 + 6t$	$v = 6 - 7 + 2$ $v = 1$
6	m value dependencies, 1 direct, m-1 indirect	 0 1 2 3 4 5 s x6 x2=f(x1,a2) x4=f(x3,a4) x6=f(x5,a6) x1=f(x0,a1) x3=f(x2,a3) x5=f(x4,a5) All nodes have value dependency on local state and remote state;	$c_s = \frac{(n)(n+1)}{2} + n * t$ $c_s = 21 + 6t$	$v = 6 - 7 + 2$ $v = 1$
7	1 input, k of m paths		$c_s = 1 + k * t$ k=1: $c_s = 1 + t$	For m = 6: $v = 6 - 2 + 2$ $v = 6$
8	k of m inputs, separate paths		$c_s = k(1 + t)$ *k=1:* $c_s = 1 + t$	For m = 6: $v = 6 - 7 + 2$ $v = 1$

#	Name	Graph / Description	NetComplex	McMcabe cyclomatic complexity
9	m direct value dependencies		$c_s = m(1+\ t)$	For m = 6: $v = 6 - 7 + 2$ $v = 1$
10	Tree	 xm/2 xm/2+1 x3 x4 x1 x2 x5 x6 xm-1 xm s	$c_s = O(m \log_2 m + mt)$	For m = 6: $v = 6 - 7 + 2$ $v = 1$

[CRK08] demonstrates that the calculation may also show that complexity of a state can be derived from value dependency and transport dependency:

$$c_s = c_s^V + c_s^T \tag{6}$$

c_s – complexity of state s,
c_s^V – „complexity contributed by value dependencies"
c_s^T – „complexity contributed by transport dependencies"

$$c_s = \sum_{x \in D_s} c_{s \leftarrow x} \tag{7}$$

$c_{s \leftarrow x}$ – influences of other states x on state s

$$c_{s \leftarrow x}^V = u_{s \leftarrow x} + c_x^V \tag{8}$$

$u_{s \leftarrow x}$ – direct value dependency impact of state *x* on state *s*.

$$c_{s \leftarrow x}^T = \sum_{y \in T_{s \leftarrow x}} \max(c_y, \varepsilon) + c_x^T \tag{9}$$

Comparing McCabe cyclomatic complexity and NetComplex does not seem to make sense, because McCabe describes complexity of execution structures and NetComplex describes complexity of network structures. But if both metrics are considered in terms of describing complexity of graphs, a comparison should be possible. While McCabe does not consider differences within the nodes or within the edges of the graph, with NetComplex it is possible to differentiate scenarios although the graph has the same structure (e.g. scenarios 1-3 and 4-6).

It is possible to provide even more information when the dependencies on specific states and on specific transports are marked specifically. If d_x marks the value dependency on state of node x and l_x marks the transport dependency from node x, equation (4) would be modified to:

$$c_{s \leftarrow x} = \begin{cases} u_{s \leftarrow x} + \sum_{y \in T_{s \leftarrow x}} l_x + c_x & \text{if } x \text{ is linked to } s \\ \varepsilon & \text{if } x \text{ is unlinked to } s. \end{cases} \tag{10}$$

and equation (5) would be modified to:

$$u_{s\leftarrow x} = \begin{cases} u_x & \text{if } x \text{ is linked to } s \text{ and } x \text{ is not dependent on local state} \\ u_x + d_x & \text{if } x \text{ is linked to } s \text{ and } x \text{ is dependent on local state} \\ \varepsilon & \text{if } x \text{ is unlinked to } s. \end{cases} \tag{11}$$

For instance the complexity for example 6 in Table 2 would be calculated as:

$$c_s = d_5 + 2d_4 + 3d_3 + 4d_2 + 5d_1 + 6d_0 + l_5 + l_4 + l_3 + l_2 + l_1 + l_0 \tag{12}$$

While the transport dependencies all have the same weight, the value dependency for a node which is connected by many intermediate nodes (e.g. d_0) has much more impact (by factor 6) than the dependency of a neighboring node (e.g. d_5).

3 New proposal: ModelComplex

Almost all engineering models are based on graphs. Examples are: process graphs, flowgraphs for software, network diagrams, block diagrams, function block diagrams, Petri nets, state machines, P&I-diagrams.

The proposal is to use one general complexity metric for all types of graphs, but to allow adaption for the different engineering models. The adaptation may occur by setting the dependency factor values differently for different models or within one model to use different factor values for different model elements.

For instance if all models are evaluated in regard to complexity with the same metric, it would be possible to use this metric for estimation of the complexity of a system element (e.g. based on its state machine) and then to use this complexity information for evaluation of the complete system, which integrates the system element.

The calculation of complexity shall follow the definitions as provided by NetComplex, but with adaptations for calculating the dependency impact and the complexity impact. For dependency impact the calculation shall be defined as:

$$u_{s\leftarrow x} = \begin{cases} u_x & \text{if } x \text{ is linked to } s \text{ and } x \text{ is not dependent on local state} \\ u_x + d_x & \text{if } x \text{ is linked to } s \text{ and } x \text{ is dependent on local state} \\ \varepsilon & \text{if } x \text{ is unlinked to } s. \end{cases} \tag{13}$$

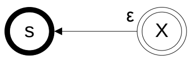
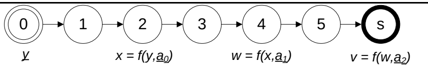
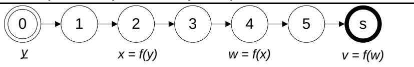
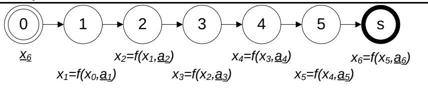
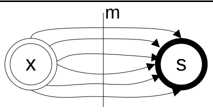
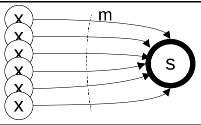
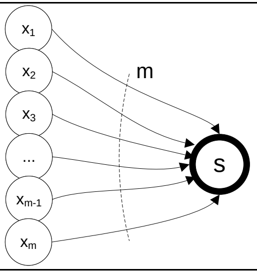
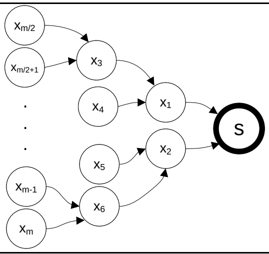
d_x - describes the impact of a local information of x on the dependent information of s (value dependency). This impact may be adapted depending on the engineering model.

For dependency impact the calculation shall be defined as:

$$c_{s\leftarrow x} = \begin{cases} u_{s\leftarrow x} + \sum\limits_{y\in T_{s\leftarrow x}} l_x + c_x & \text{if } x \text{ is linked to } s \\ \varepsilon & \text{if } x \text{ is unlinked to } s. \end{cases} \tag{14}$$

Table 3 demonstrates the resulting calculations for the canonical scenarios of [CRK08].

Table 3 - Canonical scenarios with ModelComplex

#	Name	Graph / Description	ModelComplex
1 2 3	Single input	 $s = f(x)$	$c_s = d_x + l_x$
4	m value dependencies, 1 direct, m-1 indirect	 0 1 2 3 4 5 s y x = f(y,a0) w = f(x,a1) v = f(w,a2) Nodes 4; 2; s have value dependency on local state and remote state; Nodes 5; 3; 1 have transport dependency only.	$c_s = d_4 + 2d_2 + 3d_0 + l_5 + l_4 + l_3 + l_2 + l_1 + l_0$
5	m value dependencies, 1 direct, m-1 indirect	 0 1 2 3 4 5 s y x = f(y) w = f(x) v = f(w) Nodes 4,2,s have value dependency on remote state; Nodes 5; 3; 1 have transport dependency only.	$c_s = d_0 + l_5 + l_4 + l_3 + l_2 + l_1 + l_0$
6	m value dependencies, 1 direct, m-1 indirect	 0 1 2 3 4 5 s x6 x2=f(x1,a2) x4=f(x3,a4) x6=f(x5,a6) x1=f(x0,a1) x3=f(x2,a3) x5=f(x4,a5) All nodes have value dependency on local state and remote state.	$c_s = d_5 + 2d_4 + 3d_3 + 4d_2 + 5d_1 + 6d_0 + l_5 + l_4 + l_3 + l_2 + l_1 + l_0$
7	1 input, k of m paths		$c_s = d_x + k * l_x$
8	k of m inputs, separate paths		$c_s = k(d_x + l_x)$
9	m direct value dependencies		$c_s = \sum_{i=1}^{m}(d_i + l_i)$
10	Tree	 xm/2 xm/2+1 x3 x4 x1 x5 x2 xm-1 x6 xm s	With the assumption that nodes on each level have the same impact d_x and all edges have the same transport dependency t: $c_s = \sum_{i*0}^{depth} ((2^{i+1} - 2)d_{x-i}) + (2^{depth+1} - 2) * t$

For each engineering model the decision must be made specifically on how to resolve the dependency on specific nodes. Typically within one engineering model all nodes of the same type would have the same dependency factor *d* and all edges of the same type would have the same transport factor *l*. If the dependency is not relevant within an Engineering Model, it will be possible to set the factor to *0*.

Systems in automation often provide more than one function (e.g. manufacturing systems are able to produce different products; control systems control several values). In such case complexity needs to be considered separately for each function. This means, that for each function the path in the system graph needs to be identified and the complexity of a system function will be evaluated by following this path.

4 Application Examples

The application examples discussed in this document are not intended to define how the ModelComplex metric will be used with specific models, but are intended to provide starting points for discussion.

4.1 Process diagram

If we consider the development of a networked system as shown in Figure 1 for an example for a process model, the variable d_k would describe the complexity of the model *k*. If we consider the development of a networked system from Block Diagram to Component Diagram and from Component Diagram to Network Diagram, it is accepted that the complexity of the models increase, since the models will contain increasingly more detailed information.

For a linear sequence as shown in Figure 1, the complexity impact can be calculated as defined in line 6 of For dependency impact the calculation shall be defined as:

$$c_{s \leftarrow x} = \begin{cases} u_{s \leftarrow x} + \sum_{y \in T_{s \leftarrow x}} l_x + c_x & \text{if } x \text{ is linked to } s \\ \varepsilon & \text{if } x \text{ is unlinked to } s. \end{cases} \qquad (14)$$

Table 3 demonstrates the resulting calculations for the canonical scenarios of [CRK08].

Table 3.

$$\mathrm{c_s = d_3 + 2d_2 + 3d_1 + l_3 + l_2 + l_1} \qquad (15)$$

d_3 – describes the complexity of the network diagram (which is identical to the implementation), d_2 – describes the complexity of the component diagram and d_1 – describes the complexity of the block diagram. The values of l_3, l_2, l_1 describe the complexity of the transformation from one model to the other (e.g. for data model transformation).

Increase in complexity at the block diagram will have by factor 3 more impact on the complexity of the implementation – as already mentioned in paragraph 1 of this document: "… mistakes introduced in the early development stages will require higher effort the later they are fixed in the development process.".

4.3 Network Diagram

For analyses of communication within network diagrams according to [CRK08], the dependency factors d_x all are set to value '1' and the transport factors l_x are set to values that reflect the respective transport (e.g. ε, t).

4.4 Function Block Diagram

A Function Block Diagram according to IEC 61499 is a directed graph, where the nodes represent Function Blocks and the edges represent either control flow or data flow (see Figure 2).

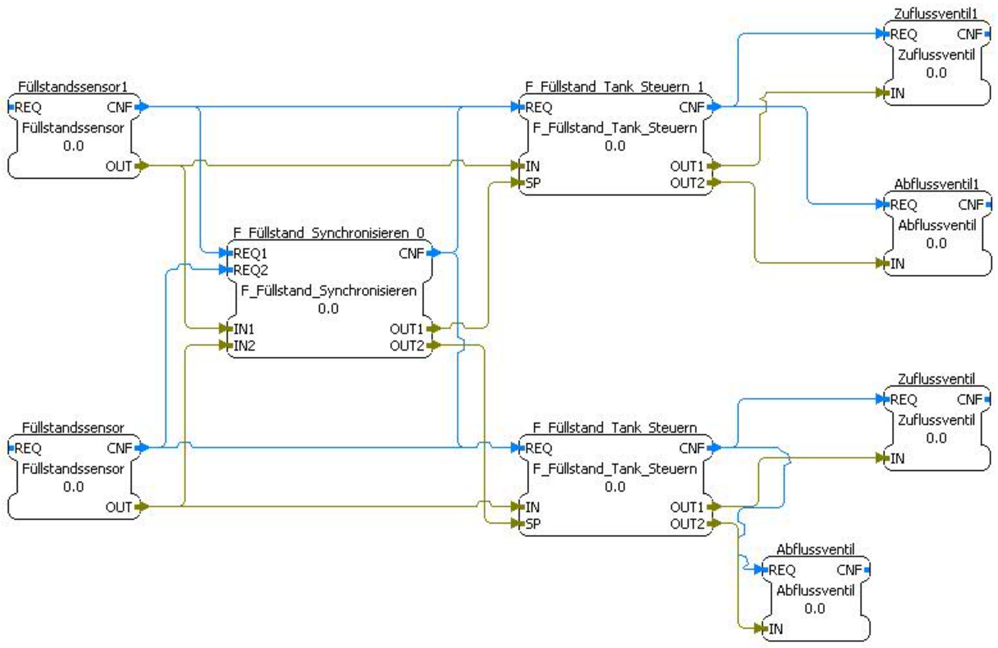


Figure 2 – Example function block diagram

Function blocks (FBs) may represent either stateless functions (e.g. Boolean logic) or functions with internal state. FBs with state can be modeled either by composition from other FBs or by definition of the internal state machine. – In both cases it will be possible to assign a complexity value to a specific type of Function Block. FBs for Boolean functions (e.g. AND, OR) should be considered to have no value dependency. They would be treated similar to nodes 1, 3, and 5 in line 4 of For dependency impact the calculation shall be defined as:

$$c_{s\leftarrow x} = \begin{cases} u_{s\leftarrow x} + \sum\limits_{y\in T_{s\leftarrow x}} l_x + c_x & \text{if } x \text{ is linked to } s \\ \varepsilon & \text{if } x \text{ is unlinked to } s. \end{cases} \quad (14)$$

Table 3 demonstrates the resulting calculations for the canonical scenarios of [CRK08].

Table 3. FBs with states (e.g. FlipFlop, Delay-FB, Counter) will have a complexity value reflecting their impact on the design.

Since FB-diagrams are created from libraries of FBs, it would be possible to define the value dependency factor for each FB-type, when it is integrated into the library. If the FB-type is used (i.e. when it is instantiated into a FB-network) it would be possible to use the metric for evaluation of the modeled FB-network.

It needs to be researched how the different types of connections (data flow / control flow) influence complexity. For discussion within this document it is assumed, that both types of connections provide the same transport dependency. If an FB-network is deployed within a distributed system (consisting of several controllers), some of the FB-connections will exist within the same controller and some of the connections will be based on communication between the controllers (see dashed connections in Figure 3).

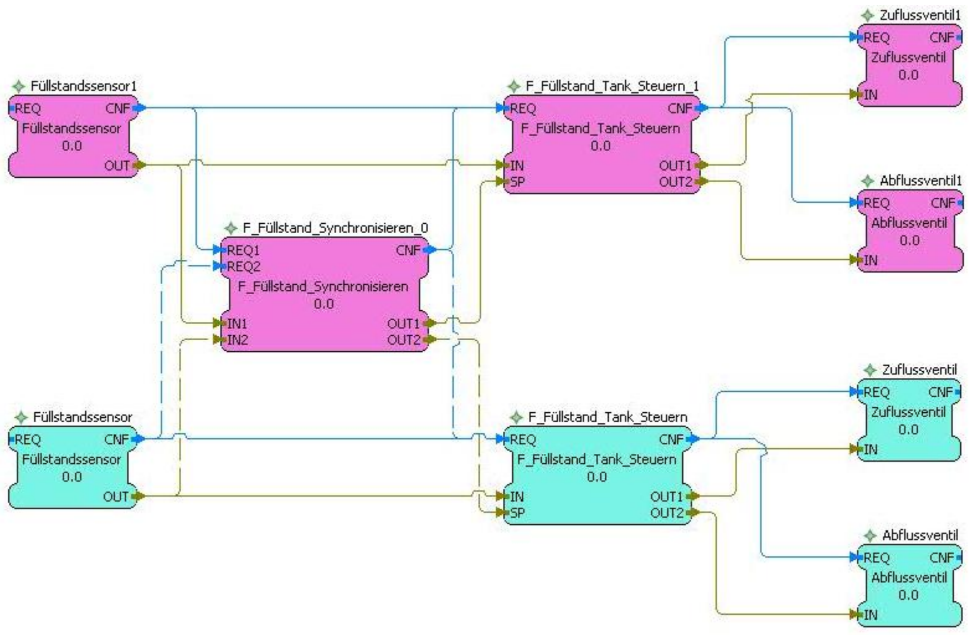


Figure 3 – Example function block diagram deployed on two controllers

While the transport dependencies for connections within one controller should have a value that is very low (e.g. 0 or ε), the transport dependency for a connection based on inter-controller communication should be assigned an appropriate high value.

Since control flow and data flow are parallel in the above FB-network and since the control flow basically controls the data flow, both are considered as integrated connections.

Figure 4 shows the FB diagram transcribed as graph.

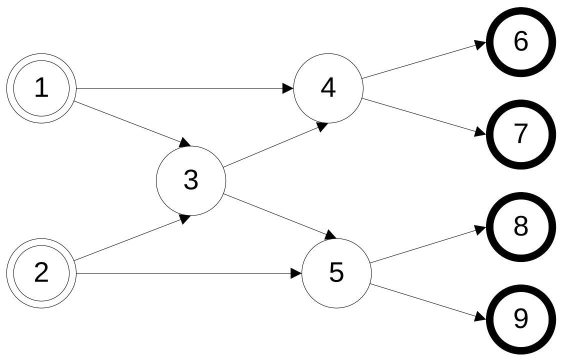


Figure 4 – Example function block diagram as graph

With this graph it becomes visible, that the system provides 4 outputs based on 2 input values. The generation of each output should be considered as separate function, each with a specific complexity.

The complexity of the first two function blocks is considered to be 0 ($c_1 = c_2 = 0$). The resulting complexities for all 4 output values of the function block network are:

$$
\begin{aligned}
c_6 &= 5d_1 + 3d_2 + 2d_3 + d_4 + 2l_{3\leftarrow 1} + l_{3\leftarrow 2} + l_{4\leftarrow 3} + l_{6\leftarrow 4} \\
c_7 &= 5d_1 + 3d_2 + 2d_3 + d_4 + 2l_{3\leftarrow 1} + l_{3\leftarrow 2} + l_{4\leftarrow 3} + l_{7\leftarrow 4} \\
c_7 &= d_5 + 3d_1 + 5d_2 + 2d_3 + l_{7\leftarrow 5} + l_{3\leftarrow 1} + l_{3\leftarrow 2} + l_{5\leftarrow 3} + l_{5\leftarrow 2} \\
c_8 &= d_5 + 3d_1 + 5d_2 + 2d_3 + l_{8\leftarrow 5} + l_{3\leftarrow 1} + l_{3\leftarrow 2} + l_{5\leftarrow 3} + l_{5\leftarrow 2}
\end{aligned} \tag{16}
$$

In a deployment scenario where all function blocks are executed on the same controller, we could assume that all $d_x = d$ (all FBs have similar complexity) and for all $l_x = \varepsilon$ (the signal exchange between the FBs is very simple). In such case the complexity for all functions would be:

$$c_s = 11d + 5\varepsilon \tag{17}$$

The complexity of the functions would be mainly defined by the complexity of the function blocks.

If we consider the deployment of the function block network onto 2 controllers (as shown in Figure 3), the transport dependencies to and from node 3 (which represents the FB “Füllstand synchronisieren” (“Synchronize level)) need to be treated differently:

$$\begin{aligned} c_6 &= 11d + 4\varepsilon + \ l_{3\leftarrow 2} \\ c_7 &= 11d + 4\varepsilon + \ l_{3\leftarrow 2} \\ c_7 &= 11d + 3\varepsilon + l_{3\leftarrow 2} + \ l_{5\leftarrow 3} \\ c_8 &= 11d + 3\varepsilon + l_{3\leftarrow 2} + \ l_{5\leftarrow 3} \end{aligned} \tag{18}$$

It becomes visible that the designed deployment has impact on all functions, but the output of nodes 7 and 8 are more affected by this distribution. The size of the impact will depend on the type of communication which will be chosen for the implementation.

Since there are no primes defined for function block networks, it is not possible to provide a hierarchy tree according to [FP97]. Still it is possible to provide a tree diagram describing the structure of the system:

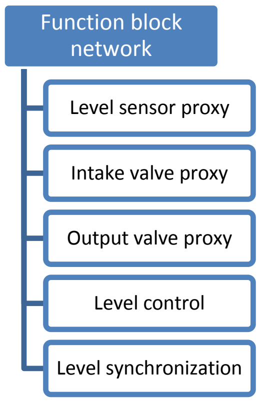


Figure 5 – Hierarchie tree of function block network

Similar to [FP97] several other characteristics of the network can be noted:

- Number of nodes: 8 function blocks
- Number of edges: 10 (each representing data flow and control flow)
- Largest function block: Level synchronization block (this is an preliminary estimate)
- Complexity of largest prime: <needs to be analyzed>
- Number of occurrences of specific primes
  - Level sensor proxy: 2
  - Intake valve proxy: 2
  - Output valve proxy: 2
  - Level control FB: 2
  - Level synchronization FB: 1

5 Summary

This paper proposes an approach to characterizing the structure of engineering models. It is not possible to characterize the structure just by one metric, but a set of metrics should be used.

Within this set of metrics, a metric for measuring complexity of engineering models is proposed. Using the same metric for different engineering models would allow to compare different models. Such comparison for instance would help to understand the development process for a system or to make decisions within the development process.

An application example is used to demonstrate how this new approach may be used to characterize a Function Block model. Also other models are discussed shortly.

The paper is intended as a starting point for further discussion on the topic.

Literaturverzeichnis

[CRK08] Chun, B.-G.; Ratnasamy, S.; Kohler, E.: NetComplex: A Complexity Metric for Networked System Designs: NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation. USENIX Assoc., Berkeley, CA, USA, 2008; S. 393–406.

[FP97] Fenton, N. E.; Pfleeger, S. L.: Software metrics. A rigorous and practical approach. PWS Pub., London ;, Boston, 1997.

[McC76] McCabe, T. J.: A Complexity Measure. In IEEE Trans. on Software Engineering, 1976, SE-2; S. 308–320.