

## MASTER

### Evaluation of supervisory control theory based on requirement evolution of LOPW

van der Schriek, Yorick I.C.

*Award date:*  
2018

[Link to publication](#)

#### **Disclaimer**

This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

#### **General rights**

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

# Evaluation of supervisory control theory based on requirement evolution of LOPW

*Master Thesis*

Student: Y.I.C. van der Schriek

Student Number: 0737693

Master: Manufacturing Systems Engineering  
Department: Mechanical Engineering  
Research Group: Control Systems Technology  
In collaboration with: ASML Netherlands B.V.

TU/e Supervisors: dr.ir. M.A. Reniers  
ir. L.J. van der Sanden  
External Supervisor: dr.ir. R.R.H. Schiffelers

CST 2018.032

Eindhoven, 2 July 2018



---

## Abstract

The demand for computer chips is continuously rising. ASML is the market leader of lithography systems used in the semi-conductor industry to produce chips. To ensure that the requests of the manufacturers are met with regard to the increasing demand, the lithography systems are continuously improved. Not only is this done by developing new systems, but also improving existing ones with regard to the control software is essential. Lot Operations, Process Wafer (LOPW) is the high level controller of the wafer logistics within the wafer scanners of ASML. LOPW makes sure the wafer logistics is executed correctly, telling components, such as the wafer handler or chuck what to do, but not how they should perform that operation. This supervising of LOPW ensures that wafers are put in, processed and removed from the machine. In addition to this, LOPW ensures maintenance, conditioning and scheduling requirements are met.

The logistical process within a wafer scanner is complex, which results in a big controller. To resolve this problem the controller is split into many smaller controllers, which are connected in a hierarchical way. Currently this is done using Analytical Systems Design (ASD), where each subcomponent of LOPW is modelled manually according to requirements. Earlier research has shown that Supervisory Control Theory is a valid method to replace the manual modelling of the components. This method uses formal requirements in combination with the uncontrolled behaviour of the system to synthesize correct supervisor controllers for the components. The elimination of manual design of the controllers should result in fewer mistakes with respect to the requirements and an improvement in development time. These benefits of SCT with regards to ASD are hypothetical and the goal of this research is to test this hypothesis.

To test the hypothesis several subcomponents of LOPW are selected and modelled in Compositional Interchange Format (CIF). CIF is a modelling language which allows supervisor synthesis. These models are then evolved according to the requirement evolution during the years. The requirements are extracted from the ASD models. Unfortunately the documentation is written for general requirements of LOPW and written after changes are made to the models. The result of this is that the requirements used for each version are not always traceable in the documentation. These models are then compared to their ASD equivalents, including the amount of changes according to metrics.

The results show that the SCT methodology might be beneficial for ASML, since the amount of changes which have to be made to CIF models are less than the amount of changes required in ASD models. Due to some limitations within CIF with regards to data transference there should be follow up research to determine certain aspects of modelling as they do not fully match the functionality as provided by ASD. The follow up research can then determine whether these functions are required and how they should be implemented. For large components the SCT methodology is not capable of synthesizing a controller. Therefore more research has to be done to conclude a method for large components to allow the SCT methodology to be applied in all components of LOPW.



---

## Preface

This is my thesis "Evaluation of supervisory control theory based on requirement evolution of LOPW". This thesis describes the research that has been performed on historical data provided by ASML. This research was conducted from November 2017 to June 2018 at ASML as part of my endeavour to meet the graduation requirements of the Manufacturing Systems Engineering master within the Mechanical Engineering Department of the Eindhoven University of Technology.

At the start of my graduation internship at ASML there was a lot unclear about the outcome of this research, but during this internship it was difficult to remain fully focussed on the main goal as there were many interesting branches to look at. Ramon Schiffelers as supervisor at ASML was a great help determining the right track together with my TU/e supervisors Michel Reniers and Bram van der Sanden. Our regular meetings gave me new insights and motivation to keep going, for which I am grateful.

I want to thank ASML for the opportunity to do an internship there and my supervisors for their guidance and insight. Thanks Josh Mengerink for help with the EMMA tool and Yuri Blankenstein and Rolf Theunissen for their help with the implementation of the ASD technology bridge.

I want to thank my parents for supporting me during my study, my friends for the joyfull distractions and Fraukje for being the drive behind my work ethic.

Thank you reader for taking time to read this thesis and I hope you enjoy the pages to come.

Yorick Isedorus Cornelis van der Schriek

# Contents

<b>Abstract</b>	<b>I</b>
<b>Preface</b>	<b>III</b>
<b>List of Abbreviations</b>	<b>VI</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Research Setting . . . . .	1
1.2 Research objective . . . . .	2
1.3 Research approach . . . . .	3
1.4 Structure of this report . . . . .	4
<b>2 Analytical System Design</b>	<b>5</b>
2.1 ASD General . . . . .	5
2.2 ASD Models . . . . .	6
2.3 ASD Actions and Semantics . . . . .	8
<b>3 Lot Operations, Process Wafer</b>	<b>10</b>
3.1 LOPW scope . . . . .	10
3.2 LOPW construction . . . . .	11
3.3 Analysis . . . . .	11
3.4 Requirements . . . . .	16
<b>4 Supervisory Control Theory and Application</b>	<b>18</b>
4.1 General . . . . .	18
4.2 CIF . . . . .	18
4.3 SBS to CIF transformation . . . . .	20
4.4 Application to LOPW . . . . .	23
<b>5 Metrics</b>	<b>27</b>
5.1 Metric Selection . . . . .	27
5.2 Counting Metrics . . . . .	28
5.3 General Model Metrics . . . . .	28
5.4 Halstead Metrics . . . . .	30
5.5 Change Metrics . . . . .	31
5.6 Metrics application to CIF . . . . .	32
<b>6 Methodologies Comparison</b>	<b>34</b>
6.1 Modelling Metric Analysis . . . . .	34
6.2 Modelling Analysis . . . . .	40
6.3 Modelling Restrictions . . . . .	41
6.4 ASD Pollution . . . . .	45

<b>7 Conclusion and Recommendations</b>	<b>46</b>
7.1 Conclusion . . . . .	46
7.2 Recommendations . . . . .	46
<b>References</b>	<b>48</b>
<b>Appendices</b>	
<b>A ASD Metric Extraction Java Scripts</b>	<b>50</b>
A.1 GitMiner to obtain all model revisions . . . . .	51
A.2 Size Measurement . . . . .	52
A.3 Metrics Extraction . . . . .	54
A.4 Metrics Definition . . . . .	57
<b>B CIF Metric Extraction Python Scripts</b>	<b>61</b>
B.1 Alphabet Generating Script . . . . .	61
B.2 Automata Counter per CIF specification . . . . .	63
B.3 Automata Name Collector . . . . .	64
B.4 Changes in CIF in automata . . . . .	66
B.5 CIF Metric Collector . . . . .	69
B.6 Runtime Script . . . . .	76
<b>C Changelogs</b>	<b>79</b>
C.1 Component A . . . . .	79
C.2 Component B . . . . .	79
C.3 Component C . . . . .	80
C.4 Component D . . . . .	80
C.5 Component E . . . . .	81
C.6 Component F . . . . .	82
C.7 Component G . . . . .	82
C.8 Component H . . . . .	83
<b>D Metric Data Per Revision</b>	<b>84</b>
D.1 Component A . . . . .	84
D.2 Component B . . . . .	85
D.3 Component C . . . . .	86
D.4 Component D . . . . .	87
D.5 Component E . . . . .	87
D.6 Component F . . . . .	88
D.7 Component G . . . . .	90
D.8 Component H . . . . .	91



---

## List of Abbreviations

ACC	Actual Cyclomatic Complexity
ASD	Analytical System Design
CC	Cyclomatic Complexity
CE	Chuck at Exposure side
CIF	Compositional Interchange Format
CM	Chuck at Measure side
CSI	Changed Source Instructions
CSV	Comma Separated Value
DM	Design Model
DU	Discharge Unit
EMMA	EMF (Meta) Model Analysis tool
EPDS	Element Performance and Design Specification
EUV	Extreme Ultraviolet
FIFO	First In First Out
FP	Function Points
IM	Interface Model
LOC	Lines of Code
LOPW	Lot Operations, Process Wafer
LTS	Labelled Transition System
MI	Maintainability Index
PU	Prealignment Unit
SCT	Supervisory Control Theory
SSI	Shipped Source Instructions
TR	Track
TU/e	Eindhoven University of Technology
VAF	Value Adjustment Factor
WH	Wafer Handler
WS	Wafer Stage

# 1. Introduction

This first chapter will introduce ASML and the research setting. The goal of my graduation project is stated after these subjects and the approach of reaching this goal is discussed. The chapter will conclude with a brief explanation on how this report is structured.

## 1.1 Research Setting

Since ASML was formed in 1984 [1] it has become worldleader in developing machines which allow the fabrication of chips. These so called wafer scanners are used by all major chip manufacturers of the world. These manufacturers require ASML to keep improving their machines to allow increase of transistor amounts per chip. This is done by using new techniques as Extreme Ultraviolet(EUV) lithography to decrease the smallest feature size, which is currently down to 10nm [26]. However not only the accuracy of the imaging had to be improved, but throughput of the ASML machines also had to be improved, leading to the Twinscan machines of ASML. These Twinscan machines were not only able to expose a wafer, but could also prepare the next wafer for this process simultaneously.

The wafers on which the chips are printed are moved through these machines. The Lot Operations, Process Wafer (LOPW) is the controller of this logistical process. However as machines are improved the movement requirements of these logistics change. If all these requirements would be incorporated in one big controller, it would become very prone to errors, as the controller would be just too complex. Therefore the LOPW is constructed as an hierarchical controller. This controller consists of many sub-controllers in an hierarchical structure. Each sub-controller is controlling a certain piece of the machine, but is itself supervised by a controller on a higher level in the hierarchy.

ASML is using Analytical Software Design (ASD) [4] to manually design these sub-controllers to conform with the requirements. These requirements are partially written down in documentation, but not all requirements are fully documented [13]. Because the hierarchical approach has been used for some time now and sub-controllers are adjusted frequently, it becomes harder to determine where certain requirements are incorporated in the controllers. This combined with the manual modelling required for the adjustment of a controller to meet new requirements has led to the research on using supervisory control synthesis [12], [22]. Supervisory control synthesis derives a controller from a collection of plants and a set of requirements. The collection of plants represent the machinery by modelling the uncontrolled behaviour which should be controlled. The requirement set specifies the behaviour which is allowed on the machinery. Both should be written down in a formal way for the synthesis algorithm to use them.

The research of Robin Loose [14], [13] has shown that supervisory control synthesis can generate controllers with the same behaviour as the manually constructed controllers. This means that both approaches could be used to ensure correct movement of the wafers within the machine. As stated before the currently used approach of manually constructing controllers has its drawbacks, however that does not directly mean that supervisor synthesis is better. For this reason this research will compare the two methodologies.

## 1.2 Research objective

Both the ASD and supervisory control synthesis approach have the same controller validity, which means both methods result in usable controllers which conform to the requirements. However, ASD requirement confirmation has to be done manually, where SCT ensures this confirmation directly. These methods also differ in methodology on how to achieve the controller. These methodologies can be evaluated according to metrics which quantify software and methodology characteristics. Even though these metrics can be used for this, the base of both methods are the requirements. Requirements have changed over time and are still continuing to change. However the rate and severity of the changes has not been mapped and is therefore unknown. Without knowledge of this the impact of another method is not realistically measurable. The main problem definition for this master's project is focusing on the differences between ASD and SCT, combined with requirement changes history and can be defined as:

**Evaluation of supervisory control theory based on requirement evolution of LOPW.**

### 1.2.1 Subproblems

To achieve the main goal, multiple sub-problems should be solved.

1. **Data analysis**

Analysis of the available (historical) data. This data is raw and should be processed to be useful. Identify the requirements per component, log the changes per requirement and for the complete component. A creation of a historical overview of requirements should be the result of this subproblem.

2. **Metrics selection**

To be able to determine differences and quantify these. There are many metrics available and the right metrics should be selected to evaluate the models. A result of this is a standard quality quantification method for both CIF and ASD. In addition to this changes between versions can be quantified for both modelling languages.

3. **Component selection**

As LOPW has hundreds of individual components, it would be too much work to evaluate all components with respect to historical developments in requirements, therefore a selection of models should be made. These models will be used as base for this research.

4. **Requirement categorization**

LOPW is built according to a set of requirements. These requirements can be gathered from different sources and can be categorized based on their source [13], functionality [21] or construction [15]. This categorization can be used to research the relative influence of a requirement of a specific category on the ASD and CIF models with respect to the selected metrics.

5. **Model generation**

There are old LOPW models available in ASD format. Without a valid CIF model on the same requirements, it is impossible to make a comparison between both modelling

languages. CIF models should be created for the selected components, which satisfy the same requirements as the ASD models.

#### 6. Model development

The selected components should be adjusted in CIF to meet newer requirements to simulate the development history of LOPW. During this process ASD models are evaluated with regards to their functionality, to make sure all non-documented requirements are still taken into account. This ensures a level of equivalence and would allow evolution analysis.

#### 7. Metrics quantification

The selected metrics have to be used to evaluate the models and the changes made after requirement changes. These metrics can eventually be used to compare the methodologies and model qualities.

#### 8. Methodology comparison

Using the quantified metrics the methodologies can be compared. Combined with the data analysis a conclusion can be drawn on the methodologies.

### 1.2.2 Scope

As LOPW is too big to analyze completely and has many aspects which can be evaluated it is important to define a clear scope. The ASD and CIF models can be converted in other languages such as C for implementation and Labelled Transition System (LTS) for model evaluation purposes. The implementation of the controller has been left out of scope in this research. Earlier research used the transformation to LTS [13] for equivalence checking between CIF and ASD. This method is not used during this research, as the goal is to create controllers based on the requirements and not to prove equivalence between two controller design methods.

An important metric when looking at software development methodologies is time duration. More specifically the time it takes to implement certain requirements. Even if it is possible to obtain the implementation time while using ASD it is impossible to compare this data with equivalent data using CIF. This is due to the lack of exact programming time logs, personal differences and environmental influences. Therefore this aspect of methodology comparison is not used and instead the Halstead Time definition is used [9], where time is defined by the complexity of the model.

### 1.3 Research approach

The sub-problems should be tackled in a rather sequential order. First all versions of each component should be obtained. This set of models should then be cleared of broken models before extracting metrics from them. These models are either corrupted models or unfinished models with errors in them. The values for these metrics can be used to select several interesting models which the research will focus on.

For each of the models a set of versions is available. Each of these models has a set of other models which are used by the main model. To be able to use these models the corresponding

model versions of used components should be added. Then the requirements of this version can be written down. When the requirements are known, the component is modelled in CIF with the formalized requirements. Finally a supervisor is synthesized to obtain the final result for this version. These steps are repeated until the entire version history of the component is covered. When all versions are modelled, both ASD and CIF models are evaluated with regards to metrics. These metrics can be compared to establish a conclusion on which method is preferable for this component. In addition to the metrics, findings during the modelling of components in CIF are used to form this conclusion. After all selected components are modelled and evaluated, the results are combined to form a grounded conclusion.

This report is a guide through this process, which is further detailed in the next section.

## 1.4 Structure of this report

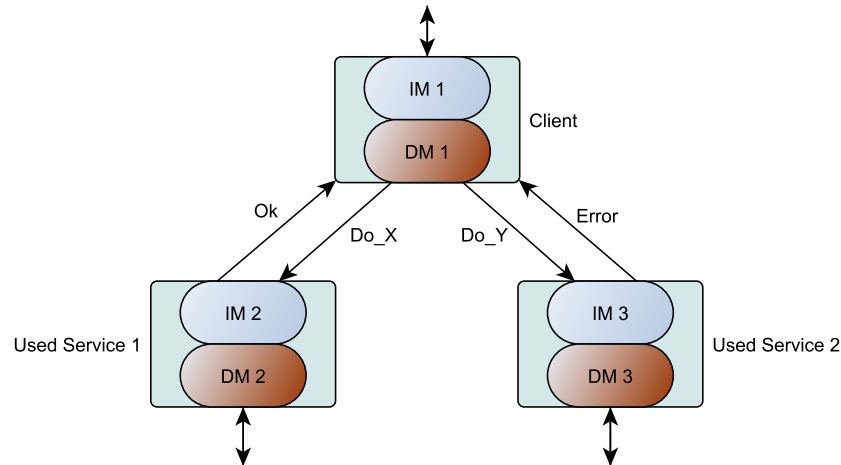
In the next chapter the working of ASD is globally described. Note that only the most relevant functionalities and semantics of ASD are described. Chapter 3 will explain LOPW in more detail and an analysis of LOPW in general is conducted. The chapter will conclude with a classification of the requirements of LOPW. The next chapter is Chapter 4. This chapter globally depicts the supervisory control theory and in more detail how to use it to convert LOPW into CIF. Chapter 5 will elaborate on the metrics used and how they will contribute to the evaluation. Chapter 6 will use the metrics of both the ASD models and CIF models to compare both methodologies. In addition to this, other findings during this research related to the difference between the methodologies are elaborated on. The report will be rounded off with a conclusion and recommendations with regard to this research.

## 2. Analytical System Design

What LOPW is and what its function is, is briefly described in the previous chapter and will be elaborated in Chapter 3. This chapter will elaborate on the building blocks of which LOPW is currently built, which are modelled with ASD. After the introduction of ASD several key aspects of ASD are described in more detail. The chapter concludes with the introduction of metrics to analyse ASD models.

### 2.1 ASD General

The building blocks of LOPW are Analytical System Design (ASD) [4] components. ASD uses sequence based specifications (SBS) [18] to model components and their controlled behaviour. The modelling language is developed by Verum Software Tools B.V. [25] as a control specification language and describes finite state automata. These automata are abstract machines with multiple states and transitions between them, but the automaton can be in only one state at a time. Regardless of the external components, ASD models can be used to define a controller. This controller can tell these external elements what to do in what order, but not how it should be done. The how is determined by the elements themselves. ASD:Suite, which is the developing arena for ASD, allows for conversion of the ASD models into executable code for implementation purposes. During this conversion the semantics and correctness is maintained to ensure right behaviour of the system.



**Figure 2.1:** Schematic illustration of a hierarchical structure of three ASD components.

ASD models can be combined in an hierarchical structure, where decisions can be made based on actions of other components. In Figure 2.1 a simple example of this is depicted where the client uses two other models as a service. The client can command a service to do action X, where the service will send a reply back to notify the client the action is complete. In return this service component can use other services in the same way as the current client uses this service component. In addition to this a client can use multiple services, but can also be a service to multiple other clients. This can be expanded to a multi level hierarchical system,

which allows for partitioning of complex systems into many smaller components to reduce the complexity. All model files are within the same folder and refer to each other within that folder.

## 2.2 ASD Models

The client, as described in the previous section, is built out of two blocks: an Interface Model (IM) and a Design Model (DM). These models are both defined by rulecases, which are the core of the Sequence Based Specification. Each rulecase exists of several declarations:

- Interface: From which interface the trigger comes.
- Trigger/Event: An incoming event which leads to the activation of the rulecase. This can either be a call event from a component higher in the hierarchy, a notification event from an used interface or an internal trigger.
- Guard: A boolean check to evaluate if this rulecase is allowed to be executed. If the guard is evaluated true when the trigger happens, the actions specified in this rulecase are executed. This is an optional attribute for a rulecase.
- Action: If the trigger happens and the guard is evaluated true, the actions listed here will be executed.
- State Variable Update: If there exists a state variable within the model, this can be updated to another value. This is optional for each rulecase.
- Target State: Defines the transition of state after this rulecase.

	Interface	Event	Guard	Actions	State Variable Updates	Target State
1	<b>NotActivated (initial state)</b>					
4	IAAlarmSystem	SwitchOn+	x<=2	WindowSensor:ISensor.Activate	x=2	Activated_Idle

**Figure 2.2:** Example of a rulecase.

Figure 2.2 shows an example of a rulecase as used in ASD. This particular rulecase defines when an alarm should be activated: when a switch is flipped and the variable 'x' is smaller or equal than two. When this is accepted the window sensor is activated and 'x' is set to two after which the model transitions to the target state. In addition to the rulecases SBS allows for specification of multiple states. In each state it is possible to define different behaviour when a trigger comes in. In the first rule of Figure 2.2 it is indicated that this state is initial with the name: NotActivated.

### 2.2.1 Interface Model

The service interface model is the top of the client, which is the interface model belonging to the DM of the component. It shows the external behaviour of the client to components higher in the hierarchy and can be used to communicate with them as can be seen in Figure 2.1. All possible actions which can be returned to components higher in the hierarchy following a

request from above are declared within the interface model. This allows for non-determinism, where a request has multiple possible replies. In Figure 2.3 an example of this is depicted, where a person triggers 'Call', this trigger event is depicted in the figure with a dotted line. Either the person he calls can 'Answer' or 'Denied', which are both action events and are depicted with a solid line.

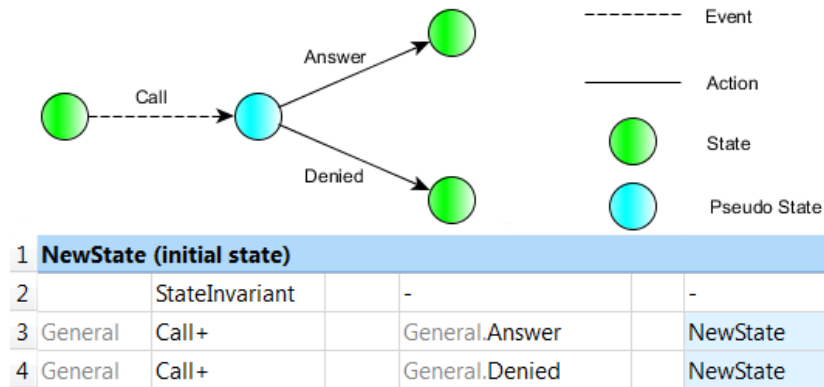


Figure 2.3: Transitions within an interface model.

In addition to the requests from above, actions that result from notification triggers from a used service are declared. These are called modelling events and it is declared that they can be triggered, but not by what, this is done in the design model. They are defined either *optional* or *inevitable*, which implies that they will occur maybe or definitely. However this is mainly a distinction in the verification process in ASD and has no definite meaning within the model.

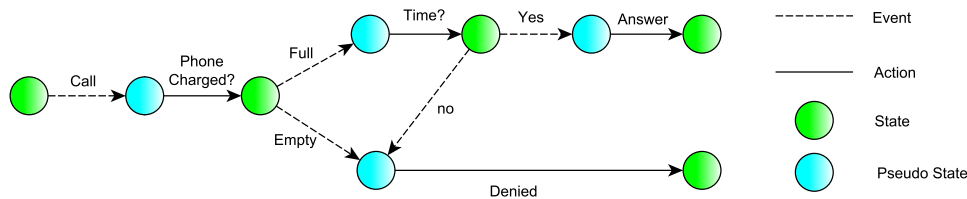
## 2.2.2 Design Model

The design model describes the internal functioning of the component. Instead of only showing what is visible to components higher in the hierarchy, it declares what should be shown. Based on a request from the top, the design model will determine what should be the reply of the interface model. This is done based on the used services. The design model can communicate with the used services by sending a request. The interface model of the used service will receive this request and will respond with an action, which in turn can be determined by the design model of the used service.

Figure 2.4 is a design model adhering to the interface model given in Figure 2.3. The interface model did not have a valid base for a choice, where the design model specifies when the call is answered: when the phone is charged and the person has time to answer.

In addition to the normal sequence of trigger and reply, there can also occur a notification event in the used service which invokes actions in the component. Notification events are events which are not initiated by an event from within the component. Notification events can trigger modelling events in the interface model. These notification events, unlike the regular events, happen asynchronously. This means they are stored in a queue when such a notification event is triggered. As soon as the client is able to process the trigger it will process the notification event. No other triggers will be processed as long as the notification event is in this queue. Multiple notifications can occur and are stored to a maximum of 7





**Figure 2.4:** *Transitions within a design model.*

per client and are then processed in a first in first out order (FIFO). This number can be adjusted, but the encountered models used the default value of 7.

## 2.3 ASD Actions and Semantics

So far all actions used in examples are named actions, which define what should happen. There are several specific action declarations within ASD which deserve some attention.

- **VoidReply:** an action which is equal to those who are named, but is defined in every ASD model. It simply replies with an action called ‘VoidReply’ to notify other components that a trigger has happened on the component.
- **Illegal:** defines that the trigger is not allowed to happen here. Only occurs in models with multiple states, where a certain trigger is allowed to happen in one state, but not in the other.
- **NoOp:** abbreviation for No Operation. This implies that a trigger is allowed, but has no corresponding action.
- **Blocked:** this design model specific declaration indicates a certain call event will not happen because it can’t. An example of this can be seen in Figure 2.5, where a call from the component (Phone Charged) can only be answered with ‘Full’ or ‘Empty’ by the used component (Phone). The call ‘Time?’ is not sent to the phone yet and therefore the answers to this call will not and cannot occur. These answers are therefore blocked.
- **Disabled:** this interface model specific action type defines the modelling events to be unable to occur in certain states.

Named actions and VoidReplies can be combined into a list on a single rulecase meaning they are executed in order of specification. The specific action declarations are mutually exclusive: a trigger can’t have NoOp and Illegal declarations as they contradict each other.

The reason for action declarations Illegal, Blocked and Disabled is the completeness requirement of SBS, which specifies that each possible trigger should be mentioned in every state, even if they are not allowed. Using these declarations this requirement can be met. In addition to this, if a guard is specified, actions should be specified for both the true evaluation of the guard and the false evaluation. An example can be seen in Figure 2.5. In addition to the IM of Figure 2.3, there is a possibility of a conference call within the interface model. However this is blocked by the design model (Illegal). Due to this requirement ASD components can

have hundreds of blocked and/or illegal rulecases. An advantage of this is that every possible action is declared, which makes it easy to see what will occur and what will not.

	Interface	Event	Guard	Actions	riable l	Target State
<b>1</b>	<b>NewState (initial state)</b>					
4	General	Call+		Battery:Battery.PhoneCharged+		CheckBattery
5	General	ConferenceC...		Illegal		-
6	Battery:Battery	Full		Blocked		-
7	Battery:Battery	Empty		Blocked		-
8	Time:OwnerTime	yes		Blocked		-
9	Time:OwnerTime	no		Blocked		-
<b>10</b>	<b>CheckBattery (synchronous return state)</b>					
13	General	Call+		Blocked		-
14	General	ConferenceC...		Blocked		-
15	Battery:Battery	Full		Time:OwnerTime.Time+		TimeCheck
16	Battery:Battery	Empty		General.Denied		NewState
17	Time:OwnerTime	yes		Blocked		-
18	Time:OwnerTime	no		Blocked		-
<b>19</b>	<b>TimeCheck (synchronous return state)</b>					
22	General	Call+		Blocked		-
23	General	ConferenceC...		Blocked		-
24	Battery:Battery	Full		Blocked		-
25	Battery:Battery	Empty		Blocked		-
26	Time:OwnerTime	yes		General.Answer		NewState
27	Time:OwnerTime	no		General.Denied		NewState

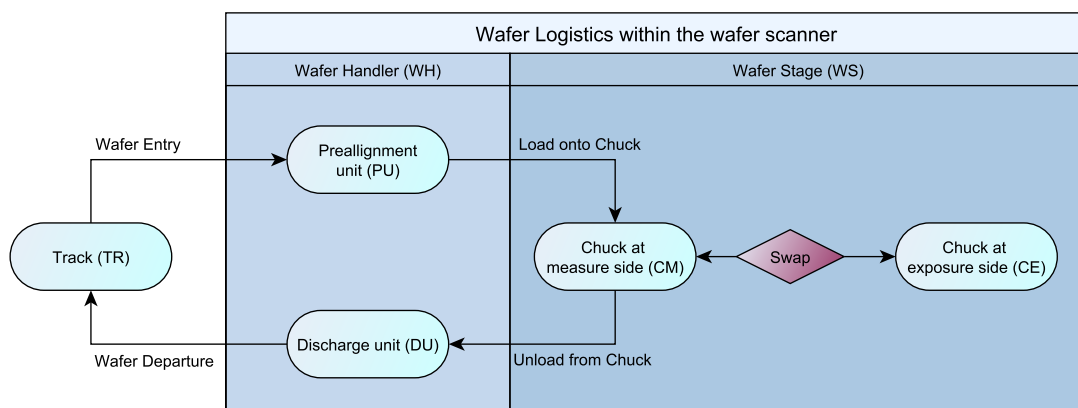
**Figure 2.5:** SBS specification of a design model equal to that of Figure 2.4

ASD has also a run-to-completion policy, which states that the triggered rulecase should be fully executed (actions and state variable updates) before the corresponding state transition can take place. This also indirectly invokes that other incoming calls are blocked until the action sequence of the rulecase is completed.

The monitor semantics of ASD mean that a component can only be used by one client at a time. This means that when another client requests something at the component, it has to wait until the component has finished the action sequence belonging to the first client. Any notifications at the component, belonging to the first client can still be triggered while the second client is served. This is because notification events are asynchronous.

### 3. Lot Operations, Process Wafer

Lot Operations, Process Wafer or LOPW is incorporated in the wafer scanners built by ASML. As described in the introduction these machines are used to perform the lithography step in the production of microchips. These microchips are printed on silicon wafers. These wafers arrive at the wafer scanner in a lot, which is a group of wafers. Each wafer in a lot should undergo the same lithography step. The wafers are already preprocessed and are physically ready to be illuminated. During this step certain parts of the wafer are exposed to the EUV light, which gives them different properties compared to non-exposed parts. Due to differences in material properties the added layer in preprocessing can be selectively etched away.



**Figure 3.1:** *Logistics of wafers within the ASML wafer scanner [3]*

Illumination of a wafer is a delicate practice and before the wafer is ready for this procedure it has to go through certain steps. A lot arrives at the track (TR), which is a wafer providing system for the wafer scanner. A wafer is then placed on the Wafer Handler (WH) at the Pre-alignment Unit (PU). Here the wafer is rotated and translated to the right specifications. When this is done the wafer can enter the Wafer Stage (WS), where it is put on a chuck. The chuck is a moving table, controlled to ensure right positioning of the wafer in the Wafer Stage. The chuck with wafer is at the measuring side (CM) when the wafer enters and the wafer is 3D measured. After this is done the chuck switches places with the other chuck to go to the exposure side (CE). Here the wafer is illuminated, before the chuck switches back to CM. Then the wafer is unloaded to the Discharge Unit (DU) and put back on the track by the track and unload robots. The other chuck is used for the same process simultaneously, but in antiphase. The described logistics process is depicted in Figure 3.1.

#### 3.1 LOPW scope

Lot Operations, Process Wafer (LOPW) is described in the Element Performance and Design Specification (EPDS) as having the responsibility of ‘streamed processing of lots’ and the ‘streamed processing of wafers’ belonging to each lot [3]. This means that the controller allocates the operations of the wafer during the processing stage within the Twinscan machines. LOPW is part of a bigger control cluster concerning the production, which in total is responsible for the wafer logistics in the Twinscan as depicted in Figure 3.1.

The process described in the previous paragraph is a simplification of reality. In reality there are exceptions in the handling of wafers and dummy wafers are present, to prevent empty chucks at the exposure side. This is crucial in immersion lithography systems, since the immersion fluid should remain under the optical system at all times. These features are controlled by the LOPW as well. Wafers are processed by a first in first out (FIFO) principle. This means that wafers can not overtake each other in case something goes wrong at one wafer. The LOPW makes sure this does not happen.

### 3.2 LOPW construction

To control a logistics system as described in the previous paragraph a controller is needed. However due to the complexity of the system, a monolithic controller would be very complex and incomprehensible. Therefore this controller is split up into separate controllers in an hierarchical structure as described in Section 2.1. In case of new requirements or changes in these, it is expected that only a few of these smaller supervisors might have to be changed in order to satisfy these changes. Software engineers are currently changing the controllers manually to implement these new requirements. These requirements are generally written down in LOPW documentation and are more deeply discussed at the end of this section. Although this hierarchical controller is easier to comprehend than one big controller, it is not always directly clear which controllers should be adapted if a requirement changes. The way the requirements are written down is the reason for this. They are not written down per controller, but per functionality of LOPW.

### 3.3 Analysis

Before any further research can be done, it is crucial to perform a good analysis of the ‘what is’ state of LOPW. This chapter will attend to the analysis of the LOPW with regards to it’s history. This will be done first very generally, but will gradually become more specific until a component level analysis is reached. During this process intermediate metric selections have been made to limit the size of the analysis without compromising too much information loss. Later on in this report another metric selection will be made for analysis of selected components, which allows for a more detailed inspection.

Historical data of LOPW ranges back to early 2013. Before this LOPW did exist, but the data belonging to this time frame has not been stored or is too scattered to be useful. Before a general master branch was introduced in the repository of the LOPW, personal branches were used. One of these was marked as the most recent, which was then used on the machines. As these indications are hard to trace back it was decided to focus the view on the master branch. The data from the branch was recovered in early November 2017, which results in a dataset ranging from January 2013 to October 2017. The analysis in this chapter is solely based on models of the master branch which were used in that time period. Since LOPW is older than this time period a startup period is expected to be seen in the analysis due to uploading to this branch in the repository.

### 3.3.1 General Analysis

The data was recovered using a gitminer supplied with the EMF (Meta) Model Analysis Tool (EMMA) [16], recovering all revisions for all existing files within the master branch, placing each revision of a file in a designated folder for that file. If a file was deleted this is indicated as well with the correct time stamp. Using this information LOPW composition can be rebuilt for each time instance if required. The actual LOPW has not been rebuilt for the whole history, but the size of LOPW has been extracted from the data. The size of LOPW is indicated in the amount of independent models at that time.

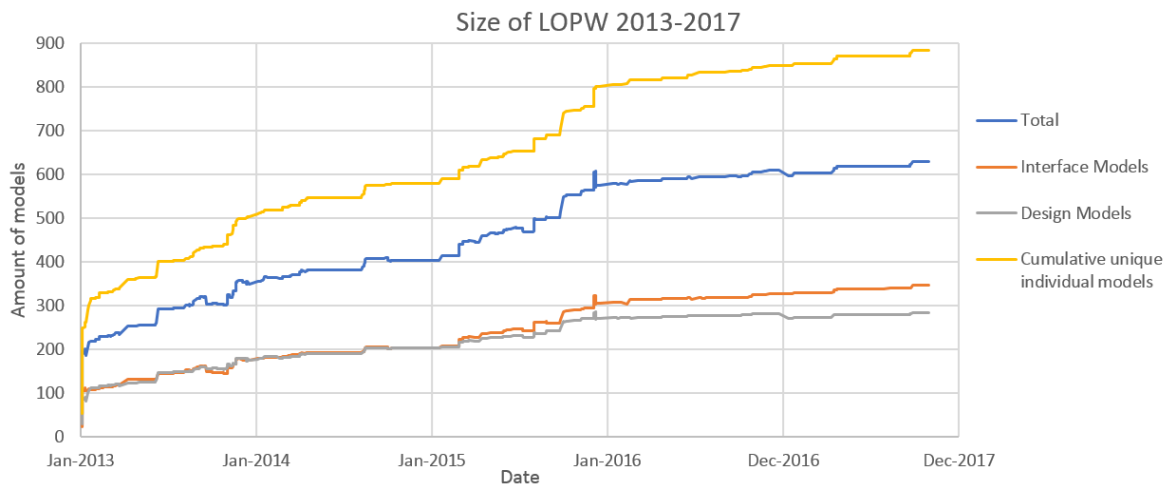
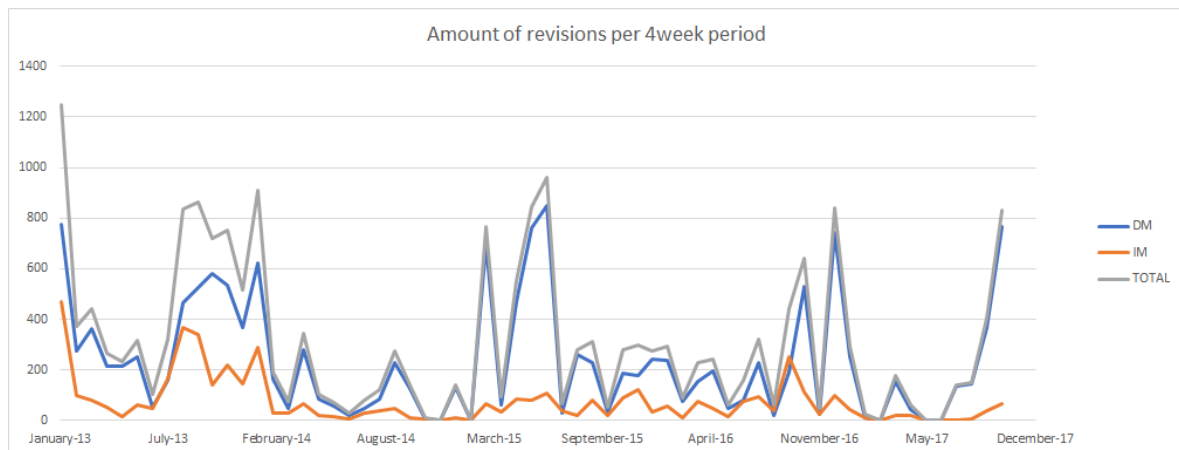


Figure 3.2: Size of LOPW in time

In Figure 3.2 this size is graphically shown. From this picture it is clear that LOPW has grown in size from just under 200 individual models to 630. In this period LOPW has seen 884 individual components, but 254 of them were discontinued. From all these models 45 to 52% is a design model, where the absolute difference is clearly visible in the later parts on the LOPW time line. In the begin this difference between the amount of interface models and design models became smaller and at a time there were more design models than interface models. This is possible since interface models can be defined once, but instantiated multiple times, with different design models for each instance. During 2014 the total amount of models was quite stable and the difference between the amount of interface and the amount of design models was not more than 1 or 2 models, however in 2015 the amount of models increased again and so did the difference between the amount of interface models and design models.

Not only did the amount of models change, the models themselves have changed as well. This results in models with a revision amount ranging from 1 to 228 revisions, cumulatively resulting in a total number of models close to 20.000. These model changes are shown in Figure 3.3. As can be seen the frequency of changes of design models is much higher than those of interface models. The design models make up 75% of all revisions. However, as is discussed later, this is mainly a result of the need of changes in a DM when an IM changes. Something else which is worth mentioning is that the increase in amount of models as shown in Figure 3.2 does not add to the amount of changes per time period.

Notice that in Figure 3.2 the increase of models on the first day is very high. This is due to

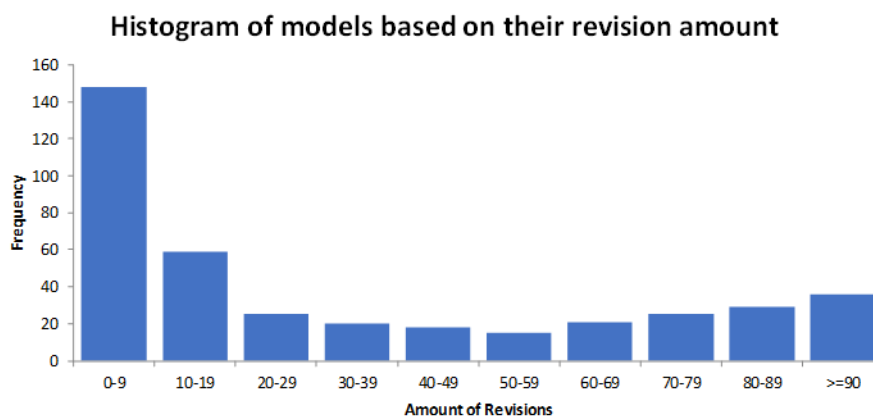


**Figure 3.3:** Changes of LOPW per 4 week period

the fact that LOPW is older than the history of data is available, all changes before this are put on the first day. Since Figure 3.3 depicts the derivative of the size which is the amount of revisions per time period. The time period is taken as 4 weeks since this compensates the variance in upload time of the revisions. The graph displays a large peak at the first period, which is clarified by the large amount of revisions on the first day. The other peaks are created by actual changes in models, but no reason for the density of changes could be found.

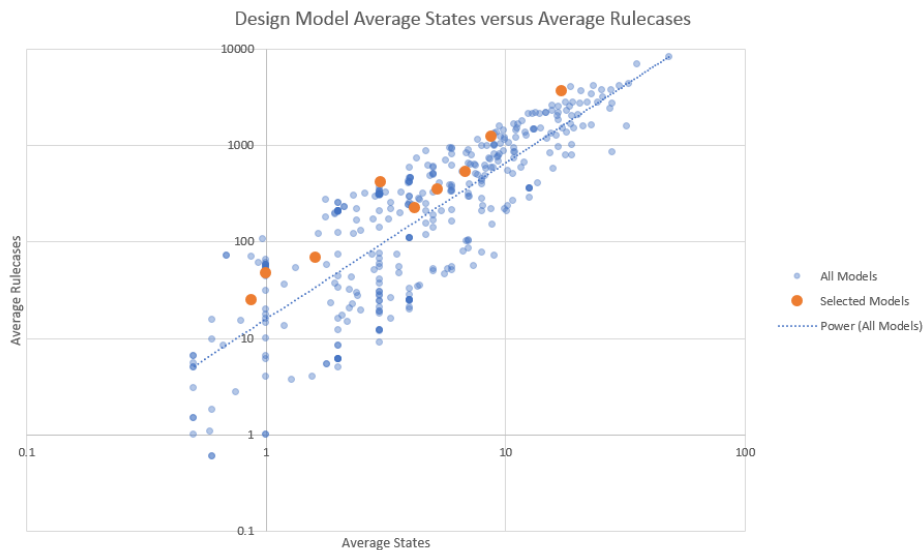
### 3.3.2 Intermediate Metric and Model Selection

As in this research changes between revisions are important, it is cause to use models which have multiple revisions. Using this as only criterion would lead to an immense amount of models which should be evaluated and modelled in CIF as can be seen in Figure 3.4. This is unachievable within the limited time of this research. Therefore certain metrics will be used to select a group of models which will be used for further evaluation. To get a good representation of LOPW, the selected models should be representative for most of the models present in LOPW. The metrics are extracted with EMMA [16].



**Figure 3.4:** Amount of models with a certain amount of revisions

To make sure all kind of models are taken into account it is also relevant to have both simple and complex models. For this the amount of rulecases and the amount of states are taken into account. As can be seen in Figure 3.5 there is a small correlation between the amount of states and the amount of rulecases. Even though there is a correlation, the spread is very wide, which makes it mandatory to evaluate both separately with respect to model selection. Each model is depicted with a blue dot, darker dots mean that several dots overlap and these are models with equal rulecases and states. As the differences in values become bigger when the absolute number increases, the graph is shown on a double logarithmic scale.



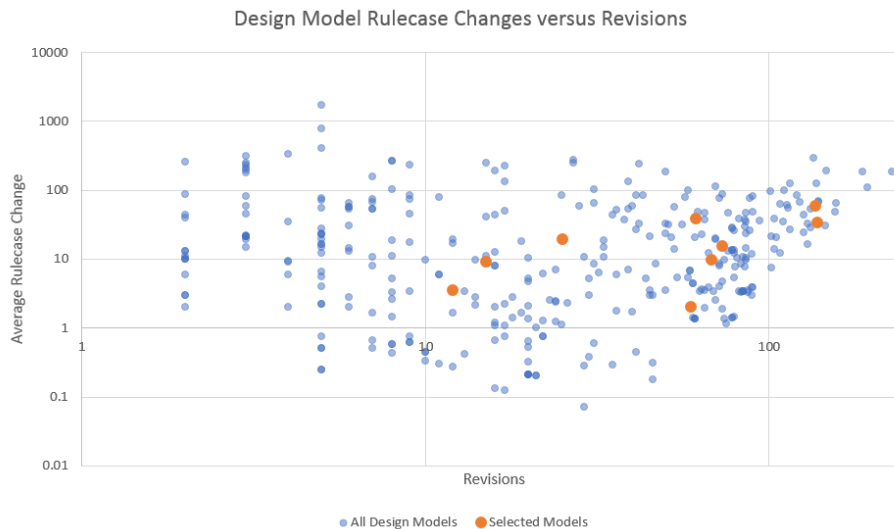
**Figure 3.5:** Average amount of states plotted to the average amount of rulecases.

For this research the amount of change is also important. There are several factors contributing to this: the amount of revisions, the average time between those revisions, change in rulecases, change in amount of states and change in the used interface models. As there is no clear correlation between these metrics, all of these metrics are evaluated separately for all models to make a decent selection of models which would represent LOPW. To ensure that there are enough changes within the scope of time, a minimum of 10 revisions is set.

From Figure 3.6 it can be concluded that there is no strong correlation between the amount of revisions and the average amount of changed rulecases. Which makes it relevant to choose models with a spread in this spectrum. As the differences in values become bigger when the absolute number increases, the graph is shown on a double logarithmic scale.

### 3.3.3 Selected Model Analysis

The model selection was not only done by looking at Figures 3.5 and 3.6. A manual iterative search was also done to identify potential interesting models with regards to the model requirements, which can be difficult to identify from the general view on LOPW that is used till now. However it is expected that the requirement changes can be recognized by changes in the design model if the interface model changes at the same time. Therefore only models with enough of these instances relative to the amount of revisions are taken into account



**Figure 3.6:** Average amount of rulecases plotted to the amount of revisions.

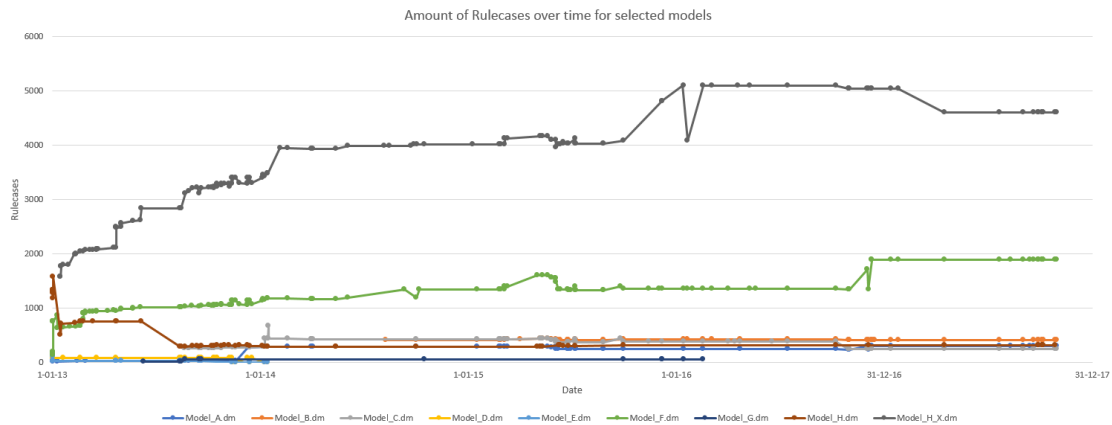
during the iterative search. Exceptions are made for the models used by Loose, which are taken into account regardless of this aspect. These models were taken into account since there was already a start for these models and they could be good reference for models with few requirement changes. There were a few reasons not all of these models were selected:

- No changes in model during entire history.
- Dummy models which are in the LOPW structure, but do not have specific control decision requirements. Therefore we consider them inappropriate for this research, as these requirements play a key role.

The other models were interesting enough for this research. However to complete the selection, more models should be selected to be representative for LOPW. These models are chosen by looking at the graphs of the states, rulecases and cumulative changes of interface models. If the first two showed a lot of change or had a trend, while the cumulative change of interface models was high the model had potential. Then this model was plotted alongside the data of the LOPW analysis graphs shown in Figures 3.5 and 3.6 (and more) to make sure a wide enough coverage of LOPW was realized. This resulted in the models depicted in Figure 3.7.

During the evaluation of the components one extra component was added to the selection. This was done due to a function change of component H. First a single component H was used, but as LOPW is suitable for all wafer scanners of ASML, the component was split. Component H became a forwarder to either component H\_Y or H\_X, depending on the machinetype. Due to this change one of these components was added. As can be seen in Figure 3.7 this component is rather large, which is why this component was not selected in the first place. This component was used in the research, but due to its size not evolved over the entire timescope. The reason for this will be discussed in Chapter 6.





**Figure 3.7:** Rulecase amount evolution for the selected models.

### 3.4 Requirements

The ASD components are built with regard to some requirements. Interface models are used to model the possible behaviour, where the Design models implement the requirements to enforce the correct behaviour. The requirements are written down in the EPDS [3] per feature of LOPW. Each feature has several requirements, which can be implemented in different subcomponents of LOPW. This makes the traceability of the requirements written down in the EPDS very difficult. Some models are not mentioned in the EPDS, which makes traceability impossible. In addition to this the document is not updated simultaneously with the models, which makes it impossible to link the requirements in the documentation to a certain LOPW revision. The only method left to obtain the requirements is to extract them from the ASD design models. This is a difficult and risky method, since some requirements might be only implementation choices and not actual component requirements. However since this was the only method left this methodology has been chosen.

#### 3.4.1 Requirement Classification

To make a distinction between different kind of requirements a classification is made of them. Requirements can be categorized on aspects like the source of a requirement [15], the functionality [21] or the construction. Where functionality sorts the requirements based on the function of the requirement, e.g. a requirement which states that the response time should be smaller than 1 second. The construction categorization divides the requirements in segments based on the influence on the coding, e.g. a requirement is written for flexibility so the code can be used in multiple machines. As the requirements are all extracted from the design model, the source is untraceable further than that. The requirements that are extracted should all be functional requirements, as the domain specific requirements do not define how the component should work. These domain specific requirements implemented in the model which are necessary to comply with the syntax of the language.

The extracted requirements from the design model are formal functional requirements, which can be further categorized. During analysis of the components the following requirement categories were encountered:

### 1. Sequence Requirements:

These are the most common requirements within the components. They are set up in a way shown in Equation 3.1. From the design model it might be that an event A can initiate multiple events. There can also be state variable conditions attached. Event B should directly succeed event A. In case multiple events are required after A, these are executed directly in arbitrary order after A.

$$\text{When A occurs, do B (and C)} \quad (3.1)$$

### 2. Information Request Requirements:

Some events require specific data which is not given to the component through incoming events. Therefore an extra event must take place before a certain action can take place. In Equation 3.2 such an occurrence is given. The component does get information on attribute X, but not on Z. Before action B can take place an action A should happen, which gathers the required data.

$$\text{Before B(Z) can occur, A(X,Z) must occur} \quad (3.2)$$

### 3. Condition Requirements

Condition requirements are sequence requirements which are based on a decision as can be seen in Equation 3.3. An action True does not necessarily mean to do X. Neither does A mean to do X, because when the response to A is false this might not be the case. These are different from Equation 3.1, as there can be more action sequences which reply with True. The sequence A-True should be executed before X is executed.

$$\text{When action A is followed by a response True do X} \quad (3.3)$$

### 4. Data Requirements

Data requirements are condition requirements with memory. Based on a certain activity history the current action is determined. An example of this is given in Equation 3.4. X should only be executed when B was executed last of A and B. If X is filling gas tank of a car, this should only be done if you took the car (B) and not when you took the bike to work(A).

$$\text{Do X when B was executed last from [A,B]} \quad (3.4)$$

## 4. Supervisory Control Theory and Application

This chapter will elaborate on Supervisory Control Theory. First a brief introductory of supervisor synthesis is given in Section 4.1, after which the supporting language CIF is explained and how to generate supervisors in Section 4.2. In Section 4.3 the transformation from SBS to CIF is explained, followed by the application of this method to LOPW in Section 4.4.

### 4.1 General

Supervisory Control Theory [19] is a method to create controllers for (hybrid) discrete event models. Formal discrete event models specifying the uncontrolled system are called plants and can be combined with formal discrete event requirements to generate a supervisor. These plants form the uncontrolled system and are themselves composed of states and transitions between those states. The state transitions are either controlled or uncontrolled, which is important to the synthesis. An uncontrolled event can't be disabled by the supervisor, an example could be an user input. The controlled events are enabled or disabled by the supervisor.

In addition to the plant automata there should be requirement automata specified. These formalized requirements specify which behaviour the system should show. Synthesis will form a supervisor which ensures the system will behave according to the specified requirements. The requirements can specify that certain events should only be executed in a certain order. The supervisor will disable the events unless the events prior to them in the sequence have occurred.

Marked states are another important attribute for supervisor synthesis. From any state there should always be a marked state reachable to prevent livelock and deadlocks. If this is not possible the current state is a blocked state, which will then be blocked by the supervisor. Note that deadlock is still possible in marked states. Every automaton needs at least one marked state. Since a supervisor can control multiple plant automata, the whole system should always be able to reach a global marked state. This global marked state is when all automata are in a marked state.

The result of synthesis is a safe supervisor which makes sure the plant behaves according to the requirements. In addition to this it ensures that a marked state can always be reached (non-blocking) and that the system is maximally permissive. Maximally permissiveness means that only events are disabled to ensure controllability, behaviour according to the requirements and non blocking behaviour, all other events are allowed. All this is done while keeping the controlled system controllable, which means that all uncontrollable events are not disabled.

Supervisory control theory to synthesize a supervisory needs a language and tool to apply it to practical cases. The next section describes this language and the toolset used.

### 4.2 CIF

The Compositional Interchange Format (CIF) [2] is a modelling language which allows for supervisor synthesis as described in the previous section. This technology has been developed at the University of Technology in Eindhoven at the Systems Engineering group of the

Mechanical Engineering Department. CIF has many possibilities, like conversions to verification languages [6] and visualisations. This has been used in previous research, for instance in [13]. The research in this thesis will continue the research into the application of CIF in combination with supervisor synthesis.

In this research CIF is used to synthesize controllers based on the ASD models of LOPW. How this is done is described in the following sections, but it is important to note that ASD uses a hierarchical modelling structure. CIF has a one level structure, which means that there is no such thing as hierarchy. This hierarchy should be manually implemented. Previous research on these topics is done [13] [14] [20], which describes this in more detail. Here only the most important parts and differences are elaborated.

Supervisory control, as described in the previous section, can be done in CIF by modelling plants and requirements. A plant can be modelled with locations and controllable and uncontrollable events. A simple button is modelled in Listing 4.1.

---

```
1 plant button:
2   uncontrollable u_pushed, u_released;
3   location Released:initial; marked;
4   edge u_pushed goto Pushed;
5
6   location Pushed:
7     edge u_released goto Released;
8 end
```

---

**Code Listing 4.1:** *Plant model of a button*

This plant models the behaviour of a button, which can not be controlled by the supervisor. It can switch from being Pushed to Released with a transition ‘u\_released’. A controlled plant can be a lamp, which can be turned on or off by the supervisor as depicted in Listing 4.2. With the transitions ‘c\_on’ and ‘c\_off’ the lamp can switch from the Off to the On position and vice versa.

---

```
1 plant lamp:
2   controllable c_on, c_off;
3   location Off:initial; marked;
4     edge c_on goto On;
5
6   location On:
7     edge c_off goto Off;
8 end
```

---

**Code Listing 4.2:** *Plant model of a lamp*

These plants can be linked. Due to synchronizing of events this can be done by a requirement which is depicted in Listing 4.3. It uses the same events as declared in the lamp plant. Therefore they will occur simultaneously. This requirement ensures that the event ‘lamp.c\_on’ can only take place when the button is pushed. Equally it can only turn off when the button is not pushed any longer.

```

1 requirement A:
2   location: marked; initial;
3   edge lamp.c_on when button.Pushed;
4   edge lamp.c_off when button.Released;
5 end

```

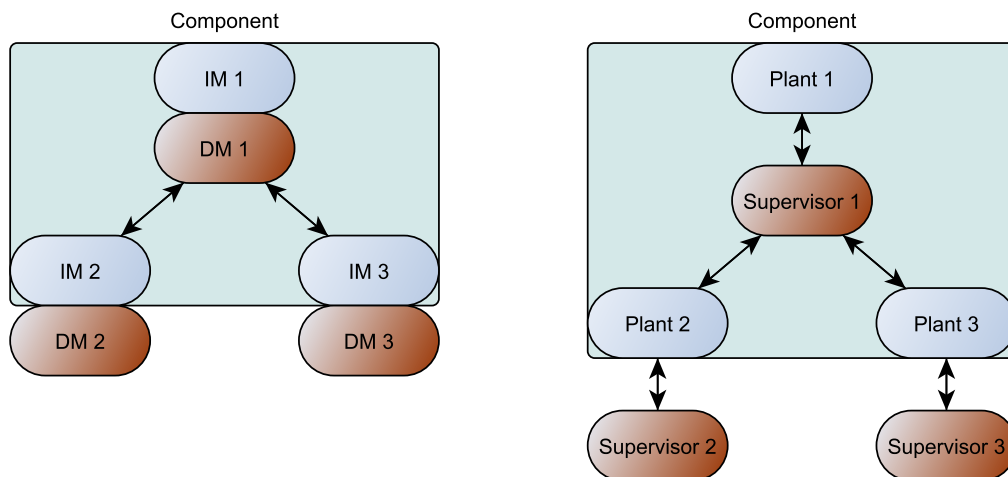
**Code Listing 4.3:** *Requirements for button and lamp*

This is a simple example which will behave correctly. However to ensure a safe, non-blocking and maximally permissive controller synthesis should be applied. Synthesis restricts the plant behaviour in such way that the requirements are met and the controller is safe, non-blocking and maximally permissive [17].

### 4.3 SBS to CIF transformation

This section will elaborate on the transformation from the ASD models to CIF. The main method is derived from the work of Loose [13], which used the method on LOPW components as well. Other research has been done to apply this method on other systems [23] [7]. This chapter will briefly describe the method used and point out any differences in the methodology.

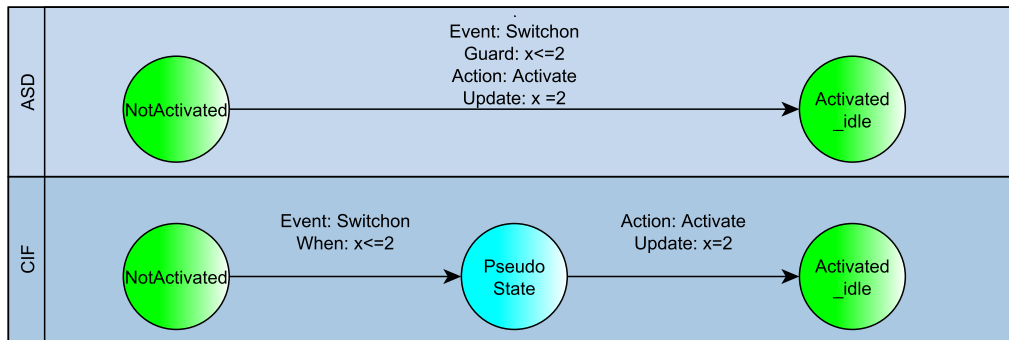
In ASD the interface models show all possible behaviours, which conforms with the definition of the uncontrolled system within the supervisory control theory. Therefore the IMs are transformed into plants, which then can be used for synthesis. The design model is the equivalent of the supervisor, because both tell which behaviour is allowed on the uncontrolled system. In Figure 4.1 a visualization of this process is given. Note that plants 2 and 3 are linked to another supervisor, which needs to be synthesized when looking at that specific component.



**Figure 4.1:** *Simplified view of an ASD component and its SCT equivalent.*

To translate the semantics of the rulecases in CIF, pseudo states have to be introduced. CIF does not allow multiple events to take place at one single event, where ASD uses rulecases

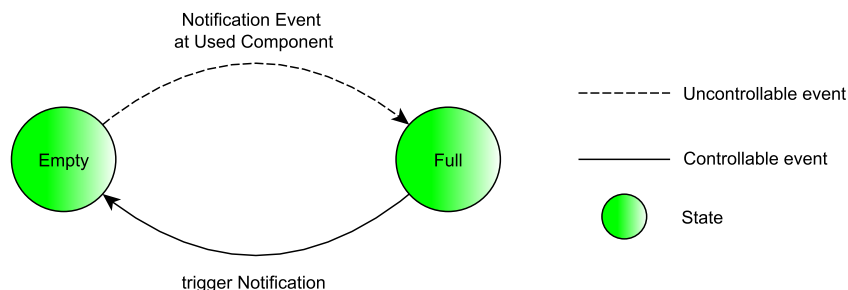
to execute multiple events in sequence. To convert this in CIF intermediate pseudo states are introduced to allow multiple events to take place before the actual state transition takes place. An transformation of the rulecase given in Figure 2.2 is shown in Figure 4.2 with a transition diagram. Note that the guard only needs to be on the first transition, as the whole rulecase should be executed in one go, which is also ensured with this conversion method.



**Figure 4.2:** A transition diagram of a rulecase in ASD and its transformation into CIF.

The controllability of the events and actions depend on the interface. When evaluating the service interface (IM 1 in Figure 4.1) the events are uncontrolled, as they are triggered by events outside the component. The replies that interfaces give are controlled. The used interface models (IM 2 and IM 3 in Figure 4.1) have this the other way around, as the triggers are controlled, but the replies are defined by the control sequences outside the component.

Besides normal synchronized events and actions, ASD can have notification events, which are processed asynchronously. These events in used services might trigger a response in other plants, but they do not have to occur immediately. CIF does not support such an event, which is why for each notification event in a used interface a queue should be introduced as shown in Figure 4.3. This queue allows a notification event to occur, but delays the required response. Note that this is not exactly how ASD handles these queues as the events are not handled FIFO and the queue size is limited to one. This issue is more extensively elaborated in Chapter 6. When there are any notification events in a queue, all incoming client calls are blocked, since the notification event should be handled first. However the client calls are uncontrolled, which means they can't be blocked by the supervisor.



**Figure 4.3:** A transition diagram of a queue automaton for a single notification event.

To solve this the incoming requests at the client (Plant 1) should be made controllable. Another reason to do this are the monitor semantics and run to completion semantics. These say that if a trigger happens with action on an used service, the whole rulecase should be executed in one go. However the monitor semantics say that an used component can only be used by one client at the same time. Therefore, when a used service is busy, the request has to be blocked, since the rulecase can't be fully executed due to the occupation of the used service. This is not a result of the model itself, but the monitor semantics of models within ASD. This results in the controllability of events as depicted in Figure 4.4, where uncontrollable events are depicted with a dashed line and controllable events with a continuous line.

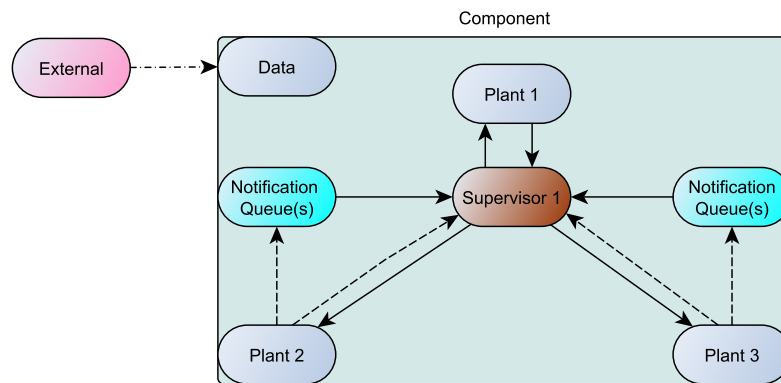
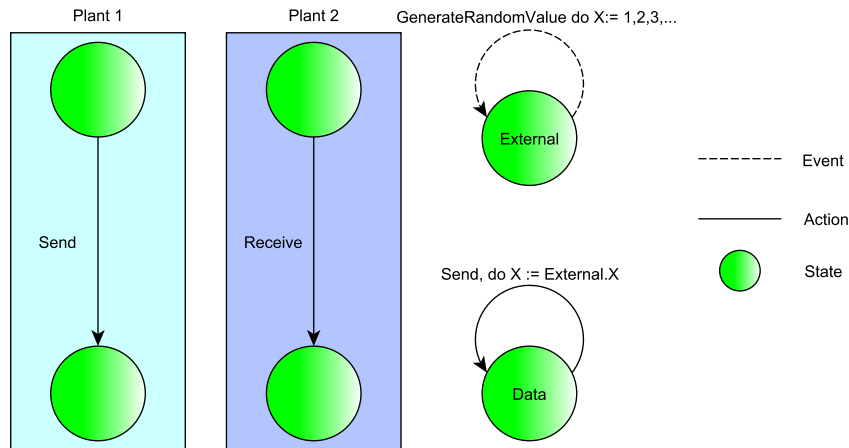


Figure 4.4: All modelled components in CIF

Besides controlled and uncontrolled events to define behaviour in CIF, there are the marked states which influence the synthesis procedure. This can be used in combination with requirements to ensure the run to completion, but also to make sure the notification events are handled before any more requests come in. This is done by marking every normal state within the plants, but not the pseudo state. A same thing is done for the queues, where the empty state is marked, but the full state is not. This will result in the desired behaviour.

Within the ASD interface models, events and actions can send and receive data. This data is then transferred from a trigger event to an action in the DM, where both trigger and action can origin from a different IM. This data is not changed, just transferred from one interface model to another in the design model. This can be modelled in CIF, but does not have any influence on the supervisor synthesis, since decisions and transitions are not based on this data. This is the reason it was not included in models in earlier research [13]. In this research however, ASD and CIF will be compared. To reduce the amount of differences between the two modelling languages, the data transmission is included in this research.

To this end a data storage unit is created. This data storage plant will update its value when data should be transferred. To receive data a single plant with an uncontrollable event is introduced which generates random data in a predefined range. This can be seen in Figure 4.5. Where these are implemented in the component is visualized in Figure 4.4. Each time data should be transferred, the data storage value is updated to the data external data value. The events which should send data are therefore all mentioned within the data plant (Send event in Figure 4.5). Since it is out of scope how the interaction between the CIF and ASD components would be during implementation, this is a simplification. However it allows for



**Figure 4.5:** *A transition diagram of a data transmission.*

the data to be read by outside components as well as being updated, for instance when event Receive in Plant 2 takes place. As synthesis does not allow arbitrary data types, a simplified ranged integer is chosen to be representative for the data to be transferred. If the data would be more extensive in the actual system, the data type can be changed after synthesis in the plants. The supervisor will still be valid, but the data type within the plants is upgraded.

Special actions of ASD might be integrated in the plant models. NoOp can and should be integrated, which is simply a single event in CIF without pseudo states. Blocked events only occur in design models, so should not be included. Illegal and disabled events can also be left out, since they are there to adhere to the completeness requirement of the SBS syntax. CIF blocks events automatically in these cases, which makes it unnecessary to include them.

#### 4.4 Application to LOPW

The SBS specification is used to transform all interface models into CIF plant models. In addition requirements should be added. In Section 3.4 it is stated where the requirements come from and were categorized, these are transformed into usable requirements. Each category has a different kind of requirement coding, which is depicted below.

1. Sequence Requirements:

A simple requirement automaton with two states is introduced. As the action sequence should be fully executed before anything else can happen according to the ASD plant behaviour, only the initial state should be marked as can be seen in Listing 4.4.



---

```

1 requirement Forward_Event_A :
2   location idle: initial;marked;
3   edge Main1.Trigger goto Fwd; // When A occurs (Equation 3.1)
4   edge Plant2.Trigger goto Fwd;// When C occurs (when combined)
5   location Fwd:
6   edge Plant1.Action goto idle;// Do B      (Equation 3.1)
7 end

```

---

**Code Listing 4.4:** *Example of a sequence requirement*

This is the most simple requirement to ensure action sequences. When multiple triggers lead to the same event this can not be modelled in two separate sequence requirements. This would require both triggers to occur before the event can be executed. Therefore these requirements need to be combined, which leads to the addition of line 4 in Code Listing 4.4.

### 2. Information Request Requirements:

This requirement is modelled in a same manner as the sequence requirements. However multiple events can have the requirement that an action should take place before that event, which is visible in Listing 4.5. For the same reasons as for the sequence requirements, requirements which require the same action to be executed first should be combined.

---

```

1 requirement Info_Request_A :
2   location idle: initial;marked;
3   edge Plant1.Action goto Fwd;// A must occur(Equation 3.2)
4   location Fwd:
5   edge Plant2.Action goto idle;// Before B can (Equation 3.2)
6   edge Plant2.Action2 goto idle;// Before C can (combined)
7 end

```

---

**Code Listing 4.5:** *Example of an information request requirement*

### 3. Condition Requirements:

These requirements are accompanied with a monitor. This monitor will keep track of events, but does not block them. First a requirement of the sequence type will make sure 'Plant1.Action1' is triggered. This is monitored as can be seen in Listing 4.6. Now there is a 'memory' of which action has been taken. As there might be more events with an equal response, it is necessary to remember to which event the reply belongs. Then there is a reply, which in the case of being 'True' should invoke line 17, but only if the reply is indeed a reply on the right event, which in this case is 'Plant1.Action1'. This is done by only transitioning to 'Fwd' in requirement 'Result\_OK' when the monitor indeed indicates that the reply is the reply on the right action.

---

```

1 plant Plant1Monitor :
2 monitor;
3   location idle:initial;marked;

```

---

---

```

4     edge Plant1.Action1 goto A1;
5     // When action A (Equation 3.3)
6     // occurs remain in A1 until response is given
7     location A1:marked;
8     edge Plant1.TRUE,Plant1.FALSE goto idle;
9 end
10
11 requirement Result_OK:
12 monitor Plant1.TRUE;
13     location idle:initial;marked;
14     edge Plant1.TRUE when Plant1Monitor.A1 goto Fwd;
15     // if True is the response on A (Equation 3.3)
16     location Fwd:
17     edge Plant2.dosomething goto idle;
18     // Do X (Equation 3.3)
19 end

```

---

**Code Listing 4.6:** *Example of a condition requirement*

#### 4. Data Requirements:

Data requirements are also in need of a monitor. This monitor will change location when one of the events required in memory occurs. Based on the state of the monitor a certain sequence of events may take place. The extra Event3 instance on line 15 in Listing 4.7 is there to ensure that the action is not blocked when this event is triggered when Event1 is not in the memory. This is necessary when Event3 is triggering other events in all cases.

---

```

1 plant Event1Event2Monitor:
2 monitor;
3     location Event1:initial;marked;
4     //When B was executed last from [A,B] (Equation 3.4)
5     edge Plant1.Event2 goto Event2; //Event A
6     location Event2:marked;
7     //When A was executed last form [A,B] (Equation 3.4)
8     edge Plant1.Event1 goto Event2; //Event B
9 end
10
11 requirement X:
12     location idle:initial;marked;
13     edge Plant1.Event3 when Event1Event2Monitor.Event1 goto Fwd;
14     //Do X when B was executed last from [A,B] (Equation 3.4)
15     edge Plant1.Event3 when not Event1Event2Monitor.Event1;
16     //Allow B to occur, but don't do X (Equation 3.4)
17     location Fwd:
18     edge Plant2.dosomething goto idle; // Event X (Equation 3.4)
19 end

```

---

**Code Listing 4.7:** *Example of a data requirement*

In addition to this a mandatory requirement is needed to make sure the plants are behaving according to the ASD semantics. Since we assume to have the ASD interface models as a

starting ground this is necessary. The client plant is the CIF equivalent of the client IM in ASD and the used plants are the equivalents of the used interfaces in ASD. The requirement for each component is that the trigger events in the client plant should only be allowed when all used plants are idle, which is done with a requirement invariant like shown in Listing 4.8. Here is the client portrayed by plant Main and the used plants are Plant1 and Plant2.

---

```
1 requirement {Main.Trigger1 ,  
2             Main.Trigger2} needs Plant1.Idle and Plant2.Idle;
```

---

**Code Listing 4.8:** *Example of a semantic invariant requirement*

If an event can be triggered in multiple states of the plant model and has different action sequence requirements in each state, the activity should be mentioned like requirement X in Listing 4.7. Another option is to monitor the activity, so it will not be blocked in this requirement. This is possible for events which are in the client plant, however notification events are not explicitly mentioned there, so this would be wrong. To solve this problem an extra requirement invariant is introduced like the one above for all notification events, which are only allowed to be handled if the client plant (Plant 1 in Figure 4.1) is in a marked state. This means the queues can only be emptied when the client plant is in a marked state as can be seen in Listing 4.9.

---

```
1 requirement {Queue1.Trigger ,Queue2.Trigger}  
2             needs Main.MarkedState1 or Main.MarkedState2;
```

---

**Code Listing 4.9:** *Example of a notification invariant requirement*

After the plants are modelled correctly and the requirements are according to specification, supervisor synthesis can be applied using data based synthesis. The result is a supervisor which ensures the right behaviour of the uncontrolled plants. When required, the data which should be transferred can be changed as described earlier.

## 5. Metrics

Metrics can be used to quantify ASD and CIF models. However there are a lot of software metrics available in the ICT scene, this research focusses on state machine models and therefore metrics specific for models are used. The chosen metrics are relevant to the research questions and both applicable to ASD models and CIF models. First the metric selection is described, how they are applied to ASD and it is stated why these metrics are used. Several metrics for ASD are extracted with EMMA [16] from the models and then used for computation of more complex metrics. JAVA scripts are used to use EMMA and can be found in Appendix A. The second part of this chapter will specify how each metric can be computed for CIF. For CIF a Python script is used to extract the metrics, which can be found in Appendix B.

### 5.1 Metric Selection

The reason to use metrics is to get quantifiable data to compare two different controller development methodologies. Preferably metrics to quantify development are used [5]:

- Shortfall: Measurement of how much the software does not meet requirements at a certain time;
- Lateness: Determines the duration of time between a new requirement and implementation;
- Adaptability: Rate of adaptation to new requirements;
- Inappropriateness: Is the total amount of time combined with the amount of shortfall.

However due to the lack of data regarding time and the immense time consumption a practical experiment would take, these metrics shall not be used to compare the two methodologies at stake. Instead metrics to compare the resulting models from both methodologies are used.

There are a lot of metrics to quantify software. Earlier research [8] has used a set of metrics to quantify the size and quantity of the ASD models. These are used for ASD and adjusted for CIF. In addition to these metrics one of the most commonly used metrics for software is used, which are function points [11]. With this set of metrics the models can be evaluated with regards to size and complexity.

Metric categories which are excluded are related to errors and failures, since the data is not available and impossible to obtain within this research. Only estimations of errors can be made. For the same reason the methodologies metrics are not used, the time related metrics are not used here as well, except estimations.

The metrics are divided in 4 different groups. The first one contains the Counting metrics, the second one are the General Model metrics and the third are the Halstead metrics. The Change metrics make the list of groups complete. These groups and their metrics will all be explained one by one in the coming subsections.

## 5.2 Counting Metrics

The first group contains 8 metrics, which describe amounts of occurrences of certain model attributes. These are easily determined by counting the amount of occurrences in a model. Table 5.1 shows these metrics with a short explanation.

**Table 5.1:** *Attribute occurrence metrics*

Metric	ASD	CIF
States	The amount of states	The amount of locations
Actions	The amount of actions	The amount of controllable events
Rulecases	The amount of rulecases, excluding blocked, illegal and NoOp actions	The amount of edges
Events	The amount of trigger events	The amount of uncontrollable events
State variables as guard	The amount of variables which are used in a guard	The amount of variables which are used in a guard
Data variables in events and actions	The amount of data variables used in events and actions	The amount of data variables which are updated
Operators on state variables	The amount of operators used on state variables	The amount of operators used on state variables
Operators on data variables	The amount of data storage and call operators	The amount of data assignments

The states, actions, rulecases and events are already explained in Chapter 2. The state variables in a guard metric is the amount of state variables which is actually used in a guard. State variables which are not used as guard will not be counted. A similar thing is happening with the data variables in events and actions metric, which only counts the data variables which are sent or received in at least one action or event. The operators are the additions which can be used in combination with the previous metrics. An example for state variables is an equal sign and for data variables a save operator. As a result the operators on state variables metric must be zero when the state variables as guard metric is zero. The data variables metrics have an equal relation.

## 5.3 General Model Metrics

In addition to the first eight metrics, which were only counting attributes, there are several metrics which give a better insight in the size and complexity. Table 5.2 is filled with these metrics.

**Table 5.2:** *General Complexity metrics*

Metric	
LOC	Lines of Code
CC	Cyclomatic Complexity
ACC	Actual Cyclomatic Complexity
Function Points	Function Points

The lines of code (LOC) is a standard metric for software [11]. The LOC is simply the amount of lines the code has. It is generally counted in either physical or a logical way. The physical way is the easiest and simply counts all the lines visible on the screen, which includes non functional pieces of text within the code. How the logical LOC (also called source lines of code or source instructions) is counted is dependant on the syntax of the code, but always only counts executable code. For ASD this is the summation of the Actions and Rulecases metric. One might say that only the rulecases are the lines of code, however one rulecase can have multiple actions, which can be read as lines.

The Cyclomatic Complexity (CC) [8] is a metric which gives a number to the amount of linearly independent paths in a code, when this code is transformed into a directed flow graph. These are the paths which are different from each other and are not constructed from other paths. The CC can be calculated by Equation 5.1, where  $E$  is the amount of transitions and  $N$  is the amount of states.

$$CC = E - N + 1 \quad (5.1)$$

In addition to this the Actual Cyclomatic Complexity (ACC) [8] can be computed. This merges all transitions which have the same origin and target state into one. These are then called unique transitions. The equation to compute the ACC is very similar to Equation 5.1, the only difference is that  $E$  is replaced by  $E_U$ , as denoted in Equation 5.2. This variable denotes the amount of unique transitions.

$$ACC = E_U - N + 1 \quad (5.2)$$

Function Points (FP) are another way of measuring the size of a piece of code. A function is a piece of code that will perform a certain task [11]. To obtain the FP the first step is to apply Equation 5.3 to obtain the function counts. As there are 5 different types of functions and 3 types of weighing per type according to Kan [11] a double sum is done. The parameters summed are the amounts of functions per type multiplied by their weighing.

$$FC = \sum_{i=1}^5 \sum_{j=1}^3 w_{ij} \cdot x_{ij} \quad (5.3)$$

The second step requires assessment of 14 system characteristics on their influence on the program in a range 0 to 5. Using Equation 5.4 [11] the FP can be calculated, where the value adjustment factor (VAF) is denoted between the brackets and is comprised from the summation of all characteristic assessments.

$$FP = FC \cdot \left( 0.65 + 0.01 \cdot \sum_{i=1}^{14} c_i \right) \quad (5.4)$$

However the ASD suite has a built in function to express the size of a model in function points. It calculates the function points with a single weight factor per function. In addition to this the VAF is equal to one [24]. This results in Equation 5.5. Where  $x_i$  are the different

functions. The weight factors are specified as  $w_i$ . Table 5.3 shows which functions and weight factors are used for ASD.

$$FP = \sum_{i=1}^5 w_i \cdot x_i \quad (5.5)$$

Since this function point metric is already built in and therefore equally used for all ASD applications, this metric is chosen.

## 5.4 Halstead Metrics

The Halstead metrics [9] quantify the effort it takes to comprehend, service and develop a program, based on the counting of operators and operands. The operands are assumed to be the state variables used as guards, states of the model and data variables in actions and events. The Operators are assumed to be the events, state variables operators and data variables operators [8]. It allows for quantifying complexity for any software language, which makes it very suitable for this research. The base of the metrics are the unique and total number of both operators ( $n_1, N_1$ ) and operands ( $n_2, N_2$ ) and combinations of these ( $n_1 + n_2 = n, N_1 + N_2 = N$ ). Using these five main metrics can be defined:

### 1. Volume

$$V = N \cdot \log(2n) \quad (5.6)$$

The Halstead metric of volume is based on the amount of mental comparisons a software engineer should make when writing a software package of length N.

### 2. Difficulty

$$D = \frac{n_1}{2} \cdot \frac{N_2}{n_2} \quad (5.7)$$

Halstead assumes that adding new operators, but reusing the existing operands, will make the program more difficult to understand. Therefore a higher value for D means that a program is more difficult. The unique operators divided by two multiplied by the average amount of occurrences of each operand will give the difficulty.

### 3. Effort

$$E = D \cdot V \quad (5.8)$$

The effort quantifies the mental effort of the developer which is necessary to develop or comprehend the program. For all mental comparisons (Equation 5.6) the developer should make several elementary comparisons (Equation 5.7). Multiplying these will give the required effort.

### 4. Time required to understand the model

$$T = \frac{E}{18} \quad (5.9)$$

Halstead empirically determined that the amount of elementary mental comparisons of an human is around 18. Combined with the required effort this would lead to the total time required to understand the model.

### 5. Expected number of Bugs

$$B = \frac{V}{3000} \quad (5.10)$$

Halstead empirically determined an estimation for the amount of bugs in the code in relation to the volume. Based on this research the estimated amount of bugs is a fraction of the volume.

Using the Halstead metric Volume, the LOC and CC, a maintainability index can be quantified as used in [8]. Equation 5.11 gives the maintainability index. This index indicates how well maintainable the model is. For good models the value should be at least 65, but preferably above 85.

$$MI = 171 - 5.2 \cdot \ln(V) - 0.23 \cdot CC - 16.2 \cdot \ln(LOC) \quad (5.11)$$

Microsoft has implemented an adjusted version in Microsoft Studio, which returns a maintainability index between 0 and 100. If this MI is used, the index for a good model is above 20. The function to obtain this index is similar to Equation 5.11 and is depicted in Equation 5.12. Where the cyclomatic complexity is replaced by the actual cyclomatic complexity. Other than this adjustment it is only scaled to be between 0 and 100.

$$MI = MAX(0, (171 - 5.2 \cdot \ln(V) - 0.23 \cdot ACC - 16.2 \cdot \ln(LOC)) * 100/171) \quad (5.12)$$

### 5.5 Change Metrics

Besides metrics which quantify a certain model, it is mandatory for this research to have some metrics which identify changes between two revisions. Many metrics can just be computed per revision and then compared, e.g. two different FP values. However this indicates a change, it does not indicate how much has changed. Two completely different models might actually have the same FP value. To overcome this problem the changed source instructions (CSI) are used [11]. These are the new lines and changed lines of code. In combination with the deleted lines of code and the amount of LOC of the previous revision the new amount of LOC or shipped source instructions (SSI) according to Equation 5.13 [10]. With  $d$  indicating the deleted amount of code and  $cc$  the changed amount of code, which will ensure it is not counted double, since it is also incorporated in the CSI.

$$SSI_{new} = SSI_{old} + CSI - d - cc \quad (5.13)$$



## 5.6 Metrics application to CIF

To be able to compare the CIF models to the ASD models, it is convenient to have the same metrics used for both languages. The general functioning of the metrics is explained in the previous subsections. This subsection will elaborate on the application of these metrics to the CIF syntax and the differences with respect to ASD. Note that the metrics are defined to facilitate the comparison with the ASD metrics.

### 5.6.1 Counting Metrics

The counting metrics are equally applied within the CIF syntax. However some other things have to be counted to be compatible with the metrics, since CIF does not have some attributes which ASD has. The states is not one of them, since they can be counted in both languages. The actions however do not exist in CIF, in CIF there are only events, but there is a distinction between controllable and uncontrollable events. Since the actions only occur when an event happens in ASD, it can be seen as controllable. Therefore the controllable events will be counted instead of the actions. The trigger events will then be counted as the uncontrollable events. Since the modelling of actions and events is not directly copied to CIF in terms of (un-)controllable events as described in Section 4.3, there are differences expected here.

In ASD the rulecases are counted, but CIF does not have such an attribute. The closest related attribute in CIF is an edge. Therefore instead of rulecases edges are counted. The other counting metrics are not changed relative to the ASD metrics, except the data variables in events and actions. CIF can not send data along events when using synthesis, but it does allow for data variable updates on edges. This is counted as a data variable in an event. As this is simplified in our modelling into a separate plant model, only this one has data variables and the comparison will be very skewed.

### 5.6.2 General Model Metrics

The general model metrics of the cyclomatic complexity are exactly the same as in ASD, since they are not based on syntax level, but on model level. The LOC however is syntax dependent. For ASD we had the amount of rulecases and the amount of actions. In CIF we count the amount of edges, since they are defining the amount of executable code. Although CIF has many more lines to define locations, events and variables, these are not counted since this is done differently in ASD. ASD does not need lines of code to do this, since the modelling environment allows for placement elsewhere. In contrast with the ASD:suite, the CIF modelling environment does not have a built in function to calculate the function points. To match this with the ASD function points the same definition of the function points is used, while replacing the entries for this function with CIF equivalents. Table 5.3 shows the ASD and CIF input for Equation 5.5.

**Table 5.3:** *Entries for Funtion Point Equation*

$w_i$	$x_i$ (ASD)	$x_i$ (CIF)
14	Trigger Events	Uncontrolled Events
2	Guard Expressions	Guard Expressions
5	Rule Cases	Edges
1	State Variable Updates	State Variable Updates
1	Action Events	Controlled Events

All other metrics are based on these metrics. To obtain the other metric values the same formulas and methods as described the previous paragraphs should be used, but using the replacements for CIF as described in this section. The change metrics are equally applied to CIF as they are applied to ASD.

## 6. Methodologies Comparison

According to the modelling method as described in Section 4 the components selected in Section 3 were modelled and a supervisor was synthesized. This chapter will cover the analysis of the findings during the modelling and the metric analysis. During these analyses both ASD and CIF will be taken into account. First the metrics are analysed to get quantifiable comparison results. After this a generic analysis is done for things which could not be captured within the metrics. The chapter will also explain about the restrictions of the synthesis algorithm and the pollution of the ASD models. Using the conclusions from all of the analysis parts, a final conclusion can be drawn with regards to both controller development methodologies.

### 6.1 Modelling Metric Analysis

A lot of quantifiable indicators were defined in Chapter 5. For all revisions and models that were selected in Chapter 3 these metrics were computed and evaluated for both the ASD and CIF specifications. As there are 217 component revisions analysed this has generated a huge amount of data, which is too much to attach to this thesis. Instead of attaching these, the analysis of this data is covered in this section. Some samples from the data is given to visualize the data. Appendix C shows the changes per revision and Appendix D shows the combined component metric values per revision. As the ASD and CIF languages are different, so are the metric values. Therefore first an analysis is done on the metrics to see what major absolute differences there are between both approaches and why. When this is done a closer look is taken at the relative differences between revisions to determine which approach takes less effort when implementing a change.

Due to its size, component H\_X became too big to synthesize a supervisor. Therefore only a small piece of the model development history was created in CIF. As discussed earlier there are a lot of revisions in the early stages of the available model history. Without the rest of the time line in CIF, this would give a skewed image of the difference over the whole period. Therefore component H\_X was not taken into account during the evaluation of the metrics. The other components were modelled and analysed, resulting in more automata than in ASD. This can be seen in Table 6.1.

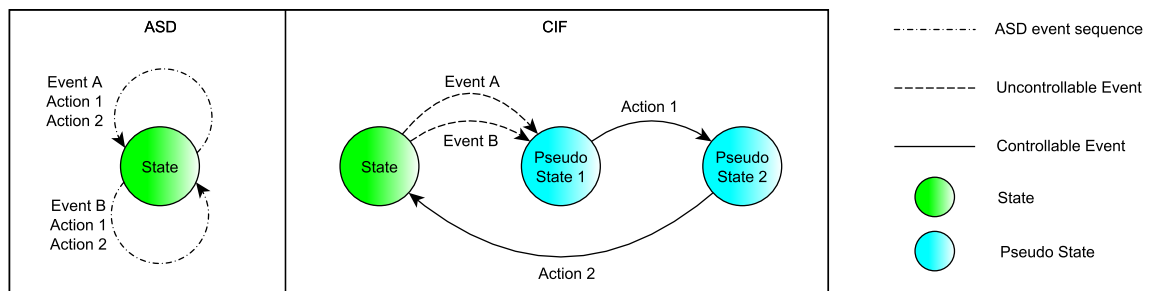
**Table 6.1:** *Amount of automata per component*

Component	ASD		CIF	
	DM	IM	Plants	Requirements
A	1	4	16	13
B	1	4	5	7
C	1	5	8	11
D	1	3	27	21
E	1	3	16	28
F	1	12	26	40
G	1	3	5	11
H	1	4	6	14

### 6.1.1 Absolute Difference Analysis

Due to the differences in languages some functionalities are modelled in a different way, which in turn results in different metric values. One of these is the data variables metric. As in CIF this is modelled in a separate plant, only there data exists. In ASD every instance of an event which transfers data has this data variable, which increases the total amount of data variable operators. Since data is simplified within CIF, this metric will not play a major role in the analysis.

Some metrics which do play a major role in the analysis are the amount of states, events and actions. In ASD a single trigger event can have multiple actions. This is an event sequence, which is modelled differently in CIF. CIF uses pseudo-states to model the same behaviour. These pseudo-states have a huge impact on several metrics. First of all the amount of states rises. Depending on the amount of different action sequences this can lead to many more states within a plant. While the amount of states increases, the amount of events or actions might decrease. Every trigger event with the same action sequence follows the same path. In CIF this means that the state changes of the plant are equal. However in each state the event is declared only once. In ASD this is different as each trigger declares the action sequence. The difference can be clearly seen in Figure 6.1. The amount of unique event and actions remains the same, however the absolute amount of actions is reduced by two in this example. In return the amount of states is increased by two.



**Figure 6.1:** Example of different modelling resulting in different metric results

Due to the lack of Blocked, Illegal and NoOp actions in CIF, these are all zero in contrast with the ASD metrics. However the amount of events and actions are translated to the amount of uncontrolled and controlled events in CIF for the metrics, it is not completely a one to one translation. This is due to the hierarchical implementation in CIF. Due to this the events in used IMs are the controlled events and the actions are uncontrolled. In the client IM the events and actions are both controlled. For the sum of both this does not have an influence, but this will result in fewer function points. This is due to the higher weight factor of uncontrolled events. However this difference is present, the metrics can still be used to compare relative changes between revisions, which is done in the next section.

The CC for ASD models is close to that of the CIF models. However the ACC is higher for interface models in CIF, since the amount of states is higher. However the complexity in the requirements is low, whereas the complexity in a DM is high. This is due to the modularity of the requirements which makes each individual requirement quite simple. A DM combines this which makes the complexity much higher. Due to this split nature each component is not

only less complex according to the CC and ACC, but also less difficult and takes less effort to comprehend according to the Halstead metrics.

Note that there are models added to CIF which are not in the ASD counterparts, like queue and data storage automata. This is necessary to get an equal functionality, but also influences the results. The other way around is also possible, since within ASD an extra interface model is sometimes used to create flow structures. One might argue that these models are not a critical part of the component and therefore should not be taken into account when evaluating the metrics. However both influence the results, so both are necessary to meet the requirements. Therefore these models are taken into account. This leads for component D to a higher volume and FP value as can be seen in Table 6.2. This component uses close to twenty queue automata and uses a combined requirement to ensure the right behaviour of notification events and their required actions. The combined requirement is a necessity, since keeping the requirements modular became too big to synthesize a supervisor.

This all results in a higher maintainability index for the CIF requirements as can be seen in Table 6.2. It must be taken into account that there are more requirements and there is only one DM. Summing up the effort of all requirements does not even come close to the effort which it would take to comprehend one DM, which makes CIF still better in this view. Even though the requirements better maintainable than the DM, the interfaces are not much improved when modelling these in CIF.

**Table 6.2:** Average values of key metrics for all components.

Comp	Revisions		States		LOC		Function Points		Volume		MI	
	ASD	CIF	ASD	CIF	ASD	CIF	ASD	CIF	ASD	CIF	ASD	CIF
A	68	29	11.3	52.0	785.0	268.0	4394.0	2480.2	2143.5	2553.0	50.8	69.0
B	59	14	7.0	47.6	1508.9	228.2	9073.1	2380.4	3866.2	2005.0	35.9	54.1
C	73	37	8.1	46.2	1029.2	330.8	5583.2	3164.6	2701.6	3023.3	43.6	68.6
D	18	6	5.0	96.5	304.0	228.0	1807.8	2331.5	865.8	2043.9	49.0	79.1
E	10	5	4.0	54.7	159.0	92.8	715.4	630.7	241.3	419.2	37.8	54.2
F	138	90	21.4	120.3	2280.7	593.2	9044.2	5569.8	6221.3	5220.1	42.3	69.8
G	12	7	4.0	26.8	316.0	107.0	2472.8	932.4	892.5	826.9	49.5	65.5
H	51	29	11.0	53.9	629.0	140.3	2145.7	1384.8	1496.0	1026.6	49.9	71.5
Avg	53.6	27.1	9.0	62.3	876.5	248.5	4680.5	2563.2	2303.5	2139.8	44.9	66.5

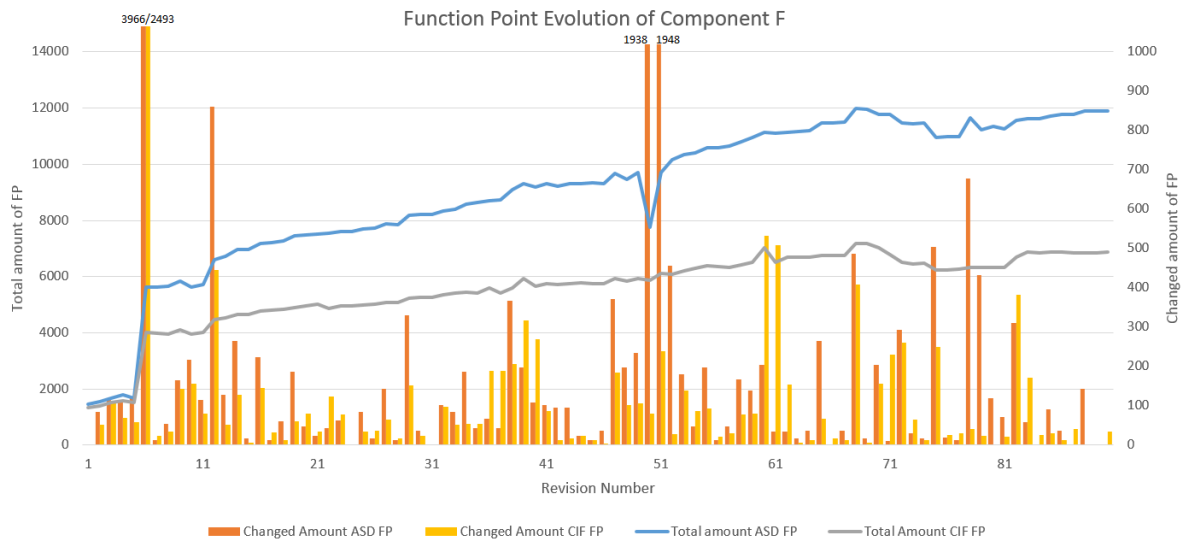
### 6.1.2 Relative Difference Analysis

This section will not focus on the absolute difference in metric values, but rather on the changes of metric values during the evolution of the component. In this way metrics can be used to quantify the change and a comparison between both methodologies with respect to change effort can be made. Due to the amount of data which is available, the graphs shown are exemplary for the whole dataset. As the entire dataset will be taken into account for analysis, there will be a table with key averages for all models.

When looking at complete components, the changes within CIF are smaller than in ASD with respect to the LOC. With on average 39.3 LOC changes per revision for ASD, the amount of LOC changes is 2.2 times higher than in CIF as can be seen in Table 6.3. This can be explained by the amount of actions which are used by more than one event. These should

be added in ASD, whereas in CIF these might already be declared. The Blocked and Illegal actions are not counted within the LOC, these should also be updated in ASD according to interface changes. However this can be done semi-automatically within the ASD tooling and therefore they are not taken into account in this metric analysis. As the actions and events make up the LOC, these follow the same trend.

The way the CIF models are built results in more states than the ASD equivalents, as described in the previous section. Due to the modelling of action sequences with help of pseudo states and the requirements the total amount of states is higher than in ASD. Not only the absolute difference is visible, but the difference in changes is even more vital. CIF has an amount of states which is 8 times higher on average as in ASD, but the amount of states added or removed per revision is 21 times higher than in ASD with 2.8 changes per revision. Due to the construction of CIF more states are needed to model things, for instance in the requirements. Changing these can even result in change of states within CIF, where there are no changes in states in ASD. As one metric is higher and the other lower in CIF, we need

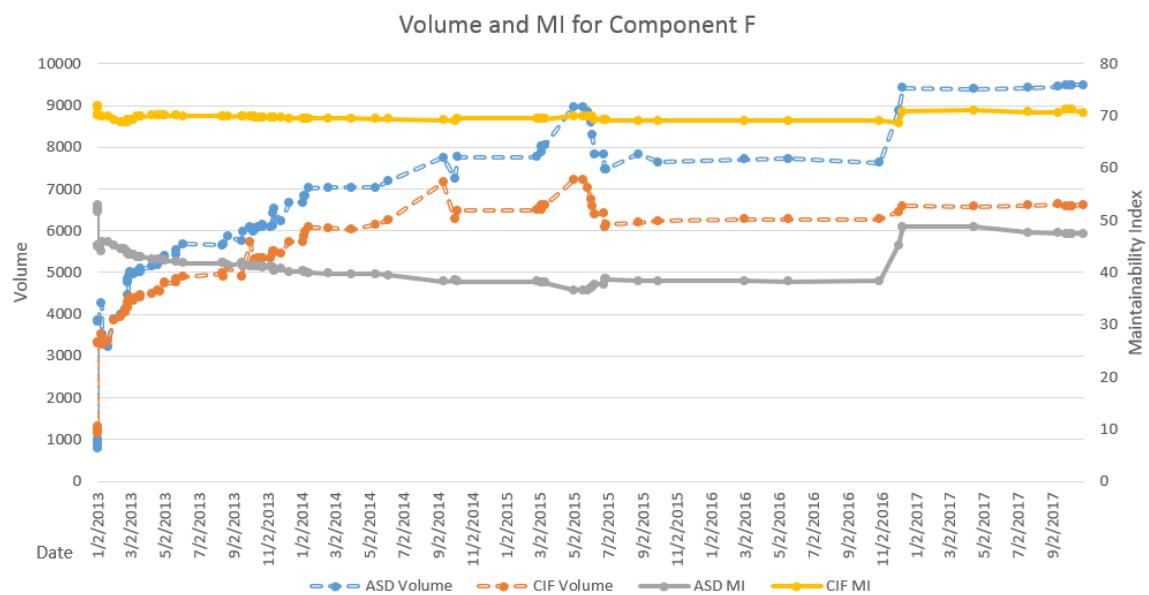


**Figure 6.2:** Evolution of the amount of function points for component *F*

a combination to make a well grounded comparison. To this end we introduced the FPs in Section 5. As we take the averages of FP changes, we can see that the amount of FP changes on average is higher in ASD than in CIF. The FP difference between revisions in ASD is on average 228.2, whereas in CIF this is 207.8 on average as can be seen in Table 6.3. Note that this difference relative to the average function points is different, as ASD has almost double the FP on average as the CIF counterparts. The difference increases with 70 FPs when a few outliers of component A are not taken into account. These outliers are due to pollution within the ASD components, which will be discussed within the next section. In CIF this pollution is partly removed to reduce the statespace, which results in big differences. These outliers are 4 revisions in one day, which makes it also very likely to be a mistake. When these are not taken into account the relative change is equal for both methodologies. For component F the evolution of the function points can be seen in Figure 6.2. Especially when there is a lot of change in FPs, the amount of FP changes at ASD is much more than in CIF. For smaller changes in FP the amount of change is bigger in ASD for most cases, but not for

all. At component G something else is the cause for the higher change in FP. This change is due to the addition of data transmission within the component. ASD does not need any extra automata to realise this, but CIF needs the extra automata as described in Section 4. With very few other changes this has a huge impact on the average FP change, giving it a higher value than the ASD counterpart.

Note that the time between each revision is not equal as Figure 6.2 would imply, but this is done for visualisation purposes. Figure 6.3 does show the revisions on the right timescale. Here it can be seen that revisions are clustered, which might imply that requirement changes are implemented in phases. This can be done for testing purposes of the code by extending the functionality piece by piece.



**Figure 6.3:** Evolution of the Volume and MI for component F

Figure 6.3 also shows that the maintainability index of CIF is higher than that of ASD. However the MI is calculated per automaton of DM/IM. The graph shows the average MI for the whole component. As CIF has many smaller components as can be seen in Table 6.1, this makes each component more easy to maintain, although there are more of them than in ASD. This holds for every component as can be seen in Table 6.2. It also explains why the MI rises even though the volume increases at the end of 2016. The volume depicts the size of the entire component. At the end of 2016 a new interface has been added to the component, which increases the volume. However this model has a higher MI than average which increases the MI for both ASD and CIF. Overall the volume of ASD is higher than that of CIF, but they both show the same tendency.

Table 6.3 shows the differences in change values between the two methodologies. Note that some metrics are not evaluated on change, for instance the MI. This is due to the nature of the metric. For the maintainability we are not interested in the change of it, since this is a value which does not imply change effort. It is a result of it and therefore we are only interested in the absolute value as depicted in Table 6.2.

**Table 6.3:** Average change of key metrics of all components

Component	States		LOC		Function Points		Volume	
	ASD	CIF	ASD	CIF	ASD	CIF	ASD	CIF
A	0.5	6.0	58.7	58.3	280.3	527.0	171.2	568.2
B	0.0	0.5	19.6	4.8	186.2	49.8	59.1	49.4
C	0.2	2.1	54.8	13.6	220.9	141.0	150.1	130.3
D	0.0	4.6	21.2	17.6	123.0	187.4	79.4	185.5
E	0.0	5.3	54.3	18.8	471.9	376.8	152.7	152.9
F	0.3	2.1	65.1	13.6	215.5	125.0	197.6	140.9
G	0.0	0.4	3.2	5.6	22.8	51.8	10.3	43.3
H	0.0	1.5	37.4	5.4	99.4	47.4	104.6	37.6
Avg	0.1	2.8	39.3	17.2	228.2	207.8	115.6	163.5



## 6.2 Modelling Analysis

In Chapter 3 an analysis on the revision count of the models was done. There it was visible that the amount of revisions for design models was significantly higher than the amount of changes in interface models. However this gives a false image of reality, since there are a few reasons a new revision is created for the ASD models:

1. An actual change has been made to the model:  
There has been made a change to the model, which results in a different revision.
2. A referenced model has changed:  
There has been made a change to a referenced model, this matters if a definition of an event or action is changed. Each verified model has a so called fingerprint in ASD, indicating that the model is verified. Even though a reference is to the same file, the fingerprint might change as the referenced model is verified again.
3. Verification data is different:  
The verification data is partly stored within the model, which is added if a verification is done. This leads to a new file, which in turn leads to a new revision.
4. Metadata is different:  
ASD stores the modelling tool version and language version within the file, if the model is saved again within another tool version this is updated, leading to an other file and in turn a new revision.
5. Old file is restored:  
There are actual changes in a new revision, but this is actually just a restoration of an older version.
6. No reason:  
There are occasions there are no changes at all to the model, however a new revision is made.

Since the analysis is conducted on design model revision lists, the severity of reasons on interface models is not determined. However reason 2 is the reason for 20% of the selected model revisions. This reason is not valid for interface models, since these references do not occur in interface models. In 27% of the revisions an actual change to the model has been made. These percentages are really dependent on the nature of the model. A simple component may have 0 revisions due to reason 2, as there are very few references, where a model with more referenced IMs can have as much as 45% of the revisions due to this reason. Since CIF does not have a verification or metadata, which is changed during modelling, and we do not create a revision when there is no reason, there are 53% less revisions on average in CIF than in ASD, considering all reasons for change, ranging between 35% and 75% per model.

These differences in reasons do have an impact on which things change within the CIF modelling. If in ASD an interface changes, this impacts the CIF modelling in an equal manner, that only the plant automaton changes. In ASD however this also impacts the design model. This can either be just in the file where the reference is changed, but it can also be a more

extensive change. As mentioned earlier, ASD needs to be complete, mentioning all possible events in each state even if they are blocked or illegal. Adding an event to an referenced model, requires that it must be added in the design model by being blocked or illegal there (in case there is no requirement concerning this event for this component). CIF does not have this completeness and therefore less change has to be made within the CIF specification. In Figure 6.4 a changelog of a component is given, where it can be seen that at version 11, 16, 17, 19 and 20 both the design model as a used plant changes. In the CIF equivalent of this component only the used plant changes.

ASD:	Version 1	Version 2	Version 4	Version 5	Version 6	Version 8	Version 11	Version 16	Version 17	Version 18	Version 19	Version 20	Version 21	Version 22	Version 23
Main.dm	Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed
Main.im	Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed
Used_Plant_1.im	Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed
Used_Plant_4.im	Added														
Used_Store_1.im					Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed
Main	Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed
Used Plant 1	Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed
Extra Plant 2	Added														
Used Plant 4	Added												Removed	Added	Removed
Extra Plant 4	Added	Changed		Changed											
Queue Automaton 1	Added	Removed													
Queue Automaton 2	Added														
Queue Automaton 3	Added														
Queue Automaton 4				Added											
Queue Automaton 5					Added										
Queue Automaton 6	Added														
Used Store 1						Added	Changed	Changed	Changed	Removed	Changed	Changed	Changed	Changed	Removed
Requirement 1															Added
Requirement 2					Added	Changed				Changed				Changed	Changed
Requirement 3	Added	Changed	Removed												
Requirement 4			Added	Removed											
Requirement 5				Added											
Requirement 6	Added														
Requirement 9	Added		Changed	Changed		Changed							Changed	Changed	Changed
Requirement 11	Added			Removed											
Requirement 12	Added	Changed	Changed	Removed											
Requirement 13						Added									Removed

Figure 6.4: A piece of a changelog of a component.

The figure also shows that the requirements are not changing one by one, which would be the optimal situation, as each revision has a single requirement change. However requirements are written down as modular as possible. Due to this a change in actual requirements might involve multiple formal requirements. An example is that a new event is added which should have 4 actions to be taken after that. This is then split into 4 different requirements, which might already exist and change. Due to this it is rarely the case that only one requirement changes within CIF. In addition to this requirements might get replaced with other requirements as can be seen clearly at version 4 of Figure 6.4.

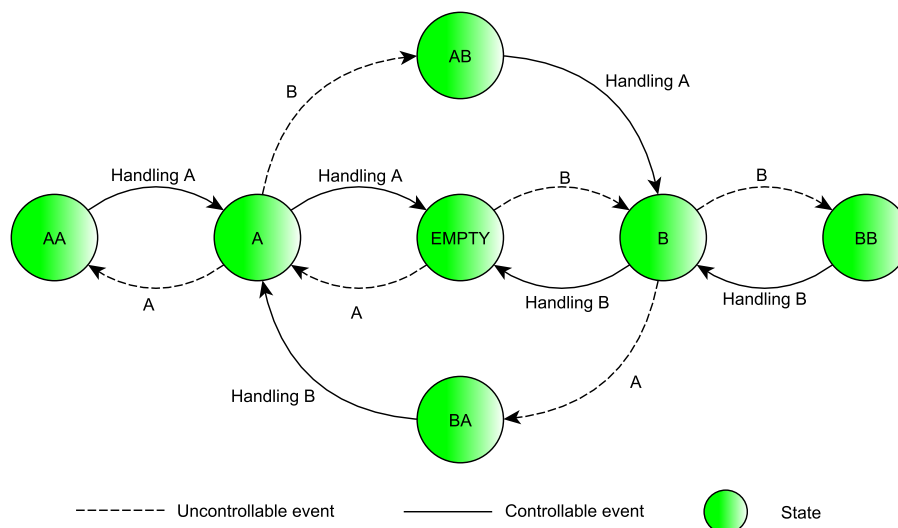
### 6.3 Modelling Restrictions

Besides differences between ASD and CIF with regard to changes and revisions, there are significant differences between the two with regards to modelling. There are three major problems during interface model conversion to CIF, which will be discussed in the coming paragraphs:

1. Notification Event Queues
2. Data transference
3. State Space Allowance

### 6.3.1 Notification Event Queues

As described in Section 4 to allow for notification events to be happening asynchronously a queue system must be introduced. This queue system should be able to hold several activities and remember in which order the activities should be executed. Such a queue is built in within the ASD hierarchical syntax and therefore not necessary to model. In CIF there is no hierarchy and no queuing system. Earlier research [13] has described how to model a functional FIFO queueing system as depicted in Figure 6.5, however this is only a feasible option for components with very few notification events, as every possible queuing order is manually described. This is due to the fact that each state represents another queue filling. The transitions represent the filling of the queue and the handling of the events. Even when this definition of the notification queue is limited to 1 instance of each notification event. This model can be upgraded to contain more than one, which would lead for two notification events to an statespace of 128 states for this queue automaton with the standard queue size of 7. With a queue size of 2 the statespace remains limited to 7 as can be seen in Figure 6.5. One of the selected components has 14 notification events. With the ASD standard set to a queue of 7, the queue automaton would have a state space of  $14^7$  states.



**Figure 6.5:** *An example of a queue with size two and two activities*

Using data an alternative queue model can be set up with less states. This was unfortunately only ready for implementation after the analysis was done and there was no time left to redo the analysis. In Listing 6.1 the model is depicted for a FIFO queue with capacity 7. For each notification event there are 8 events added to the queue automaton, assuming a maximum queue size of 7. Depending on the amount of events already in the queue, the right line of code is executed (e.g. queue is filled with 4 events and Notification 1 occurs, then line 12 will execute). This is done because the synthesis algorithm does not allow the use of lists, which would be used to place the event on the right place in the queue to ensure a FIFO handling of events. When the queue is full the notification event should still not be blocked, as it is uncontrollable. The notification handling is done by a single event introduced within the queue model, when the corresponding notification event is first in the queue. Afterwards the

actual amount of events in the queue is updated and all present events are shifted one place. This allows for synthesis of a supervisor, which uses modular requirements for notification events even for a large amount (19) of notification events. Therefore this is a suitable way to model the notification queue.

---

```

1 plant Queue:
2   controllable   Not_Forward_1,           //1
3                 Not_Forward_2;         //2
4   disc int[0..2] P1,P2,P3,P4,P5,P6,P7 = 0; //places to store events
5   disc int[0..7] Filled = 0;           //how many events in queue
6   marked P1=0,P2=0,P3=0,P4=0,P5=0,P6=0,P7=0,Filled=0;
7
8   location Empty: initial; marked;
9   edge Plant1.Not_Event_1 when Filled = 0 do P1:=1, Filled:=Filled+1;
10  edge Plant1.Not_Event_1 when Filled = 1 do P2:=1, Filled:=Filled+1;
11  edge Plant1.Not_Event_1 when Filled = 2 do P3:=1, Filled:=Filled+1;
12  edge Plant1.Not_Event_1 when Filled = 3 do P4:=1, Filled:=Filled+1;
13  edge Plant1.Not_Event_1 when Filled = 4 do P5:=1, Filled:=Filled+1;
14  edge Plant1.Not_Event_1 when Filled = 5 do P6:=1, Filled:=Filled+1;
15  edge Plant1.Not_Event_1 when Filled = 6 do P7:=1, Filled:=Filled+1;
16  edge Plant1.Not_Event_1 when Filled = 7;
17
18  edge Plant1.Not_Event_2 when Filled = 0 do P1:=2, Filled:=Filled+1;
19  edge Plant1.Not_Event_2 when Filled = 1 do P2:=2, Filled:=Filled+1;
20  edge Plant1.Not_Event_2 when Filled = 2 do P3:=2, Filled:=Filled+1;
21  edge Plant1.Not_Event_2 when Filled = 3 do P4:=2, Filled:=Filled+1;
22  edge Plant1.Not_Event_2 when Filled = 4 do P5:=2, Filled:=Filled+1;
23  edge Plant1.Not_Event_2 when Filled = 5 do P6:=2, Filled:=Filled+1;
24  edge Plant1.Not_Event_2 when Filled = 6 do P7:=2, Filled:=Filled+1;
25  edge Plant1.Not_Event_2 when Filled = 7;
26
27  edge Not_Forward_1 when P1 = 1 do Filled:= Filled-1,
28     P1:=P2, P2:=P3, P3:=P4, P4:=P5, P5:=P6, P6:=P7, P7:=0;
29  edge Not_Forward_2 when P1 = 2 do Filled:= Filled-1,
30     P1:=P2, P2:=P3, P3:=P4, P4:=P5, P5:=P6, P6:=P7, P7:=0;
31 end

```

---

**Code Listing 6.1:** Queue model with queue size 7 for two notification events.

Note that this new queue modelling will have an influence on the metrics, however the influence on the effort and time is non significant. As can be seen in Listing 6.1 the automaton is quite generic, which allows for automation of this process. In this way the introduction of this automaton will influence metrics like the FP, Volume and LOC, but not the effort which it takes to create and adapt the queue. Since the metrics are used to indicate the effort and time it takes to introduce changes and the effort of this queue is low due to automation, the introduction of this automaton should not influence the results significantly with respect to effort and time it takes to create and change the model.

### 6.3.2 Data Transference

In ASD events and actions can transfer data along. CIF edges do not have this possibility without using channels, which are not compatible with the synthesis algorithm. The edges are able to update a defined value within the plant. However an edge can not update a value which is not in the same plant as the edge, which makes it impossible to do this in the requirements. This is a problem, since the ASD syntax does not always define where data is going to or where it is coming from in the interface models. Channels are not suitable for two reasons, they need the same sender as receiver name and more importantly, supervisor synthesis does not support channels.

This is now solved as described in Section 4, which is a simplification. It does however allow for data to be read from an external source and stored within the component. Since the actual replacement of a component within LOPW with a CIF model is not within the scope of this research, this might be sufficient. However more research should be conducted to make grounded statements about this part of the models.

### 6.3.3 State Space

Another problem is generated by the difference in actuation of events within both languages. In ASD there is completeness, which tells if an event is either allowed or not for all events. This is possible since an ASD component is only active when a call is made to the component. Due to this the events are only blocked when the component is active, allowing other components to use this event. In CIF there is neither a completeness requirement nor selective activity of a component. If an activity is not allowed within the specification, it is never allowed since the CIF component is always active. However since CIF does not have a completeness requirement, the in ASD blocked events are not specified in the requirements at all. Therefore they are allowed, unless it prohibits the actual functioning of the component.

This is a difference which can be easily bridged, by making the CIF specification exactly as the ASD specification, by blocking all events in the used plants, when the component is active. However this is superfluous as the normal requirements are making sure the action sequences are fully enveloped before any other event can take place on an used plant. The real problem is in the fact that this will generate a huge statespace. Especially in components with many used interface models. Component B has only 68 states when controlled in the statespace when modelling only the used parts of the referenced interfaces. However if the interface models are completely modelled this grows with a factor 260 to 17.640 states. Note that this is a small component and the statespace will grow with more complex models, generating huge statespaces for the bigger components like component F and H. This makes the synthesis process slow and memory consuming, resulting in long waiting times or out of memory errors. In addition to this a lot of behaviour is modelled with no influence on the component, which makes it harder to identify the actual behaviour of the component in the statespace. This might invoke problems during verification of the component.

## 6.4 ASD Pollution

In Section 6.1.2 we already encountered some major differences between ASD and CIF in some cases. The reason for this has ground in the evolution of the ASD models. Some models were created in 2012 and are updated and changed during the years. During this evolution many actions, references and variables were added and removed. This had to be done manually. Since ASD does not have any negative effects of this pollution, one might forget to remove one of these attributes. These attributes are not used within the IM or DM anymore, but are still defined. This is what is meant with pollution.

Functionally the models are still correct and do not have any noticeable negative effects of this pollution, however the model definition becomes larger and more difficult to comprehend. For reply-events this is the case in frequently used IMs. Here a certain rulecase is removed, but not all reply event definition only used in this rulecase are removed. Variables can be found in both IMs and DMs, even if they are not used. The references are particularly interesting, as they are taken into account when calculating the Function Points. When a reference is made, but no single event is used from that interface, the events do not have to be declared according to the completeness requirement of ASD. This results in a reference, which adds to the components size, but is not actually part of the component.

In CIF the pollution of variables was not present, as most of the variables which cause the pollution pre-date the first available models for this research. The reference pollution was not remodelled in CIF, as there was no need to do this. The reply event pollution was however taken over from ASD. This was done to maintain the conformity between the IMs and the plants within CIF. In CIF however this pollution is easily removed as the unused events are indicated by the tooling. The ASD:Suite does not indicate the unused reply events, which makes it a more extensive task to do.

## 7. Conclusion and Recommendations

### 7.1 Conclusion

This research investigates the difference of two methodologies to make controllers for LOPW. The current methodology uses ASD to manually describe controllers, whereas CIF uses supervisor synthesis to generate controllers. To compare both methodologies a selection has been made of several components of LOPW. From these models requirements are extracted and then according to these CIF specifications were made. These were then compared to their ASD counterpart using metrics.

A general analysis shows that a lot of revisions are there without actual requirement changes for that component. Due to changes in interfaces these revisions came to be or due to meta-data changes as explained in Chapter 6.2. In both methodologies this should be adapted, however in ASD the DM is adapted in many cases as well mainly due to completeness semantics of ASD, whereas CIF does not need requirement changes. This is supported by the metrics, which conclude that the amount of changes within CIF is lower than the amount of changes within ASD. This means that the effort it takes to keep the models up to date with the requirements is lower when using the SCT methodology.

Unfortunately it is not that simple, since some constructions are difficult to implement in CIF. These constructions are features within ASD which are not or very inefficient to implement in CIF. These constructions are the implemented queues in ASD, which have to be added manually in CIF, and the data-transference between models, which can only be done with a workaround in CIF. In addition to this, large components with more than 2000 lines of code like component F or H.X can take a few hours to synthesize or run out of memory with the standard modelling techniques.

However taking into account the reduction in amount of changes needed in CIF by 40% counted in function points, we can conclude that CIF is a more efficient method to generate controllers for models of LOPW, which are not too big. In some cases the CIF methodology might not be preferred over ASD, because the implementation has influence on the efficiency of the modelling process, which is still subject for research. Also the SCT methodology might not work for large models, because of the big state space. To this end more research has to be done, which is described in the next section.

### 7.2 Recommendations

As described in Section 3 it is difficult to trace requirements from the documentation to the various components that are realizing these requirements. This is due to the fact that the requirements mentioned in the documentation are often not one to one or many to one related to a single component. Instead there are many requirements which are then implemented in many models. Therefore it was difficult to find out which requirements a component should meet. Therefore it is recommended to ASML to detail their LOPW EPDS even more and add timestamps of requirement implementation. When this is done the EPDS can be used to find requirements for each model. In addition to this requirements can be traced back to a certain revision. This will not only improve the CIF modelling, but will also be helpful for the current methodology using ASD. In addition to this I would recommend ASML to clean up

their ASD models, as some models are in the running for 6 years. This has resulted in a lot of pollution in the models, which makes them harder to read and understand. This clean-up would should consist of removing unused references, actions and variables.

There are still some bumps in the CIF methodology as described in Section 6, such as queues and data transference. Further research could clarify the implementation of the queues for notification events as this is still a major restriction with regards to functionality on the CIF implementation used in this research. As this is an ASD specific function, this might be solved without a queue within CIF, depending on the LOPW structure requirements. The implementation of the proposed solution for the correct queue model and its automation is highly recommended to research, to see the influence on the methodology.

Another difficulty is the data transference, which is now simplified in the CIF specifications used. Due to the inability of the synthesis algorithm to cope with channels, this is currently no option. However even with channels they can only send data over to an edge with the same name. Therefore the CIF3 language and synthesis algorithm should be adjusted to make this a viable solution. Depending on the component and LOPW it might not even be needed to send data trough the CIF specifications. This makes it even more interesting for a follow up research.

The interface models were now taken as plants, which leads to a lot of events which are not needed in the actual component and are blocked within ASD. Research should be done to determine if these events are needed to model and if so, if they should be disabled. In both cases this is related to the collaboration of components within LOPW and the implementation of CIF within the hierarchy. As there are many differences between ASD and CIF in the implementation, this might cause crucial differences within the way of modelling.

During this research the tool EMF (Meta) Model Analysis(EMMA) [16] was used to extract metrics from the ASD models. To this end the supplied gitminer is used which mines all revisions of all files of LOPW. Unfortunately ASD components refer to used interface models within the same folder. Since the miner stores all versions of a certain model in its own folder, these references are not working any longer. To add to this the name of the model is changed into the date of the revision, which might not be the same as the referenced models as they are older. Therefore EMMA is not able to resolve the events and actions in the DMs which take place on the interface models. To solve this a function should be added to update the references to the right folder and then finding the last version of the referenced model with regards to the timestamp of the DM. As this is very specific to ASD models this might be inefficient for a global tool, but it might be very convenient for ASML. When these references are resolved, much more detailed information about events can be extracted with EMMA.



---

## References

- [1] ASML. ASML Company History. <https://www.asml.com/company/history/en/s277?rid=51985>, 2017. Online; accessed 2017-11-08.
- [2] Beek van D.A. Fokkink W.J. Hendriks D., Hofkamp A., Markovski J. van de Mortel-Fronczak J.M., Reniers M.A. Cif 3: Model-based engineering of supervisory controllers. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 575–580, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
- [3] Blitterswijk M. EPDS LOPW. Confidential D000486800-00, ASML , May 2017.
- [4] Broadfoot G.H., Hopcroft P.J. *Analytical Software Design.* -, 2003.
- [5] Comer E.R., Bersoff E.H., Davis A.M. A Strategy for Comparing Alternative Software Development Life Cycle Models. In *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, volume 14, 1988.
- [6] Eindhoven University of Technology. MCRL2 Homepage. [http://www.mcrl2.org/web/user\\_manual/index.html](http://www.mcrl2.org/web/user_manual/index.html), 2017. Online; accessed 08-11-2017.
- [7] Forschelen S.T.J., van de Mortel-Fronczak J.M., Su R., Rooda, J.E. Application of supervisory control theory to theme park vehicles. *Discrete Event Dynamic Systems*, 22(4):511–540, Dec 2012.
- [8] Groote J.F., Marincic J., Osaiweran A. Assessing the Quality of Tabular State Machines through Metrics. In *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 426–433, July 2017.
- [9] Halstead, M.H. *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., New York, NY, USA, 1977.
- [10] IEEE Standards Board. IEEE Standard for Software Productivity Metrics. *IEEE Std 1045-1992*, 1993.
- [11] Kan, S.H. *Metrics and Models in Software Quality Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.
- [12] Kock de E. , Fokkink W.J. Control and performance analysis of wafer flow in wafer handling systems. Master’s thesis, Eindhoven University of Technology, 2014.
- [13] Loose R. Component-wise Supervisory Controller Synthesis using existing plant models in a client/server structure. Master’s thesis, Eindhoven University of Technology, 2017.
- [14] Loose R., van der Sanden B., and Reniers M.A., Schiffelers R.R.H. Component-wise supervisory controller synthesis in a client/server architecture. In *Workshop on Discrete Event Systems (WODES)*, 2018.
- [15] Chemuturi M. *Requirements Engineering and Management for Software Development Projects*. SpringerLink : Bücher. Springer New York, 2012.

- 
- [16] Mengerink J.G.M., Serebrenik A., Schiffelers R.R.H., van den Brand M.G.J. Automated analyses of model-driven artifacts: Obtaining insights into industrial application of mde. In *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement, IWSM Mensura '17*, pages 116–121, New York, NY, USA, 2017. ACM.
- [17] Ouedraogo L., Kumar R., Malik R., Akesson K. Nonblocking and safe control of discrete-event systems modeled as extended finite automata. *IEEE Transactions on Automation Science and Engineering*, 8(3):560–569, July 2011.
- [18] Poore J.H. Prowell S.J. Foundations of sequence-based software specification. *IEEE Trans. Softw. Eng.*, 29(5):417–429, May 2003.
- [19] Ramadge P.J., Wonham W.M. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.
- [20] Romero Sahagun E. Specification of Timed Service Contracts in Component Based Architectures. Master's thesis, Eindhoven University of Technology, 2015.
- [21] Sharma R., Biswas K. Functional Requirements Categorization - Grounded Theory Approach. *ENASE 2015 - Proceedings of the 10th International Conference on Evaluation of Novel Approaches to Software Engineering*, pages 301–307, 01 2015.
- [22] Starke M.J. Supervisory control using extended finite automata for ASML wafer scanners. Master's thesis, Eindhoven University of Technology, 2013.
- [23] Swartjes L, van Beek D.A., W.J. Fokkink van Eekelen J.A.W.M. Model-based design of supervisory controllers for baggage handling systems. *Simulation Modelling Practice and Theory*, 78:28–50, 11 2017.
- [24] Verum. Verum Knowledge Base. <http://community.verum.com/knowledgebase.aspx>, 2017. Online; accessed 2018-05-08.
- [25] Verum. Verum Software Tools B.V. <https://www.verum.com/company/>, 2017. Online; accessed 2017-11-07.
- [26] Wagner C., Harned N. Lithography gets extreme. *Nature Photonics*, 4:24, January 2010.

## A. ASD Metric Extraction Java Scripts

Packages are not included in scripts for reading purposes. Instead they are shown in the scripting below. Note that not all packages are used in all scripts.

```
1  import java.io.File;
   import java.io.FileReader;
   import java.io.IOException;
   import java.io.LineNumberReader;
5  import java.nio.file.Files;
   import java.nio.file.Path;
   import java.nio.file.Paths;
   import java.util.ArrayList;
   import java.util.List;
10 import org.eclipse.emf.common.util.URI;
   import org.eclipse.emf.common.util.TreeIterator;
   import org.eclipse.emf.ecore.EPackage;
   import org.eclipse.emf.ecore.resource.Resource;
   import org.eclipse.emf.ecore.EObject;
15 import com.asml.generics.qvto.java.QvtoTransformationEnvironment;
   import com.asml.generics.qvto.util.QvtoTransformationException;
   import Analysis.AnalyzerImpls.MetricsOverTime.ObjectMetrics;
   import Analysis.AnalyzerImpls.MetricsOverTime.ResourceMetrics;
   import Analysis.AnalyzerImpls.MetricsPuller.core.MetricsPuller;
20 import JoshUtils.EMF.CustomResourceSet;
   import JoshUtils.Debugging.SysOutDebugger;
   import JoshUtils.General.Configuration;
   import Persistors.Database.DatabasePersistor;
   import Simple.ExampleMetric;
25 import Utils.IO.DataRepository;
   import Utils.IO.HistoricalFile;
   import Utils.IO.Revision;
   import nl.altran.verum.asd.ecore.asd.*;
   import nl.altran.verum.asd.ecore.AsdStandaloneSetup;
30 import nl.altran.verum.asd.transform.AsdXsdToAsdEcoreTransformation;
   import nl.altran.verum.asd.xsd.ASD9.ASD9Package;
   import nl.altran.verum.asd.xsd.ASD9.ASD9StandaloneSetup;
   import Analysis.AnalyzerImpls.MetricsPuller.core.Metric;
   import Analysis.AnalyzerImpls.MetricsPuller.core.MetricsPackage;
35 import RepositoryMiners.GitMiner.GitMiner;
```

## A.1 GitMiner to obtain all model revisions

This Java script uses the gitminer to obtain all file revisions. It places each revision of a individual model in a folder for that designated model.

```
1  /** Script to mine LOPW from the repository */
   public class Miner {
       public static void mine(String url, String name) {

5      File dir = new File("./tmpDir/" + name);

       // use the GitMiner to mine git repositories
       GitMiner gm = new GitMiner();
       gm.setDatasetName("LOPW");

10      // open the repo for analysis
       gm.openRepo("C:\\Users\\yschriek\\Documents\\LOPW_GIT");

       // Set data types to mine
15      gm.addFileTypeToFind(".pdf"); // for EPDS
       gm.addFileTypeToFind(".im"); // for IM models
       gm.addFileTypeToFind(".dm"); // for DM models

       // actually get them
20      gm.findRevisionsOfFileTypes();
   }
}
```

## A.2 Size Measurement

This Java script counts the size of LOPW for each occurring revision number. When a certain revision number for a model does not exist, was present before and was not deleted, it is still counted for the size.

```

1  /** * This script will run for LOPW and determine its size */
    public class Sizemeasurement {

        public static void main(String[] args) throws IOException {
2      // Load settings and open files
        JoshUtils.General.Configuration.init("C:\\Users\\yschriek\\workspace\\EMMAv3\\emma.ini");
        FileWriter fileWriter = new FileWriter("Size_Individual.csv");
        PrintWriter printWriter = new PrintWriter(fileWriter);
        // Create list of all occurring timestamps in 'integers'
10     Set<Integer> integers = new TreeSet<>();
        for (HistoricalFile hf : DataRepository.instance.getDataset("LOPW_Upgraded -
        Copy").getHistoricalFiles()) {
            for (Revision rev : hf.getRevisions()) {
                if (!integers.contains(rev.getTimestamp())) {
15                 integers.add(rev.getTimestamp());
                }
            }
        }
        // Concert set of timestamps into vector
        int[] list = new int[integers.size()];
20     int i = 0;
        for (Integer integer : integers) {
            list[i] = integer;
            i++;
        }
25     Arrays.sort(list);
        //Loop to compute size at each timestamp
        for (int timest : list){
            int count =0 ;
            //check for all files
30         for (HistoricalFile hf : DataRepository.instance.getDataset("LOPW_Upgraded -
        Copy").getHistoricalFiles()) {
            Revision revisionAt=null;
            Revision prev = null;
            //check all revisions of this file
            for (Revision f : hf.getRevisions()) {
35                 int file_rev = f.getTimestamp();
                // if file revision is lower than current time store that revision, unless it is
                deleted
                if (file_rev <= timest){
                    prev = f;
                    revisionAt = f;
40                 //comment out if interested in cumulative unique models
                    if (f.getName().contains("deleted")){
                        revisionAt = null;
                    };
                }
            }
45         // if file revision is higher than current time, store latest value, unless it is
            deleted
            else if (file_rev > timest) {
                revisionAt = prev;
            }
        }
    }

```

```
50     //comment out if interested in cumulative unique models
    if (f.getName().contains("deleted")){
        revisionAt = null;
    };
    break;
}
}
55 // If there was a revision of this file of the type DM, add to count
// Comment out if only interested in IMs
if (hf.getExtension().equals(".dm")) {
    if (revisionAt != null) {
        count++;
60    }
}
// If there was a revision of this file of the type IM, add to count
// Comment out if only interested in DMs
65 else if(hf.getExtension().equals(".im")) {
    //if there is one
    if (revisionAt != null) {
        count++;
    }
}
70 }
    long epoch= timest;
    String date = new java.text.SimpleDateFormat("MM/dd/yyyy HH:mm:ss").format(new
java.util.Date (epoch*1000));
    System.out.println(count + " " + timest + " " + date );
    printWriter.println(count +","+timest+","+date);
75 }
    printWriter.close();
    fileWriter.close();
}
}
```

### A.3 Metrics Extraction

This Java script extracts metrics from all the model revisions within a certain folder. The values are stored within a SQL database.

```

1  /** This EMMA script will run for LOPW and evaluate metrics*/
public class AdvancedExample {
    public static void main(String[] args) throws IOException {
        JoshUtils.General.Configuration.init("C:\\Users\\yschriek\\workspace\\EMMAv3\\emma.ini");
5
        // Setup for the ASD technology bridge
        ASD9StandaloneSetup.doSetup();
        AsdStandaloneSetup.doSetup();
        EPackage.Registry.INSTANCE.put(null, ASD9Package.eINSTANCE);
10
        //define the .dm/im to .asd transformation
        AsdXsdToAsdEcoreTransformation transformation = new AsdXsdToAsdEcoreTransformation();
        QvtoTransformationEnvironment env = new
        QvtoTransformationEnvironment(transformation.getDefaultUriTransformer());
15
        int hfcounter = 0;
        //start loop over all different files
        for (HistoricalFile hf :
            DataRepository.instance.getDataset("LOPW_Upgraded").getHistoricalFiles() {
                DatabasePersistor persistor = new DatabasePersistor();
20                int dsid = persistor.getDatasetId("LOPW_Upgraded");
                String file_id = hf.getFullName();//
                // create resourceset, to enable us to load files into resources
                CustomResourceSet crs = new CustomResourceSet();
                //use hfcounter to determine which historical files should be evaluated:
25                //for LOPW around 900 different models: 0 - 225 - 450 - 675 - 900 or
                0-150-300-450-600-750-900
                if (hfcounter>=0 && hfcounter<1000){
                    int fid = persistor.getFileId(file_id, dsid);
                    //for each revision:
                    for (Revision r : hf.getRevisions()) {
30                        String versionname = fid + r.getName();
                        //if the file does not end on .asd , these are all DMs and IMs:
                        if (r.getName().endsWith(".asd")==false){
                            int linecount = countLines(r.getAbsolutePath());
                            if (linecount >100) {
35                                URI rv_uri;
                                //if there is a .deleted file, add .dm or .im to allow the tech bridge to cope
                                with this file extension
                                //then create an URI
                                if( r.getName().endsWith(".deleted")){
                                    Path source = Paths.get(r.getAbsolutePath());
                                    Files.move(source,
40                                source.resolveSibling(r.getName().concat(hf.getExtension())));
                                    rv_uri = URI.createFileURI(r.getAbsolutePath().concat(hf.getExtension()));
                                }
                                else{
                                    rv_uri = URI.createFileURI(r.getAbsolutePath());
45                                }
                                //Transform model into meta-model
                                try {

```

```

        transformation.transformModel(rv_uri, env);
    }
50    catch (QvtoTransformationException e) {
        e.printStackTrace();
    }
    //set a new uri to the converted uri
55    URI rv_out_uri = env.getUriTransformer().transformUri(rv_uri);
    Resource rv_resource = crs.getResource(rv_out_uri,true);

    //Disabled the proxy resolver to save time, enable for more details
    //EcoreUtil.resolveAll(crs);

60    //set which metrics should be used:
    MetricsPuller puller = new MetricsPuller();
    puller.register(new ExampleMetric());

    //calculate them
65    ResourceMetrics rm_metrics = puller.execute(rv_resource);

    //save them:
    for (ObjectMetrics metrics : rm_metrics) {
        for (String key : metrics.keySet()) {
70            if (!key.equals("object_id")) {
                Object value = metrics.get(key);
                //if the name contains deleted it means the file is deleted and all
                metrics should equal 0,
                //else the right value is assigned
                if(!r.getName().contains("deleted")){
75                    persistor.storeMetrics(
                        (int)metrics.get("object_id"),
                        r.getTimestamp(),
                        fid,
                        key,
                        value.toString()
80                        );
                }
                else{
                    persistor.storeMetrics(
85                        (int)metrics.get("object_id"),
                        r.getTimestamp(),
                        fid,
                        key,
                        "0"
90                        );
                }
            }
        }
    }
95    }
    }
    }
    }
    }
    hfcounter++;
100 }
    //if not within analysing range, just continue until in range or the end
    else{hfcounter++;}
    double progpercent = (hfcounter + 1) / 884;
    System.out.println("Filenumber = " + hfcounter + " Progress = %.2f" + progpercent );

```



```
    }  
105 }  
  
    //simple function to determine size of a model in amount of XML lines  
    //if these are too few, the model is old and the conversion will give an error  
110 public static int countLines(String filename) throws IOException {  
    LineNumberReader reader = new LineNumberReader(new FileReader(filename));  
    int cnt = 0;  
    String lineRead = "";  
    while ((lineRead = reader.readLine()) != null) {}  
    cnt = reader.getLineNumber();  
115 reader.close();  
    return cnt;  
    }  
}
```

## A.4 Metrics Definition

This Java script defines the metrics which are extracted from the models by Script A.3.

```

1 public class ExampleMetric extends MetricsPackage {
  @Metric(name="NumStatesPerFile") // Count all instances of state per model
  public int numStatesPerFile(Model asdroot) {
    int counter = 0;
5   TreeIterator<EObject> allObjects = asdroot.eAllContents();
    while (allObjects.hasNext()) {
      EObject obj = allObjects.next();
      if (obj.eClass().getName().toString().equals("State")) {
10        counter++;
      }
    }
    return counter;
  }

15  @Metric(name="NumActionsPerFile") // Count all actions per model
  public int numActionsPerFile(Model asdroot) {
    int counter = 0;
    TreeIterator<EObject> allObjects = asdroot.eAllContents();
    while (allObjects.hasNext()) {
20      EObject obj = allObjects.next();
      if (obj instanceof Action) {
        counter++;
      }
    }
25    return counter;
  }

  @Metric(name="NumInterfacePerFile") // count all referenced interfaces per model
  public int numInterfacePerFile(Model asdroot) {
30    int counter = 0;
    TreeIterator<EObject> allObjects = asdroot.eAllContents();
    while (allObjects.hasNext()) {
      EObject obj = allObjects.next();
      if (obj instanceof Interface) {
35        counter++;
      }
    }
    return counter;
  }

40  @Metric(name="NumIllegalPerFile") // count all illegal actions per model
  public int numIllegalPerFile(Model asdroot) {
    int counter = 0;
    TreeIterator<EObject> allObjects = asdroot.eAllContents();
45    while (allObjects.hasNext()) {
      EObject obj = allObjects.next();
      if (obj instanceof IllegalAction) {
        counter++;
      }
50    }
    return counter;
  }
}

```

```
55 @Metric(name="NumBlockedPerFile") // count all blocked actions per model
public int numBlockedPerFile(Model asdroot) {
    int counter = 0;
    TreeIterator<EObject> allObjects = asdroot.eAllContents();
    while (allObjects.hasNext()) {
        EObject obj = allObjects.next();
60     if (obj instanceof BlockedAction) {
            counter++;
        }
    }
    return counter;
65 }

@Metric(name="NumNoOpPerFile") // count all noop actions per model
public int numNoOpPerFile(Model asdroot) {
    int counter = 0;
70     TreeIterator<EObject> allObjects = asdroot.eAllContents();
    while (allObjects.hasNext()) {
        EObject obj = allObjects.next();
        if (obj instanceof NoOpAction) {
75             counter++;
        }
    }
    return counter;
}

80 @Metric(name="NumDisabledActionPerFile") // count all disabled events per model
public int numDisabledActionPerFile(Model asdroot) {
    int counter = 0;
    TreeIterator<EObject> allObjects = asdroot.eAllContents();
    while (allObjects.hasNext()) {
85     EObject obj = allObjects.next();
        if (obj instanceof DisabledAction) {
            counter++;
        }
    }
90     return counter;
}

@Metric(name="NumEventReferencesPerFile") // count all events per model
public int numEventReferencesPerFile(Model asdroot) {
95     int counter = 0;
    TreeIterator<EObject> allObjects = asdroot.eAllContents();
    while (allObjects.hasNext()) {
        EObject obj = allObjects.next();
        if (obj instanceof EventReference) {
100             counter++;
        }
    }
    return counter;
}

105 @Metric(name="NumGuardsPerFile") // count all guards per model
public int numGuardsPerFile(Model asdroot) {
    int counter = 0;
    TreeIterator<EObject> allObjects = asdroot.eAllContents();
110     while (allObjects.hasNext()) {
```

```

    EObject obj = allObjects.next();
    if (obj instanceof Guard) {
        counter++;
    }
115 }
    return counter;
}

@Metric(name="NumRuleCasesPerFile") // count all rulecases per model
120 public int numRuleCasesPerFile(Model asdroot) {
    int counter = 0;
    TreeIterator<EObject> allObjects = asdroot.eAllContents();
    while (allObjects.hasNext()) {
        EObject obj = allObjects.next();
125     if (obj instanceof RuleCase ) {
            counter++;
        }
    }
    return counter;
130 }

@Metric(name="NumStateVariablesPerFile") // count state variables per model
public int numStateVariablesPerFile(Model asdroot) {
    int counter = 0;
135     TreeIterator<EObject> allObjects = asdroot.eAllContents();
    while (allObjects.hasNext()) {
        EObject obj = allObjects.next();
        if (obj instanceof StateVariable) {
140             counter++;
        }
    }
    return counter;
}

145 @Metric(name="NumDataVariablesPerFile") // count all data variables per model
public int numDataVariablesPerFile(Model asdroot) {
    int counter = 0;
    TreeIterator<EObject> allObjects = asdroot.eAllContents();
    while (allObjects.hasNext()) {
150         EObject obj = allObjects.next();
        if (obj instanceof DataVariable) {
            counter++;
        }
    }
155     return counter;
}

@Metric(name="NumChangedReferenceInterfacesPerFile")
// count how much referenced interfaces are changed w.r.t. the previous revision
160 public int NumChangedReferenceInterfacesPerFile(Model asdroot) {
    int counter = 0;
    TreeIterator<EObject> allObjects = asdroot.eAllContents();
    while (allObjects.hasNext()) {
        EObject obj = allObjects.next();
165         if (obj instanceof UsedServiceReference) {
            String version = asdroot.eResource().getURI().segment(7);
            version = version.replaceAll("\\D+", "");
        }
    }
}

```

```
    for (HistoricalFile hf :
DataRepository.instance.getDataset("LOPW").getHistoricalFiles()) {
    if((hf.getName()+hf.getExtension()).equals((UsedServiceReference
obj).getName().toString()+".im") ){
170     for (Revision r : hf.getRevisions()) {
        if(r.getName().equals(version+".im.asd")){
            counter++;
            break;
        }
175     }
        break;
    }
}
180 }
return counter;
}

@Metric(name="ThisInterfaceChangedperFile") // determine if the client interface has
changed
185 public int ThisInterfaceChangedperFilePerFile(Model asdroot) {
    int counter = 0;
    String version = asdroot.eResource().getURI().segment(7);
    version = version.replaceAll("\\D+", "");
    for (HistoricalFile hf : DataRepository.instance.getDataset("LOPW").getHistoricalFiles())
    {
190     if((hf.getName()+hf.getExtension()).equals(asdroot.getName().toString()+".im") ){
        for (Revision r : hf.getRevisions()) {
            if(r.getName().equals(version+".im.asd")){
                counter++;
                break;
195             }
        }
        break;
    }
}
200 return counter;
}
}
```

## B. CIF Metric Extraction Python Scripts

### B.1 Alphabet Generating Script

This Python function creates a vector which contains all controllable or uncontrollable events present within a CIF specification, depending on the input being ‘Controllable’ or ‘Uncontrollable’.

```

1  # Yorick van der Schriek 1-5-2018 - ASML - TU/e - CIF models of LOPW

import math

5  # generate the Event alphabet of entire cif specification
def generatealphabet(path, Declaration):
    f = open(path)
    alphabet = []
    Plant = f.readline()
10  #for all lines
    while Plant:
        # read Plant name
        if Plant.count("plant") == 1:
            NameTmp = Plant
15            NameTmp = Plant.split("plant ")
            NameTmp = NameTmp[1].split(":")
            Name = NameTmp[0]
        # if Controllable or Uncontrollables are starting to be defined
        if Plant.count(Declaration)==1:
20            Ccount=0
            #count all within this declaration
            while Ccount==0:
                #save first event
                if Plant.count(Declaration)==1:
25                    Names = Plant.split(Declaration)
                    Names = Names[1].split(",")
                    #if last event strip of non-name attributes
                    if Plant.count(";")==1:
30                        Tmp = Names[len(Names)-1].split(";")
                        Names[len(Names)-1] = Tmp[0]
                    #else remove the comma
                    else:
                        Names = Names[:-1]
35
                #save other events
                else:
                    Names = Plant.split(",")
                    if Plant.count(";")==1:
40                        Names[len(Names)-1] = Names[len(Names)-1].split(";")[0]
                    else:
                        Names = Names[:-1]
                Z=0
                # add the new declared events in this line to alphabet when they are
                # not commented
45
                while Z < len(Names):
                    if not Names[Z].strip().startswith("//"):
                        Names[Z] = Name + "." + Names[Z].strip()
                        alphabet.append(Names[Z])

```

```
    Z=Z+1
    # ; declares end of the declaration so end loop if so
50 Ccount = Plant.count(";")
    # else goto next line
    if Plant.count(";") ==0:
        Plant = f.readline()

55 #if declaration is finished read lines untill new event declarations are
    found

    Plant =f.readline()
    f.close()
    return alphabet
```

## B.2 Automata Counter per CIF specification

This Python function counts the amount of automata within a CIF specification.

```
1 # Yorick van der Schriek 1-5-2018 - ASML - TU/e - CIF models of LOPW
import math
5 def determine_automata(path):
    f = open(path)
    Requirements = 0
    Plants = 0
    #for each line
10 for line in f:
    Line = line.strip()
    #if not commented
    if not Line.startswith("//"):.
        #add if new requirement or plant is declared
15     Requirements = Requirements + line.count("requirement ")
        Plants = Plants + line.count("plant ")
    f.close
    return (Requirements+Plants)
```



### B.3 Automata Name Collector

This Python function collects all names of the automata within a CIF specification.

```

1  # Yorick van der Schriek 1-5-2018 - ASML - TU/e - CIF models of LOPW

import math

5  def determine_plants_requirements(path):
    nr_automata = determine_automata(path)
    f = open(path)
    loopcount = 0
    invariant = 1
10  Namelist = []
    # for all automata
    while loopcount < nr_automata:
        count = 0
        Name = "none"

15     #for all lines in this automata
        while count == 0:
            Plant= f.readline()

20     # if line commented delete it from analysis
            if Plant.strip().startswith("//"):
                Plant = " "

        # part of line commented, remove commented part
25     if Plant.count("//")>=1:
            Temp = Plant.split("//")
            x=Plant.count("//")
            Temp=Temp[0:len(Temp)-x]
            Plant= ''.join(Temp)

30     # if plant, save name
            if Plant.count("plant ") == 1:
                NameTmp = Plant
                NameTmp = Plant.split("plant ")
35                NameTmp = NameTmp[1].split(":")
                Name = NameTmp[0]

        # if requirement, save name
40     if Plant.count("requirement ") == 1:
            NameTmp = Plant
            NameTmp = Plant.split("requirement ")

        # if requirement invariant:
45     if NameTmp[1].count("{")==1:
            NameTmp = NameTmp[1].split("{")
            NameTmp = "Invariant " + str(invariant)
            invariant = invariant + 1
            Name = "Requirement " + NameTmp

50     # if normal requirement:
            elif NameTmp[1].count(":")==1:
                NameTmp = NameTmp[1].split(":")
                Name = "Requirement " + NameTmp[0]

```

```
55     # if line is requirement invariant with 1 event
    elif NameTmp[1].count("needs")==1:
        NameTmp = NameTmp[1].split("needs")
        NameTmp = "Invariant " + str(invariant)
        invariant = invariant +1
60     Name = "Requirement " + NameTmp

    # add every new automaton in list
    if not Name == "none" and not Name in Namelist:
        Namelist.append(Name)
65

    # if end of automaton goto next one
    count = Plant.count("end") + Plant.count("needs")
    loopcount = loopcount +1

70 f.close()
    return (Namelist)
```

## B.4 Changes in CIF in automata

This Python function has two inputs, 'old' and 'new' both CIF specifications. It calculates for each automaton in both specifications if it is introduced, changed or deleted in the 'new' CIF specification. Afterwards three lists with names of introduced, changed and deleted automata are returned.

```

1  # Yorick van der Schriek 1-5-2018 - ASML - TU/e - CIF models of LOPW

import math

5  def determine_changed_CIF(old,new):
    GenMatrix = [] # Creating an empty matrices
    PandQ_old = []
    PandQ_new = []
    All_introduced = []
10  All_removed = []
    All_changed = []

    #if first component, no old exists
    if not old == "empty":
15  PandQ_old = determine_plants_requirements(old)
    PandQ_new = determine_plants_requirements(new)

    #for all 'new' components
    for PQ_new in PandQ_new:
20
        # edit names where requirement is written with capital letter for
        # unification

        if PQ_new.count("Requirement")>0:
            PQ_cif = PQ_new.replace("Requirement", "requirement")
        else:
25  PQ_cif = PQ_new

        # if the 'new' component does not exist in the old one add to introduced
        if not PQ_new in PandQ_old:
            #print "introduced: " + PQ_new
30  All_introduced.append(PQ_new)

        # if it does exist:
        else:
            FullNewAutomaton=[]
            f = open(new)
            searchcount=0
            #search automata
            while searchcount==0:
35  Plant=f.readline()
                Plant = Plant.strip()

                #remove requirement form the name
                if Plant.count("Requirement")>0:
40  Plant=Plant.split("Requirement")[1].strip()

            # when automaton found end loop
            if ((Plant.count(PQ_cif)==1 and Plant.count("plant")+Plant.count("
            requirement")==1) or ((Plant.
            count("{")==1 or Plant.count("

```

```

                                                    needs")==1) and PQ_new.count("
                                                    Invariant")==1 ) )and not Plant
                                                    .startswith("//"):

searchcount=1

50 # from here store the lines untill the automon ends
if ((Plant.count(PQ_cif)==1 and Plant.count("plant")+Plant.count("
                                                    requirement")==1) or ((Plant.
                                                    count("{")==1 or Plant.count("
                                                    needs")==1) and PQ_new.count("
                                                    Invariant")==1 ) )and not Plant.
                                                    startswith("//"):

FullNewAutomaton.append(Plant)
endcount=0
while endcount==0:
55     Plant = Plant.strip()
        FullNewAutomaton.append(Plant)
        if Plant.count("end")==1 or Plant.count("}") or Plant.count("needs")
            :
            endcount=1
        else:
60         Plant=f.readline()
f.close()

# do exactly the same for the old cif specification
FullOldAutomaton=[]
65 f = open(old)
searchcount=0
while searchcount==0:
    Plant=f.readline()
    Plant = Plant.strip()
70     if Plant.count("Requirement")>0:
        Plant=Plant.split("Requirement")[1].strip()
        if ((Plant.count(PQ_cif)==1 and Plant.count("plant")+Plant.count("
                                                    requirement")==1) or ((Plant.
                                                    count("{")==1 or Plant.count("
                                                    needs")==1) and PQ_new.count("
                                                    Invariant")==1 ) )and not Plant
                                                    .startswith("//"):

            searchcount=1
if ((Plant.count(PQ_cif)==1 and Plant.count("plant")+Plant.count("
                                                    requirement")==1) or ((Plant.
                                                    count("{")==1 or Plant.count("
                                                    needs")==1) and PQ_new.count("
                                                    Invariant")==1 ) )and not Plant.
                                                    startswith("//"):

75 FullOldAutomaton.append(Plant)
endcount=0
while endcount==0:

    Plant = Plant.strip()
80     FullOldAutomaton.append(Plant)
        if Plant.count("end")==1 or Plant.count("}") or Plant.count("needs")
            :
            endcount=1
        else:
            Plant=f.readline()

```

```
85     f.close

    # remove all empty specifications
    FullNewAutomaton = filter(None, FullNewAutomaton)
    FullOldAutomaton = filter(None, FullOldAutomaton)
90     # check whether all specifications which were already in the old one are
        the same
    # if not they are added to the list of changed automata
    if not (FullNewAutomaton == FullOldAutomaton):
        All_changed.append(PQ_new)

95     # check if there are automata in old, but not in new -> these are removed
    for PQ_old in PandQ_old:
        if not PQ_old in PandQ_new:
            All_removed.append(PQ_old)

100    return (All_introduced, All_removed, All_changed)
```

## B.5 CIF Metric Collector

This python function takes a CIF specification and returns all values for the metrics from that specification.

```

1  # Yorick van der Schriek 1-5-2018 - ASML - TU/e - CIF models of LOPW

import math

5

def count_string_occurrence_CIF(path):
    GenMatrix = [] # Creating an empty matrix
    controlledalphabet = generatealphabet(path, " controllable")
    uncontrolledalphabet = generatealphabet(path, " uncontrollable")
10  nr_automata = determine_automata(path)
    f = open(path)
    loopcount = 0
    invariant = 1
    # for all automata within path
15  while loopcount < nr_automata:
        #set all amounts to zero
        count = 0
        uncontrolled = 0
        controlled = 0
20  locations = 0
        marked_states = 0
        edges = 0
        guards=0
        TotalControlled=0
25  TotalUncontrolled = 0
        updates = 0
        gotos = 0
        StateVarcounts=0
        Integercounts = 0
30  TotalOperators = 0
        UniqueOperators = 0
        ListOfOperators = []
        ListOfEventNames = []
        Total_Operators =0
35  Unique_Operators=0
        Total_Operands = 0
        Unique_Operands =0
        Name = "none"

40  # while still in the same automaton
        while count == 0:
            Plant= f.readline()
            # remove commented lines
            if Plant.strip().startswith("//"):
45  Plant = " "

            # if partly commented, remove commented part
            if Plant.count("//")>=1:
50  Temp = Plant.split("//")
                x=Plant.count("//")
                Temp=Temp[0:len(Temp)-x]
                Plant= ''.join(Temp)

```

```

55     # if plant encountered save name
    if Plant.count("plant ") == 1:
        NameTmp = Plant
        NameTmp = Plant.split("plant ")
        NameTmp = NameTmp[1].split(":")
        Name = NameTmp[0]

60     # if requirement encountered save name
    if Plant.count("requirement ") == 1:
        NameTmp = Plant
        NameTmp = Plant.split("requirement ")
65     if NameTmp[1].count("{")==1:
        NameTmp = NameTmp[1].split("{")
        NameTmp = "Invariant " + str(invariant)
        invariant = invariant + 1
        Name = "Requirement " + NameTmp
70     elif NameTmp[1].count(":")==1:
        NameTmp = NameTmp[1].split(":")
        Name = "Requirement " + NameTmp[0]
    elif NameTmp[1].count("needs")==1:
        NameTmp = NameTmp[1].split("needs")
75     NameTmp = "Invariant " + str(invariant)
        invariant = invariant + 1
        Name = "Requirement " + NameTmp

    # add the amount of occurrences of these metrics to the total for this
    automaton
80     locations      = locations + Plant.count("location ")
    marked_states    = marked_states + Plant.count("marked;")
    guards           = guards + Plant.count(" when ")
    updates         = updates + Plant.count(" :=")
    gotos           = gotos + Plant.count(" goto ")
85

    # Update total and unique lists of operators
    if Plant.count(" not ")>=1:
        ListOfOperators.append("not")
        TotalOperators = TotalOperators + Plant.count(" not ")
90     if Plant.count(" and ")>=1:
        ListOfOperators.append("and")
        TotalOperators = TotalOperators + Plant.count(" and ")
    if Plant.count(" or ")>=1:
        ListOfOperators.append("or")
95     TotalOperators = TotalOperators + Plant.count(" or ")
    if Plant.count("=")>=1 and Plant.count("=")>Plant.count(":="):
        ListOfOperators.append("=")
        TotalOperators = TotalOperators + Plant.count("=") - Plant.count(":=")
100

    # count amount of discrete variable declarations
    if Plant.count("disc ")>=1:
        # count the amount of discrete integers untill a ; is encountered
        if Plant.count(" int")>=1:
105            Noend = 0
            while Noend == 0:
                Integercounts = Plant.count(",")+1
                if Plant.count(";") == 0:
                    Plant = f.readline()

```

```

110         else:
            Noend = 1
            # count the amount of discrete booleans untill a ; is encountered
            elif Plant.count(" bool ")>=1 or Plant.count(" enum ")>=1:
                Noend =0

115         while Noend ==0:
            StateVarcounts = Plant.count(",")+1
            if Plant.count(";") ==0:
                Plant = f.readline()
            else:
120                 Noend = 1

            #count amount of controllable event declarations in this automaton
            if Plant.count("controllable ")==1 and Plant.count("uncontrollable ") ==
                0:

125                Ccount=0
            while Ccount==0:
                C = 1 + Plant.count(",")
                if Plant.count(";") ==0:
                    Plant = f.readline()
                Ccount = Plant.count(";")
130                controlled= controlled + C

            # count amount of uncontrollable event declarations in this automaton
            if Plant.count("uncontrollable ")==1:
135                UCcount=0
            while UCcount==0:
                UC = 1 + Plant.count(",")
                if Plant.count(";") ==0:
                    Plant = f.readline()
                UCcount = Plant.count(";")
140                uncontrolled= uncontrolled + UC

            # count amount of events
            if (Plant.count("edge ")==1 or Plant.count("{")):

145                ControlledEdges=0
                UncontrolledEdges=0
                edgcount=0
                # edges can cover multiple lines so untill a ; is encountered:
                while edgcount==0:
150                    # add to counter
                    thisedge = 1 + Plant.count(",")

                    # save line in Names, by removing comma at end
                    if Plant.count("edge ")==1:
155                        Names = Plant.split("edge ")
                        Names = Names[len(Names)-1].split(",")
                    elif Plant.count("{")==1:
                        Names = Plant.split("{")
                        Names = Names[len(Names)-1].split(",")
160                    else:
                        Names = [Plant]

                    # if first line of the event declarations
                    if Plant.count("edge ")==1 or Plant.count("{"):

```



```

165     # if also last one
    if Plant.count(";")==1 or Plant.count("}"):

        # remove closure sign and assignments/guards
170     Tmp = Names[len(Names)-1].split(";")
        Tmp = Names[len(Names)-1].split("}")
        Names[len(Names)-1] = Tmp[0]
        if Plant.count("when ")==1:
            Tmp = Names[len(Names)-1].split("when ")
175     Names[len(Names)-1] = Tmp[0]
        elif Plant.count("do ")==1:
            indices = [i for i, s in enumerate(Names) if 'do ' in s]
            Tmp = Names[indices[0]].split("do ")
            Names[indices[0]] = Tmp[0]
180     elif Plant.count("goto ")==1:
            Tmp = Names[len(Names)-1].split("goto ")
            Names[len(Names)-1] = Tmp[0]
        else:
            Names = Names[:-1]
185

    # if not first one
    elif Plant.count(",")>=1:
        Names = Plant.split(",")

190     # if also last one remove closure sign and assignments/guards
    if Plant.count(";")==1 or Plant.count("}"):
        Names[len(Names)-1] = Names[len(Names)-1].split(";")[0]
        Names[len(Names)-1] = Names[len(Names)-1].split("}") [0]

195     if Plant.count("when ")==1:
            Tmp = Names[len(Names)-1].split("when ")
            Names[len(Names)-1] = Tmp[0]
        elif Plant.count("do ")==1:
            indices = [i for i, s in enumerate(Names) if 'do ' in s]
200     Tmp = Names[indices[0]].split("do ")
            Names[indices[0]] = Tmp[0]
        elif Plant.count("goto ")==1:
            Tmp = Names[len(Names)-1].split("goto ")
            Names[len(Names)-1] = Tmp[0]
205     else:
            Names = Names[:-1]

    # if last one
    else:
210     if Plant.count(";")==1 or Plant.count("}"):
            Names[len(Names)-1] = Names[len(Names)-1].split(";")[0]
            Names[len(Names)-1] = Names[len(Names)-1].split("}") [0]
        else:
            Names = Names
215     if Plant.count("when ")==1:
            Tmp = Names[len(Names)-1].split("when ")
            Names[len(Names)-1] = Tmp[0]
        elif Plant.count("do ")==1:
            indices = [i for i, s in enumerate(Names) if 'do ' in s]
220     Tmp = Names[indices[0]].split("do ")
            Names[indices[0]] = Tmp[0]

```

```

elif Plant.count("goto")==1:
    Tmp = Names[len(Names)-1].split("goto ")
    Names[len(Names)-1] = Tmp[0]
225 Z=0

while Z < len(Names):
    # check for correct names, if not adjust
    # if correct check with alphabet for controllability of event and
    # add to correct one
230 Names[Z]=Names[Z].strip()
    if Names[Z].count(".")==1:
        ListOfEventNames.append(Names[Z])
    if Names[Z] in controlledalphabet:
        ControlledEdges = ControlledEdges+1
235 if Names[Z] in uncontrolledalphabet:
        UncontrolledEdges = UncontrolledEdges+1
    Names[Z] = Name + "." + Names[Z].strip()
    if Names[Z].count(".")==1:
        ListOfEventNames.append(Names[Z])
240 if Names[Z] in controlledalphabet:
        ControlledEdges = ControlledEdges+1
    if Names[Z] in uncontrolledalphabet:
        UncontrolledEdges = UncontrolledEdges+1
    Z=Z+1
245

# if not yet the end of event declaration do readline and do loop
# again for next line
edgecount = Plant.count(";")+Plant.count("}")
if Plant.count(";") + Plant.count("}") ==0:
    Plant = f.readline()
250

#save total amount of controlled and uncontrolled events per automaton
TotalControlled = TotalControlled + ControlledEdges
TotalUncontrolled = TotalUncontrolled + UncontrolledEdges
edges = TotalControlled +TotalUncontrolled
255

#when automaton end goto next one
count = Plant.count("end") + Plant.count("needs")

# calculation of other metrics and compensation for errors in invariant
# requirements
260 count =0
if gotos ==0:
    gotos = 1
if TotalControlled ==0:
    TotalControlled = 1
265 import re
Rev = int(re.search(r'\d+', path).group())
if Name.count("Invariant")==0:
    pseudo_states = locations - marked_states
    FP = 12* TotalUncontrolled + 5* edges + 1*TotalControlled + 2*guards +
    updates
270 CC = edges - locations +1
ACC = gotos - locations +1
Unique_Events = len(list(set(ListOfEventNames).intersection(
    controlledalphabet)) +list(set(
    ListOfEventNames).intersection(

```

```

                uncontrolledalphabet)))
Unique_Operators = len(list(set(ListOfOperators)))+Unique_Events
Total_Operators = edges + TotalOperators
275 Unique_Operands = locations + StateVarcounts
Total_Operands = locations + guards

Volume = (Total_Operators +Total_Operands) * math.log(2*(
                Unique_Operators +
                Unique_Operands),2)
Difficulty = (float(Unique_Operators)/2) * (float(Total_Operands)/float(
                Unique_Operands))
280 Effort = float(Volume) * float(Difficulty)
Time = float(Effort) / 18
Errors = float(Volume)/3000
if Volume>0 and edges>0:
    MI= 171-5.2*math.log(float(Volume))-0.23*float(CC)-16.2*math.log(float
                (edges))
285    MIC = max(0,( 171-5.2*math.log(float(Volume))-0.23*ACC-16.2*math.log(
                float(edges))))*(float(100)/172
                )

    else:
        MI =0
        MIC = 0
else:
290 pseudo_states = 0
FP = 12* TotalUncontrolled + 5* edges + 1*TotalControlled + 2*guards +
                updates

CC = 0
ACC = 0
Unique_Events = len(list(set(ListOfEventNames).intersection(
                controlledalphabet)) +list(set(
                ListOfEventNames).intersection(
                uncontrolledalphabet)))
295 Unique_Operators = len(list(set(ListOfOperators)))+Unique_Events
Total_Operators = edges + TotalOperators
Unique_Operands = locations + StateVarcounts
Total_Operands = locations + guards

300
U00 = max(1,Unique_Events+Unique_Operands)
Volume = (Total_Operators +Total_Operands) * math.log(2*(U00),2)
Difficulty = 0
Effort = float(Volume) * float(Difficulty)
305 Time = float(Effort) / 18
Errors = float(Volume)/3000

MI= 0
MIC = 0
310 ThisPart = [Rev, Name, FP, locations, TotalControlled, "0", "0", "0", edges,
                TotalUncontrolled, StateVarcounts,
                "0" , UniqueOperators,
                TotalOperators, "0", "0", CC, ACC,
                edges, Unique_Operators,
                Total_Operators, Unique_Operands,
                Total_Operands, Volume, Difficulty,
                Effort, Time, Errors, MI, MIC ]

GenMatrix= GenMatrix + [ThisPart]

```

```
    loopcount = loopcount + 1  
f.close()  
315 return (GenMatrix)
```

## B.6 Runtime Script

This python script uses all previous Python functions to generate a changelog and a metrics matrix for all CIF specifications in the folder the file is in. It writes the results into two comma separated value(CSV) files.

```

1  # Yorick van der Schriek 1-5-2018 - ASML - TU/e - CIF models of LOPW

matrix_CIF = [[0]*(items)]*0 # Creating an empty matrix
5  matrix_Changed = [[0]*(items)]*0 # Creating an empty matrix
count = 0
filecount = 0
currentfilecount = 0
Complete_PandReq_list = []
10 Reqlist = []
Plantlist = []

import os
15 Old = "empty"
print "Generating Complete list of Plants and Requirements"

# for all cif specifications in this folder:
for subdir, dirs, files in os.walk("."):
20     for filename in files:
        filepath = subdir + os.sep + filename
        if filepath.startswith(".") + os.sep + "1" and filepath.endswith(".cif"):
            filecount=filecount+1
    for filename in files:
25     filepath = subdir + os.sep + filename
        if filepath.startswith(".") + os.sep + "1" and filepath.endswith(".cif"):
            # print progress
            currentfilecount = currentfilecount + 1
            print "Progress : " + str(int((float(currentfilecount)/float(filecount))
30                                     *100)) + "%"

#check which automata are added in each revision
Thischanged = determine_changed_CIF(Old, filepath)[0]
for i in Thischanged:
    # add them to list
35     Complete_PandReq_list.append(i)

# remove duplicates
Complete_PandReq_list = list(set(Complete_PandReq_list))

40 # sort in requirements and plants
for i in Complete_PandReq_list:
    if i.startswith("Requirement"):
        Reqlist.append(i)
    else:
45     Plantlist.append(i)
# sort them alphabetically
Reqlist = sorted(list(set(Reqlist)), key=str.lower)
Plantlist = sorted(list(set(Plantlist)), key=str.lower)
# add together, plants sorted and requirements sorted
50 Complete_PandReq_list=Plantlist+Reqlist

```

```

        count = count + 1
        #set current file to old for next iteration
        Old = filepath

55 #add extra entry and use as header in matrix
    Complete_PandReq_list.insert(0,"")
    matrix_Changed.append(Complete_PandReq_list)

    Old = "empty"
60 currentfilecount =0
    print "Start extraction of Metrics and Evaluation of Changes"
    # for all cif files in this folder:
    for subdir, dirs, files in os.walk("."):
        for filename in files:
65         filepath = subdir + os.sep + filename
            if filepath.startswith(".") +os.sep+"1" and filepath.endswith(".cif"):
                # print progress
                currentfilecount = currentfilecount +1
                print "Progress : " + str(int((float(currentfilecount)/float(filecount))
                    *100)) + "%"

70         # add metrics to matrix
            matrix_CIF.append(count_string_occurence_CIF(filepath))
            # create vector for changes
            ChangedVector = [""] * len(Complete_PandReq_list)
75         # find out which automata are changed
            ThisAdded ,ThisRemoved, ThisChanged= determine_changed_CIF(Old, filepath
                )

            # print nice in matrix
            for i in ThisAdded:
                AutomatonIndex = Complete_PandReq_list.index(i)
80                 ChangedVector[AutomatonIndex] = "Added"
            for i in ThisRemoved:
                AutomatonIndex = Complete_PandReq_list.index(i)
                ChangedVector[AutomatonIndex] = "Removed"
            for i in ThisChanged:
85                 AutomatonIndex = Complete_PandReq_list.index(i)
                ChangedVector[AutomatonIndex] = "Changed"
            # add filename
            ChangedVector[0]=filename
            # append to total matrix
90            matrix_Changed.append(ChangedVector)

        count = count + 1
        Old = filepath
    print "Analysis Complete"
95 # Create a header for Metrics csv output
    Header =["Revision"," Model"," Function Points","States"," Actions","Blocked
        ", "Illegal","NoOp","Rulecases","Events"
        , "State variables as guard","Data
        Variables in events and actions","XX
        Unique Operators on state variables (
        NOT, and, or, >,<, ==, +,- and
        otherwise)"," Operators on state
        variables (NOT, and, or, >,<, ==, +,-
        and otherwise)","Unique Operators on
        data vaiables (>>,<<,>< and $)","

```

```

Operators on data vaiables (>>, <<, ><
and $)", "CC", "ACC", "LOC", " Halstead n1
", "Halstead N1", "Halstead n2", "Halstead
N2", " Volume", "Difficulty", "Effort", "
Time", " Expected number of Bugs", "MI",
"MI (0-100)"]

# print all data to CSV
import csv
100 metricsfilename = "metrics.csv"
changesfilename = "changes.csv"
changesfilename_tp = "changes_tp.csv"

#Metrics
105 with open(metricsfilename, "wb") as f:
    writer = csv.writer(f)
    x_count=0
    writer.writerow(Header)
    print "Start Writing metrics to CSV file: " + metricsfilename
110 for x in matrix_CIF:
    for y in matrix_CIF[x_count]:
        writer.writerow(y)
        x_count = x_count+1

115 f.close

#Changed Automata
with open(changesfilename, "wb") as f:
    writer = csv.writer(f)
120 x_count=0
    print "Start Writing changes to CSV file: " + changesfilename
    for x in matrix_Changed:
        writer.writerow(x)
        x_count = x_count+1

125 f.close
print "Finished changes to CSV file: " + changesfilename
# Changed Automata Transposed:
print "Transpose Changes File: " + changesfilename_tp
130 from itertools import izip
a = izip(*csv.reader(open(changesfilename, "rb")))
csv.writer(open(changesfilename_tp, "wb")).writerows(a)
print " Finished!! "

```





### C.3 Component C

ASD:	1357112586	1357112727	1355673387	1361291375	1363781023	1361414063
D.dim	Added	Changed	Changed	Changed	Changed	Changed
D.im	Added	Changed	Changed	Changed	Changed	Changed
P1.im	Added	Changed	Changed	Changed	Changed	Changed
P2.im	Added	Changed	Changed	Changed	Changed	Changed
P3.im	Added	Changed	Changed	Changed	Changed	Changed
P4.im	Added	Changed	Changed	Changed	Changed	Changed
CIF:						
D	Added	Changed	Changed	Changed	Changed	Changed
P1.im	Added	Changed	Changed	Changed	Changed	Changed
P2.im	Added	Changed	Changed	Changed	Changed	Changed
EP1	Added	Removed	Added	Removed	Removed	Removed
EP2	Added	Added	Added	Added	Added	Added
EP3	Added	Added	Added	Added	Added	Added
R1	Added	Changed	Added	Changed	Changed	Changed
R2	Added	Changed	Added	Changed	Changed	Changed
R3	Added	Changed	Added	Changed	Changed	Changed
R4	Added	Added	Added	Added	Added	Added
R5	Added	Added	Added	Added	Added	Added
R6	Added	Added	Added	Added	Added	Added
R7	Added	Added	Added	Added	Added	Added
R8	Added	Added	Added	Added	Added	Added
R9	Added	Added	Added	Added	Added	Added
R10	Added	Added	Added	Added	Added	Added
R11	Added	Added	Added	Added	Added	Added

Figure C.3: Changelog of component C

### C.4 Component D

ASD:	1357112586	1357112727	1355673387	1361291375	1363781023	1361414063
D.dim	Added	Changed	Changed	Changed	Changed	Changed
D.im	Added	Changed	Changed	Changed	Changed	Changed
P1.im	Added	Changed	Changed	Changed	Changed	Changed
P2.im	Added	Changed	Changed	Changed	Changed	Changed
CIF:						
D	Added	Changed	Changed	Changed	Changed	Changed
P1.im	Added	Changed	Changed	Changed	Changed	Changed
P2.im	Added	Changed	Changed	Changed	Changed	Changed
EP1	Added	Added	Added	Added	Added	Added
EP2	Added	Added	Added	Added	Added	Added
Q1	Added	Added	Added	Added	Added	Added
Q2	Added	Added	Added	Added	Added	Added
Q3	Added	Added	Added	Added	Added	Added
Q4	Added	Added	Added	Added	Added	Added
Q5	Added	Added	Added	Added	Added	Added
Q6	Added	Added	Added	Added	Added	Added
Q7	Added	Added	Added	Added	Added	Added
Q8	Added	Added	Added	Added	Added	Added
Q9	Added	Added	Added	Added	Added	Added
Q10	Added	Added	Added	Added	Added	Added
Q11	Added	Added	Added	Added	Added	Added
Q12	Added	Added	Added	Added	Added	Added
Q13	Added	Added	Added	Added	Added	Added
Q14	Added	Added	Added	Added	Added	Added
Q15	Added	Added	Added	Added	Added	Added
Q16	Added	Added	Added	Added	Added	Added
Q17	Added	Added	Added	Added	Added	Added
Q18	Added	Added	Added	Added	Added	Added
Q19	Added	Added	Added	Added	Added	Added
Q20	Added	Added	Added	Added	Added	Added
Q21	Added	Added	Added	Added	Added	Added
Q22	Added	Added	Added	Added	Added	Added
Q23	Added	Added	Added	Added	Added	Added
R1	Added	Added	Added	Added	Added	Added
R2	Added	Added	Added	Added	Added	Added
R3	Added	Added	Added	Added	Added	Added
R4	Added	Added	Added	Added	Added	Added
R5	Added	Added	Added	Added	Added	Added
R6	Added	Added	Added	Added	Added	Added
R7	Added	Added	Added	Added	Added	Added
R8	Added	Added	Added	Added	Added	Added
R9	Added	Added	Added	Added	Added	Added
R10	Added	Added	Added	Added	Added	Added
R11	Added	Added	Added	Added	Added	Added
R12	Added	Added	Added	Added	Added	Added
R13	Added	Added	Added	Added	Added	Added
R14	Added	Added	Added	Added	Added	Added
R15	Added	Added	Added	Added	Added	Added
R16	Added	Added	Added	Added	Added	Added
R17	Added	Added	Added	Added	Added	Added
R18	Added	Added	Added	Added	Added	Added
R19	Added	Added	Added	Added	Added	Added
R20	Added	Added	Added	Added	Added	Added
R21	Added	Added	Added	Added	Added	Added

Figure C.4: Changelog of component D

## C.5 Component E

	1357112445	1357112586	1357112727	1366723611	1367490482
E.dm	Added	Changed	Changed		Changed
E.im	Added	Changed	Changed		Changed
PL.im	Added	Changed	Changed	Changed	Changed
P2.im	Added			Changed	
<b>CIF:</b>					
E	Added	Changed	Changed		Changed
P1	Added	Changed	Changed	Changed	Changed
P2	Added			Changed	
EP1	Added				
EP2	Added	Changed			Changed
Q1	Added				
Q2			Added		
Q3	Added	Removed			
Q4	Added	Removed			
Q5	Added				
Q6					Added
Q7			Added		
Q8	Added				
Q9	Added				
Q10	Added				
Q11	Added				
R1	Added				
R2	Added				
R3	Added	Removed			
R4			Added		
R5	Added	Removed			
R6	Added	Removed			
R7	Added	Removed			
R8	Added				
R9	Added				
R10					Added
R11					Added
R12			Added		
R13	Added	Removed			
R14	Added	Changed			
R15	Added	Removed			
R16	Added	Removed			
R17	Added	Removed			
R18	Added				
R19	Added				
R20	Added	Removed			
R21	Added				
R22	Added				
R23	Added	Removed			
R24	Added				
R25	Added				
R26	Added				
R27	Added				
R28	Added	Removed			

Figure C.5: Changelog of component E

## C.6 Component F

Figure C.6: Changelog of component F

## C.7 Component G

ASD:	1377154381	1379339175	1379596412	1413460241	1443626973	1449484128	1449497233
G.dm	Added						
G.im	Added						
P1.im	Added		Changed				
P2.im	Added	Changed		Changed	Changed		
CIF:							
G	Added				Changed		
P1	Added		Changed				
P2	Added	Changed		Changed	Changed	Changed	
EP1					Added		
EP2					Added		
R1	Added				Changed		
R2	Added						
R3	Added						
R4	Added						
R5	Added						
R6	Added						
R7	Added						
R8	Added						
R9	Added						
R10	Added						
R11	Added						

Figure C.7: Changelog of component G

## C.8 Component H

ASD:	1358E+09	136E+09	136E+09	1367E+09	137E+09	1376E+09	1377E+09	1378E+09	1379E+09	1379E+09	138E+09	138E+09	1382E+09	1383E+09	1383E+09	1385E+09	1385E+09	1387E+09	1388E+09	1389E+09	139E+09	14E+09	143E+09	1434E+09	1435E+09	1435E+09	1444E+09	1432E+09	1507E+09	
H.dm	Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	
H.lm	Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	
P1.lm	Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	
P2.lm	Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	
P3.lm	Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	
CIF:																														
H	Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	
P1	Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	
P2	Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	
E1	Added																													
E2	Added																													
E3	Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	
E4	Added																													
R1	Added																													
R2	Added																													
R3	Added																													
R4	Added																													
R5	Added																													
R6	Added																													
R7	Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	
R8		Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	Added	
R9																														
R10																														
R11																														
R12	Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	
R13	Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	
R14	Added	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	Changed	

Figure C.8: Changelog of component H

## D. Metric Data Per Revision

For each component the most important metric results are shown in a table in this Appendix. The sum is taken over all automata and models within the component for most metrics, except the difficulty and the MI, which are taken average.

### D.1 Component A

**Table D.1:** *Most imporant metrics ASD for Component A for each revision*

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1357112586	91		174.88		7.63	62.19	10		107	383	
1357112631	103	12	200.43	25.55	8.63	60.84	10	0	119	434	51
1357742150	139	36	283.54	83.11	10.75	58.41	12	2	165	522	88
1357837223	71	68	136.23	147.32	6.75	64.00	8	4	79	354	168
1360749381	114	43	237.04	100.82	9.88	59.25	10	2	133	494	140
1361971024	124	10	274.03	36.99	11.00	58.33	10	0	143	539	45
1366723611	126	2	268.56	5.47	11.25	57.47	10	0	145	557	18
1386690679	1002	876	2765.92	2497.35	55.00	44.70	11	1	1269	5545	4988
1389007705	1040	38	2884.77	118.85	56.90	44.50	11	0	1319	5755	210
1392718014	1042	2	2891.28	6.51	57.00	44.49	11	0	1321	5809	54
1396278097	1026	16	2840.75	50.53	56.20	44.57	11	0	1301	5725	84
1424852602	1042	16	2894.20	53.45	57.20	44.50	11	0	1317	5869	144
1425576006	1046	4	2907.24	13.04	57.40	44.49	11	0	1321	5905	36
1425640172	1054	8	2934.06	26.81	57.90	44.46	11	0	1329	5977	72
1426086770	1056	2	2940.59	6.54	58.00	44.45	11	0	1331	5995	18
1430901241	1072	16	2989.08	48.49	58.70	44.36	11	0	1355	6079	84
1432655913	1030	42	2858.92	130.16	56.70	44.58	11	0	1297	5857	222
1433171121	1044	14	2912.80	53.88	57.40	45.24	11	0	1309	5969	112
1433350588	966	78	2671.25	241.56	53.60	45.68	11	0	1203	5641	328
1435164023	968	2	2677.73	6.48	53.70	45.67	11	0	1205	5659	18
1435340586	908	60	2475.54	202.19	49.80	45.96	11	0	1149	5299	360
1436285398	958	50	2591.88	116.34	51.80	45.33	11	0	1232	5481	182
1436367075	919	39	2507.54	84.33	50.30	45.68	11	0	1160	5319	162
1480579833	977	58	2745.20	237.66	51.29	56.34	14	3	1272	5353	34
1480690994	915	62	2481.04	264.16	36.07	56.80	13	1	1146	5343	10
1481285276	983	68	2664.91	183.86	36.64	56.33	14	1	1278	5377	34
1492167019	975	8	2637.43	27.48	36.23	56.42	14	0	1270	5317	60
1500551082	985	10	2651.91	14.48	37.00	54.22	14	0	1280	5425	108
1505139377	989	4	2662.45	10.54	37.23	54.01	14	0	1284	5443	18
Average	785	58.71	2143.49	171.21	40.96	50.80	11.34	0.5	994.45	4393.97	280.29

**Table D.2:** *Most important metrics CIF for Component A for each revision*

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1357112586	47		368.58		2.15	72.02	28		48	408	
1357112631	58	11	466.88	98.30	2.77	71.27	31	3	59	521	113
1357742150	72	14	640.22	173.35	3.27	70.11	36	5	73	613	92
1357837223	53	19	428.68	211.55	3.06	70.81	27	9	54	495	118
1360749381	63	10	530.47	101.79	2.63	71.89	33	6	64	558	63
1361971024	69	6	603.39	72.92	2.77	72.60	36	3	70	613	55
1366723611	70	1	608.53	5.14	2.81	72.40	36	0	71	619	6
1386690679	350	280	3365.94	2757.40	10.17	68.43	61	25	351	3315	2696
1389007705	360	10	3483.53	117.59	10.49	68.40	62	1	361	3408	93
1392718014	361	1	3486.77	3.24	10.58	68.41	61	1	362	3392	16
1396278097	357	4	3443.95	42.82	10.46	68.42	61	0	358	3357	35
1424852602	365	8	3529.67	85.72	10.71	68.40	61	0	366	3405	48
1425576006	367	2	3551.14	21.47	10.77	68.40	61	0	368	3417	12
1425640172	371	4	3594.15	43.00	10.89	68.38	61	0	372	3441	24
1426086770	372	1	3604.91	10.76	10.92	68.38	61	0	373	3447	6
1430901241	377	5	3669.58	64.67	11.08	68.36	62	1	378	3499	52
1432655913	363	14	3489.99	179.58	10.71	68.43	59	3	364	3349	150
1433171121	358	5	3431.83	58.16	11.16	67.47	56	3	359	3304	45
1433350588	344	14	3285.94	145.89	10.99	67.64	55	1	345	3066	238
1435164023	345	1	3296.58	10.64	11.02	67.64	55	0	346	3072	6
1435340586	322	23	3073.07	223.51	10.89	67.56	52	3	323	2915	157
1436285398	367	45	3546.75	473.67	11.39	67.42	58	6	368	3402	487
1436367075	72	295	697.25	2849.49	3.42	71.32	36	22	75	782	2620
1480579833	351	279	3376.66	2679.40	10.25	69.05	59	23	354	3281	2499
1480690994	72	279	697.25	2679.40	3.42	71.32	36	23	75	782	2499
1481285276	365	293	3444.67	2747.41	8.90	67.03	66	30	369	3359	2577
1492167019	363	2	3423.41	21.26	8.86	67.03	66	0	367	3347	12
1500551082	368	5	3444.01	20.60	8.98	66.24	66	0	372	3377	30
1505139377	369	1	3454.63	10.62	9.00	66.23	66	0	373	3383	6
Average	267.97	58.29	2553.05	568.19	8.09	69.00	52	6	269.59	2480.24	526.96

## D.2 Component B

**Table D.3:** *Most important metrics ASD for Component B for each revision*

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1407419954	1503		3851.78		62.80	36.20	7		1912	8922	
1424852602	1509	6	3870.60	18.82	63.10	36.16	7	0	1918	8976	54
1425579459	1543	34	3974.58	103.98	64.90	35.93	7	0	1970	9204	228
1425640172	1551	8	3993.74	19.16	64.60	35.90	7	0	1978	9132	72
1430901241	1577	26	4075.82	82.08	66.20	35.70	7	0	2010	9384	252
1432038457	1575	2	4067.29	8.53	65.90	35.71	7	0	2008	9366	18
1433777296	1513	62	3877.85	189.43	64.10	35.90	7	0	1922	9108	258
1435340586	1459	54	3717.83	160.03	62.80	36.14	7	0	1868	8790	318
1436285398	1477	18	3769.96	52.13	63.40	36.03	7	0	1904	8862	72
1436367075	1467	10	3742.45	27.51	63.10	36.10	7	0	1876	8850	12
1443626973	1470	3	3749.16	6.71	62.70	36.15	7	0	1888	9456	606
1456942487	1496	26	3823.60	74.44	64.30	35.80	7	0	1914	9006	450
1477672466	1494	2	3814.61	8.99	64.10	35.79	7	0	1906	9018	12
1492167019	1490	4	3798.15	16.46	63.80	35.77	7	0	1902	8950	68
Average	1508.86	19.62	3866.24	59.10	63.99	35.95	7	0	1926.86	9073.14	186.15

**Table D.4:** *Most important metrics CIF for Component B for each revision*

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1407419954	213		1851.41		7.92	54.27	46		217	2209	
1424852602	214	1	1860.98	9.57	7.96	54.26	46	0	218	2215	6
1425579459	226	12	1989.76	128.78	8.29	54.15	48	2	230	2397	182
1425640172	221	5	1931.63	58.13	8.08	54.19	47	1	225	2323	74
1430901241	225	4	1968.76	37.13	8.25	54.14	47	0	229	2347	24
1432038457	225	0	1968.76	0.00	8.25	54.14	47	0	229	2347	0
1433777296	227	2	1988.14	19.38	8.33	54.13	47	0	231	2359	12
1435340586	232	5	2041.90	53.76	8.54	54.05	48	1	236	2422	63
1436285398	222	10	1940.52	101.38	8.13	54.17	47	1	226	2329	93
1436367075	232	10	2041.90	101.38	8.54	54.05	48	1	236	2422	93
1443626973	235	3	2078.05	36.15	8.67	54.00	49	1	239	2462	40
1456942487	237	2	2097.46	19.41	8.75	53.99	49	0	241	2474	12
1477672466	241	4	2135.72	38.26	8.92	53.95	49	0	245	2498	24
1492167019	245	4	2174.77	39.05	9.08	53.92	49	0	249	2522	24
Average	228.21	4.77	2004.98	49.41	8.41	54.10	47.64	0.54	232.21	2380.43	49.77

### D.3 Component C

**Table D.5:** *Most important metrics ASD for Component C for each revision*

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1377096893	878		2305.17		49.63	45.53	7		1127	4933	
1377701992	882	4	2315.98	10.81	49.63	45.54	7	0	1133	4979	46
1379338175	890	8	2339.23	23.25	50.00	45.47	7	0	1145	5015	36
1380621927	924	34	2446.16	106.92	52.00	45.26	7	0	1183	5135	120
1381158560	904	20	2384.17	61.99	50.88	45.39	7	0	1159	5129	6
1381422145	922	18	2439.71	55.54	51.88	45.27	7	0	1181	5231	102
1382600014	930	8	2463.10	23.39	52.25	45.21	7	0	1193	5257	26
1382601245	924	6	2443.74	19.35	51.88	45.24	7	0	1187	5225	32
1384368727	926	2	2445.53	1.78	52.25	44.59	7	0	1189	5243	18
1389007705	983	57	2615.62	170.09	56.13	43.23	7	0	1258	5588	345
1389197464	995	12	2654.40	38.78	56.63	43.12	7	0	1278	5648	60
1389278792	1147	152	3019.59	365.19	57.13	41.85	9	2	1574	5696	48
1389773888	1643	496	4309.17	1289.58	62.60	38.02	10	1	2310	7816	2120
1389773899	1173	470	3075.65	1233.52	47.10	46.30	10	0	1600	5890	1926
1392718014	1161	12	3034.61	41.04	46.60	46.18	10	0	1588	5830	60
1396278097	1145	16	2988.42	46.19	46.10	46.22	10	0	1565	5759	71
1396278328	1127	18	2958.45	29.97	56.38	41.72	9	1	1547	5637	122
1424852602	1145	18	3014.89	56.43	57.63	41.44	9	0	1565	5799	162
1425576006	1149	4	3027.89	13.00	57.88	41.42	9	0	1569	5835	36
1425640172	1157	8	3053.93	26.04	58.38	41.38	9	0	1577	5907	72
1426086770	1159	2	3060.44	6.52	58.50	41.37	9	0	1579	5925	18
1430901241	1179	20	3118.38	57.94	59.25	41.25	9	0	1611	6009	84
1432655913	1129	50	2974.78	143.60	57.13	41.55	9	0	1537	5787	222
1433350588	1067	62	2791.64	183.14	55.25	41.63	9	0	1433	5667	120
1435164023	1069	2	2798.10	6.46	55.38	41.62	9	0	1435	5685	18
1435340586	1009	60	2609.04	189.06	51.38	42.28	9	0	1375	5313	372
1440512008	1017	8	2634.59	25.55	51.88	42.24	9	0	1383	5373	60
1443103043	1091	74	2831.44	196.85	54.25	41.05	9	0	1523	6093	720
1443626973	1137	46	3007.72	176.28	59.38	40.93	9	0	1515	5985	108
1460552700	1045	92	2710.07	297.65	53.63	41.41	9	0	1423	5517	468
1477656517	908	137	2377.15	332.91	53.38	45.61	7	2	1153	5531	14
1477672466	886	22	2311.68	65.47	52.25	45.78	7	0	1123	5405	126
1480579833	878	8	2286.15	25.53	51.75	45.83	7	0	1115	5333	72
1480690994	886	8	2311.68	25.53	52.25	45.78	7	0	1123	5405	72
1481285276	878	8	2286.15	25.53	51.75	45.83	7	0	1115	5333	72
1492167019	870	8	2260.67	25.48	51.25	45.88	7	0	1107	5333	0
1500551082	868	2	2254.31	6.36	51.13	45.89	7	0	1105	5333	0
Average	1029.22	54.78	2701.61	150.08	53.59	43.63	8.14	0.17	1367.11	5583.22	220.94

**Table D.6:** *Most important metrics CIF for Component C for each revision*

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1377096893	294		2659.18		10.98	66.52	39		295	2935	
1377701992	285	9	2601.47	57.71	11.32	67.13	37	2	286	2785	150
1379338175	288	3	2633.15	31.68	11.45	67.10	37	0	289	2814	29
1380621927	303	15	2797.99	164.84	12.00	66.99	38	1	304	2948	134
1381158560	295	8	2707.25	90.74	11.77	67.07	37	1	296	2856	92
1381422145	303	8	2797.99	90.74	12.00	66.99	38	1	304	2948	92
1382600014	305	2	2819.30	21.31	12.09	66.96	38	0	306	2971	23
1382601245	303	2	2797.99	21.31	12.00	66.97	38	0	304	2959	12
1384368727	305	2	2810.37	12.37	12.09	66.46	39	1	306	2982	23
1389007705	331	26	3055.33	244.97	10.43	69.96	47	8	332	3193	211
1389197464	332	1	3066.05	10.72	10.46	69.95	47	0	333	3199	6
1389278792	340	8	3127.49	61.43	9.47	70.87	52	5	341	3286	87
1389773888	449	109	4090.39	962.91	11.33	69.28	68	16	450	4338	1052
1389773899	351	98	3217.04	873.35	8.75	71.66	57	11	352	3376	962
1392718014	348	3	3173.26	43.79	8.72	71.64	56	1	349	3336	40
1396278097	354	6	3220.90	47.64	9.39	69.78	56	0	355	3476	140
1396278328	349	5	3187.65	33.25	9.82	69.54	53	3	350	3435	41
1424852602	358	9	3280.25	92.60	10.09	69.47	53	0	359	3489	54
1425576006	360	2	3301.76	21.51	10.15	69.46	53	0	361	3501	12
1425640172	364	4	3344.84	43.08	10.26	69.45	53	0	365	3525	24
1426086770	365	1	3355.62	10.78	10.29	69.45	53	0	366	3531	6
1430901241	370	5	3420.40	64.78	10.44	69.42	54	1	371	3583	52
1432655913	358	12	3269.50	150.90	10.09	69.48	52	2	359	3467	116
1433350588	342	16	3086.73	182.77	9.94	69.57	50	2	343	3184	283
1435164023	343	1	3097.38	10.65	9.97	69.57	50	0	344	3190	6
1435340586	327	16	2935.06	162.32	9.50	69.69	50	0	328	3094	96
1440512008	330	3	2966.83	31.77	9.59	69.68	50	0	331	3112	18
1443103043	368	38	3379.50	412.67	10.38	69.25	54	4	369	3571	459

Continued on next page

Table D.6 – Continued from previous page

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1443626973	335	33	3012.09	367.41	9.74	69.48	51	3	336	3164	407
1460552700	336	1	3022.69	10.61	9.76	69.48	51	0	337	3170	6
1477656517	317	19	2915.77	106.93	13.27	66.85	38	13	318	2898	272
1477672466	309	8	2821.89	93.88	12.95	66.93	37	1	310	2817	81
1480579833	305	4	2779.60	42.29	12.77	66.94	37	0	306	2793	24
1480690994	309	4	2821.89	42.29	12.95	66.93	37	0	310	2817	24
1481285276	305	4	2779.60	42.29	12.77	66.94	37	0	306	2793	24
1492167019	303	2	2758.49	21.11	12.68	66.95	37	0	304	2781	12
1500551082	302	1	2747.94	10.55	12.64	66.96	37	0	303	2775	6
Average	330.84	13.56	3023.26	130.28	10.93	68.56	46.24	2.11	331.84	3164.65	141

## D.4 Component D

Table D.7: Most important metrics ASD for Component D for each revision

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1357112586	264		709.93		28.50	50.80	5		295	1555	
1357112727	271	7	728.72	18.79	29.25	50.52	5	0	303	1585	30
1358679387	324	53	953.52	224.80	35.75	48.29	5	0	358	1920	335
1361291375	302	6	854.71	6.00	28.38	49.01	5	0	334	1810	6
1363781023	321	19	945.39	90.68	35.50	48.37	5	0	355	1923	113
1381414063	342	21	1002.36	56.97	37.75	46.88	5		378	2054	131
Average	304	21.2	865.77	79.45	32.52	48.98	5	0	337.17	1807.83	123

Table D.8: Most important metrics CIF for Component D for each revision

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1357112586	248		2130.29		5.67	78.06	92		249	2049	
1357112727	217	31	1924.54	205.00	5.14	79.14	94	2	219	2271	222
1358679387	246	29	2409.72	485.18	8.02	79.08	99	5	248	2759	488
1361291375	208	6	1817.27	6.00	4.86	79.22	93	6	210	2191	6
1363781023	219	11	1933.10	115.83	5.03	79.44	98	5	221	2307	116
1381414063	230	11	2048.76	115.66	5.20	79.58	103	5	232	2412	105
Average	228	17.6	2043.95	185.53	5.65	79.09	96.5	4.6	229.83	2331.5	187.4

## D.5 Component E

Table D.9: Most important metrics ASD for Component E for each revision

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1357112445	274		715.49		23.63	51.04	7		363	1321	
1357112586	108	166	286.80	428.69	13.75	59.23	4	3	118	714	607
1357112727	120	12	307.38	20.59	13.50	58.23	4	0	130	774	60
1366723611	134	14	384.78	77.39	17.13	55.68	4	0	144	888	114
1367490482	159	25	468.97	84.19	20.25	53.88	4	0	171	1022	134
Average	159	54.25	432.68	152.72	17.65	55.61	4.6	0.75	185.2	943.8	228.75

Table D.10: Most important metrics CIF for Component E for each revision

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1357112445	136		1040.23		2.25	80.91	73		143	1175	
1357112586	76	60	575.34	464.89	2.00	79.79	44	29	82	688	487
1357112727	80	4	606.36	31.02	1.79	81.69	52	8	88	736	48
1366723611	81	1	611.50	5.14	1.80	81.59	52	0	89	742	6
1367490482	91	10	722.10	110.60	1.84	82.19	60	8	100	844	102
Average	92.8	18.75	711.11	152.91	1.93	81.23	56.2	11.25	100.4	837	160.75



## D.6 Component F

Table D.11: Most important metrics ASD for Component F for each revision

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1357112586	354		790.29		18.25	52.93	15		474	1461	
1357112589	372	18	835.33	45.04	19.08	52.58	15	0	502	1545	84
1357112598	414	42	940.73	105.40	20.58	51.96	15	0	574	1665	120
1357112679	443	29	1016.11	75.38	22.00	51.51	15	0	618	1779	114
1357112685	425	18	967.20	48.91	20.50	51.92	15	0	600	1659	120
1357112696	1471	1046	3817.37	2850.17	50.13	45.13	19	4	2202	5625	3966
1357112719	1471	0	3817.37	0.00	50.13	45.13	19	0	2202	5613	12
1357112725	1481	10	3845.05	27.68	50.75	44.95	19	0	2212	5667	54
1357742150	1632	151	4268.54	423.49	52.38	44.12	21	2	2475	5832	165
1357837223	1353	279	3519.96	748.58	50.44	45.91	17	4	1974	5614	218
1358679387	1371	18	3230.85	289.11	44.19	45.94	17	0	2004	5728	114
1359650278	1486	115	3871.21	640.36	51.39	45.12	18	1	2121	6588	860
1360667327	1527	41	3985.51	114.30	53.67	44.55	18	0	2162	6715	127
1361272800	1551	24	4053.79	68.28	53.89	44.50	18	0	2210	6979	264
1361342930	1573	22	4126.80	73.01	55.11	44.40	18	0	2232	6961	18
1361782346	1697	124	4455.77	328.97	55.61	43.95	19	1	2464	7185	224
1361782348	1813	116	4761.48	305.71	55.78	43.75	20	1	2692	7197	12
1361782356	1819	6	4781.90	20.42	56.11	43.73	20	0	2698	7257	60
1361786874	1841	22	4851.20	69.30	57.33	43.56	20	0	2720	7443	186
1361971024	1858	17	4912.62	61.42	57.89	43.40	20	0	2741	7490	47
1362061911	1888	30	5006.45	93.83	58.78	43.33	20	0	2787	7514	24
1362673655	1872	16	4951.65	54.79	57.89	43.38	20	0	2771	7556	42
1363164832	1893	21	5021.14	69.49	58.61	43.07	20	0	2796	7619	63
1363779418	1893	0	5021.14	0.00	58.61	43.07	20	0	2796	7619	0
1363796837	1913	20	5085.25	64.11	59.33	43.02	20	0	2824	7703	84
1365507551	1931	18	5143.46	58.21	60.33	42.56	20	0	2842	7721	18
1366379836	1945	14	5179.79	36.33	60.33	42.49	20	0	2872	7865	144
1366723611	1965	20	5253.21	73.42	61.83	42.42	20	0	2884	7853	12
1367491328	2005	40	5372.20	118.99	62.89	42.25	20	0	2948	8183	330
1367492036	2017	12	5409.24	37.04	63.17	42.23	20	0	2968	8219	36
1369232619	2023	6	5425.54	16.31	63.50	42.14	20	0	2974	8219	0
1369232646	2059	36	5533.93	108.38	64.33	42.02	20	0	3034	8321	102
1370266650	2103	44	5677.18	143.25	66.39	41.84	20	0	3086	8405	84
1376398539	2097	6	5656.21	20.97	66.06	41.86	20	0	3080	8591	186
1376467846	2103	6	5677.18	20.97	66.39	41.84	20	0	3086	8633	42
1377154381	2169	66	5880.15	202.98	69.67	41.35	20	0	3160	8699	66
1379338175	2127	42	5753.86	126.30	67.33	41.76	20	0	3118	8741	42
1379596412	2203	76	5980.46	226.61	70.39	41.20	20	0	3218	9108	367
1380621927	2233	30	6081.27	100.81	71.67	41.14	20	0	3256	9306	198
1381158560	2197	36	5967.01	114.27	70.06	41.31	20	0	3212	9198	108
1381422145	2231	34	6074.22	107.21	71.56	41.15	20	0	3254	9300	102
1381492557	2219	12	6039.68	34.54	70.89	41.27	20	0	3242	9204	96
1381848999	2235	16	6088.33	48.65	71.78	41.14	20	0	3258	9300	96
1382596072	2235	0	6088.33	0.00	71.78	41.14	20	0	3258	9324	24
1382600014	2255	20	6143.01	54.68	72.50	40.97	20	0	3286	9336	12
1383838666	2233	22	6078.44	64.57	71.67	41.10	20	0	3256	9300	36
1384239499	2351	118	6425.53	347.09	74.33	40.67	20	0	3454	9672	372
1384244021	2251	100	6129.26	296.27	72.67	40.90	20	0	3274	9474	198
1384342565	2389	138	6539.09	409.83	76.44	40.35	20	0	3492	9708	234
1385385041	2279	110	6215.51	323.58	73.83	40.75	20	0	3310	7770	1938
1386690679	2425	146	6672.10	456.59	78.44	40.18	20	0	3528	9718	1948
1388756776	2425	0	6677.03	4.93	78.44	40.24	20	0	3528	10174	456
1389007705	2473	48	6828.65	151.62	79.94	40.12	20	0	3600	10354	180
1389197464	2475	2	6839.03	10.38	80.44	40.09	20	0	3594	10402	48
1389608813	2535	60	7026.52	187.49	82.61	39.88	20	0	3678	10600	198
1392718014	2539	4	7031.57	5.06	82.83	39.78	20	0	3682	10588	12
1396278097	2539	0	7031.89	0.32	83.22	39.73	20	0	3674	10636	48
1400080538	2539	0	7031.89	0.00	83.22	39.73	20	0	3674	10804	168
1401889752	2591	52	7189.94	158.05	84.94	39.50	20	0	3750	10942	138
1410448296	2791	200	7751.81	561.87	86.06	38.27	23	3	4119	11145	203
1412162090	2621	170	7256.50	495.31	85.33	38.45	22	1	3793	11111	34
1412688204	2795	174	7766.01	509.51	86.28	38.27	23	1	4123	11145	34
1424852602	2795	0	7766.01	0.00	86.28	38.27	23	0	4123	11163	18
1425576006	2829	34	7873.26	107.25	88.17	38.06	23	0	4157	11199	36
1425579459	2879	50	8024.62	151.36	88.28	38.05	23	0	4261	11463	264
1425640172	2881	2	8030.37	5.75	88.39	38.03	23	0	4263	11463	0
1426086770	2883	2	8037.48	7.12	88.50	38.02	23	0	4265	11499	36
1430495736	3183	300	8960.72	923.23	92.56	36.61	26	3	4754	11986	487
1432038457	3183	0	8960.72	0.00	92.56	36.61	26	0	4754	11968	18
1432655913	3145	38	8846.56	114.16	92.44	36.61	26	0	4676	11765	203
1433171121	3075	70	8595.42	251.14	89.89	37.01	26	0	4592	11776	11
1433350588	2979	96	8293.10	302.32	88.06	37.12	26	0	4426	11482	294
143377296	2822	157	7835.48	457.62	87.39	37.71	24	2	4143	11451	31
1435164023	2822	0	7835.48	0.00	87.39	37.71	24	0	4143	11469	18
1435340586	2715	107	7467.58	367.90	82.94	38.71	23	1	4029	10965	504
1435592842	2718	3	7481.67	14.10	83.06	38.67	23	0	4032	10984	19

Continued on next page

Table D.11 – Continued from previous page

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1440512008	2820	102	7841.49	359.82	88.72	38.43	23	0	4134	10972	12
1443626973	2771	49	7641.02	200.47	84.67	38.44	23	0	4112	11650	678
1456942487	2791	20	7705.11	64.09	85.78	38.33	23	0	4132	11218	432
1463670046	2797	6	7726.13	21.02	86.11	38.32	23	0	4138	11338	120
1477672466	2765	32	7623.54	102.59	85.22	38.37	23	0	4088	11266	72
1480579833	3201	436	8871.16	1247.62	66.75	45.27	28	5	4895	11577	311
1481285276	3386	185	9427.14	555.98	62.27	48.73	30	2	5248	11636	59
1492167019	3378	8	9399.25	27.89	61.96	48.74	30	0	5240	11636	0
1500551082	3390	12	9421.25	21.99	62.42	47.63	30	0	5252	11726	90
1505139377	3400	10	9451.54	30.30	62.81	47.50	30	0	5262	11762	36
1506503094	3406	6	9472.05	20.51	63.04	47.49	30	0	5268	11762	0
1507216914	3412	6	9491.79	19.74	63.27	47.47	30	0	5274	11906	144
1507298943	3412	0	9491.79	0.00	63.27	47.47	30	0	5274	11906	0
1509109642	3412	0	9491.79	0.00	63.27	47.47	30	0	5274	11906	0
Average	2280.71	65.10	6221.26	197.57	68.08	42.26	21.42	0.35	3373.61	9044.19	215.52

Table D.12: Most important metrics CIF for Component F for each revision

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1357112586	144		1146.20		3.27	71.91	65		145	1348	
1357112589	149	5	1193.80	47.60	3.36	71.84	66	1	150	1400	52
1357112598	156	7	1258.93	65.13	3.49	71.76	67	1	157	1508	108
1357112679	162	6	1314.35	55.42	3.56	71.70	68	1	163	1577	69
1357112685	156	6	1252.65	61.70	3.43	71.77	67	1	157	1519	58
1357112696	388	232	3333.47	2080.82	6.28	70.19	97	30	389	4012	2493
1357112719	386	2	3317.49	15.98	6.25	70.24	97	0	387	3988	24
1357112725	384	2	3293.84	23.66	6.22	70.27	96	1	385	3954	34
1357742150	404	20	3509.76	215.92	7.03	69.86	101	5	405	4097	143
1357837223	381	23	3280.61	229.15	6.79	69.98	92	9	382	3941	156
1358679387	387	6	3345.08	64.47	6.82	69.97	93	1	388	4021	80
1359650278	449	62	3845.33	500.25	7.83	69.20	100	7	450	4466	445
1360667327	457	8	3925.32	80.00	8.02	68.90	100	0	458	4518	52
1361272800	471	14	4059.80	134.47	8.24	68.78	102	2	472	4646	128
1361342930	470	1	4051.75	8.04	8.22	68.79	102	0	471	4640	6
1361782346	485	15	4168.10	116.35	8.20	68.74	105	3	486	4785	145
1361782348	487	2	4176.52	8.42	7.98	69.22	107	2	488	4817	32
1361782356	489	2	4197.07	20.55	8.01	69.22	107	0	490	4829	12
1361786874	499	10	4293.34	96.27	8.15	69.18	107	0	500	4889	60
1361971024	508	9	4372.14	78.81	8.54	69.20	109	2	509	4969	80
1362061911	510	2	4396.45	24.30	8.56	69.18	110	1	511	5003	34
1362673655	504	6	4334.26	62.18	8.56	69.19	109	1	505	4879	124
1363164832	513	9	4410.41	76.15	8.52	69.86	113	4	514	4957	78
1363779418	513	0	4408.24	2.17	8.52	69.85	113	0	514	4957	0
1363796837	517	4	4449.36	41.12	8.57	69.84	113	0	518	4992	35
1365507551	522	5	4492.89	43.53	8.54	70.09	115	2	523	5028	36
1366379836	529	7	4562.91	70.02	8.63	70.07	115	0	530	5092	64
1366723611	528	1	4552.59	10.33	8.62	70.07	115	0	529	5075	17
1367491328	546	18	4739.17	186.59	8.84	70.02	116	1	547	5227	152
1367492036	548	2	4760.01	20.84	8.87	70.02	116	0	549	5250	23
1369232619	548	0	4760.01	0.00	8.87	70.02	116	0	549	5250	0
1369232646	553	5	4833.56	73.55	8.93	70.03	117	1	554	5348	98
1370266650	560	7	4888.64	55.07	8.98	69.97	118	1	561	5399	51
1376398539	569	9	4974.77	86.14	9.10	69.94	118	0	570	5453	54
1376467846	560	9	4888.64	86.14	8.98	69.97	118	0	561	5399	54
1377154381	586	26	5137.50	248.86	9.31	69.86	119	1	587	5588	189
1379338175	560	26	4888.64	248.86	8.98	69.97	118	1	561	5399	189
1379596412	587	27	5144.74	256.10	9.37	69.83	119	1	588	5605	206
1380621927	622	35	5716.63	571.89	12.17	69.87	120	1	623	5922	317
1381158560	595	27	5225.55	491.09	9.46	69.81	119	1	596	5653	269
1381422145	604	9	5324.28	98.73	9.55	69.79	120	1	605	5740	87
1381492557	602	2	5306.98	17.31	9.53	69.80	120	0	603	5728	12
1381848999	605	3	5332.96	25.99	9.56	69.78	120	0	606	5746	18
1382596072	607	2	5354.18	21.21	9.59	69.78	120	0	608	5769	23
1382600014	605	2	5332.96	21.21	9.56	69.78	120	0	606	5757	12
1383838666	606	1	5341.95	8.99	9.56	69.78	120	0	607	5752	5
1384239499	624	18	5511.52	169.57	9.79	69.70	121	1	625	5937	185
1384244021	620	4	5461.87	49.65	9.73	69.71	120	1	621	5836	101
1384342565	623	3	5497.48	35.61	9.77	69.70	121	1	624	5942	106
1385385041	619	4	5459.38	38.09	9.69	69.72	121	0	620	5863	79
1386690679	649	30	5738.86	279.48	10.20	69.55	123	2	650	6101	238
1388756776	648	1	5721.61	17.25	10.19	69.56	122	1	649	6073	28
1389007705	662	14	5877.77	156.16	10.35	69.53	123	1	663	6212	139
1389197464	671	9	5967.74	89.97	10.43	69.51	124	1	672	6299	87
1389608813	681	10	6076.53	108.78	10.55	69.48	125	1	682	6392	93
1392718014	681	0	6062.73	13.79	10.58	69.48	124	1	682	6370	22
1396278097	678	3	6029.40	33.33	10.54	69.48	124	0	679	6341	29

Continued on next page

Table D.12 – Continued from previous page

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1400080538	691	13	6156.73	127.33	10.70	69.45	124	0	692	6419	78
1401889752	699	8	6247.08	90.35	10.80	69.43	125	1	700	6500	81
1410448296	766	67	7162.71	915.63	14.49	69.14	133	8	767	7032	532
1412162090	702	64	6276.99	885.73	10.89	69.07	125	8	703	6524	508
1412688204	724	22	6479.69	202.70	11.05	69.48	133	8	725	6678	154
1424852602	725	1	6490.42	10.73	11.06	69.47	133	0	726	6684	6
1425576006	727	2	6511.89	21.47	11.09	69.47	133	0	728	6696	12
1425579459	738	11	6620.95	109.06	11.22	69.45	133	0	739	6762	66
1425640172	735	3	6588.66	32.30	11.18	69.45	133	0	736	6744	18
1426086770	737	2	6607.91	19.26	11.21	69.44	133	0	738	6756	12
1430495736	788	51	7211.67	603.75	11.88	69.99	150	17	789	7164	408
1432038457	789	1	7222.46	10.79	11.89	69.99	150	0	790	7170	6
1432655913	774	15	7032.08	190.38	11.75	70.00	147	3	775	7014	156
1433171121	751	23	6754.92	277.16	10.75	69.87	143	4	752	6784	230
1433350588	735	16	6586.09	168.83	10.66	69.89	142	1	736	6523	261
1433777296	721	14	6396.44	189.65	10.88	69.26	133	9	722	6458	65
1435164023	723	2	6416.15	19.71	10.89	69.26	133	0	724	6470	12
1435340586	691	32	6093.91	322.24	10.53	69.19	127	6	692	6220	250
1435592842	694	3	6147.08	53.17	10.55	69.15	128	1	695	6245	25
1440512008	699	5	6195.86	48.78	10.61	69.14	128	0	700	6275	30
1443626973	702	3	6231.06	35.20	10.65	69.12	129	1	703	6315	40
1456942487	706	4	6269.43	38.37	10.70	69.11	129	0	707	6339	24
1463670046	706	0	6269.43	0.00	10.70	69.11	129	0	707	6339	0
1477672466	708	2	6273.07	3.63	10.74	69.10	128	1	709	6318	21
1480579833	738	30	6460.78	187.71	9.30	68.67	146	18	740	6700	382
1481285276	757	19	6587.24	126.46	8.83	70.88	155	9	758	6871	171
1492167019	754	3	6584.81	2.43	8.80	71.05	155	0	755	6845	26
1500551082	759	5	6605.50	20.69	8.85	70.73	155	0	760	6875	30
1505139377	761	2	6628.59	23.09	8.86	70.66	156	1	762	6887	12
1506503094	754	7	6593.13	35.46	8.79	71.14	156	0	756	6846	41
1507216914	754	0	6593.13	0.00	8.79	71.14	156	0	756	6846	0
1507298943	754	0	6593.13	0.00	8.79	71.14	156	0	756	6846	0
1509109642	760	6	6618.07	24.94	8.85	70.66	156	0	761	6881	35
Average	593.21	13.57	5220.06	140.90	9.18	69.83	120.28	2.12	594.26	5569.82	124.98

## D.7 Component G

Table D.13: Most important metrics ASD for Component G for each revision

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1377154381	306		860.87		38.38	50.36	4		394	2420	
1379338175	306	0	860.87	0.00	38.38	50.36	4	0	394	2420	0
1379596412	308	2	867.04	6.17	38.63	50.32	4	0	396	2420	0
1413460241	310	2	870.31	3.27	38.88	49.38	4	0	398	2438	18
1443626973	318	8	900.16	29.85	39.88	49.23	4	0	408	2438	0
1449484128	322	4	912.60	12.44	40.38	49.16	4	0	412	2534	96
1449497233	322	0	912.60	0.00	40.38	49.16	4	0	412	2534	0
Average	316	3.2	892.54	10.35	39.63	49.45	4	0	405.2	2472.8	22.8

Table D.14: Most important metrics CIF for Component G for each revision

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1377154381	96		742.23		5.91	65.36	26		97	830	
1379338175	96	0	742.23	0.00	5.91	65.36	26	0	97	830	0
1379596412	96	0	742.23	0.00	5.91	65.36	26	0	97	830	0
1413460241	96	0	742.23	0.00	5.91	65.36	26	0	97	830	0
1443626973	97	1	750.89	8.66	6.04	65.33	26	0	98	836	6
1449484128	122	25	940.59	189.70	6.94	65.65	28	2	124	1077	241
1449497233	124	2	958.51	17.92	7.19	65.59	28	0	126	1089	12
Average	107	5.6	826.89	43.26	6.40	65.46	26.8	0.4	108.4	932.4	51.8

## D.8 Component H

**Table D.15:** *Most important metrics ASD for Component H for each revision*

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1358263057	668		1576.17		24.8	52.57	10		1153	1963	
1360337247	942	274	2330.09	753.92	32.7	49.38	11	1	1651	2431	468
1361272800	973	31	2414.05	83.96	33.6	48.81	11	0	1707	2695	264
1366716021	985	12	2454.29	40.23	34.3	48.60	11	0	1723	2757	62
1370505495	1004	19	2512.53	58.24	35.6	48.11	11	0	1747	2819	62
1376398539	539	465	1243.97	1268.56	28	50.24	11	0	815	1919	900
1377154381	541	2	1253.82	9.85	28.2	50.43	11	0	817	1921	2
1378308156	549	8	1279.78	25.96	28.6	50.42	11	0	829	1983	62
1378998596	564	15	1321.79	42.01	29.5	49.98	11	0	849	2045	62
1379338175	543	21	1258.17	63.62	28.3	50.32	11	0	821	1921	124
1379596412	549	6	1279.78	21.61	28.6	50.42	11	0	829	1983	62
1381158560	564	15	1321.79	42.01	29.5	49.98	11	0	849	2045	62
1382004243	579	15	1363.98	42.19	30.4	49.58	11	0	869	2107	62
1382600014	570	9	1335.50	28.48	29.9	49.64	11	0	857	2045	62
1383062076	585	15	1381.15	45.66	31	49.47	11	0	875	2107	62
1384790897	576	9	1352.67	28.48	30.5	49.53	11	0	863	2087	20
1385385041	581	5	1370.74	18.07	30.6	49.64	11	0	871	2083	4
1386852798	573	8	1345.78	24.96	30.3	49.70	11	0	859	2087	4
1388756776	567	6	1328.61	17.17	29.7	49.82	11	0	853	2045	42
1389197464	557	10	1303.54	25.06	29.7	49.87	11	0	831	2015	30
1389608813	561	4	1315.01	11.46	30.1	49.79	11	0	835	2033	18
1399987407	571	10	1349.08	34.08	32.6	49.82	11	0	845	2069	36
1430901241	577	6	1366.55	17.47	33.2	49.71	11	0	851	2129	60
1433783288	595	18	1407.56	41.01	32	49.37	11	0	887	2195	66
1434555494	570	25	1346.05	61.51	30.3	50.64	11	0	857	2099	96
1435340586	564	6	1328.58	17.47	29.7	50.74	11	0	851	2099	0
1443626973	598	34	1421.73	93.15	31.6	50.34	11	0	903	2165	66
1492167019	598	0	1410.10	11.63	31.6	50.35	11	0	903	2189	24
1506603857	598	0	1410.10	0.00	31.6	50.35	11	0	903	2189	0
Average	629	37.43	1495.96	104.57	30.57	49.92	10.97	0.04	982.86	2145.69	99.36

**Table D.16:** *Most important metrics CIF for Component H for each revision*

Revision	LOC	LOC change	V	Vchange	Difficulty	MI	States	State change	Actions & Events	FP	FP change
1358263057	145		935.34		2.78	61.81	48		147	1592	
1360337247	114	31	809.22	126.12	2.22	68.90	47	1	118	1200	392
1361272800	127	13	935.65	126.43	2.58	68.69	49	2	131	1322	122
1366716021	134	7	989.10	53.44	2.58	69.67	52	3	138	1376	54
1370505495	139	5	1022.59	33.49	2.55	70.65	54	2	143	1407	31
1376398539	141	2	1039.81	17.23	2.60	70.63	54	0	145	1419	12
1377154381	135	6	992.89	46.93	2.60	70.78	52	2	139	1350	69
1378308156	140	5	1026.34	33.46	2.57	71.68	54	2	144	1381	31
1378998596	144	4	1054.70	28.36	2.52	72.58	56	2	148	1405	24
1379338175	135	9	992.89	61.82	2.60	70.78	52	4	139	1350	55
1379596412	140	5	1026.34	33.46	2.57	71.68	54	2	144	1381	31
1381158560	144	4	1054.70	28.36	2.52	72.58	56	2	148	1405	24
1382004243	148	4	1083.29	28.59	2.48	73.41	58	2	152	1429	24
1382600014	143	5	1049.35	33.94	2.50	72.67	56	2	147	1398	31
1383062076	148	5	1083.29	33.94	2.48	73.41	58	2	152	1429	31
1384790897	143	5	1049.35	33.94	2.50	72.67	56	2	147	1398	31
1385385041	149	6	1090.44	41.09	2.48	73.41	58	2	153	1435	37
1386852798	144	5	1056.50	33.94	2.50	72.67	56	2	148	1404	31
1388756776	141	3	1022.05	34.46	2.43	72.70	55	1	145	1364	40
1389197464	141	0	1020.53	1.52	2.41	72.70	55	0	145	1364	0
1389608813	142	1	1029.12	8.58	2.43	72.69	55	0	146	1370	6
1399987407	142	0	1029.12	0.00	2.43	72.69	55	0	146	1370	0
1430901241	145	3	1054.99	25.87	2.50	72.66	55	0	149	1388	18
1433783288	150	5	1108.77	53.78	2.64	72.61	56	1	154	1451	63
1434555494	137	13	1018.29	90.47	2.67	72.03	52	4	141	1328	123
1435340586	137	0	1018.29	0.00	2.67	72.03	52	0	141	1328	0
1443626973	140	3	1053.50	35.20	2.74	71.99	53	1	144	1368	40
1492167019	141	1	1062.34	8.84	2.76	71.98	53	0	145	1374	6
1506603857	141	0	1062.34	0.00	2.76	71.98	53	0	145	1374	0
Average	140.34	5.36	1026.59	37.62	2.55	71.54	53.93	1.46	144.28	1384.83	47.36

## Declaration concerning the TU/e Code of Scientific Conduct for the Master's thesis

I have read the TU/e Code of Scientific Conduct<sup>1</sup>.

I hereby declare that my Master's thesis has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct

Date

01-05-2018

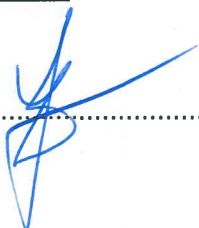
Name

Yorick van der Schriek

ID-number

0737693

Signature



*Submit the signed declaration to the student administration of your department.*

<sup>1</sup> See: <http://www.tue.nl/en/university/about-the-university/integrity/scientific-integrity/>  
The Netherlands Code of Conduct for Academic Practice of the VSNU can be found here also.  
More information about scientific integrity is published on the websites of TU/e and VSNU