

**Object-Oriented Design Quality Adaptation: A System Dynamics Simulation**

by

Asmae Mesbahi El Aouame

A dissertation submitted to the Graduate Faculty of  
Auburn University  
in partial fulfillment of the  
requirements for the Degree of  
Doctor of Philosophy

Auburn, Alabama  
May 5, 2013

Keywords: Design quality adaptation, System dynamics simulation

Copyright 2013 by Asmae Mesbahi El Aouame

Approved by

David A. Umphress, Chair, Associate Professor of Computer Science & Software Engineering  
James H. Cross II, Professor of Computer Science & Software Engineering  
Dean. Hendrix, Associate Professor of Computer Science & Software Engineering  
Lloyd S. Riggs, Professor of Electrical & Computer Engineering

## Abstract

Despite the increasing interest of the software engineering research community in assessing and improving quality at the early phases of the software development lifecycle such as the design phase, less attention is devoted to adapting software quality to changing environment conditions especially at the design stage. Design quality can be significantly impacted when design decisions are modified due to changes in requirements or design strategies. This research sheds light on possible adaptation mechanisms that can effectively mitigate any decrease in object-oriented design quality due to a particular design change. To forecast the impact of design changes and possible adaptations on design quality, a system dynamics simulation is developed in Powersim<sup>®</sup>. The simulation variables are grouped into sub-models and represented by the quality factors of the Quality Model for Object Oriented Design (QMOOD) developed by Bansiya and Davis. Each sub-model simulates the interactions between a specific QMOOD quality attribute and its corresponding design properties as well as design metrics. The simulation is developed by following Pfahl and Ruhe's System Dynamics Model. If after a design change, the simulated design quality decreases below a defined reference value, designers can apply one of the suggested adaptation equations that are extracted from the QMOOD quality attributes equations.

The simulation is validated by applying design changes and their adaptations on real academic designs and computing correlations between the simulated and the real values of each QMOOD quality attribute after adaptation. High correlations are obtained for all the quality

attributes, which shows the effectiveness of the adaptation mechanisms in adjusting design quality.

## Acknowledgments

I am deeply grateful and thankful to God, my source of power and enlightenment. Thank you God for your infinite blessings and opening the right doors in times of happiness and sorrow.

After spending long days and nights working on this dissertation, I would like to thank all the people who touched my life and helped me overcome all the encountered challenges.

I am so thankful for being raised and surrounded by amazing parents and family. Mom, dad, I am dedicating this dissertation to you. Thank you for believing in me and bearing my craziness, sometimes. I hope that I was able to achieve what you taught me and to be your source of pride. I am also dedicating my work to my beautiful sisters Salma and Aya as well as my handsome brother Mohamad. Thank you my dear family. You have all showered me with love and support.

My deep thanks and gratitude go to my major professor and my advisory committee members. Thank you Dr. Umphress for your support and enlightening guidance. Thank you for believing in me and giving me the opportunity to complete my PhD. in good conditions. You are and will be one of my inspiring role models as a professor, a mentor, an advisor, and above all as a caring instructor towards all of his students. I would like also to thank my advisory committee members. Thank you Dr. Cross and Dr. Hendrix for honoring my research with your valuable feedback and support. I am also deeply thankful to Dr. Lloyd Riggs for honoring my research as an outside reader with his valuable feedback and comments. My deep gratitude and thanks go also to Dr. Yilmaz and Dr. Chapman for helping me in the validation of my research.

Above all, I would like to thank all of my PCSE research group fellows and my friends for their support and help. You were all inspiring models to me and allowed me to enjoy my journey in Auburn.

## Table of Contents

Abstract .....	ii
List of Tables .....	viii
List of Illustrations .....	x
Chapter 1: Introduction .....	1
Statement of the problem and anticipated benefits .....	2
Research approach .....	4
Simulation-based Virtual Software Engineering Laboratories (VSEL) .....	4
Chapter 2: Literature Survey .....	9
Simulation techniques and tools .....	9
Software Process Simulation Modeling (SPSM) .....	11
SPSM of design phase .....	21
SPSM of adaptive systems .....	24
The verification and validation of SPSM simulations .....	28
Chapter 3: Simulation development and verification .....	31
Phase 0 of the SDM: Pre-study and research hypotheses definition .....	32
Phase 1 of the SDM: initial model development .....	33
Phase 2 of the SDM: Model enhancement .....	48
Chapter 4: Simulation validation .....	74
Design 1 (D1): Library Information System (LIS) .....	75

Quality attributes correlation results .....	92
Chapter 5: Conclusions and future research work .....	96
Conclusions.....	96
Future research work.....	97
References.....	100
Appendix A: System dynamics concepts.....	113
Appendix B: Simulation source code .....	116
Appendix C: simulation validation on D2-D10 designs .....	166

## List of Tables

Table 1: Discrete-event simulation process activities .....	6
Table 2: SDM role model .....	7
Table 3: Representative publications with their simulation techniques and treated software process topics .....	20
Table 4: Reusability adaptation equations .....	41
Table 5: Classification of reusability's adaptation equations .....	41
Table 6: Flexibility adaptation equations .....	54
Table 7: Classification of flexibility's adaptation equation .....	55
Table 8: Understandability adaptation equations.....	59
Table 9: Classification of understandability's adaptation equation.....	59
Table 10: Functionality adaptation equations .....	61
Table 11: Classification of functionality's adaptation equation .....	62
Table 12: Extendibility adaptation equations .....	64
Table 13: Classification of extendibility's adaptation equation .....	65
Table 14: Effectiveness adaptation equations.....	68
Table 15: Classification of effectiveness's adaptation equation.....	68
Table 16: QMOOD design metrics formula .....	78



Table 17: The initial QMOOD design metrics values for the ten designs .....	91
Table 18: The reference values of the quality attributes for the ten designs .....	92
Table 19: Pearson' r correlation degrees .....	93
Table 20: Correlation computations of understandability .....	94
Table 21: Correlation degrees of the QMOOD quality attributes.....	94
Table 22: Correlation computations of extendibility .....	274
Table 23: Correlation computations of flexibility .....	274
Table 24: Correlation computations of reusability .....	275
Table 25: Correlation computations of functionality.....	275
Table 26: Correlation computations of effectiveness .....	276

## List of Illustrations

Figure 1: Sargent’s Simulation Modeling Process.....	5
Figure 2: SDM process model .....	8
Figure 3: SPSM goals .....	12
Figure 4: Design cycle for self-adaptive systems .....	25
Figure 5: Simulation verification techniques .....	30
Figure 6: Simulation validation techniques .....	30
Figure 7: The hierarchical structure of the QMOOD .....	34
Figure 8: QMOOD quality attributes .....	34
Figure 9: QMOOD quality attributes equations.....	35
Figure 10: Design properties definitions.....	35
Figure 11: Design metrics and their corresponding design properties .....	36
Figure 12: Design metrics definitions.....	37
Figure 13: Reusability design properties and metrics.....	38
Figure 14: Reusability reference simulation model .....	42
Figure 15: A snapshot of the simulation interface .....	44
Figure 16: The reusability quality attribute and the design properties values before applying changes .....	45
Figure 17: Scenario 2 results without adaptation.....	46
Figure 18: Scenario 2 results with adaptation.....	46
Figure 19: Scenario 3 results without adaptation.....	47

Figure 20: Scenario 3 results with adaptation.....	47
Figure 21: PowerSim <sup>®</sup> workspace .....	50
Figure 22: Reusability sub-model.....	53
Figure 23: Flexibility design properties and metrics .....	54
Figure 24: Flexibility sub-model .....	57
Figure 25: Understandability design properties and metrics .....	58
Figure 26: Understandability sub-model .....	60
Figure 27: Functionality design properties and metrics.....	61
Figure 28: Functionality sub-model.....	63
Figure 29: Extendibility design properties and metrics .....	64
Figure 30: Extendibility sub-model .....	66
Figure 31: Effectiveness design properties and metrics .....	67
Figure 32: Effectiveness sub-model.....	69
Figure 33: The welcome page of the simulation.....	70
Figure 34: The input menu of the simulation.....	71
Figure 35: The simulation results page.....	72
Figure 36: The library server classes of LIS .....	76
Figure 37: The database server classes of LIS.....	77
Figure 38: Understandability adaptation results of D1 .....	80
Figure 39: The classes used in D1's understandability design change.....	81
Figure 40: Extendibility adaptation results of D1 .....	83
Figure 41: Flexibility adaptation results of D1 .....	83
Figure 42: The changed parts of D1 for flexibility and extendibility adaptations.....	85

Figure 43: Reusability adaptation results of D1 .....	87
Figure 44: Functionality adaptation results of D1 .....	87
Figure 45: Effectiveness adaptation results of D1 .....	89
Figure 46: The changed parts of D1 for reusability, functionality, and effectiveness adaptation.....	90
Figure 47: Pearson's formula .....	92
Figure 48: A feedback loop that shows the relationships .....	114
Figure 49: Reinforcing feedback loop .....	114
Figure 50: Balancing feedback loop .....	114
Figure 51: level-rate diagram example .....	115
Figure 52: The banker system class diagram .....	168
Figure 53: Understandability adaptation results of D2 .....	169
Figure 54: D2's understandability design change and adaptation .....	170
Figure 55: Flexibility adaptation results of D2 .....	172
Figure 56: Extendibility adaptation results of D2.....	173
Figure 57: Flexibility and extendibility design change and adaptation in D2 .....	174
Figure 58: Functionality adaptation results of D2 .....	175
Figure 59: Reusability adaptation results of D2 .....	176
Figure 60: Functionality and reusability design change and adaptation in D2.....	177
Figure 61: Effectiveness adaptation results of D2 .....	179
Figure 62: The workstation classes of D3.....	180
Figure 63: The server classes of D3.....	180
Figure 64: The monitoring device classes of D3 .....	181
Figure 65: Understandability adaptation results of D3 .....	182

Figure 66: Understandability design change and adaptation in D3 .....	184
Figure 67: Extendibility adaptation results of D3 .....	184
Figure 68: Flexibility adaptation results of D3 .....	185
Figure 69: Flexibility, extendibility, reusability, and functionality design changes with their adaptations in D3 .....	187
Figure 70: Reusability adaptation results of D3 .....	187
Figure 71: Functionality adaptation results of D3 .....	188
Figure 72: D4 class diagram .....	190
Figure 73: Understandability adaptation results of D4 .....	191
Figure 74: Understandability design change and adaptation in D4 .....	192
Figure 75: Understandability design change and adaptation in D4 .....	193
Figure 76: Flexibility adaptation results of D4 .....	194
Figure 77: Extendibility adaptation results of D4 .....	195
Figure 78: Flexibility and extendibility design change and adaptation in D4 .....	196
Figure 79: Flexibility and extendibility design change and adaptation in D4 .....	197
Figure 80: Reusability adaptation results of D4 .....	198
Figure 81: Functionality adaptation results of D4 .....	199
Figure 82: Reusability and functionality design change and adaptation in D4 .....	200
Figure 83: Reusability and functionality design change and adaptation in D4 .....	202
Figure 84: D5 class diagram .....	203
Figure 85: Understandability adaptation results of D5 .....	204
Figure 86: Understandability design change and adaptation in D5 .....	205
Figure 87: Flexibility adaptation results of D5 .....	206
Figure 88: Extendibility adaptation results of D5 .....	207

Figure 89: Extendibility and flexibility design change and adaptation in D5 .....	209
Figure 90: Reusability adaptation results of D5 .....	209
Figure 91: Functionality adaptation results of D5 .....	210
Figure 92: Effectiveness adaptation results of D5 .....	211
Figure 93: Reusability, functionality, and effectiveness design change and adaptation in D5 .....	213
Figure 94: D6 class diagram .....	214
Figure 95: Understandability adaptation results of D6 .....	214
Figure 96: Understandability design change and adaptation in D6 .....	216
Figure 97: Flexibility adaptation results of D6 .....	216
Figure 98: Extendibility adaptation results of D6 .....	217
Figure 99: Extendibility and flexibility design change and adaptation in D6 .....	219
Figure 100: Reusability adaptation results of D6 .....	219
Figure 101: Functionality adaptation results of D6 .....	220
Figure 102: Effectiveness adaptation results of D6 .....	220
Figure 103: Reusability, functionality, and effectiveness design change and adaptation in D6 .....	221
Figure 104: D7 class diagram (1).....	222
Figure 105: D7 class diagram (2).....	223
Figure 106: D7 class diagram (3).....	225
Figure 107: Understandability adaptation results of D7 .....	226
Figure 108: Understandability design change and adaptation in D7 (1) .....	227
Figure 109: Understandability design change and adaptation in D7 (2) .....	228
Figure 110: Understandability design change and adaptation in D7 (3) .....	229

Figure 111: Extendibility adaptation results of D7.....	230
Figure 112: Flexibility adaptation results of D7 .....	231
Figure 113: Extendibility and flexibility design change and adaptation in D7 (1).....	233
Figure 114: Extendibility and flexibility design change and adaptation in D7 (2).....	233
Figure 115: Extendibility and flexibility design change and adaptation in D7 (3).....	234
Figure 116: Reusability adaptation results of D7 .....	234
Figure 117: Functionality adaptation results of D7 .....	235
Figure 118: Effectiveness adaptation results of D7 .....	236
Figure 119: Functionality, reusability, and effectiveness design changes and adaptations in D7 (1).....	237
Figure 120: Functionality, reusability, and effectiveness design changes and adaptations in D7 (2).....	239
Figure 121: Functionality, reusability, and effectiveness design changes and adaptations in D7 (3).....	240
Figure 122: D8 class diagram .....	241
Figure 123: Understandability adaptation results of D8.....	242
Figure 124: Understandability design changes and adaptations of D8.....	243
Figure 125: Extendibility adaptation results of D8.....	244
Figure 126: Extendibility adaptation results of D8.....	245
Figure 127: Extendibility and flexibility design changes and adaptations in D8 .....	246
Figure 128: Reusability adaptation results of D8 .....	243
Figure 129: Functionality adaptation results of D8 .....	244
Figure 130: Reusability and functionality design changes and adaptations in D8 .....	245
Figure 131: Effectiveness adaptation results of D8 .....	247
Figure 132: Effectiveness design change and adaptation in D8 .....	248

Figure 133: D9 Class diagram .....	250
Figure 134: Understandability adaptation results of D9 .....	251
Figure 135: Understandability design change and adaptation in D9 .....	252
Figure 136: Flexibility adaptation results of D9 .....	253
Figure 137: Extendibility adaptation results of D9 .....	254
Figure 138: Flexibility and extendibility design change and adaptation in D9 .....	255
Figure 139: Second extendibility adaptation results of D9 .....	256
Figure 140: Second understandability adaptation results of D9 .....	257
Figure 141: Second extendibility and Understandability design change and adaptation in D9 .....	259
Figure 142: Reusability adaptation results of D9 .....	259
Figure 143: Functionality adaptation results of D9 .....	260
Figure 144: Effectiveness adaptation results of D9 .....	261
Figure 145: Reusability, functionality, and effectiveness design change and adaptation in D9 .....	262
Figure 146: D10 class diagram (1).....	263
Figure 147: D10 class diagram (2).....	264
Figure 148: D10 class diagram (3).....	265
Figure 149: D10 class diagram (4).....	266
Figure 150: D10 class diagram (5).....	267
Figure 151: Understandability adaptation results of D10 .....	267
Figure 152: Understandability design change and adaptation of D10 .....	268
Figure 153: Extendibility adaptation results of D10 .....	269
Figure 154: Flexibility adaptation results of D10 .....	270



Figure 155: Flexibility and extendibility design change and adaptation of D10.....	271
Figure 156: Reusability adaptation results of D10 .....	272
Figure 157: Functionality adaptation results of D10 .....	272
Figure 158: Effectiveness adaptation results of D10 .....	272
Figure 159: Reusability, functionality, and effectiveness design change and adaptation of D10.....	273

## Chapter 1: Introduction

Defining practices that produce high quality software has been the focus of software engineering research for many years. Researchers and practitioners have become conscious of the importance of following a proven process and how it greatly affects product quality. Besides, a myriad of studies (e.g. [Abreu et.al 1996], [Briand et.al 2000], and [Bansiya and Davis 2002]) show that quality improvement in the early phases of the development cycle, such as design, can greatly boost the end product quality. Enhancing design quality by choosing the design alternatives that minimize costs can also be a rewarding investment activity for software organizations [Lazic et.al 2009]. Through design quality evaluation, designers can determine the most profitable design decisions, reduce costs of maintenance, and minimize possible risks of rework. Experience and economical investigations depict numerous benefits of investing on design quality [Sullivan et. al 1998]:

- 1) Increase the flexibility of the whole development process through a well-structured design.
- 2) Cancel unprofitable projects early in their lifecycles.
- 3) Increase the adaptability of design to changing market conditions.
- 4) Prevent possible uncertainties such as failures.

From the literature, we can categorize design quality investment approaches into three groups: analytical/static design quality evaluation, simulation-based design quality evaluation, and design quality adaptation. Extensive design quality research was devoted to the first group

through the definition of quality metrics, quality attributes, and combining them in quality models. Other quality tools depicted the required design activities that determine a specific maturity level of a process such as in CMMI [CMMI 2011]. Further, some standards such as the IEEE standard for developing software life cycle processes suggested a set of desired design activities [IEEE Std 1074-2006]. As a part of software quality assurance activities, technical reviews are also applied in assessing design quality. In the second group of design quality investment options, some studies (see [Xu et.al 2006], [Bogado et. al 2010], [Chiang et.al 2002]) use simulations as a decision support tool that assists designers in evaluating and choosing optimum design strategies. Finally, the quality of design is not only determined from its assessment but also from its ability to adapt to changing environment conditions. This area of research, which represents the third group of quality investment, is still immature despite the vital role of adaptation in stabilizing high quality design.

### **1.1 Statement of the problem and anticipated benefits**

Since design is the blueprint of software, a high quality design is likely to produce a high quality software product. On the one side, solving design problems early in the lifecycle and sustaining high quality values in design attributes is a key success factor in improving not only design quality but also the remaining process phases. Furthermore, it allows software engineers to reduce defect amplification and the number of latent bugs. On the other side, various factors may destabilize design quality. Changing design decisions because of continuous requirements or design strategies alterations may lead to a possible decline in design quality. Other possible reasons of quality change include quality reviews and design flaws. Under such conditions design quality can be restored to its predetermined level at strategic points in the design phase by identifying which design characteristics can be changed as a quality adaptation mechanism.

Although run-time adaptation deals with applying specific quality attribute trade-offs at the implementation phase to regulate a software product quality, it does not take into consideration quality adaptation at design phase. In addition, adaptation strategies are applied at run-time with no beforehand knowledge of their effectiveness or risks. Thus, choosing wrong adaptation strategies or running several adaptations simultaneously can have a conflicting impact and affect a system's performance [Yang et. al 2009]. The main goal of this dissertation is to shed light on OO design quality adaptation through cost-effective and safer techniques such as System Dynamics (SD) simulations. Through design quality adaptation simulation, various stakeholders such as software engineers, software architects, and software quality assurance agents will also be able to benefit from the following additional research goals:

- 1) Extend the use of software process simulation to process lifecycle phases' quality adaptation.
- 2) Save cost: experiment design quality adaptation through simulation instead of costly real time adaptation.
- 3) Save time: a simulation allows us to visualize the impact of changes quicker than in the case of real experimentations.
- 4) Understand the interactions between OO design quality characteristics.
- 5) Depict and test feedback mechanisms for changes in design decisions.
- 6) Perform "what if" analysis of design decisions changes and forecast the needed quality compensations for quality attributes disequilibrium.
- 7) Reduce the cost of new adaptation experimental scenarios through simulation.
- 8) Maximize OO design quality.

## **1.2 Research approach**

### **1.2.1 Simulation-based Virtual Software Engineering Laboratories (VSEL)**

A simulation model is a simplification of a complex system that is hardly understood via analytical methodologies [Muller and Pfahl 2008]. Simulations have been widely employed in a plethora of disciplines such as business case-studies, sociology, physics, biology, and engineering. The application of computer simulations in software process was first initiated by [Abdel-Hamid and Madnick 1991]. Software process simulations can be applied to a specific life cycle phase such as requirements or code testing. It can also model the whole development project as well as multiple simultaneous projects. According to Kellner et.al, those simulations can be used to explore six software process topics namely training and learning, strategic management, process improvement and technology adoption, planning, control and operational management, and understanding [Kellner et. al 1999]. Muller and Pfahl extend Kellner's simulation categories by adding the new trends of process simulation goals such as software acquisition management and COTS, risk management, and product-lines [Muller and Pfahl 2008]. My research is part of software quality adaptation which is a new software process simulation goal that extends Muller and Pfahl's categories. My work is also founded upon the concept of VSEL introduced by [Münch et.al 2005]. VSEL uses simulations to experiment with software process policies or decisions, detect their possible problems, and test their corrective procedures before they are applied in real projects. VSEL can help project managers in finding trade-offs between project duration, needed effort, and product quality. In my research, SD simulation is used to explore both the impact of changing design decisions on design quality and the adaptation strategies that compensate for those quality changes.



Rus et. al defined an SMP for discrete-event simulations based on the generic model of Sargent [Rus et. al 2003]. Their process is not only composed of traditional engineering activities such as model design and implementation but also from managerial activities (see table 1).

<b>Engineering activities</b>	<b>Managerial activities</b>
Requirements identification and specification for the model to be built.	Model development planning.
Analysis and specification of the modeled process.	Model development tracking.
Design of the model.	Measurement of the simulation model.
Implementation of the model.	Measurement of the model development process.
Verification and validation throughout development.	Risk management of risk factors such as changes in customer's requirements and in the description of the modeled process.

**Table1: Discrete-event simulation process activities**

To achieve my research goals, I will apply SD simulation process such as in [Pfahl and Ruhe 2002]. In Pfahl and Ruhe's SMP, System Dynamics Model (SDM) can be produced with the help of four models: phase, role, product, and process.

The phase model defines four main stages in the development of an SD model: pre-study, initial model development, model enhancement, and model application. In the pre-study stage, the simulation modeler identifies the model goals and users. During the initial model development, the behavior of a subset of the system's parameters is illustrated through a reference model to get an idea about the dynamics of the studied software process issues. Then, the reference model is extended to include all system parameters and made ready for problem analysis in the model enhancement stage. Improvement and maintenance of the produced SDM are part of the last stage.

The role model (table 2) identifies the set of stakeholders involved in the development of SDM. According to Pfahl and Ruhe, six actors impact the production process of SDM: Customer

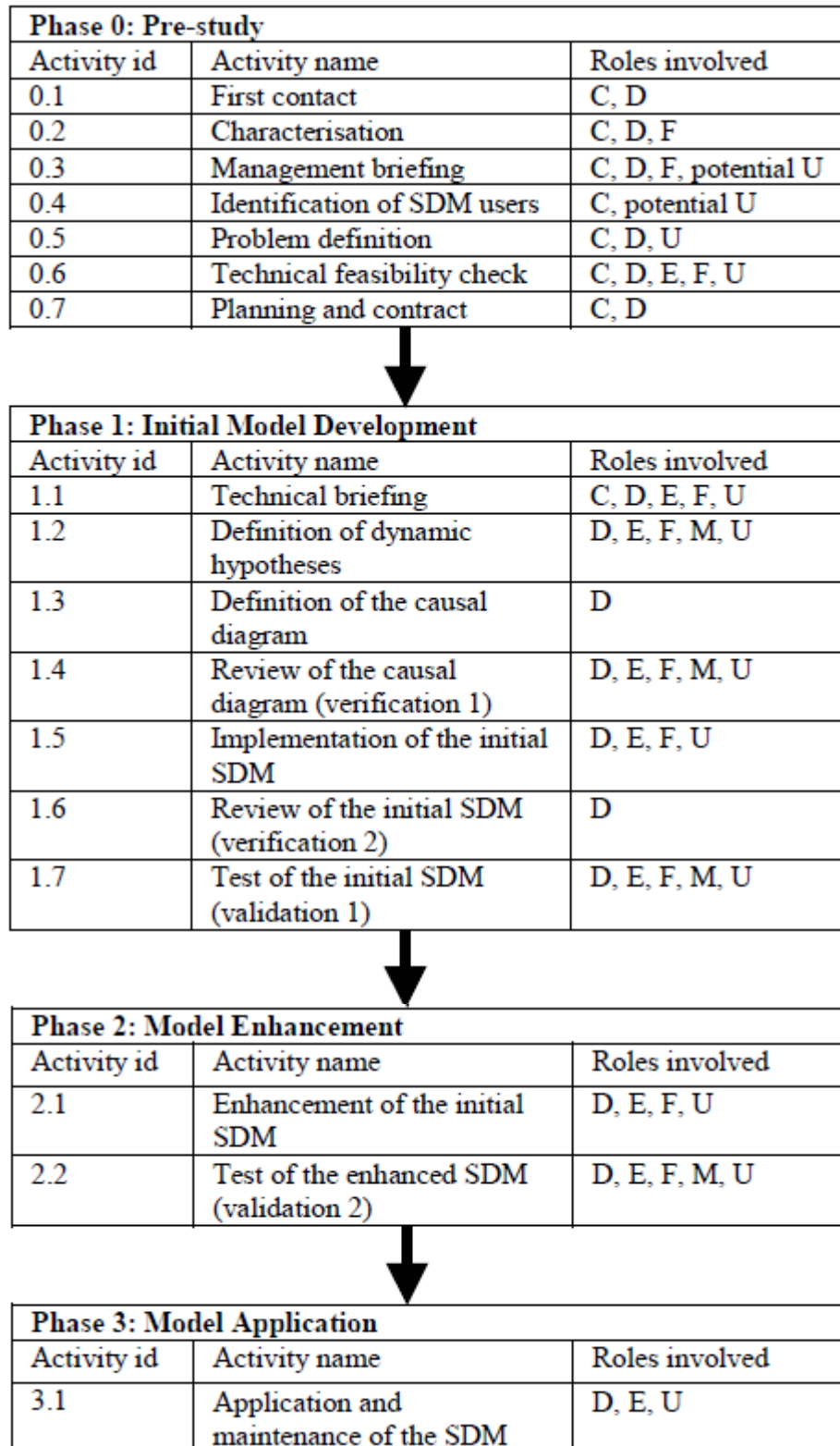
(C), User (U), Developer (D), Facilitator (F), Moderator (M), and SE subject matter Expert (E) [Pfahl and Ruhe 2002].

<b>Actor</b>	<b>C</b>	<b>U</b>	<b>D</b>	<b>F</b>	<b>M</b>	<b>E</b>
<b>Role</b>	Sponsor of the project.	Future user of SDM.	Responsible of producing the SDM.	Plan and arrange project meetings.	Guide workshops and meetings of D with E.	Provide managerial and technical consultancy for SDM production.

**Table 2: SDM role model**

The product model matches SDM process artifacts to their corresponding phases in the phase model (e.g., technical briefing materials and minutes are delivered in the initial model development phase of the phase model). Finally, the process model combines between all of the previous models in a control-flow-oriented scheme (figure 2).





**Figure 2: SDM process model [Pfahl and Ruhe 2002]**

## **Chapter 2: Literature Survey**

Software processes can be modeled by applying several techniques (e.g. the discrete-event (DE) and SD paradigms) to shed light on various process issues such as planning, understanding, and improvement. This led to the emergence of a new field in simulation modeling known as software process simulation modeling (SPSM) devoted to track process issues (table 3).

### **2.1 Simulation techniques and tools**

#### **2.1.1 Simulation techniques for software process**

Simulations are either modeled using deterministic or stochastic paradigms. In the case of deterministic modeling, the simulation runs always lead the same results for given input parameters [Müller and Pfahl 2008]. Stochastic modeling relies on random input parameters, which vary the resulting output from one simulation run to another. Stochastic simulations are also described as static simulations since they track models' variables at a specific point of time. On the other hand, deterministic simulations can be modeled statically such as stochastic simulations or dynamically by tracing a model's behavior over a specific period of time.

There are two types of dynamic simulations: continuous and event-driven. Both groups can be decomposed into quantitative and qualitative techniques. Simulation models' variables are updated at a fixed time step interval in continuous simulations and modified in Event-Driven (ED) simulations when new events occur [Müller and Pfahl 2008]. Quantitative simulation techniques are useful in depicting the complexities in a system's behavior and require enough

historical data or experts' estimation to run the model. Qualitative techniques overcome the lack of historical data and simulate the general simple trend of a system's behavior. Quantitative continuous simulations apply System Dynamics (SD) technique, created by Jay Forrester in 1961 to model a system [Forrester 1961]. SD employs differential equations to describe the cause-effect relationships in the feedback loops of a system [Martin and Raffo 2000] [De Oliveira et.al 2011]. Qualitative Analysis of causal Feedback (QUAF) and Qualitative SIMulation (QSIM) represent the qualitative continuous simulation paradigm [Müller and Pfahl 2008]. Instead of initializing the parameters' simulation with numerical values, QUAF feeds the model with the parameters' relative values and QSIM uses the model's functions polarity (positive or negative) to specify the increase and decrease of variables.

ED simulations can be modeled quantitatively or qualitatively if the simulation's events are based on non-quantitative conditions. The DE technique is one type of ED modeling paradigm that represents a system's activities as a linked set of stations whose statuses change when events are altered [Martin and Raffo 2000]. Other types of ED simulation techniques such as Petri-net, rule-based, state-based and agent-based modeling are employed respectively to provide us with a description of distributed systems, define simulation models as a set of rules, represent the significant events that drive the software process to progress, and show interactions and actions between agents [Huang et.al 2010] [Drappa and Ledwig 1998] [Raffo et. al 1999] [Phillips and Yilmaz 2006].

To benefit from the advantages of more than one simulation technique, hybrid simulations are created by combining the previously described modeling methods. For example, a hybrid simulation can use both deterministic and stochastic techniques. Or, a hybrid simulation can combine the benefits of both continuous and event-driven paradigms. Over all of the

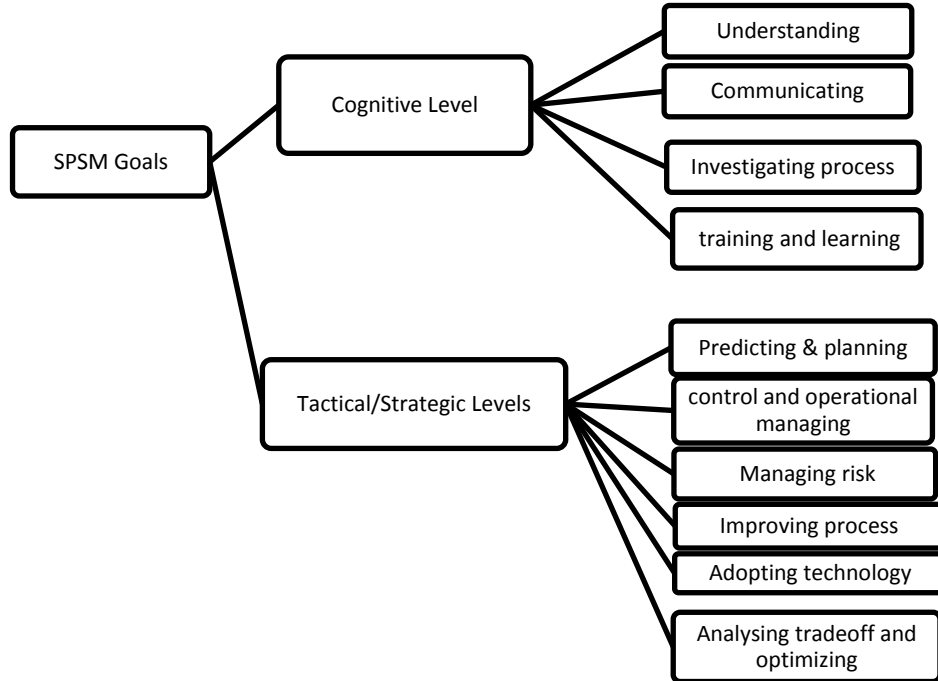
described simulation modeling techniques, SD and DE (49% and 31% of the available Software Process Simulation Modeling (SPSM) research respectively) are the most widely used ones for process simulation [Zhang et.al 2008]. Table 3 presents representative publications with their corresponding modeling techniques and software process coverage.

### **2.1.2 Simulation tools**

There exist several commercial tools that support most of the presented simulation techniques. For example, Stella<sup>®</sup>/iThink<sup>®</sup> specializes in modeling both static and continuous simulations [Stella/iThink 1985]. @RISK<sup>®</sup> supports Monte Carlo stochastic modeling [@RISK 1987]. Extend<sup>®</sup> enables modelers to create both DE and SD simulations through its graphical modeling language ModL and VENSIM<sup>®</sup> produces SD simulations [Extend 1988] [VENSIM 1985].

## **2.2 Software Process Simulation Modeling (SPSM)**

Software processes are complex systems whose description can be effectively simplified through SPSM. One aspect of software processes' complexity is their composite structure made of several interrelated components through cause-effect relationships and feedback loops [Sterman 1992]. Another characteristic of software process complexity is the influence of "soft" qualitative variables such as team motivation on determining the quantitative attributes such as project delivery date and cost. To effectively visualize those complex characteristics, SPSM simulate software processes to understand them at cognitive, and tactical or strategic levels by using the previously described simulation techniques (figure 3) [Zhang et.al 2008].



**Figure 3: SPSM goals**

### 2.2.1 SPSM through SD and continuous simulation techniques

Abdelhamid and Madnick's software projects dynamics research is the seminal application of SD in software process [Abdelhamid and Madnick 1991]. Their SD process model simulates management policies dealing with scheduling, project control, quality assurance, productivity and staffing. The model is divided into 4 sections: human resource management, software production, project planning, and software control.

1) The human resource management: illustrates activities related to developers participating on a software development project. It handles personnel hiring, firing, and transferring to other projects.

2) The software production: allocates available developers to several software development activities such as training, designing, coding, testing, reworking, and quality assurance. It also

handles team motivation, developer's exhaustion, and productivity overhead factors such as communication and rework.

3) The software control: measures software production activities. This section controls overtime work, schedule pressure, and project funding consumption.

4) The software planning: provides initial values for the software project parameters: project size, initial underestimated factor, initial team size, expected conclusion date. It also controls upper management willingness to hire new developers depending on project expected conclusion date.

Abdelhamid's model variables were determined from field interviews, literature review, and peer /expert reviews. The validity of the model was demonstrated by reviewing its behavior when subjected to sensitivity analysis and comparing it with the actual characteristics of real projects. The model can be used, for example, to estimate the optimal quality assurance effort that avoids an increase in testing cost due to a low quality assurance level and minimizes any unnecessary quality expenses.

In their Software Engineering Process Simulation Model (SEPS), Lin and his colleagues adopt the same simulation model structure applied by Abdelhamid and Madnick [Lin et. al 1997]. SD is employed to study the interactions between SEPS sub-models namely production, staff/effort, scheduling, and budget. In addition, project managers can perform what-if –analysis of their managerial policies through SEPS to detect trade-offs of cost, schedule, and functionality.

One of the drawbacks of Abdelhamid and Madnick's software project dynamics model is the huge number of variables that need to be initialized to run the simulation. To overcome this limitation, Ruiz and his colleagues suggest a simplification of the software project dynamics

model by eliminating unnecessary feedback loops in the analysis of the needed behavior, which reduces the number of initialized variables to half the quantity used by Abdelhamid [Ruiz et. al 2001]. Ruiz's model can be used to train project managers in choosing appropriate strategies that reduce a project's development time and cost. The model was verified by reviewing its equations' consistency and validated by comparing its runs' results to a real system's behavior.

Software process simulations can also help project managers in achieving higher CMMI levels [Miller et.al 2002]. To increase the Organizational Process Performance process area of the 4th CMMI maturity level, simulation is a useful analysis environment in detecting appropriate performance measures for process cycle time, product quality, and development time. Similarly, process simulations can improve the Organizational Innovation and Deployment process area of the 5th CMMI maturity level by allowing the project managers to forecast the impact of process changes on process performance such as new staffing policies and decision rules.

Ruiz's simulation model can be used to analyze the impact of each key process area in CMMI level 2 on productivity, product quality and ability to meet deadlines [Araujo et.al 2007]. Productivity is mainly impacted by software quality assurance process areas whereas scheduling is affected by all the CMMI level 2 process areas. Product quality is also mostly impacted by all process areas except the software project planning and software project tracking process areas. Applying CMMI level 2 activities improves product quality by decreasing the number of errors when compared to CMMI level 1.

To study the impact of human resources allocation on the lead time of the project and product quality, an SD simulation of the requirements and test phases was modeled [Andersson et al. 2002]. The simulation runs show that product quality increases when the effort spent in the

requirements phase is increased. In addition, the lead time is decreased thanks to improved specification accuracy when devoting more time to the requirements phase.

SD paradigm is also used by Madachy to track error generation rates, defect amplification between phases, staffing policies, schedule compression, and personnel experience in software processes [Madachy 1996]. The simulation runs show that despite the 10% effort addition in the design and coding phases due to inspections, testing and integration effort is reduced by 50%.

Software evolution process, which is concerned with the adaptation and enhancement of software systems, can be explored through SD simulation to balance progressive and anti-regressive evolution policies [Kahen et. al 2001]. Progressive activities deal with system functionality by adding or modifying code whereas anti-regressive activities cover dead code removal, refactoring, system re-engineering, and system restructuring. The authors' model simulates the consequences of anti-regressive activities on long term system growth. The simulation runs show that assigning 30% of resources to anti-regressive activities results in a significant extension of a system's life span. Moreover, the imbalance between progressive and regressive activities is the main cause behind software evolution productivity.

System dynamics simulations are useful in evaluating the business value of the applied product and process strategies and their corresponding return on investment [Madachy et.al 2006]. By simulating software processes, marketing practices and financial measures over time, both business analysts and software developers can determine the required process activities to meet their business goals.

Continuous simulations can be represented qualitatively by using QSIM, QUAF, and Integrated Measurement, Modeling and Simulation (IMMoS). Unlike quantitative modeling that requires precise data (e.g. SD); qualitative modeling can overcome the lack of accurate variables



quantities by applying abstract techniques such as model functions polarity [Zhang et.al 2009]. This technique allows simulation modelers to represent the increase or decrease in variables through positive or negative polarity instead of specific numbers. IMMoS is another simulation implementation methodology that can resolve the lack of accurate quantitative modeling data. IMMOS combines between SD, static modeling techniques such as Process Modeling (used to identify the variables of a simulation model) and measurement-based Quantitative Modeling (establishes the functional relationships between a simulation's variables) [Pfahl and Ruhe 2002] [Pfahl and Lebsanft 1999].

### **2.2.2 SPSM through DE simulation technique**

One of the goals of developing software process simulations is to train future project managers and improve their managerial skills. Drappa and his colleagues suggest a quality assurance discrete-event simulation system that exposes project managers to real situations and allows them to watch the consequences of their managerial decisions such as changing resource allocation, and skipping requirements specification and reviews on the project's quality [Drappa et. al 1999]. The quality assurance model enables students and novice project managers to test different managerial scenarios for small to medium-size projects. Managers can also plan and control their simulated projects and assign tasks to their virtual developers. The simulation system displays the expected results of the software project and suggests improvements to its management.

To analyze and improve a specific software process phase such as maintenance, Podnar and his colleagues developed a discrete-event simulation based on decision tree representation where entities are Modification Requests (MR) and their corresponding Technical Actions (TA)

[Podnar et.al 2001]. The goal of the simulation is to ensure high quality TAs from maintenance administrators.

Reliability can be simulated in all process phases by using discrete-event or continuous paradigms [Rus et.al 1999]. Besides tracking defects and failures over the entire project, reliability simulations can be used to predict acceptable defect levels at delivery. Furthermore, this type of simulations allows managers to determine the tradeoffs between reliability strategies such as defect detection and prevention techniques based on their impact on cost, staffing, and ability to detect failures.

### **2.2.3 SPSM through hybrid simulation technique**

Since the software process can be represented as a set of DE phases implemented in a continuously changing environment, a hybrid simulation that combines both of those discrete and dynamic aspects is a more realistic representation of software process [Donzelli and Iazeolla 2001b]. Therefore, process activities are modeled by a DE queuing network where activities are described with their interactions and artifacts. The environment is modeled using either an analytical function such as COCOMO or a purely dynamic simulation paradigm such as SD or both of them to illustrate the resources, time, and effort consumed during process activities. The hybrid simulation model enables managers to analyze the impact of their various changing requirements scenarios on effort, delivery time and productivity.

In the Dynamic Capability Model (DCM) introduced by Donzelli and Iazeolla, the effect of process management elements (e.g. reviews) on process quality factors (e.g. effort, delivery time, and productivity) is analyzed by applying a hybrid modeling approach [Donzelli and Iazeolla 2001a]. Since the waterfall process phases are sequential and their corresponding

artifacts are simultaneous, the DE paradigm is employed in the first category and analytical techniques such as COCOMO as well as custom continuous functions are integrated in the second category. DCM allows managers to test their defect detection policies either in the early phases of the software process, in the middle or late in the process lifecycle. Simulation results show that the early detection policy where defects are discovered and corrected in the same injection phase reduces resources consumption, effort and delivery time.

Hybrid simulations can be employed to model different kinds of software projects such as Global Software Development (GSD), geographically distributed developed software. A GSD simulation model illustrates the different interactions between fundamental factors such as communication problems, strategic factors such as distribution overhead, and organizational factors such as team formulation [Setamanit et. al 2006]. Project managers can test the impact of their managerial decisions on effort, quality, and duration of GSD projects. Thus, managers can depict appropriate planning and management tracking policies for offshore sites versus near-shore projects.

<b>Simulation Techniques</b>	SD	DE	Static/ Stochastic	Petri-net	Rule-based	State-based	Hybrid (DE and SD)
<b>Software Process topics</b>							
Planning / Process engineering	[Rus et.al 1999] [Pfahl and Lebsanft 1999] [Powell et.al 1999] [Williford and Chang 1999]			[Bandinelli et.al 1995]		[Kellner 1991]	[Setamanit et. al 2006]

Process improvement & technology adoption	[Araujo et.al 2007] [Abdelhamid and Madnick 1991] [Lin et.al 1997] [Ruiz et.al 2001] [Madachy 1994] [Pfahl and Lefsanft 1999] [Powell et. al 1999] [Andersson et. al 2002] [Donzelli & Iazeolla 2001a] [Madachy 1996] [Madachy et.al 2006]	[Podnar & Mikac 2001] [Ferayorni et. al 2007] [Bogado et.al 2010]	[Chiang and Menzies 2002]			[Raffo 1996] [Raffo et.al 1999]	[Sataman-it et.al 2006] [Donzelli & Iazeolla 2001b] [Martin and Raffo 2000]
Understanding	[Andersson et.al 2002]  [Donzelli & Iazeolla 2001a]  [Madachy 1996]  [Powell et. al 1999] [Wernick and Lehman 1999]				[Drappa and Ludewig 1999]	[Raffo et.al 1999] [Kellner 1991]	[Setaman-it et. al 2006]

Training & learning		[Drappa & Ludewig 1999]			[Drappa et.al 1995] [Drappa and Ludewi-g 1999]		
Project management	[Araujo et.al 2007] [Abdelhamid and Madnick 1991] [Lin et.al 1997] [Ruiz et.al 2001] [Wiliford and Chang 1999]						[Pfahl and Ruhe 1999]
Risk management	[Rus et. al 1999]	[Rus et. al 1999]	[Briand and Pfahl 2000]				
Quality assurance & management	[Ruiz et. al 2001] [Miller et. al 2002] [Araujo et. al 2007] [Andersson et. al 2002]	[Drappa et. al 1999] [Ferayorni at. Al 2007] [Bogado et. al 2010]	[Briand and Pfahl 2000]	[Huang et. al 2010]			
Software maintenance & evolution	[Kahen et.al 2001]						

**Table 3: Representative publications with their simulation techniques and treated software process topics**

Besides modeling the impact of managerial decisions on the overall software project time, effort, quality (errors percentage, errors detection, and rework effort), and size, SPSM simulates also the characteristics of specific software process phases. For example, SPSM can help in determining the design phase quality attributes tradeoffs (e.g. [Chiang et.al 2002]).

### **2.3 SPSM of design phase**

A software system that satisfies non functional requirements early in the design phase is likely to minimize the cost of correcting them late in the software lifecycle. Xu and his colleagues propose a simulation that experiments with design alternatives and how they achieve the non functional requirements by taking into consideration their conflicting, crosscutting, and open-ended nature [Xu et. al 2006]. Non functional requirements can have conflicting goals such as in the case of implementing encryption that reduces a system's responsiveness. They are also crosscutting since integrating security in the system, for instance, require changes in different locations including but not limited to the server and client modules. Furthermore, implementing a specific quality attribute can have an open-ended set of possible solutions (e.g. security can be implemented using authentication only or combined with encryption). Through Xu's simulation, software architects can choose their desired design alternatives that are modeled as state-charts. Then, they can run the simulation and detect design alternatives that satisfy their non-functional requirements.

Resolving the conflicting nature of quality attributes yields high-quality software architecture [Xu 2008]. Xu proposes a Multiple-Objective Decision Analysis (MODA) methodology that relies on multiple-objective decision theory [Xu 2008]. Thus, the conjoint scaling interview scheme from decision theory is applied to gather stakeholders' judgments

about the system's design alternatives and their quality attributes support. A high ranking is assigned to the design alternative that supports most of the required quality attributes. Moreover, value gaps are applied to forecast the most relevant quality attributes for future stages in the design phase. Through a design simulation, stakeholders identify the value gap between the ideal level of a specific quality attribute and its current value. Quality attributes that are characterized by a high value gap are selected as the most important attributes for future design phases.

Simulations can also assist software project managers in determining quality tradeoffs of design decisions in the early phases of software development lifecycle. Chiang and his colleagues present a soft goal simulation that support decision making during design phase [Chiang et.al 2002]. In the case of projects where quality assurance is mandatory, the simulation goal is to identify design alternatives that achieve optimum quality attributes. A soft goal model is composed of three types of soft goals: Non-functional-Requirement (NFR), operationalizing, and claim. NFR soft goals deal with quality requirements such as time performance. Operationalizing soft goals represent possible design alternatives that implement the NFR soft goals (e.g. incorporate java script in an online storefront). Claim soft goals explain the context for a soft goal (e.g., a claim may argue that client-side scripting loads faster).

According to Ferayorni and his colleagues, a design simulation helps architects in improving their architecture's quality and reduces the overall software development effort especially if it integrates domain knowledge by including appropriate design patterns [Ferayorni et.al 2007]. Discrete Event System Specification (DEVS) supports the modeling of complex hierarchical interacting components. The authors extend this modeling and simulation framework to support domain knowledge by adding design patterns suitable to a specific application domain such as Composite, Façade, and Observer patterns. DEVS can also be

employed in simulating software architecture and determining design quality attributes at run time [Bogado et.al 2010]. It decouples the conceptual model of the architecture from the simulator. DEVS simulation allows designers to evaluate performance quality attribute design scenarios.

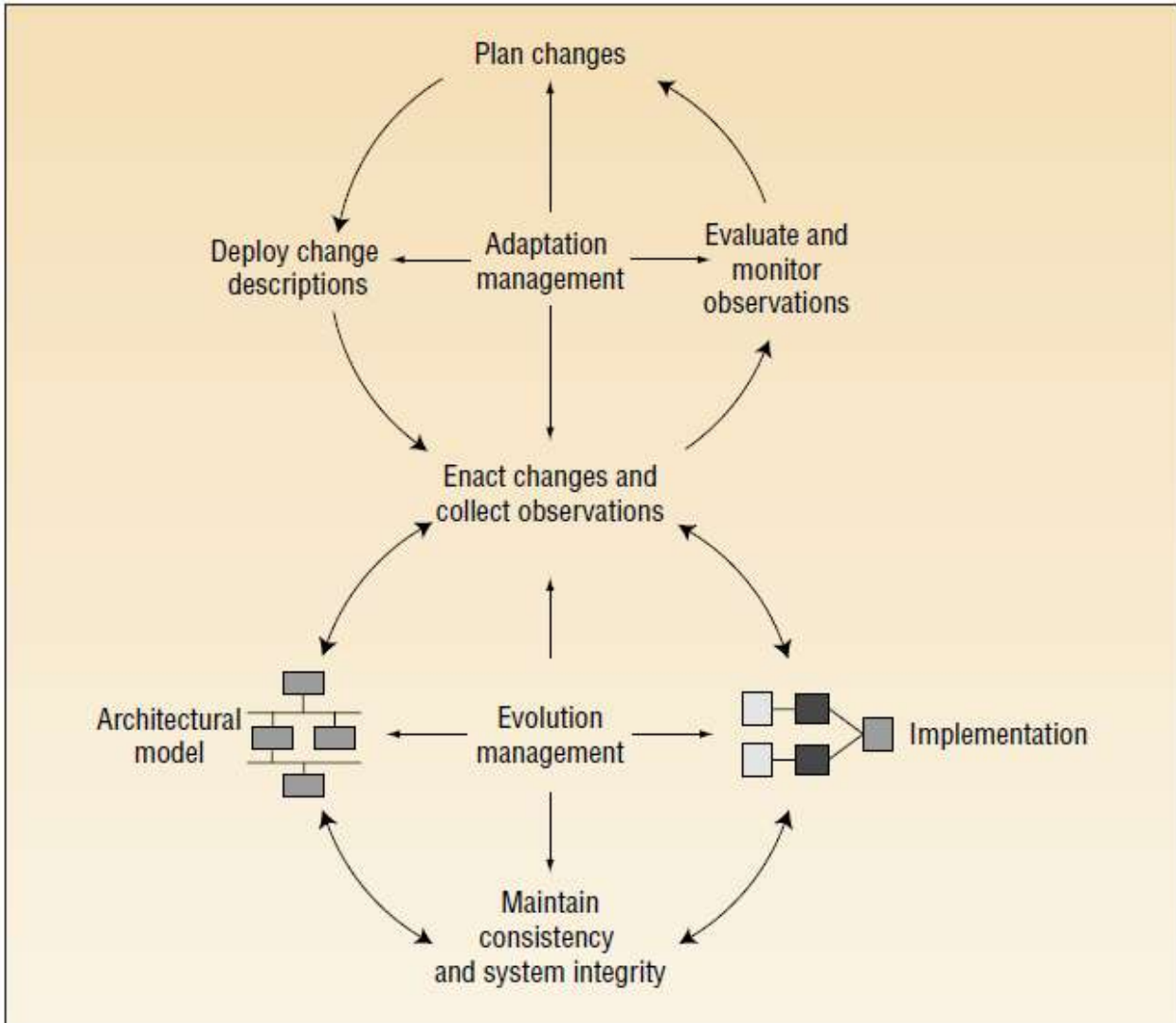
Real-time embedded multiprocessor systems' design is classified as a complex structure due to the interactions between many hardware and software elements. That is why Thuente suggests the combination between rapid simulation and rapid prototyping to depict the optimum design of embedded systems [Thuente 1991]. Rapid simulation is similar to rapid prototyping since it relies on delivering high level simulations characterized by quick development and rapid changes. The initial simulations can be extended as far as rapid prototyping produces additional input for the simulation. Through rapid simulation, the hardware and software components go through iterative refinements based on their performance.

SPSM is also applied in modeling self-adaptive systems and how they adjust to changes in requirements at run-time (e.g. [Yau et.al 2009], [Kumar et.al 2009], and [Beckmann et.al 2009]). An interesting extension of SPSM applications would be to simulate software design's adaptation at design phase instead of leaving it to run-time such as in [Yang et. al 2009]. Simulating adaptation at design stage is likely to minimize risks and ensure an early assessment of adaptation strategies.



## **2.4 SPSM of adaptive systems**

Self-adaptive systems adjust their behaviors to face any changes in their environment such as a decrease in response time, system failures, and new requirements [Oreizy et.al 1999]. Unlike closed adaptive systems, open-adaptive systems apply adaptations during run-time. Conditional expressions are an example of open self-adaptive systems since an application's behavior changes based on the result of the evaluated expression. Oreizy and his colleagues come up with a design cycle for self-adaptive systems [Oreizy et.al 1999]. It is composed of two inter-connected cycles: adaptation management and evolution management (figure 4). The evolution management cycle tracks possible changes to the application either in its architecture or code. Then, those enacted changes are handled by the adaptation management cycle, which plans their possible solutions before deploying them on the system.



**Figure 4: Design cycle for self-adaptive systems [Oreizy et.al 1999]**

In self-adaptive systems, adaptation can be weak if it deals with minor low cost changes such as changing parameters (e.g. bandwidth limit) whereas strong adaptation is concerned with changing, adding/substituting, and removing system artifacts. In addition, self-adaptiveness has several facets such as self-configuring, self-protecting, and self-healing adaptiveness [Salehie and Tahvildari 2009]. To adapt to changes, self-configuring systems can decompose or update system artifacts. Self-protecting systems defend the application against security breaches and self-healing systems repair any dysfunction. According to the authors, self-adaptiveness facets

impact software quality factors. For instance, the self-configuring facet influences the maintainability, the functionality, the portability, and the usability quality factors. Organic computing (OC) systems are self-organized and can self-adapt to any changes in their environment [Schmeck et.al 2010]. Thus, OC systems can maintain a specific robustness level despite the variations in the environment's variables without any external control. Digital evolution is a design methodology where a population of self-replicating computer programs known as digital organisms is subject to mutations and natural selection [Beckmann et.al 2009]. Those digital organisms optimize their resources to survive. Beckmann and his colleagues employ digital evolution in the design of self-adaptive control software for mobile robots. Their approach is composed of four phases: cultivation, translation, simulation, and deployment in order to adapt the system to its environment. Digital Petri dishes where digital organisms develop new computational behaviors to fit their environment are the main components of the cultivation phase. Those evolved programs are translated into the programming language of the target hardware platforms. The simulation and deployment phases are used to evaluate the effectiveness of the evolved digital organisms.

Design alternatives of design decisions can be analytically adapted and traced through a design tree where the leaves are the completed designs and nodes are in-transition designs [Noppen et.al 2011]. This methodology enables software engineers to evaluate design alternatives and choose the ones that best fit the functional and non-functional requirements of the system. On the other hand, a system's performance model can facilitate the dynamic adaptation of software systems to run-time changes in the host and network environments [Kumar et.al 2009]. An example of a performance model for adaptive software processes is the transactional user workload of request/ response such as HTTP workload. Adaptive Service-

Based Software (ASBS) systems identify tradeoffs among conflicting Quality of Service (QoS) aspects and adapt service configurations to satisfy multiple QoS requirements simultaneously through DEVS simulation [Yau et.al 2009]. Those services determine the runtime properties of the service, such as authentication mechanism, priority, and maximum bandwidth. Zhang and his colleagues apply dynamic adaptation to legacy systems by using aspect-oriented paradigm [Zhang et. al 2007]. This approach ensures that adaptation code is separated from legacy code. Then, to enable adaptation in legacy programs, the constructors of the non-adaptive classes are replaced with those of the adaptive classes.

Yang and his colleagues plan software non functional requirements (e.g. performance and availability) adaptation strategies at design phase and apply the appropriate ones at runtime [Yang et. al 2009]. Adaptive strategies can have conflicting effects. A security adaptive strategy that applies encryption may decrease the impact of system responsiveness strategies. Although planning adaptive strategies at design phase without experimenting them does not guarantee their success at run-time, the author's approach is novel in terms of integrating adaptive strategies at design phase.

To maximize SPSM simulations reliability, verification and validation procedures (figures 5 and 6) are a key element in reducing errors and achieving simulations' goals.

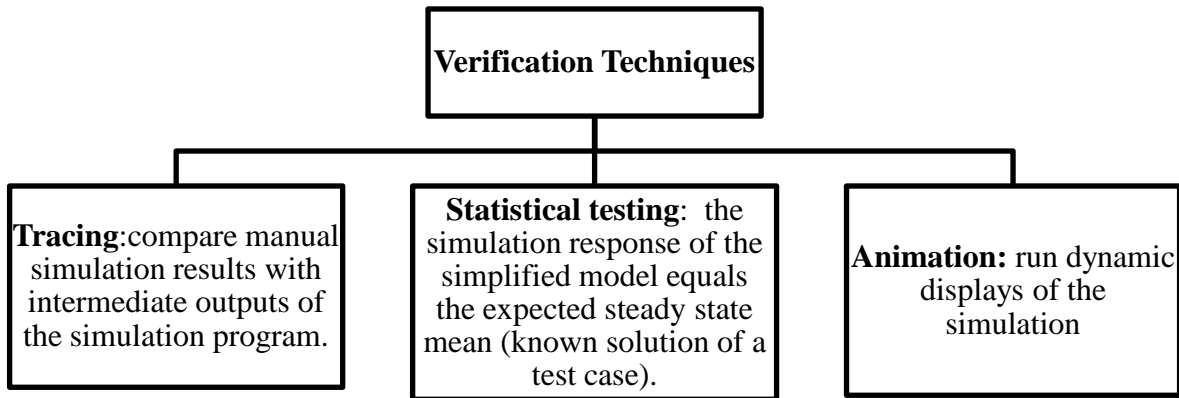
## 2.5 The verification and validation of SPSM simulations

The credibility of a software process simulation model is defined as the level of confidence in its results and is measured through its verification and validation results [Kleijnen 1995]. For example, the verification and validation of simulation models is part of NASA's credibility assessment scale described in the NASA-STD-7009 modeling and simulation standard [Thomas et.al 2011]. Model verification ensures that the simulation program is error-free and works correctly whereas model validation verifies that the implemented model is an accurate representation of the real system and achieves the simulation goals [Sargent 1998]. On the one hand, simulation verification techniques can be grouped into two categories: static testing and dynamic testing. A simulation program can be verified statically by employing walk-throughs, correctness proofs, and examining the structure properties of the program [Sargent 1998]. Code tracing and analyzing execution samples are the main dynamic verification techniques. On the other hand, validation techniques are either subjective relying on experts' judgment or objective applying statistical techniques. In the subjective validation approaches, the model's validity is either determined by the development team, the user of the model, or an independent third party. Besides the objective validation techniques presented by Kleijmen in [Kleijmen 1995], Sargent describes a detailed set of validation techniques such as the historical methods and the multistage validation. There are three types of historical methods: rationalism, empiricism, and positive economics. In the rationalism method, any validity judgments are based on the models accepted assumptions. Empiricism is based on empirically validating the model's assumptions and results. Positive economics assesses the ability of the model to forecast the future. All of those historical approaches are combined in a multistage process that represents the multistage validation technique.

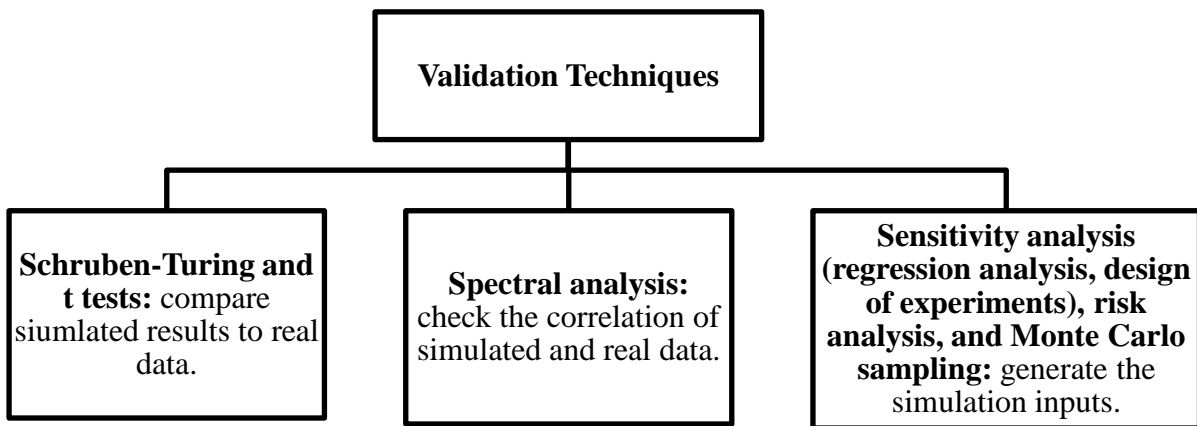
Chwif and his colleagues suggest a verification and validation approach for Discrete-event simulation whose core element is the causal influence matrix [Chwif et.al 2006]. The influence matrix is composed of correlations between a simulation inputs and outputs described as positive, negative, or neutral (e.g. airport check-in desk: higher times between customers' arrivals imply lower waiting times in the queue). Although those relationships are biased by experts' judgments and difficult to track when several input variables are involved the authors' method can overcome the lack of real parameters' values.

Heuristic search algorithms increase the effectiveness of verification and validation by avoiding exhaustive simulation testing and targeting unusual parameters combinations that may lead to exceptions. Scatter/tabu is a heuristic algorithm where the simulation modeler can input her business rules and constraints [Wakeland et.al 2011]. Based on those rules, adequate unusual parameter values, which can be fed on the simulation, are generated.

A set of simulation verification and validation techniques are summarized in figures 5 and 6. To verify and validate software process simulations, each of the described techniques can be used individually or combined with the other procedures. A simulation model's verification and validation are important phases in its development process since they increase the correctness level of the simulation results.



**Figure 5: Simulation verification techniques**



**Figure 6: Simulation validation techniques**

### **Chapter 3: Simulation development and verification**

Object-oriented design quality can be affected by several factors throughout the software development lifecycle, which requires the application of appropriate adaptation strategies that can be tested through SD simulation. Several design quality attributes such as reusability, flexibility and effectiveness can be negatively destabilized by the following possible reasons:

- 1) Changes in system requirements that affect the design structure, such as the addition or the omission of components (e. g: classes, methods).
- 2) Deviating from good design principles such as increasing coupling between classes and producing less cohesive components.
- 3) Changes in a system's implementation such as in timing, storage, and input/output transfers that can lead to major redesign actions [Royce 1970].
- 4) The addition of new functionalities and design modifications in an iterative development process that can lead to changes in design decisions and quality at each iteration.
- 5) Modifications in design decisions issued after design reviews (Preliminary Design Review (PDR) and Critical Design Review (CDR)/ Final Design Review (FDR)).

Since design quality can change over the software development lifecycle time, SD is the appropriate modeling technique. Besides simulating the impact of those destabilizing factors on design quality, the goal of the research is to show how a set of adaptation mechanisms, described in the following sections, can counterbalance any possible quality decrease. The simulation of



OO design quality can be considered as a decision-support tool where software designers can assess the impact of design changes and their corresponding adaptation mechanisms on design quality before applying them on real designs. The simulation is created by following the phases 0-3 (application without maintenance) of the SDM process model (figure 2) and implemented by using the academic version of PowerSim<sup>®</sup> studio. PowerSim<sup>®</sup> is a simulation modeling environment devoted to SD paradigm [PowerSim 1985].

### **3.1 Phase 0 of the SDM: Pre-study and research hypotheses definition**

The main tasks of this phase dealt with identifying the simulation model users and the modeling goal represented by the research hypotheses (tasks ID 0.4 and 0.5 in figure 2).

#### **3.1.1 Simulation model users**

The potential users of the simulation model are the software designers since they are responsible of producing design and integrating any required changes into it.

#### **3.1.2 Research hypotheses**

Besides the research goals defined in chapter 1, the simulation model is used to evaluate the following research hypotheses:

H<sub>0</sub>: Design quality without adaptation mechanisms is the same as design quality with adaptation mechanisms.

H<sub>1</sub>: Design quality with adaptation mechanisms is higher than design quality without adaptation mechanisms.

The goal of the research is to reject the null hypothesis ( $H_0$ ) in favor of the alternative hypothesis ( $H_1$ ).

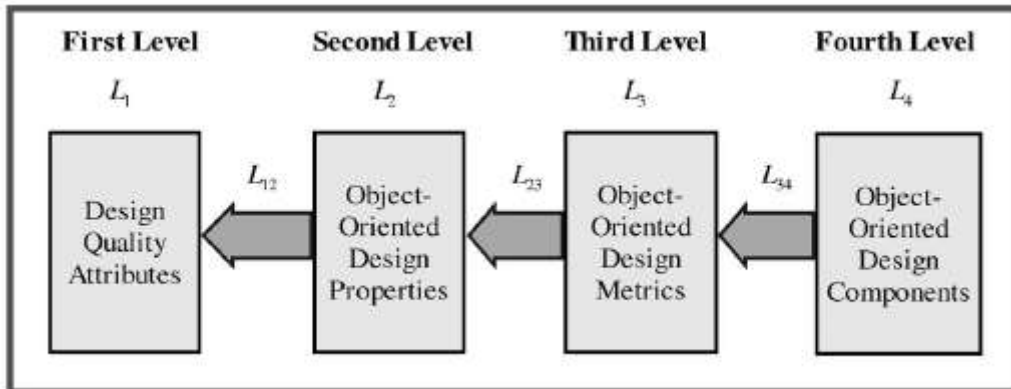
### **3.2 Phase 1 of the SDM: initial model development**

The main product of this phase was a reference simulation model that illustrates the impact of design changes on quality and the mechanisms of quality adaptation. The reference model is the nucleus of the final simulation that is developed in phase 2 of the SDM.

#### **3.2.1 Initial model creation**

By using the SD modeling paradigm (appendix A), OO design was modeled in terms of its quality factors that are part of the hierarchical Quality Model for Object Oriented Design (QMOOD) [Bansiya and Davis 2002]. Unlike McCall et.al, ISO 9126, and Dromey' s quality models, QMOOD (figure 7) establishes clear linkages between the high-level quality attributes (e.g. reusability, flexibility) of a design and its sub-attributes or design properties (e.g. coupling, cohesion) [McCall et.al 1977] [ISO 9126] [Dromey 1996] [Bansiya and Davis 2002]. In addition, QMOOD provides software architects with a set of numerical equations that define the polarity (positive or negative) and the weights of the design properties that characterize each quality attribute (figure 9). The design quality attributes defined in the QMOOD are the main sensors of quality (figure 8). According to Bansiya and Davis, design can be represented by six quality attributes such as extendibility and flexibility that represent QMOOD' s first level (figure 7) (figure 8) [Bansiya and Davis 2002]. Those quality attributes' values are the outcome of specific design properties such as abstraction and polymorphism that are combined in numerical weighted equations based on an extensive review of existing literature and experience (figures 9

and 10) (second level in figure 7). The values of design properties are extracted from design components such as classes and objects through a set of design metrics (figures 11 and 12).



**Figure 7: The hierarchical structure of the QMOOD [Bansiya and Davis 2002]**

Quality Attribute	Definition
Reusability	Reflects the presence of object-oriented design characteristics that allow a design to be reapplied to a new problem without significant effort.
Flexibility	Characteristics that allow the incorporation of changes in a design. The ability of a design to be adapted to provide functionally related capabilities.
Understandability	The properties of the design that enable it to be easily learned and comprehended. This directly relates to the complexity of the design structure.
Functionality	The responsibilities assigned to the classes of a design, which are made available by the classes through their public interfaces.
Extendibility	Refers to the presence and usage of properties in an existing design that allow for the incorporation of new requirements in the design.
Effectiveness	This refers to a design's ability to achieve the desired functionality and behavior using object-oriented design concepts and techniques.

**Figure 8: QMOOD quality attributes [Bansiya and Davis 2002]**

Quality Attribute	Index Computation Equation
Reusability	$-0.25 * \text{Coupling} + 0.25 * \text{Cohesion} + 0.5 * \text{Messaging} + 0.5 * \text{Design Size}$
Flexibility	$0.25 * \text{Encapsulation} - 0.25 * \text{Coupling} + 0.5 * \text{Composition} + 0.5 * \text{Polymorphism}$
Understandability	$-0.33 * \text{Abstraction} + 0.33 * \text{Encapsulation} - 0.33 * \text{Coupling} + 0.33 * \text{Cohesion} - 0.33 * \text{Polymorphism} - 0.33 * \text{Complexity} - 0.33 * \text{Design Size}$
Functionality	$0.12 * \text{Cohesion} + 0.22 * \text{Polymorphism} + 0.22 * \text{Messaging} + 0.22 * \text{Design Size} + 0.22 * \text{Hierarchies}$
Extendibility	$0.5 * \text{Abstraction} - 0.5 * \text{Coupling} + 0.5 * \text{Inheritance} + 0.5 * \text{Polymorphism}$
Effectiveness	$0.2 * \text{Abstraction} + 0.2 * \text{Encapsulation} + 0.2 * \text{Composition} + 0.2 * \text{Inheritance} + 0.2 * \text{Polymorphism}$

Figure 9: QMOOD quality attributes equations [Bansiya and Davis 2002]

Design Property	Definition
Design Size	A measure of the number of classes used in a design.
Hierarchies	Hierarchies are used to represent different generalization-specialization concepts in a design. It is a count of the number of non-inherited classes that have children in a design.
Abstraction	A measure of the generalization-specialization aspect of the design. Classes in a design which have one or more descendants exhibit this property of abstraction.
Encapsulation	Defined as the enclosing of data and behavior within a single construct. In object-oriented designs the property specifically refers to designing classes that prevent access to attribute declarations by defining them to be private, thus protecting the internal representation of the objects.
Coupling	Defines the interdependency of an object on other objects in a design. It is a measure of the number of other objects that would have to be accessed by an object in order for that object to function correctly.
Cohesion	Assesses the relatedness of methods and attributes in a class. Strong overlap in the method parameters and attribute types is an indication of strong cohesion.
Composition	Measures the "part-of," "has," "consists-of," or "part-whole" relationships, which are aggregation relationships in an object-oriented design.
Inheritance	A measure of the "is-a" relationship between classes. This relationship is related to the level of nesting of classes in an inheritance hierarchy.
Polymorphism	The ability to substitute objects whose interfaces match for one another at run-time. It is a measure of services that are dynamically determined at run-time in an object.
Messaging	A count of the number of public methods that are available as services to other classes. This is a measure of the services that a class provides.
Complexity	A measure of the degree of difficulty in understanding and comprehending the internal and external structure of classes and their relationships.

Figure 10: Design properties definitions [Bansiya and Davis 2002]

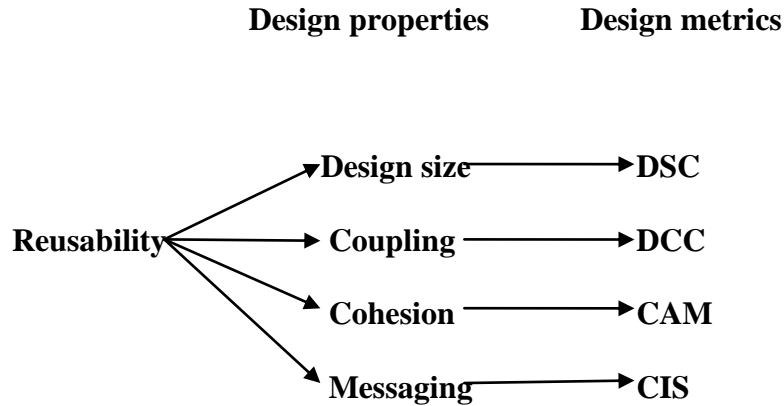
Design Property	Derived Design Metric
Design Size	Design Size in Classes (DSC)
Hierarchies	Number of Hierarchies (NOH)
Abstraction	Average Number of Ancestors (ANA)
Encapsulation	Data Access Metric (DAM)
Coupling	Direct Class Coupling (DCC)
Cohesion	Cohesion Among Methods in Class (CAM)
Composition	Measure of Aggregation (MOA)
Inheritance	Measure of Functional Abstraction (MFA)
Polymorphism	Number of Polymorphic Methods (NOP)
Messaging	Class Interface Size (CIS)
Complexity	Number of Methods (NOM)

**Figure 11: Design metrics and their corresponding design properties [Bansiya and Davis 2002]**

<u>METRIC</u>	<u>NAME</u>	<u>DESCRIPTION</u>
DSC	Design Size in Classes	This metric is a count of the total number of classes in the design.
NOH	Number of Hierarchies	This metric is a count of the number of class hierarchies in the design.
ANA	Average Number of Ancestors	This metric value signifies the average number of classes from which a class inherits information. It is computed by determining the number of classes along all paths from the "root" class(es) to all classes in an inheritance structure.
DAM	Data Access Metric	This metric is the ratio of the number of private (protected) attributes to the total number of attributes declared in the class. A high value for DAM is desired. (Range 0 to 1)
DCC	Direct Class Coupling	This metric is a count of the different number of classes that a class is directly related to. The metric includes classes that are directly related by attribute declarations and message passing (parameters) in methods.
CAM	Cohesion Among Methods of Class	This metric computes the relatedness among methods of a class based upon the parameter list of the methods [3]. The metric is computed using the summation of the intersection of parameters of a method with the maximum independent set of all parameter types in the class. A metric value close to 1.0 is preferred. (Range 0 to 1)
MOA	Measure of Aggregation	This metric measures the extent of the part-whole relationship, realized by using attributes. The metric is a count of the number of data declarations whose types are user defined classes.
MFA	Measure of Functional Abstraction	This metric is the ratio of the number of methods inherited by a class to the total number of methods accessible by member methods of the class. (Range 0 to 1)
NOP	Number of Polymorphic Methods	This metric is a count of the methods that can exhibit polymorphic behavior. Such methods in C++ are marked as virtual.
CIS	Class Interface Size	This metric is a count of the number of public methods in a class
NOM	Number of Methods	This metric is a count of all the methods defined in a class.

**Figure 12: Design metrics definitions [Bansiya and Davis 2002]**

The initial model simulates one quality attribute, namely reusability with its corresponding design properties and metrics (figure 13). The remaining quality attributes are modeled in Phase 2 of the SDM.



**Figure 13: Reusability design properties and metrics**

By applying the principles of the SD paradigm (appendix A), reusability’s initial model was developed in PowerSim<sup>®</sup> (figure 14). The model is applying six types of constants represented as diamonds. The first type of constants holds the values of design metrics such as “DCC”, “CIS”, and “CAM”. The second type of constants represents the needed time to compute each metric such as “DCC\_ ExecuteTime”, “CIS\_ ExecuteTime”, and “CAM\_ ExecuteTime”. The third type of constants represents the possible changes in design metrics such as “CISChange1” and “DSCChange2”. Designers can input up to three possible changes in one simulation run. Those changes are executed at specific points of time in the simulation, which are also represented as diamonds such as “CISTime Change1”, “DSCTimeChange2”, and “CISTimeChange3”. The fifth type of constants, namely “DesignScenarios” receives the chosen

changes scenario by the designer from the simulation interface (figure 15). The last type of constants represents the reference values of quality attributes. The goal of quality adaptation is to counterbalance any decrease in a quality attribute below its reference value. Those optimum quality values are not defined on literature. In this research, reference values are either the initial values of quality attributes before applying any changes or the quality attributes values after an adaptation and before applying a new change. In the initial model, “InitialReusability” is the reference value of reusability.

Other variable types are applied in the initial simulation model to represent the reusability quality attribute, its design properties, the design changes equations, and their corresponding adaptations. Since design changes and adaptations are applied at specific points of time in the simulation run, PowerSim<sup>®</sup>'s time step functions (appendix B) are applied and enclosed in auxiliary clock-like variables such as “Computed DCC” and “Computed CIS” (figure 16). A design change is applied to design properties through their design metrics. Therefore, a design change is computed within the clock-like variable by either increasing or decreasing a specific metric value at a specific point of time (change time) in the simulation (appendix B). To overcome the decrease in reusability after applying a design change, one of reusability's design properties (apart from the changed design property) is increasingly accumulated by applying its corresponding adaptation equation (table 4) through several simulation runs until the reference value is reached. After applying a set of mathematical manipulations to the QMOOD's reusability equation, we came up with its corresponding adaptation equations (table 4). For each design property change, reusability can reach its reference value when the best fit adaptation equation is applied. If the design property of the best fit equation is already at its optimum level in the studied design, an alternate adaptation equation can be applied (table 5). In this case,



reusability is slightly increased above its reference value. This classification of the adaptation equations is obtained empirically from various simulation runs. The initial model's simulation enables designers to experiment specific combinations of changes and adaptations per simulation run as it is described in the verification of the model.

In the initial model, the reusability quality attribute is represented as an auxiliary variable (circle) and fed with its corresponding quality equation from figure 9 whereas the design properties are illustrated as levels (rectangles). The accumulation degree of levels (i.e. increase or decrease) is controlled by rates (valves) such as "change in Messaging". The rates determine the difference between the design property before and after changing it through a differential equation (appendix B) such as in the equation of the rate "change in Messaging":  $\text{'Change in Messaging} = (\text{Computed CIS-Messaging})/\text{CIS\_ExecuteTime}$ '. Any sensed change is sent from the rate to the design property through a quality flow represented by a double-lined arrow. Then, the new design property value is sent to the quality attribute variable to update its equation. If the newly computed reusability's equation is lower than the reference value, the adaptation equations in the clock-like variables are executed. The communication between the other model variables is established through single-lined information arrows (figure 14). A variable that is enclosed within brackets is a shortcut to an already existing variable in the diagram to avoid long awkward links from the source variable (e.g. "Design size" and "Messaging" shortcut variables in figure 14). The "StopCondition" auxiliary variable (circle) compares between the value of reusability after applying an adaptation equation and its reference value ("InitialReusability"). The simulation stops when those variables are equal.

<b>Design property for adaptation</b>	<b>Adaptation equation</b>
Design size	$2 * \text{reusability} + 0.5 * \text{coupling} - 0.5 * \text{cohesion} - \text{messaging}$ .
Messaging	$2 * \text{reusability} + 0.5 * \text{coupling} - 0.5 * \text{cohesion} - \text{design size}$ .
Cohesion	$4 * \text{reusability} + \text{coupling} - 2 * \text{messaging} - 2 * \text{design size}$ .
Coupling	$- 4 * \text{reusability} + \text{cohesion} + 2 * \text{messaging} + 2 * \text{design size}$ .

**Table 4: Reusability adaptation equations**

<b>Design change/ Change in design property</b>	<b>Best fit adaptation equation from table 4</b>	<b>Alternate adaptation equation from table 4</b>
Decrease in messaging.	Cohesion equation.	Design size equation.
Decrease in design size.	Cohesion equation.	Messaging equation.
Decrease in cohesion.	No best fit.	Coupling, messaging, and design size equations.
Increase in coupling.	Cohesion equation.	Messaging and design size equations.
Change more than one design property.	Cohesion equation.	Any design property's equation except the changed one (s).

**Table 5: Classification of reusability's adaptation equations**



### 3.2.2 Initial model verification

The goal of model verification is to show that the initial simulation model is working correctly and that the adaptation equations bring back the value of reusability to its reference value when design changes are applied. The initial model was expanded in Phase 2 of the SDM and its validation is part of the final simulation model validation described in chapter 4.

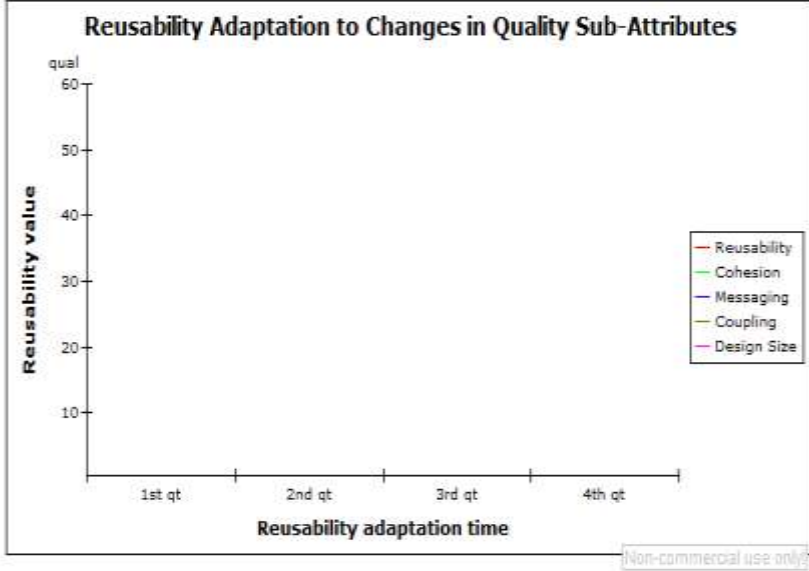
After inputting fictitious design metrics values, design changes and times of changes in the simulation interface (figure 15), the following simulation scenarios were executed:

- 1) Simulate the initial reusability quality attribute before changing any design properties (figure 14), which represents the reference value of reusability.
- 2) Decrease messaging at a specific time in the software development lifecycle and apply the adaptation equation of cohesion (figures 17 and 18).
- 3) Decrease design size and cohesion at a specific time in the software development lifecycle and apply the adaptation equation of messaging (figures 19 and 20).

Unlike the first simulation scenario that illustrates the initial design quality without any changes, the remaining scenarios enable designers to experiment more than one design change at different points of time in the software development lifecycle. In each scenario, the simulation stops when reusability is completely adapted and at least equals the reference value (InitialReusability). In PowerSim<sup>®</sup>, each variable value is characterized by a specific metric. In figure 15, the metric of the quality attributes' values is "qual" (i.e. quality) and the adopted metric for the simulation steps is "da" as an abbreviation of day.



- Design Changes
- Decrease Messaging only
  - Decrease Design Size and Cohesion
  - Initial Design Quality
  - Decrease Messaging without adaptation
  - Decrease Design Size & Cohesion without adaptation



Quality Metrics Non-commercial use only

CIS	30.00 qual
CISChange1	15.00 qual
CISChange2	20.00 qual
CISChange3	25.00 qual
CISTimeChange1	30.00 da
CISTimeChange2	60.00 da
CISTimeChange3	90.00 da

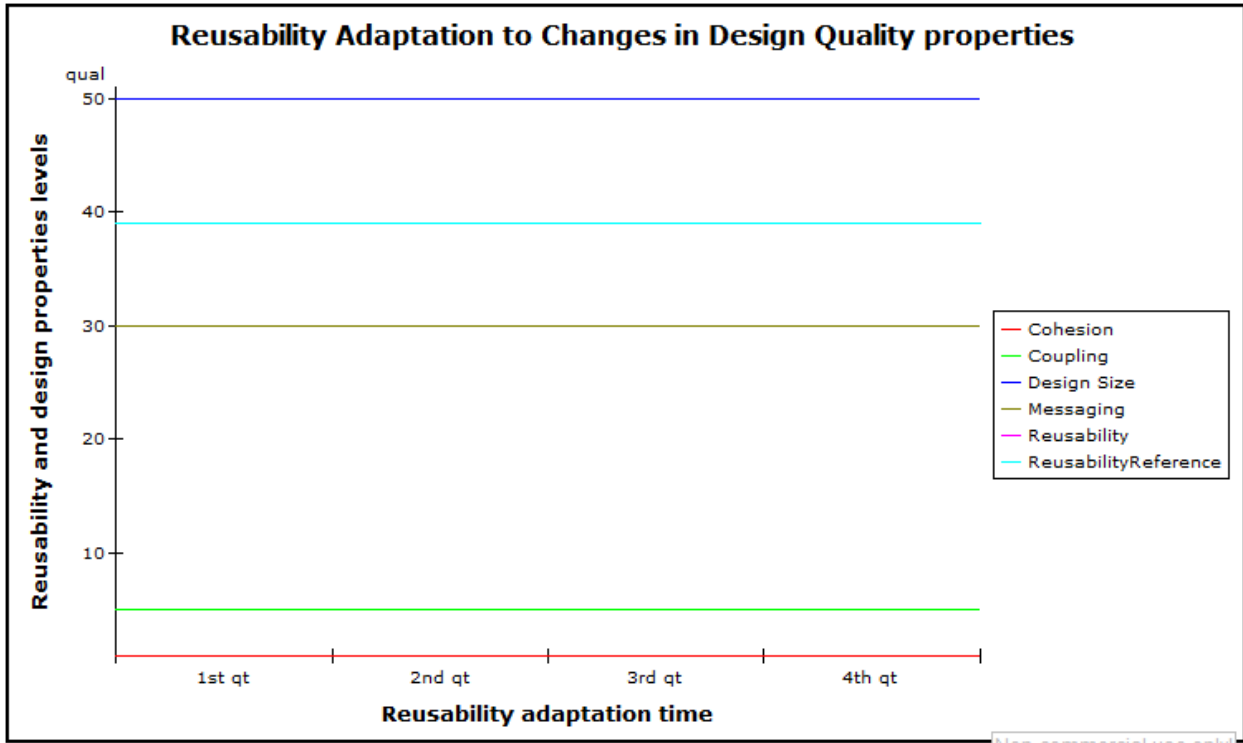
Quality Metrics parameters

CAM	0.90 qual
DSC	30.00 qual
DSCChange1	15.00 qual
DSCChange2	25.00 qual
DSCTimeChange1	40.00 da

**Figure 15: A snapshot of the simulation interface**

The results show the effectiveness of the feedback equations in adjusting the reusability quality level. In the second simulation scenario, if the messaging changes are injected without applying the cohesion adaptation equation, the reusability quality attribute keeps decreasing and never reaches its reference value (the reference value is '38.98' in this example from figure 17). Figure 18 shows that adaptation through cohesion in the three changes of messaging value was sufficient and effective in adjusting reusability to at least its reference value. According to this simulation scenario, cohesion should be increased by a factor of '51' to compensate for the decrease in messaging in order for reusability to reach its reference value. The same observations are depicted in the third simulation scenario (figures 19 and 20). In this particular example,

messaging should be increased by a factor of ‘60’ to compensate for the decrease in design size and cohesion in order for reusability to reach its reference value. Since the change in cohesion does not have a best-fit equation (table 5), adaptation through messaging increases the value of reusability above its reference value.



**Figure 16: The reusability quality attribute and the design properties values before applying changes**

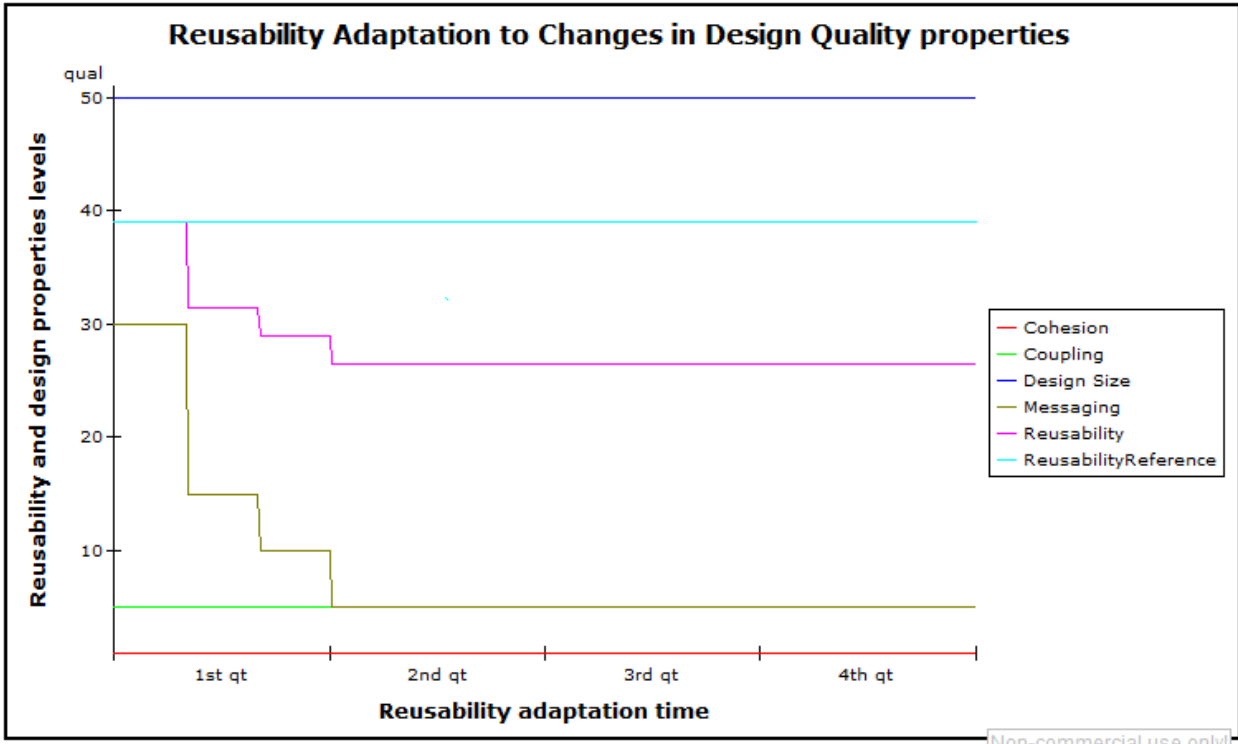


Figure 17: Scenario 2 results without adaptation

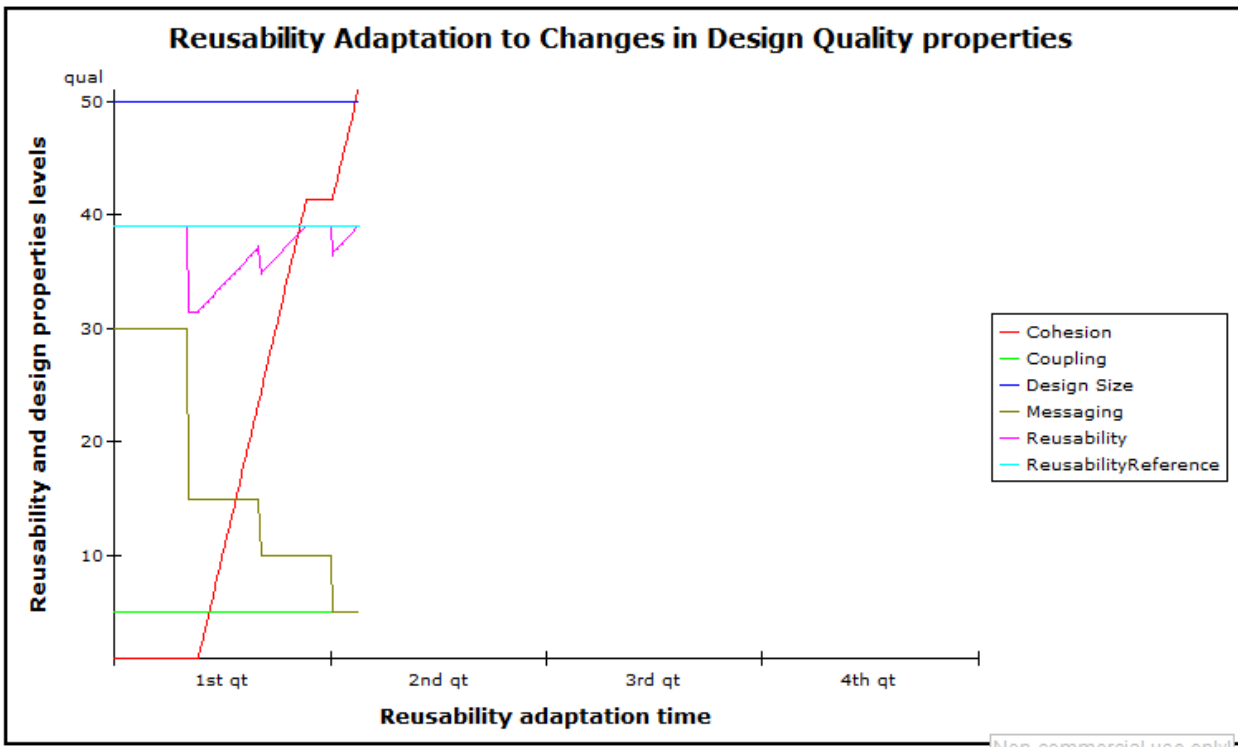


Figure 18: Scenario 2 results with adaptation

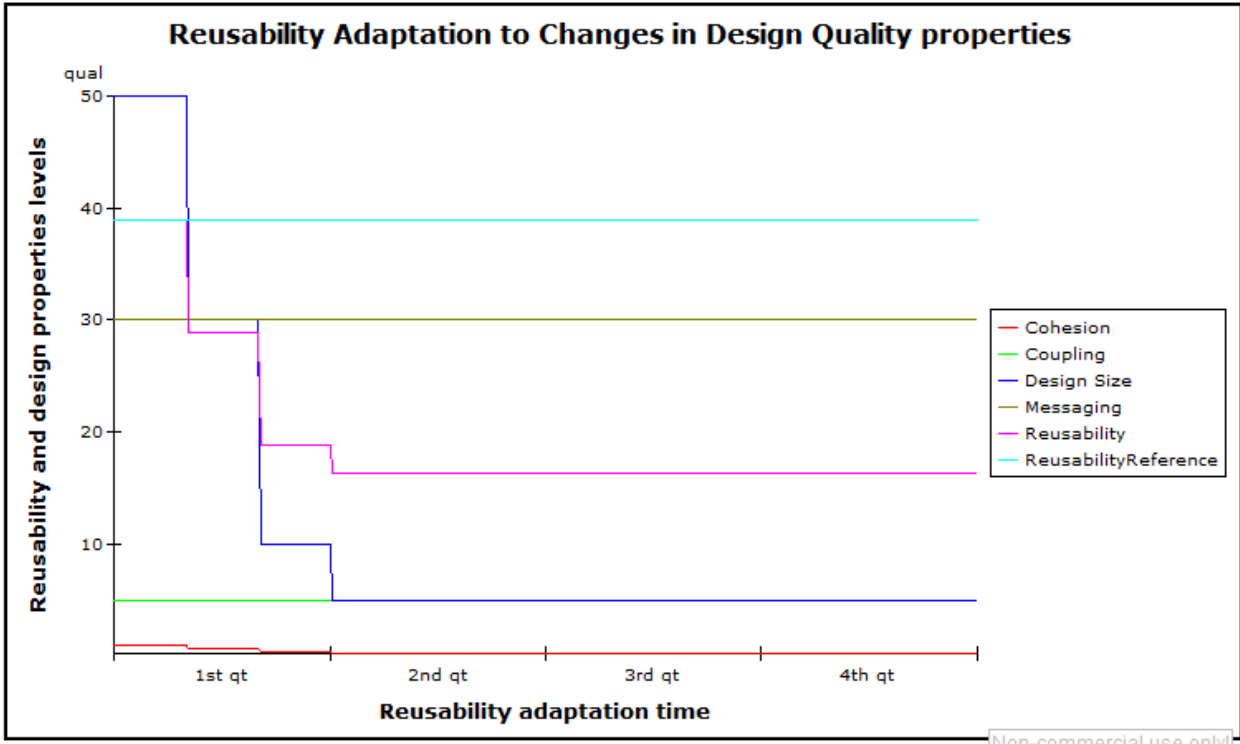


Figure 19: Scenario 3 results without adaptation

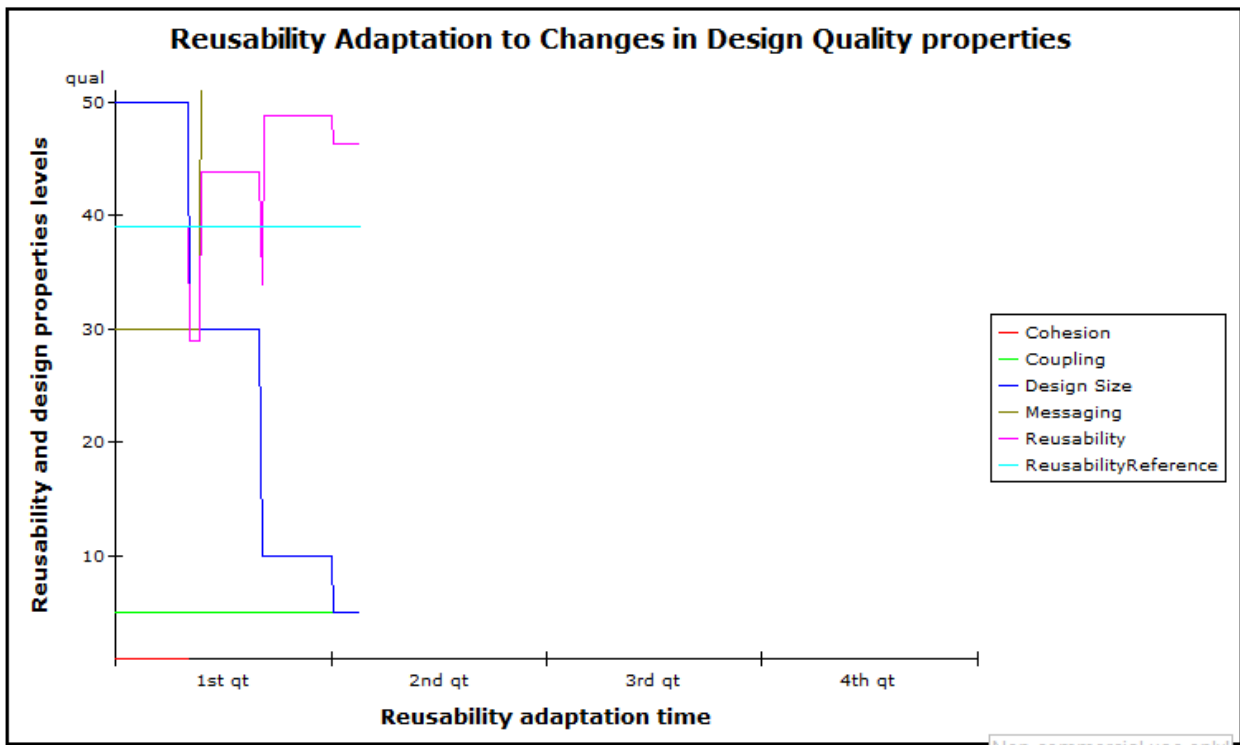


Figure 20: Scenario 3 results with adaptation



### 3.3 Phase 2 of the SDM: Model enhancement

The goal of model enhancement is to improve the initial simulation model and incorporate the remaining QMOOD quality components. The application and validation of the complete simulation model is described in chapter 4.

In the PowerSim<sup>®</sup> workspace (figure 21), OO design quality was modeled as a set of QMOOD quality attributes sub-models. In each sub-model, design quality is described in terms of its quality components and quality flow. In addition, the quality sub-models, as well as their input panel and simulation results can be accessed through a simulation interface that was also produced in the PowerSim<sup>®</sup> workspace (figures 32, 33, and 34).


A set of changes were applied to the initial model and adopted in the remaining quality sub-models including the reusability sub-model. All of the variable types, definitions of reference values, and the classification as well as the identification procedure of the adaptation equations of phase 1 were adopted in phase 2 of the SDM except the following changes and features:

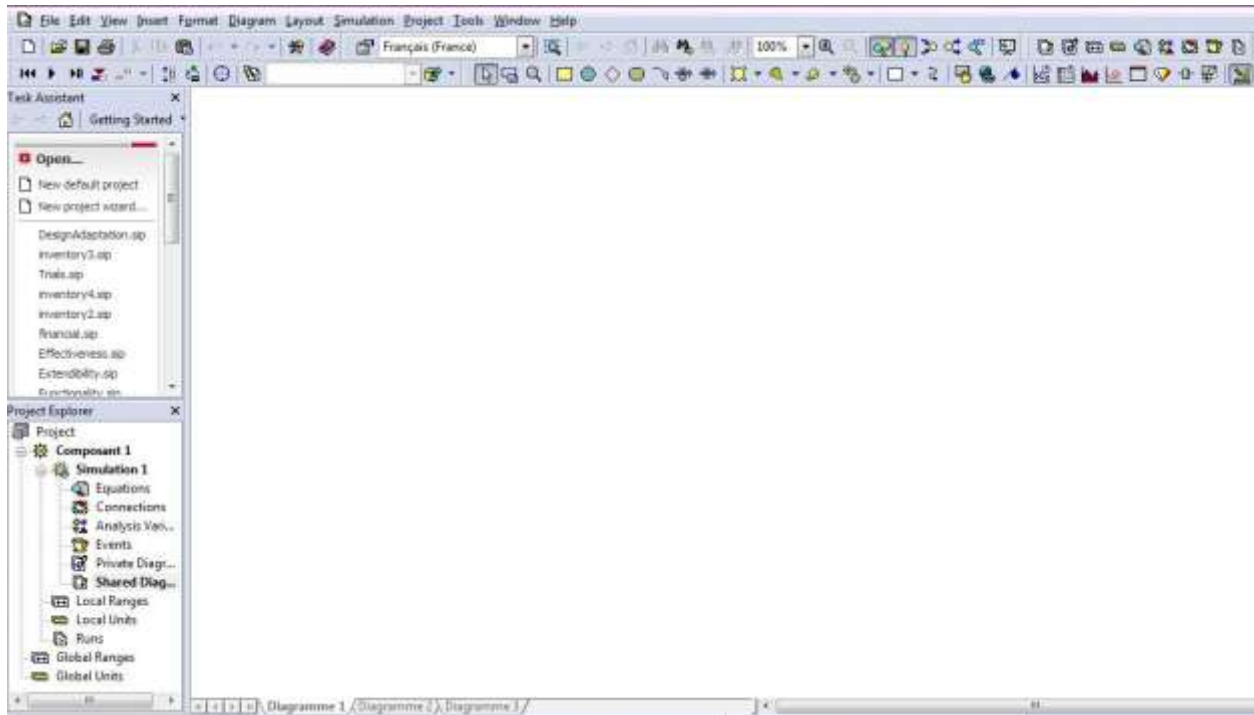
- 1) Delete the “DesignScenarios” variable. Since the goal of creating the initial model is to illustrate the research idea, a limited set of design changes and adaptations were adopted and stored as design scenarios. In the updated version of the reusability sub-model and the other quality sub-models, designers can experiment different combinations of design changes and adaptations per simulation run. The maximum number of design properties that can be changed in each quality attribute per simulation run equals the total number of design properties of that attribute minus 1.
- 2) Omit the “StopCondition” variable. In phase 1, once the adapted quality attribute reaches its reference value after applying all of the design changes, the simulation stops. To prove the

correctness of the simulation results, the enhanced model keeps running even after reaching the reference values. Once the reference value is reached, it never decreases again throughout the simulation run as long as no more design changes are applied.

3) Add new variables linked to each design property in the sub-models such as “DCC-FirstChange” and “CIS- FirstChange” (figure 22). The function of those variables is to keep track of each design property’s last value whether it is equal to the initial quality value, updated after an adaptation, or after all adaptations. This procedure ensures that each design change is applied on the correct value of its corresponding design property.

4) Add new variables linked to each design metric and property in the sub-models such as “DSCInitial” and “CISInitial” (figure22). Design metrics values change from one simulation run to another. To ensure the correctness of the simulation results, those variables keep track of the input metrics in each run.

5) Apply PowerSim<sup>®</sup>’s sliced variables technique. A sliced variable is characterized by distinct aspects in its definition so that each aspect is defined in a specific sub-model. Apart from the complexity design property, each QMOOD design property is shared by more than one quality attribute’s equation. To correctly implement the behavior of the design property that corresponds to each quality attribute, PowerSim<sup>®</sup>’s sliced variables technique is adopted. A variable slice distinguishes itself from an ordinary variable in a sub-model by a slice indicator () in the upper left corner of the variable.



**Figure 21: PowerSim® workspace**

### 3.3.1 Reusability sub-model

#### 3.3.1.1 Reusability quality components

The reusability enhanced sub-model adopts the QMOOD design properties and metrics described in figure 13. It also simulates the output of reusability's adaptation equations and their classifications (tables 4 and 5).

#### 3.3.1.2 Reusability quality flow

After entering the metrics, the combination of changes and the adaptation options of reusability (figure 33), the sub-model assigns those values to their corresponding variables and implements the reusability's appropriate adaptation and quality equations (figure 22). In the QMOOD, each design metric corresponds to a specific design property and the simulation enables designers to change up to eleven design properties. However, the design properties that

impact a specific quality attribute cannot all be changed at once since one of them, at least, should be applied as an adaptation option. In the case of reusability, the maximum number of design properties that can be changed in one simulation run equals 3. Before running the simulation, the designer enters the values of the metrics in the interface such as “DCC”, “CIS”, and “CAM”. He also enters the amounts of changes of each metric such as “DCC\_Change1” and “CAM\_Change2” (figures 22 and 33). Designers can enter up to three changes for each metric in one simulation run. Then, the time of each change is entered and assigned to its corresponding variable such as “DCC\_TimeChange1”, “CIS\_TimeChange2”, and “CAM\_TimeChange3”. Once the simulation runs, the initial quality value of reusability is computed and stored in the “Reusability” auxiliary variable (figure 22 and appendix B). The initial value of reusability is also stored in the “ReusabilityReference” variable (figure 22). If a design change is applied with its corresponding adaptation, the updated values of design properties will be entered by the user in the subsequent simulation run to compute the updated “ReusabilityReference” value.

After computing the initial and the reference values of “Reusability”, the clock-like auxiliary variables such as “Computed CAM” do not sense any change in the value of the design properties. When a given time step of the simulation equals one of the entered change times, the corresponding change amount of a specific metric is applied and stored in its corresponding clock-like variable (e.g. at day 30 of the simulation, the value of CAM is decreased by 10 and stored in the rate “Computed CAM”). That change (either an increase or a decrease) is sent to its corresponding rate such as “Change in cohesion”. The role of the rate is to compute the difference between the value of the design property (e.g. Cohesion) before and after any change. The rate is like a valve that increases or decreases the level value of the design property (e.g. “Cohesion”). The same quality flows are applied in all the level-rate variables in the sub-model

at each time step of the simulation (appendix B). When the design properties of reusability are updated, their values are sent to the “Reusability” variable to re-compute its quality equation. If the new “Reusability” value is lower than the stored “ReusabilityReference”, the corresponding adaptation equations of the checked design properties in the interface are computed in the clock-like variables (appendix B). Adaptation through design properties is implemented in a set of slices that differs from one quality attribute to another. In the case of the reusability sub-model, the adaptation slices are represented by the adaptation equations of coupling, cohesion, messaging, and design size properties that are devoted to the reusability quality attribute (tables 4 and 5). Once the adaptation equations are computed, the new difference in the design properties values is sensed by the rates (“Change in coupling”, “Change in cohesion”, “Change in messaging”, and “Change in Design Size”) and sent again to the level design properties (“Coupling”, “Cohesion”, “Messaging”, and “Design Size”) . Then, the updated values of the design properties are sent to the “Reusability” quality attribute to compute its new adapted value. The simulation keeps running and adapting any decrease in the reusability quality attribute until the end of the simulation time. The detailed implementation of the reusability sub-model is described in appendix B.

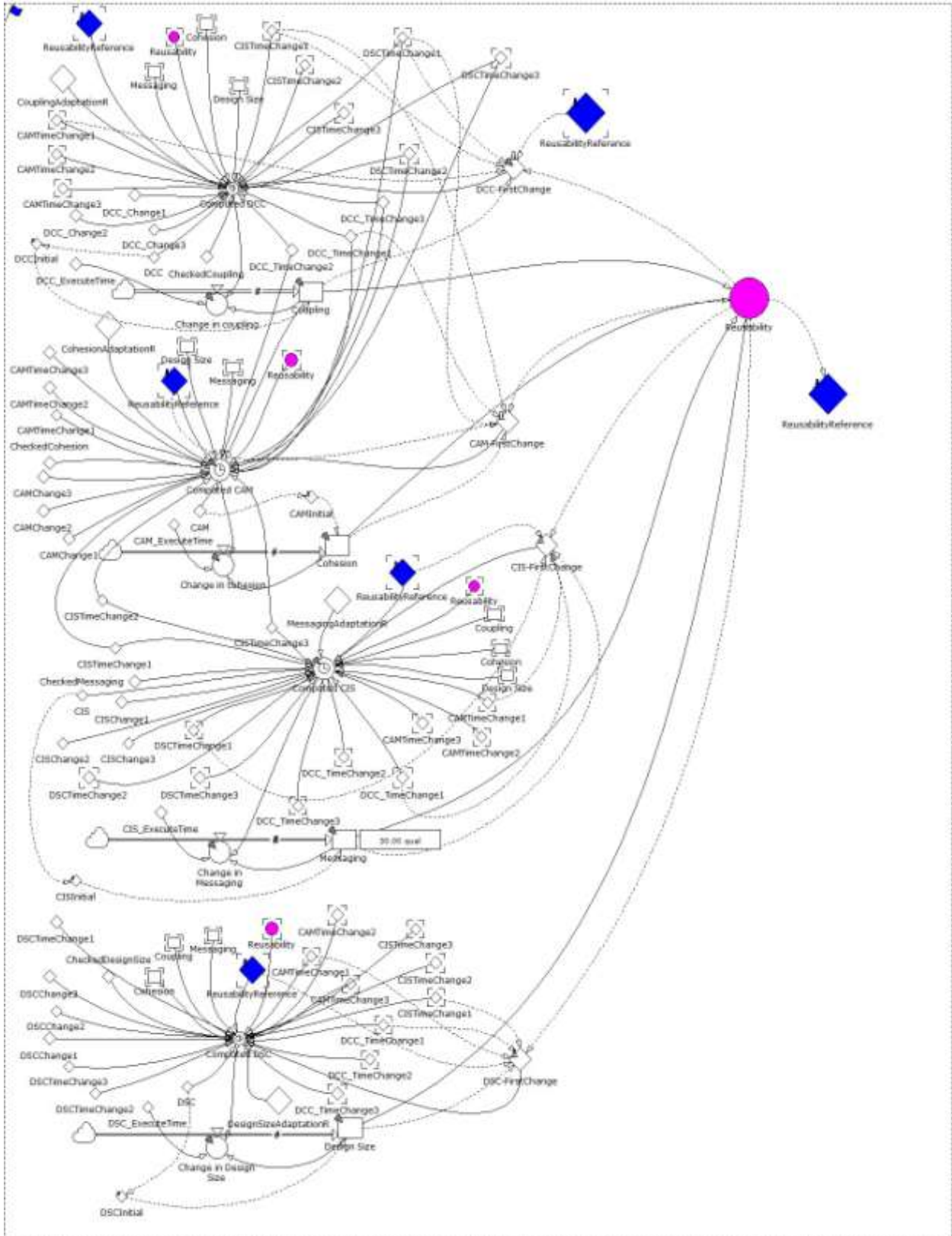
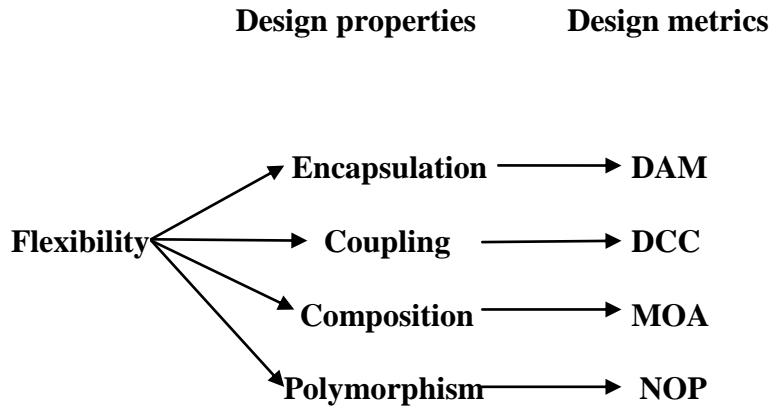


Figure 22: Reusability sub-model

### 3.3.2 Flexibility sub-model

#### 3.3.2.1 Flexibility quality components

In figure 24, flexibility is simulated with its corresponding design properties and metrics (figure 23). After evaluating the values of the design properties, flexibility's quality equation is computed and stored in its variable (figures 9 and 24). In addition, design properties' slices implement flexibility's adaptation equations (tables 6 and 7).



**Figure 23: Flexibility design properties and metrics**

<b>Design property for adaptation</b>	<b>Adaptation equation</b>
Encapsulation	$4 * flexibility + coupling - 2 * composition - 2 * polymorphism.$
Coupling	$- 4 * flexibility + encapsulation + 2 * composition + 2 * polymorphism.$
Composition	$2 * flexibility - 0.5 * encapsulation + 0.5 * coupling - polymorphism.$
Polymorphism	$2 * flexibility - 0.5 * encapsulation + 0.5 * coupling - composition.$

**Table 6: Flexibility adaptation equations**

<b>Design change/ Change in design property</b>	<b>Best fit adaptation equation from table 6</b>	<b>Alternate adaptation equation from table 6</b>
Increase in coupling.	Encapsulation equation.	Composition & polymorphism equations.
Decrease in composition.	Encapsulation equation.	Abstraction & polymorphism equations.
Decrease in polymorphism.	Encapsulation equation.	Composition equation.
Change more than one design property.	Encapsulation equation.	Any design property's equation except the changed one (s).

**Table 7: Classification of flexibility's adaptation equation**

### 3.3.2.2 Flexibility quality flow

In the implementation of the flexibility sub-model (appendix B), the input metrics, the design changes, and the adaptation options are assigned to their corresponding variables (figure 24). In the case of flexibility, the maximum number of design properties that can be changed in one simulation run equals 3. After entering the simulation variables, the initial quality value of flexibility is computed and stored in the "Flexibility" auxiliary variable (figure 24 and appendix B). The initial value of flexibility is also stored in the "FlexibilityReference" variable (figure 24). If a design change is applied with its corresponding adaptation, the updated values of design properties will be entered by the user in the subsequent simulation run to compute the updated "FlexibilityReference" value.

Before running the simulation, the designer enters the values of the metrics in the interface such as "DAM", "DCC", and "MOA". He also enters the amounts of changes of each metric such as "DAM\_Change1" and "MOA\_Change2" (figures 24 & 33). Then, the time of each change is entered and assigned to its corresponding variable such as "DAM\_TimeChange1", "MOA\_TimeChange2", and "DCC\_TimeChange3". Once the simulation runs, the initial quality value of flexibility is computed and stored in the "Flexibility" auxiliary variable (figure 24 and appendix B). The initial value of flexibility is also stored in the



“FlexibilityReference” variable (figure 24). If a design change is applied with its corresponding adaptation, the updated values of design properties will be entered by the user in the subsequent simulation run to compute the updated “FlexibilityReference” value. The same quality flow that was implemented in the reusability sub-model was adopted in the flexibility sub-model and the remaining sub-models of the simulation. In the case of the flexibility sub-model, the adaptation slices are represented by the adaptation equations of coupling, encapsulation, composition, and polymorphism properties that are devoted to the flexibility quality attribute (tables 6 and 7). The detailed implementation of the flexibility sub-model is available in appendix B.



### 3.3.3 Understandability sub-model

#### 3.3.3.1 Understandability quality components

In figure 26, understandability is simulated with its corresponding design properties and metrics (figure 25). After evaluating the values of the design properties, understandability's quality equation is computed and stored in its variable (figures 9 and 26). In addition, design properties' slices implement understandability's adaptation equations (tables 8 and 9).

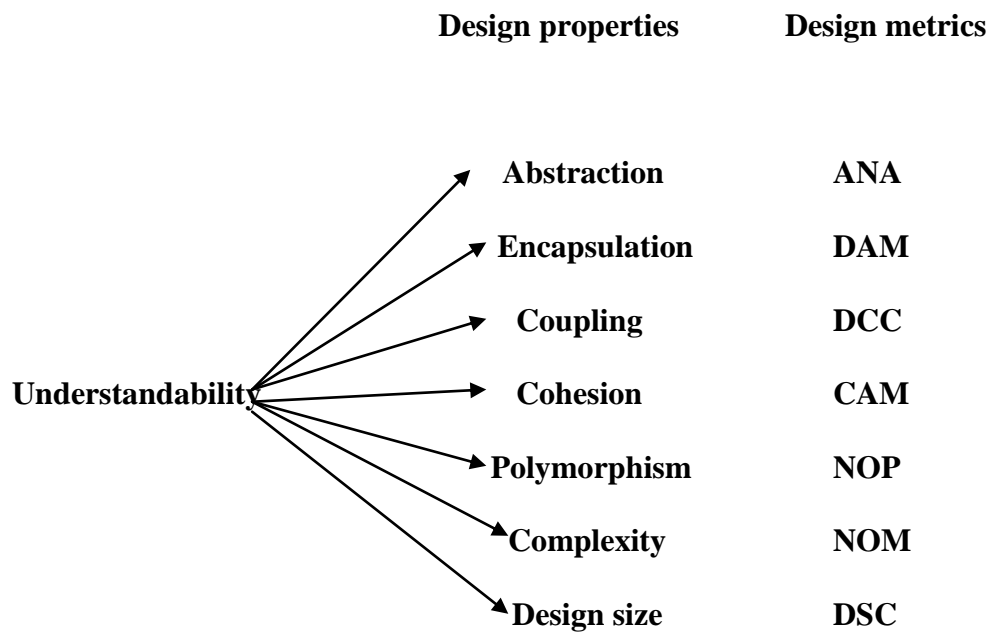


Figure 25: Understandability design properties and metrics

<b>Design property for adaptation</b>	<b>Adaptation equation</b>
Abstraction	-3.03 * understandability + encapsulation - coupling + cohesion – polymorphism – complexity – design size.
Encapsulation	3.03* understandability + abstraction + coupling – cohesion + polymorphism + complexity + design size.
Coupling	-3.03 * understandability – abstraction + encapsulation + cohesion – polymorphism – complexity – design size.
Cohesion	3.03 * understandability + abstraction – encapsulation + coupling + polymorphism + complexity + design size.
Polymorphism	-3.03 * understandability + abstraction + encapsulation – coupling + cohesion – complexity – design size.
Complexity	-3.03 * understandability – abstraction + encapsulation – coupling + cohesion – polymorphism – design size.
Design size	-3.03 * understandability – abstraction + encapsulation – coupling + cohesion - polymorphism – complexity.

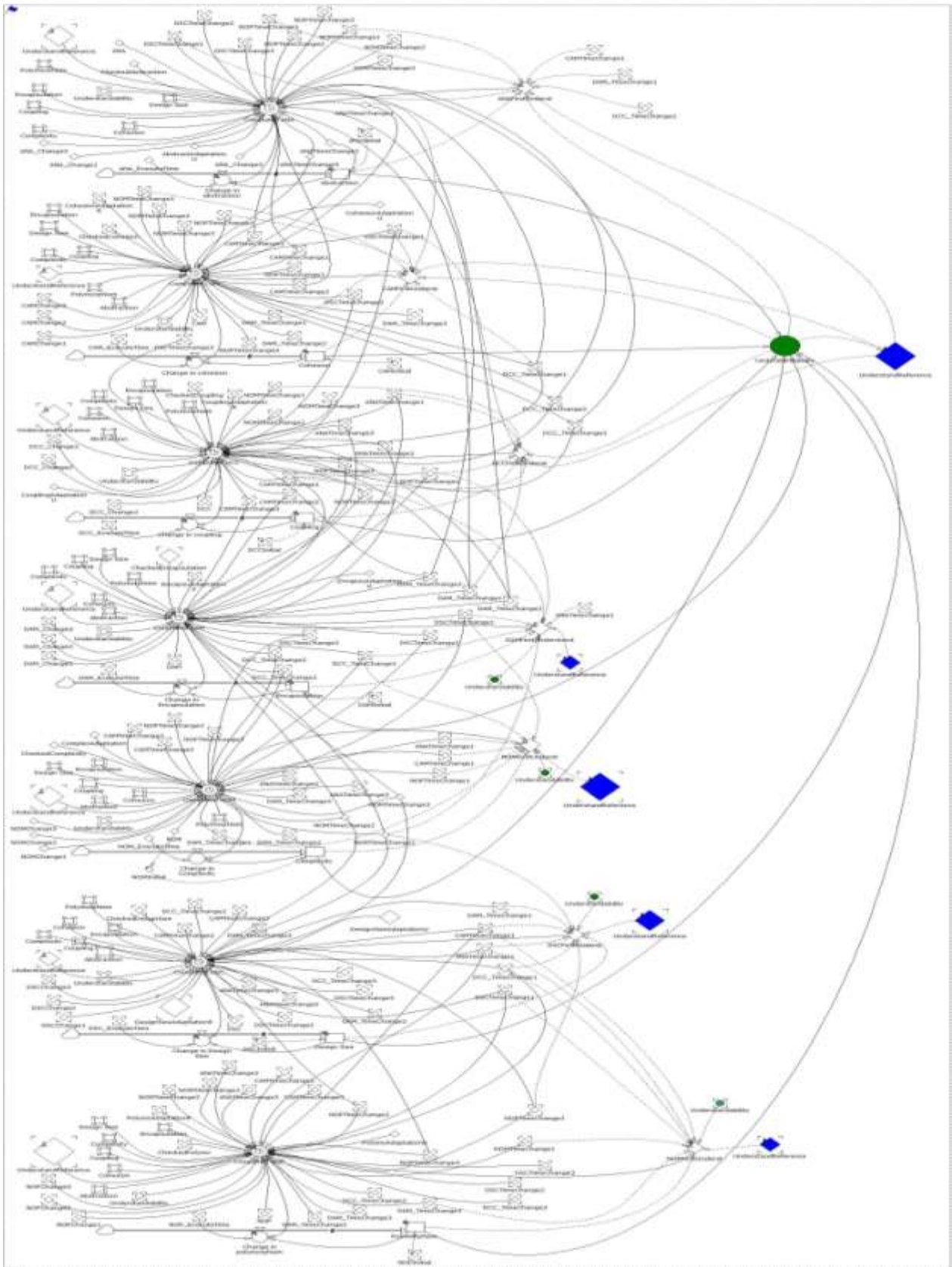
**Table 8: Understandability adaptation equations**

<b>Design change/ Change in design property</b>	<b>Best fit adaptation equation from table 8</b>	<b>Alternate adaptation equation from table 8</b>
Increase in coupling.	Encapsulation equation.	No alternate.
Increase in complexity.	Encapsulation equation.	No alternate.
Increase in design size.	Encapsulation equation.	No alternate.
Change more than one design property.	Encapsulation equation.	Any design property's equation except the changed one (s).

**Table 9: Classification of understandability's adaptation equation**

### 3.3.3.2 Understandability quality flow

On the one hand, the understandability sub-model implements the same quality flows described in the previous sub-models (figure 26). On the other hand, the maximum number of design properties that can be changed in one simulation run equals 6. In addition, the adaptation slices are represented by the adaptation equations of abstraction, cohesion, coupling, encapsulation, complexity, design size, and polymorphism properties that are devoted to the understandability quality attribute (tables 8 and 9). Appendix B describes the detailed implementation of the understandability sub-model.

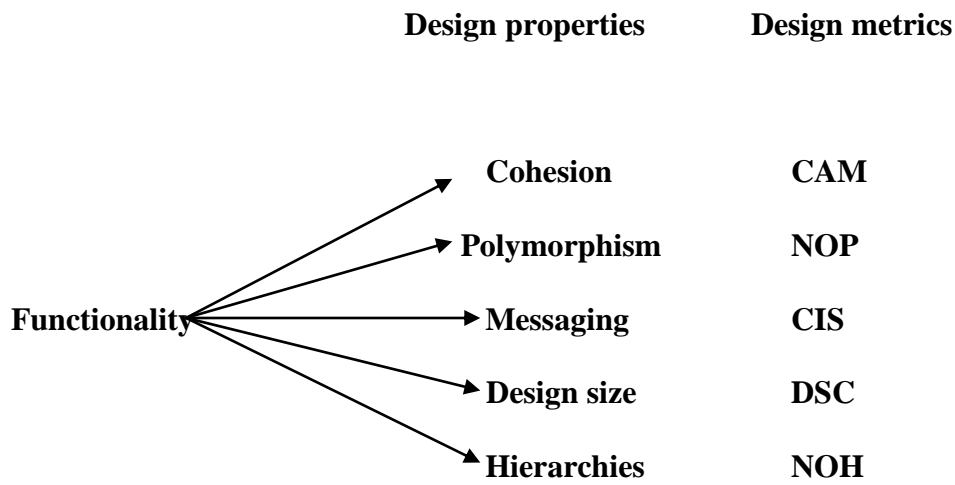


**Figure 26: Understandability sub-model**

### 3.3.4. Functionality sub-model

#### 3.3.4.1. Functionality quality components

In figure 28, functionality is simulated with its corresponding design properties and metrics (figure 25). After evaluating the values of the design properties, functionality's quality equation is computed and stored in its variable (figures 9 and 28). In addition, design properties' slices implement functionality's adaptation equations (tables 10 and 11).



**Figure 27: Functionality design properties and metrics**

<b>Design property for adaptation</b>	<b>Adaptation equation</b>
Cohesion	$8.33 * \text{functionality} - 1.83 * \text{polymorphism} - 1.83 * \text{messaging} - 1.83 * \text{design size} - 1.83 * \text{hierarchies}.$
Polymorphism	$4.54 * \text{functionality} - 0.54 * \text{cohesion} - \text{messaging} - \text{design size} - \text{hierarchies}.$
Messaging	$4.54 * \text{functionality} - 0.54 * \text{cohesion} - \text{polymorphism} - \text{design size} - \text{hierarchies}.$
Design size	$4.54 * \text{functionality} - 0.54 * \text{cohesion} - \text{polymorphism} - \text{messaging} - \text{hierarchies}.$
Hierarchies	$4.54 * \text{functionality} - 0.54 * \text{cohesion} - \text{polymorphism} - \text{messaging} - \text{design size}.$

**Table 10: Functionality adaptation equations**

<b>Design change/ Change in design property</b>	<b>Best fit adaptation equation from table 10</b>	<b>Alternate adaptation equation from table 10</b>
Decrease in polymorphism.	Cohesion equation.	Messaging, design size, and hierarchies equations.
Decrease in messaging.	Cohesion equation.	Polymorphism, design size, and hierarchies equations.
Decrease in design size.	Cohesion equation.	Polymorphism, messaging, and hierarchies equations.
Change more than one design property.	Cohesion equation.	Any design property's equation except the changed one (s).

**Table 11: Classification of functionality's adaptation equation**

### 3.3.4.2 Functionality quality flow

Like the previous sub-models, functionality implements the same quality characteristics (figure 28). However, the maximum number of design properties that can be changed in one simulation run equals 4. Moreover, the adaptation slices are represented by the adaptation equations of messaging, cohesion, design size, hierarchies, and polymorphism properties that are devoted to the functionality quality attribute (tables 10 and 11). The detailed implementation of the functionality sub-model is available in appendix B.

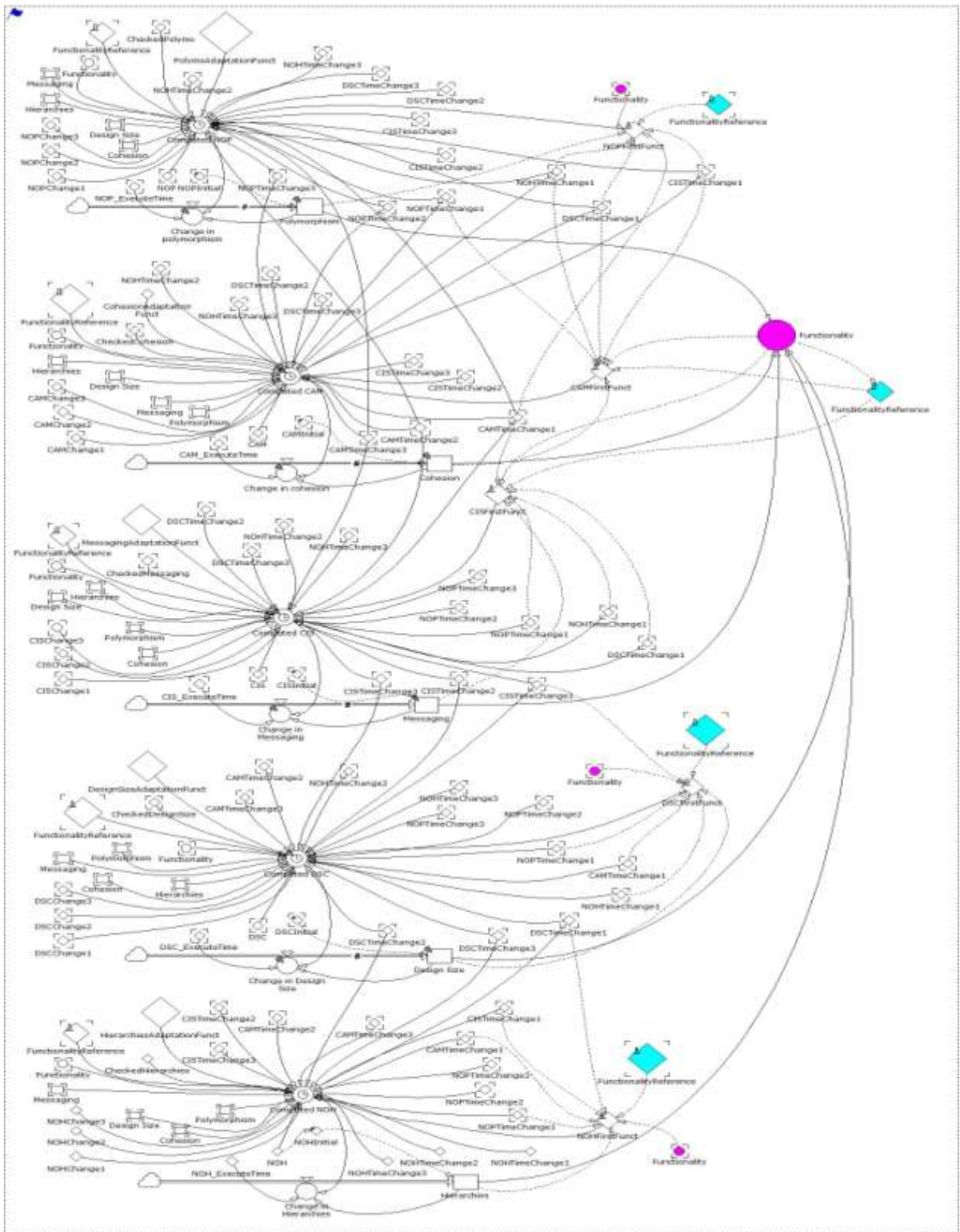


Figure 28: Functionality sub-model



### 3.3.5 Extendibility sub-model

#### 3.3.5.1 Extendibility quality components

In figure 29, extendibility is simulated with its corresponding design properties and metrics (figure 29). After evaluating the values of the design properties, extendibility's quality equation is computed and stored in its variable (figures 9 and 29). In addition, design properties' slices implement extendibility's adaptation equations (tables 12 and 13). For each design change in figure 13, all of the possible adaptation equations of extendibility were experimented. However, no adaptation equation made extendibility equal to its reference value (i.e. no best fit).

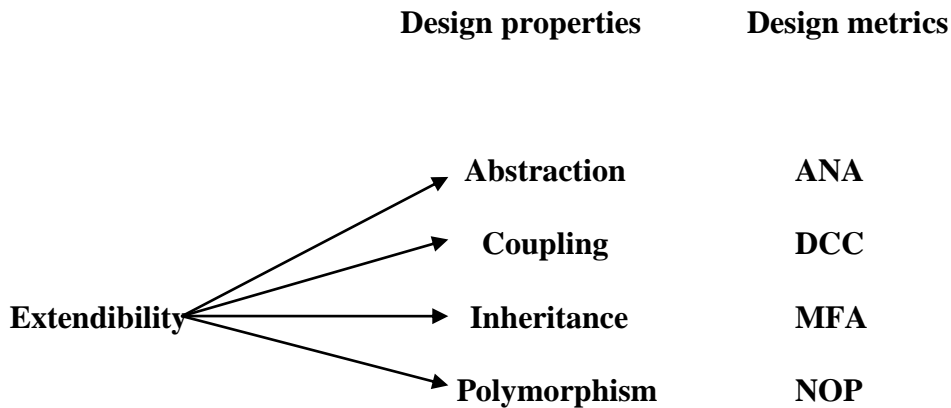


Figure 29: Extendibility design properties and metrics

Design property for adaptation	Adaptation equation
Abstraction	$2 * \text{extendibility} + \text{coupling} - \text{inheritance} - \text{polymorphism}.$
Coupling	$-2 * \text{extendibility} - \text{abstraction} - \text{inheritance} - \text{polymorphism}.$
Inheritance	$2 * \text{extendibility} - \text{abstraction} + \text{coupling} - \text{polymorphism}.$
Polymorphism	$2 * \text{extendibility} - \text{abstraction} + \text{coupling} - \text{inheritance}.$

Table 12: Extendibility adaptation equations

<b>Design change/ Change in design property</b>	<b>Best fit adaptation equation from table 12</b>	<b>Alternate adaptation equation from table 12</b>
Decrease in abstraction.	No best fit.	Coupling and polymorphism equations.
Increase in coupling.	No best fit.	Abstraction and polymorphism equations.
Decrease in inheritance.	No best fit.	Polymorphism and abstraction equations.
Decrease in polymorphism.	No best fit.	Abstraction and coupling equations.
Change more than one design property.	No best fit.	Any design property's equation except the changed one (s).

**Table 13: Classification of extendibility's adaptation equation**

### 3.3.5.2 Extendibility quality flow

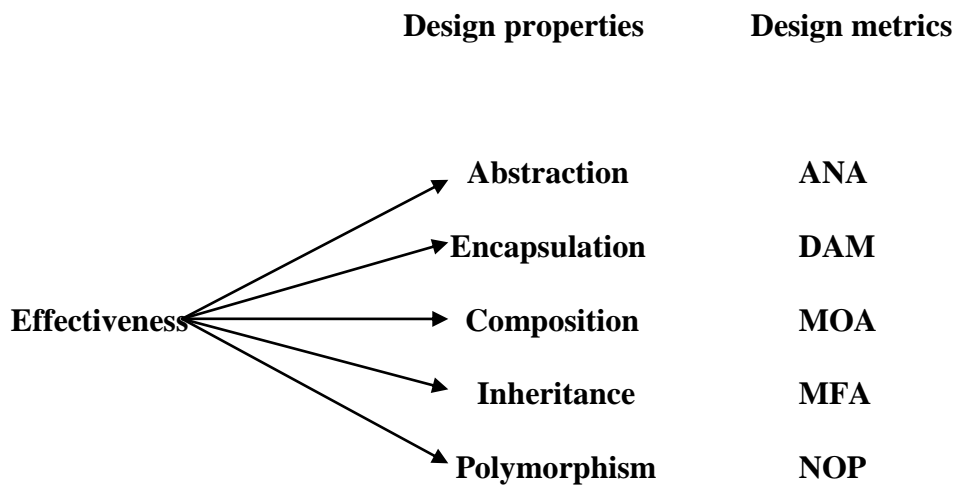
The quality flow in the extendibility sub-model is similar to the previous sub-models (figure 30). Furthermore, the maximum number of design properties that can be changed in one simulation run equals 3. In the case of the extendibility sub-model, the adaptation slices are represented by the adaptation equations of abstraction, coupling, inheritance, and polymorphism properties that are devoted to the extendibility quality attribute (tables 12 and 13). The complete implementation of the extendibility sub-model is described in appendix B.



### 3.3.6 Effectiveness sub-model

#### 3.3.6.1 Effectiveness quality components

In figure 32, effectiveness is simulated with its corresponding design properties and metrics (figure 31). After evaluating the values of the design properties, effectiveness's quality equation is computed and stored in its variable (figures 9 and 31). In addition, design properties' slices implement effectiveness's adaptation equations (tables 14 and 15).



**Figure 31: Effectiveness design properties and metrics**

<b>Design property for adaptation</b>	<b>Adaptation equation</b>
Abstraction	5 * effectiveness – encapsulation – composition – inheritance – polymorphism.
Encapsulation	5 * effectiveness – abstraction - composition – inheritance – polymorphism.
Composition	5 * effectiveness – abstraction – encapsulation – inheritance – polymorphism.
Inheritance	5 * effectiveness – abstraction – encapsulation – composition – polymorphism.
Polymorphism	5 * effectiveness – abstraction – encapsulation – composition – inheritance.

**Table 14: Effectiveness adaptation equations**

<b>Design change/ Change in design property</b>	<b>Best fit adaptation equation from table 14</b>	<b>Alternate adaptation equation from table 14</b>
Decrease in abstraction.	Encapsulation equation.	Composition and polymorphism equations.
Decrease in composition.	Encapsulation equation.	Abstraction and polymorphism equations.
Decrease in inheritance.	Encapsulation equation.	Polymorphism, abstraction, and composition equations.
Decrease in polymorphism.	Encapsulation equation.	Abstraction and composition equations.
Change more than one design property.	Encapsulation equation.	Any design property's equation except the changed one (s).

**Table 15: Classification of effectiveness's adaptation equation**

### 3.3.6.2 Effectiveness quality flow

The effectiveness sub-model is applying the same quality characteristics of the previous sub-models. In this case, the maximum number of design properties that can be changed in one simulation run equals 4. In addition, the adaptation slices are represented by the adaptation equations of abstraction, encapsulation, composition, inheritance, and polymorphism properties that are devoted to the effectiveness quality attribute (tables 14 and 15). The detailed implementation of the effectiveness sub-model is available in appendix B.

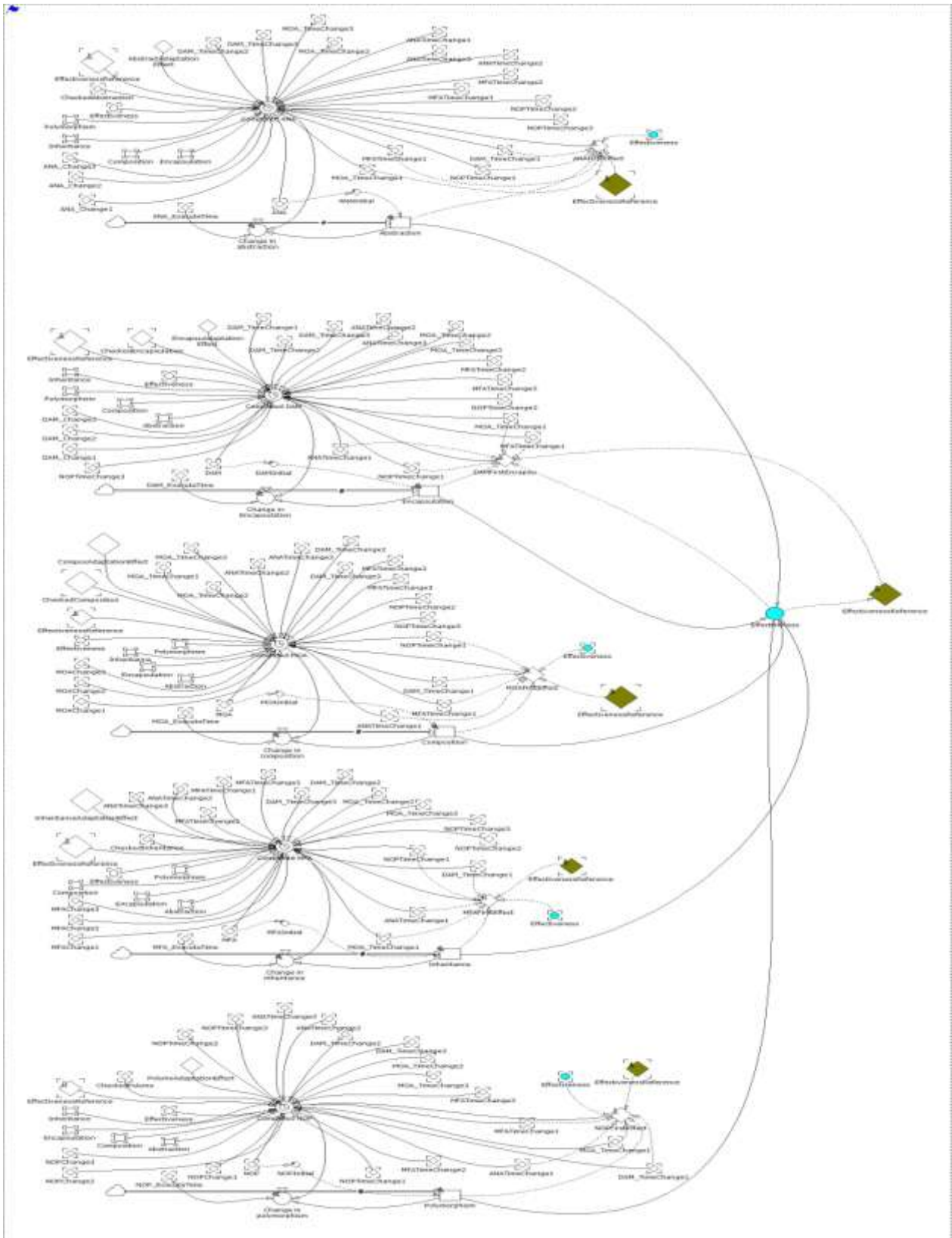


Figure 32: Effectiveness sub-model



**Figure 33: The welcome page of the simulation**



## Input Menu

Check the design properties that will be changed

Messaging     Design Size     Cohesion     Hierarchies     Abstraction     Polymorphism  
 Encapsulation     Coupling     Composition     Inheritance     Complexity

### Reusability adaptation design properties

Coupling     Cohesion     Messaging     Design Size

### Flexibility adaptation design properties

Coupling     Encapsulation     Composition     Polymorphism

### Understandability adaptation design properties

Abstraction     Encapsulation     Coupling     Cohesion  
 Polymorphism     Design Size

### Functionality adaptation design properties

Cohesion     Polymorphism     Messaging     Design Size  
 Hierarchies

### Extendibility adaptation design properties

Abstraction     Coupling     Polymorphism

### Effectiveness adaptation design properties

Abstraction     Encapsulation     Composition     Polymorphism

### Class Interface Size-CIS

CIS	30.00 qual
CISChange1	-15.00 qual
CISChange2	-20.00 qual
CISChange3	-25.00 qual
CISTimeChange1	30.00 da
CISTimeChange2	60.00 da
CISTimeChange3	90.00 da

### Design Size in Classes-DSC

DSC	50.00 qual
DSCChange1	-20.00 qual
DSCChange2	-40.00 qual
DSCChange3	-45.00 qual
DSCTimeChange1	30.00 da
DSCTimeChange2	60.00 da
DSCTimeChange3	90.00 da

### Average Number of Ancestors-ANA

ANA	20.00 qual
ANA_Change1	-5.00 qual
ANA_Change2	-10.00 qual
ANA_Change3	-15.00 qual
ANATimeChange1	30.00 da
ANATimeChange2	60.00 da
ANATimeChange3	90.00 da

### Data Access Metric-DAM

DAM	0.90 qual
DAM_Change1	-0.30 qual
DAM_Change2	-0.50 qual
DAM_Change3	-0.70 qual
DAM_TimeChange1	30.00 da
DAM_TimeChange2	60.00 da
DAM_TimeChange3	90.00 da

### Direct Class Coupling-DCC

DCC	5.00 qual
DCC_Change1	5.00 qual
DCC_Change2	10.00 qual
DCC_Change3	15.00 qual
DCC_TimeChange1	30.00 da
DCC_TimeChange2	60.00 da
DCC_TimeChange3	90.00 da

### Cohesion Among Methods of Classes-CAM

CAM	0.90 qual
CAMChange1	-0.20 qual
CAMChange2	-0.50 qual
CAMChange3	-0.70 qual
CAMTimeChange1	30.00 da
CAMTimeChange2	60.00 da
CAMTimeChange3	90.00 da

### Number Of Methods-NOM

NOM	5.00 qual
NOMChange1	5.00 qual
NOMChange2	10.00 qual
NOMChange3	15.00 qual
NOMTimeChange1	30.00 da
NOMTimeChange2	60.00 da
NOMTimeChange3	90.00 da

### Number Of Polymorphic methods NOP

NOP	20.00 qual
NOPChange1	-5.00 qual
NOPChange2	-10.00 qual
NOPChange3	-15.00 qual
NOPTimeChange1	30.00 da
NOPTimeChange2	60.00 da
NOPTimeChange3	90.00 da

### Number Of Hierarchies-NOH

NOH	20.00 qual
NOHChange1	-5.00 qual
NOHChange2	-10.00 qual
NOHChange3	-15.00 qual
NOHTimeChange1	30.00 da
NOHTimeChange2	60.00 da
NOHTimeChange3	90.00 da

### Measure Of Functional Abstraction -MFA

MFA	0.90 qual
MFAChange1	-0.20 qual
MFAChange2	-0.50 qual
MFAChange3	-0.70 qual
MFATimeChange1	30.00 da
MFATimeChange2	60.00 da
MFATimeChange3	90.00 da

### Measure Of Aggregation- MOA

MOA	20.00 qual
MOAChange1	-5.00 qual
MOAChange2	-10.00 qual
MOAChange3	-15.00 qual
MOA_TimeChange1	30.00 da
MOA_TimeChange2	60.00 da
MOA_TimeChange3	90.00 da

Figure 34: The input menu of the simulation



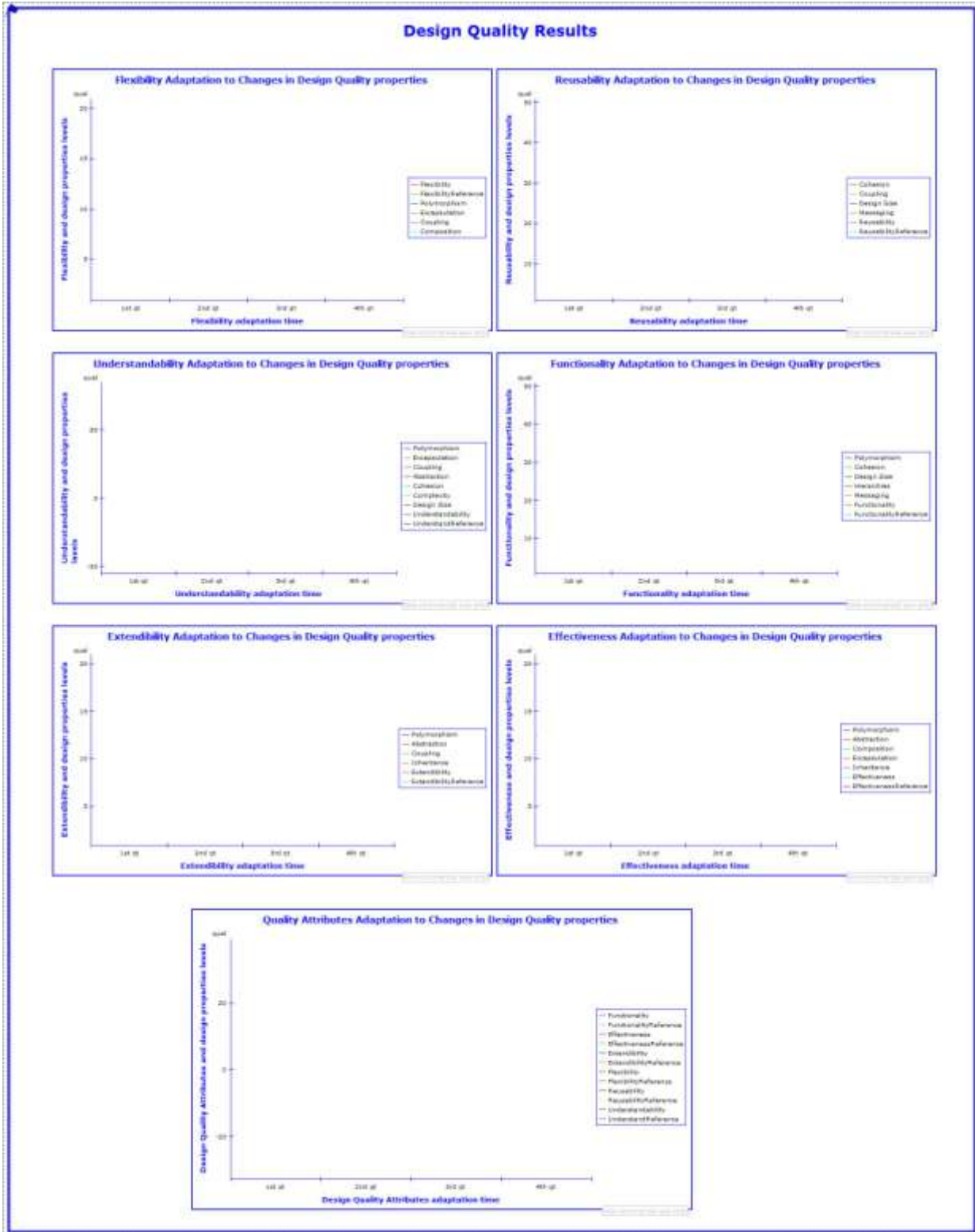


Figure 35: The simulation results page

Chapter 3 was devoted to the creation of the different components of design quality's simulation by following phases 0-2 of the SDM. In PowerSim<sup>®</sup>, each QMOOD quality attribute was developed in a separate sub-model that showed how its quality components interact with each other to face any decrease in design quality. Instead of testing a limited set of scenarios, the enhancement of the initial simulation model enables designers to experiment with all possible combinations of design changes and adaptations. Therefore, the resulting simulation from Phase 2 of the SDM is ready to be applied and validated in a set of real OO designs as illustrated in chapter 4.

## Chapter 4: Simulation validation

To validate the simulation sub-models and apply the suggested adaptations, ten academic design class diagrams were studied (phases 2.2 and 3 of the SDM process in figure 2). To illustrate the validation process and the application of the adaptation mechanisms, one design is described thoroughly in this chapter and the remaining designs are discussed in appendix C. The design documents were produced by students from two different classes offered by the Computer Science and Software Engineering department at Auburn University: Software Modeling and Design (COMP 3700) and the Senior Design Project (COMP 4710). The designs illustrate the components of small to medium-sized systems in different application areas such as healthcare and education. The design changes and their adaptations were validated by applying the following steps in the designs:

- 1) Initial design quality measured: The QMOOD metrics, design properties and quality equations were extracted manually and computed for each design class diagram before applying any design changes. The initial values of the quality attributes in each design are considered as the reference quality values for that particular design in both the simulation and the real results.
- 2) Design changes and adaptations simulated: A set of design changes was experimentally applied to each design through the simulation. It is assumed that changes on the requirements from the client side trigger the design changes. The obtained results depicted

the affected QMOOD quality attributes by those changes. If the quality values were lower than their initial values (i.e. the reference values of the quality attributes) the impact of the selected design properties' adaptation equation was also simulated. The main result of the simulation is the adaptation amount of the selected design property (i.e. how much the adaptation design property should be increased/ decreased to reach the reference value of the affected quality attribute).

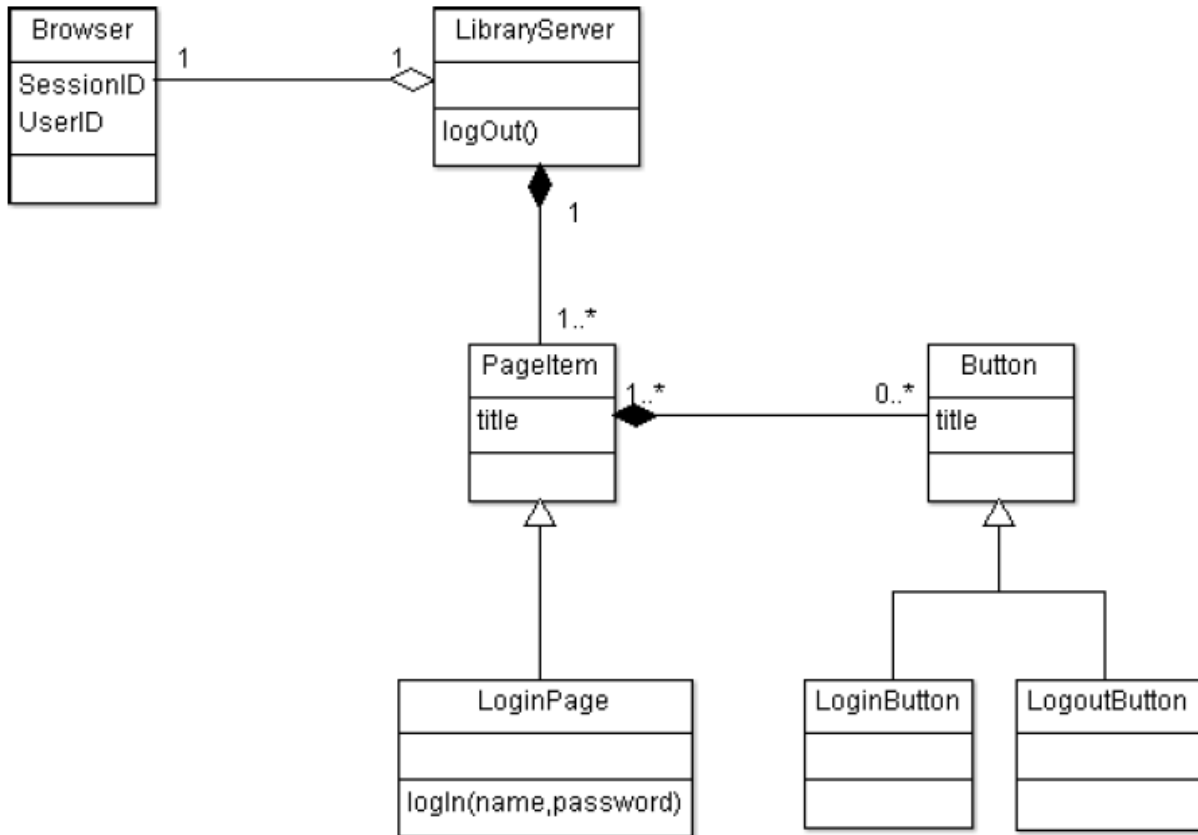
- 3) Design changes and adaptations applied on the designs' class diagrams: The same simulated changes were applied on the real designs as well as their corresponding simulated adaptation amounts. The QMOOD quality attributes were also computed for the adapted class diagrams.
- 4) Correlations between the simulated and the real quality attributes' values from the simulation and the class diagrams were computed: The correlations were calculated by applying the Pearson product-moment coefficient or Pearson's r. Pearson's r determines the linear relationship between two sets of values (e.g. set X and set Y) and can range from -1 to 1 [Jackson 2011]. In this research, X represents the set of all the adapted simulated values of a specific quality attribute and Y represents the set of all the adapted values of the same quality attribute that are computed from the class diagrams.

## **4.1 Design 1 (D1): Library Information System (LIS)**

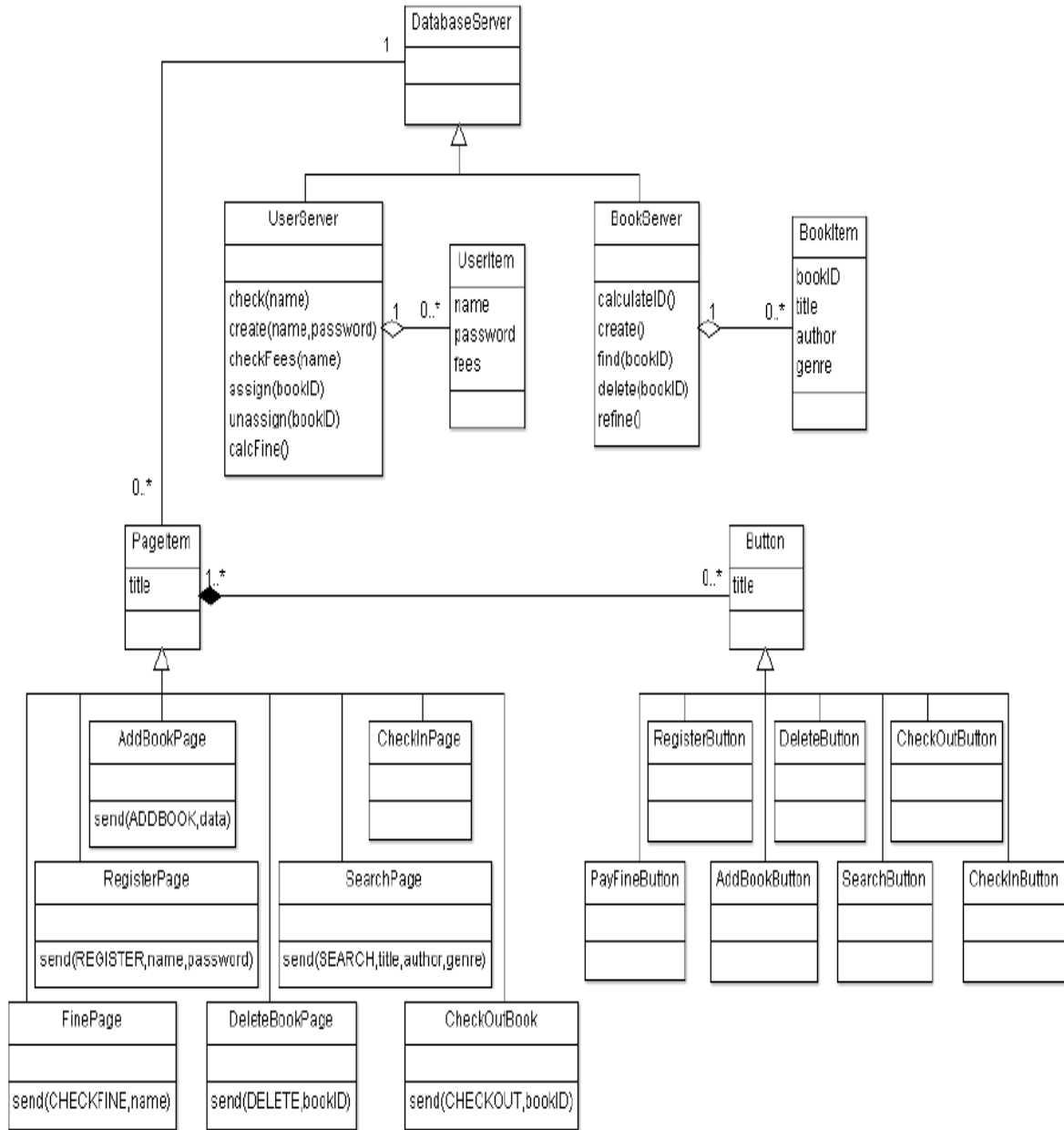
### **4.1.1 System description and reference quality values**

The class diagram in figures 36 and 37 represents the design components of a library information system (LIS). The main goal of the LIS is to automate the operations of library management such as checking books in and out; adding books to the library; and handling

outstanding fees. The initial values of the QMOOD metrics of this class diagram and the remaining designs were extracted by applying the formula in table 16. Both values of the metrics and their corresponding quality attributes' reference values are recorded in tables 17 and 18.



**Figure 36: The library server classes of LIS**



**Figure 37: The database server classes of LIS**

<b>Design metric</b>	<b>Formula</b>
Design Size in Classes (DSC)	$\Sigma$ of classes in the class diagram.
Number Of Hierarchies (NOH)	$\Sigma$ of class hierarchies in the class diagram.
Average Number of Ancestors (ANA)	$\frac{\sum_{i=1}^{i=\text{number of leaf classes}} \text{of ancestors}}{\text{sum of leaf classes in the design}}$
Average Data Access Metric (DAM)	$\frac{\text{Total number of private (protected) attributes in a class}}{\text{Total number of attributes}}$ $\frac{\text{Total number of classes in the design}}{0 \leq \text{DAM} \leq 1 \text{ in each class}}$
Average Direct Class Coupling (DCC)	$\Sigma$ of classes a class is related to.
Average Cohesion Among Methods of classes (CAM)	$\frac{\sum_{i=1}^{i=\text{number of parameters of all methods in a class}} \text{number of methods that share } i}{\text{total number of parameters in a class}}$ $\frac{\text{Total number of classes in the design}}{0 \leq \text{CAM} \leq 1 \text{ in each class}}$
Measure Of Aggregation (MOA)	The number of part-whole relationships in the class diagram.
Average Measure of Functional Abstraction (MFA)	$\frac{\text{The number of methods inherited by a class}}{\text{Total number of methods accessible}}$ $\frac{\text{Total number of classes in the design}}{0 \leq \text{MFA} \leq 1 \text{ in each class}}$
Number Of Polymorphic methods (NOP)	$\Sigma$ of polymorphic methods in the class diagram.
Sum of Classes Interfaces Size (CIS)	$\Sigma$ of public methods in the class diagram.
Number Of Methods (NOM)	$\Sigma$ of all methods in the class diagram.

**Table 16: QMOOD design metrics formula**

## **4.1.2 Design changes**

After extracting the design metrics values from the class diagram in figures 36 and 37, a set of design changes was applied to validate the results of the simulation. Each design change affected at least one quality attribute and enabled us to validate its corresponding simulation sub-model.

### **4.1.2.1 Design changes affecting the understandability quality attribute**

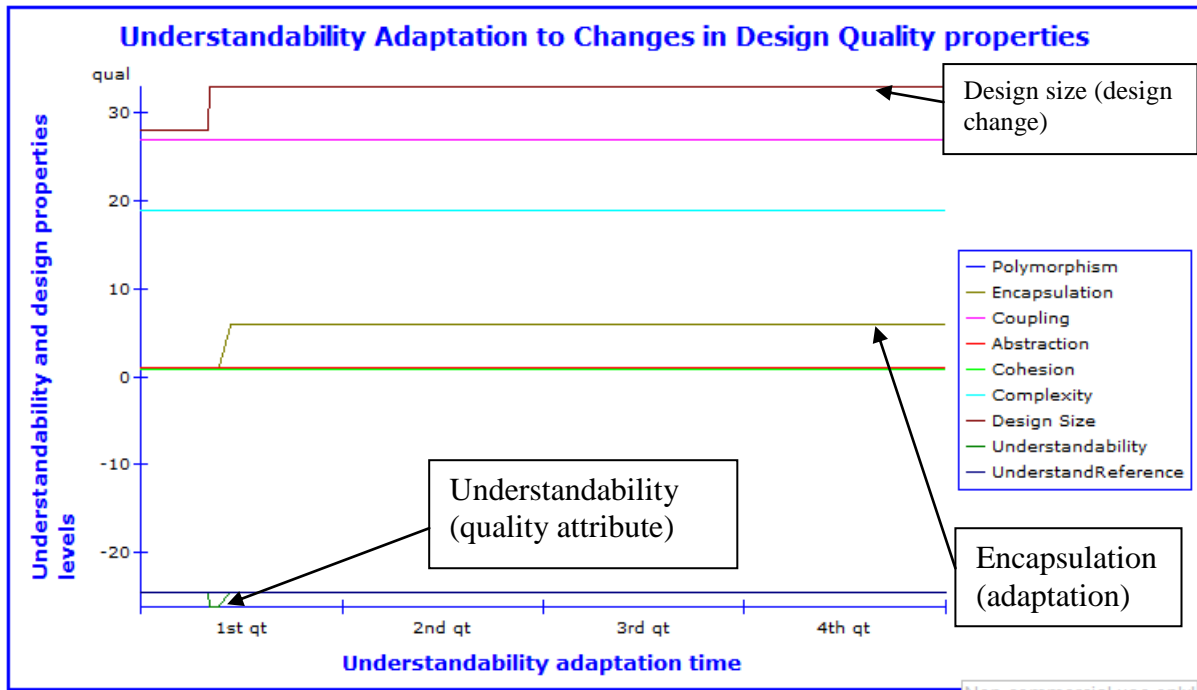
After entering D1's design metrics values from table 16 in the simulation interface, the initial understandability value (i.e. the targeted reference value) was evaluated (Table 17). One of the new requirements received from the system's client is to add new functionalities that deal with books management, library events organization, amenities reservation, and complaints management. Therefore, five new classes were added to D1's class diagram: "Client service", "Books suggestion", "Library events", "Library amenities", and "Post complaints" that deal with new operations such as allowing users to suggest books and post complaints about any library service. The impact of this design change on understandability and the adopted equation of encapsulation were first experimentally simulated in Powersim<sup>®</sup> before applying them on D1's class diagram.

#### **1) Simulated results**

Increasing the DSC metric of D1 (table 15) by increasing the number of classes led to a decrease in understandability below its reference value (table 17). To counterbalance the impact of that design change, the adaptation equation of encapsulation from table 8 was applied as illustrated in figure 38.



The increase in design size (dark pink) led to a decrease in understandability (green) below its reference value (blue). Understandability started to increase and reached its reference value when the adaptation equation of encapsulation (brown) was applied.



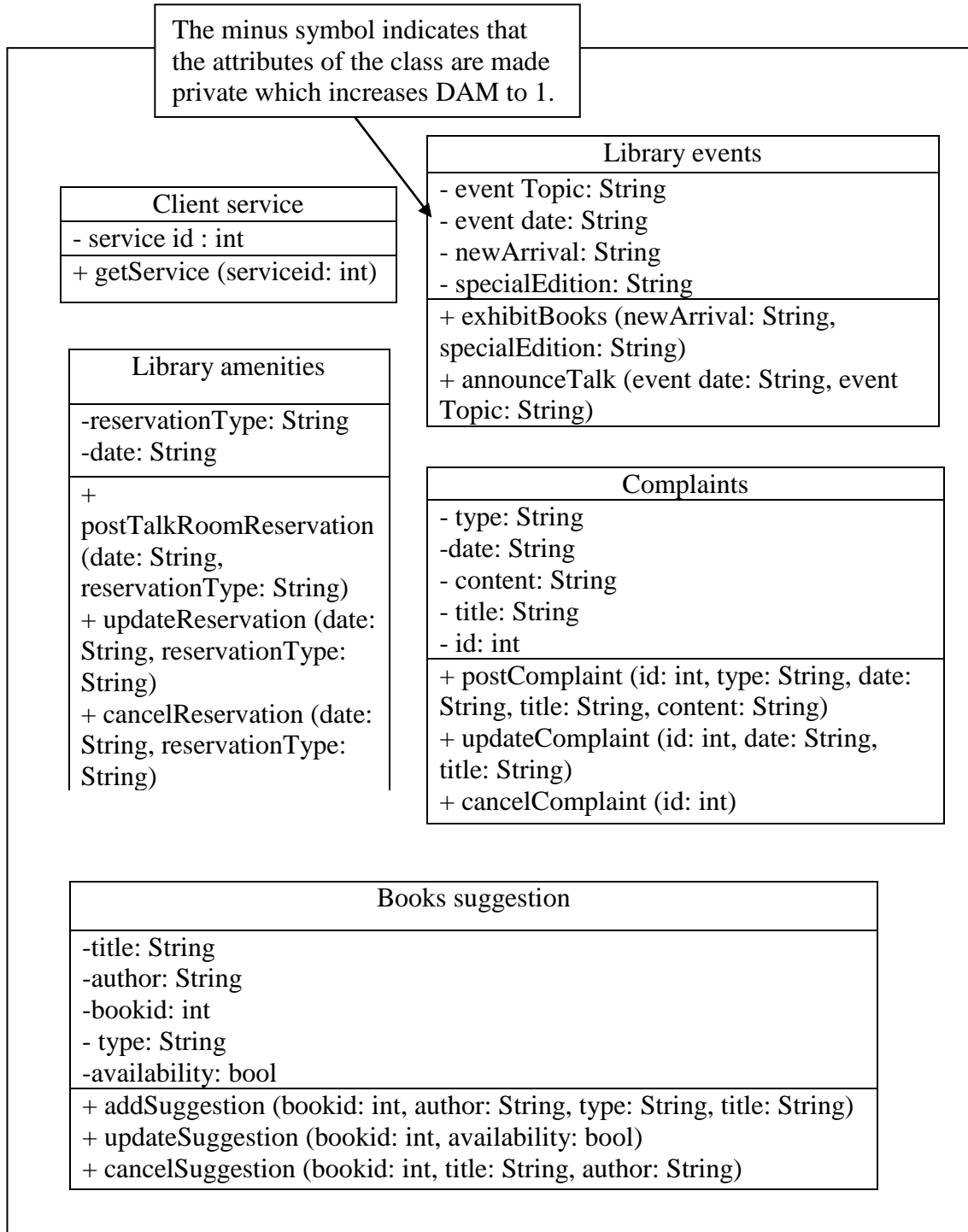
**Figure 38: Understandability adaptation results of D1**

From figure 38, the decrease in understandability is counterbalanced when encapsulation increases from its original value of 1 to 5, or a factor of five. The encapsulation of D1 can be improved by increasing the DAM values to 1 in the following classes: “Client service”, “Books suggestion”, “Library events”, “Post complaints”, and “Library amenities”.

## 2) Real results

After experimenting the impact of the design change on understandability and the effectiveness of the encapsulation adaptation equation through Powersim<sup>®</sup>, the same changes and the obtained adaptation from the simulation were applied on D1’s class diagram (figure 39). The encapsulation adaptation, which is based on increasing DAM in D1’s classes, is illustrated through the UML minus symbol in front of the classes’ attributes. Before applying the simulation

results on the real design, the attributes of the classes in figure 39 were all public. The increase in encapsulation as an adaptation mechanism is based on making those attributes private by adding the minus sign in front of them.



**Figure 39: The classes used in D1's understandability design change**

After applying the encapsulation adaptation on D1's design, the real QMOOD understandability value was computed. Table 20 results show that the simulated value of understandability after adaptation matches its real value.

#### **4.1.2.2 Design changes affecting the extendibility and the flexibility quality attributes**

Additional design changes were simulated and then applied on the adapted version of D1 after the first design change. To illustrate the effect of design changes on extendibility and flexibility, the "Client service" class from figure 39 was modified to be a subclass of "PageItem". In addition, each of the remaining four classes in figure 39 was modified to be a specific client service. The impact of those design changes on extendibility and flexibility as well as the adopted polymorphism adaptation mechanisms was first simulated in Powersim<sup>®</sup> and then applied on the real class diagram of D1.

##### **1) Simulated results**

The described design changes led to an increase in the number of hierarchies, inheritance, complexity, abstraction, messaging, complexity, and coupling design properties. Figures 40 and 41 show that both quality attributes decreased below their reference values before applying the polymorphism adaptation equation (the blue and green lines represent the reference values of extendibility and flexibility respectively. The reference values are also recorded in table 17). From table 7, the best fit adaptation equation for flexibility is the equation of encapsulation. However, encapsulation was already at its optimum level (DAM=1 in D1's classes) and did not need an additional increasing. In this case, the alternate adaptation equation of polymorphism described in tables 6 and 7 was simulated. In the case of extendibility, table 13 shows that this quality attribute has no best fit adaptation but can be adapted by applying the alternate equation

of polymorphism. From figures 40 and 41, the decrease in flexibility and extensibility was counterbalanced when polymorphism was increased by a factor of ten.

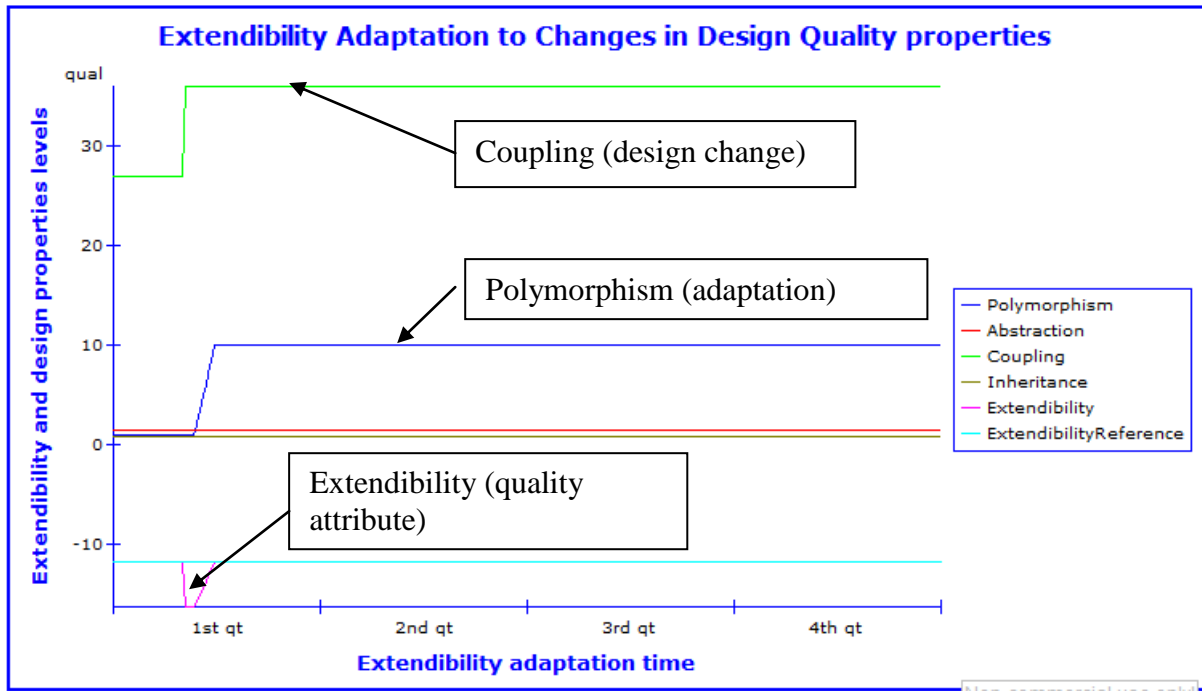


Figure 40: Extensibility adaptation results of D1

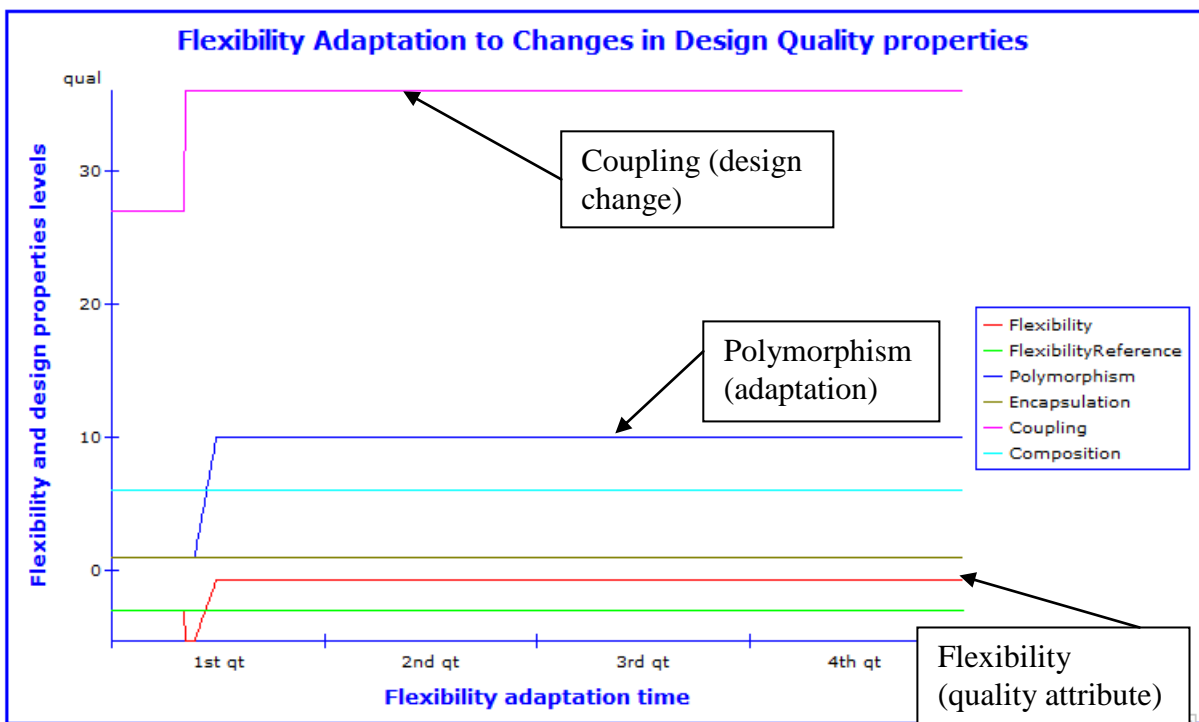
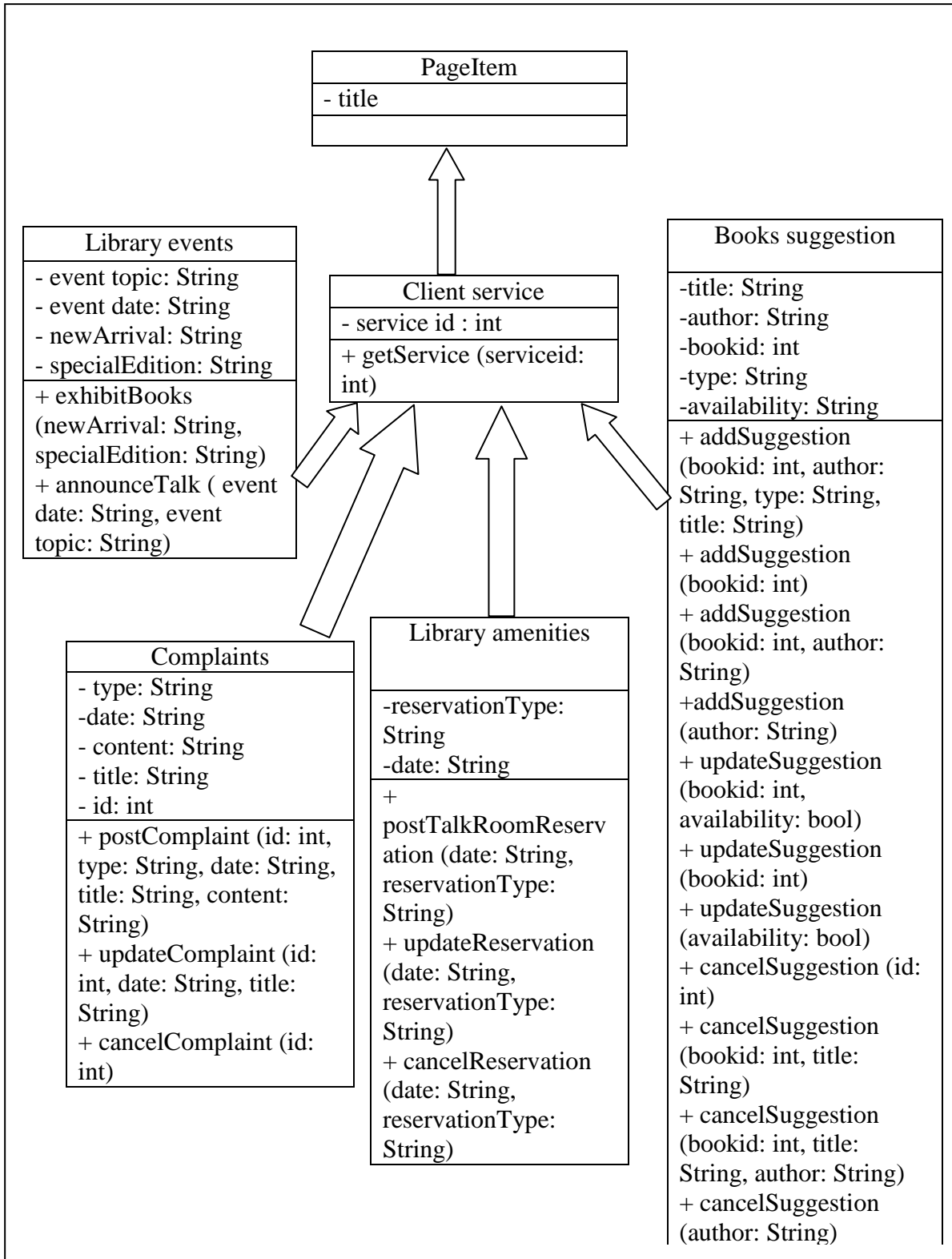


Figure 41: Flexibility adaptation results of D1

## **2) Real results**

The simulated changes and their corresponding adaptations for extendibility and flexibility were applied on D1 as shown in figure 42. The simulated adaptation was applied in the “Books suggestion” class where the polymorphic forms of its methods are increased to nine. Then, the computed values of extendibility and flexibility from D1 after adaptation were compared to their simulated values. The results show a strong connection between the real and the simulated values of both quality attributes (tables 22 and 23 in appendix C).



**Figure 42: The changed parts of D1 for flexibility and extensibility adaptations**

#### **4.1.2.3 Design changes affecting the reusability and the functionality quality attributes**

After adapting the values of the extendibility and the flexibility quality attributes, additional design changes were experimented in the simulation then applied on D1's class diagram. The classes "CheckInPage", "CheckOutBook", and "FinePage" in figure 37 were deleted from D1.

##### **1) Simulated results**

The new design changes led to a decrease in the design size of D1 and two QMOOD quality-attributes below their reference values: reusability and functionality (table 17, figures 43 and 44). The best-fit adaptation equation for reusability and functionality is the equation of cohesion as it is described in tables 5 and 11. When the value of cohesion was increased after applying its adaptation equation, the reusability and the functionality values increased and reached their reference values as illustrated in figures 43 and 44. Furthermore, cohesion is measured through the CAM metric, which represents the degree of relatedness among the methods of a design's classes. The simulation results in figures 43 and 44 suggested that cohesion/ CAM should equal 1 in six classes of D1. The application of those simulated suggestions in the real design was described in figure 46.

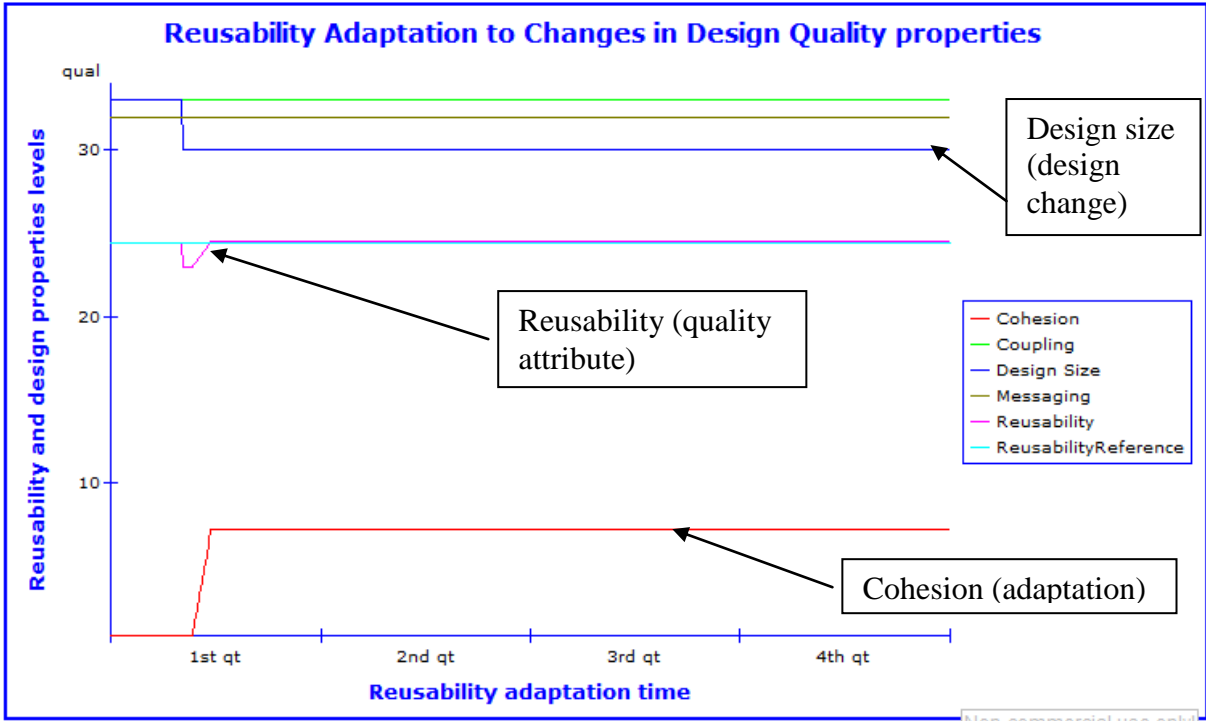


Figure 43: Reusability adaptation results of D1

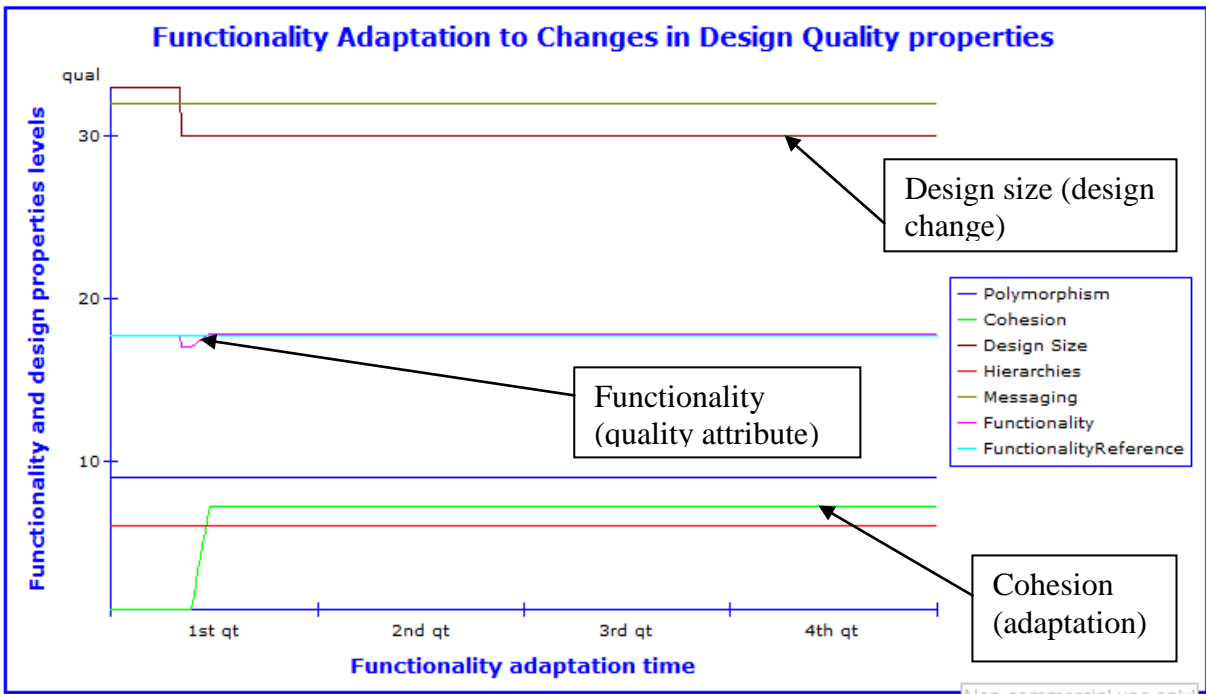


Figure 44: Functionality adaptation results of D1



## **2) Real results**

The changes that affect reusability and functionality as well as the suggested adaptation from the simulation were applied on D1. After applying the changes, the parameters in the classes “Library events”, “Complaints”, and “Library amenities” were shared by most of their methods, increasing the value of CAM from 0.8 to 1. This adaptation mechanism counterbalances the decrease in both quality attributes and makes the real values nearly equal their simulated counterparts (tables 24 and 25 in appendix C).

### **4.1.2.4 Design changes affecting the effectiveness quality attribute**

In the last design change to D1, the “Client service” class was deleted, leaving the remaining classes to inherit characteristics directly from the “PageItem” class (figure 46). The impact of this design change on effectiveness and the adopted adaptation equation was first simulated in Powersim<sup>®</sup> and then applied on the real class diagram of D1.

#### **1) Simulated results**

This design change led to a decrease in the abstraction design property and the effectiveness quality attribute. From table 15, the equation of encapsulation is the best fit adaptation of effectiveness. However, encapsulation was already at its optimum level (i.e. DAM = 1 in D1’s classes) and did not need to be increased. As a result, the alternate adaptation equation of polymorphism described in tables 14 and 15 was simulated. Figure 45 illustrates that polymorphism must increase from 9 to 18 to accommodate the design change. Thus, the adaptation through NOP, the QMOOD measure of polymorphism, should be increased by nine polymorphic methods in the D1’s class diagram.

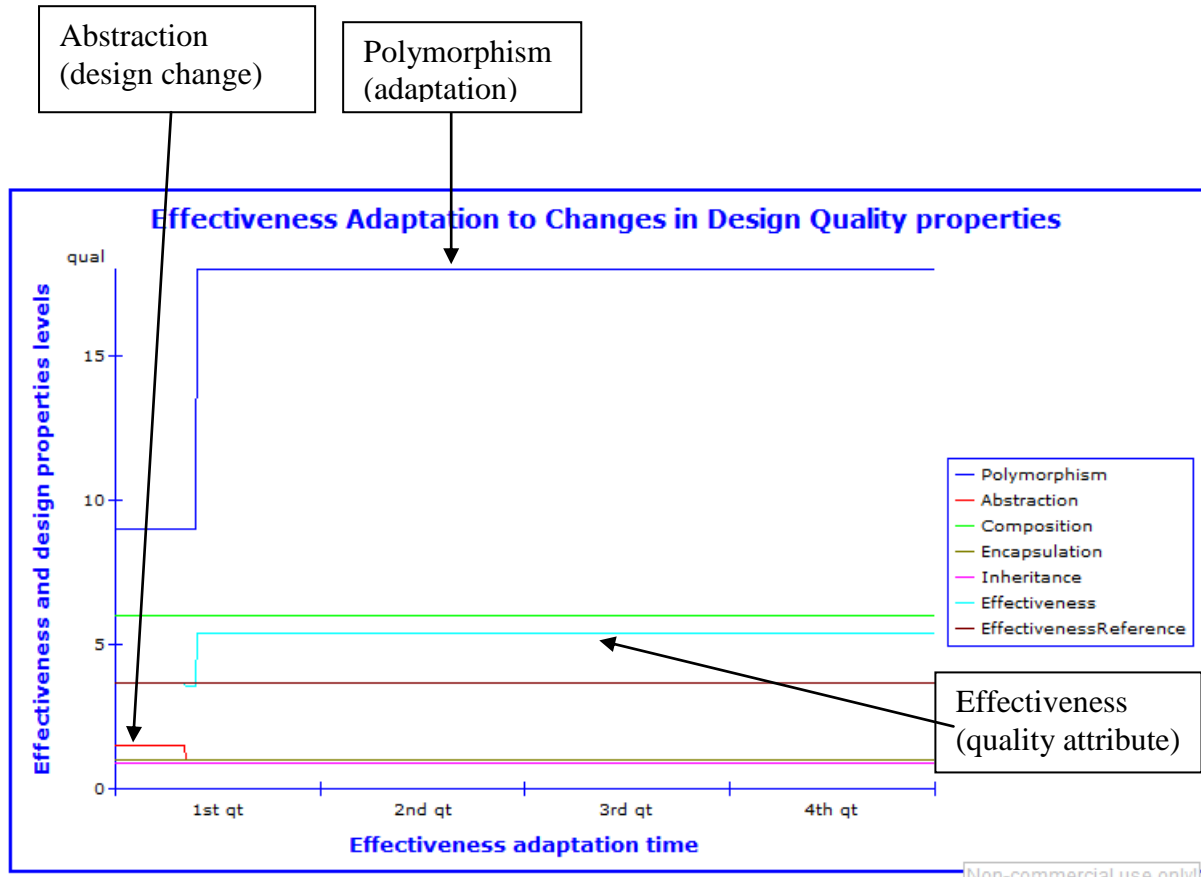
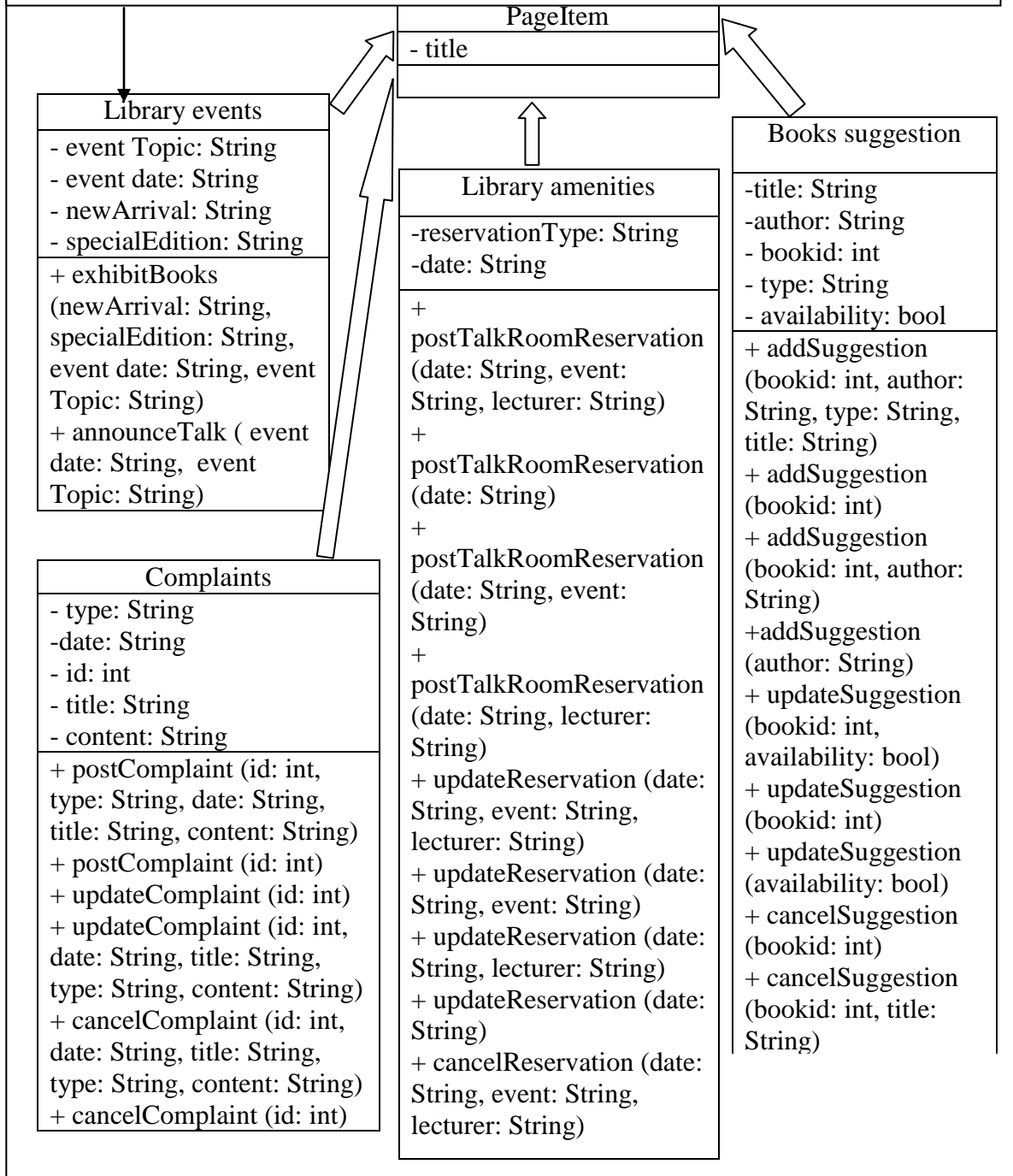


Figure 45: Effectiveness adaptation results of D1

## 2) Real results

The simulated changes and adaptations that affect effectiveness were applied on D1's class diagram. The polymorphic adaptation from the simulation was applied in the "Library amenities" and "Complaints" classes where the polymorphic forms of its methods were increased to eighteen (figure 46). The computed value of effectiveness from D1's class diagram after adaptation nearly equals its simulated value (table 26 in appendix C).

The intersection between the parameters of the methods and the overall class attributes is high so that CAM = 1. All of the class attributes are almost part of each method's parameters. Thus, the cohesion of the class that corresponds to CAM is increased, which adapts the values of reusability and functionality. Adding polymorphic methods, such as "updateSuggestion (availability: bool)" in 'Books suggestion' class, adapted the value of effectiveness.



**Figure 46: The changed parts of D1 for reusability, functionality, and effectiveness adaptation**

The same validation process was applied in D2-D10 designs where the data in tables 17 and 18 were used in running the simulation and checking that the obtained quality attributes reached their reference values after adaptation. The detailed description of design changes and adaptations of D2-D10 is described in appendix C.

<b>Designs</b> <b>QMOOD</b> <b>Metrics</b>	<b>D1</b>	<b>D2</b>	<b>D3</b>	<b>D4</b>	<b>D5</b>	<b>D6</b>	<b>D7</b>	<b>D8</b>	<b>D9</b>	<b>D10</b>
<b>DSC</b>	28	13	47	38	40	21	10	41	7	12
<b>NOH</b>	5	1	3	5	5	1	3	4	0	0
<b>ANA</b>	1	1	1	1.45	1	1	1.75	1.06	0	0
<b>DAM</b>	1	0.7	0.25	1	1	1	1	1	1	1
<b>DCC</b>	27	11	9	19	21	10	14	25	16	6
<b>CAM</b>	0.8	1	0.19	1	0.2	1	1	1	1	1
<b>MOA</b>	6	7	0	3	2	2	1	8	16	6
<b>MFA</b>	0.5	0.8	0	0	0.98	0.2	0.27	0.66	0	0
<b>NOP</b>	1	1	1	1	1	1	1	1	1	3
<b>CIS</b>	19	33	58	18	40	101	22	65	31	98
<b>NOM</b>	19	33	58	18	40	101	22	65	31	98

**Table 17: The initial QMOOD design metrics values for the ten designs**

Designs Reference quality values	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
	<b>Reusability</b>	23.2	31.75	51.8	33	38.05	47.75	61.75	20	21.25
<b>Flexibility</b>	- 3	- 1.5	- 1.5	- 2.5	- 3.5	- 1.5	- 0.25	- 2.25	- 4.75	- 3.25
<b>Understandability</b>	- 24.4 9	- 18.9	- 38.13	- 24.90	- 33.59	- 43.25	- 43.56	- 15.43	- 17.49	- 38.61
<b>Functionality</b>	17.4 8	18.38	26.64	20.8	22.02	25.86	30.26	12.44	13.98	27.84
<b>Extendibility</b>	- 12.2 5	- 4.10	- 3.50	- 8.13	- 9.06	- 11.14	- 3.88	- 5.45	- 6.75	- 0.7
<b>Effectiveness</b>	3.6 8	4.12	0	0	2.18	3.34	2.45	1.82	6.5	2.92

**Table 18: The reference values of the quality attributes for both the simulated and the real results in the ten designs**

#### 4.2 Quality attributes correlation results

After experimenting and applying a set of design changes and their adaptations on the simulation and the real designs, the resulting simulated and real quality attributes were generated and computed. Then, the Pearson product-moment correlation coefficient (Pearson's r) was computed for each design quality attribute between its simulated and its real values. Pearson's r coefficient determines the association level of two sets of variables X and Y [Jackson 2011]. It can be computed by applying the following formula where n represents the population's size (e.g. size of X):

$$r_{xy} = \frac{N\sum XY - (\sum X)(\sum Y)}{\sqrt{[N\sum X^2 - (\sum X)^2][N\sum Y^2 - (\sum Y)^2]}}$$

**Figure 47: Pearson's r formula [Jackson 2011]**

Pearson's  $r$  value ranges from -1 to 1 with specific correlation strength (table 19). The sets X and Y can be perfectly correlated when  $r = 1/-1$ . Moreover, X and Y are considered not linearly related or correlated when  $r = 0$ . In the validation of this research, X represents the simulated values of the quality attributes after adaptation while Y is the set of their corresponding real values. A high or very high correlation between X and Y shows the effectiveness of the adaptation equations in adjusting design quality (table 19). The detailed application of Pearson's  $r$  in each design quality attribute is illustrated in tables 19-24. Pearson's  $r$  computes the correlation between two sets of variables. Therefore, Pearson's  $r$  is computed for each QMOOD quality attribute between the set X of all its simulated values and the set Y of all its real values in all the designs. Table 20 illustrates the computation of Pearson's  $r$  for understandability between the set X of its simulated values and the set Y of its real values over all designs. The correlations of the remaining quality attributes are presented in appendix C.

<b>Correlation strength</b>	<b>Negative correlation value</b>	<b>Positive correlation value</b>
Very low	$-0.3 < r < 0$	$0 < r < 0.3$
Low	$-0.5 < r < -0.3$	$0.3 < r < 0.5$
Moderate	$-0.7 < r < -0.5$	$0.5 < r < 0.7$
High	$-0.9 < r < -0.7$	$0.7 < r < 0.9$
Very high	$-1 < r < -0.9$	$0.9 < r < 1$

**Table 19: Pearson's  $r$  correlation degrees [Jackson 2011]**

<b>X: Simulated understandability in D1-D10</b>	<b>Y: Real understandability in D1-D10</b>	<b>XY</b>	<b>X<sup>2</sup></b>	<b>Y<sup>2</sup></b>
-24.49	-24.47	599.27	599.76	598.78
-18.90	-18.89	357.19	357.55	356.83
-38.13	-38.13	1453.89	1453.89	1453.89
-24.90	-24.88	619.51	620.01	619.01
-33.59	-33.58	1127.95	1128.28	1127.61
-43.25	-43.24	1870.13	1870.56	1869.69
-43.56	-43.53	1896.16	1897.47	1894.86
-15.43	-15.42	237.93	238.08	237.77
-17.49	-17.49	305.90	305.90	305.90
-38.61	-38.60	1490.34	1490.73	1489.96
<b>Σ X= -298.35</b>	<b>Σ Y= -298.23</b>	<b>Σ XY= 9958.27</b>	<b>Σ X<sup>2</sup>= 9962.23</b>	<b>Σ Y<sup>2</sup>= 9954.30</b>
<b>n =10</b>				

**Table 20: Correlation computations of understandability**

$$r_{xy} = \frac{10 (9958.27) - (-298.35) (-298.23)}{\sqrt{|10 (9962.23) - (-298.35)^2| * |10 (9954.30) - (-298.23)^2|}}$$

= 1 very high correlation

<b>Quality attribute</b>	<b>Correlation value</b>	<b>Correlation degree</b>
Understandability	1	Very high
Extendibility	0.99	Very high
Flexibility	0.90	Very high
Functionality	0.99	Very high
Reusability	0.99	Very high
Effectiveness	0.97	Very high

**Table 21: Correlation degrees of the QMOOD quality attributes**

From the obtained Pearson's  $r$  coefficient values in table 21, the simulated values of the six quality attributes after adaptation highly correlate with their real values. Therefore, the simulation results are valid and the suggested adaptation equations are effective. In addition, the obtained results concretely reject the null hypothesis of this research in favor of the alternative hypothesis.



## Chapter 5: Conclusions and future research work

### 5.1 Conclusions

The presented research extracted a set of adaptation equations from the QMOOD to face any possible decrease in design quality due to changes in design decisions. Those adaptation equations as well as the different components of the QMOOD were modeled as a system dynamics simulation implemented in Powersim<sup>®</sup>. The simulation was an experimental environment where designers can test the impact of changes before applying them on real designs. The simulation was also useful in defining the appropriate adaptation equation for each design change that decreased the value of a specific QMOOD quality attribute.

The validation of the simulation showed that the suggested adaptations can be applied effectively and smoothly in the real designs. After applying a set of design changes on the real designs, design quality dropped below its defined reference values. Then, the obtained increase in the adaptation design properties from the simulation was applied on the real designs. The validation of the research showed that the simulated results highly correlated with the real adaptations. Therefore, the suggested adaptations can effectively adjust design quality. Moreover, the application of any of the adaptation equations for a specific quality attribute had no side effect on the other quality attributes since they kept their high level and did not drop below their reference values.

## 5.2 Future research work

The simulation produced in this research can be extended to study more design quality variables and adapt other phases of the software process. The current simulation uses only the quality attributes defined in the QMOOD. However, software design can also be characterized by other quality attributes such as reliability whose assessment and adaptation can improve design quality. Moreover, design quality can also be impacted by other exogenous soft factors such as the motivation of designers, their commitment and their organizational culture. The simulation can help in defining the impact of those soft factors on QMOOD quality attributes and depict the required adaptations in this case. In addition, a mapping catalog between each quality attribute and its set of impacting soft factors can be defined. For example, the simulation can define the organizational, the economic and the psychological factors impacting the reusability quality attribute such as the required effort, the existing incentives and the job security threats respectively. The simulation can also be extended to assess and adapt other software process phases such as the requirements phase. Through the simulation of the complete software process, software engineers will be able to determine the impact of a specific process phase on the quality of other phases. Furthermore, software engineers will also be able to evaluate the impact of skipping a process phase on the quality of the other phases and the produced software.

The current research defines local reference values for each particular design. The reference values are defined as the initial values of the quality attributes before any design change. As an improvement, those reference values can be determined statistically along many sets of projects or from the history of projects in a company. The reference values can also be updated from one simulation run to another or fixed from the first run. Similarly, the same

methodologies can determine the required initial values of the design properties to run the simulation for any project type. Therefore, designers will be able to determine a uniform proportion for each design property that can be run in any design's simulation. For example, statistical studies over a large set of designs may suggest that the third of any design should be composed of hierarchies and then assign the value .3 to the hierarchies design property and its corresponding metric in the simulation.

Besides design quality evaluation and adaptation, the simulation can be extended to include other options that will help designers in optimizing their designs' quality and estimating the costs of adaptations. To find realistic trade-offs between design decisions and the intended quality attributes values, a quality optimization mechanism can effectively guide designers in forecasting those trade-offs. Therefore, the simulation can be extended to provide designers with the needed values of design properties to reach any targeted quality attributes levels. Another interesting extension to the simulation is to run a cost-benefit analysis of the simulated design changes and their corresponding adaptations. Hence, designers will be able to forecast the costs of their design changes and adaptations before applying them in the real designs.

To consolidate the obtained results from this research, more validations are required by including other types of design deliverables and simulating industry-based software designs. Besides the UML class diagrams that were studied in this research, the simulation will be extended to evaluate and adapt other design data obtained from other types of design deliverables such as the entity-relationship and the data flow diagrams. As an industrial validation to this research, the simulation results will be applied in open-source software. This type of validation is based on a long time consuming process and constitute by itself a research project. This type of validation may specialize on one type of open source software such as android applications or

consider different types of software at the same time. Then, those applications should be reverse-engineered to transform code to UML class diagrams. Design quality should be assessed from one reverse-engineered release to another by applying QMOOD. The reverse-engineered designs should also be thoroughly analyzed to detect any design changes from one release to another. A decrease in design quality indicates that changes need to be counterbalanced by applying one of the presented adaptation mechanisms in this research. All of this data should be experimented in the simulation before applying the obtained adaptations on the real reversed design. Then, a second QMOOD quality evaluation should be performed on the real adapted design to check the effectiveness of adaptations. The same methodology is applied on the different releases of an open-source project as well as in the remaining projects. By the end, Pearson's  $r$  correlation factors are computed between the simulated and the real results.

## References

ABDEL-HAMID, T. , AND MADNICK, S.E. 1991. *Software project dynamics: an integrated approach*. Prentice-Hall, Inc., Upper Saddle River, NJ.

ABREU, F.B., AND MELO, W. 1996. Evaluating the impact of object-oriented design on software quality. In *Proceedings of the 3<sup>rd</sup> International Software Metrics Symposium (Metrics'96)*, Berlim, Alemanha.

ANDERSSON, C., KARLSSON, L., NEDSTAM, J., HOST, M., AND NILSSON, B. 2002. Understanding software processes through system dynamics simulation: a case study. In *Proceedings of the 9<sup>th</sup> Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, Lund, Sweden, 41-48.

ARAUJO, E., CASSIVI, L., CLOUTIER, M. , AND ELIA, E. 2007. Improving the software development process: a dynamic model using the capacity maturity model. In *Proceedings of the 25<sup>th</sup> International Conference of the System Dynamics Society*, Boston, 1-19.

BANSIYA, J., DAVIS, D .C.G. 2002. A hierarchical model for object-oriented design quality assessment. *Journal of IEEE Transactions on Software Engineering*, 28, 4-17.

BECKMANN, B.E., GRABOWSKI, L.M., MCKINLEY, P.K., AND OFRIA, C. 2009. Applying digital evolution to the design of self-adaptive software. In *Proceedings of the IEEE Symposium on Artificial life*, Nashville, TN, 100-107.

BOGADO, V., GONNET, S., AND HORACIO, L. 2010. An approach Based on DEVS for Evaluating Quality Attributes. In *Proceedings of the XXIX International Conference of the Chilean Computer Science Society*, Antofagasta, Chile, 110-118.

BRANDON, J. 2007. Similarity of temporal query logs. Doctoral dissertation. University of California, Los Angeles.

BRIAND, L.C., WUST, J., DALY, J.W. , AND PORTER, D.V. 2000. Exploring the relationships between design measures and software quality in object-oriented systems. *The Journal of Systems and Software*, 51, 245-273.

CHIANG, E., AND MENZIES, T. 2003. Simulations for very early lifecycle quality evaluations. *Software Process: Improvement and Practice* 7, 141-159.

CHWIF, L., MUNIZ SILVA, P.S., STURLINI, R.N., AND SHIMADA, L.M. 2006. A prescriptive technique for V&V of simulation models when no real-life data are available. In *Proceedings of the 38<sup>th</sup> Winter Simulation Conference*, Monterey, Canada, 911-918.

DE OLIVEIRA, M. , LIMA WERNER, L.C., AND TRAVASSOS, G.H. 2000. Using process modeling and dynamic simulation to support software process quality management. In *Proceedings of the 14<sup>th</sup> Brazilian Symposium in Software Engineering*, Paraíba, Brazil.

DIETMAR, P., AND LEBSANT, K. 1999. Integration of system dynamics modeling with descriptive process modeling and goal-oriented measurement. *Journal of Systems and Software*, 46, 135-150.

DONZELLI, P., AND IAZEOLLA, G. 2001. A dynamic simulator of software processes to test process assumptions. *Journal of Systems and Software*, 56, 81-90.

DONZELLI, P., AND IAZEOLLA, G. 2001. Hybrid simulation modeling of the software process. *Journal of Systems and Software* 59, 227-235.

DOOLEY, J. 2011. Object-Oriented design principles. *Journal of Software Development and Professional Practice*, 115-136.

DRAPPA, A., AND LUDEWIG, J. 1998. Simulation model development based on the function point metric. In *Proceedings of the European Measurement Conference FESMA* Seite, Antwerpen, 515-523.

DRAPPA, A., AND LUDEWIG, J. 1999. Quantitative modeling for the interactive simulation of software projects. *Journal of Systems and Software*, 46, 113-122.

DROMEY, G.R. 1996. Cornering the chimera. *Journal of IEEE software*, 13, 33 – 43.

Extend<sup>®</sup> corporation, 1988. <http://www.extendsim.com/>.

FERAYORNI, A.E., AND SARJOUGHIAN, H.S. 2007. Domain Driven Simulation Modeling for Software Design. In *Proceedings of the Summer Computer Simulation Conference*, San Diego, CA, July 2007, 297- 304.

FORRESTER, J.W. 1961. *Industrial dynamics*. MIT Press, Cambridge, MA.

HUANG, L., JIDONG, G., BOEHM, B., AND JIAN, L. 2010. Modeling the value-based software process with object-petri nets. *International Journal of Software Informatics*, 4, 101-119.

IEEE Standard for Developing a Software Project Life Cycle Process. IEEE Std 1074-2006 (Revision of IEEE Std 1074-1997), 0\_1-104.



IOANA, R., COLLOFELLO, J., AND LAKEY, P. 1999. Software process simulation for reliability management. *Journal of Systems and Software*, 46, 173-182.

ISO-9126, ISO/ IEC standard. 1991. Software product evaluation, quality characteristics and guidelines for their use.

JACKSON, S.L. 2011. *Research methods and statistics: a critical thinking approach*. Cengage learning, Belmont, CA.

KAHEN, G., LEHMAN, M., RAMIL, J.F., AND WERNICK, P. 2001. System dynamics modeling of software evolution processes for policy investigation: approach and example. *Journal of Systems and Software*, 59, 271-281.

KELLNER, M. 1991. Software process modeling support for management planning and control. *In Proceedings of the 1st International Conference on the Software Process*, Los Alamitos, California, 8-28.

KELLNER, M. , I., MADACHY, R.J., AND RAFFO, D.M. 1999. *Software process simulation modeling: why? what? How?* , *Journal of Systems and Software*, 46, 91-105.

KLEIJNEN, J.P.C. 1995. Verification and validation of simulation models. *European Journal of Operational Research*, 82, 145-162.

KUMAR, D., TANTAWI, A., and ZHANG, L. 2009. Real-time performance modeling for adaptive software systems. ACM SIGMETRICS: ACM Special Interest Group on Measurement and Evaluation, In *Proceedings of the 4<sup>th</sup> International Conference on Performance Evaluation Methodologies and TOOLS*, Pisa, Italy, 1-4.

LIN, C. Y, ABDEL-HAMID, T., AND SHERIF, J.S. 1997. Software engineering process simulation model (SEPS). *Journal of Systems and Software* 38, 263-277.

LOSAVIO, F., CHIRINOS, L., AND PEREZ, M.A. 2001. Quality models to design software architectures. In *Proceedings of the Technology of Object-Oriented Languages and Systems*, Zurich, Switzerland, 123-135.

MADACHY, R., A. 1994. A Software project dynamics model for process cost, schedule and risk assessment. Doctoral dissertation. University of Southern California, Los Angeles, California.

MADACHY, R.J. 1996. System Dynamics modeling of an inspection-based process. In *Proceedings of the 18<sup>th</sup> International Conference on Software Engineering (ICSE)*, Berlin, Germany, 376-386.

MADACHY, R. 2006. Simulation for business value and software process/product trade-off decisions. ACM SIGSOFT: ACM Special Interest Group on Software Engineering, In *proceedings of the 2006 International Workshop on Economics Driven Software Engineering Research*, 25-30.

MARINESCU, R., AND RATIU, D. 2004. Quantifying the quality of object-oriented design: the factor-strategy model. In *Proceedings of the 11<sup>th</sup> Working Conference on Reverse Engineering*, Delft, The Netherlands, 192-201.

MARTIN, R. , AND RAFFO, D. 2000. A Model of the software development process using both continuous and discrete models. *International Journal of Software Process Improvement and Practice* 5, 147-157.

MCCALL, J.A., RICHARDS, P.K., AND WALTERS, G.F. 1977. Factors in Software Quality. *National Technical Information Service*, 1, 2, 3, AD/A-049- 015/055.

MILLER, M.J., PULGAR-VIDAL, F., AND FERRIN, D.M. 2002. Achieving higher levels of CMMI maturity using simulation. In *Proceedings of the 34<sup>th</sup> Winter Simulation Conference*, San Diego, CA, 1473-1478.

MÜLLER, M., AND PFAHL, D. 2008. Simulation Methods. In *The Guide to Advanced Empirical Software Engineering*. J. Singer, F. Shull, D. Sjøberg, Eds, Springer-Verlag, London, 117-152.

NOPPEN, J., VAN DEN BROEK, P. , AND AKSIT, M. 2005. A model for quality optimization in software design processes. *In Net.Objectdays*, Erfurt, Germany, 529-541.

OREIZY, P., GORLICK, M.M., TAYLOR, R.N. , HEIMBIGNER, D., JOHNSON, G., MEDVIDOVIC, N., QUILICI, A., ROSENBLUM, D.S., AND WOLF, A.L. 1999. An architecture-based approach to self-adaptive software. *Journal of IEEE Intelligent Systems*, 14, 54-62.

PFAHL, D., AND LEBSANFT, K. 1999. Integration of system dynamics modeling with descriptive process modeling and goal-oriented measurement. *Journal of Systems and Software*, 46, 135-150.

PFAHL, D., AND RUHE, G. 2002. IMMoS: a methodology for integrated measurement, modeling and simulation. *Software Process: Improvement and Practice*, 7, 189-210.

PHILLIPS, J., AND YILMAZ, L. 2006. An agent-based simulation study of cooperative team behavior in software development with rational unified process. *In Proceedings of the Agent-Directed Simulation Symposium of the Spring Simulation Multi-conference*, Huntsville, Alabama, 73-80.

PODNAR, I. , AND MIKAC, B. 2001. Software maintenance process analysis using discrete-event simulation. In *Proceedings of the fifth European Conference on Software Maintenance and Reengineering*, Lisbon, Portugal, 192-195.

POWELL, A., MANDER, K., AND BROWN, D. 1999. Strategies for lifecycle concurrency and iteration- A system dynamics approach. *Journal of Systems and Software*, 46, 151-161.

PowerSim<sup>®</sup> corporation. 1993. <http://www.powersim.com/>.

RAFFO, D. 1996. Modeling software processes quantitatively and assessing the impact of potential process changes on process performance. Doctoral dissertation. University of Carnegie Mellon, Pittsburgh, Pennsylvania.

RAFFO, D., KALTIO, T., PARTRIDGE, D., PHALP, K., AND RAMIL, J, F. 1999. Empirical studies applied to software process models. *Journal of Empirical Software Engineering*, 4, 353-369.

RAFFO, D., M, VANDEVILLE, J.V., AND MARTIN, R, H. 1999. Software process simulation to achieve higher CMM levels. *Journal of Systems and Software*, 46, 163-172.

ROYCE, W. 1970. Managing the development of large software systems. In *Proceedings of IEEE WESCON, Los Angeles*, 1-9.

RUIZ, M., RAMOS, I., TORO, M. 2001. A simplified model of software project dynamics. *Journal of Systems and Software*, 59, 299-309.

RUS, I., COLLOFELLO, J., AND LAKEY, P. 1999. Software process simulation for reliability management. *Journal of Systems and Software*, 46, 173-182.

RUS, I. , NEU, H., AND MÜNCH, J. 2003. A systematic methodology for developing discrete event simulation models of software development processes. In *Proceedings of the 4<sup>th</sup> Workshop on Software Process Simulation and Modeling*, Portland, Oregon.

SALEHIE, M. AND TAHVILDARI, L. 2009. Self-adaptive software: landscape and research challenges. *Journal of ACM Transactions on Autonomous and Adaptive Systems*, 4, 1-40.

SARGENT, R.G. Verification and validation of simulation models. 1998. In *Proceedings of the 30<sup>th</sup> conference on Winter Simulation*, Washington, DC, 121-130.

SCHMECK, H. , MÜLLER-SCHLOER, C., ÇAKAR, E., MNIF, M. AND RICHTER, U. 2010. Adaptivity and self-organization in organic computing systems. *ACM Transactions On Autonomous and Adaptive Systems*, 5, 1-32.

SETAMANIT, S., WAKELAND, W., AND RAFFO, D. 2006. Planning and improving global software development process using simulation. ACM SIGSOFT: ACM Special Interest Group on Software Engineering, In *Proceedings of the 2006 International Workshop on Global software development for the practitioner*, 8-14.

Stella<sup>®</sup>/iThink<sup>®</sup> corporation. 1985. <http://www.iseesystems.com/>.

STERMAN, J.D. 1992. System dynamics modeling for project management. Technical Report. MIT System Dynamics Group, Cambridge, MA.

THOMAS, D., JOINER, A., LIN, W., LOWRY, M., AND PRESSBURGER, T. 2010. The unique aspects of simulation verification and validation. In *Proceedings of the 2010 IEEE Aerospace Conference*, Big Sky, MT.

THUENTE, D.J. 1991. Rapid simulation and software prototyping for the architectural Design of embedded multiprocessor systems. In *Proceedings of the 19<sup>th</sup> ACM Annual Conference on Computer Science*, San Antonio, Texas, 113-121.

VENSIM<sup>®</sup> corporation. 1985. <http://www.vensim.com/software.html>.

WAKELAND, W., SHERVAIS, S., AND RAFFO, D. 2004. Heuristic verification and validation of software process simulation models. In *Proceedings of the 5<sup>th</sup> International Workshop on Software Process Simulation and Modeling ProSim*, Edinburg, Scotland, UK, 113-119.

WERNICK, P., AND LEHMAN, M.M. 1999. Software process white box modeling for FEAST/1. *Journal of Systems and Software*, 46, 193-201.

WILLIFORD, J., AND CHANG, A. 1999. Modeling the FedExIT division: a system dynamics approach to strategic IT planning. *Journal of Systems and Software*, 46, 203-211.

XU, L., HENDRICKSON, S.A., HETTWER, E., ZIV, H., VAN DER HOEK, A., AND RICHARDSON, D.J. 2006. Towards supporting the architecture design process through evaluation of design alternatives. In *Proceedings of the International Symposium of Software Testing and Analysis*, Portland, Maine, 81-87.

XU, L.2008. Moda - Multiple Objective Decision Analysis: balancing quality attributes in software architectures. ACM Special Interest Group on Software Engineering: In *Proceedings of the 30<sup>th</sup> International Conference on Software Engineering*, Leipzig, Germany, 1019-1022.

YANG, J., HUANG, G., ZHU, W., CUI, X., AND MEI, H. 2009. Quality attributes tradeoff through adaptive architectures at runtime. *Journal of Systems and Software*, 82, 319-332.



YAU, S.S., YE, N., SARJOUGHIAN, H.S., HUANG, D., ROONTIVA, A., BAYDOGAN, M., AND MUQTISH, M.A. 2009. Toward development of adaptive service-based software systems. *IEEE Transactions on Services Computing*, 2, 247-260.

ZHANG, J., AND CHENG, B.H.C. 2007. Towards re-engineering legacy systems for assured dynamic adaptation. *International Workshop on Modeling in Software Engineering*, Minneapolis, MN, 10.

ZHANG, H. , KITCHENHAM, B., AND PFAHL, D. 2008. Software process simulation modeling: facts, trends and directions. In *Proceedings of the 15<sup>th</sup> Asia-Pacific Software Engineering Conference*, Beijing, China, 59-66.

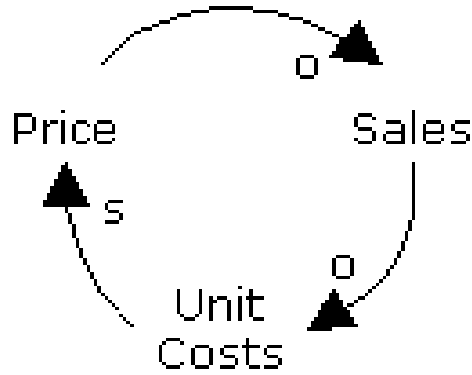
ZHANG, H., KITCHENHAM, B., AND JEFFERY, R. 2009. Qualitative vs. Quantitative software process simulation modeling: conversion and comparison. In *Proceedings of the 2009 Australian Software Engineering Conference*, Gold Coast, QLD, 345-354.

@RISK<sup>®</sup> corporation , 1987. <http://www.palisade.com/>.

## **Appendix A: System dynamics concepts**

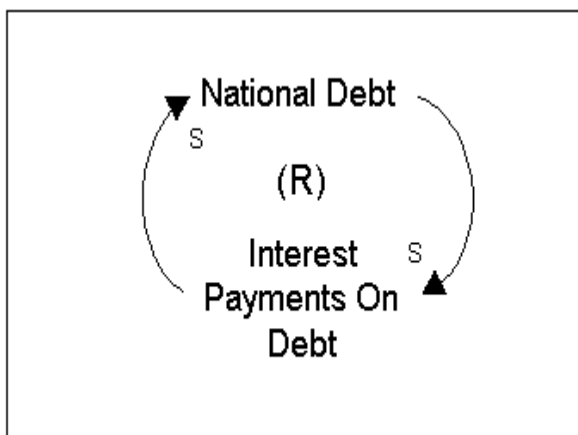
System Dynamics (SD) is a computer-based simulation modeling methodology developed at the Massachusetts Institute of Technology (MIT) in the 1950s as a tool for managers to analyze complex problems. It is used to model systems' behavior that changes over time.

System dynamics simulations are based on the principle of cause and effect relationships between outputs that both respond and influence inputs in a closed feedback loop. There are two types of feedback loops: positive and negative. Positive loops represent self-reinforcing systems that are either growing or declining. Negative loops represent goal-seeking systems that keep improving or get stabilized over time. The direction of causality between the variables in a feedback loop is represented by a minus or a positive sign at the head of each arrow. The positive sign indicates that the variable at the tail of each arrow causes a change in the variable at the head of each arrow in the same direction and vice versa. The positive sign is also represented by S (same direction) and the negative sign is represented by O (opposite direction). Figure 48 shows an example of a feedback loop where an increase in price leads to a decrease in sales.

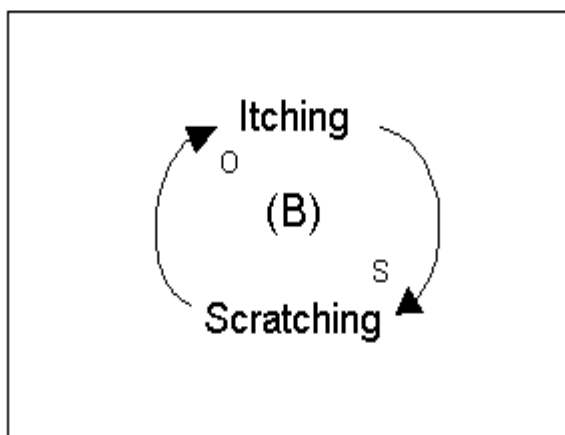


**Figure 48: A feedback loop that shows the relationships between price, sales, and unit costs**

The overall polarity (positive or negative) of the feedback loop is determined by multiplying all of its arrows' signs. If the resulting sign is negative, the feedback loop describes a balancing (B) or a counteracting (C) behavior to adjust and stabilize the status of a system. When the resulting sign is positive, the feedback loop represents a reinforcing (R) behavior towards the growth or the decline of a given system. Figure 49 illustrates a reinforcing feedback loop that describes the growth of the national debt due to the compounding of interest payments. Figure 50 presents a balancing loop that stabilizes the rate of itching by applying regular scratching.

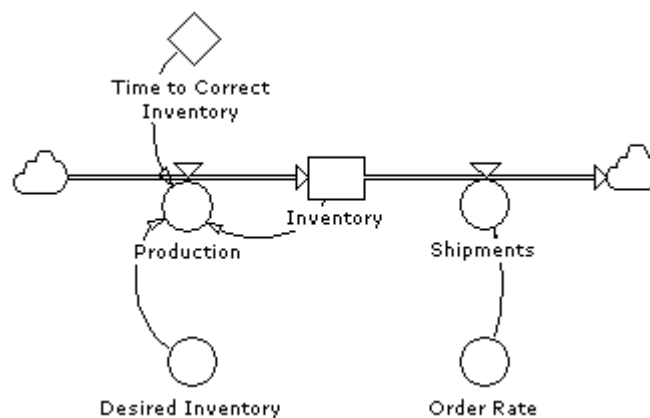


**Figure 49: Reinforcing feedback loop**



**Figure 50: Balancing feedback loop**

Besides the cause and effect relationships, dynamic systems' variables accumulate over time due to continuous flows of policies. Those accumulations are represented as levels such as an inventory level that increases due to increasing production flows. To control the production, a specific production rate is applied. Levels (rectangle symbol), flows (double arrows), and rates (valve symbol) are the main representations of variables in SD. In addition, rates and levels can be influenced by other external variables that are modeled as constants (diamond) or auxiliaries (circle) linked by information links (single arrows). Figure 51 represents a simple simulation model that helps us understand the interactions between the ordered merchandise from clients (order rate), the available goods (inventory), and production rate (production). Those interactions are computed numerically through a set of differential equations (e.g.;  $\text{Production} = (\text{Desired Inventory} - \text{Inventory}) / \text{Inventory Adjustment Time}$ ). The simulation can help business managers in estimating the optimum level of inventory to cover their future market demands.



**Figure 51: level-rate diagram example**

## **Appendix B: Simulation source code**

The following appendix illustrates the implementation code of the simulation in the Powersim<sup>®</sup> environment that uses a C-like syntax.

```

mainmodel Composant 1 {
  const AbstractAdaptationEffect {
    autotype Real
    init 0
  }
  const AbstractAdaptationU {
    autotype Real
    init 0
  }
  const AbstractAdaptEx {
    autotype Real
    init 0
  }
  level Abstraction {
    autotype Real
    autounit qual
    init ANAINitial
    inflow { autodef 'Change in abstraction' }
  }
  const ANA {
    autotype Real
    autounit qual
    init 20<<qual>>
  }
  const ANA_Change1 {
    autotype Real
    autounit qual
    init -5<<qual>>
  }
  const ANA_Change2 {
    autotype Real
    autounit qual
    init -10<<qual>>
  }
  const ANA_Change3 {
    autotype Real
    autounit qual
    init -15<<qual>>
  }
  const ANA_ExecuteTime {
    autotype Real
    autounit da
    init 1<<da>>
  }
  const ANAFirstEffect {
    autotype Real
    autounit qual
    init INITIF(((TIME >=(STARTTIME + (DAM_TimeChange1)+ 5 <<da>>)) OR ((TIME >=(STARTTIME +
(MFATimeChange1)+ 5 <<da>>)) OR ((TIME >=(STARTTIME + (MOA_TimeChange1)+ 5 <<da>>))
OR ((TIME >=(STARTTIME + (NOPTimeChange1)+ 5 <<da>>)))) AND (Effectiveness >=
EffectivenessReference)), Abstraction)
  }
  const ANAFirstExt {
    autotype Real
    autounit qual
    init INITIF(((TIME >=(STARTTIME + (DCC_TimeChange1) + 5 <<da>>)) OR ((TIME >=(STARTTIME +
(MFATimeChange1)+ 5 <<da>>)) OR ((TIME >=(STARTTIME + (NOPTimeChange1)+ 5 <<da>>))))
AND (Extendibility >= ExtendibilityReference)), Abstraction)
  }
  const ANAFirstUnderst {
    autotype Real
    autounit qual
    init INITIF(((TIME >=(STARTTIME + (DCC_TimeChange1) + 5 <<da>>)) OR ((TIME >=(STARTTIME +
(DAM_TimeChange1)+ 5 <<da>>)) OR ((TIME >=(STARTTIME + (CAMTimeChange1)+ 5 <<da>>))
OR ((TIME >=(STARTTIME + (DSCTimeChange1)+ 5 <<da>>)) OR ((TIME >=(STARTTIME +
(NOMTimeChange1)+ 5 <<da>>)))OR ((TIME >=(STARTTIME + (NOPTimeChange1)+ 5 <<da>>))))
  }
}

```

```

    AND (Understandability >= UnderstandReference)), Abstraction)
}
const ANAInitial {
  autotype Real
  autounit qual
  init INITIF(TIME = STARTTIME, ANA)
}
const ANATimeChange1 {
  autotype Real
  autounit da
  init 30<<da>>
}
const ANATimeChange2 {
  autotype Real
  autounit da
  init 60<<da>>
}
const ANATimeChange3 {
  autotype Real
  autounit da
  init 90<<da>>
}
const CAM {
  autotype Real
  autounit qual
  init 0.9<<qual>>
}
const CAM-FirstChange {
  autotype Real
  autounit qual
  init INITIF((((TIME >=(STARTTIME + (CISTimeChange1)+ 5 <<da>>)) OR ((TIME >=(STARTTIME +
    (DCC_TimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (DSCTimeChange1)+ 5 <<da>>))))
    AND (Reusability >= ReusabilityReference)), Cohesion)
}
const CAM_ExecuteTime {
  autotype Real
  autounit da
  init 1<<da>>
}
const CAMChange1 {
  autotype Real
  autounit qual
  init (-0.30)<<qual>>
}
const CAMChange2 {
  autotype Real
  autounit qual
  init (-0.5)<<qual>>
}
const CAMChange3 {
  autotype Real
  autounit qual
  init (-0.7)<<qual>>
}
const CAMFirstFunct {
  autotype Real
  autounit qual
  init INITIF((((TIME >=(STARTTIME + (CISTimeChange1)+ 5 <<da>>)) OR ((TIME >=(STARTTIME +
    (NOPTimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (DSCTimeChange1)+ 5 <<da>>)))
    OR ((TIME >=(STARTTIME + (NOHTimeChange1)+ 5 <<da>>)))) AND (Functionality >=
    FunctionalityReference)), Cohesion)
}
const CAMFirstUnderst {
  autotype Real
  autounit qual
  init INITIF((((TIME >=(STARTTIME + (DCC_TimeChange1) + 5 <<da>>)) OR ((TIME >=(STARTTIME +

```

```

(DAM_TimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (ANATimeChange1)+ 5 <<da>>)))
OR ((TIME >=(STARTTIME + (DSCTimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME +
(NOMTimeChange1)+ 5 <<da>>)))OR ((TIME >=(STARTTIME + (NOPTimeChange1)+ 5 <<da>>)))
AND (Understandability >= UnderstandReference)), Cohesion)
}
const CAMInitial {
  autotype Real
  autounit qual
  init INITIF(TIME = STARTTIME, CAM)
}
const CAMTimeChange1 {
  autotype Real
  autounit da
  init 30<<da>>
}
const CAMTimeChange2 {
  autotype Real
  autounit da
  init 60<<da>>
}
const CAMTimeChange3 {
  autotype Real
  autounit da
  init 90<<da>>
}
aux Change in abstraction {
  autotype Real
  autounit qual/da
  def ('Computed ANA'-Abstraction)/ANA_ExecuteTime
}
aux Change in cohesion {
  autotype Real
  autounit qual/da
  def ('Computed CAM'-Cohesion)/CAM_ExecuteTime
}
aux Change in Complexity {
  autotype Real
  autounit qual/da
  def ('Computed NOM'-Complexity)/NOM_ExecuteTime
}
aux Change in composition {
  autotype Real
  autounit qual/da
  def ('Computed MOA'-Composition)/MOA_ExecuteTime
}
aux Change in coupling {
  autotype Real
  autounit qual/da
  def ('Computed DCC'-Coupling)/DCC_ExecuteTime
}
aux Change in Design Size {
  autotype Real
  autounit qual/da
  def ('Computed DSC'-Design Size)/DSC_ExecuteTime
}
aux Change in Encapsulation {
  autotype Real
  autounit qual/da
  def ('Computed DAM'-Encapsulation)/DAM_ExecuteTime
}
aux Change in Hierarchies {
  autotype Real
  autounit qual/da
  def ('Computed NOH'-Hierarchies)/NOH_ExecuteTime
}
aux Change in inheritance {

```



```

    autotype Real
    autounit qual/da
    def ('Computed MFA'-Inheritance)/MFA_ExecuteTime
}
aux Change in Messaging {
    autotype Real
    autounit qual/da
    def ('Computed CIS'-Messaging)/CIS_ExecuteTime
}
aux Change in polymorphism {
    autotype Real
    autounit qual/da
    def ('Computed NOP'-Polymorphism)/NOP_ExecuteTime
}
const CheckedAbstraction {
    autotype Real
    init 0
}
const CheckedCohesion {
    autotype Real
    init 0
}
const CheckedComplexity {
    autotype Real
    init 0
}
const CheckedComposition {
    autotype Real
    init 0
}
const CheckedCoupling {
    autotype Real
    init 0
}
const CheckedDesignSize {
    autotype Real
    init 0
}
const CheckedEncapsulation {
    autotype Real
    init 0
}
const CheckedHierarchies {
    autotype Real
    init 0
}
const CheckedInheritance {
    autotype Real
    init 0
}
const CheckedMessaging {
    autotype Real
    init 0
}
const CheckedPolymo {
    autotype Real
    init 0
}
const CIS {
    autotype Real
    autounit qual
    init 30<<qual>>
}
const CIS-FirstChange {
    autotype Real
    autounit qual

```

```

init INITIF(((TIME >=(STARTTIME + (CAMTimeChange1)+ 5 <<da>>)) OR ((TIME >=(STARTTIME +
(DCC_TimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (DSCTimeChange1)+ 5 <<da>>))))
AND (Reusability >= ReusabilityReference)), Messaging)
}
const CIS_ExecuteTime {
  autotype Real
  autounit da
  init 1<<da>>
}
const CISChange1 {
  autotype Real
  autounit qual
  init -15<<qual>>
}
const CISChange2 {
  autotype Real
  autounit qual
  init -20<<qual>>
}
const CISChange3 {
  autotype Real
  autounit qual
  init -25<<qual>>
}
const CISFirstFunc {
  autotype Real
  autounit qual
  init INITIF(((TIME >=(STARTTIME + (CAMTimeChange1)+ 5 <<da>>)) OR ((TIME >=(STARTTIME +
(NOTimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (DSCTimeChange1)+ 5 <<da>>))))
OR ((TIME >=(STARTTIME + (NOHTimeChange1)+ 5 <<da>>)))) AND (Functionality >=
FunctionalityReference)), Messaging)
}
const CISInitial {
  autotype Real
  autounit qual
  init INITIF(TIME = STARTTIME, CIS)
}
const CISTimeChange1 {
  autotype Real
  autounit da
  init 30<<da>>
}
const CISTimeChange2 {
  autotype Real
  autounit da
  init 60<<da>>
}
const CISTimeChange3 {
  autotype Real
  autounit da
  init 90<<da>>
}
level Cohesion {
  autotype Real
  autounit qual
  init CAMInitial
  inflow { autodef 'Change in cohesion' }
}
const CohesionAdaptationFunc {
  autotype Real
  init 0
}
const CohesionAdaptationR {
  autotype Real
  init 0
}

```

```

const CohesionAdaptationU {
  autotype Real
  init 0
}
const ComplexAdaptation {
  autotype Real
  init 0
}
level Complexity {
  autotype Real
  autounit qual
  init NOMInitial
  inflow { autodef 'Change in Complexity' }
}
const ComposAdaptationEffect {
  autotype Real
  init 0
}
const ComposAdaptationF {
  autotype Real
  init 0
}
level Composition {
  autotype Real
  autounit qual
  init MOAInitial
  inflow { autodef 'Change in composition' }
}
aux Computed ANA {
  autotype Real
  autounit qual
  def ANA +
    IF(CheckedAbstraction =1, IF( (TIME >=(STARTTIME + ANATimeChange1)) AND (TIME <
      (STARTTIME + ANATimeChange2)),ANA_Change1, IF( (TIME >=(STARTTIME + ANATimeChange1)
      ) AND (ANA_Change2 = 0 <<qual>>) AND (ANA_Change3 = 0 <<qual>>),ANA_Change1,
      IF((TIME >=(STARTTIME + ANATimeChange2)) AND (TIME <(STARTTIME +
      ANATimeChange3)),ANA_Change2, IF((TIME >=(STARTTIME + ANATimeChange2)) AND
      (ANA_Change1 = 0 <<qual>>) AND (ANA_Change3 = 0 <<qual>>),ANA_Change2,IF( (TIME >=
      (STARTTIME + ANATimeChange1)) AND (TIME <(STARTTIME + ANATimeChange3)) AND
      (ANA_Change2 = 0 <<qual>>),ANA_Change1, IF((TIME >=(STARTTIME +ANATimeChange2)) AND
      (ANA_Change3 = 0 <<qual>>),ANA_Change2,
      IF( (TIME >=(STARTTIME + ANATimeChange3)),ANA_Change3))))),
    IF(AbtractAdaptationU =1,
      IF( ((TIME >=(STARTTIME + (DCC_TimeChange1 + 5<<da>>))) AND
      (Understandability < UnderstandReference)) , ((-3.03*Understandability) + Encapsulation - Coupling
      + Cohesion - Polymorphism - Complexity - 'Design Size'),
      IF( (TIME >=(STARTTIME + (DCC_TimeChange1)+ 5 <<da>>)) AND
      (Understandability >= UnderstandReference), ((- ANA) + ANAFirstUnderst),
      IF( ((TIME >=(STARTTIME + (DCC_TimeChange2 + 5<<da>>))) AND
      (Understandability < UnderstandReference)) , ((-3.03*Understandability) + Encapsulation - Coupling
      + Cohesion - Polymorphism - Complexity - 'Design Size'),
      IF( (TIME >=(STARTTIME + (DCC_TimeChange2)+ 5 <<da>>)) AND
      (Understandability >= UnderstandReference), ((- ANA) + ANAFirstUnderst), IF( ((TIME >=
      (STARTTIME + (DCC_TimeChange3 + 5<<da>>))) AND (Understandability < UnderstandReference))
      , ((-3.03*Understandability) + Encapsulation - Coupling + Cohesion - Polymorphism - Complexity -
      'Design Size'),
      IF( (TIME >=(STARTTIME + (DCC_TimeChange3)+ 5 <<da>>)) AND
      (Understandability >= UnderstandReference), ((- ANA) + ANAFirstUnderst), IF( ((TIME >=
      (STARTTIME + (DCC_TimeChange1 + 5<<da>>))) AND (Understandability< UnderstandReference))
      ,((-3.03*Understandability) + Encapsulation - Coupling + Cohesion - Polymorphism - Complexity -
      'Design Size'),
      IF( ((TIME >=(STARTTIME + (CAMTimeChange1 + 5<<da>>))) AND
      (Understandability >= UnderstandReference)) , ((-3.03*Understandability) + Encapsulation - Coupling
      + Cohesion - Polymorphism - Complexity - 'Design Size'),
      IF( (TIME >=(STARTTIME + (CAMTimeChange1)+ 5 <<da>>)) AND

```















```

Encapsulation - Composition - Inheritance - Polymorphism))))))))))))))))))))))))))))))))))))))))))
}
aux Computed CAM {
  autotype Real
  autounit qual
  def CAM +
    IF(CheckedCohesion =1, IF( (TIME >=(STARTTIME + CAMTimeChange1)) AND (TIME <
    (STARTTIME + CAMTimeChange2)),CAMChange1,IF( (TIME >=(STARTTIME + CAMTimeChange1))
    AND (CAMChange2 = 0 <<qual>>) AND (CAMChange3 = 0 <<qual>>),CAMChange1,
      IF((TIME >=(STARTTIME + CAMTimeChange2)) AND (TIME <(STARTTIME +
    CAMTimeChange3)),CAMChange2, IF((TIME >=(STARTTIME + CAMTimeChange2)) AND
    (CAMChange1 = 0 <<qual>>) AND (CAMChange3 = 0 <<qual>>),CAMChange2,IF( (TIME >=
    (STARTTIME + CAMTimeChange1)) AND (TIME <(STARTTIME + CAMTimeChange3)) AND
    (CAMChange2 = 0 <<qual>>),CAMChange1, IF((TIME >=(STARTTIME + CAMTimeChange2)) AND
    (CAMChange3 = 0 <<qual>>),CAMChange2,
      IF( (TIME >=(STARTTIME + CAMTimeChange3)),CAMChange3))))),
    IF(CohesionAdaptationR =1, IF( ((TIME >=(STARTTIME + (CISTimeChange1 + 5<<da>>))) AND
    (Reusability < ReusabilityReference)) , (4*Reusability + Coupling + (-2*Messaging) + (-2*Design
    Size')),
      IF( (TIME >=(STARTTIME + (CISTimeChange1)+ 5 <<da>>)) AND (Reusability >=
    ReusabilityReference), ((- CAM) + 'CAM-FirstChange'),
      IF( ((TIME >=(STARTTIME + (CISTimeChange2 + 5<<da>>))) AND (Reusability <
    ReusabilityReference)) , (4*Reusability + Coupling + (-2*Messaging) + (-2*Design Size')),
      IF( (TIME >=(STARTTIME + (CISTimeChange2)+ 5 <<da>>)) AND (Reusability >=
    ReusabilityReference), ((- CAM) + 'CAM-FirstChange'), IF( ((TIME >=(STARTTIME +
    (CISTimeChange3 + 5<<da>>))) AND (Reusability < ReusabilityReference)) , (4*Reusability +
    Coupling + (-2*Messaging) + (-2*Design Size')),
      IF( (TIME >=(STARTTIME + (CISTimeChange3)+ 5 <<da>>)) AND (Reusability >=
    ReusabilityReference), ((- CAM) + 'CAM-FirstChange'), IF( ((TIME >=(STARTTIME +
    (CISTimeChange1 + 5<<da>>))) AND (Reusability < ReusabilityReference)) , (4*Reusability +
    Coupling + (-2*Messaging) + (-2*Design Size')),
      IF( ((TIME >=(STARTTIME + (DSCTimeChange1 + 5<<da>>))) AND (Reusability <
    ReusabilityReference)) , (4*Reusability + Coupling + (-2*Messaging) + (-2*Design Size')),
      IF( (TIME >=(STARTTIME + (DSCTimeChange1)+ 5 <<da>>)) AND (Reusability >=
    ReusabilityReference), ((- CAM) + 'CAM-FirstChange'),
      IF( ((TIME >=(STARTTIME + (DSCTimeChange2 + 5<<da>>))) AND (Reusability <
    ReusabilityReference)) , (4*Reusability + Coupling + (-2*Messaging) + (-2*Design Size')),
      IF( (TIME >=(STARTTIME + (DSCTimeChange2)+ 5 <<da>>)) AND (Reusability >=
    ReusabilityReference), ((- CAM) + 'CAM-FirstChange'), IF( ((TIME >=(STARTTIME +
    (DSCTimeChange3+ 5<<da>>))) AND (Reusability < ReusabilityReference)) , (4*Reusability +
    Coupling + (-2*Messaging) + (-2*Design Size')),
      IF( (TIME >=(STARTTIME + (DSCTimeChange3)+ 5 <<da>>)) AND (Reusability >=
    ReusabilityReference), ((- CAM) + 'CAM-FirstChange'), IF( ((TIME >=(STARTTIME +
    (DSCTimeChange1 + 5<<da>>))) AND (Reusability < ReusabilityReference)) , (4*Reusability +
    Coupling + (-2*Messaging) + (-2*Design Size')),
      IF( ((TIME >=(STARTTIME + (DCC_TimeChange1 + 5<<da>>))) AND (Reusability <
    ReusabilityReference)) , (4*Reusability + Coupling + (-2*Messaging) + (-2*Design Size')),
      IF( (TIME >=(STARTTIME + (DCC_TimeChange1)+ 5 <<da>>)) AND (Reusability >=
    ReusabilityReference), ((- CAM) + 'CAM-FirstChange'),
      IF( ((TIME >=(STARTTIME + (DCC_TimeChange2 + 5<<da>>))) AND (Reusability <
    ReusabilityReference)) , (4*Reusability + Coupling + (-2*Messaging) + (-2*Design Size')),
      IF( (TIME >=(STARTTIME + (DCC_TimeChange2)+ 5 <<da>>)) AND (Reusability >=
    ReusabilityReference), ((- CAM) + 'CAM-FirstChange'), IF( ((TIME >=(STARTTIME +
    (DCC_TimeChange3 + 5<<da>>))) AND (Reusability < ReusabilityReference)) , (4*Reusability +
    Coupling + (-2*Messaging) + (-2*Design Size')),
      IF( (TIME >=(STARTTIME + (DCC_TimeChange3)+ 5 <<da>>)) AND (Reusability >=
    ReusabilityReference), ((- CAM) + 'CAM-FirstChange'), IF( ((TIME >=(STARTTIME +
    (DCC_TimeChange1 + 5<<da>>))) AND (Reusability < ReusabilityReference)) , (4*Reusability +
    Coupling + (-2*Messaging) + (-2*Design Size'))))))))))))))))))))))))))))))))))))))))))))
    IF(CohesionAdaptationU =1,IF( ((TIME >=(STARTTIME + (ANATimeChange1 + 5<<da>>))) AND
    (Understandability < UnderstandReference)) , ((3.03)*Understandability + Abstraction -
    Encapsulation+ Coupling + Polymorphism + Complexity + 'Design Size'),
      IF( (TIME >=(STARTTIME + (ANATimeChange1)+ 5 <<da>>)) AND
    (Understandability < UnderstandReference), ((- CAM) + CAMFirstUnderst),
      IF( ((TIME >=(STARTTIME + (ANATimeChange2 + 5<<da>>))) AND
    (Understandability < UnderstandReference)) , ((3.03)*Understandability + Abstraction -

```











```

(NOTimeChange1 + 5<<da>>)) AND (Functionality < FunctionalityReference)) , ((8.33*Functionality)
- (1.83* Polymorphism) - (1.83* Messaging) - (1.83*Design Size) - (1.83*Hierarchies)),
    IF( ((TIME >=(STARTTIME + (DSCTimeChange1 + 5<<da>>))) AND (Functionality >
= FunctionalityReference)) , ((8.33*Functionality) - (1.83* Polymorphism) - (1.83* Messaging) - (1.83*
'Design Size) - (1.83*Hierarchies)),
        IF( (TIME >=(STARTTIME + (DSCTimeChange1)+ 5 <<da>>)) AND (Functionality >=
FunctionalityReference), ((- CAM) + CAMFirstFunct),
            IF( ((TIME >=(STARTTIME + (DSCTimeChange2 + 5<<da>>))) AND (Functionality >=
FunctionalityReference)) , ((8.33*Functionality) - (1.83* Polymorphism) - (1.83* Messaging) - (1.83*
'Design Size) - (1.83*Hierarchies)),
                IF( (TIME >=(STARTTIME + (DSCTimeChange2+ 5 <<da>>))) AND (Functionality >=
FunctionalityReference), ((- CAM) + CAMFirstFunct), IF( ((TIME >=(STARTTIME +
(DSCTimeChange3+ 5<<da>>))) AND (Functionality < FunctionalityReference)) , ((8.33*Functionality)
- (1.83* Polymorphism) - (1.83* Messaging) - (1.83*Design Size) - (1.83*Hierarchies)),
                    IF( (TIME >=(STARTTIME + (DSCTimeChange3+ 5 <<da>>))) AND (Functionality >=
FunctionalityReference), ((- CAM) + CAMFirstFunct), IF( ((TIME >=(STARTTIME +
(DSCTimeChange1 + 5<<da>>))) AND (Functionality < FunctionalityReference)) , ((8.33*Functionality)
- (1.83* Polymorphism) - (1.83* Messaging) - (1.83*Design Size) - (1.83*Hierarchies)),
                        IF( ((TIME >=(STARTTIME + (NOHTimeChange1 + 5<<da>>))) AND (Functionality <
FunctionalityReference)) , ((8.33*Functionality) - (1.83* Polymorphism) - (1.83* Messaging) - (1.83*
'Design Size) - (1.83*Hierarchies)),
                            IF( (TIME >=(STARTTIME + (NOHTimeChange1)+ 5 <<da>>)) AND (Functionality >=
FunctionalityReference), ((- CAM) + CAMFirstFunct),
                                IF( ((TIME >=(STARTTIME + (NOHTimeChange2 + 5<<da>>))) AND (Functionality <
FunctionalityReference)) , ((8.33*Functionality) - (1.83* Polymorphism) - (1.83* Messaging) - (1.83*
'Design Size) - (1.83*Hierarchies)),
                                    IF( (TIME >=(STARTTIME + (NOHTimeChange2)+ 5 <<da>>)) AND (Functionality >=
FunctionalityReference), ((- CAM) + CAMFirstFunct), IF( ((TIME >=(STARTTIME +
(NOHTimeChange3 + 5<<da>>))) AND (Functionality < FunctionalityReference)) , ((8.33*
Functionality) - (1.83* Polymorphism) - (1.83* Messaging) - (1.83*Design Size) - (1.83*Hierarchies)),
                                        IF( (TIME >=(STARTTIME + (NOHTimeChange3)+ 5 <<da>>)) AND (Functionality >=
FunctionalityReference), ((- CAM) + CAMFirstFunct), IF( ((TIME >=(STARTTIME +
(NOHTimeChange1 + 5<<da>>))) AND (Functionality < FunctionalityReference)) , ((8.33*Functionality)
- (1.83* Polymorphism) - (1.83* Messaging) - (1.83*Design Size) - (1.83*Hierarchies))))))))))))))))))
))))))))))
}
aux Computed CIS {
    autotype Real
    autounit qual
    def CIS +
        IF(CheckedMessaging =1, IF( (TIME >=(STARTTIME + CiSTimeChange1)) AND (TIME <
(STARTTIME + CiSTimeChange2)),CiSChange1, IF( (TIME >=(STARTTIME + CiSTimeChange1))
AND (CiSChange2 = 0 <<qual>>) AND (CiSChange3 = 0 <<qual>>),CiSChange1,
            IF( (TIME >=(STARTTIME + CiSTimeChange2)) AND (TIME <(STARTTIME +
CiSTimeChange3)),CiSChange2,IF((TIME >=(STARTTIME + CiSTimeChange2)) AND (CiSChange1
= 0 <<qual>>) AND (CiSChange3 = 0 <<qual>>),CiSChange2, IF( (TIME >=(STARTTIME +
CiSTimeChange1)) AND (TIME <(STARTTIME + CiSTimeChange3)) AND (CiSChange2 = 0 <<qual>>
),CiSChange1, IF((TIME >=(STARTTIME + CiSTimeChange2)) AND (CiSChange3 = 0 <<qual>>),
CiSChange2,
                IF( (TIME >=(STARTTIME + CiSTimeChange3)),CiSChange3 ))))))).

IF(MessagingAdaptationR =1, IF( ((TIME >=(STARTTIME + (CAMTimeChange1 + 5<<da>>))) AND
(Reusability < ReusabilityReference)) , ((2*Reusability) + (0.5*Coupling) + (-0.5*Cohesion) -Design
Size'),
    IF( (TIME >=(STARTTIME + (CAMTimeChange1)+ 5 <<da>>)) AND (Reusability >=
ReusabilityReference), ((- CIS) + 'CIS-FirstChange'),
        IF( ((TIME >=(STARTTIME + (CAMTimeChange2 + 5<<da>>))) AND (Reusability <
ReusabilityReference)) , ((2*Reusability) + (0.5*Coupling) + (-0.5*Cohesion) -Design Size'),
            IF( (TIME >=(STARTTIME + (CAMTimeChange2)+ 5 <<da>>)) AND (Reusability >=
ReusabilityReference), ((- CIS) + 'CIS-FirstChange'), IF( ((TIME >=(STARTTIME +
(CAMTimeChange3+ 5<<da>>))) AND (Reusability < ReusabilityReference)) , ((2*Reusability) + (0.5*
Coupling) + (-0.5*Cohesion) -Design Size'),
                IF( (TIME >=(STARTTIME + (CAMTimeChange3)+ 5 <<da>>)) AND (Reusability >=
ReusabilityReference), ((- CIS) + 'CIS-FirstChange'), IF( ((TIME >=(STARTTIME +
(CAMTimeChange1 + 5<<da>>))) AND (Reusability < ReusabilityReference)) , ((2*Reusability) + (0.
5*Coupling) + (-0.5*Cohesion) -Design Size'),

```





```

IF( ((TIME >=(STARTTIME + (DSCTimeChange2 + 5 <<da>>))) AND (Functionality >=
FunctionalityReference)) , ((4.54*Functionality) - (0.54* Cohesion) - Polymorphism - 'Design Size' -
Hierarchies),
IF( (TIME >=(STARTTIME + (DSCTimeChange2+ 5 <<da>>))) AND (Functionality >=
FunctionalityReference), ((- CIS) + CISFirstFunc), IF( ((TIME >=(STARTTIME + (DSCTimeChange3+
5<<da>>))) AND (Functionality < FunctionalityReference)) , ((4.54*Functionality) - (0.54* Cohesion) -
Polymorphism - 'Design Size' - Hierarchies),
IF( (TIME >=(STARTTIME + (DSCTimeChange3+ 5 <<da>>))) AND (Functionality >=
FunctionalityReference), ((- CIS) + CISFirstFunc), IF( ((TIME >=(STARTTIME + (DSCTimeChange1
+ 5<<da>>))) AND (Functionality< FunctionalityReference)) ,((4.54*Functionality) - (0.54* Cohesion) -
Polymorphism - 'Design Size' - Hierarchies),
IF( ((TIME >=(STARTTIME + (NOHTimeChange1 + 5<<da>>))) AND (Functionality <
FunctionalityReference)) , ((4.54*Functionality) - (0.54* Cohesion) - Polymorphism - 'Design Size' -
Hierarchies),
IF( (TIME >=(STARTTIME + (NOHTimeChange1)+ 5 <<da>>)) AND (Functionality >=
FunctionalityReference), ((- CIS) + CISFirstFunc),
IF( ((TIME >=(STARTTIME + (NOHTimeChange2 + 5<<da>>))) AND (Functionality <
FunctionalityReference)) , ((4.54*Functionality) - (0.54* Cohesion) - Polymorphism - 'Design Size' -
Hierarchies),
IF( (TIME >=(STARTTIME + (NOHTimeChange2)+ 5 <<da>>)) AND (Functionality >=
FunctionalityReference), ((- CIS) + CISFirstFunc), IF( ((TIME >=(STARTTIME + (NOHTimeChange3
+ 5<<da>>))) AND (Functionality < FunctionalityReference)) , ((4.54*Functionality) - (0.54* Cohesion)
- Polymorphism - 'Design Size' - Hierarchies),
IF( (TIME >=(STARTTIME + (NOHTimeChange3)+ 5 <<da>>)) AND (Functionality >=
FunctionalityReference), ((- CIS) + CISFirstFunc), IF( ((TIME >=(STARTTIME + (NOHTimeChange1
+ 5<<da>>))) AND (Functionality< FunctionalityReference)) ,((4.54*Functionality) - (0.54* Cohesion) -
Polymorphism - 'Design Size' - Hierarchies))))))))))))))))))))))))))))))))))))))))))))))

```

```

}
aux Computed DAM {
autotype Real
autounit qual
def DAM +
IF(CheckedEncapsulation =1, IF( (TIME >=(STARTTIME + DAM_TimeChange1)) AND (TIME <
(STARTTIME + DAM_TimeChange2)),DAM_Change1, IF( (TIME >=(STARTTIME +
DAM_TimeChange1)) AND (DAM_Change2 = 0 <<qual>>) AND (DAM_Change3 = 0 <<qual>>),
DAM_Change1,
IF((TIME >=(STARTTIME + DAM_TimeChange2)) AND (TIME <(STARTTIME +
DAM_TimeChange3)),DAM_Change2, IF((TIME >=(STARTTIME + DAM_TimeChange2)) AND
(DAM_Change1 = 0 <<qual>>) AND (DAM_Change3 = 0 <<qual>>),DAM_Change2,IF( (TIME >=
(STARTTIME + DAM_TimeChange1)) AND (TIME <(STARTTIME + DAM_TimeChange3)) AND
(DAM_Change2 = 0 <<qual>>),DAM_Change1, IF((TIME >=(STARTTIME +DAM_TimeChange2))
AND (DAM_Change3 = 0 <<qual>>),DAM_Change2,
IF( (TIME >=(STARTTIME + DAM_TimeChange3)),DAM_Change3))))))

IF(EncapsuAdaptationF =1,
IF( ((TIME >=(STARTTIME + (DCC_TimeChange1 + 5<<da>>))) AND (Flexibility <
FlexibilityReference)) , ((4*Flexibility) + Coupling - (2*Composition) - (2*Polymorphism)),
IF( (TIME >=(STARTTIME + (DCC_TimeChange1)+ 5 <<da>>)) AND (Flexibility >=
FlexibilityReference), ((- DAM) + DAMFirstFlexibility),
IF( ((TIME >=(STARTTIME + (DCC_TimeChange2 + 5<<da>>))) AND (Flexibility <
FlexibilityReference)) , ((4*Flexibility) + Coupling - (2*Composition) - (2*Polymorphism)),
IF( (TIME >=(STARTTIME + (DCC_TimeChange2)+ 5 <<da>>)) AND (Flexibility >=
FlexibilityReference), ((- DAM) + DAMFirstFlexibility), IF( ((TIME >=(STARTTIME +
(DCC_TimeChange3 + 5<<da>>))) AND (Flexibility < FlexibilityReference)) , ((4*Flexibility) + Coupling
- (2*Composition) - (2*Polymorphism)),
IF( (TIME >=(STARTTIME + (DCC_TimeChange3)+ 5 <<da>>)) AND (Flexibility >=
FlexibilityReference), ((- DAM) + DAMFirstFlexibility), IF( ((TIME >=(STARTTIME +
(DCC_TimeChange1 + 5<<da>>))) AND (Flexibility< FlexibilityReference)) ,((4*Flexibility) + Coupling
- (2*Composition) - (2*Polymorphism)),
IF( (TIME >=(STARTTIME + (MOA_TimeChange1 + 5<<da>>))) AND (Flexibility <
FlexibilityReference)) , ((4*Flexibility) + Coupling - (2*Composition) - (2*Polymorphism)),
IF( (TIME >=(STARTTIME + (MOA_TimeChange1)+ 5 <<da>>)) AND (Flexibility >=
FlexibilityReference), ((- DAM) + DAMFirstFlexibility),
IF( ((TIME >=(STARTTIME + (MOA_TimeChange2 + 5<<da>>))) AND (Flexibility <

```















```

}
aux Computed DCC {
  autotype Real
  autounit qual
  def DCC +
    IF(CheckedCoupling =1, IF( (TIME >=(STARTTIME + DCC_TimeChange1)) AND (TIME <
    (STARTTIME + DCC_TimeChange2)),DCC_Change1, IF( (TIME >=(STARTTIME +
    DCC_TimeChange1)) AND (DCC_Change2 = 0 <<qual>>) AND (DCC_Change3 = 0 <<qual>>),
    DCC_Change1,
      IF((TIME >=(STARTTIME + DCC_TimeChange2)) AND (TIME <(STARTTIME +
    DCC_TimeChange3)),DCC_Change2, IF((TIME >=(STARTTIME + DCC_TimeChange2)) AND
    (DCC_Change1 = 0 <<qual>>) AND (DCC_Change3 = 0 <<qual>>),DCC_Change2,IF( (TIME >=
    (STARTTIME + DCC_TimeChange1)) AND (TIME <(STARTTIME + DCC_TimeChange3)) AND
    (DCC_Change2 = 0 <<qual>>),DCC_Change1, IF((TIME >=(STARTTIME + DCC_TimeChange2))
    AND (DCC_Change3 = 0 <<qual>>),DCC_Change2,
      IF( (TIME >=(STARTTIME + DCC_TimeChange3)),DCC_Change3)))))))).

IF(CouplingAdaptationR =1, IF( ((TIME >=(STARTTIME + (CISTimeChange1 + 5<<da>>))) AND
(Reusability < ReusabilityReference)) , -((-4*Reusability) + Cohesion + (2*Messaging) + (2*Design
Size')),
  IF( (TIME >=(STARTTIME + (CISTimeChange1)+ 5 <<da>>)) AND (Reusability >=
ReusabilityReference), ((- DCC) + 'DCC-FirstChange'),
  IF( ((TIME >=(STARTTIME + (CISTimeChange2 + 5<<da>>))) AND (Reusability <
ReusabilityReference)) , -((-4*Reusability) + Cohesion + (2*Messaging) + (2*Design Size')),
  IF( (TIME >=(STARTTIME + (CISTimeChange2)+ 5 <<da>>)) AND (Reusability >=
ReusabilityReference), ((- DCC) + 'DCC-FirstChange'), IF( ((TIME >=(STARTTIME +
(CISTimeChange3 + 5<<da>>))) AND (Reusability < ReusabilityReference)) , -((-4*Reusability) +
Cohesion + (2*Messaging) + (2*Design Size')),
  IF( (TIME >=(STARTTIME + (CISTimeChange3)+ 5 <<da>>)) AND (Reusability >=
ReusabilityReference), ((- DCC) + 'DCC-FirstChange'), IF( ((TIME >=(STARTTIME +
(CISTimeChange1 + 5<<da>>))) AND (Reusability < ReusabilityReference)) , -((-4*Reusability) +
Cohesion + (2*Messaging) + (2*Design Size')),
  IF( ((TIME >=(STARTTIME + (DSCTimeChange1 + 5<<da>>))) AND (Reusability <
ReusabilityReference)) , -((-4*Reusability) + Cohesion + (2*Messaging) + (2*Design Size')),
  IF( (TIME >=(STARTTIME + (DSCTimeChange1)+ 5 <<da>>)) AND (Reusability >=
ReusabilityReference), ((- DCC) + 'DCC-FirstChange'),
  IF( ((TIME >=(STARTTIME + (DSCTimeChange2 + 5<<da>>))) AND (Reusability <
ReusabilityReference)) , -((-4*Reusability) + Cohesion + (2*Messaging) + (2*Design Size')),
  IF( (TIME >=(STARTTIME + (DSCTimeChange2)+ 5 <<da>>)) AND (Reusability >=
ReusabilityReference), ((- DCC) + 'DCC-FirstChange'), IF( ((TIME >=(STARTTIME +
(DSCTimeChange3 + 5<<da>>))) AND (Reusability < ReusabilityReference)) , -((-4*Reusability) +
Cohesion + (2*Messaging) + (2*Design Size')),
  IF( (TIME >=(STARTTIME + (DSCTimeChange3)+ 5 <<da>>)) AND (Reusability >=
ReusabilityReference), ((- DCC) + 'DCC-FirstChange'), IF( ((TIME >=(STARTTIME +
(DSCTimeChange1 + 5<<da>>))) AND (Reusability < ReusabilityReference)) , -((-4*Reusability) +
Cohesion + (2*Messaging) + (2*Design Size')),
  IF( ((TIME >=(STARTTIME + (CAMTimeChange1 + 5<<da>>))) AND (Reusability <
ReusabilityReference)) , -((-4*Reusability) + Cohesion + (2*Messaging) + (2*Design Size')),
  IF( (TIME >=(STARTTIME + (CAMTimeChange1)+ 5 <<da>>)) AND (Reusability >=
ReusabilityReference), ((- DCC) + 'DCC-FirstChange'),
  IF( ((TIME >=(STARTTIME + (CAMTimeChange2 + 5<<da>>))) AND (Reusability <
ReusabilityReference)) , -((-4*Reusability) + Cohesion + (2*Messaging) + (2*Design Size')),
  IF( (TIME >=(STARTTIME + (CAMTimeChange2)+ 5 <<da>>)) AND (Reusability >=
ReusabilityReference), ((- DCC) + 'DCC-FirstChange'), IF( ((TIME >=(STARTTIME +
(CAMTimeChange3 + 5<<da>>))) AND (Reusability < ReusabilityReference)) , -((-4*Reusability) +
Cohesion + (2*Messaging) + (2*Design Size')),
  IF( (TIME >=(STARTTIME + (CAMTimeChange3)+ 5 <<da>>)) AND (Reusability >=
ReusabilityReference), ((- DCC) + 'DCC-FirstChange'), IF( ((TIME >=(STARTTIME +
(CAMTimeChange1 + 5<<da>>))) AND (Reusability < ReusabilityReference)) , -((-4*Reusability) +
Cohesion + (2*Messaging) + (2*Design Size')))))))))))))))))))))))))))))))))))))))))))))).
IF(CouplingAdaptationF =1,IF( ((TIME >=(STARTTIME + (DAM_TimeChange1 + 5<<da>>))) AND
(Flexibility < FlexibilityReference)) , -((-4*Flexibility) + Encapsulation + (2*Composition) + (2*
Polymorphism)),
  IF( (TIME >=(STARTTIME + (DAM_TimeChange1)+ 5 <<da>>)) AND (Flexibility >=
FlexibilityReference), ((- DCC) + DCCFirstFlexibility),

```













```

aux Computed DSC {
  autotype Real
  autounit qual
  def DSC +
    IF(CheckedDesignSize =1, IF( (TIME >=(STARTTIME + DSCTimeChange1)) AND (TIME <
    (STARTTIME + DSCTimeChange2)),DSCChange1,IF( (TIME >=(STARTTIME + DSCTimeChange1))
    AND (DSCChange2 = 0 <<qual>>) AND (DSCChange3 = 0 <<qual>>),DSCChange1,
      IF((TIME >=(STARTTIME + DSCTimeChange2)) AND (TIME <(STARTTIME +
    DSCTimeChange3)),DSCChange2,IF((TIME >=(STARTTIME + DSCTimeChange2)) AND
    (DSCChange1 = 0 <<qual>>) AND (DSCChange3 = 0 <<qual>>),DSCChange2, IF( (TIME >=
    (STARTTIME + DSCTimeChange1)) AND (TIME <(STARTTIME + DSCTimeChange3)) AND
    (DSCChange2 = 0 <<qual>>),DSCChange1, IF((TIME >=(STARTTIME + DSCTimeChange2)) AND
    (DSCChange3= 0 <<qual>>),DSCChange2,
      IF( (TIME >=(STARTTIME + DSCTimeChange3)),DSCChange3))))),
    IF(DesignSizeAdaptationR =1,IF( ((TIME >=(STARTTIME + (CAMTimeChange1 + 5<<da>>))) AND
    (Reusability < ReusabilityReference)) , ((2*Reusability) + (0.5*Coupling) + (-0.5*Cohesion) -
    Messaging),
      IF( (TIME >=(STARTTIME + (CAMTimeChange1)+ 5 <<da>>)) AND (Reusability >=
    ReusabilityReference), ((- DSC) + 'DSC-FirstChange'),
      IF( ((TIME >=(STARTTIME + (CAMTimeChange2 + 5<<da>>))) AND (Reusability <
    ReusabilityReference)) , ((2*Reusability) + (0.5*Coupling) + (-0.5*Cohesion) -Messaging),
      IF( (TIME >=(STARTTIME + (CAMTimeChange2)+ 5 <<da>>)) AND (Reusability >=
    ReusabilityReference), ((- DSC) + 'DSC-FirstChange'), IF( ((TIME >=(STARTTIME +
    (CAMTimeChange3+ 5<<da>>))) AND (Reusability < ReusabilityReference)) , ((2*Reusability) + (0.5*
    Coupling) + (-0.5*Cohesion) -Messaging),
      IF( (TIME >=(STARTTIME + (CAMTimeChange3)+ 5 <<da>>)) AND (Reusability >=
    ReusabilityReference), ((- DSC) + 'DSC-FirstChange'), IF( ((TIME >=(STARTTIME +
    (CAMTimeChange1 + 5<<da>>))) AND (Reusability < ReusabilityReference)) , ((2*Reusability) + (0.
    5*Coupling) + (-0.5*Cohesion) -Messaging),
      IF( ((TIME >=(STARTTIME + (CISTimeChange1+ 5<<da>>))) AND (Reusability <
    ReusabilityReference)) , ((2*Reusability) + (0.5*Coupling) + (-0.5*Cohesion) -Messaging),
      IF( (TIME >=(STARTTIME + (CISTimeChange1)+ 5 <<da>>)) AND (Reusability >=
    ReusabilityReference), ((- DSC) + 'DSC-FirstChange'),
      IF( ((TIME >=(STARTTIME + (CISTimeChange2 + 5<<da>>))) AND (Reusability <
    ReusabilityReference)) , ((2*Reusability) + (0.5*Coupling) + (-0.5*Cohesion) -Messaging),
      IF( (TIME >=(STARTTIME + (CISTimeChange2)+ 5 <<da>>)) AND (Reusability >=
    ReusabilityReference), ((- DSC) + 'DSC-FirstChange'), IF( ((TIME >=(STARTTIME +
    (CISTimeChange3+ 5<<da>>))) AND (Reusability < ReusabilityReference)) , ((2*Reusability) + (0.5*
    Coupling) + (-0.5*Cohesion) -Messaging),
      IF( (TIME >=(STARTTIME + (CISTimeChange3)+ 5 <<da>>)) AND (Reusability >=
    ReusabilityReference), ((- DSC) + 'DSC-FirstChange'), IF( ((TIME >=(STARTTIME +
    (CISTimeChange1 + 5<<da>>))) AND (Reusability < ReusabilityReference)) , ((2*Reusability) + (0.5*
    Coupling) + (-0.5*Cohesion) -Messaging),
      IF( ((TIME >=(STARTTIME + (DCC_TimeChange1 + 5<<da>>))) AND (Reusability <
    ReusabilityReference)) , ((2*Reusability) + (0.5*Coupling) + (-0.5*Cohesion) -Messaging),
      IF( (TIME >=(STARTTIME + (DCC_TimeChange1)+ 5 <<da>>)) AND (Reusability >=
    ReusabilityReference), ((- DSC) + 'DSC-FirstChange'),
      IF( ((TIME >=(STARTTIME + (DCC_TimeChange2 + 5<<da>>))) AND (Reusability <
    ReusabilityReference)) , ((2*Reusability) + (0.5*Coupling) + (-0.5*Cohesion) -Messaging),
      IF( (TIME >=(STARTTIME + (DCC_TimeChange2)+ 5 <<da>>)) AND (Reusability >=
    ReusabilityReference), ((- DSC) + 'DSC-FirstChange'), IF( ((TIME >=(STARTTIME +
    (DCC_TimeChange3 + 5<<da>>))) AND (Reusability < ReusabilityReference)) , ((2*Reusability) + (0.
    5*Coupling) + (-0.5*Cohesion) -Messaging),
      IF( (TIME >=(STARTTIME + (DCC_TimeChange3)+ 5 <<da>>)) AND (Reusability >=
    ReusabilityReference), ((- DSC) + 'DSC-FirstChange'), IF( ((TIME >=(STARTTIME +
    (DCC_TimeChange1 + 5<<da>>))) AND (Reusability < ReusabilityReference)) , ((2*Reusability) + (0.
    5*Coupling) + (-0.5*Cohesion) -Messaging))))))))))))))))))))))))))))))))))))))))))))))))))))))
    IF(DesignSizeAdaptationU =1,
      IF( ((TIME >=(STARTTIME + (DCC_TimeChange1 + 5<<da>>))) AND
    (Understandability < UnderstandReference)) , ((-3.03*Understandability) -Abstraction + Encapsulation
    - Coupling + Cohesion - Polymorphism - Complexity),
      IF( (TIME >=(STARTTIME + (DCC_TimeChange1)+ 5 <<da>>)) AND
    (Understandability >= UnderstandReference), ((- DSC) + DSCFirstUnderst),
      IF( ((TIME >=(STARTTIME + (DCC_TimeChange2 + 5<<da>>))) AND
    (Understandability < UnderstandReference)) , ((-3.03*Understandability) -Abstraction + Encapsulation
    - Coupling + Cohesion - Polymorphism - Complexity),

```







```

- (0.54* Cohesion) - Polymorphism - Messaging - Hierarchies),
  IF( ((TIME >=(STARTTIME + (CISTimeChange1 + 5<<da>>))) AND (Functionality >=
FunctionalityReference)) , ((4.54*Functionality) - (0.54* Cohesion) - Polymorphism - Messaging -
Hierarchies),
  IF( (TIME >=(STARTTIME + (CISTimeChange1)+ 5 <<da>>)) AND (Functionality >=
FunctionalityReference), ((- DSC) + DSCFirstFunc),
  IF( ((TIME >=(STARTTIME + (CISTimeChange2+ 5<<da>>))) AND (Functionality>=
FunctionalityReference)) , ((4.54*Functionality) - (0.54* Cohesion) - Polymorphism - Messaging -
Hierarchies),
  IF( (TIME >=(STARTTIME + (CISTimeChange2+ 5 <<da>>))) AND (Functionality >=
FunctionalityReference), ((- DSC) + DSCFirstFunc), IF( ((TIME >=(STARTTIME +
(CISTimeChange3+ 5<<da>>))) AND (Functionality < FunctionalityReference)) , ((4.54*Functionality)
- (0.54* Cohesion) - Polymorphism - Messaging - Hierarchies),
  IF( (TIME >=(STARTTIME + (CISTimeChange3+ 5 <<da>>))) AND (Functionality >=
FunctionalityReference), ((- DSC) + DSCFirstFunc), IF( ((TIME >=(STARTTIME +
(CISTimeChange1 + 5<<da>>))) AND (Functionality< FunctionalityReference)) ,((4.54*Functionality) -
(0.54* Cohesion) - Polymorphism - Messaging - Hierarchies),
  IF( ((TIME >=(STARTTIME + (NOHTimeChange1 + 5<<da>>))) AND (Functionality <
FunctionalityReference)) , ((4.54*Functionality) - (0.54* Cohesion) - Polymorphism - Messaging -
Hierarchies),
  IF( (TIME >=(STARTTIME + (NOHTimeChange1)+ 5 <<da>>)) AND (Functionality >=
FunctionalityReference), ((- DSC) + DSCFirstFunc),
  IF( ((TIME >=(STARTTIME + (NOHTimeChange2 + 5<<da>>))) AND (Functionality <
FunctionalityReference)) , ((4.54*Functionality) - (0.54* Cohesion) - Polymorphism - Messaging -
Hierarchies),
  IF( (TIME >=(STARTTIME + (NOHTimeChange2)+ 5 <<da>>)) AND (Functionality >=
FunctionalityReference), ((- DSC) + DSCFirstFunc), IF( ((TIME >=(STARTTIME +
(NOHTimeChange3 + 5<<da>>))) AND (Functionality < FunctionalityReference)) , ((4.54*
Functionality) - (0.54* Cohesion) - Polymorphism - Messaging - Hierarchies),
  IF( (TIME >=(STARTTIME + (NOHTimeChange3)+ 5 <<da>>)) AND (Functionality >=
FunctionalityReference), ((- DSC) + DSCFirstFunc), IF( ((TIME >=(STARTTIME +
(NOHTimeChange1 + 5<<da>>))) AND (Functionality< FunctionalityReference)) ,((4.54*Functionality)
- (0.54* Cohesion) - Polymorphism - Messaging - Hierarchies))))))))))))))))))))))))))))))))))
}
aux Computed MFA {
  autotype Real
  autounit qual
  def MFA +
    IF(CheckedInheritance =1, IF( (TIME >=(STARTTIME + MFATimeChange1)) AND (TIME <
(STARTTIME + MFATimeChange2)),MFACHange1, IF( (TIME >=(STARTTIME + MFATimeChange1))
AND (MFACHange2 = 0 <<qual>>) AND (MFACHange3 = 0 <<qual>>),MFACHange1,
    IF((TIME >=(STARTTIME + MFATimeChange2)) AND (TIME <(STARTTIME +
MFATimeChange3)),MFACHange2, IF((TIME >=(STARTTIME + MFATimeChange2)) AND
(MFACHange1 = 0 <<qual>>) AND (MFACHange3 = 0 <<qual>>),MFACHange2,IF( (TIME >=
(STARTTIME + MFATimeChange1)) AND (TIME <(STARTTIME+ MFATimeChange3)) AND
(MFACHange2 = 0 <<qual>>),MFACHange1, IF((TIME >=(STARTTIME +MFATimeChange2)) AND
(MFACHange3 = 0 <<qual>>),MFACHange2,
    IF( (TIME >=(STARTTIME + MFATimeChange3)),MFACHange3))))),
  IF(InheritanceAdaptationExt =1,
    IF( ((TIME >=(STARTTIME + (DCC_TimeChange1 + 5<<da>>))) AND (Extendibility <
ExtendibilityReference)) , ((2*Extendibility) - Abstraction + Coupling- Polymorphism),
    IF( (TIME >=(STARTTIME + (DCC_TimeChange1)+ 5 <<da>>)) AND (Extendibility>=
ExtendibilityReference), ((- MFA) + MFAFirstExt),
    IF( ((TIME >=(STARTTIME + (DCC_TimeChange2 + 5<<da>>))) AND (Extendibility <
ExtendibilityReference)) , ((2*Extendibility) - Abstraction + Coupling- Polymorphism),
    IF( (TIME >=(STARTTIME + (DCC_TimeChange2)+ 5 <<da>>)) AND (Extendibility >
= ExtendibilityReference), ((- MFA) + MFAFirstExt), IF( ((TIME >=(STARTTIME +
(DCC_TimeChange3 + 5<<da>>))) AND (Extendibility < ExtendibilityReference)) , ((2*Extendibility) -
Abstraction + Coupling- Polymorphism),

```







```

>= EffectivenessReference)) , ((5*Effectiveness) - Abstraction - Encapsulation - Composition -
Polymorphism),
    IF( (TIME >=(STARTTIME + (MOA_TimeChange1)+ 5 <<da>>)) AND (Effectiveness
>= EffectivenessReference), ((- MFA) + MFAFirstEffect),
    IF( ((TIME >=(STARTTIME + (MOA_TimeChange2 + 5<<da>>))) AND
(Effectiveness>= EffectivenessReference)) , ((5*Effectiveness) - Abstraction - Encapsulation -
Composition - Polymorphism),
    IF( (TIME >=(STARTTIME + (MOA_TimeChange2+ 5 <<da>>))) AND (Effectiveness
>= EffectivenessReference), ((- MFA) + MFAFirstEffect), IF( ((TIME >=(STARTTIME +
(MOA_TimeChange3+ 5<<da>>))) AND (Effectiveness < EffectivenessReference)) , ((5*
Effectiveness) - Abstraction - Encapsulation - Composition - Polymorphism),
    IF( (TIME >=(STARTTIME + (MOA_TimeChange3+ 5 <<da>>))) AND (Effectiveness
>= EffectivenessReference), ((- MFA) + MFAFirstEffect), IF( ((TIME >=(STARTTIME +
(MOA_TimeChange1 + 5<<da>>))) AND (Effectiveness< EffectivenessReference)) ,((5*
Effectiveness) - Abstraction - Encapsulation - Composition - Polymorphism),
    IF( ((TIME >=(STARTTIME + (ANATimeChange1 + 5<<da>>))) AND (Effectiveness <
EffectivenessReference)) , ((5*Effectiveness) - Abstraction - Encapsulation - Composition -
Polymorphism),
    IF( (TIME >=(STARTTIME + (ANATimeChange1)+ 5 <<da>>)) AND (Effectiveness >
= EffectivenessReference), ((- MFA) + MFAFirstEffect),
    IF( ((TIME >=(STARTTIME + (ANATimeChange2 + 5<<da>>))) AND (Effectiveness <
EffectivenessReference)) , ((5*Effectiveness) - Abstraction - Encapsulation - Composition -
Polymorphism),
    IF( (TIME >=(STARTTIME + (ANATimeChange2)+ 5 <<da>>)) AND (Effectiveness >
= EffectivenessReference), ((- MFA) + MFAFirstEffect), IF( ((TIME >=(STARTTIME +
(ANATimeChange3 + 5<<da>>))) AND (Effectiveness < EffectivenessReference)) , ((5*Effectiveness)
- Abstraction - Encapsulation - Composition - Polymorphism),
    IF( (TIME >=(STARTTIME + (ANATimeChange3)+ 5 <<da>>)) AND (Effectiveness >
= EffectivenessReference), ((- MFA) + MFAFirstEffect), IF( ((TIME >=(STARTTIME +
(ANATimeChange1 + 5<<da>>))) AND (Effectiveness< EffectivenessReference)) ,((5*Effectiveness) -
Abstraction - Encapsulation - Composition - Polymorphism))))))))))))))))))))))))))))))
}
aux Computed MOA {
  autotype Real
  autounit qual
  def MOA +
    IF(CheckedComposition =1, IF( (TIME >=(STARTTIME + MOA_TimeChange1)) AND (TIME <
(STARTTIME + MOA_TimeChange2)),MOAChange1, IF( (TIME >=(STARTTIME +
MOA_TimeChange1)) AND (MOAChange2 = 0 <<qual>>) AND (MOAChange3 = 0 <<qual>>),
MOAChange1,
    IF((TIME >=(STARTTIME + MOA_TimeChange2)) AND (TIME <(STARTTIME +
MOA_TimeChange3)),MOAChange2, IF((TIME >=(STARTTIME + MOA_TimeChange2)) AND
(MOAChange1 = 0 <<qual>>) AND (MOAChange3 = 0 <<qual>>),MOAChange2,IF( (TIME >=
(STARTTIME + MOA_TimeChange1)) AND (TIME <(STARTTIME + MOA_TimeChange3)) AND
(MOAChange2 = 0 <<qual>>),MOAChange1, IF((TIME >=(STARTTIME + MOA_TimeChange2)) AND
(MOAChange3 = 0 <<qual>>),MOAChange2,
    IF( (TIME >=(STARTTIME + MOA_TimeChange3)),MOAChange3)))))),
    IF(ComposAdaptationF =1,
    IF( ((TIME >=(STARTTIME + (DCC_TimeChange1 + 5<<da>>))) AND (Flexibility <
FlexibilityReference)) , ((2*Flexibility) -(0.5*Encapsulation) + (0.5* Coupling) - Polymorphism),
    IF( (TIME >=(STARTTIME + (DCC_TimeChange1)+ 5 <<da>>)) AND (Flexibility >=
FlexibilityReference), ((- MOA) + MOAFirstFlexibility),
    IF( ((TIME >=(STARTTIME + (DCC_TimeChange2 + 5<<da>>))) AND (Flexibility <
FlexibilityReference)) , ((2*Flexibility) -(0.5*Encapsulation) + (0.5* Coupling) - Polymorphism),
    IF( (TIME >=(STARTTIME + (DCC_TimeChange2)+ 5 <<da>>)) AND (Flexibility >=
FlexibilityReference), ((- MOA) + MOAFirstFlexibility), IF( ((TIME >=(STARTTIME +
(DCC_TimeChange3 + 5<<da>>))) AND (Flexibility < FlexibilityReference)) , ((2*Flexibility) -(0.5*
Encapsulation) + (0.5* Coupling) - Polymorphism),
    IF( (TIME >=(STARTTIME + (DCC_TimeChange3)+ 5 <<da>>)) AND (Flexibility >=
FlexibilityReference), ((- MOA) + MOAFirstFlexibility), IF( ((TIME >=(STARTTIME +
(DCC_TimeChange1 + 5<<da>>))) AND (Flexibility< FlexibilityReference)) ,((2*Flexibility) -(0.5*
Encapsulation) + (0.5* Coupling) - Polymorphism),
    IF( ((TIME >=(STARTTIME + (DAM_TimeChange1 + 5<<da>>))) AND (Flexibility <
FlexibilityReference)) , ((2*Flexibility) -(0.5*Encapsulation) + (0.5* Coupling) - Polymorphism),
    IF( (TIME >=(STARTTIME + (DAM_TimeChange1)+ 5 <<da>>)) AND (Flexibility >=

```





```

        IF ( TIME >=(STARTTIME + (DAM_TimeChange2+ 5 <<da>>))) AND (Effectiveness
        >= EffectivenessReference), ((-MOA) + MOAFirstEffect), IF( ((TIME >=(STARTTIME +
        (DAM_TimeChange3+ 5<<da>>))) AND (Effectiveness < EffectivenessReference)) , ((5*
        Effectiveness) - Abstraction - Encapsulation - Inheritance - Polymorphism),
        IF( TIME >=(STARTTIME + (DAM_TimeChange3+ 5 <<da>>))) AND (Effectiveness
        >= EffectivenessReference), ((-MOA) + MOAFirstEffect), IF( ((TIME >=(STARTTIME +
        (DAM_TimeChange1 + 5<<da>>))) AND (Effectiveness< EffectivenessReference)) ,((5*Effectiveness)
        - Abstraction - Encapsulation - Inheritance - Polymorphism),
        IF( ((TIME >=(STARTTIME + (MFATimeChange1 + 5<<da>>))) AND (Effectiveness <
        EffectivenessReference)) , ((5*Effectiveness) - Abstraction - Encapsulation - Inheritance -
        Polymorphism),
        IF( TIME >=(STARTTIME + (MFATimeChange1)+ 5 <<da>>)) AND (Effectiveness >
        = EffectivenessReference), ((-MOA) + MOAFirstEffect),
        IF( ((TIME >=(STARTTIME + (MFATimeChange2 + 5<<da>>))) AND (Effectiveness <
        EffectivenessReference)) , ((5*Effectiveness) - Abstraction - Encapsulation - Inheritance -
        Polymorphism),
        IF( TIME >=(STARTTIME + (MFATimeChange2)+ 5 <<da>>)) AND (Effectiveness >
        = EffectivenessReference), ((-MOA) + MOAFirstEffect), IF( ((TIME >=(STARTTIME +
        (MFATimeChange3 + 5<<da>>))) AND (Effectiveness < EffectivenessReference)) , ((5*Effectiveness)
        - Abstraction - Encapsulation - Inheritance - Polymorphism),
        IF( TIME >=(STARTTIME + (MFATimeChange3)+ 5 <<da>>)) AND (Effectiveness >
        = EffectivenessReference), ((-MOA) + MOAFirstEffect), IF( ((TIME >=(STARTTIME +
        (MFATimeChange1 + 5<<da>>))) AND (Effectiveness< EffectivenessReference)) ,((5*Effectiveness) -
        Abstraction - Encapsulation - Inheritance - Polymorphism))))))))))))))))))))))))))
    }
    aux Computed NOH {
        autotype Real
        autounit qual
        def NOH +
            IF(CheckedHierarchies =1, IF( TIME >=(STARTTIME + NOHTimeChange1)) AND (TIME <
            (STARTTIME + NOHTimeChange2)),NOHChange1, IF( TIME >=(STARTTIME + NOHTimeChange1
            ) AND (NOHChange2 = 0 <<qual>>) AND (NOHChange3 = 0 <<qual>>),NOHChange1,
            IF((TIME >=(STARTTIME + NOHTimeChange2)) AND (TIME <(STARTTIME +
            NOHTimeChange3)),NOHChange2, IF((TIME >=(STARTTIME + NOHTimeChange2)) AND
            (NOHChange1 = 0 <<qual>>) AND (NOHChange3 = 0 <<qual>>),NOHChange2,IF( TIME >=
            (STARTTIME + NOHTimeChange1)) AND (TIME <(STARTTIME + NOHTimeChange3)) AND
            (NOHChange2 = 0 <<qual>>),NOHChange1, IF((TIME >=(STARTTIME + NOHTimeChange2)) AND
            (NOHChange3 = 0 <<qual>>),NOHChange2,
            IF( TIME >=(STARTTIME + NOHTimeChange3),NOHChange3)))))
        IF(HierarchiesAdaptationFunct =1,
            IF( ((TIME >=(STARTTIME + (CISTimeChange1 + 5<<da>>))) AND (Functionality <
            FunctionalityReference)) , ((4.54*Functionality) - (0.54* Cohesion) - Polymorphism - Messaging -
            'Design Size'),
            IF( TIME >=(STARTTIME + (CISTimeChange1)+ 5 <<da>>)) AND (Functionality >=
            FunctionalityReference), ((- NOH) + NOHFirstFunct),
            IF( ((TIME >=(STARTTIME + (CISTimeChange2 + 5<<da>>))) AND (Functionality <
            FunctionalityReference)) , ((4.54*Functionality) - (0.54* Cohesion) - Polymorphism - Messaging -
            'Design Size'),
            IF( TIME >=(STARTTIME + (CISTimeChange2)+ 5 <<da>>)) AND (Functionality >=
            FunctionalityReference), ((- NOH) + NOHFirstFunct), IF( ((TIME >=(STARTTIME + (CISTimeChange3
            + 5<<da>>))) AND (Functionality < FunctionalityReference)) , ((4.54*Functionality) - (0.54* Cohesion)
            - Polymorphism - Messaging - 'Design Size'),
            IF( TIME >=(STARTTIME + (CISTimeChange3)+ 5 <<da>>)) AND (Functionality >=
            FunctionalityReference), ((- NOH) + NOHFirstFunct), IF( ((TIME >=(STARTTIME +
            (CISTimeChange1 + 5<<da>>))) AND (Functionality< FunctionalityReference)) ,((4.54*Functionality) -
            (0.54* Cohesion) - Polymorphism - Messaging - 'Design Size'),
            IF( ((TIME >=(STARTTIME + (CAMTimeChange1 + 5<<da>>))) AND (Functionality >
            = FunctionalityReference)) , ((4.54*Functionality) - (0.54* Cohesion) - Polymorphism - Messaging -
            'Design Size'),

```



```

IF( (TIME >=(STARTTIME + (CAMTimeChange1)+ 5 <<da>>)) AND (Functionality >=
FunctionalityReference), ((- NOH) + NOHFirstFunc),
IF( ((TIME >=(STARTTIME + (CAMTimeChange2 + 5<<da>>))) AND (Functionality >
= FunctionalityReference)) , ((4.54*Functionality) - (0.54* Cohesion) - Polymorphism - Messaging -
'Design Size'),
IF( (TIME >=(STARTTIME + (CAMTimeChange2+ 5 <<da>>))) AND (Functionality >=
FunctionalityReference), ((- NOH) + NOHFirstFunc), IF( ((TIME >=(STARTTIME +
(CAMTimeChange3+ 5<<da>>))) AND (Functionality < FunctionalityReference)) , ((4.54*
Functionality) - (0.54* Cohesion) - Polymorphism - Messaging - 'Design Size'),
IF( (TIME >=(STARTTIME + (CAMTimeChange3+ 5 <<da>>))) AND (Functionality >=
FunctionalityReference), ((- NOH) + NOHFirstFunc), IF( ((TIME >=(STARTTIME +
(CAMTimeChange1 + 5<<da>>))) AND (Functionality< FunctionalityReference)) ,((4.54*Functionality)
- (0.54* Cohesion) - Polymorphism - Messaging - 'Design Size'),
IF( ((TIME >=(STARTTIME + (DSCTimeChange1 + 5<<da>>))) AND (Functionality >
= FunctionalityReference)) , ((4.54*Functionality) - (0.54* Cohesion) - Polymorphism - Messaging -
'Design Size'),
IF( (TIME >=(STARTTIME + (DSCTimeChange1)+ 5 <<da>>)) AND (Functionality >=
FunctionalityReference), ((- NOH) + NOHFirstFunc),
IF( ((TIME >=(STARTTIME + (DSCTimeChange2 + 5<<da>>))) AND (Functionality >=
FunctionalityReference)) , ((4.54*Functionality) - (0.54* Cohesion) - Polymorphism - Messaging -
'Design Size'),
IF( (TIME >=(STARTTIME + (DSCTimeChange2+ 5 <<da>>))) AND (Functionality >=
FunctionalityReference), ((- NOH) + NOHFirstFunc), IF( ((TIME >=(STARTTIME +
(DSCTimeChange3+ 5<<da>>))) AND (Functionality < FunctionalityReference)) , ((4.54*Functionality)
- (0.54* Cohesion) - Polymorphism - Messaging - 'Design Size'),
IF( (TIME >=(STARTTIME + (DSCTimeChange3+ 5 <<da>>))) AND (Functionality >=
FunctionalityReference), ((- NOH) + NOHFirstFunc), IF( ((TIME >=(STARTTIME +
(DSCTimeChange1 + 5<<da>>))) AND (Functionality< FunctionalityReference)) ,((4.54*Functionality)
- (0.54* Cohesion) - Polymorphism - Messaging - 'Design Size'),
IF( ((TIME >=(STARTTIME + (NOPTimeChange1 + 5<<da>>))) AND (Functionality <
FunctionalityReference)) , ((4.54*Functionality) - (0.54* Cohesion) - Polymorphism - Messaging -
'Design Size'),
IF( (TIME >=(STARTTIME + (NOPTimeChange1)+ 5 <<da>>)) AND (Functionality >=
FunctionalityReference), ((- NOH) + NOHFirstFunc),
IF( ((TIME >=(STARTTIME + (NOPTimeChange2 + 5<<da>>))) AND (Functionality <
FunctionalityReference)) , ((4.54*Functionality) - (0.54* Cohesion) - Polymorphism - Messaging -
'Design Size'),
IF( (TIME >=(STARTTIME + (NOPTimeChange2)+ 5 <<da>>)) AND (Functionality >=
FunctionalityReference), ((- NOH) + NOHFirstFunc), IF( ((TIME >=(STARTTIME +
(NOPTimeChange3 + 5<<da>>))) AND (Functionality < FunctionalityReference)) , ((4.54*
Functionality) - (0.54* Cohesion) - Polymorphism - Messaging - 'Design Size'),
IF( (TIME >=(STARTTIME + (NOPTimeChange3)+ 5 <<da>>)) AND (Functionality >=
FunctionalityReference), ((- NOH) + NOHFirstFunc), IF( ((TIME >=(STARTTIME +
(NOPTimeChange1 + 5<<da>>))) AND (Functionality< FunctionalityReference)) ,((4.54*Functionality)
- (0.54* Cohesion) - Polymorphism - Messaging - 'Design Size'))))))))))))))))))))))))))))))))

```

```

}
aux Computed NOM {
  autotype Real
  autounit qual
  def NOM +
    IF(CheckedComplexity =1, IF( (TIME >=(STARTTIME + NOMTimeChange1)) AND (TIME <
(STARTTIME + NOMTimeChange2)),NOMChange1, IF( (TIME >=(STARTTIME +
NOMTimeChange1)) AND (NOMChange2 = 0 <<qual>>) AND (NOMChange3 = 0 <<qual>>),
NOMChange1,
    IF((TIME >=(STARTTIME + NOMTimeChange2)) AND (TIME <(STARTTIME +
NOMTimeChange3)),NOMChange2, IF((TIME >=(STARTTIME + NOMTimeChange2)) AND
(NOMChange1 = 0 <<qual>>) AND (NOMChange3 = 0 <<qual>>),NOMChange2,IF( (TIME >=
(STARTTIME + NOMTimeChange1)) AND (TIME <(STARTTIME + NOMTimeChange3)) AND

```





```

- Coupling + Cohesion - Polymorphism - 'Design Size'),
  IF( (TIME >=(STARTTIME + (DAM_TimeChange2)+ 5 <<da>>)) AND
(Understandability >= UnderstandReference), ((- NOM) + NOMFirstUnderst), IF( ((TIME >=
(STARTTIME + (DAM_TimeChange3 + 5<<da>>))) AND (Understandability < UnderstandReference))
, ((-3.03*Understandability) -Abstraction + Encapsulation - Coupling + Cohesion - Polymorphism -
'Design Size'),
  IF( (TIME >=(STARTTIME + (DAM_TimeChange3)+ 5 <<da>>)) AND
(Understandability >= UnderstandReference), ((- NOM) + NOMFirstUnderst), IF( ((TIME >=
(STARTTIME + (DAM_TimeChange1 + 5<<da>>))) AND (Understandability< UnderstandReference))
,((-.03*Understandability) -Abstraction + Encapsulation - Coupling + Cohesion - Polymorphism -
'Design Size'),
  IF( ((TIME >=(STARTTIME + (DSCTimeChange1 + 5<<da>>))) AND
(Understandability < UnderstandReference)), ((-3.03*Understandability) -Abstraction + Encapsulation
- Coupling + Cohesion - Polymorphism - 'Design Size'),
  IF( (TIME >=(STARTTIME + (DSCTimeChange1)+ 5 <<da>>)) AND
(Understandability >= UnderstandReference), ((- NOM) + NOMFirstUnderst),
  IF( ((TIME >=(STARTTIME + (DSCTimeChange2 + 5<<da>>))) AND
(Understandability < UnderstandReference)), ((-3.03*Understandability) -Abstraction + Encapsulation
- Coupling + Cohesion - Polymorphism - 'Design Size'),
  IF( (TIME >=(STARTTIME + (DSCTimeChange2)+ 5 <<da>>)) AND
(Understandability >= UnderstandReference), ((- NOM) + NOMFirstUnderst), IF( ((TIME >=
(STARTTIME + (DSCTimeChange3 + 5<<da>>))) AND (Understandability < UnderstandReference)) ,
((-3.03*Understandability) -Abstraction + Encapsulation - Coupling + Cohesion - Polymorphism -
'Design Size'),
  IF( (TIME >=(STARTTIME + (DSCTimeChange3)+ 5 <<da>>)) AND
(Understandability >= UnderstandReference), ((- NOM) + NOMFirstUnderst), IF( ((TIME >=
(STARTTIME + (DSCTimeChange1 + 5<<da>>))) AND (Understandability< UnderstandReference)) ,(
(-3.03*Understandability) -Abstraction + Encapsulation - Coupling + Cohesion - Polymorphism -
'Design Size'),
  IF( ((TIME >=(STARTTIME + (ANATimeChange1 + 5<<da>>))) AND
(Understandability < UnderstandReference)), ((-3.03*Understandability) -Abstraction + Encapsulation
- Coupling + Cohesion - Polymorphism - 'Design Size'),
  IF( (TIME >=(STARTTIME + (ANATimeChange1)+ 5 <<da>>)) AND
(Understandability >= UnderstandReference), ((- NOM) + NOMFirstUnderst),
  IF( ((TIME >=(STARTTIME + (ANATimeChange2 + 5<<da>>))) AND
(Understandability < UnderstandReference)), ((-3.03*Understandability) -Abstraction + Encapsulation
- Coupling + Cohesion - Polymorphism - 'Design Size'),
  IF( (TIME >=(STARTTIME + (ANATimeChange2)+ 5 <<da>>)) AND
(Understandability >= UnderstandReference), ((- NOM) + NOMFirstUnderst), IF( ((TIME >=
(STARTTIME + (ANATimeChange3 + 5<<da>>))) AND (Understandability < UnderstandReference)) ,
((-3.03*Understandability) -Abstraction + Encapsulation - Coupling + Cohesion - Polymorphism -
'Design Size'),
  IF( (TIME >=(STARTTIME + (ANATimeChange3)+ 5 <<da>>)) AND
(Understandability >= UnderstandReference), ((- NOM) + NOMFirstUnderst), IF( ((TIME >=
(STARTTIME + (ANATimeChange1 + 5<<da>>))) AND (Understandability< UnderstandReference)) ,(
(-3.03*Understandability) -Abstraction + Encapsulation - Coupling + Cohesion - Polymorphism -
'Design Size')

```

```

))))))))))))))))))))))))))))))))))))))))))

```

```

}
aux Computed NOP {
  autotype Real
  autounit qual
  def NOP +
  IF(CheckedPolymo =1, IF( (TIME >=(STARTTIME + NOPTimeChange1)) AND (TIME <(STARTTIME
+ NOPTimeChange2)),NOPChange1, IF( (TIME >=(STARTTIME + NOPTimeChange1)) AND
(NOPChange2 = 0 <<qual>>) AND (NOPChange3 = 0 <<qual>>),NOPChange1,
  IF((TIME >=(STARTTIME + NOPTimeChange2)) AND (TIME <(STARTTIME +
NOPTimeChange3)),NOPChange2, IF((TIME >=(STARTTIME + NOPTimeChange2)) AND
(NOPChange1 = 0 <<qual>>) AND (NOPChange3 = 0 <<qual>>),NOPChange2,IF( (TIME >=
(STARTTIME + NOPTimeChange1)) AND (TIME <(STARTTIME + NOPTimeChange3)) AND
(NOPChange2 = 0 <<qual>>),NOPChange1, IF((TIME >=(STARTTIME + MOA_TimeChange2)) AND

```



















```

}
level Coupling {
  autotype Real
  autounit qual
  init DCCInitial
  inflow { autodef 'Change in coupling' }
}
const CouplingAdaptationEx {
  autotype Real
  init 0
}
const CouplingAdaptationF {
  autotype Real
  init 0
}
const CouplingAdaptationR {
  autotype Real
  init 0
}
const CouplingAdaptationU {
  autotype Real
  init 0
}
const DAM {
  autotype Real
  autounit qual
  init 0.9 <<qual>>
}
const DAM_Change1 {
  autotype Real
  autounit qual
  init - 0.3 <<qual>>
}
const DAM_Change2 {
  autotype Real
  autounit qual
  init -0.5<<qual>>
}
const DAM_Change3 {
  autotype Real
  autounit qual
  init -0.7 <<qual>>
}
const DAM_ExecuteTime {
  autotype Real
  autounit da
  init 1<<da>>
}
const DAM_TimeChange1 {
  autotype Real
  autounit da
  init 30 <<da>>
}
const DAM_TimeChange2 {
  autotype Real
  autounit da
  init 60 <<da>>
}
const DAM_TimeChange3 {
  autotype Real

```

```

    autounit da
    init 90<<da>>
}
const DAMFirstEncapsu {
    autotype Real
    autounit qual
    init INITIF((((TIME >=(STARTTIME + (ANATimeChange1)+ 5 <<da>>)) OR ((TIME >=(STARTTIME +
    (MFATimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (MOA_TimeChange1)+ 5 <<da>>)))
    OR ((TIME >=(STARTTIME + (NOPTimeChange1)+ 5 <<da>>)))) AND (Effectiveness >=
    EffectivenessReference)), Encapsulation)
}
const DAMFirstFlexibility {
    autotype Real
    autounit qual
    init INITIF((((TIME >=(STARTTIME + (DCC_TimeChange1) + 5 <<da>>)) OR ((TIME >=(STARTTIME +
    (MOA_TimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (NOPTimeChange1)+ 5 <<da>>))))
    AND (Flexibility >= FlexibilityReference)), Encapsulation)
}
const DAMFirstUnderstand {
    autotype Real
    autounit qual
    init INITIF((((TIME >=(STARTTIME + (CAMTimeChange1) + 5 <<da>>)) OR ((TIME >=(STARTTIME +
    (DCC_TimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (ANATimeChange1)+ 5 <<da>>)))
    OR ((TIME >=(STARTTIME + (DSCTimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME +
    (NOMTimeChange1)+ 5 <<da>>)))OR ((TIME >=(STARTTIME + (NOPTimeChange1)+ 5 <<da>>))))
    AND (Understandability >= UnderstandReference)), Encapsulation)
}
const DAMInitial {
    autotype Real
    autounit qual
    init INITIF(TIME = STARTTIME, DAM)
}
const DCC {
    autotype Real
    unit qual
    init 5<<qual>>
}
const DCC-FirstChange {
    autotype Real
    autounit qual
    init INITIF((((TIME >=(STARTTIME + (CISTimeChange1)+ 5 <<da>>)) OR ((TIME >=(STARTTIME +
    (CAMTimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (DSCTimeChange1)+ 5 <<da>>))))
    AND (Reusability >= ReusabilityReference)), Coupling)
}
const DCC_Change1 {
    autotype Real
    autounit qual
    init 5<<qual>>
}
const DCC_Change2 {
    autotype Real
    autounit qual
    init 10<<qual>>
}
const DCC_Change3 {
    autotype Real
    autounit qual
    init 15<<qual>>
}
const DCC_ExecuteTime {
    autotype Real
    autounit da
    init 1<<da>>
}
const DCC_TimeChange1 {
    autotype Real

```



```

autounit da
init 30<<da>>
}
const DCC_TimeChange2 {
autotype Real
autounit da
init 60<<da>>
}
const DCC_TimeChange3 {
autotype Real
autounit da
init 90<<da>>
}
const DCCFirstExt {
autotype Real
autounit qual
init INITIF(((TIME >=(STARTTIME + (ANATimeChange1) + 5 <<da>>)) OR ((TIME >=(STARTTIME +
(MFATimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (NOPTimeChange1)+ 5 <<da>>))))
AND (Extendibility >= ExtendibilityReference)), Coupling)
}
const DCCFirstFlexibility {
autotype Real
autounit qual
init INITIF(((TIME >=(STARTTIME + (DAM_TimeChange1) + 5 <<da>>)) OR ((TIME >=(STARTTIME +
(MOA_TimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (NOPTimeChange1)+ 5 <<da>>))))
AND (Flexibility >= FlexibilityReference)), Coupling)
}
const DCCFirstUnderst {
autotype Real
autounit qual
init INITIF(((TIME >=(STARTTIME + (CAMTimeChange1) + 5 <<da>>)) OR ((TIME >=(STARTTIME +
(DAM_TimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (ANATimeChange1)+ 5 <<da>>)))
OR ((TIME >=(STARTTIME + (DSCTimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME +
(NOMTimeChange1)+ 5 <<da>>)))OR ((TIME >=(STARTTIME + (NOPTimeChange1)+ 5 <<da>>))))
AND (Understandability >= UnderstandReference)), Coupling)
}
const DCCInitial {
autotype Real
autounit qual
init INITIF(TIME = STARTTIME, DCC)
}
level Design Size {
autotype Real
autounit qual
init DSCInitial
inflow { autodef 'Change in Design Size' }
}
const DesignSizeAdaptationFunct {
autotype Real
init 0
}
const DesignSizeAdaptationR {
autotype Real
init 0
}
const DesignSizeAdaptationU {
autotype Real
init 0
}
const DSC {
autotype Real
autounit qual
init 50<<qual>>
}
const DSC-FirstChange {
autotype Real

```

```

autounit qual
init INITIF(((TIME >=(STARTTIME + (CAMTimeChange1)+ 5 <<da>>)) OR ((TIME >=(STARTTIME +
(DCC_TimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (CISTimeChange1)+ 5 <<da>>))))
AND (Reusability >= ReusabilityReference)), 'Design Size')
}
const DSC_ExecuteTime {
autotype Real
autounit da
init 1<<da>>
}
const DSCChange1 {
autotype Real
autounit qual
init (-20)<<qual>>
}
const DSCChange2 {
autotype Real
autounit qual
init (-40)<<qual>>
}
const DSCChange3 {
autotype Real
autounit qual
init (-45)<<qual>>
}
const DSCFirstFunc {
autotype Real
autounit qual
init INITIF(((TIME >=(STARTTIME + (CAMTimeChange1)+ 5 <<da>>)) OR ((TIME >=(STARTTIME +
(NOTimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (CISTimeChange1)+ 5 <<da>>)))
OR ((TIME >=(STARTTIME + (NOHTimeChange1)+ 5 <<da>>)))) AND (Functionality >=
FunctionalityReference)), 'Design Size')
}
const DSCFirstUnderst {
autotype Real
autounit qual
init INITIF(((TIME >=(STARTTIME + (DCC_TimeChange1) + 5 <<da>>)) OR ((TIME >=(STARTTIME +
(DAM_TimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (CAMTimeChange1)+ 5 <<da>>)))
OR ((TIME >=(STARTTIME + (NOMTimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME +
(ANATimeChange1)+ 5 <<da>>)))OR ((TIME >=(STARTTIME + (NOTimeChange1)+ 5 <<da>>))))
AND (Understandability >= UnderstandReference)), 'Design Size')
}
const DSCInitial {
autotype Real
autounit qual
init INITIF(TIME = STARTTIME, DSC)
}
const DSCTimeChange1 {
autotype Real
autounit da
init 30<<da>>
}
const DSCTimeChange2 {
autotype Real
autounit da
init 60<<da>>
}
const DSCTimeChange3 {
autotype Real
autounit da
init 90<<da>>
}
aux Effectiveness {
autotype Real
autounit qual
def ((0.2*Encapsulation)+(0.2*Abstraction)+(0.2*Composition)+(0.2*Polymorphism)+(0.2*Inheritance))

```



```

}
const EffectivenessReference {
  autotype Real
  autounit qual
  init INITIF(TIME = STARTTIME, Effectiveness)
}
const EncapsuAdaptationEffect {
  autotype Real
  init 0
}
const EncapsuAdaptationF {
  autotype Real
  init 0
}
const EncapsuAdaptationU {
  autotype Real
  init 0
}
level Encapsulation {
  autotype Real
  autounit qual
  init DAMInitial
  inflow { autodef 'Change in Encapsulation' }
}
aux Extendibility {
  autotype Real
  autounit qual
  def (((0.5)*Abstraction) + ((-0.5)*Coupling) + ((0.5)*Inheritance) + ((0.5)*Polymorphism))
}
const ExtendibilityReference {
  autotype Real
  autounit qual
  init INITIF(TIME = STARTTIME, Extendibility)
}
aux Flexibility {
  autotype Real
  autounit qual
  def (((0.25*Encapsulation)-0.25*Coupling)+(0.5*Composition)+(0.5*Polymorphism))
}
const FlexibilityReference {
  autotype Real
  autounit qual
  init INITIF(TIME = STARTTIME, Flexibility)
}
aux Functionality {
  autotype Real
  autounit qual
  def (((0.22)*Polymorphism) + ((0.12)*Cohesion) + ((0.22)*Messaging)+ ((0.22)*'Design Size')+ ((0.22)*
  Hierarchies))
}
const FunctionalityReference {
  autotype Real
  autounit qual
  init INITIF(TIME = STARTTIME, Functionality)
}
level Hierarchies {
  autotype Real
  autounit qual
  init NOHInitial
  inflow { autodef 'Change in Hierarchies' }
}
const HierarchiesAdaptationFunct {
  autotype Real
  init 0
}
level Inheritance {

```

```

    autotype Real
    unit qual
    init MFAInitial
    inflow { autodef 'Change in inheritance' }
}
const InheritanceAdaptationEffect {
    autotype Real
    init 0
}
const InheritanceAdaptationExt {
    autotype Real
    init 0
}
level Messaging {
    autotype Real
    autounit qual
    init CISInitial
    inflow { autodef 'Change in Messaging' }
}
const MessagingAdaptationFunc {
    autotype Real
    init 0
}
const MessagingAdaptationR {
    autotype Real
    init 0
}
const MFA {
    autotype Real
    autounit qual
    init 0.9 <<qual>>
}
const MFA_ExecuteTime {
    autotype Real
    unit da
    init 1
}
const MFACHange1 {
    autotype Real
    autounit qual
    init -0.3<<qual>>
}
const MFACHange2 {
    autotype Real
    autounit qual
    init -0.5<<qual>>
}
const MFACHange3 {
    autotype Real
    autounit qual
    init -0.7<<qual>>
}
const MFAFirstEffect {
    autotype Real
    autounit qual
    init INITIF((((TIME >=(STARTTIME + (DAM_TimeChange1)+ 5 <<da>>)) OR ((TIME >=(STARTTIME +
    (ANATimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (MOA_TimeChange1)+ 5 <<da>>))))
    OR ((TIME >=(STARTTIME + (NOPTimeChange1)+ 5 <<da>>)))) AND (Effectiveness >=
    EffectivenessReference)), Inheritance)
}
const MFAFirstExt {
    autotype Real
    autounit qual
    init INITIF((((TIME >=(STARTTIME + (DCC_TimeChange1) + 5 <<da>>)) OR ((TIME >=(STARTTIME +
    (ANATimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (NOPTimeChange1)+ 5 <<da>>))))
    AND (Extendibility >= ExtendibilityReference)), Inheritance)
}

```

```

}
const MFAInitial {
  autotype Real
  autounit qual
  init INITIF(TIME = STARTTIME,MFA)
}
const MFATimeChange1 {
  autotype Real
  autounit da
  init 30<<da>>
}
const MFATimeChange2 {
  autotype Real
  autounit da
  init 60<<da>>
}
const MFATimeChange3 {
  autotype Real
  autounit da
  init 90<<da>>
}
const MOA {
  autotype Real
  autounit qual
  init 20<<qual>>
}
const MOA_ExecuteTime {
  autotype Real
  autounit da
  init 1<<da>>
}
const MOA_TimeChange1 {
  autotype Real
  autounit da
  init 30<<da>>
}
const MOA_TimeChange2 {
  autotype Real
  autounit da
  init 60<<da>>
}
const MOA_TimeChange3 {
  autotype Real
  autounit da
  init 90<<da>>
}
const MOAChange1 {
  autotype Real
  autounit qual
  init -5 <<qual>>
}
const MOAChange2 {
  autotype Real
  autounit qual
  init -10<<qual>>
}
const MOAChange3 {
  autotype Real
  autounit qual
  init -15<<qual>>
}
const MOAFirstEffect {
  autotype Real
  autounit qual
  init INITIF((((TIME >=(STARTTIME + (DAM_TimeChange1)+ 5 <<da>>)) OR ((TIME >=(STARTTIME +
(MFATimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (ANATimeChange1)+ 5 <<da>>))))

```

```

    OR ((TIME >=(STARTTIME + (NOTimeChange1)+ 5 <<da>>)))) AND (Effectiveness >=
    EffectivenessReference)), Composition)
}
const MOAFirstFlexibility {
    autotype Real
    autounit qual
    init INITIF((((TIME >=(STARTTIME + (DCC_TimeChange1) + 5 <<da>>)) OR ((TIME >=(STARTTIME +
    (DAM_TimeChange1)+ 5 <<da>>)))) OR ((TIME >=(STARTTIME + (NOTimeChange1)+ 5 <<da>>))))
    AND (Flexibility >= FlexibilityReference)), Composition)
}
const MOAInitial {
    autotype Real
    autounit qual
    init INITIF(TIME = STARTTIME, MOA)
}
const NOH {
    autotype Real
    autounit qual
    init 20<<qual>>
}
const NOH_ExecuteTime {
    autotype Real
    autounit da
    init 1<<da>>
}
const NOHChange1 {
    autotype Real
    autounit qual
    init -5<<qual>>
}
const NOHChange2 {
    autotype Real
    autounit qual
    init -10<<qual>>
}
const NOHChange3 {
    autotype Real
    autounit qual
    init -15<<qual>>
}
const NOHFirstFunc {
    autotype Real
    autounit qual
    init INITIF((((TIME >=(STARTTIME + (CAMTimeChange1)+ 5 <<da>>)) OR ((TIME >=(STARTTIME +
    (NOTimeChange1)+ 5 <<da>>)))) OR ((TIME >=(STARTTIME + (CISTimeChange1)+ 5 <<da>>))))
    OR ((TIME >=(STARTTIME + (DSCTimeChange1)+ 5 <<da>>)))) AND (Functionality >=
    FunctionalityReference)), Hierarchies)
}
const NOHInitial {
    autotype Real
    autounit qual
    init INITIF(TIME = STARTTIME, NOH)
}
const NOHTimeChange1 {
    autotype Real
    autounit da
    init 30<<da>>
}
const NOHTimeChange2 {
    autotype Real
    autounit da
    init 60<<da>>
}
const NOHTimeChange3 {
    autotype Real
    autounit da
}

```

```

    init 90<<da>>
}
const NOM {
    autotype Real
    autounit qual
    init 5<<qual>>
}
const NOM_ExecuteTime {
    autotype Real
    autounit da
    init 1<<da>>
}
const NOMChange1 {
    autotype Real
    autounit qual
    init 5<<qual>>
}
const NOMChange2 {
    autotype Real
    autounit qual
    init 10<<qual>>
}
const NOMChange3 {
    autotype Real
    autounit qual
    init 15<<qual>>
}
const NOMFirstUnderst {
    autotype Real
    autounit qual
    init INITIF(((TIME >=(STARTTIME + (DCC_TimeChange1) + 5 <<da>>)) OR ((TIME >=(STARTTIME +
(DAM_TimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (CAMTimeChange1)+ 5 <<da>>)))
OR ((TIME >=(STARTTIME + (DSCTimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME +
(ANATimeChange1)+ 5 <<da>>)))OR ((TIME >=(STARTTIME + (NOPTimeChange1)+ 5 <<da>>))))
AND (Understandability >= UnderstandReference)), Complexity)
}
const NOMInitial {
    autotype Real
    autounit qual
    init INITIF(TIME = STARTTIME, NOM)
}
const NOMTimeChange1 {
    autotype Real
    autounit da
    init 30<<da>>
}
const NOMTimeChange2 {
    autotype Real
    autounit da
    init 60<<da>>
}
const NOMTimeChange3 {
    autotype Real
    autounit da
    init 90<<da>>
}
const NOP {
    autotype Real
    autounit qual
    init 20<<qual>>
}
const NOP_ExecuteTime {
    autotype Real
    autounit da
    init 1<<da>>
}
}

```



```

const NOPChange1 {
  autotype Real
  autounit qual
  init -5<<qual>>
}
const NOPChange2 {
  autotype Real
  autounit qual
  init -10<<qual>>
}
const NOPChange3 {
  autotype Real
  autounit qual
  init -15<<qual>>
}
const NOPFirstEffect {
  autotype Real
  autounit qual
  init INITIF(((TIME >=(STARTTIME + (DAM_TimeChange1)+ 5 <<da>>)) OR ((TIME >=(STARTTIME +
(MFATimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (MOA_TimeChange1)+ 5 <<da>>)))
OR ((TIME >=(STARTTIME + (ANATimeChange1)+ 5 <<da>>)))) AND (Effectiveness >=
EffectivenessReference)), Polymorphism)
}
const NOPFirstExt {
  autotype Real
  autounit qual
  init INITIF(((TIME >=(STARTTIME + (DCC_TimeChange1) + 5 <<da>>)) OR ((TIME >=(STARTTIME +
(MFATimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (ANATimeChange1)+ 5 <<da>>))))
AND (Extendibility >= ExtendibilityReference)), Polymorphism)
}
const NOPFirstFlexibility {
  autotype Real
  autounit qual
  init INITIF(((TIME >=(STARTTIME + (DCC_TimeChange1) + 5 <<da>>)) OR ((TIME >=(STARTTIME +
(DAM_TimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (MOA_TimeChange1)+ 5 <<da>>)))
)) AND (Flexibility >= FlexibilityReference)), Polymorphism)
}
const NOPFirstFunct {
  autotype Real
  autounit qual
  init INITIF(((TIME >=(STARTTIME + (CISTimeChange1)+ 5 <<da>>)) OR ((TIME >=(STARTTIME +
(CAMTimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (DSCTimeChange1)+ 5 <<da>>)))
OR ((TIME >=(STARTTIME + (NOHTimeChange1)+ 5 <<da>>)))) AND (Functionality >=
FunctionalityReference)), Polymorphism)
}
const NOPFirstUnderst {
  autotype Real
  autounit qual
  init INITIF(((TIME >=(STARTTIME + (DCC_TimeChange1) + 5 <<da>>)) OR ((TIME >=(STARTTIME +
(DAM_TimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME + (CAMTimeChange1)+ 5 <<da>>)))
OR ((TIME >=(STARTTIME + (NOMTimeChange1)+ 5 <<da>>))) OR ((TIME >=(STARTTIME +
(ANATimeChange1)+ 5 <<da>>)))OR ((TIME >=(STARTTIME + (DSCTimeChange1)+ 5 <<da>>))))
AND (Understandability >= UnderstandReference)), Polymorphism)
}
const NOPInitial {
  autotype Real
  autounit qual
  init INITIF(TIME = STARTTIME, NOP)
}
const NOPTimeChange1 {
  autotype Real
  autounit da
  init 30<<da>>
}
const NOPTimeChange2 {
  autotype Real

```



```

    autounit da
    init 60<<da>>
}
const NOPTimeChange3 {
    autotype Real
    autounit da
    init 90<<da>>
}
const PolymoAdaptationEffect {
    autotype Real
    init 0
}
const PolymoAdaptationExt {
    autotype Real
    init 0
}
const PolymoAdaptationF {
    autotype Real
    init 0
}
const PolymoAdaptationFunct {
    autotype Real
    init 0
}
const PolymoAdaptationU {
    autotype Real
    init 0
}
level Polymorphism {
    autotype Real
    autounit qual
    init NOPInitial
    inflow { autodef 'Change in polymorphism' }
}
aux Reusability {
    autotype Real
    autounit qual
    def (((-0.25)*Coupling)+((0.5)*Messaging)+ ((0.5)*"Design Size")+ ((0.25)*Cohesion))
}
const ReusabilityReference {
    autotype Real
    autounit qual
    init INITIF(TIME = STARTTIME, Reusability)
}
aux Understandability {
    autotype Real
    autounit qual
    def ((0.33*Encapsulation)-(0.33*Coupling)-(0.33*Abstraction)-(0.33*Polymorphism)+(0.33*Cohesion)-(0.33*Complexity)-(0.33*"Design Size"))
}
const UnderstandReference {
    autotype Real
    autounit qual
    init INITIF(TIME = STARTTIME, Understandability)
}
}
unit qual {
    def ATOMIC
}

```

## **Appendix C: simulation validation on D2-D10 designs**

This appendix presents the design changes and adaptations that were applied on D2-D10 designs. It also describes the correlations between the simulated and the real values of the QMOOD quality attributes other than understandability.

### **C.1 Design 2 (D2): Banker system**

#### **C.1.1 System description and reference quality values**

The banker system handles all banking operations such as accounts management and money withdrawals through two interfaces: clerk and administrator interfaces. On the one hand, a clerk can manage loans, check balances, and handle account operations. On the other hand, an administrator manages staff profiles (figure 56). The quality and reference values of the banker system are illustrated in tables 17 and 18.

#### **C.1.2 Design changes**

##### **C.1.2.1 Design changes affecting the understandability quality attribute**

The understandability quality attribute of D2 decreased below its reference value when design size increased by seven classes. To counterbalance the impact of that design change, encapsulation is the best fit adaptation mechanism both in the simulated and the real results (tables 8 and 9).

## **1) Simulated results**

D2 simulation results show that encapsulation should be increased by a factor of seven to compensate for the decrease in understandability (figure 53). Therefore, the “DAM” metric should equal 1 in all the newly added seven classes.

## **2) Real results**

The simulated results were applied on D2’s class diagram as illustrated in figure 54. Since encapsulation is maximized in all the seven classes (i.e. the ratio of private attributes to the total number of attributes in each class equals 1), the real value of understandability after adaptation equals its simulated value.

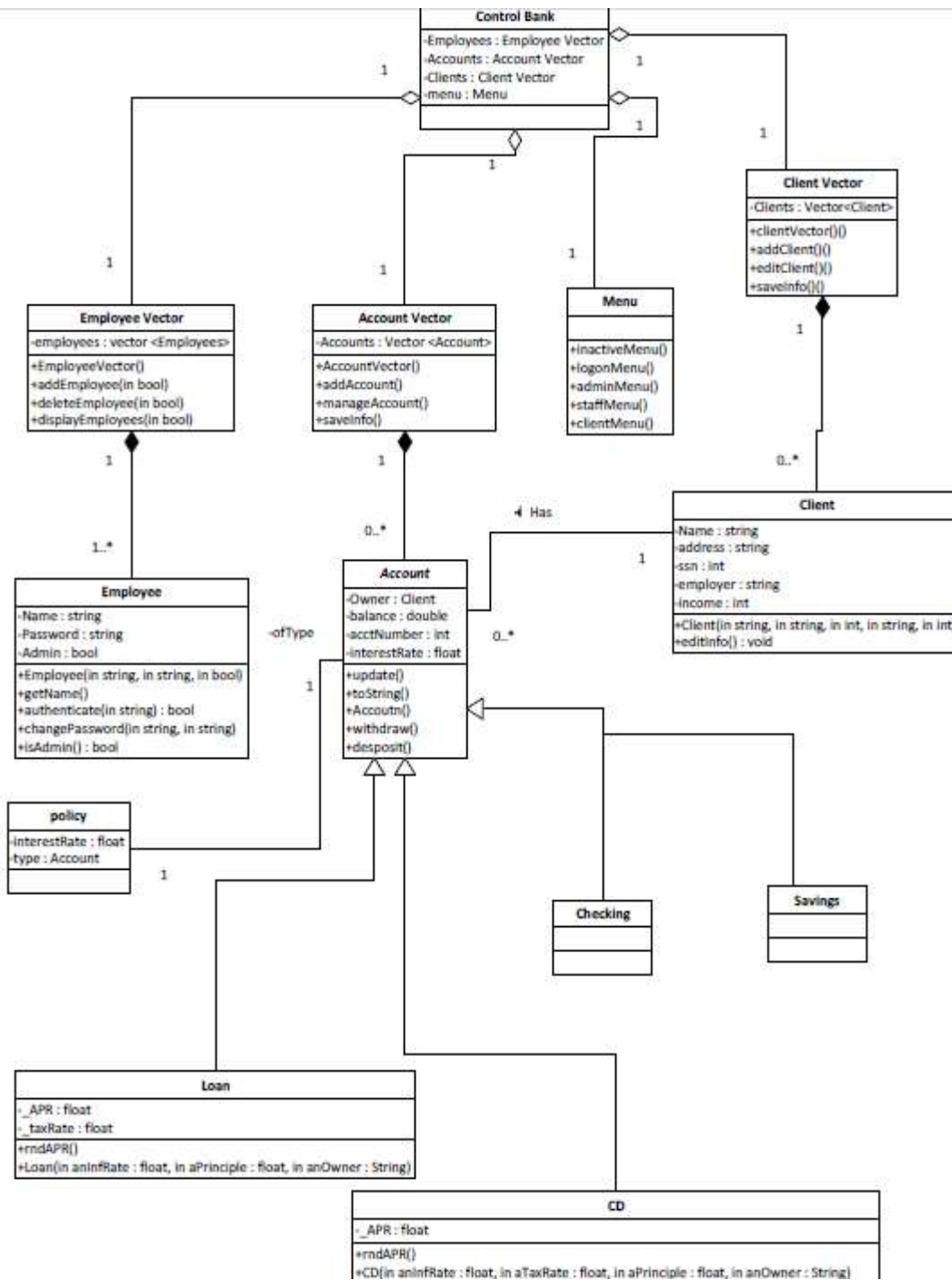


Figure 52: The banker system class diagram

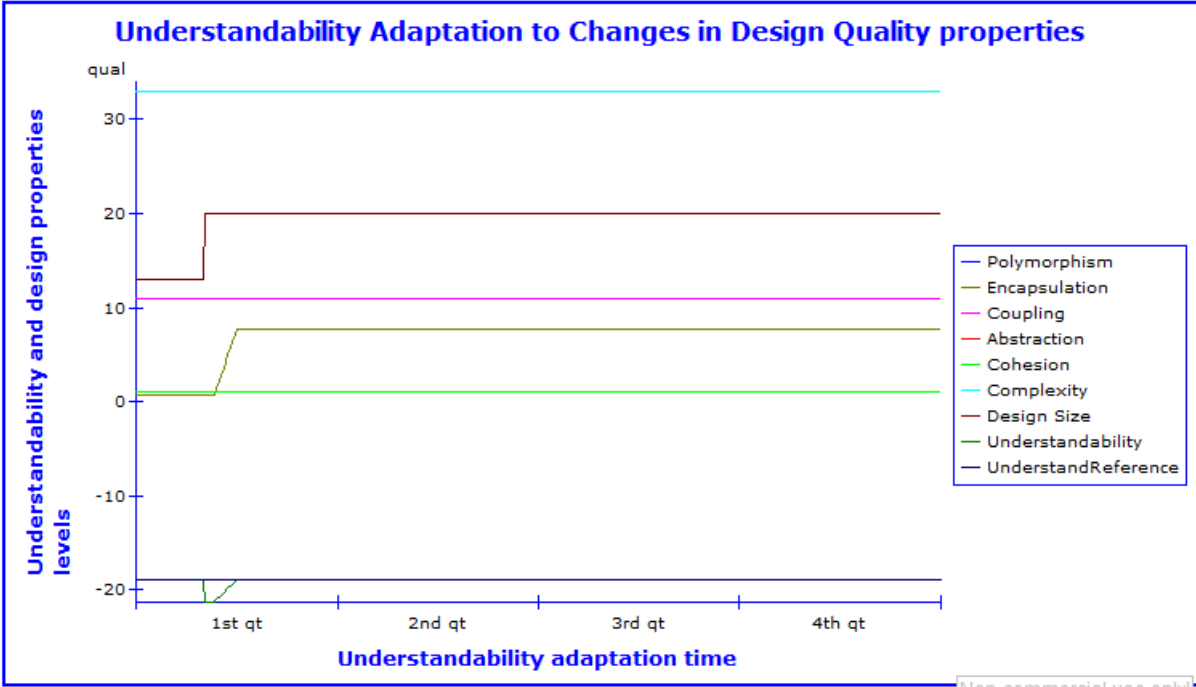
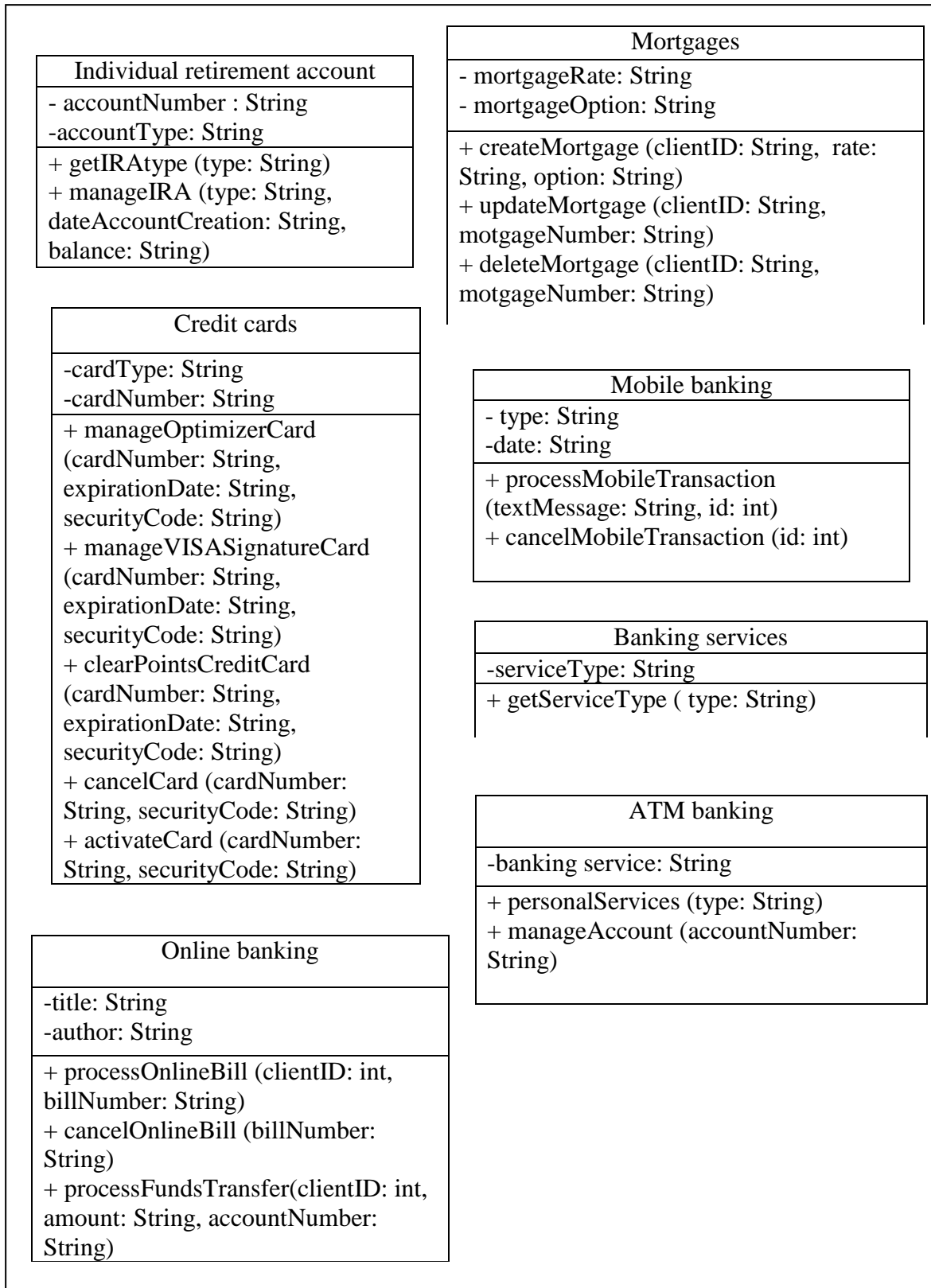


Figure 53: Understandability adaptation results of D2



**Figure 54: D2's understandability design change and adaptation**



### **C.1.2.2 Design changes affecting the extendibility and the flexibility quality attributes**

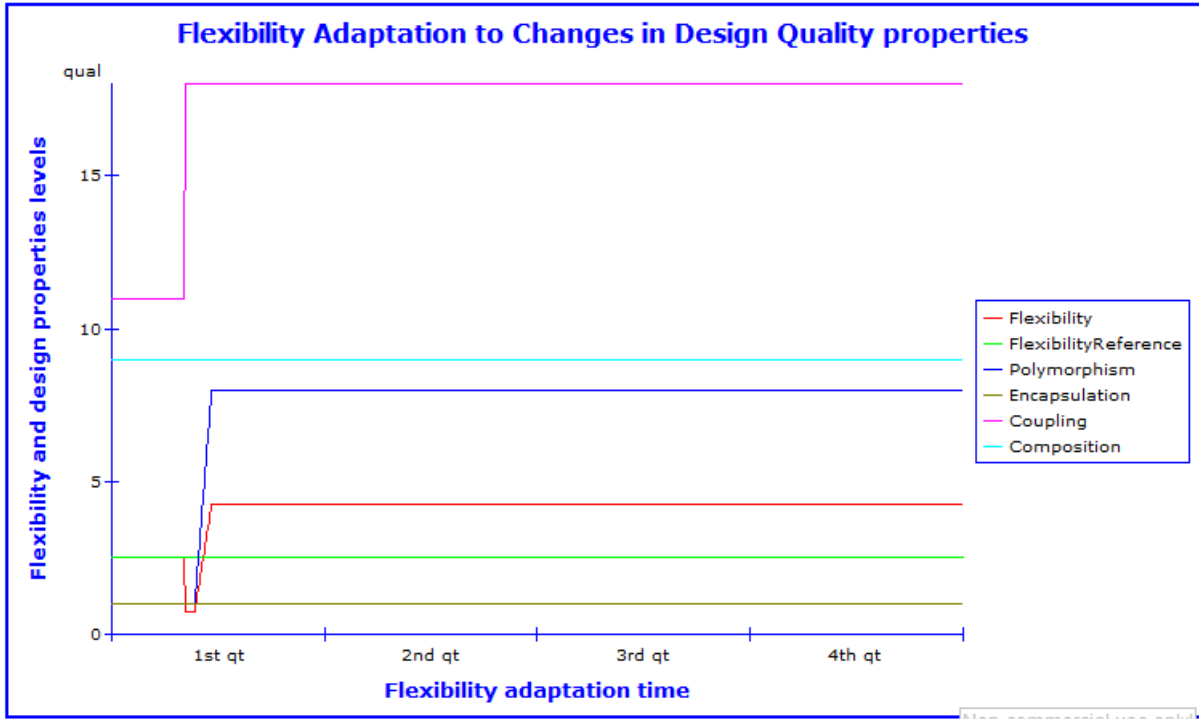
After adding seven new classes to D2's class diagram, new aggregation and inheritance relationships were identified as illustrated in figure 57. The aggregation relationships are represented by a diamond symbol. Since the "Banking services", "Credit cards", and "Mortgages" classes' responsibilities are part of the "Control bank" class, the defined relationship between them is aggregation. The "individual retirement account" inherits the characteristics of the "Account" class. Another inheritance relationship is defined between the "banking services" class and three classes: "Online banking", "Mobile banking", and "ATM banking". Those new changes and their adaptation were both simulated and applied in D2's class diagram.

#### **1) Simulated results**

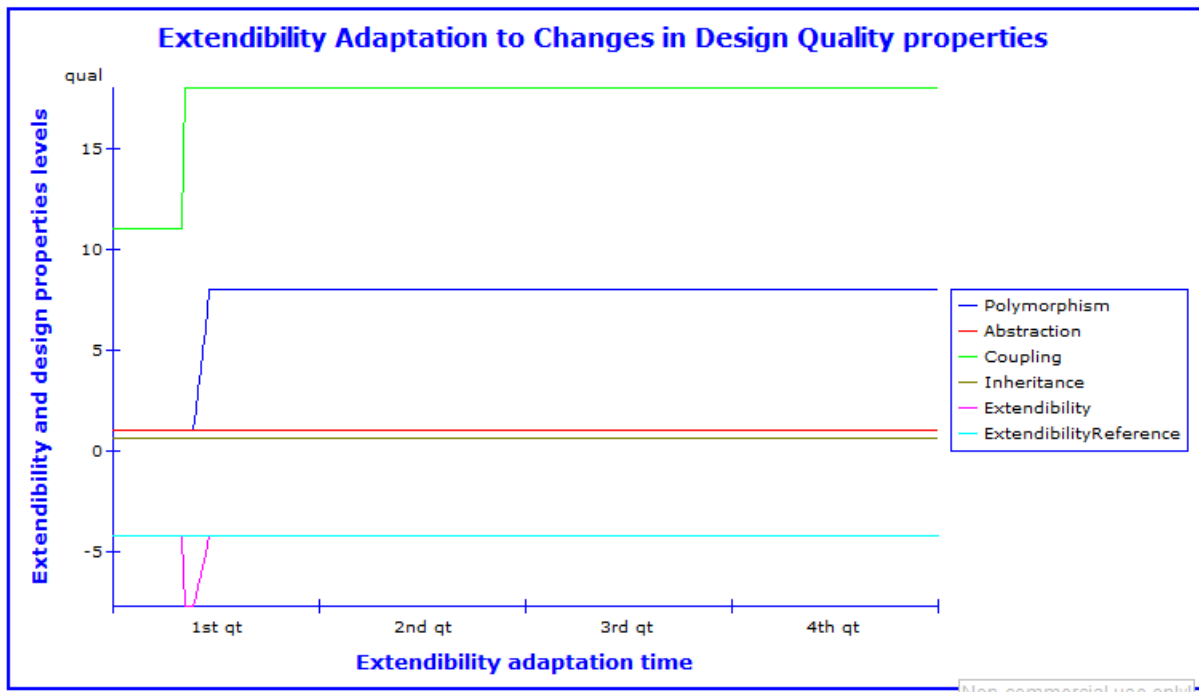
The identification of new inheritance and aggregation relationships led to an increase in coupling and a decrease in the flexibility and the extendibility quality attributes. After applying the adaptation equation of polymorphism, the extendibility quality attribute reaches its reference value and the decrease in flexibility is also counterbalanced (figures 55 and 56). The simulation results show that the NOP metric should be increased by a factor of seven to effectively adapt the flexibility and the extendibility values.

#### **2) Real results**

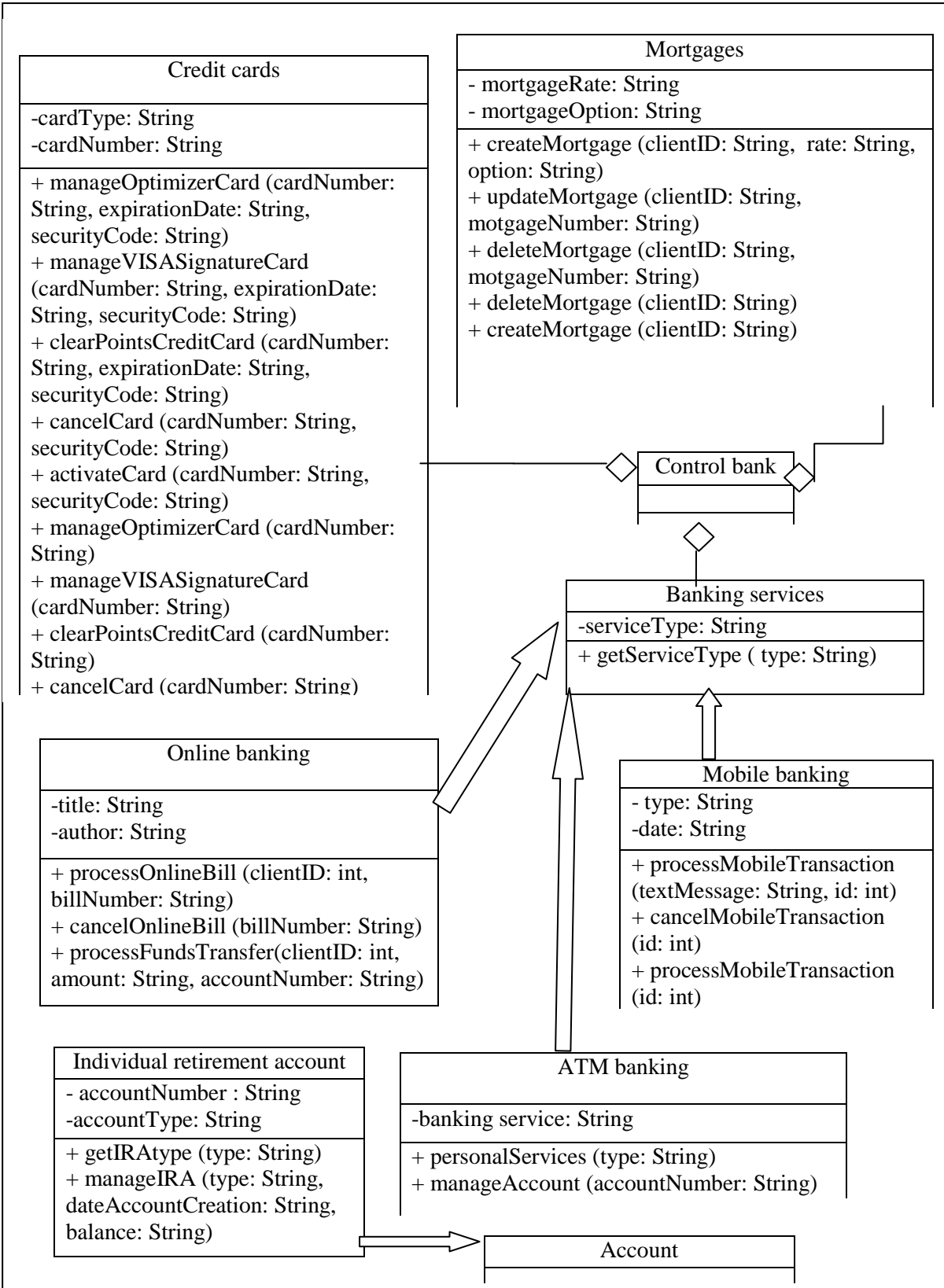
The simulated design changes and adaptations were applied on D2. Seven polymorphic methods were added to D2's classes as suggested by the simulation (figure 57). The values of flexibility and extendibility after integrating their adaptations on D2 nearly equal their simulated.



**Figure 55: Flexibility adaptation results of D2**



**Figure 56: Extendibility adaptation results of D2**



**Figure 57: Flexibility and extensibility design change and adaptation in D2**

### C.1.2.3 Design changes affecting the reusability and the functionality quality attributes

The third design change applied in D2 leads to a decrease in design size. The class “policy” was deleted and its attributes were merged in the “Account” class. Furthermore, the “checking” and the “savings” classes were deleted and their responsibilities were processed by the “Account” class.

#### 1) Simulated results

The simulation results of the third design change in Powersim<sup>®</sup> show a decrease in the reusability and the functionality quality attributes below their reference values (figures 58 and 59). This decrease is counterbalanced by applying the adaptation equation of cohesion. From the simulation graph, cohesion should be increased by a factor of six. Therefore, the relatedness among the methods of six classes in D2 should be maximized (i.e. CAM equals 1).

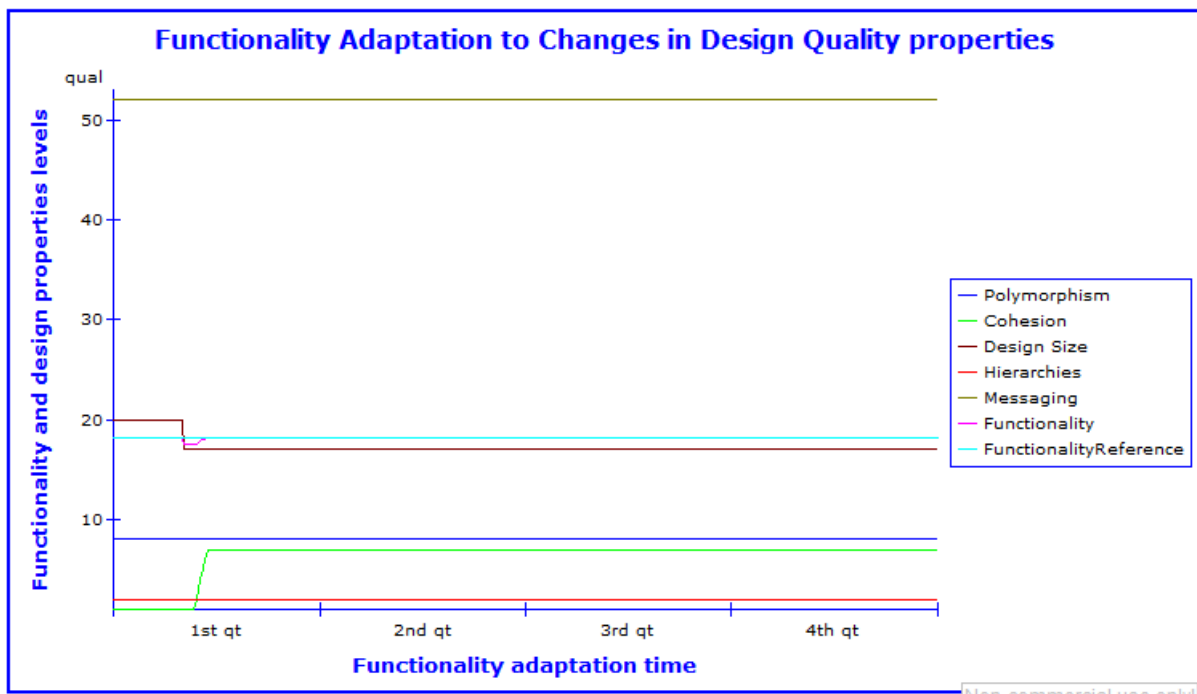
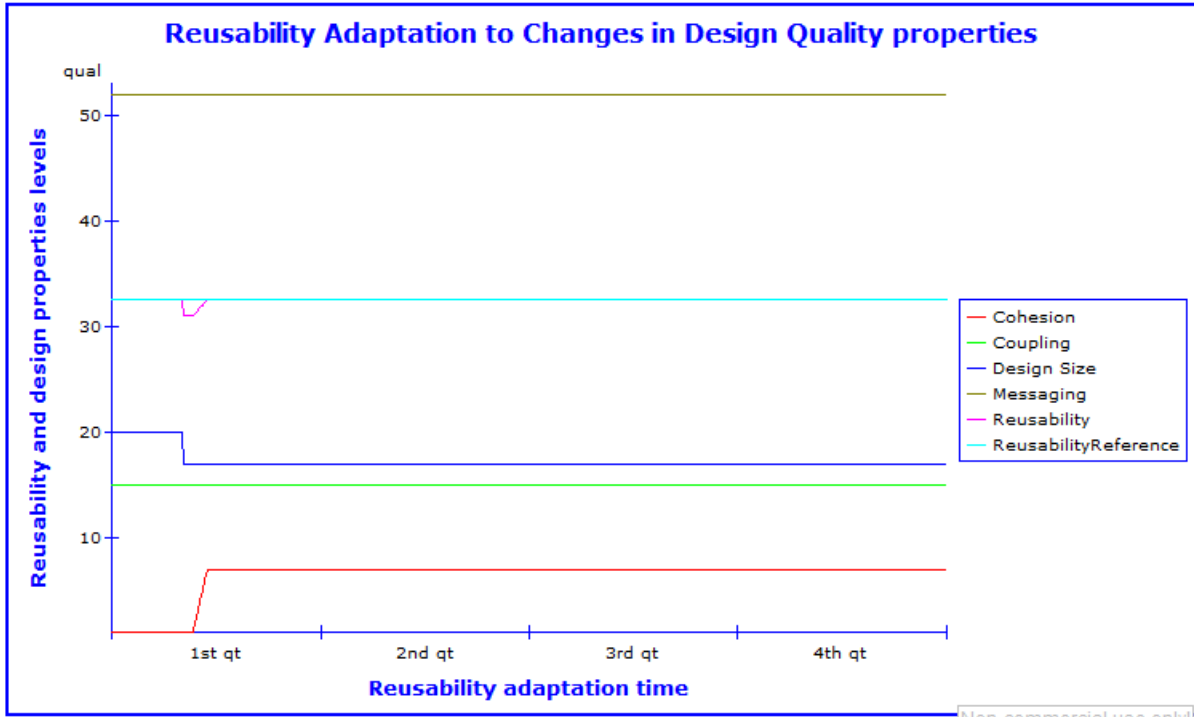


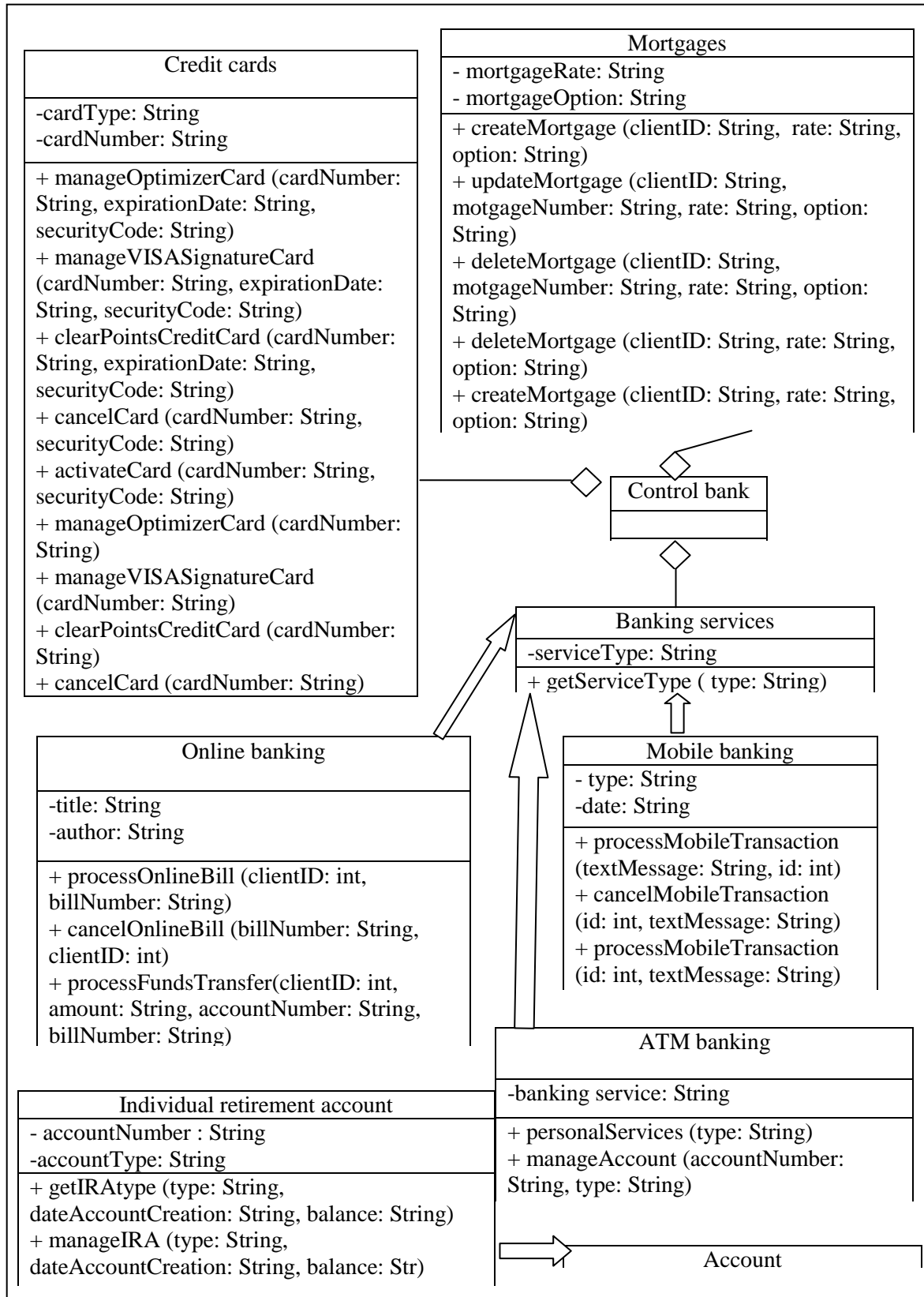
Figure 58: Functionality adaptation results of D2



**Figure 59: Reusability adaptation results of D2**

## 2) Real results

The simulated adaptation through cohesion was applied in D2 to counterbalance the effect of design size decrease. Hence, cohesion among the six following classes was maximized by increasing the relatedness among their methods: “Individual retirement account”, “ATM banking”, “Mobile banking”, “Online banking”, “Mortgages”, and “Banking services” (figure 60). The values of functionality and reusability after the real adaptation almost equal their forecasted values in the simulation.



**Figure 60: Functionality and reusability design change and adaptation in D2**



#### C.1.2.4 Design changes affecting the effectiveness quality attribute

Composition is decreased by deleting two aggregation relationships from D2's class diagram. As a consequence, the effectiveness value dropped below its reference value. The aggregation relationship between the "Mortgage" and the "Account" classes was deleted. Instead, the "Mortgage" class becomes one of the children of the "Account" class. Moreover, the "Menu" class was deleted and its methods were merged in the "Control Bank" class.

##### 1) Simulated results

According to the simulation results, polymorphism should be increased by a factor of eight to adapt to the decrease in effectiveness (figure 61).

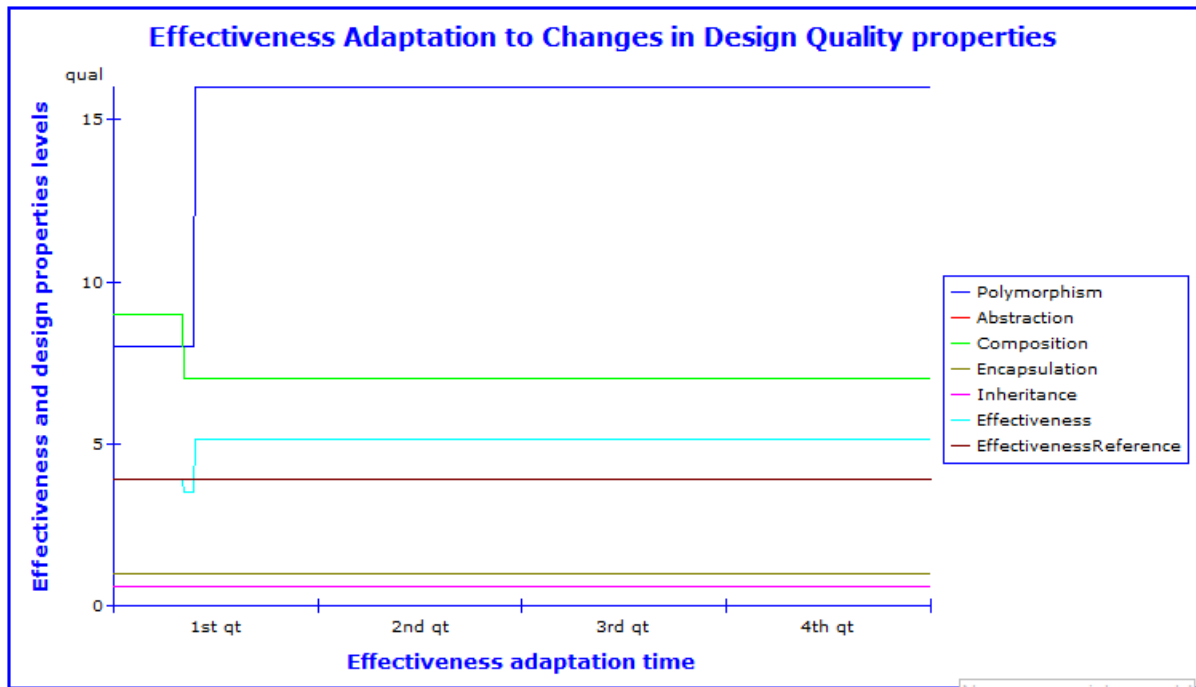


Figure 61: Effectiveness adaptation results of D2

## **2) Real results**

The same simulated results were noticed after applying the changes and their adaptations in D2's class diagram. The adaptation mechanism was applied in D2 by adding eight polymorphic equations such as "+ activateCard (cardNumber: String)" in the "Credits cards" class and "+ updateMortgage (clientID: String)" in the "Mortgages" class. The value of effectiveness from D2 after its adaptation is similar to its simulated expectations.

### **C.2. Design 3 (D3): The Sol security system**

#### **C.2.1. System description and reference quality values**

The Sol security system allows users to control the security options of their homes such as the cameras and the motion sensors remotely. Figures 62, 63, and 64 represent the different components of the system class diagram. Moreover, the initial quality and reference values of D3 are illustrated in tables 17, 18.

Workstation Design Class Diagram

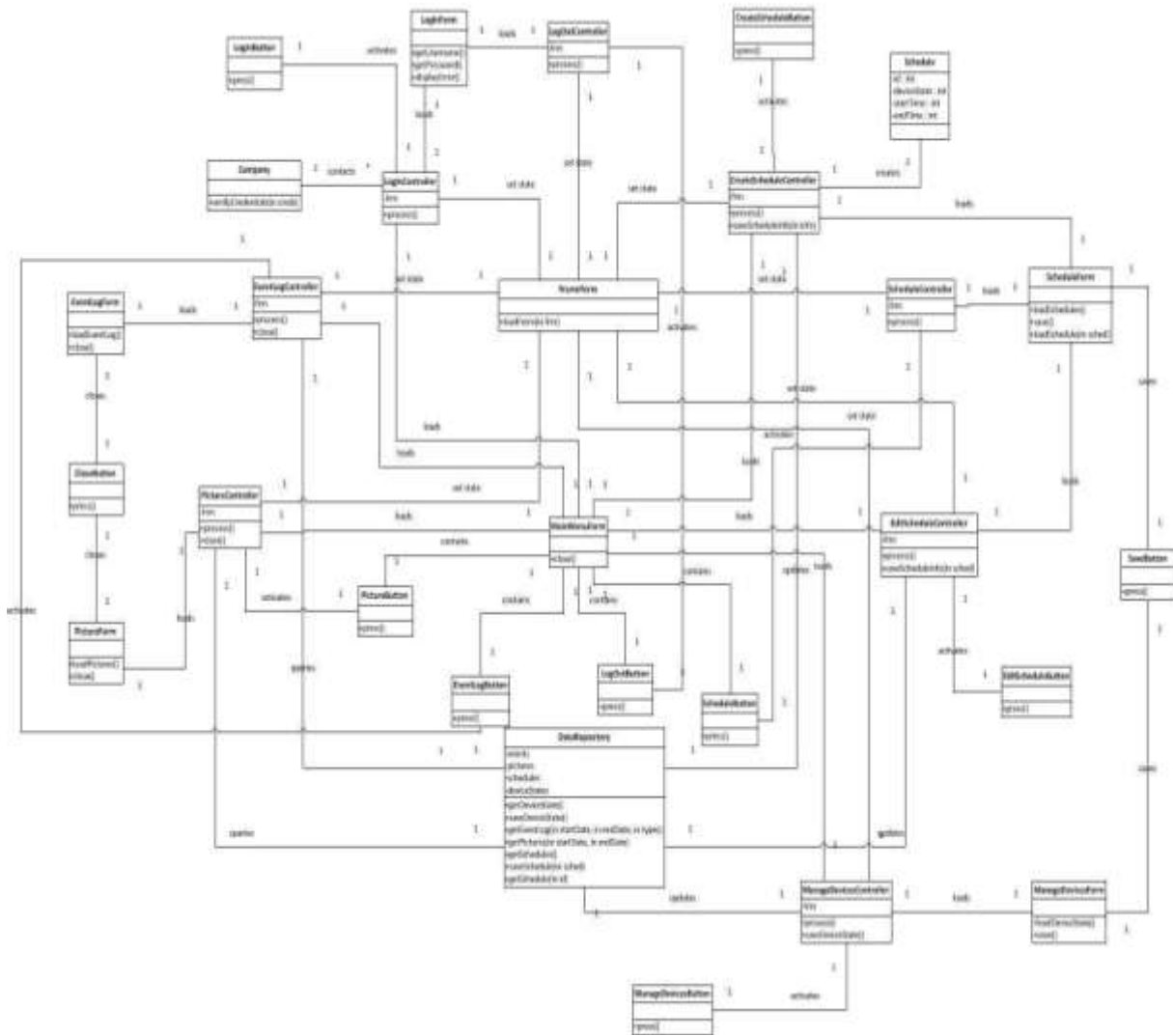
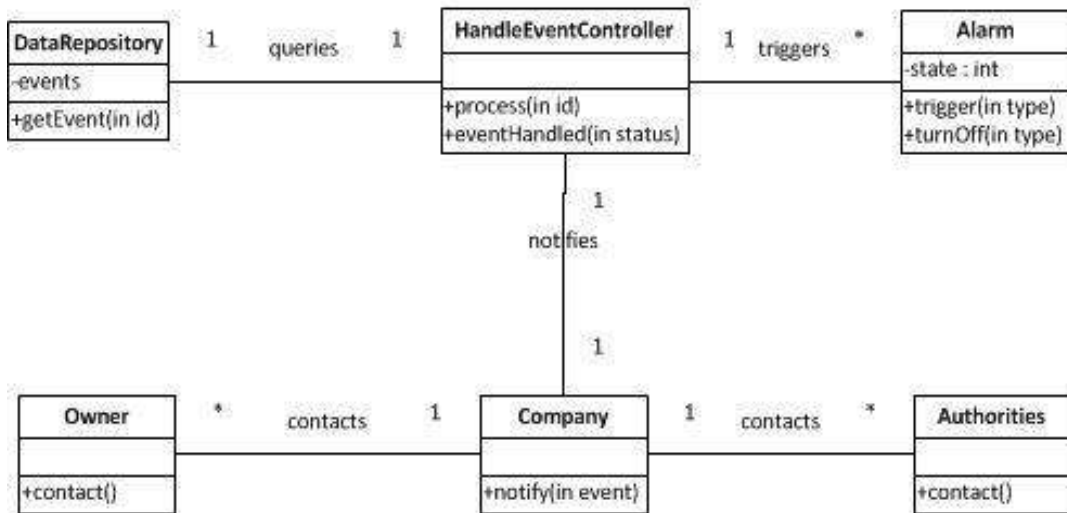


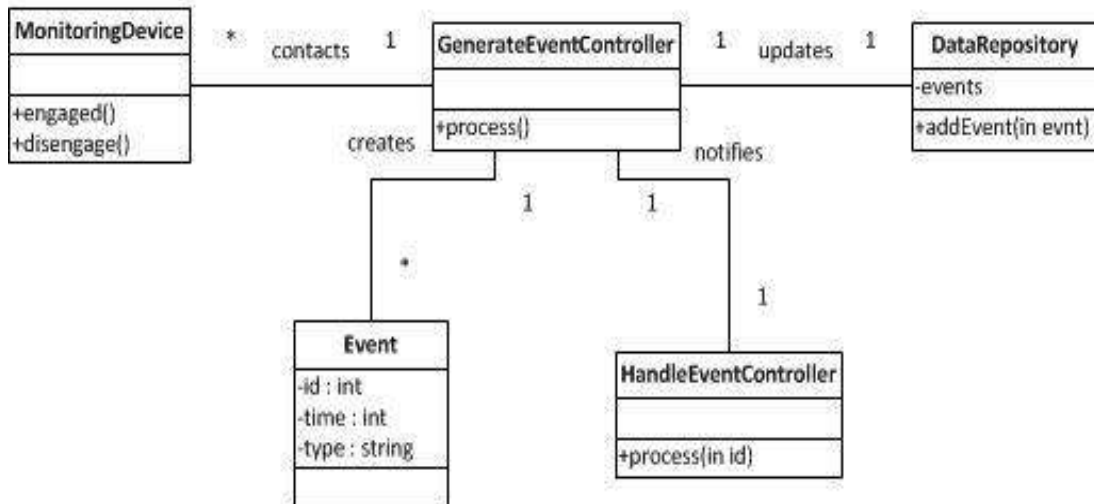
Figure 62: The workstation classes of D3

### Server Design Class Diagram



**Figure 63: The server classes of D3**

### Monitoring Device Design Class Diagram



**Figure 64: The monitoring device classes of D3**

## C.2.2 Design changes

### C.2.2.1 Design changes affecting the understandability quality attribute

When design size increased by six classes, D3's understandability decreased below its reference value. The encapsulation adaptation equation effectively counterbalances the resulting decrease in D3's quality as it is depicted in the simulation and the real results of D3.

#### 1) Simulated results

As it is illustrated in figure 65, D3 overcomes the decrease in understandability when encapsulation increases by a factor of six (i.e. DAM should equal 1 in all the newly added six classes).

#### 2) Real results

After increasing the design size of D3 and applying the adaptation equation of encapsulation, the obtained real understandability reaches again its reference and equals its simulated value (figure 66).

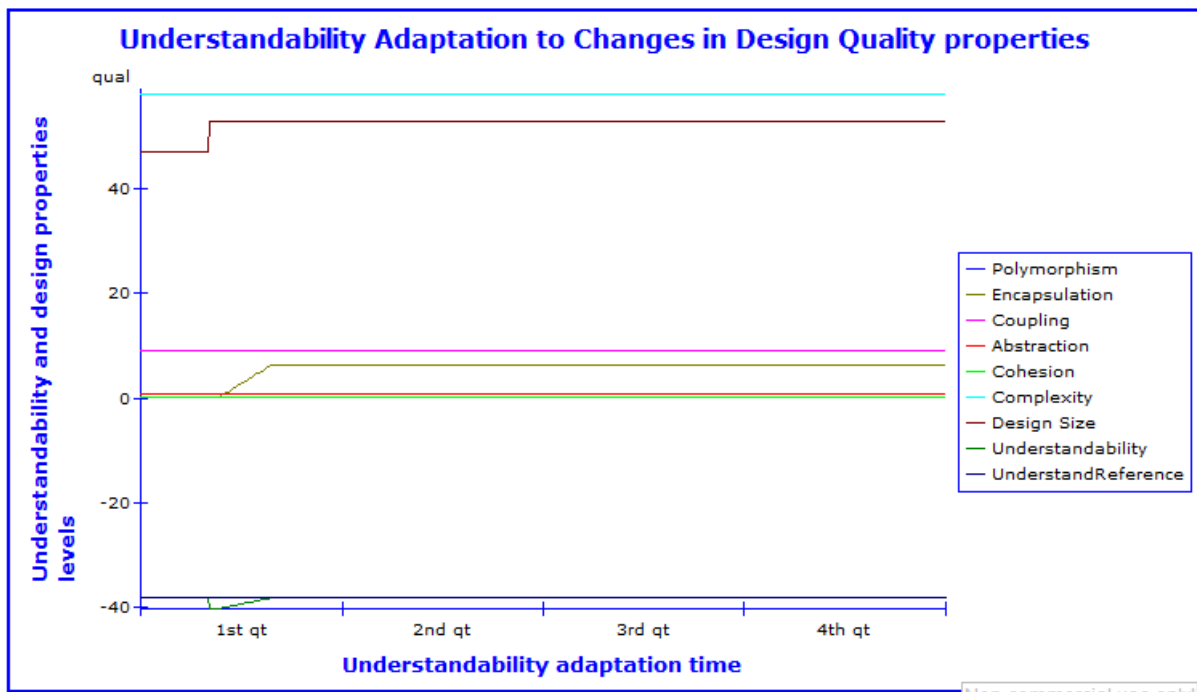
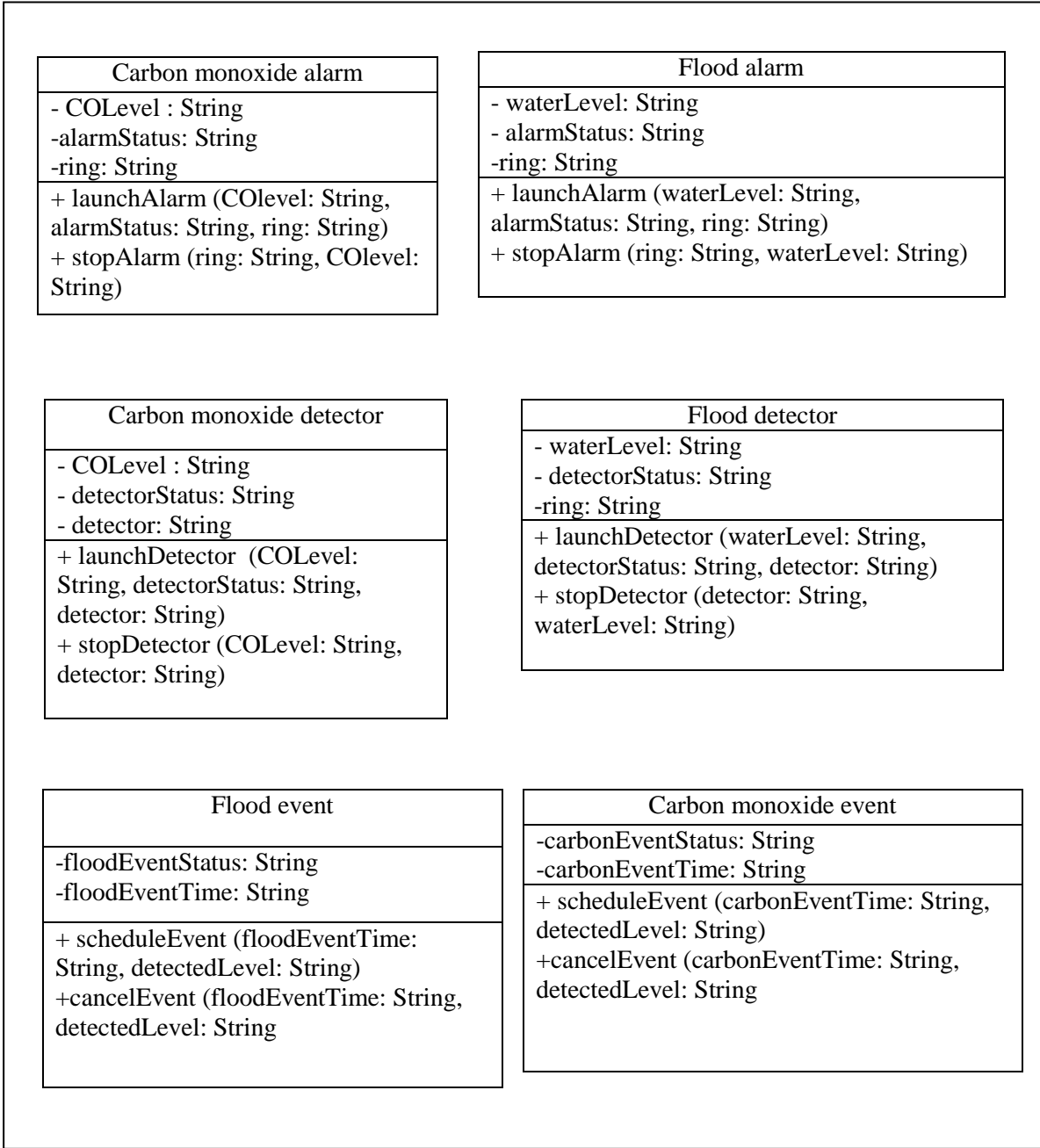


Figure 65: Understandability adaptation results of D3



**Figure 66: Understandability design change and adaptation in D3**



### **C.2.2.3 Design changes affecting the extendibility and the flexibility quality attributes**

The newly added classes in D3 from the previous design change were linked through inheritance relationships, which increase the coupling rate in the design. In figure 69, The “Alarm” class is the ancestor of the “Fire alarm” and the “Intrusion alarm” classes. In addition, the “Carbon monoxide detector” and the “Flood detector” classes are inheriting the characteristics of the “Monitoring device” class. Another inheritance relationship was established between the “Event” class and its children: “Carbon monoxide event” and “Flood event” classes. The impact of increasing coupling on D3’s quality and its corresponding adaptation was both simulated and applied in the real design.

#### **1) Simulated results**

From figures 67 and 68, the increase in coupling led to a decrease in both the flexibility and the extendibility quality attributes. The simulation results show that polymorphism should be increased by a factor of six to overcome the decrease in quality.

#### **2) Real results**

Adding six polymorphic methods to D3 as suggested by the simulation counterbalances the decrease in quality caused by the increase in coupling (figure 69). The real values of extendibility and flexibility after adaptation nearly equal their simulated values.

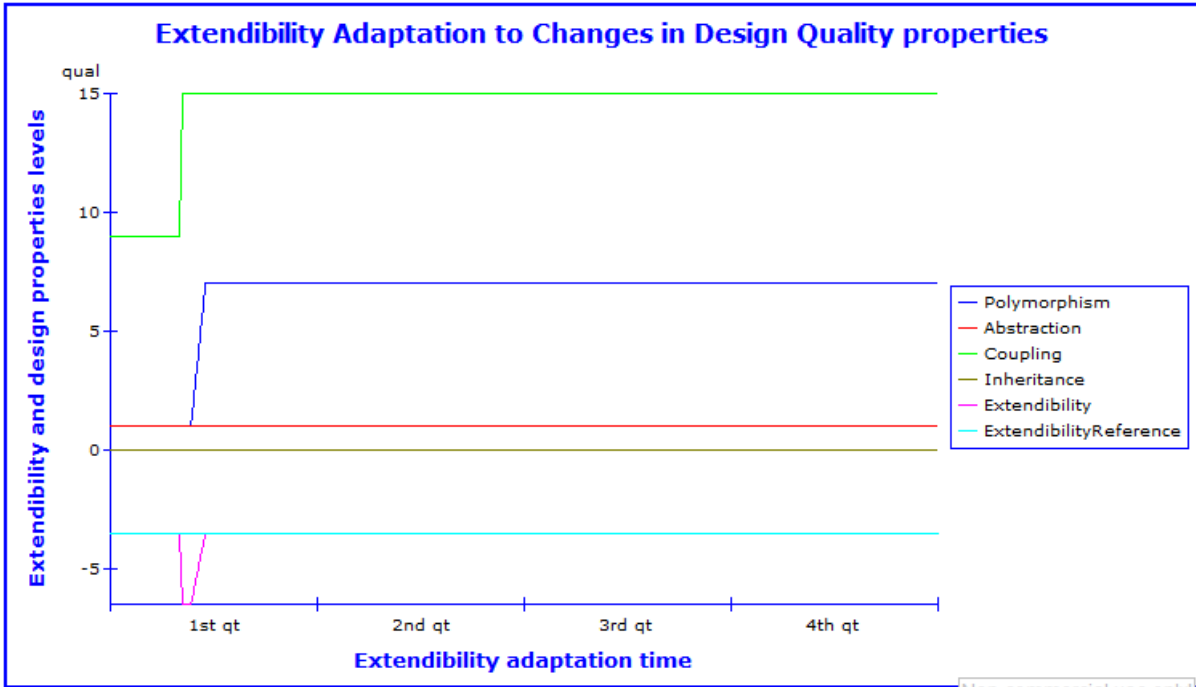


Figure 67: Extendability adaptation results of D3

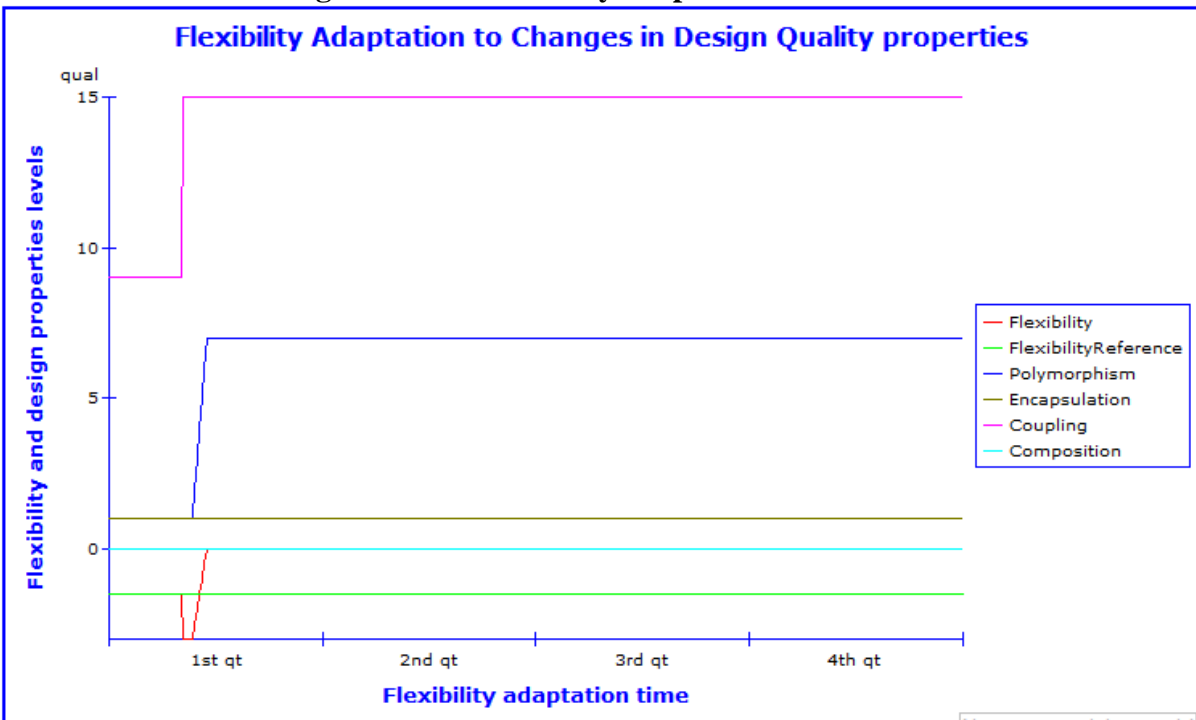
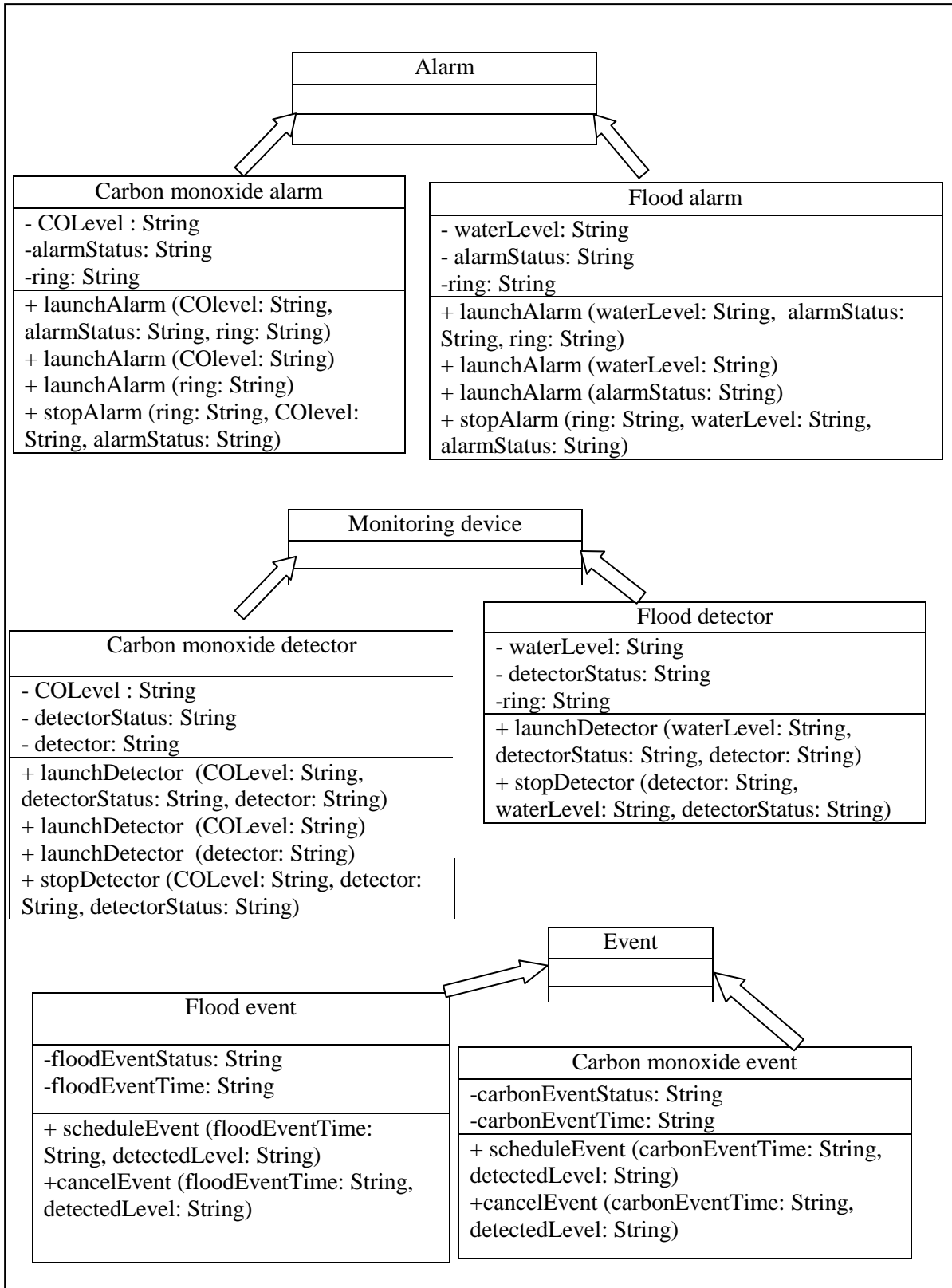


Figure 68: Flexibility adaptation results of D3



**Figure 69: Flexibility, extendibility, reusability, and functionality design changes with their adaptations in D3**

### **C.2.2.3 Design changes affecting the reusability and the functionality quality attributes**

The values of the reusability and the functionality quality attributes decreased below their reference values after dropping three classes from D3. Thus, the design size of D3 decreased when the following classes were omitted from the class diagram in figures 62-64: “Schedule Button”, “Edit Schedule”, and “Event Log Button”. Those design changes and their adaptations were both simulated and applied in the real design of D3.

#### **1) Simulated results**

The simulated results illustrated in figures 70 and 71 showed that cohesion should be increased by a factor of six to counterbalance the decrease in functionality and reusability.

#### **2) Real results**

Cohesion was maximized among six classes of D3 as suggested by the simulation to adapt the quality levels of reusability and functionality (figure 69). The real quality values of both quality attributes after adaptation almost equal their simulated values.

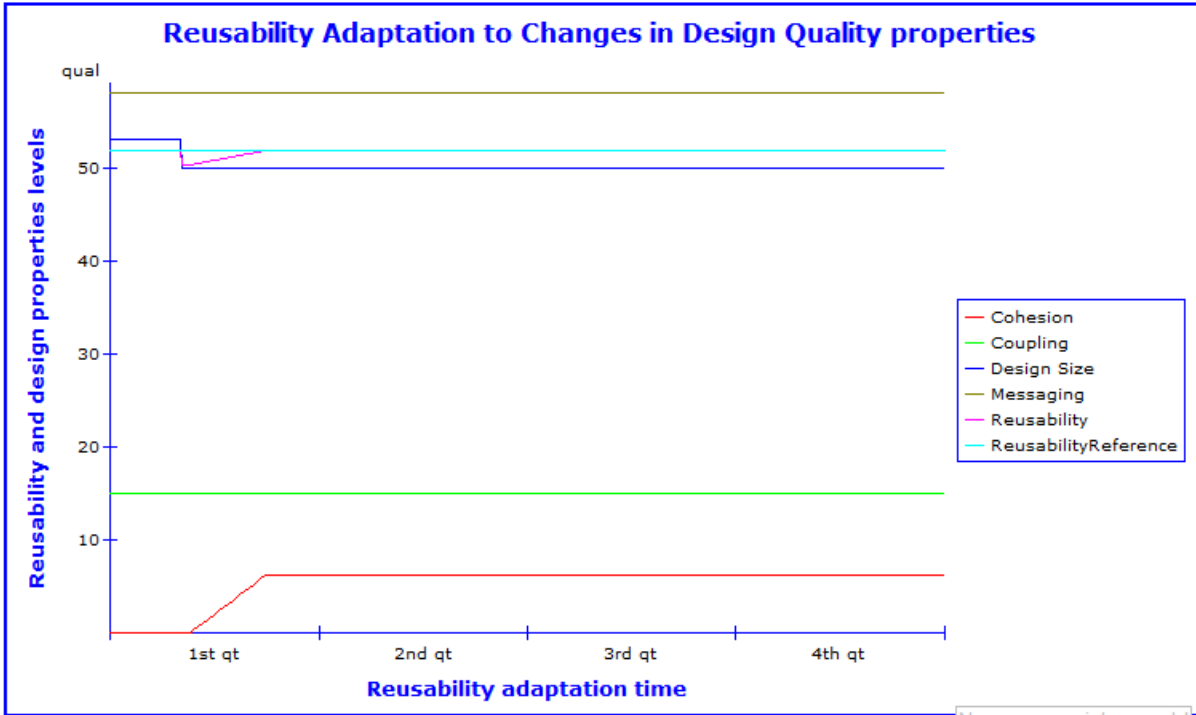


Figure 70: Reusability adaptation results of D3

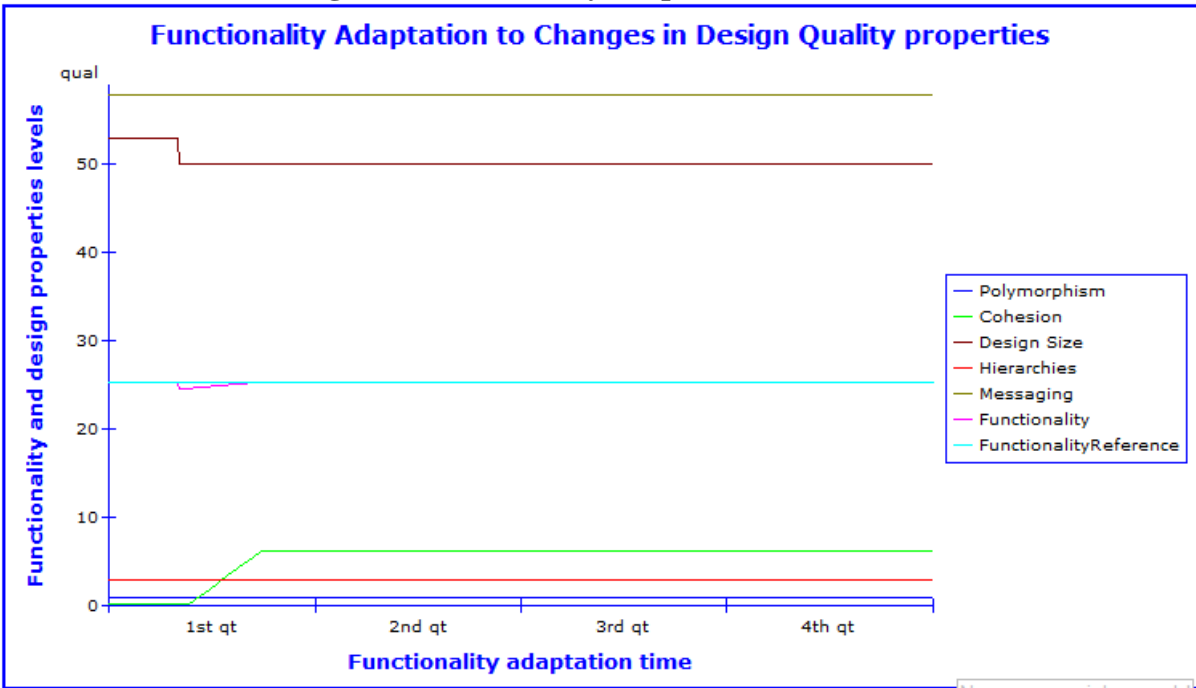
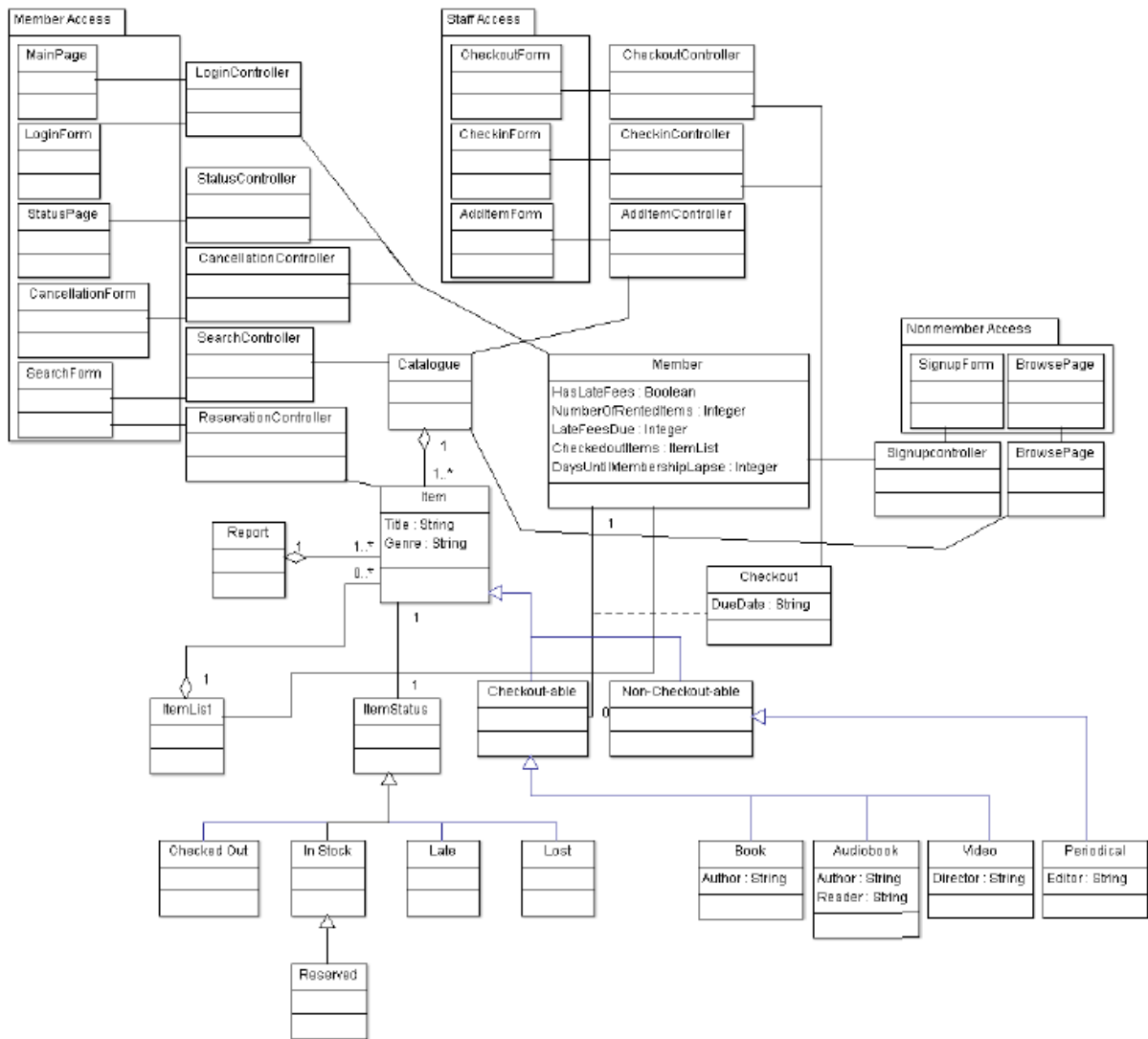


Figure 71: Functionality adaptation results of D3

### C.3 Design 4 (D4): Alexandria web-based library system

#### C.3.1 System description and reference quality values

The Alexandria library system deals with the operations of its members, staff and catalogue. Through D4's system, a member can benefit from many options such as checkout items, return items, and pay late fees. Staff members' options are multiple such as collect late fees, issuing, renewing, and cancelling memberships. The library catalogue can be managed through several functionalities such as checkout status and item type. D4 class diagram is represented in figure 72. Moreover, the reference values of D4 can be depicted in tables 16 and 17.



**Figure 72: D4 class diagram**



## **C.3.2 Design changes**

### **C.3.2.1 Design changes affecting the understandability quality attribute**

After increasing D4 design size by adding ten new classes, understandability decreased below its reference value. From both the simulation and the real results, encapsulation effectively brings back the value of understandability to its reference value.

#### **1) Simulated results**

According to the simulated results in figure 73, encapsulation should equal one in all the newly added classes to counterbalance the decrease in understandability.

#### **2) Real results**

Figures 74 and 75 illustrate the increase in D4's design size and its corresponding adaptation through encapsulation. The computed real value of understandability after adaptation almost equals its simulated value.

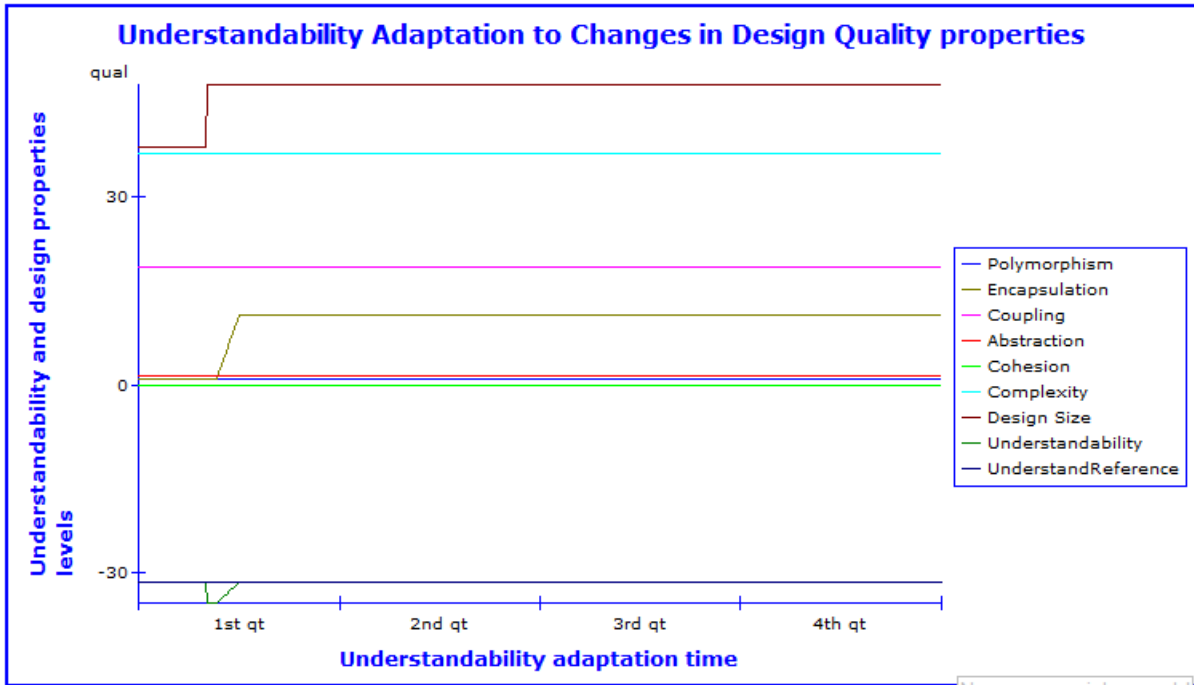
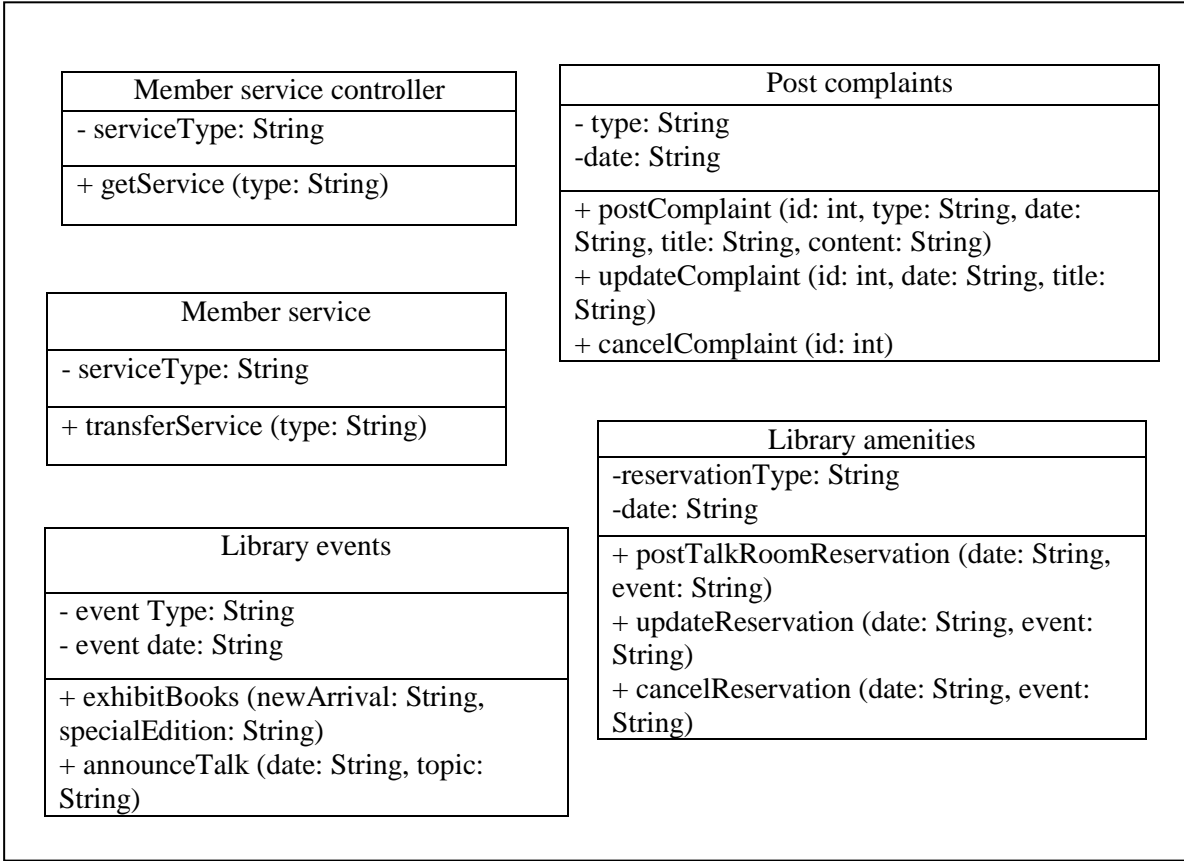
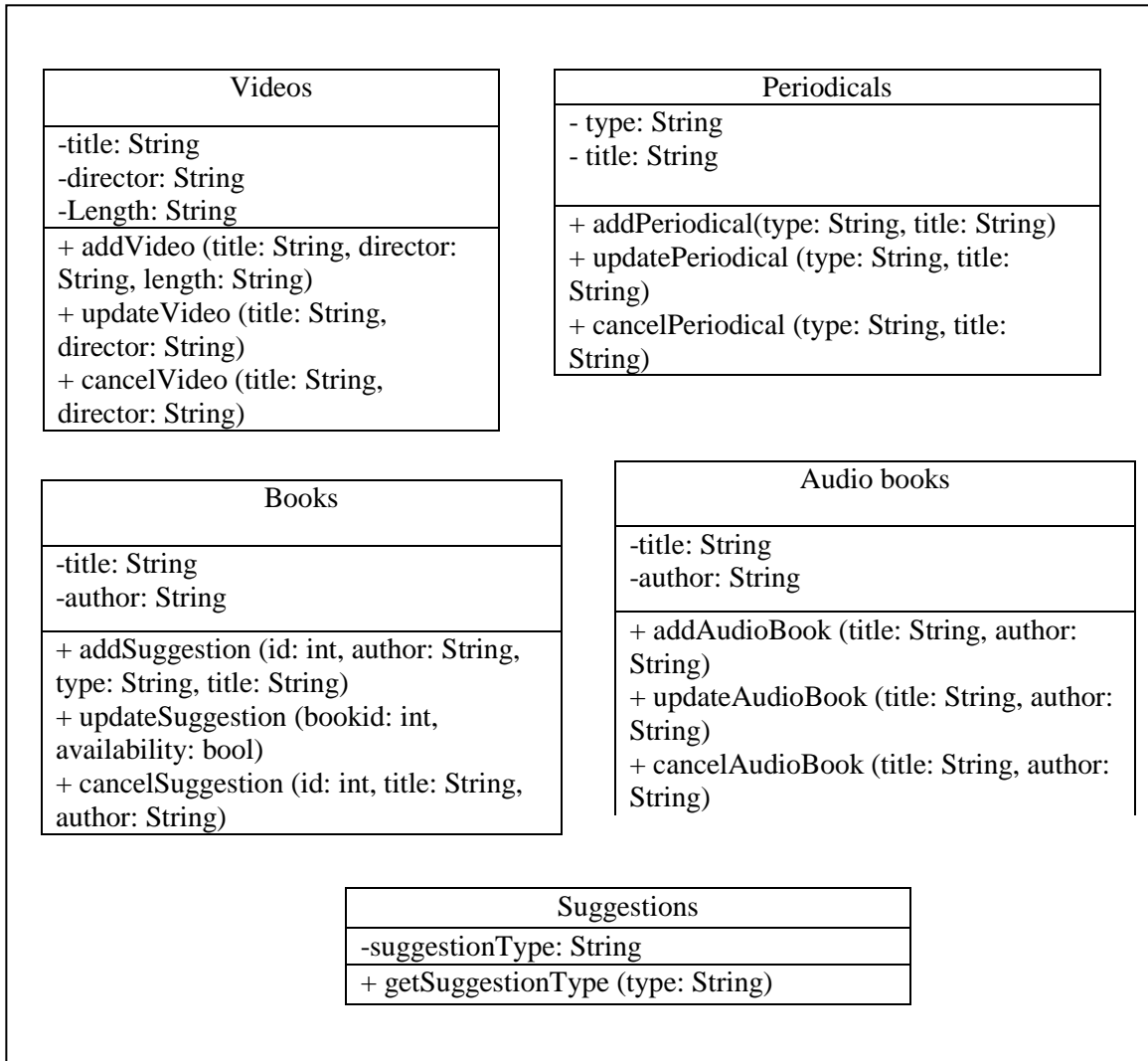


Figure 73: Understandability adaptation results of D4



**Figure 74: Understandability design change and adaptation in D4**



**Figure 75: Understandability design change and adaptation in D4**

### C.3.2.2 Design changes affecting the extendibility and the flexibility quality attributes

The eight added classes to D4 were linked together through two types of relationships: aggregation and inheritance. An aggregation relationship is identified between the “Member service controller” class and three classes namely “Post complaints”, “Library amenities”, and “Library events”. The remaining classes are linked through inheritance. The “item” class is the ancestor of the “Suggestions” class. The “Books”, the “Audio books”, the “Videos”, and the “Periodicals” classes inherit the characteristics of the “Suggestions” class. The impacts of those

design changes and their adaptations through polymorphism were experimented in the simulation and in the real design of D4.

### 1) Simulated results

After increasing coupling, flexibility and extendibility decreased below their reference values (Figures 76 and 77). From the simulation results, polymorphism should be increased by eight to compensate for the decrease in quality.

### 2) Real results

The increase in coupling and the suggested polymorphism adaptation value from the simulation were applied on the real class diagram of D4 (figures 78 and 79). The computed real values of flexibility and extendibility after adaptation nearly equal their simulated values.

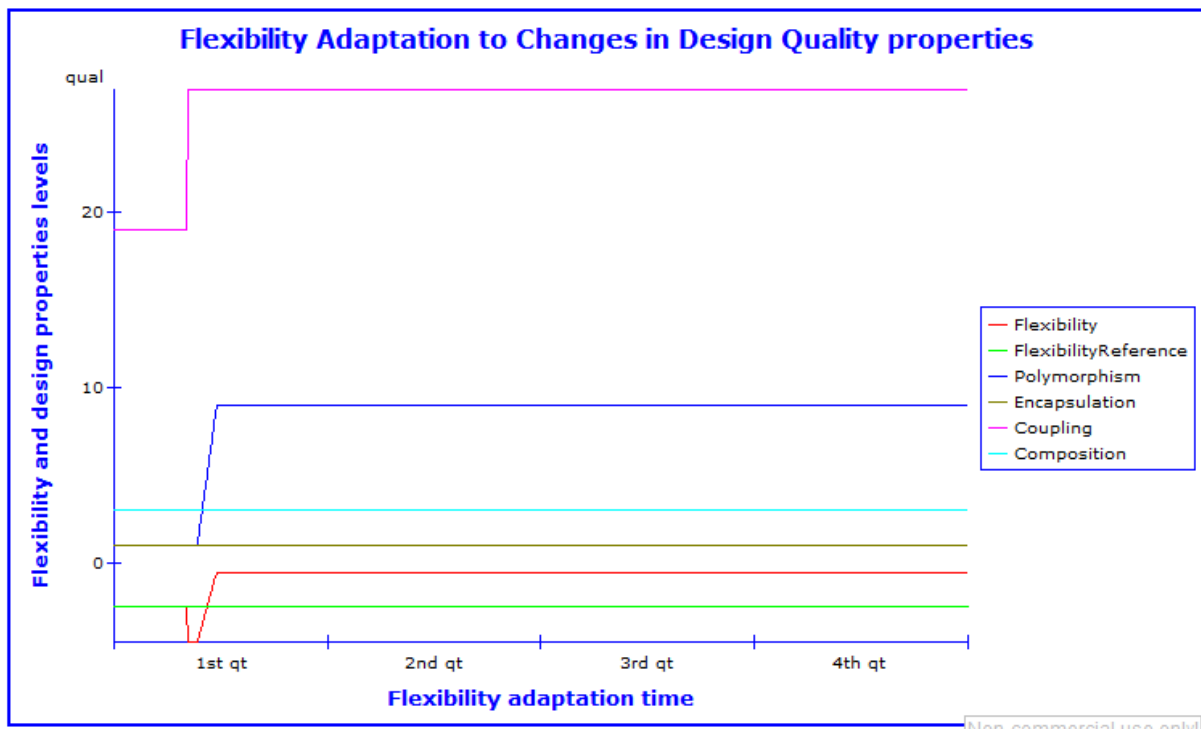


Figure 76: Flexibility adaptation results of D4

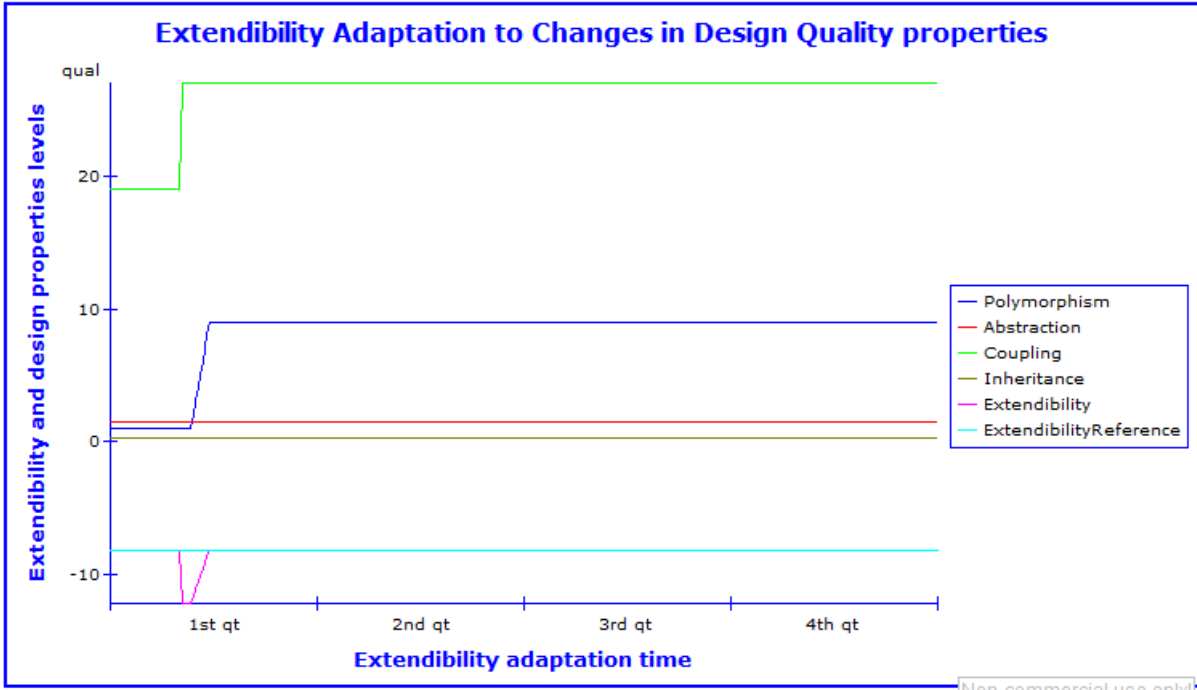
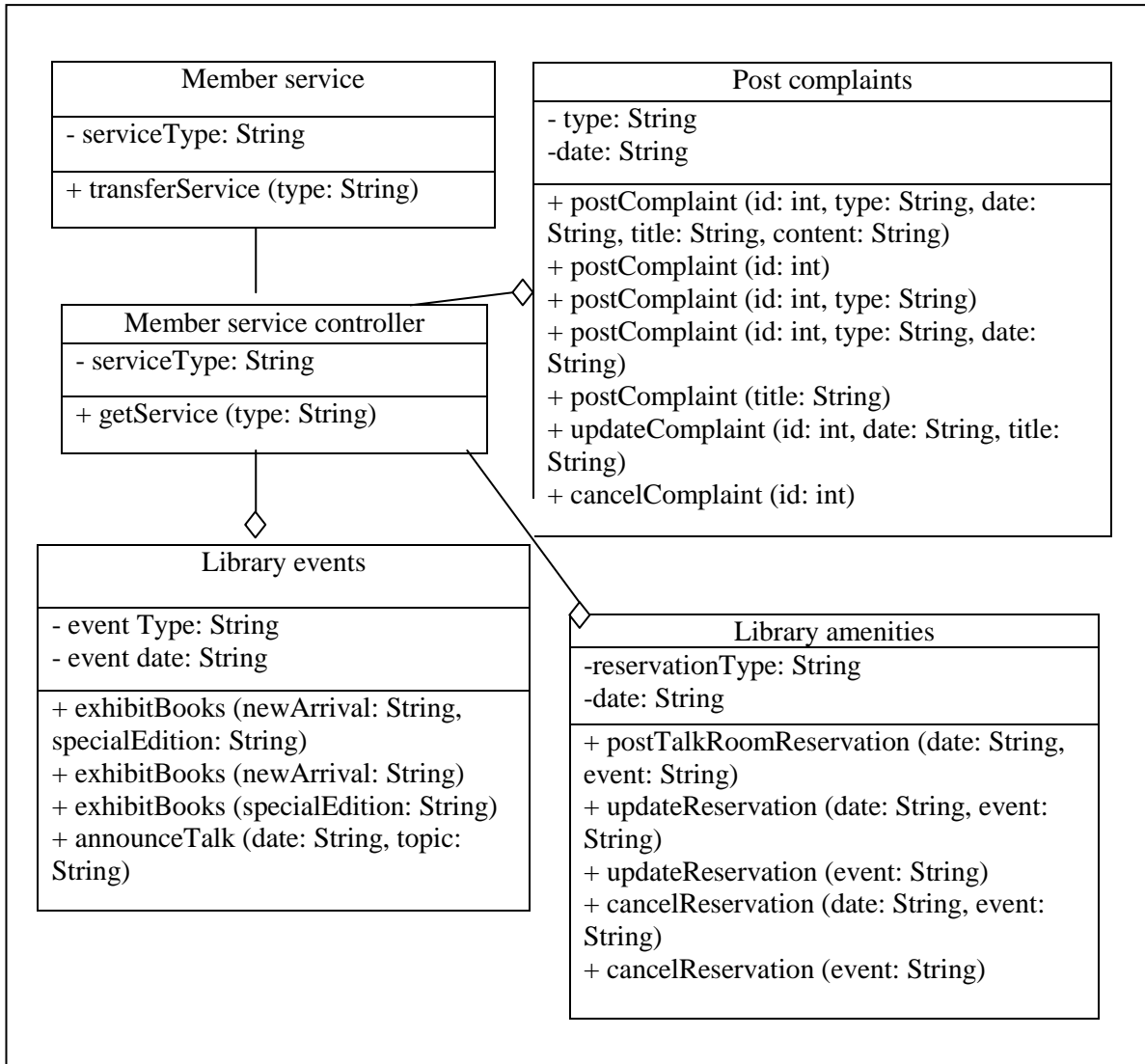
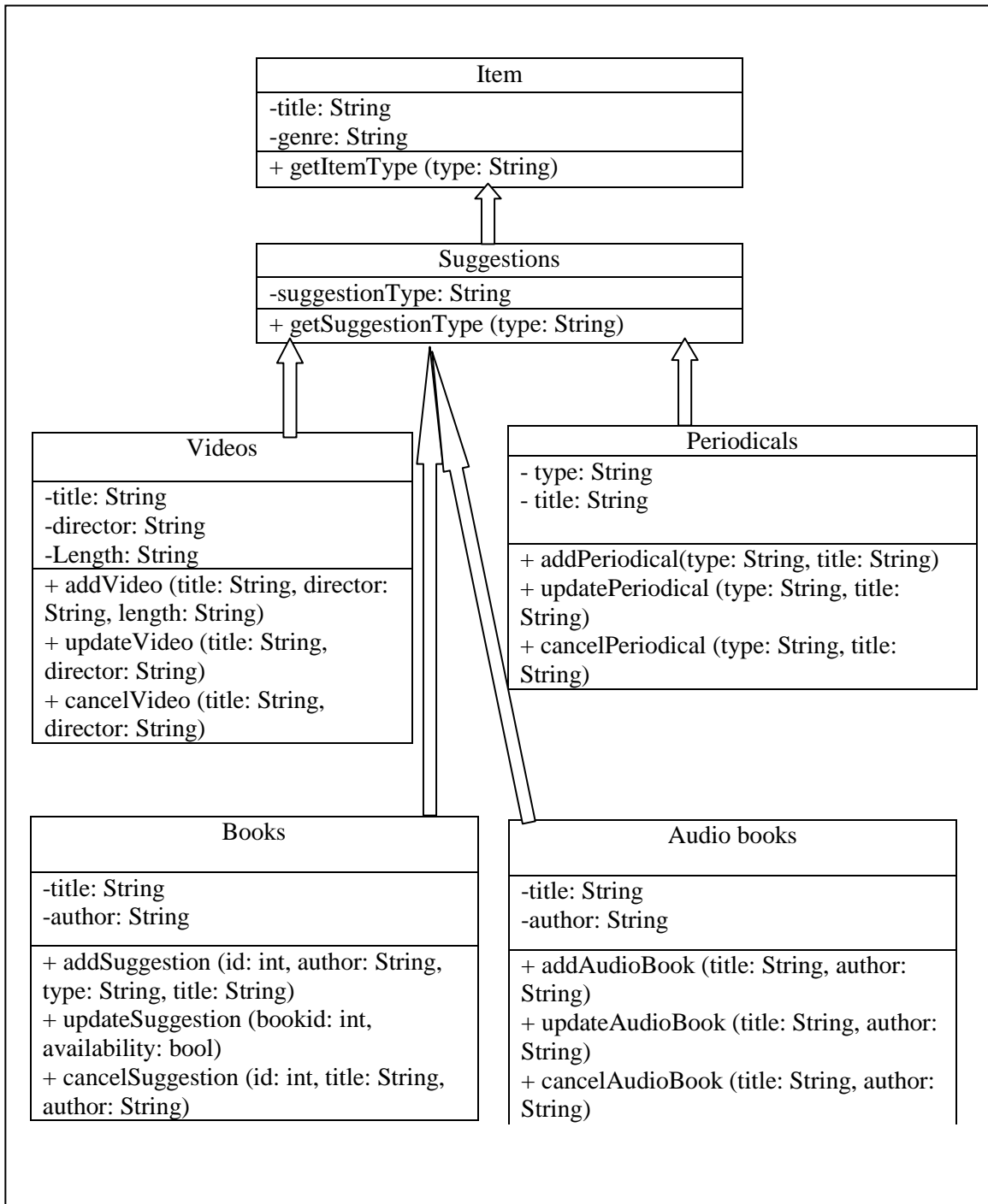


Figure 77: Extendibility adaptation results of D4



**Figure 78: Flexibility and extendibility design change and adaptation in D4**





**Figure 79: Flexibility and extendibility design change and adaptation in D4**

### C.3.2.3 Design changes affecting the reusability and the functionality quality attributes

D4's design size was decreased by dropping three classes namely "post complaints", "Library amenities", and "Library events". Then, the responsibilities of those classes were added to the "Member" class. Consequently, reusability and functionality decreased below their reference values and was effectively adapted by increasing cohesion as illustrated in the simulated and the real results.

#### 1) Simulated results

D4's cohesion should be maximized in six classes to bring back the value of reusability and functionality to their reference levels (figures 80 and 81).

#### 2) Real results

The application of the design change in D4 and the suggested cohesion level by the simulation make the real values of functionality and reusability identical to their simulated values (figures 82 and 83).

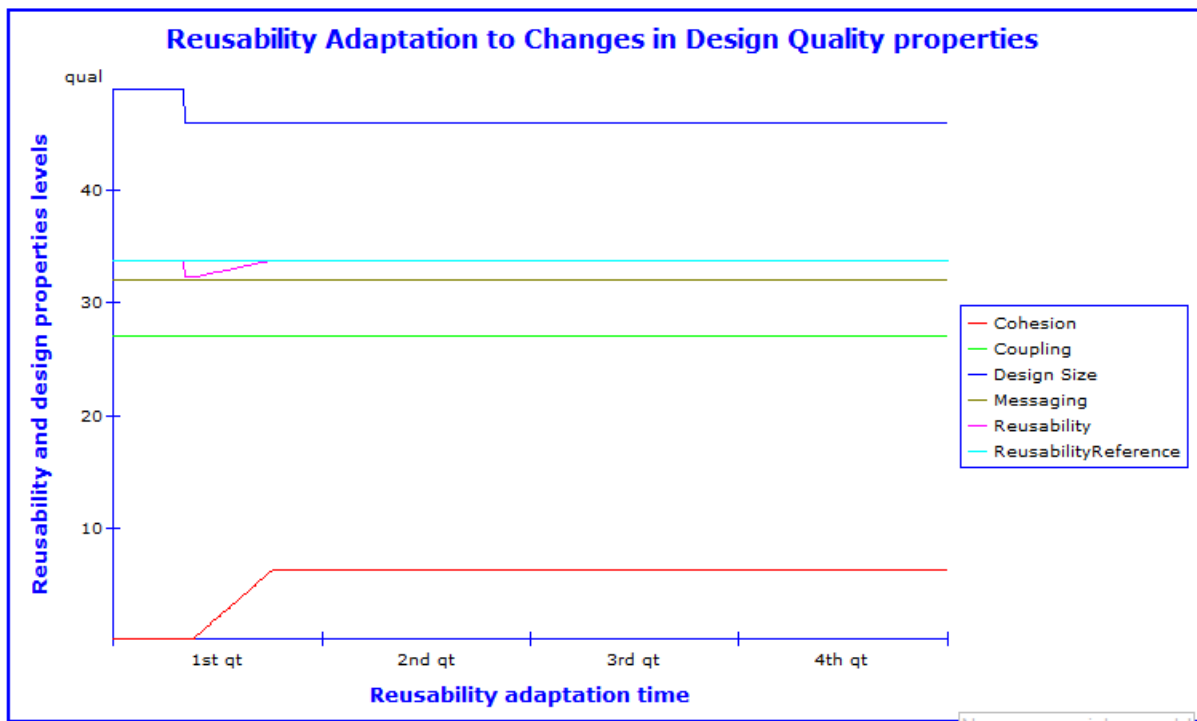


Figure 80: Reusability adaptation results of D4

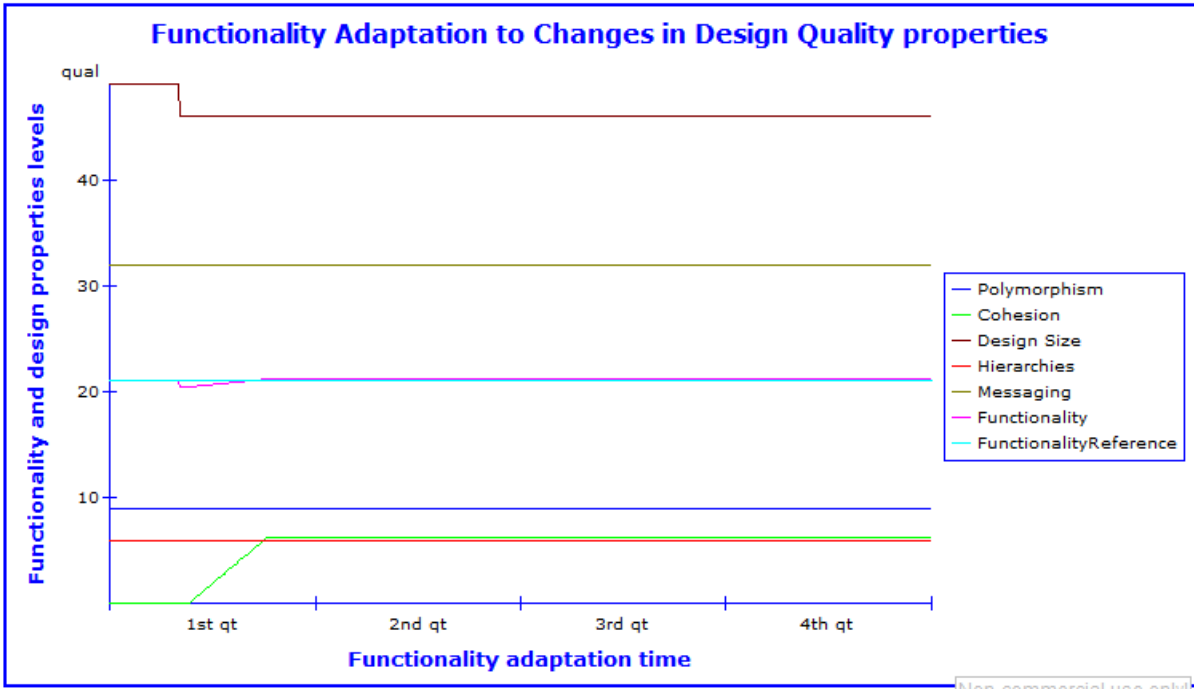
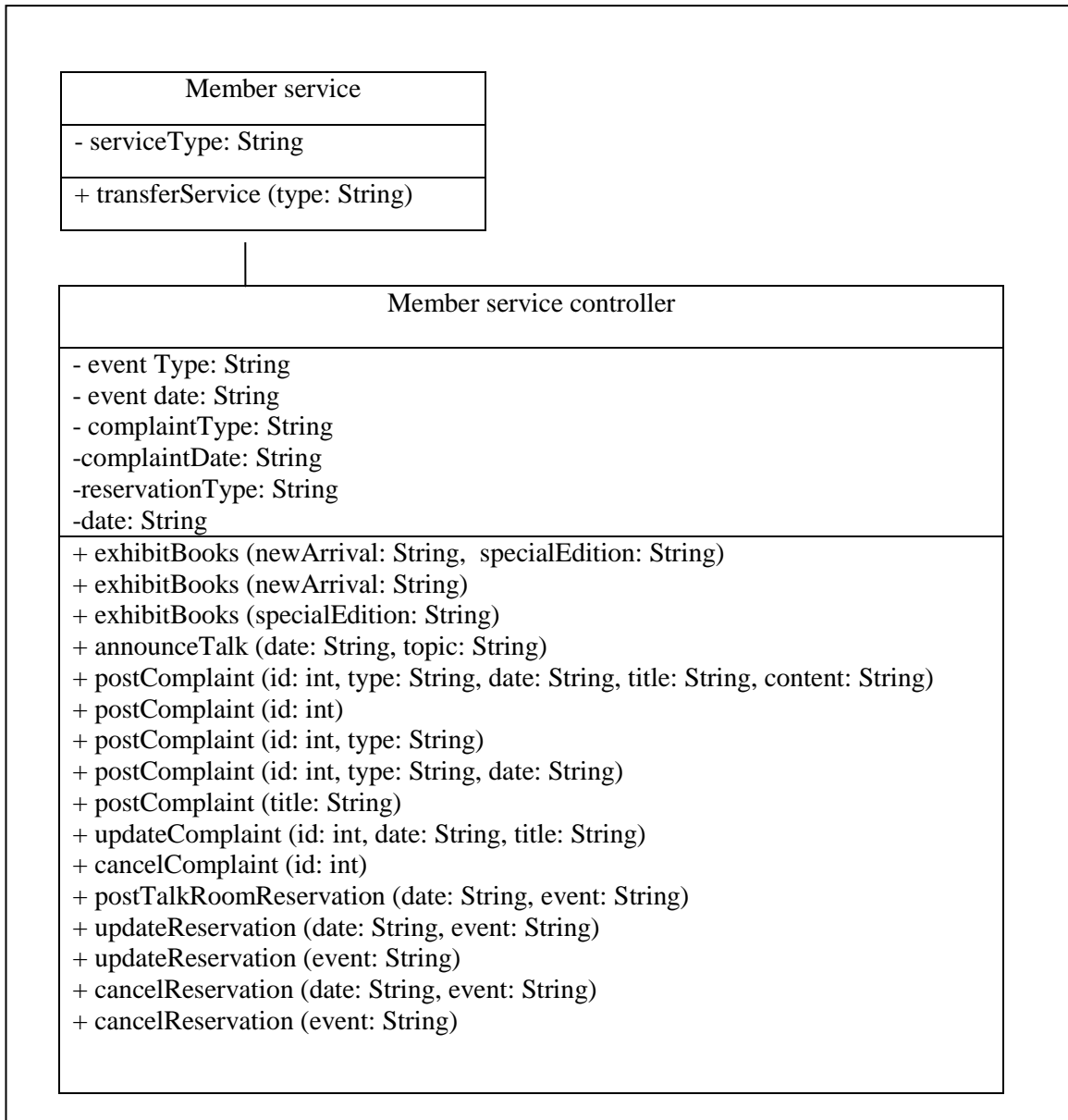
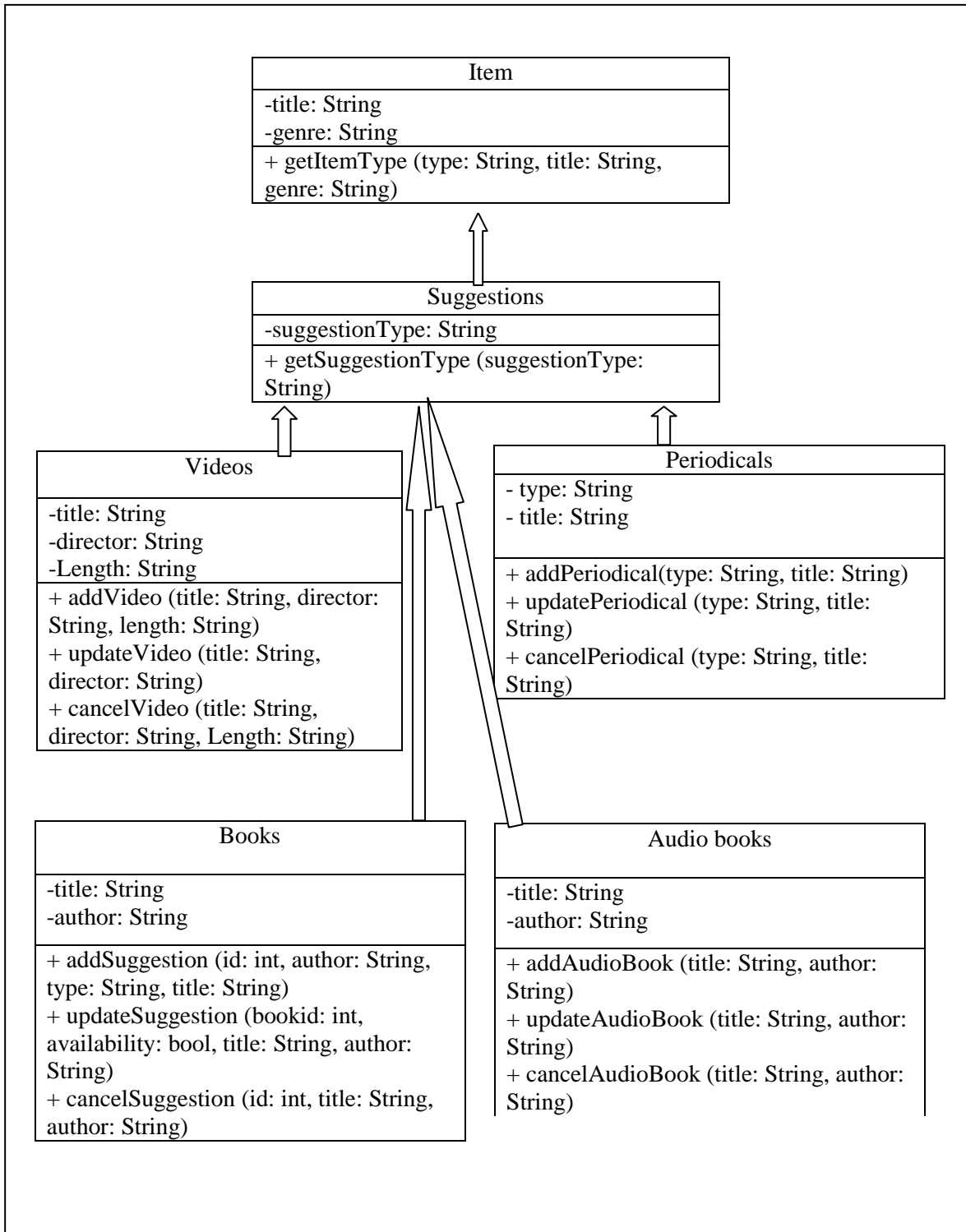


Figure 81: Functionality adaptation results of D4

Non-commercial use only!



**Figure 82: Reusability and functionality design change and adaptation in D4**



**Figure 83: Reusability and functionality design change and adaptation in D4**

## **C.4. Design 5 (D5): Third eye Home security system**

### **C.4.1. System description and reference quality values**

Third eye enables users to set up a remote security system for their homes or businesses. The user can view streaming videos of his cameras while at work. The system can also take snapshots of specific areas of the user's home and email them to him. If an intruder is detected, the user is notified through text messages and emails with a snapshot from the security camera. Through third eye, the user can set up the notification option by choosing the type of events and the means of contact when a threat is detected. The class diagram of D5 represents the different options that are available in the system (figure 84). The reference quality values of D5 can be accessed in tables 17 and 18.





## 1) Simulated results

To compensate for the decrease in understandability after increasing the design size of D5, encapsulation should be increased by a factor of 4 (figure 85).

## 2) Real results

The simulated results of D5 were applied in its class diagram as illustrated in figure 86. Encapsulation was maximized in the four classes (i.e. DAM = 1) and the real understandability value after adaptation is identical to its simulated value.

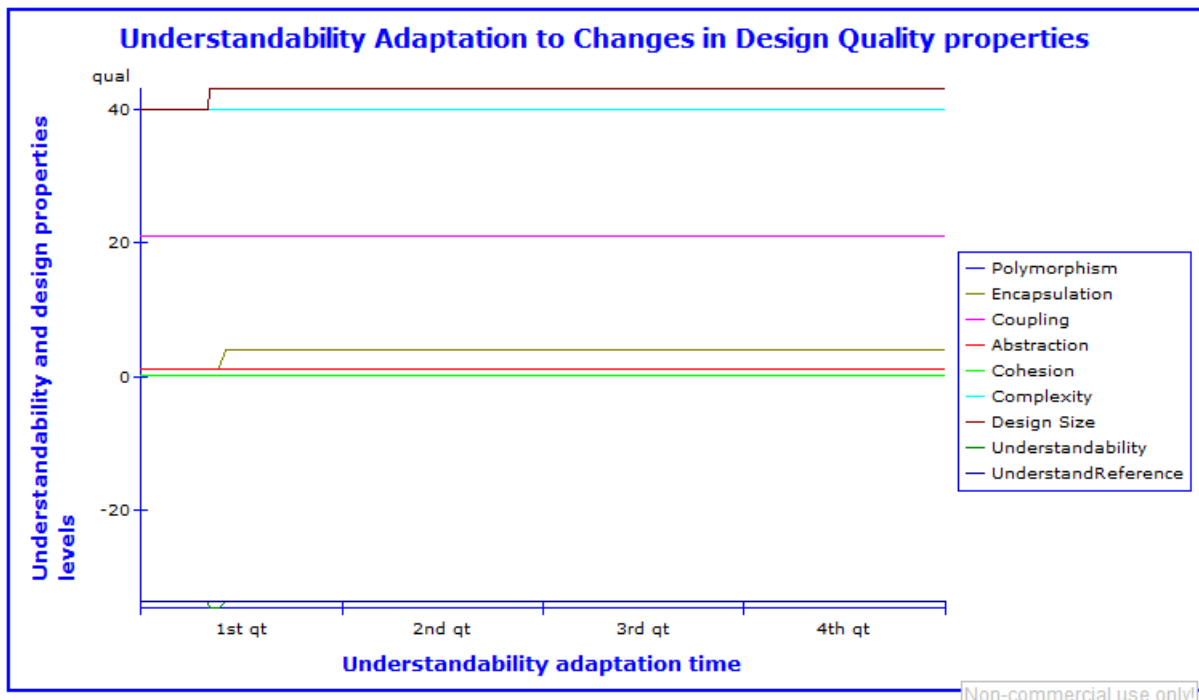
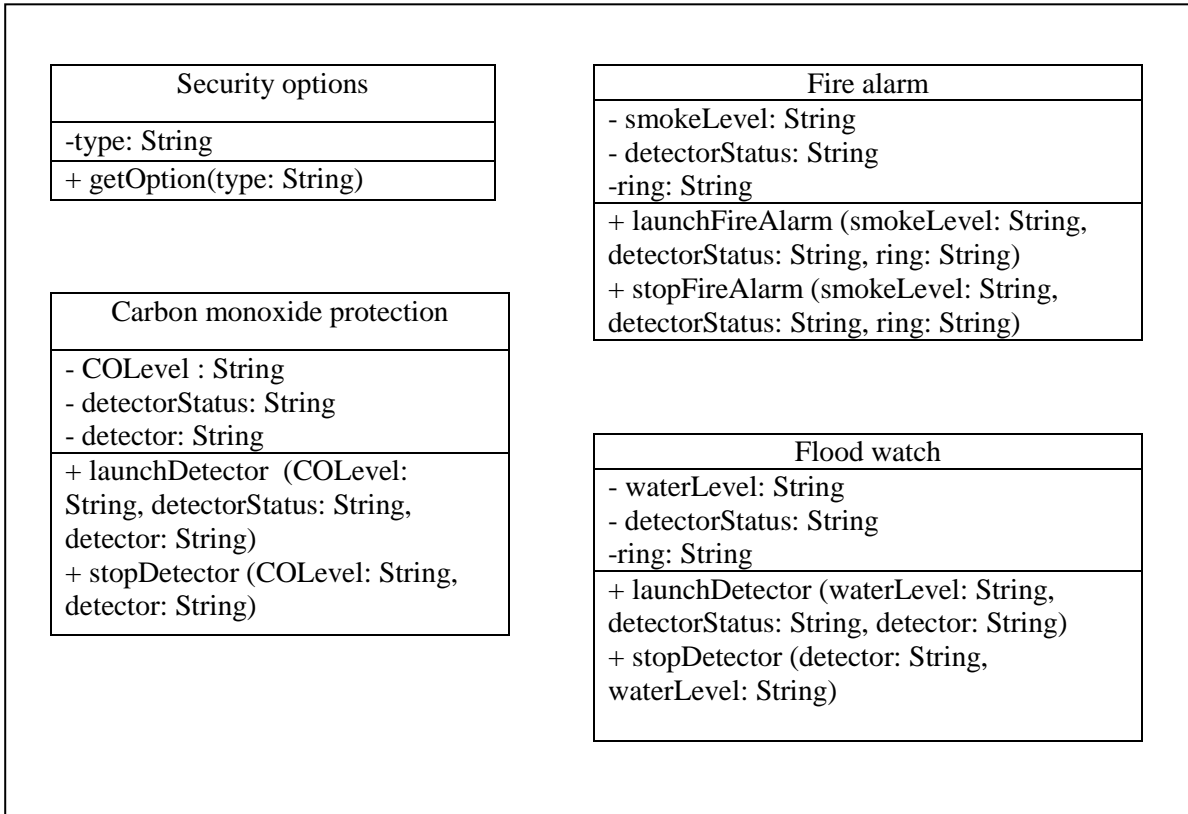


Figure 85: Understandability adaptation results of D5



**Figure 86: Understandability design change and adaptation in D5**

**C.4.2.2 Design changes affecting the flexibility and the extendibility quality attributes**

The value of coupling in D5 increased by a factor of four as it is represented in figure 89. The “security options” class became the ancestor of four classes: “Camera”, “Fire alarm”, “Flood watch”, and “Carbon monoxide protection”. As a consequence, the flexibility and the extendibility quality attributes decrease below their reference values. Those design changes and the applied adaptation strategies are described in the simulation and the real results of D5.

**1) Simulated results**

To minimize the decrease in understandability after increasing the design size of D5, polymorphism should be increased by a factor of 4 (figure 85).

## 2) Real results

The simulated results of D5 were applied in its class diagram as illustrated in figure 89. Four polymorphic methods were added to the class diagram of D5 such as “launchDetector (waterLevel: String)”. Then, the computed real flexibility and extendibility values after adaptation are similar to their simulated ones.

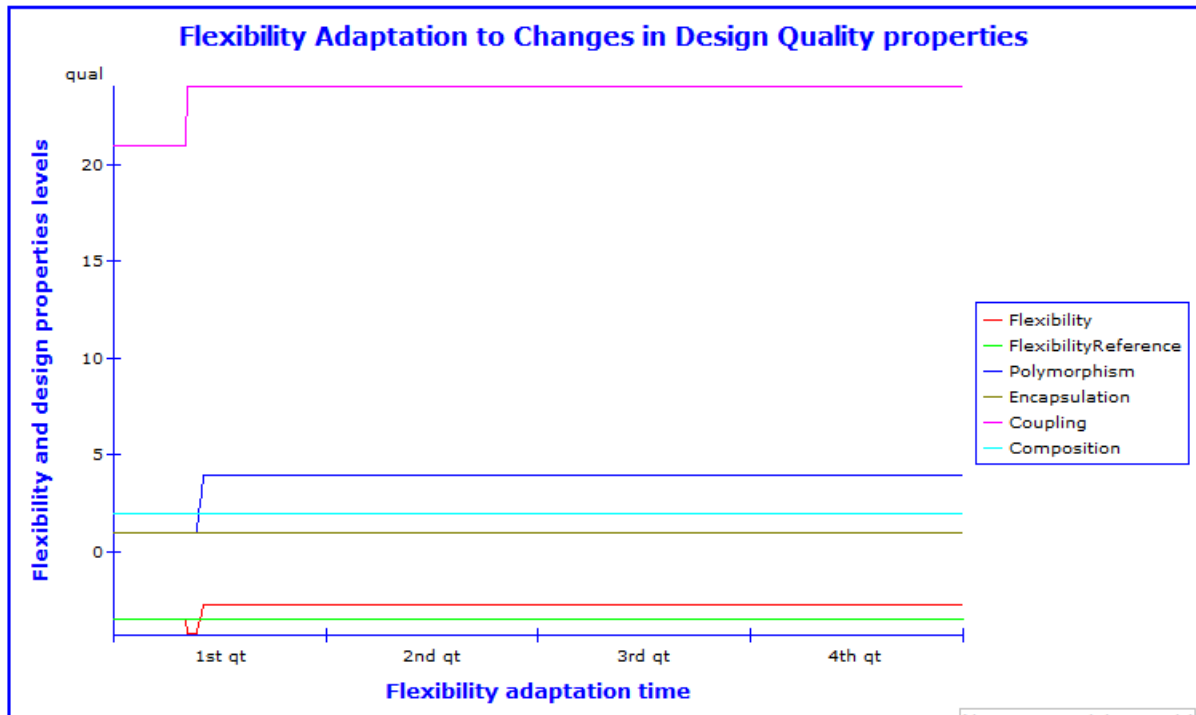


Figure 87: Flexibility adaptation results of D5

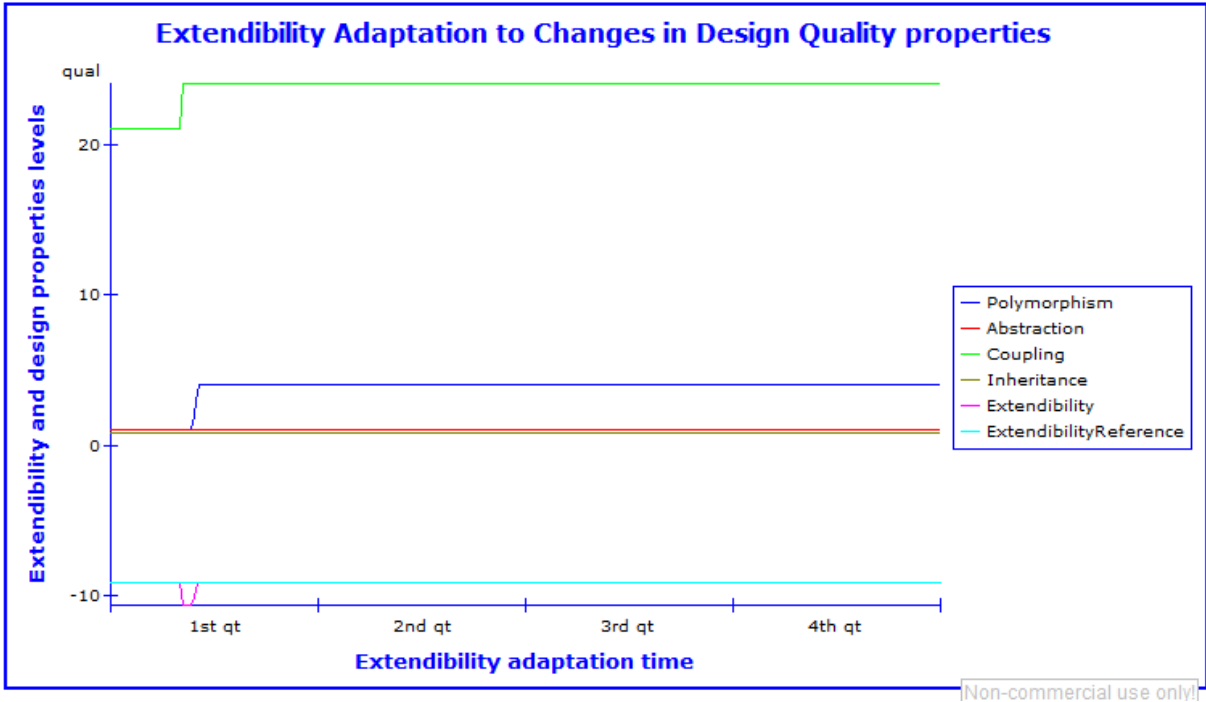
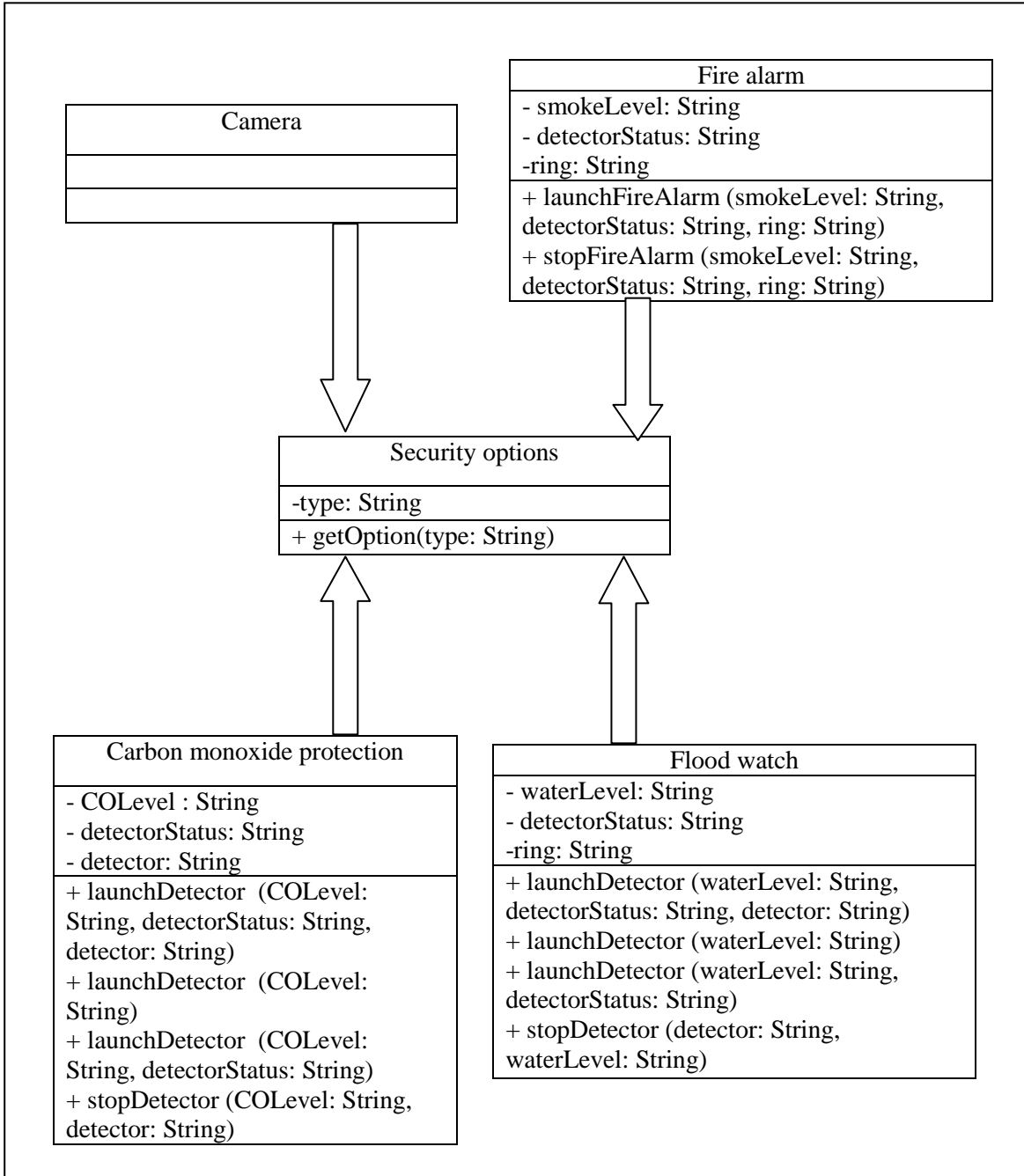


Figure 88: Extendibility adaptation results of D5

Non-commercial use only!



**Figure 89: Extendibility and flexibility design change and adaptation in D5**

### **C.4.2.3 Design changes affecting the reusability, the functionality, and the effectiveness quality attributes**

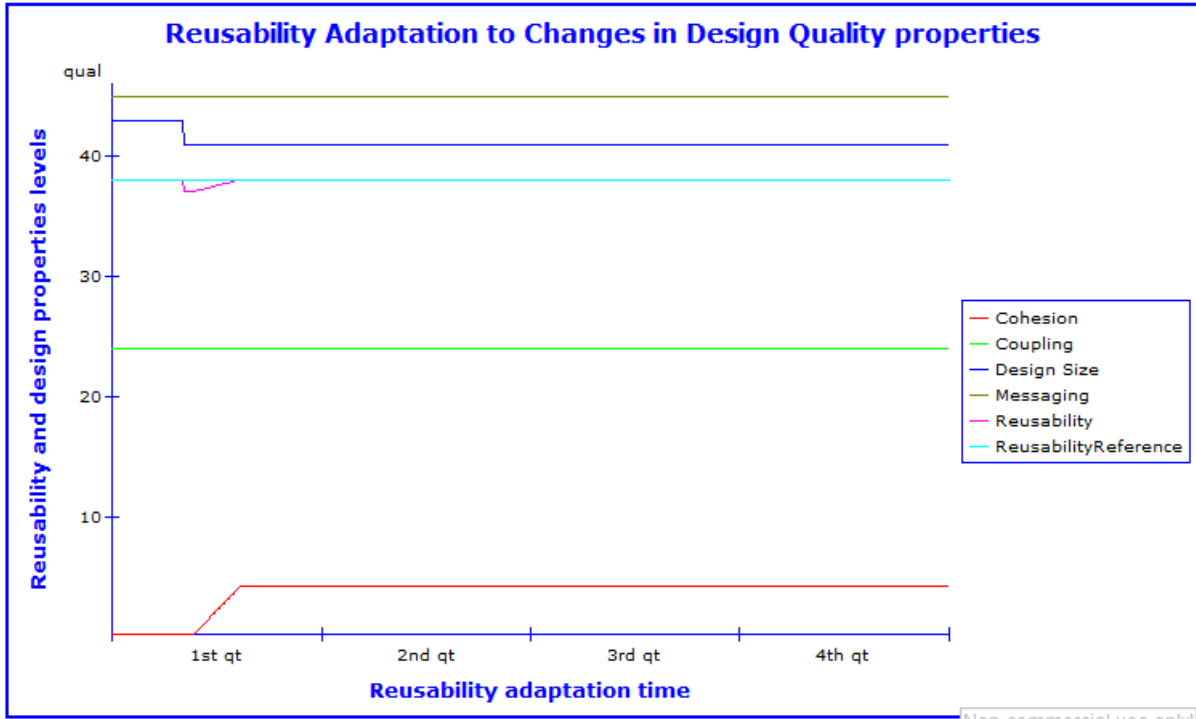
The design size of D5 was decreased by dropping two classes from its class diagram: “MenuInterface” and “DropDownMenu”. This design change led also to a decrease in composition since the omitted class diagrams are related to other classes through aggregation relationships. Therefore, those design changes affect the reusability, the functionality, and the effectiveness quality attributes as it is illustrated in the simulated and the real results.

#### **1) Simulated results**

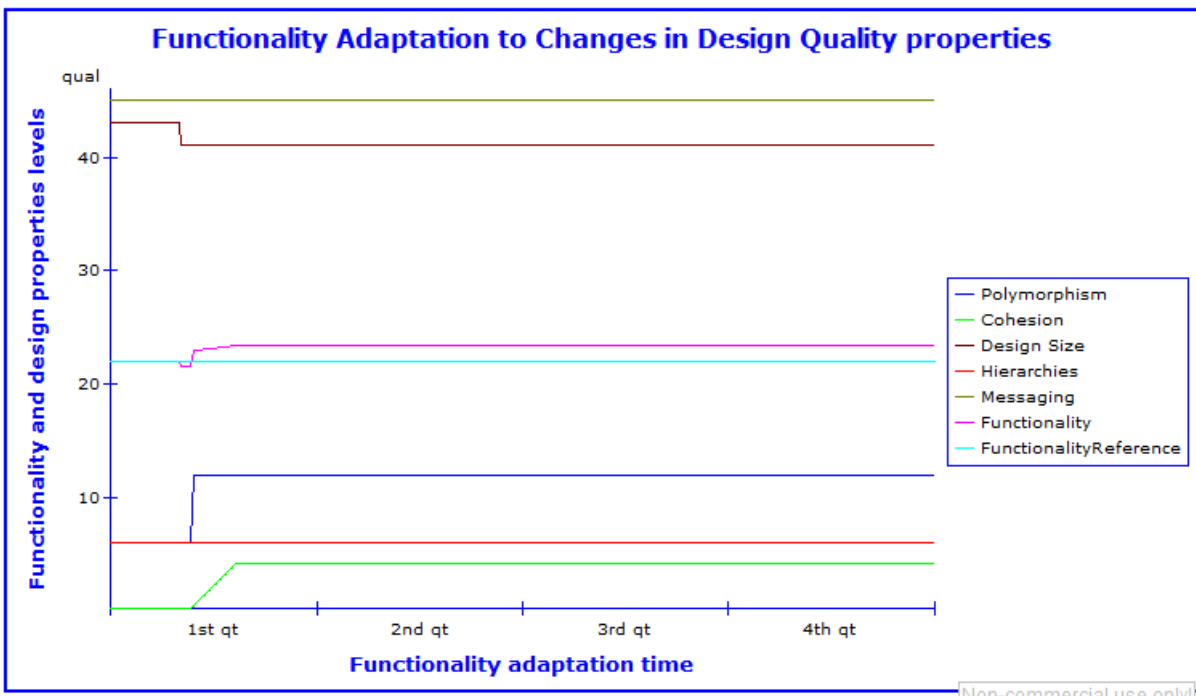
On the one hand, the decrease in design size leads to a drop in the values of the reusability and the functionality quality attributes. In this case, cohesion should be increased by four as an adaptation strategy (figures 91 and 92). On the other hand, the decrease in composition drops the value of effectiveness below its reference value. To overcome this decrease, polymorphism is increased by six (figure 93).

#### **2) Real results**

The simulated design changes and adaptations were applied in the class diagram of D5 as illustrated in figure 94. The computed real values of the quality attributes after adaptation nearly equal their simulated ones.



**Figure 90: Reusability adaptation results of D5**



**Figure 91: Functionality adaptation results of D5**



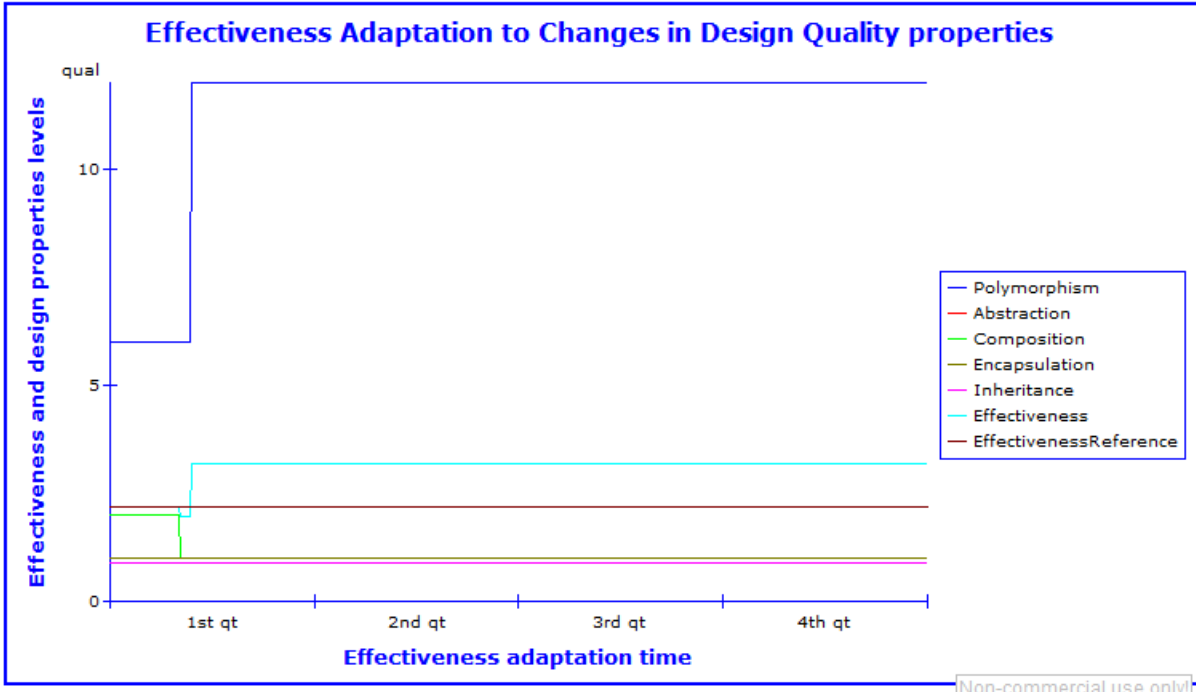
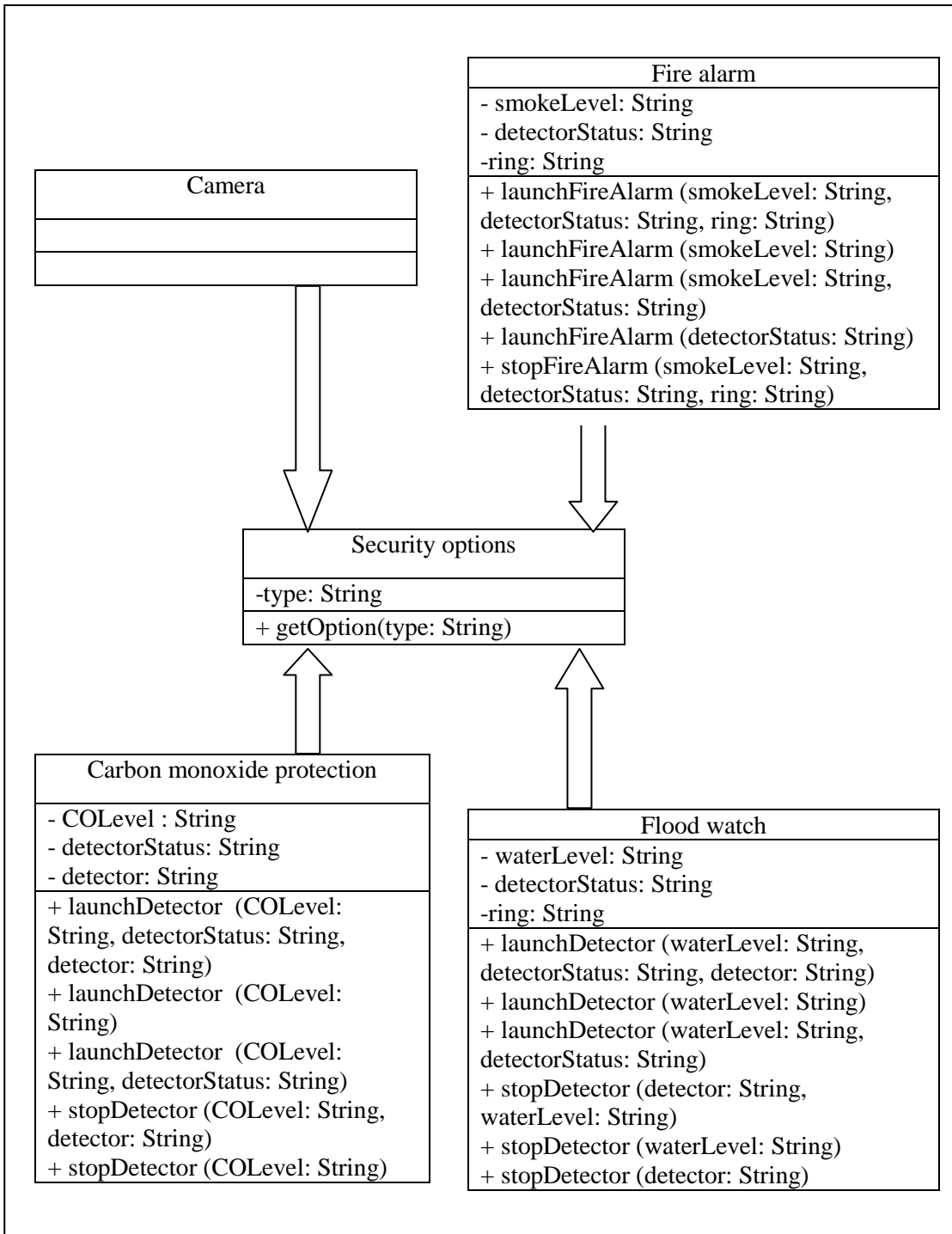


Figure 92: Effectiveness adaptation results of D5

Non-commercial use only



**Figure 93: Reusability, functionality, and effectiveness design change and adaptation in D5**

## **C.5 Design 6 (D6): HAMK UNIVERSITY online registration system**

### **C.5.1 System description and reference quality values**

HAMK is an online university registration system that can be used by students and staff members. Students have access to many operations such as searching for a specific class, viewing a listing of the taken classes, and checking time conflicts between classes. The different operations and characteristics of the system are represented in the class diagram of figure 94 and the reference values of D6 are illustrated in tables 17 and 18.

### **C.5.2 Design changes**

#### **C.5.2.1 Design changes affecting the understandability quality attribute**

Three new classes were added to D6, which increases the design size attribute and drops the understandability quality attribute below its reference value. The three classes are: “HAMKEntity”, “Transferred courses”, and “Registration holds” (figure 96). Adaptation through encapsulation brings back understandability to its reference value both in the simulation and the real results.

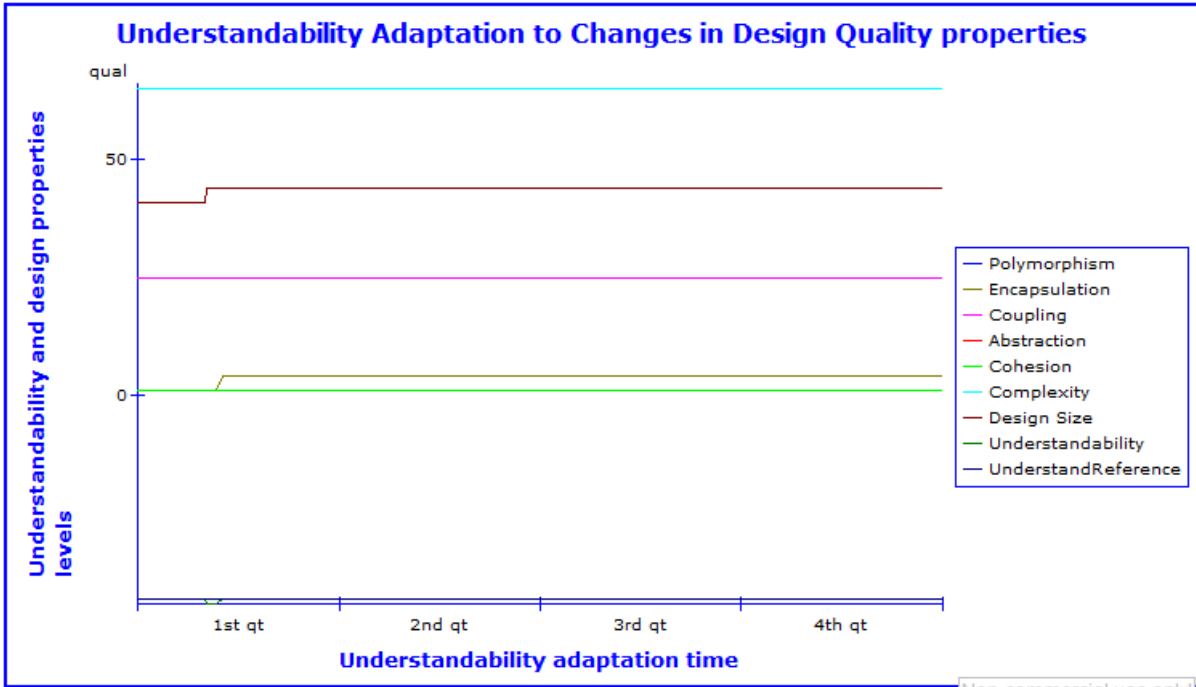
#### **1) Simulated results**

Encapsulation should be maximized in the three newly added classes (i.e. DAM = 1) (figure 95) to successfully adapt the value of understandability.

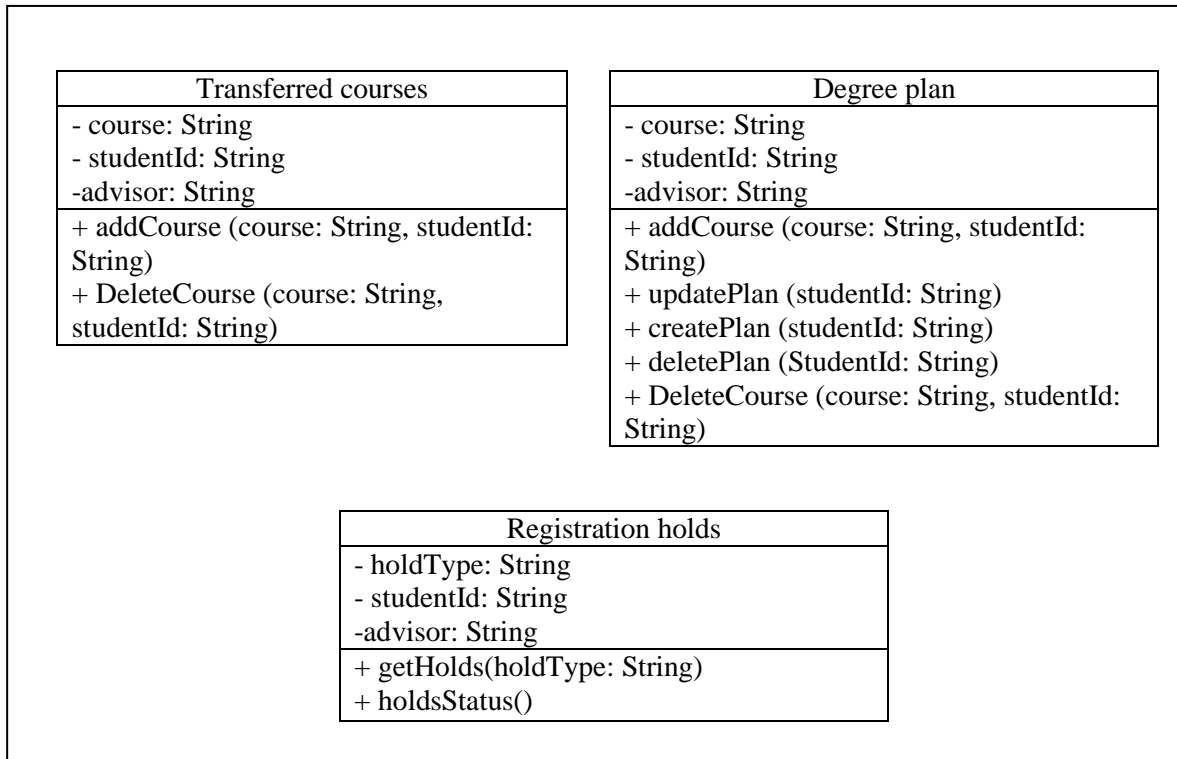
#### **2) Real results**

After maximizing encapsulation in the new three classes as represented in figure 96, the resulting value of understandability is similar to its simulated one.





**Figure 95: Understandability adaptation results of D6**



**Figure 96: Understandability design change and adaptation in D6**

### **C.5.2.2 Design changes affecting the flexibility and the extendibility quality attributes**

Two new inheritance relationships were identified in D5 where the “HamkEntity” class is the ancestor (figure 99). Those two relationships led to an increase in the coupling design property and both the flexibility and the extendibility quality attributes. Flexibility was adapted by increasing composition while extendibility was adjusted by increasing polymorphism as it is illustrated in the simulation and real results.

#### **1) Simulated results**

From the simulation graphs in figures 97 and 98, composition should be increased by a factor of three and polymorphism should be increased by a factor of two to adapt the quality values of flexibility and extendibility.

#### **2) Real results**

From figure 99, the simulated adaptations were successfully applied in the real design and the values of the resulting quality attributes nearly equal their simulated values. Two polymorphic methods such as “holdsStatus(status: String)” were added to adapt the value of extendibility. Three aggregation relationships (i.e. composition) were added such as the part-whole relationship between the classes “Registration holds” and “Enrollment summary”, which successfully adjusts the value of the flexibility attribute.

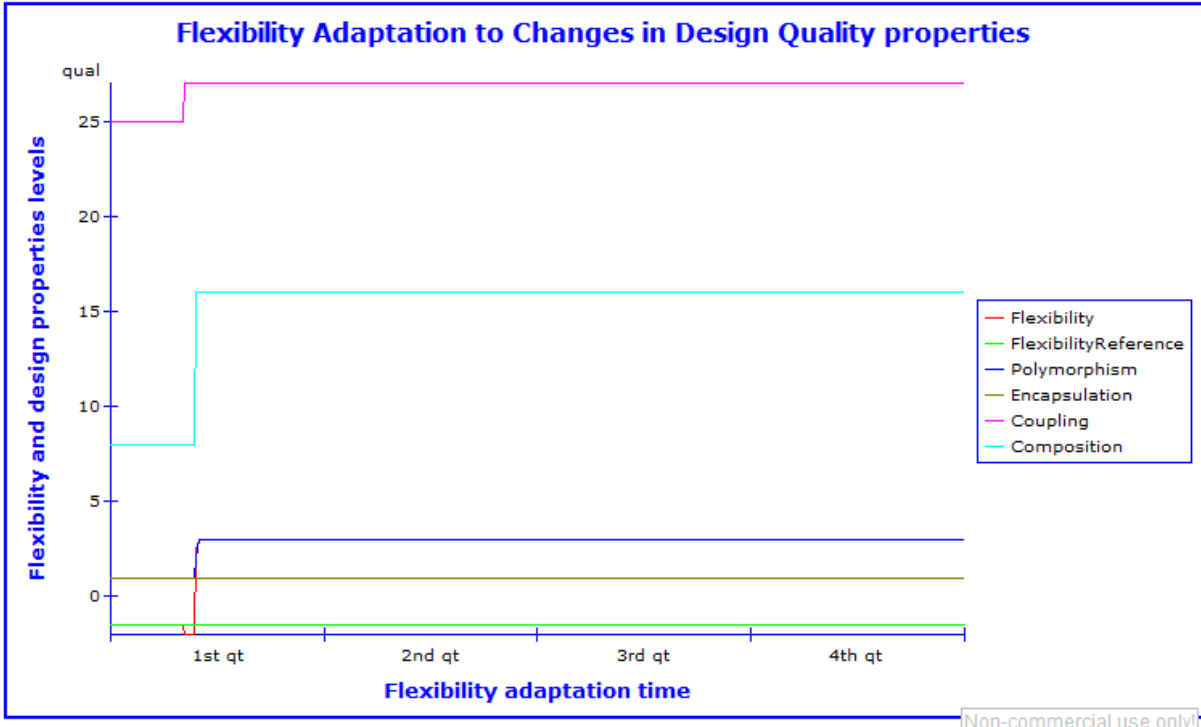


Figure 97: Flexibility adaptation results of D6

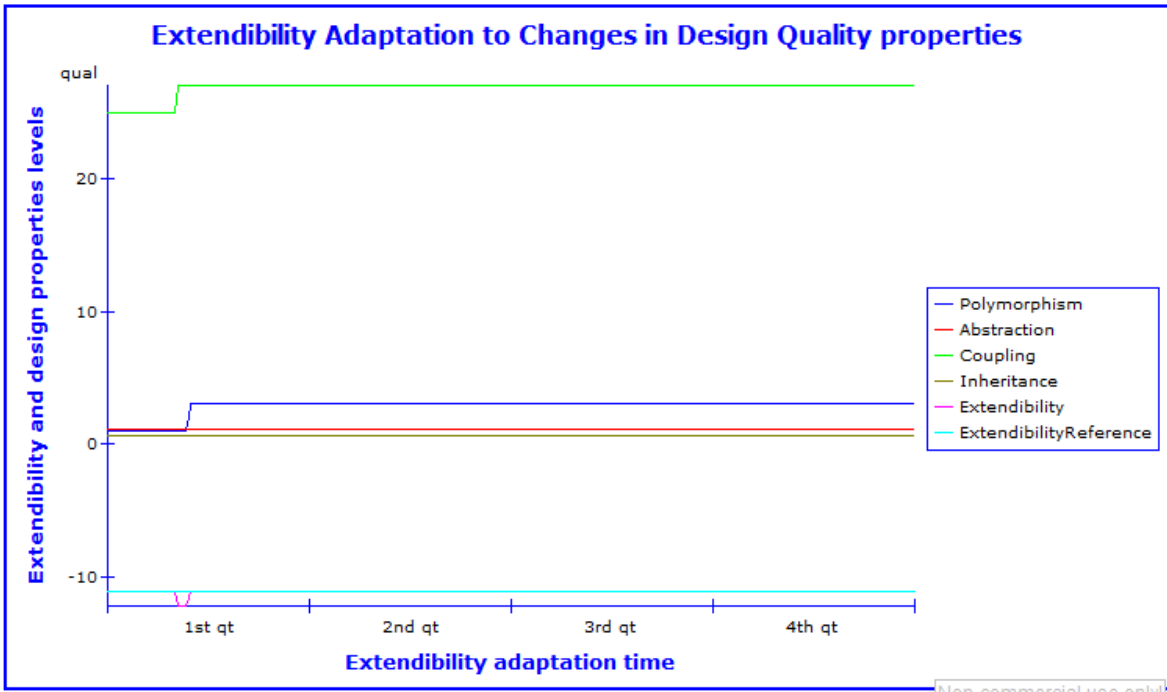
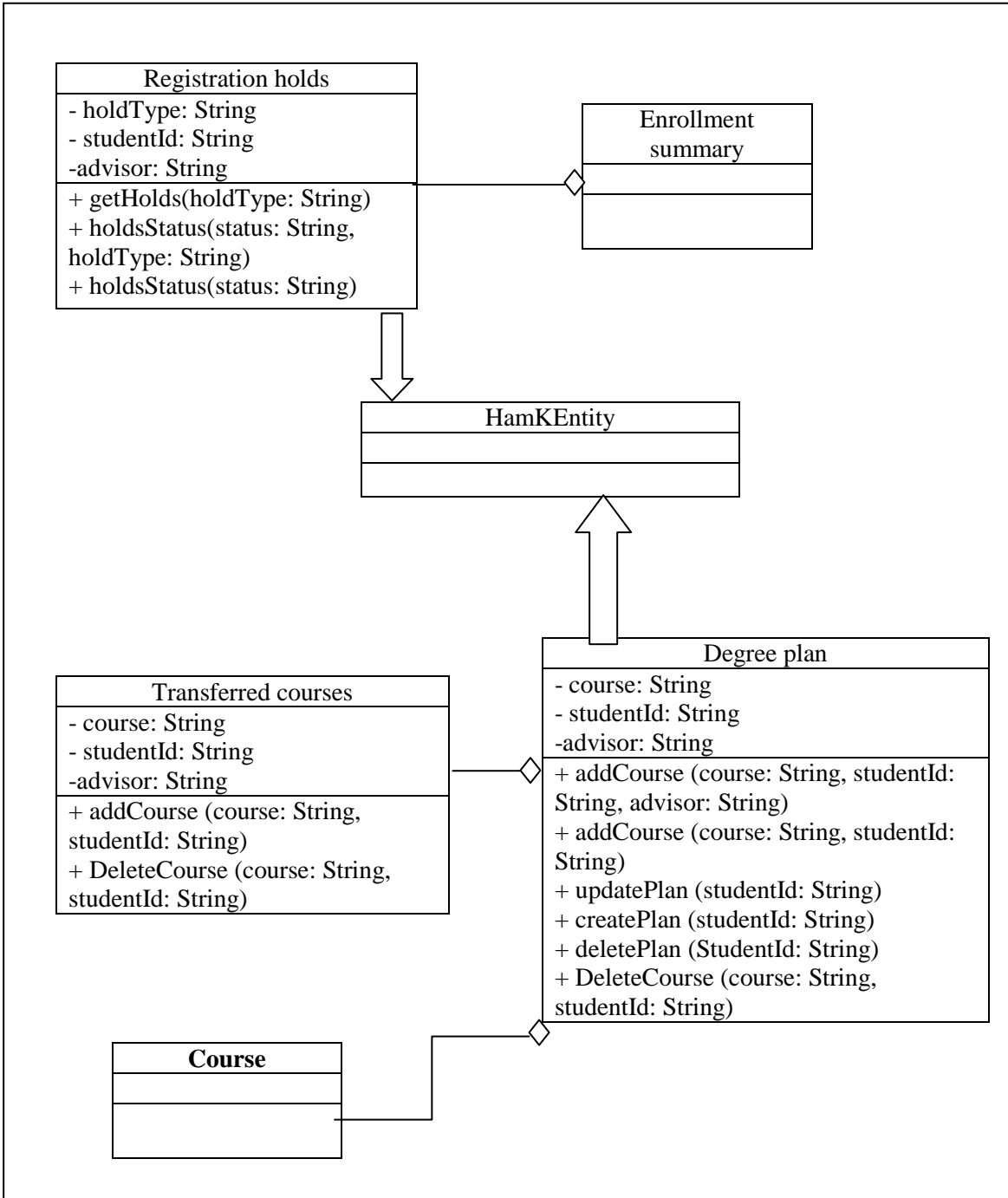


Figure 98: Extendibility adaptation results of D6





**Figure 99: Extendibility and flexibility design change and adaptation in D6**

### **C.5.2.3 Design changes affecting the reusability, the functionality, and the effectiveness quality attributes**

The last design changes in D6 decreased the values of the reusability, the functionality, and the effectiveness attributes below their reference values. The classes “Registration holds” and “Transferred courses” were deleted and their functions were merged in the classes “Registration” and “Course” respectively, which decreased the design size of D6 and its quality attributes reusability and functionality. In this case, cohesion is increased as an adaptation mean. The omission of classes from D6 led also to a deletion of their aggregation relationships, which decreased the value of effectiveness. The polymorphism adaptation equation is applied in this case to counterbalance the decrease in effectiveness.

#### **1) Simulated results**

As it is illustrated in figures 100-102, cohesion and polymorphism should be increased by factors of two and three respectively to compensate for the decrease in reusability, functionality, and effectiveness.

#### **2) Real results**

The simulated design changes and their corresponding adaptations were successfully applied on the real design of D6 as it is represented in figure 103. The real values of the quality attributes after adaptation are similar to their simulated ones.

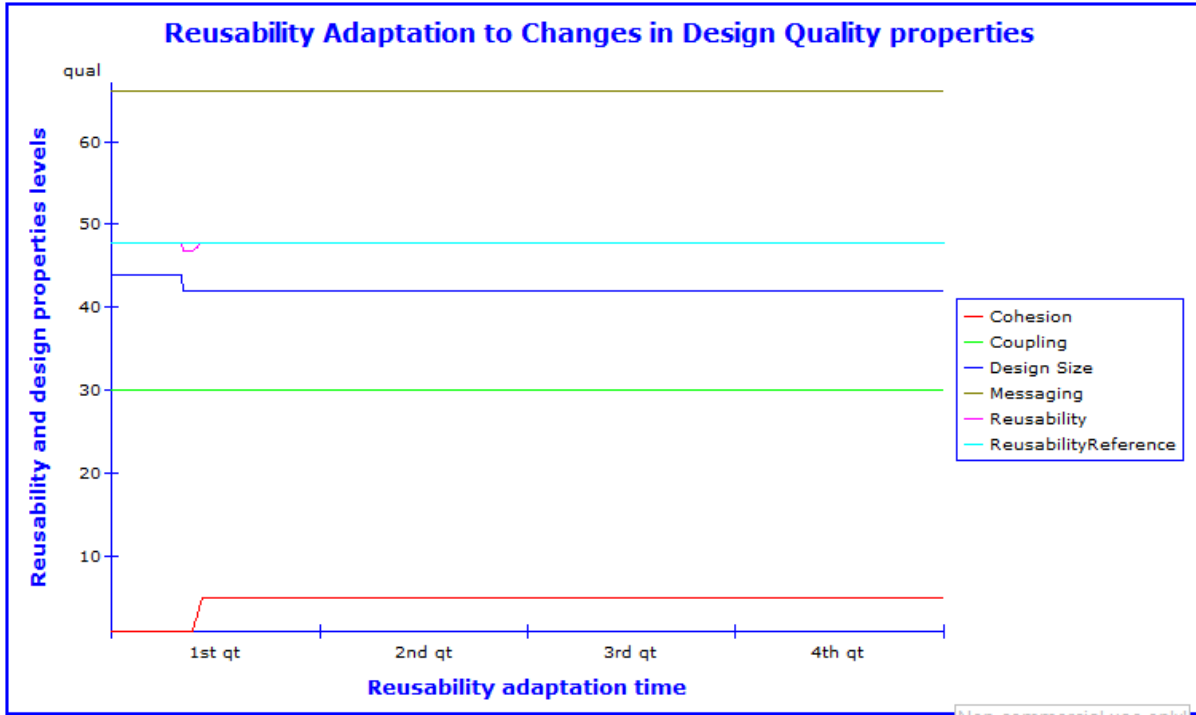


Figure 100: Reusability adaptation results of D6

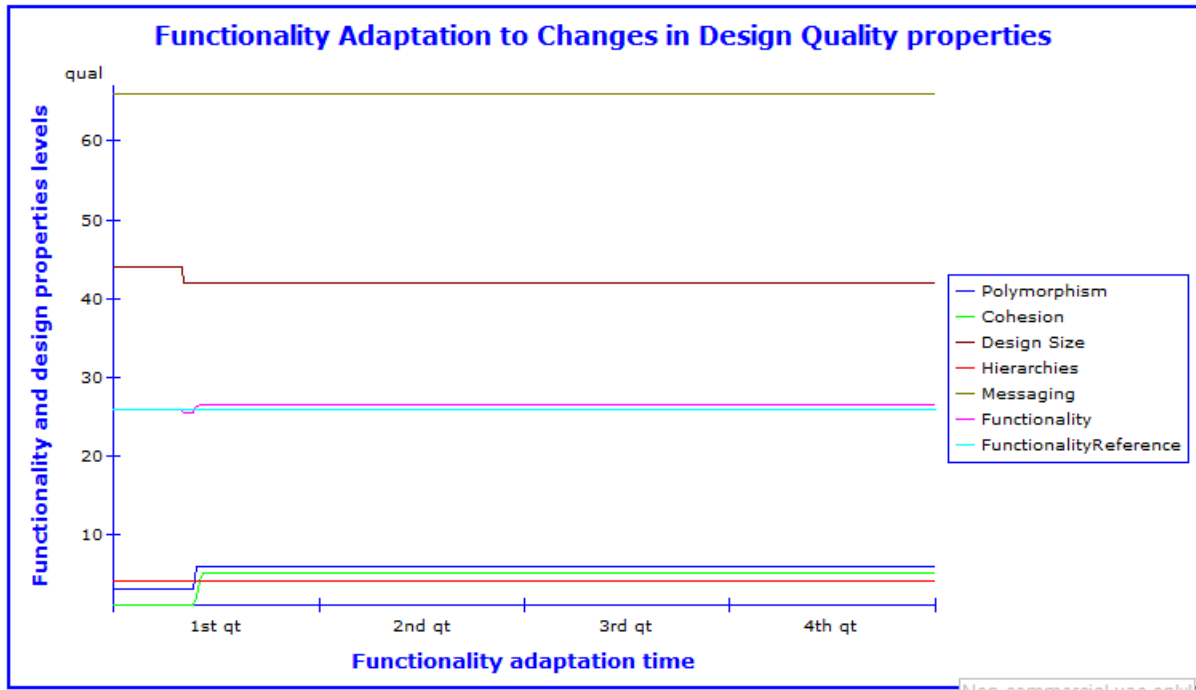


Figure 101: Functionality adaptation results of D6

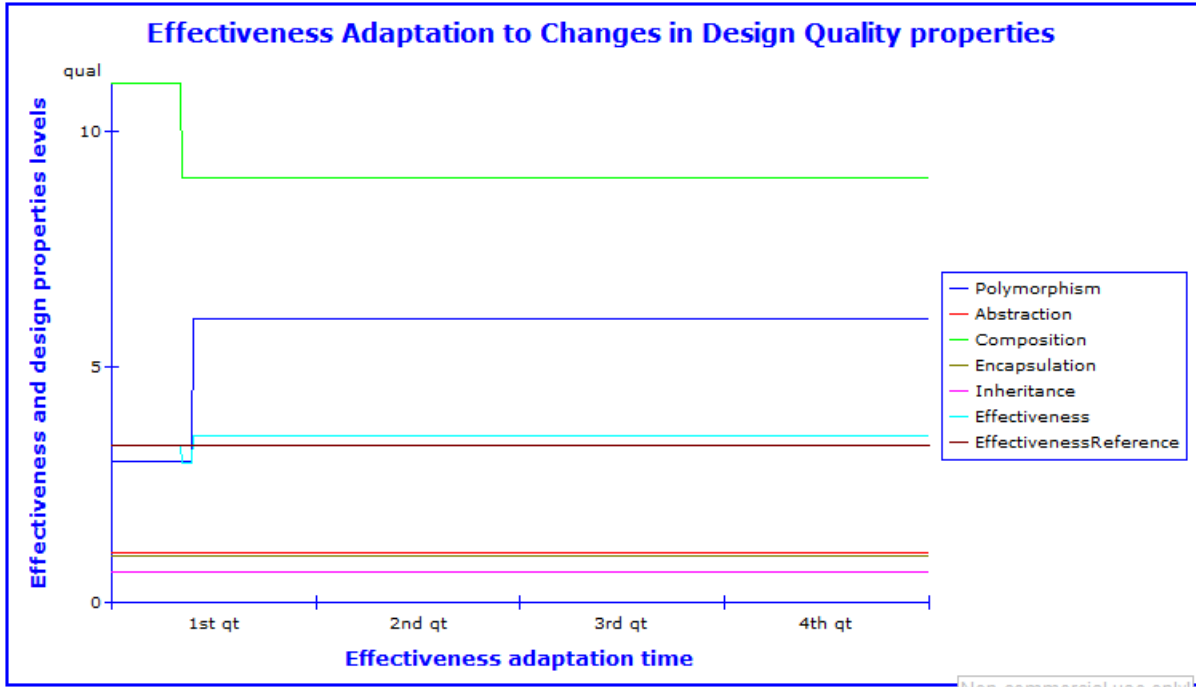


Figure 102: Effectiveness adaptation results of D6

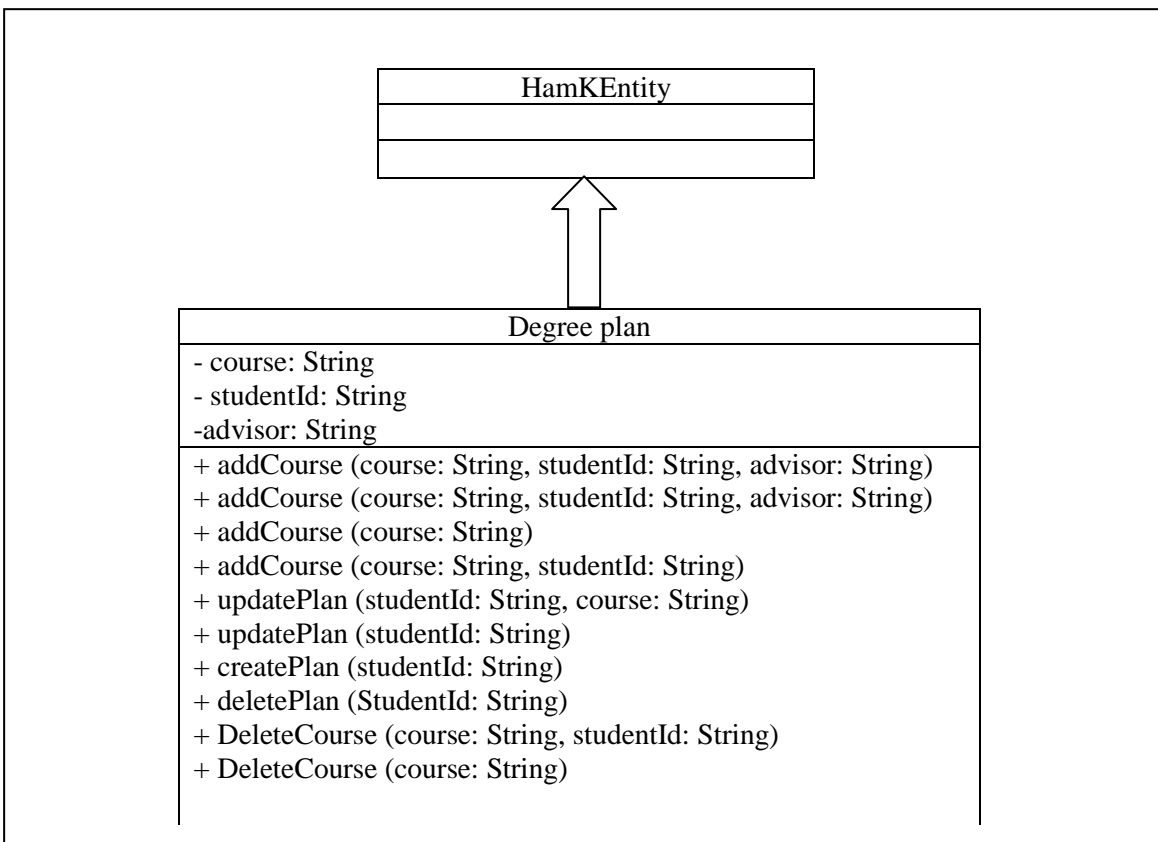


Figure 103: Reusability, functionality, and effectiveness design change and adaptation in D6

## C.6 Design 1 (D7): Music On the Brain (MOB)

### C.6.1 System description and reference quality values

MOB is an online web service allowing users to build their music playlists based on their preferences. The site is also recommending songs to users and keeping track of their liked and disliked music. The main components of D7 design are represented in figures 104-106. The reference quality values of D7 are recorded in tables 17 and 18.

# Design Class Diagram

## Controllers

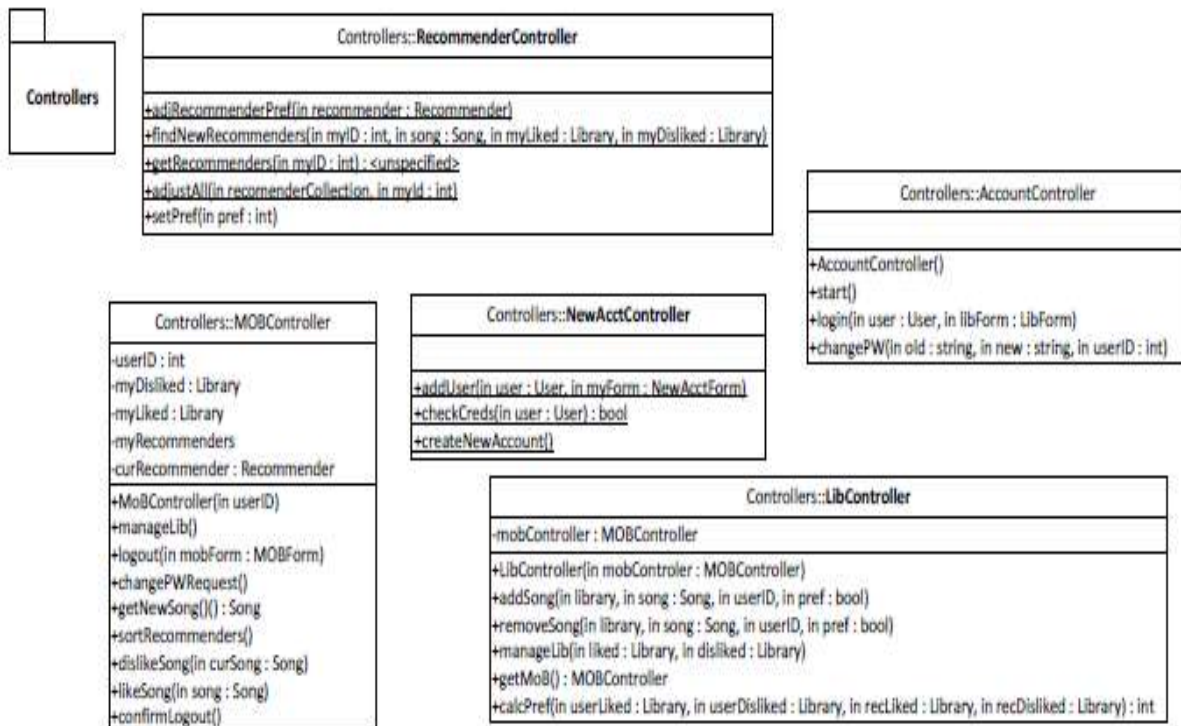


Figure 104: D7 class diagram (1)

# GUI Components

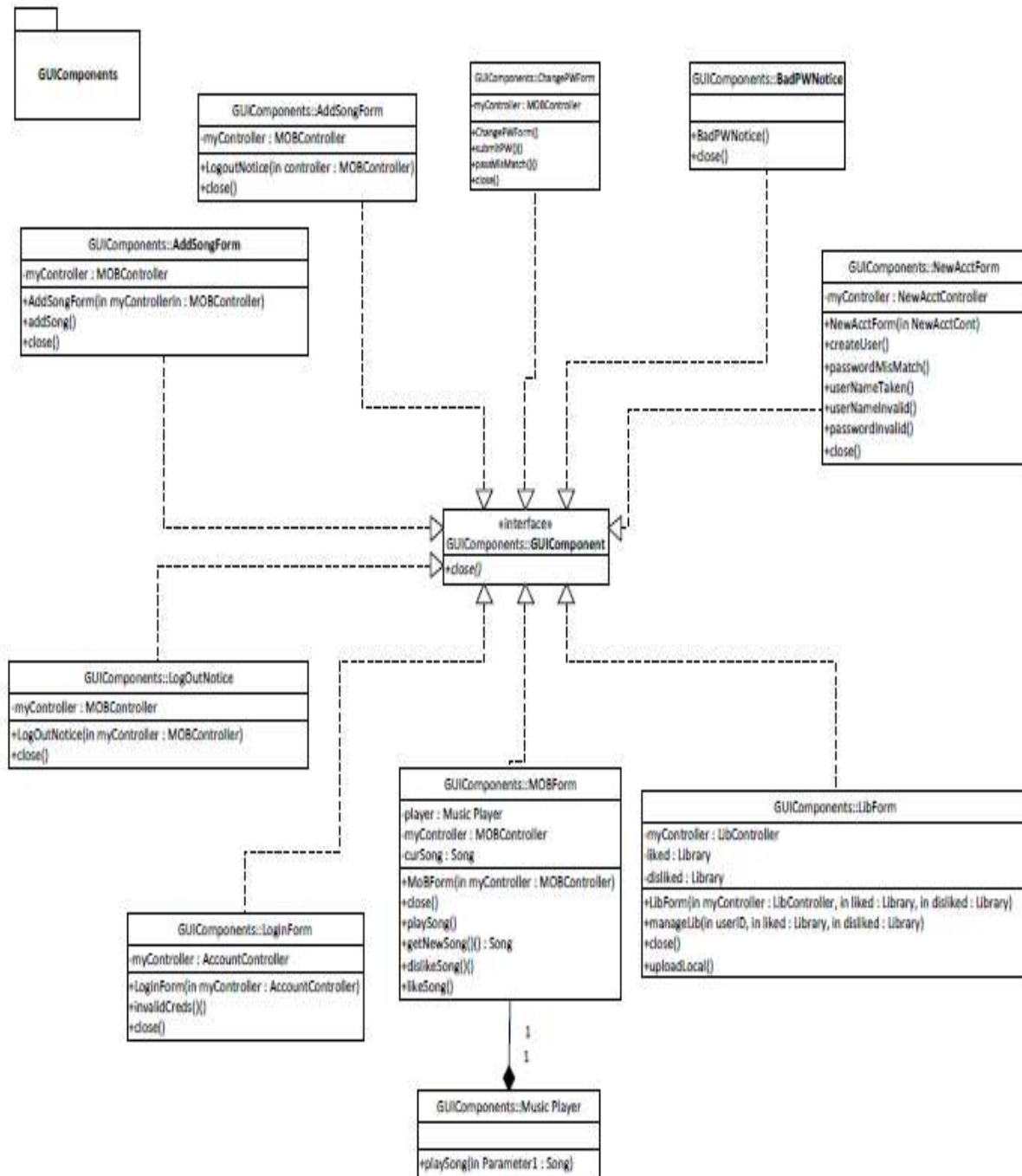


Figure 105: D7 class diagram (2)

# Classes

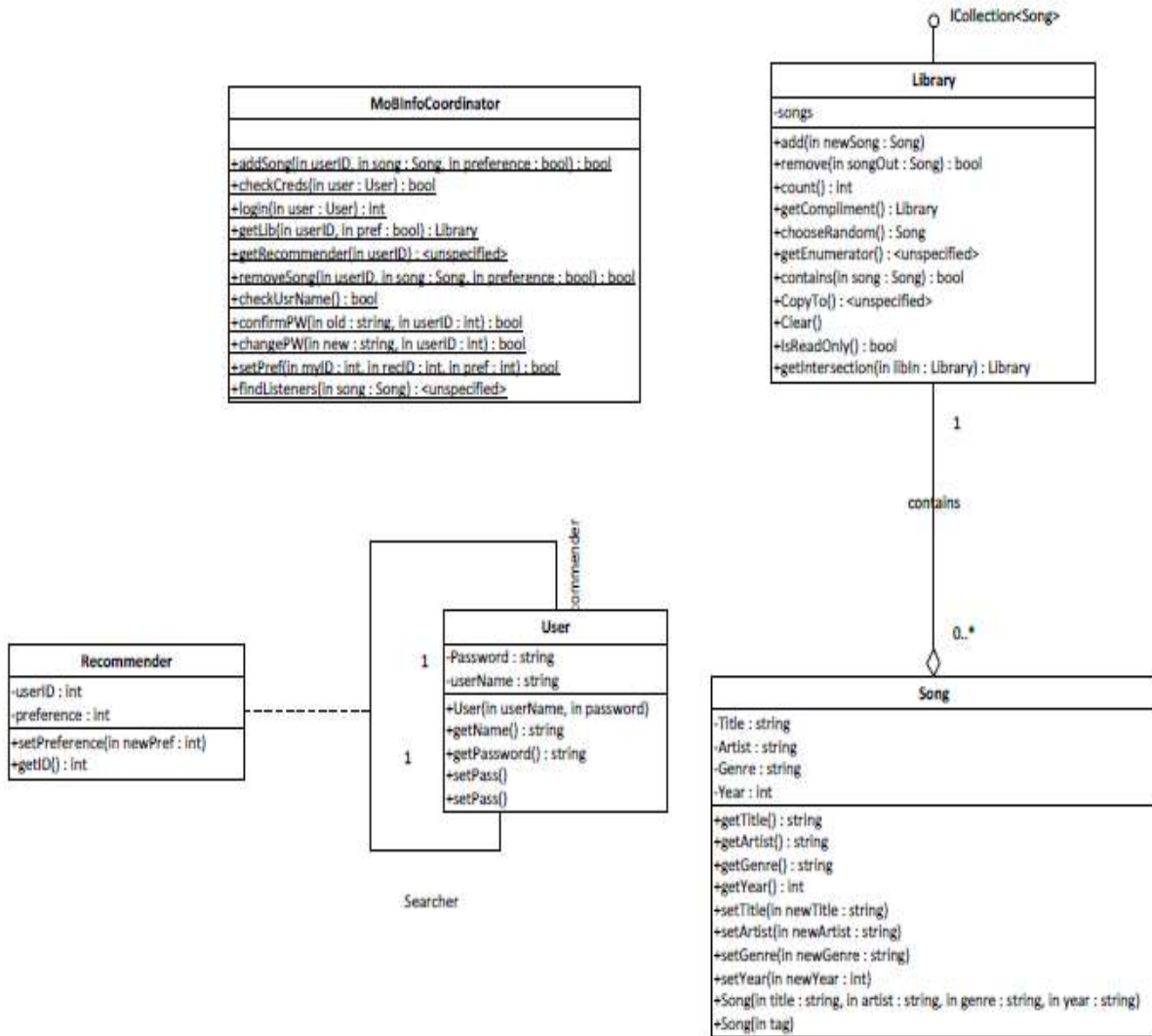


Figure 106: D7 class diagram (3)



## **C.6.2 Design changes**

### **C.6.2.1 Design changes affecting the understandability quality attribute**

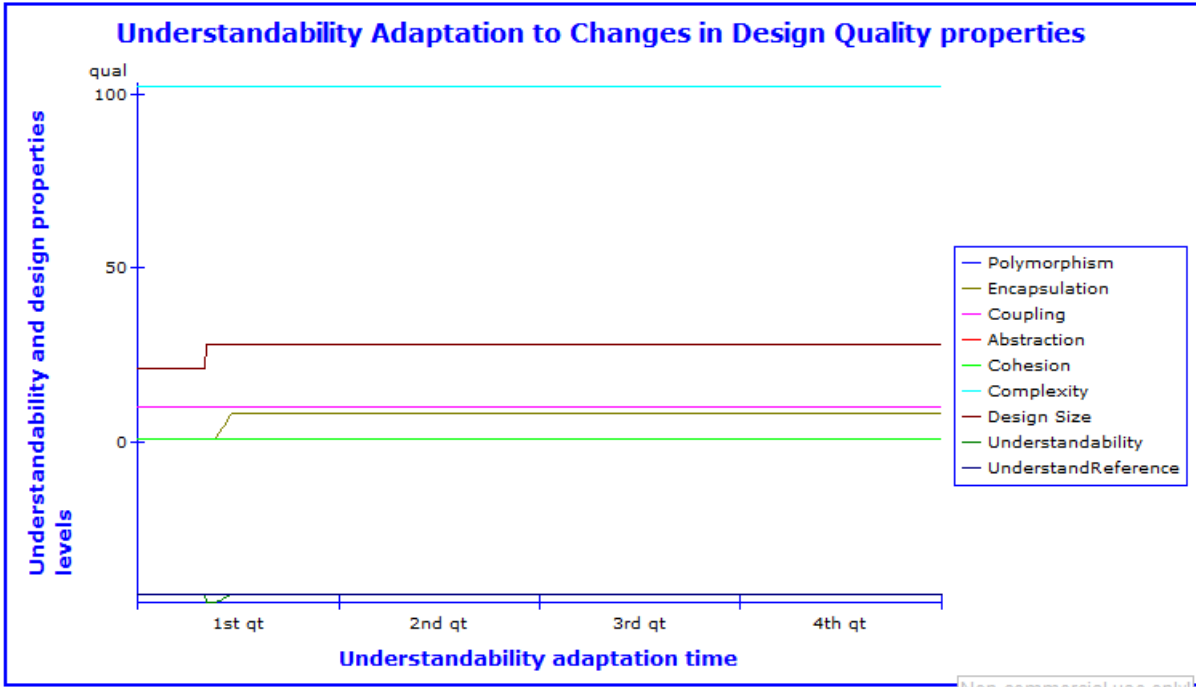
The design of D7 was extended by adding seven classes: “online purchase”, “songsbasket”, “SongBasket”, “MusicVideosLibrary”, “MusicVideo”, “VideoLibrary”, and “Video” (figures 109-111). As a result, the design size of D7 increased and its understandability decreased below its reference value. The impact of this design change and the applied encapsulation adaptation equation is illustrated in the following simulation and real results.

#### **1) Simulated results**

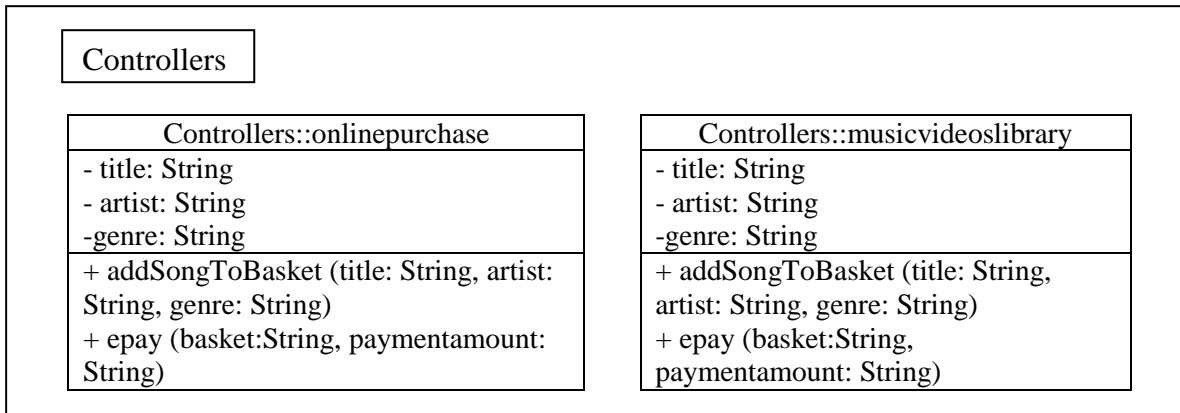
From figure 107, encapsulation should be increased by seven to bring back understandability to its reference value.

#### **2) Real results**

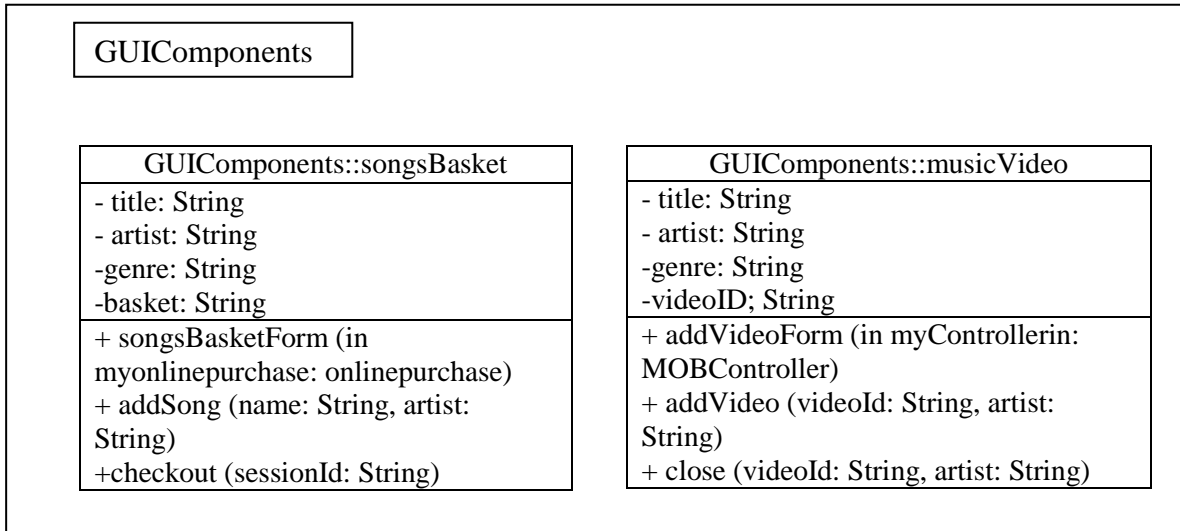
Figures 108-110 show that the simulated encapsulation adaptation can be easily applied on the real design of D7, which makes the resulting adapted understandability equal its simulated value.



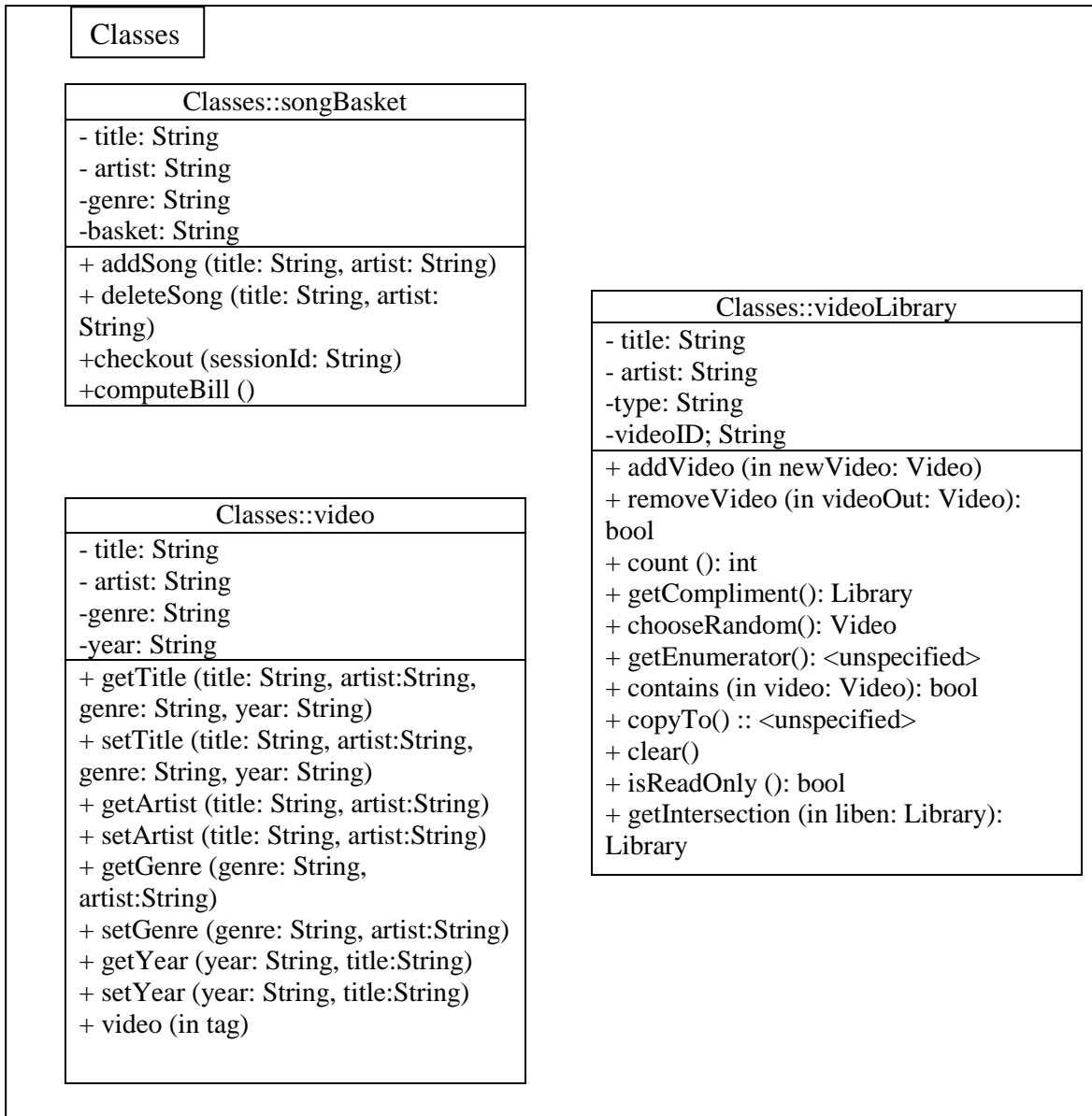
**Figure 107: Understandability adaptation results of D7**



**Figure 108: Understandability design change and adaptation in D7 (1)**



**Figure 109: Understandability design change and adaptation in D7 (2)**



**Figure 110: Understandability design change and adaptation in D7 (3)**

### C.6.2.2 Design changes affecting the extendibility and the flexibility quality attributes

After adding seven classes to D7, four new relationships were identified. The first linkage is an inheritance relationship between the “GUIComponent” and the “SongsBasket” classes. Another inheritance relationship was depicted between the “GUIComponent” and the “MusicVideo” classes. The remaining two relationships are aggregation relationships such as the

relationship between the “Song” and the “SongBasket” classes (figures 113-115). Those design changes led to an increase in coupling and extendibility and flexibility quality attributes as it is illustrated in the simulated and the real results.

### 1) Simulated results

The obtained results from the simulation in figures 111 and 112 show that polymorphism should be increased by four to adapt the values of extendibility and flexibility.

### 2) Real results

Four polymorphic methods were added to D7 such as “deleteSong (title: String)” as represented in figures 113-115. As a result, flexibility and reusability successfully increased to at least their reference values. Moreover, the real values of the quality attributes almost equal their simulated values.

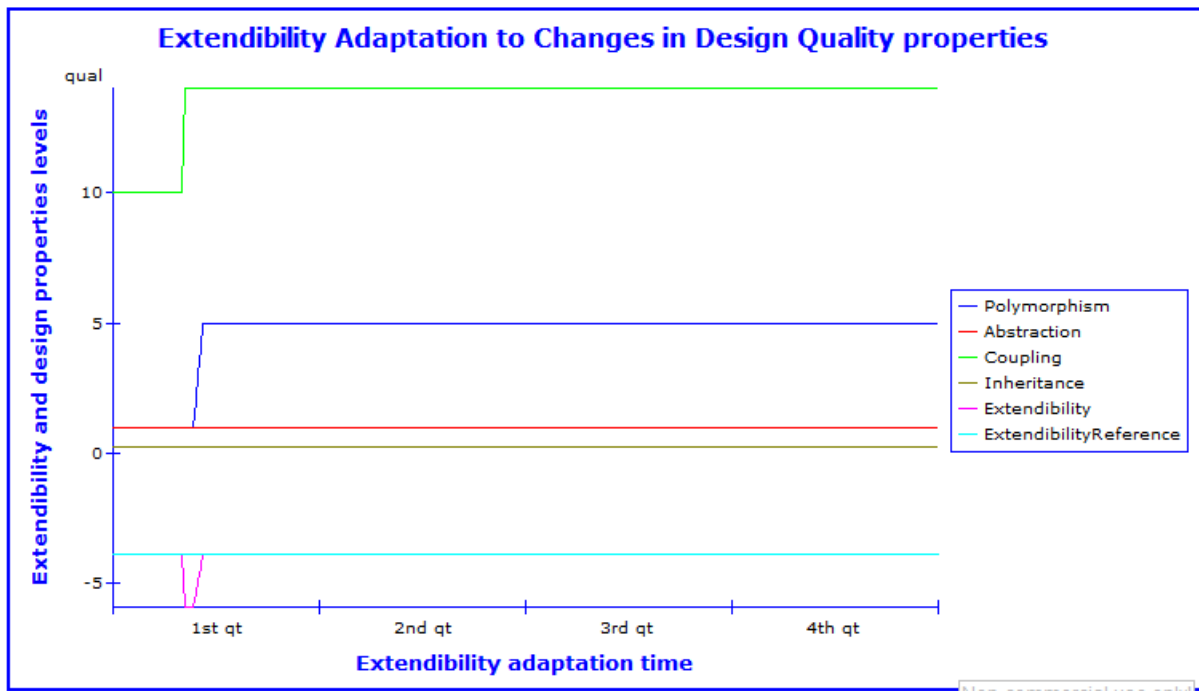


Figure 111: Extendibility adaptation results of D7

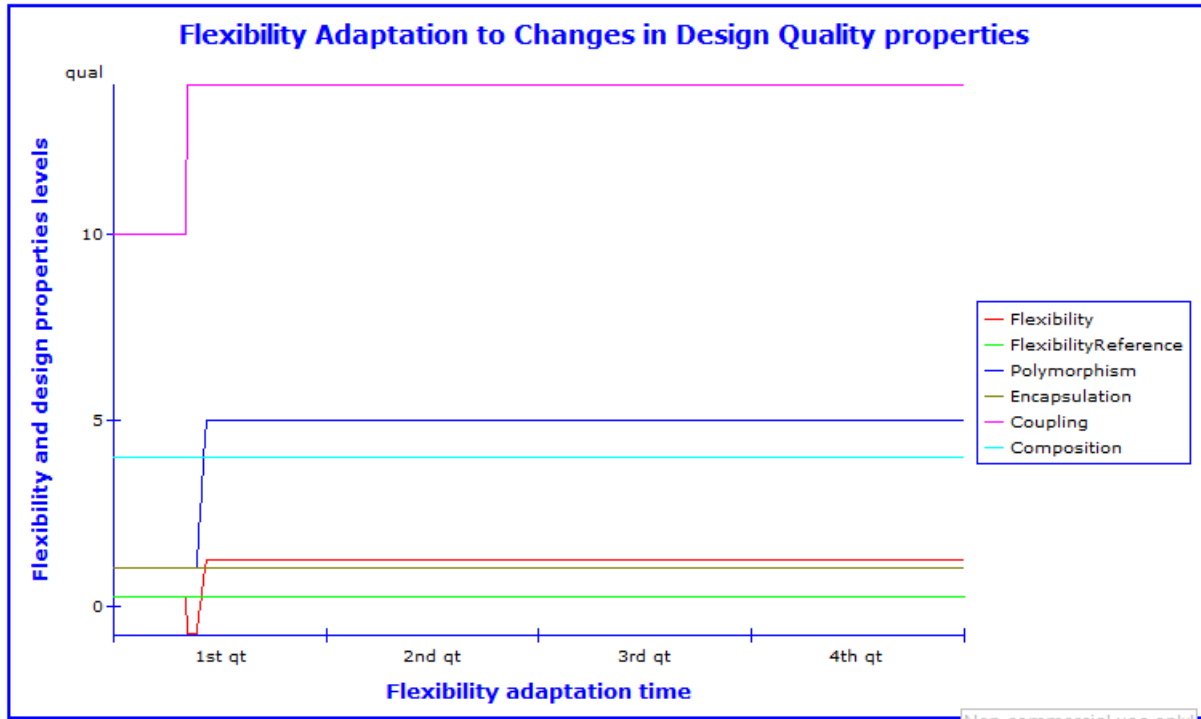


Figure 112: Flexibility adaptation results of D7

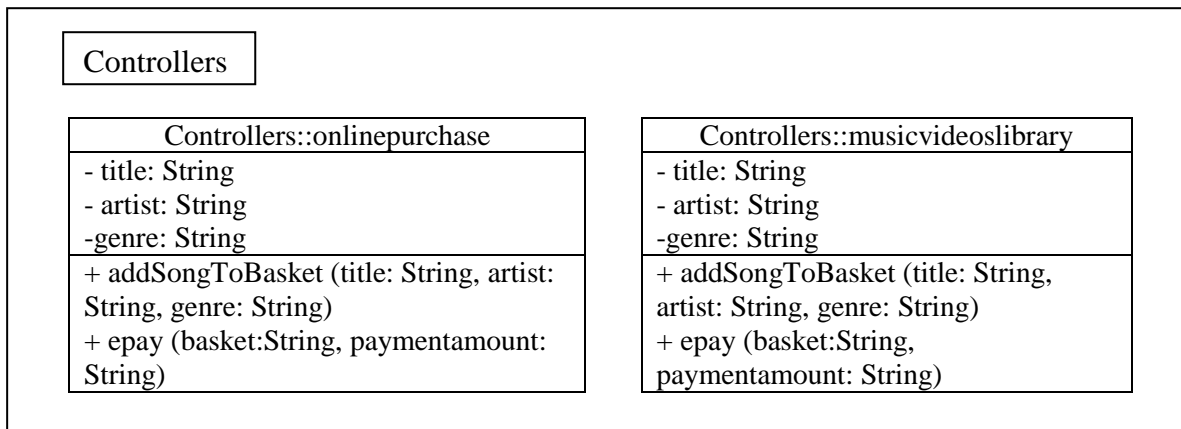
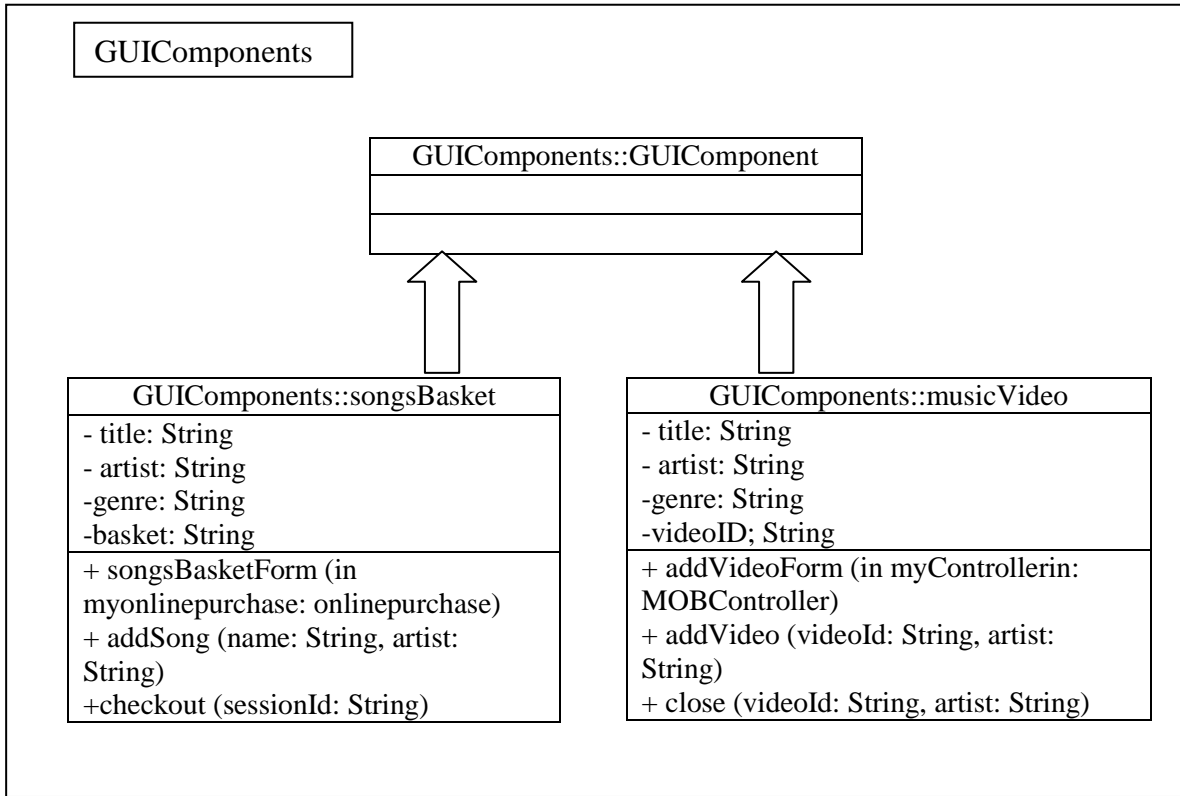
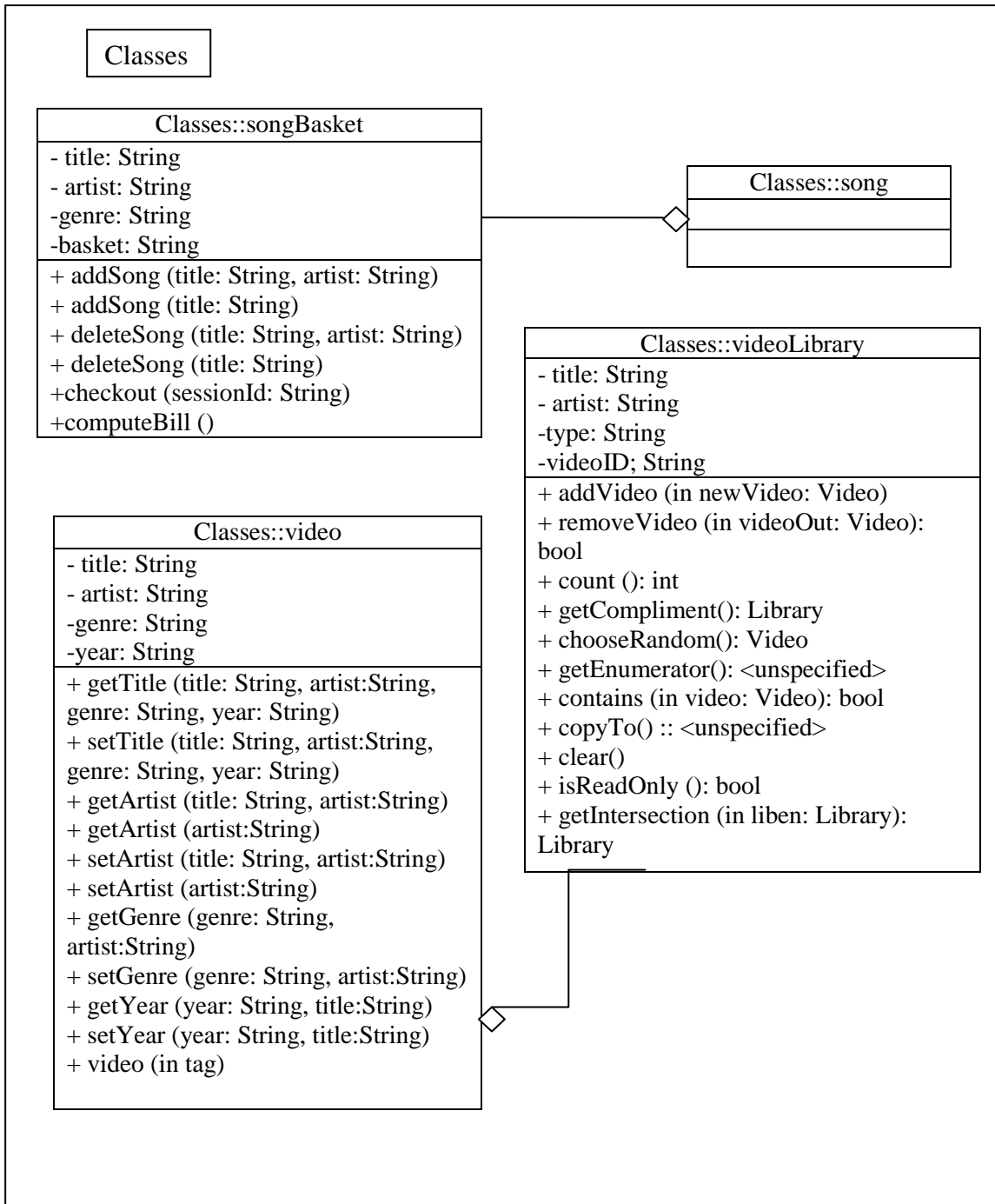


Figure 113: Extensibility and flexibility design change and adaptation in D7 (1)



**Figure 114: Extensibility and flexibility design change and adaptation in D7 (2)**





**Figure 115: Extensibility and flexibility design change and adaptation in D7 (3)**

### **C.6.2.3 Design changes affecting the reusability, the functionality and the effectiveness quality attributes**

The last design change in D7 led both to a decrease in design size and composition. The classes “Song” and “Video” were deleted as well as their composition relationships. On the one hand, the decrease in design size led to a decrease in the functionality and the reusability quality attributes. On the other hand, the decrease in composition led to a decrease in the effectiveness quality attribute. The simulated and the real results illustrate the impact of those changes and the applied adaptation equations of cohesion and polymorphism.

#### **1) Simulated results**

To counterbalance the values of reusability and functionality, cohesion should be maximized (i.e.  $CAM = 1$ ) at least in two classes (figures 116 and 117). The value of effectiveness can be adapted when polymorphism is increased by six (figure 118).

#### **2) Real results**

Figures 119-121 illustrate the application of the simulated changes and their corresponding adaptations. The cohesion adaptation mechanism was applied in the “songBasket” and the “videoLibrary” classes. In addition, six polymorphic methods were added to D7 such as “addVideo (title: String, artist: String)” to adapt the value of effectiveness. The resulting real values of the quality attributes are nearly similar to their simulated ones.

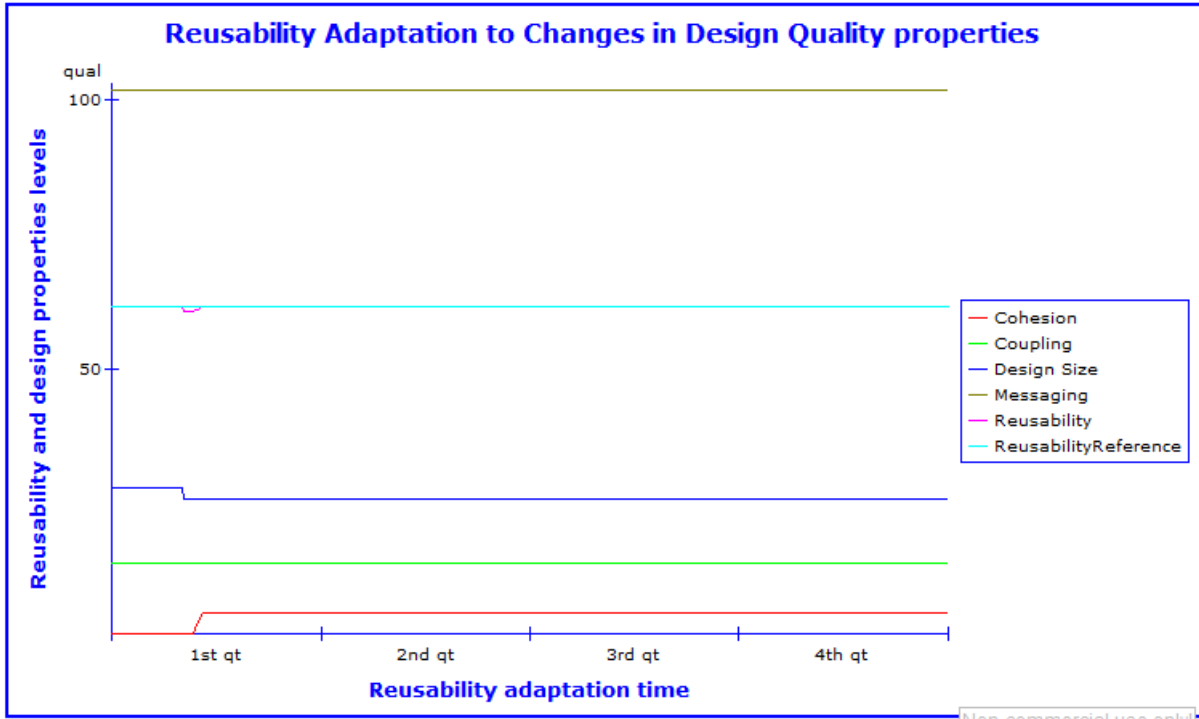


Figure 116: Reusability adaptation results of D7

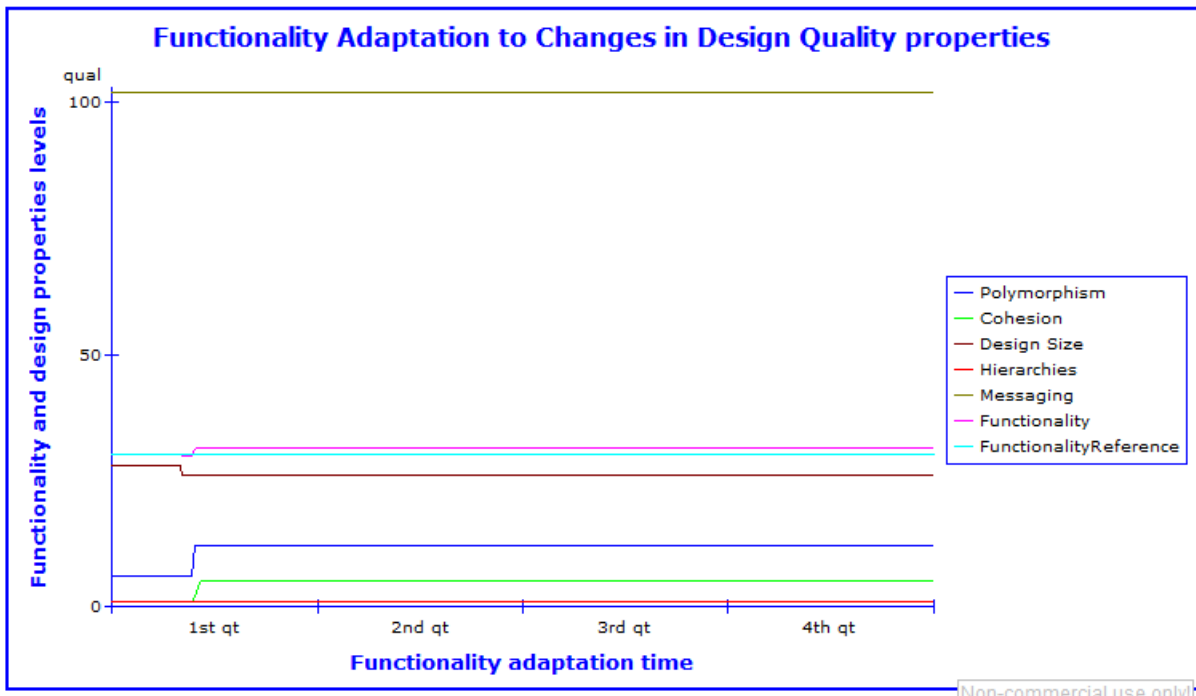


Figure 117: Functionality adaptation results of D7

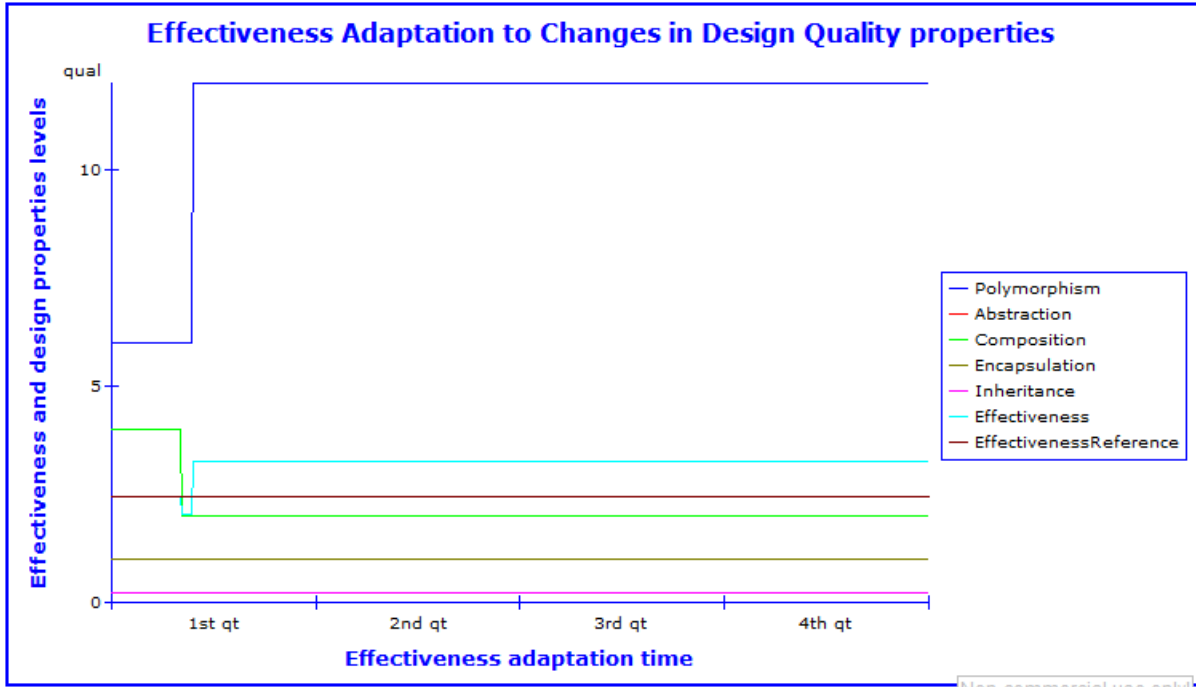


Figure 118: Effectiveness adaptation results of D7

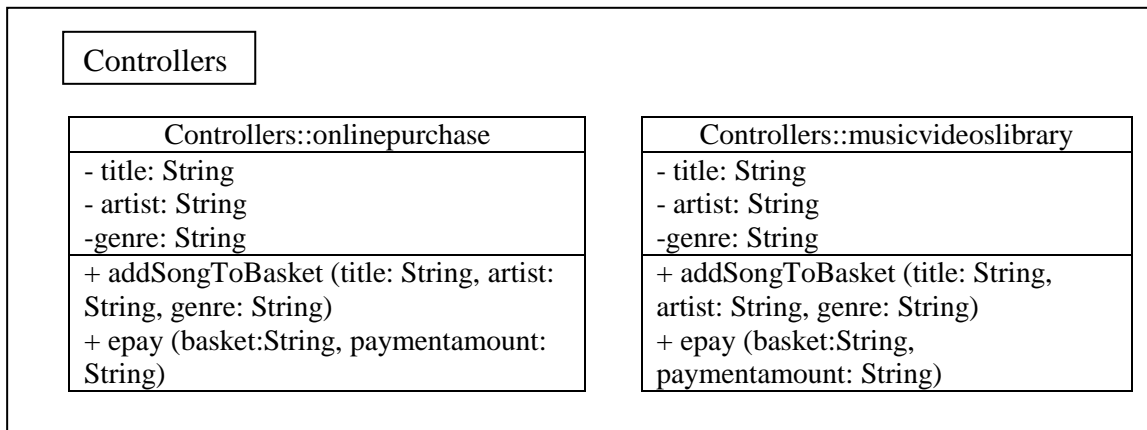
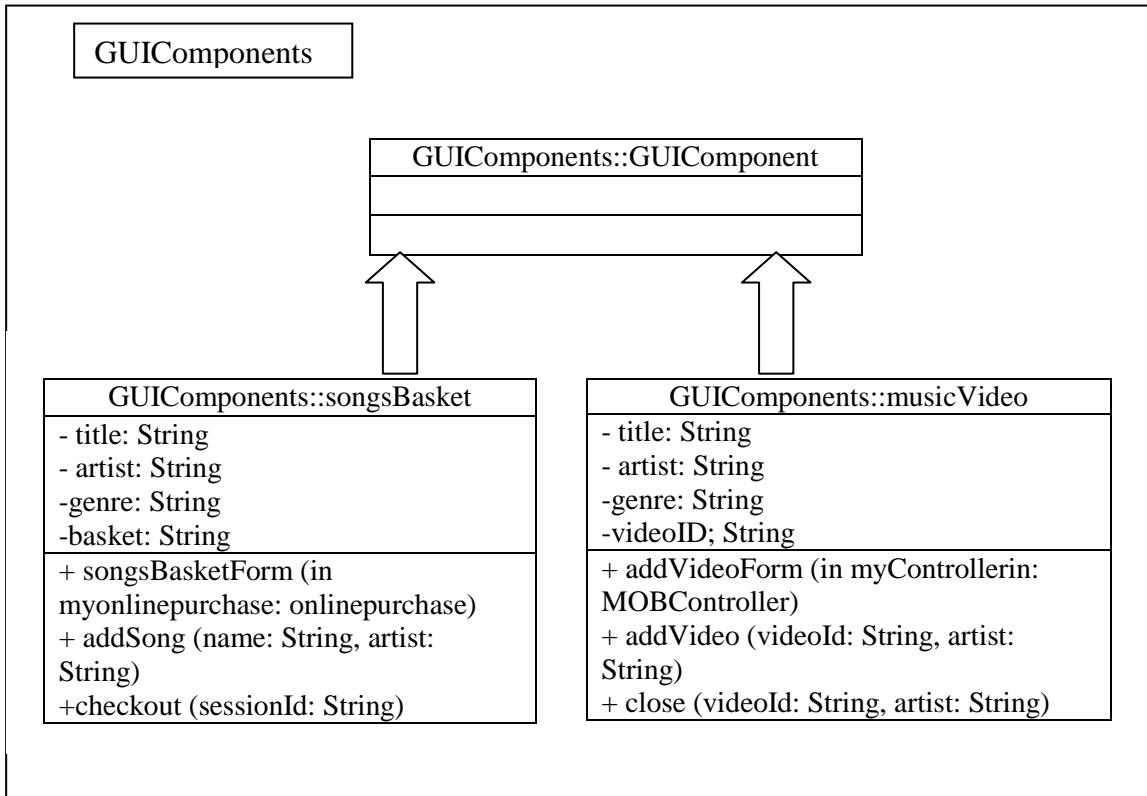
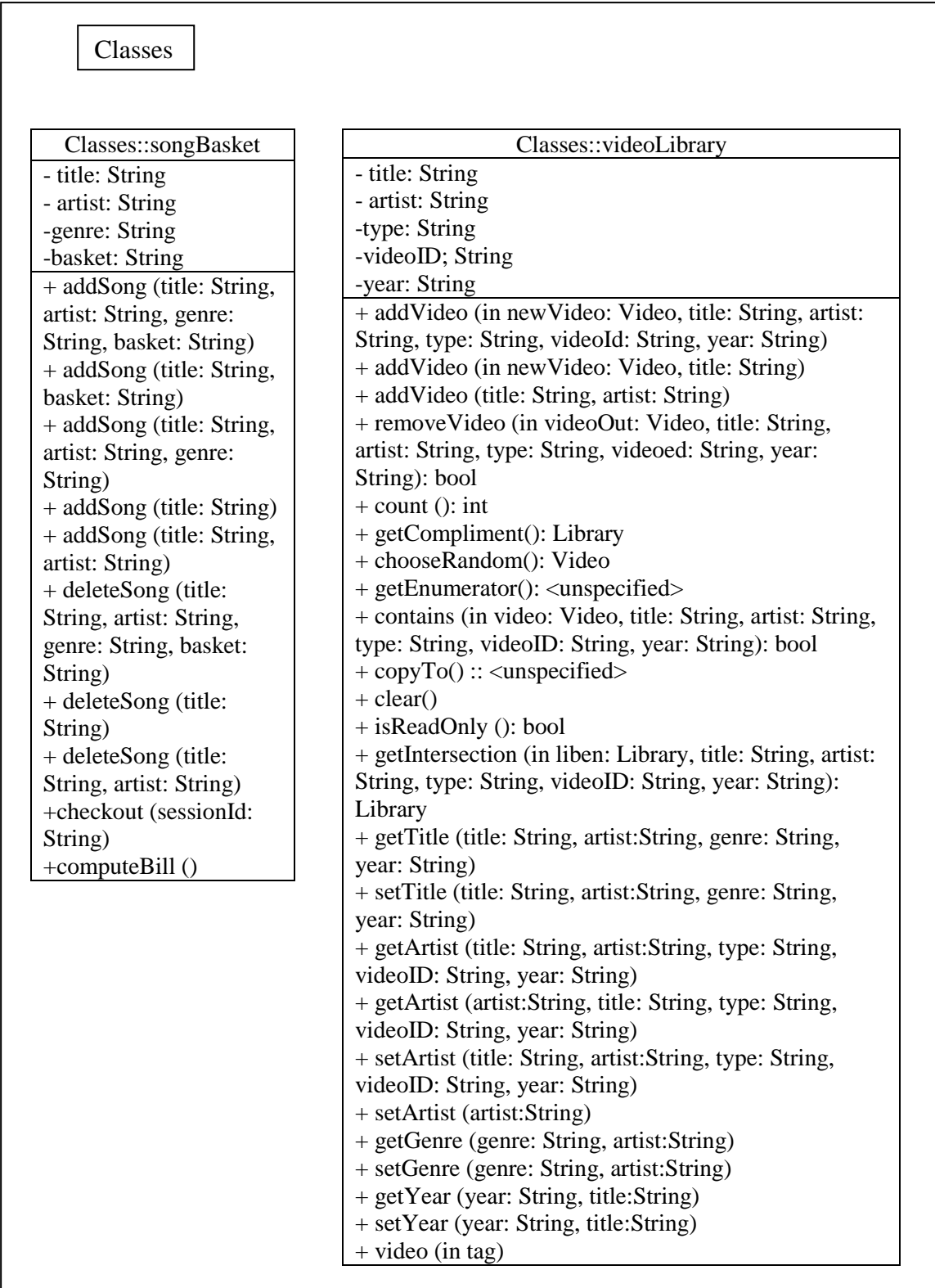


Figure 119: Functionality, reusability, and effectiveness design changes and adaptations in D7 (1)



**Figure 120: Functionality, reusability, and effectiveness design changes and adaptations in D7 (2)**



**Figure 121: Functionality, reusability, and effectiveness design changes and adaptations in D7 (3)**

## C.7 Design 8 (D8): Skye Net Home Security

### C.7.1 System description and reference quality values

Skye Net is a computerized home security system that is owned by the homeowner and maintained remotely by the system’s producer. The protection offered by the system includes many options such as door/window alarms, smoke detector, and carbon monoxide detector. In addition, the system has different security modes such as the “away”, “vacation”, and “in home” modes (figure 122). The reference quality values of D8 are recorded in tables 17 and 18.

### Design Class Diagram

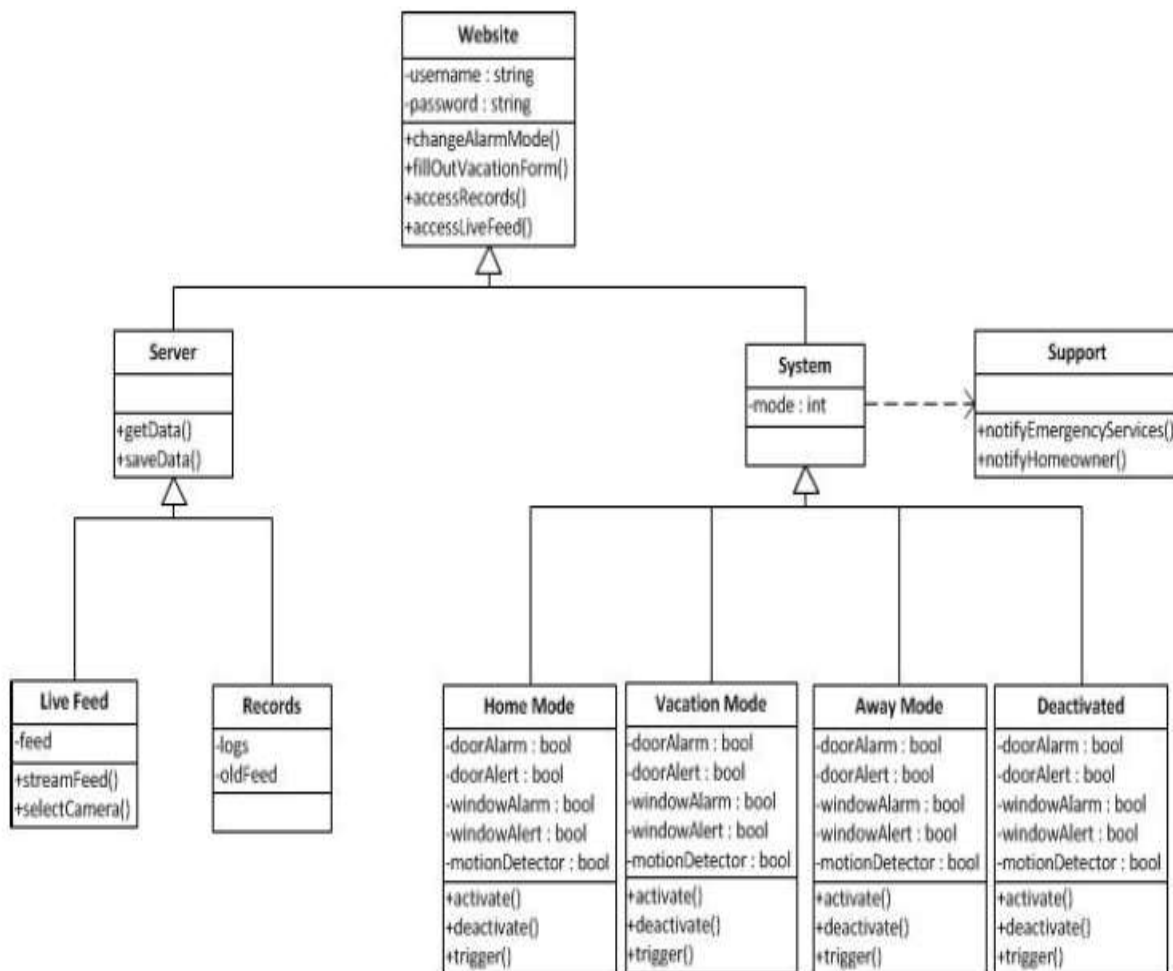


Figure 122: D8 class diagram

## **C.7.2 Design changes**

### **C.7.2.1 Design changes affecting the understandability quality attribute**

To upgrade the capabilities of the system, four new classes were added to D8: “System updates”, “System feedback”, “System maintenance”, and “System expansion request” (figure 124). Consequently, the design size of D8 increased and its understandability decreased. The simulated and the real results show the effectiveness of the encapsulation equation in adapting understandability.

#### **1) Simulated results**

To counterbalance the decrease in understandability, encapsulation should be maximized in the four newly added classes (figure 123).

#### **2) Real results**

The simulated changes and their corresponding adaptations were applied in the real design of D8 as illustrated in figure 124. The obtained understandability from the real design of D8 after adaptation equals its simulated value.



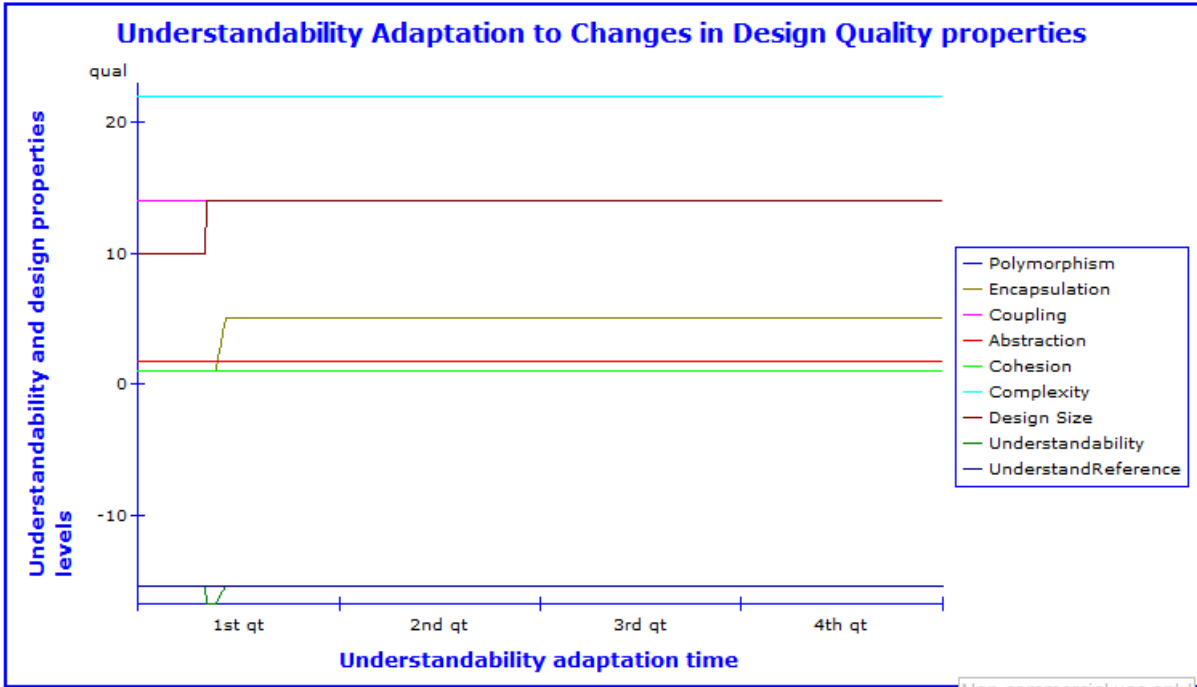
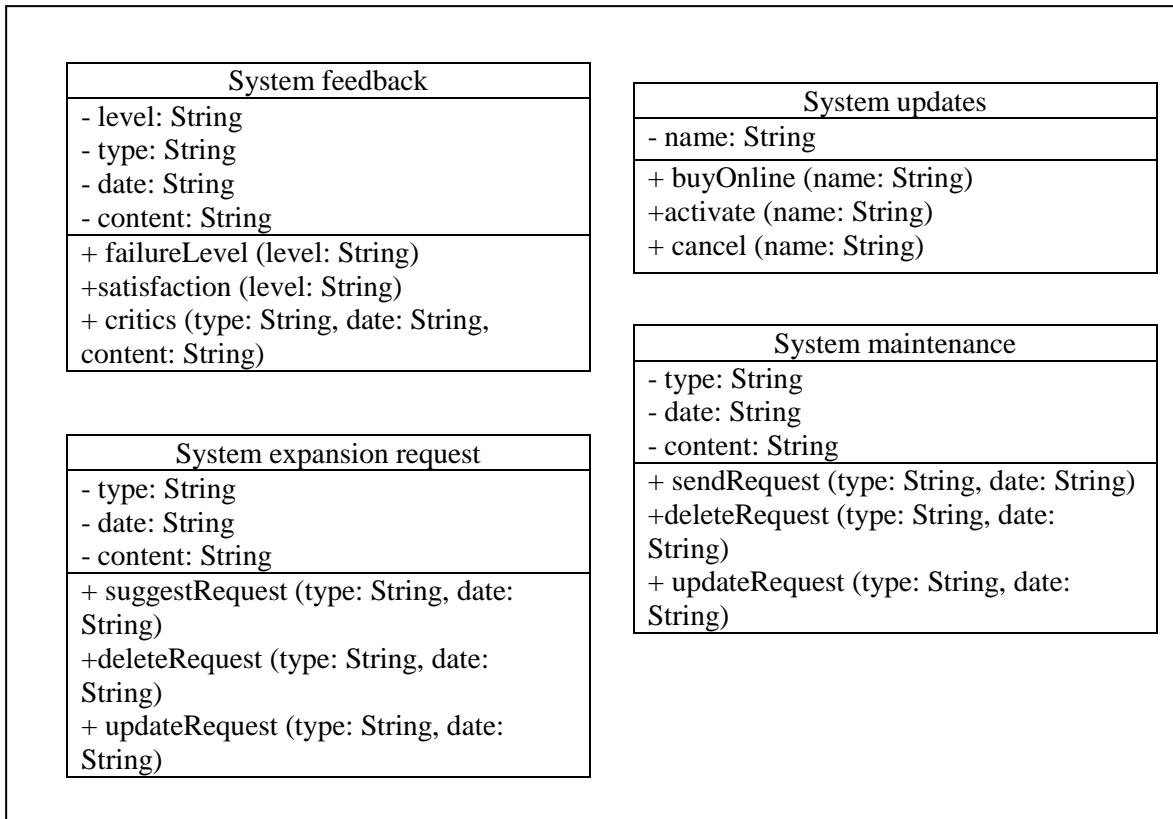


Figure 123: Understandability adaptation results of D8



**Figure 124: Understandability design changes and adaptations of D8**

### C.7.2.2 Design changes affecting the extendibility and the flexibility quality attributes

The newly added classes in the first design change were linked to the class “System” through aggregation relationships, which increased the coupling property of D8 (figure 127). Consequently, the flexibility and the extendibility quality attributes dropped below their reference values as illustrated in the following simulated and real results.

#### 1) Simulated results

The decrease in extendibility and flexibility can be adapted by increasing polymorphism to three polymorphic methods (figures 125 and 126).

## 2) Real results

As suggested in the simulated results, three polymorphic methods such as “suggestRequest (type: String)” were added to D8 to adapt the values of extendibility and flexibility (figure 127). The real computed values of the quality attributes after adaptation are similar to their simulated values.

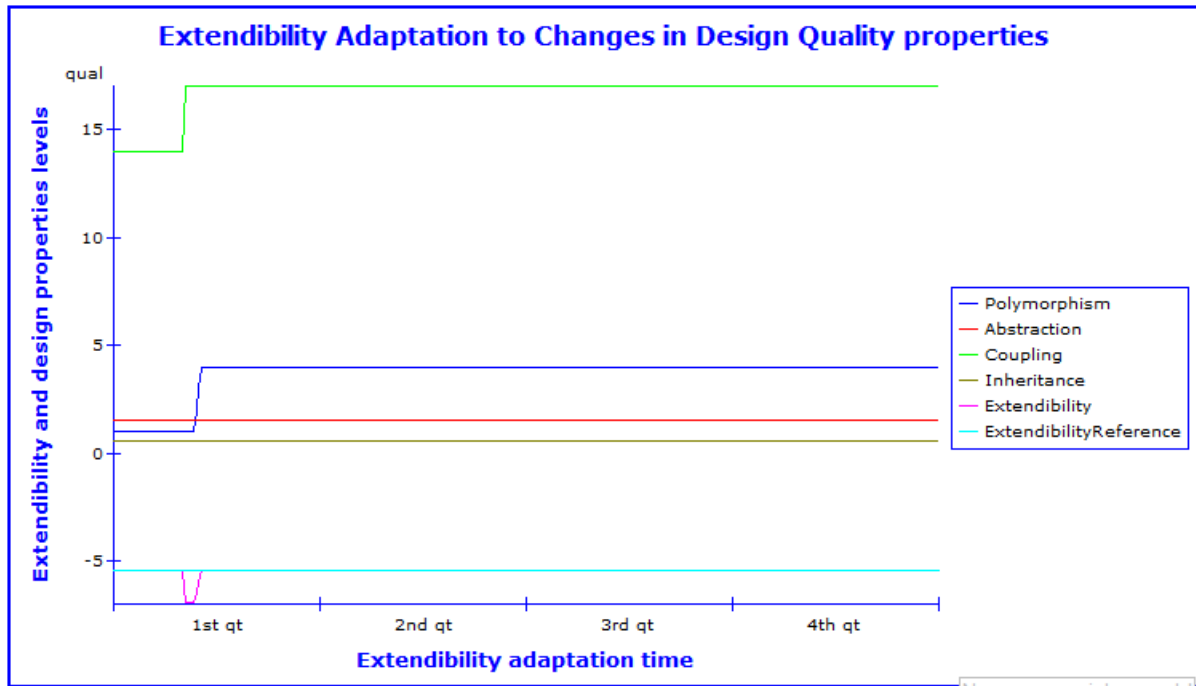


Figure 125: Extendibility adaptation results of D8

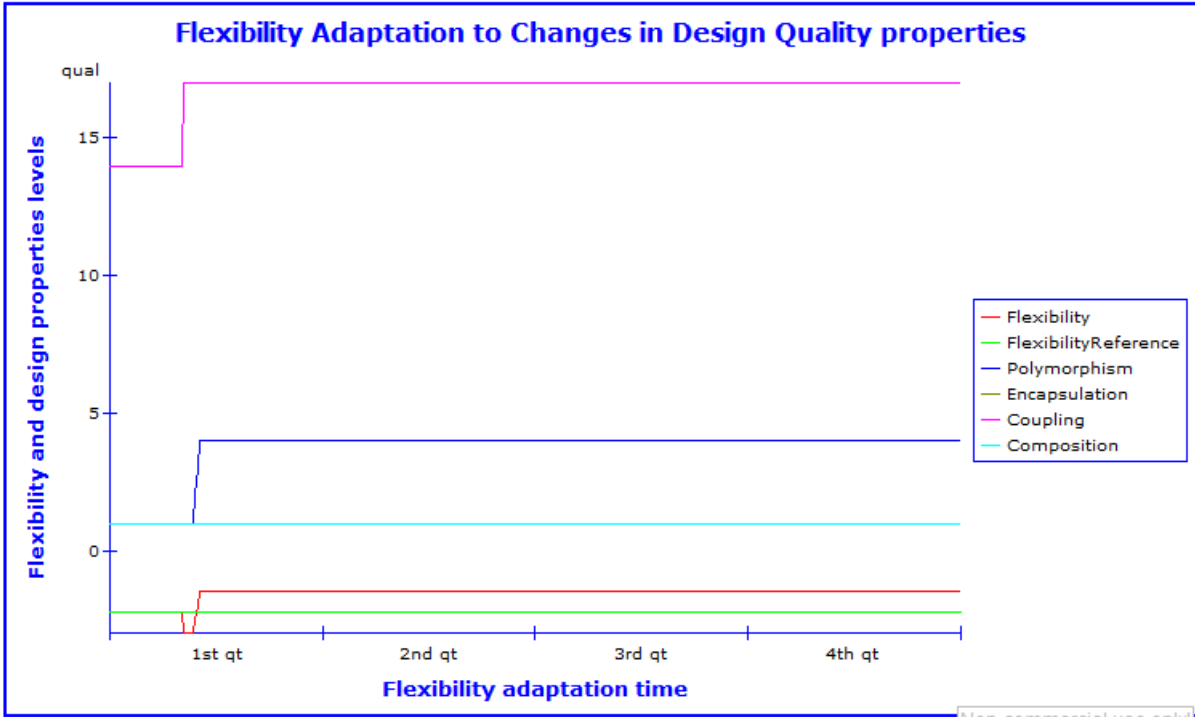
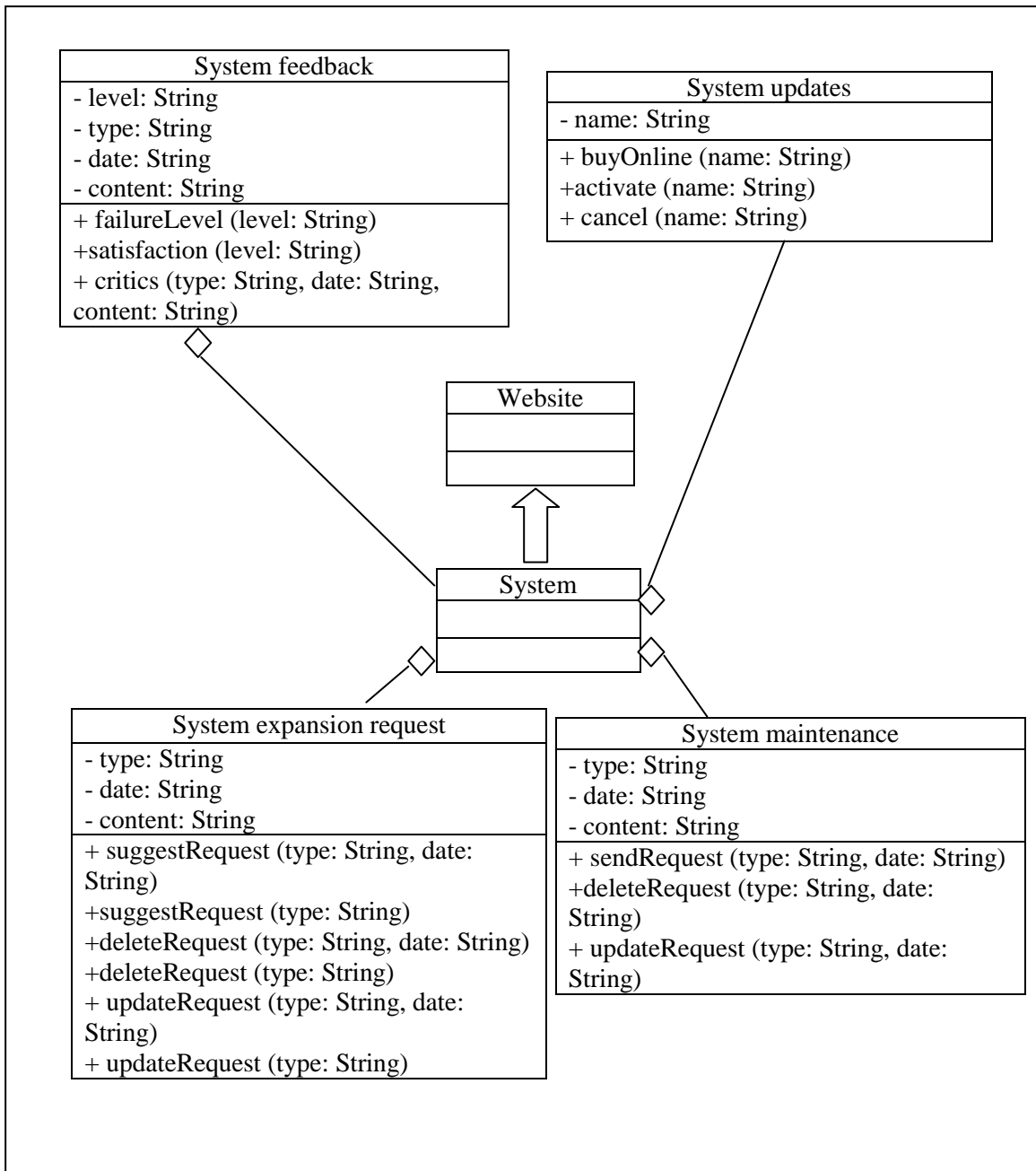


Figure 126: Extendibility adaptation results of D8



**Figure 127: Extensibility and flexibility design changes and adaptations in D8**

### C.7.2.3 Design changes affecting the functionality and the reusability quality attributes

After merging the “Vacation mode” functionalities in the “Away Mode” class, the design size of D8 decreased as well as its reusability and functionality. The simulated and the real results show the effectiveness of cohesion in adapting the values of reusability and functionality.

#### 1) Simulated results

The reusability and functionality of D8 can be adapted when cohesion is maximized in two classes (figures 128 and 129).

#### 2) Real results

In figure 130, cohesion was maximized in the “System feedback” and the “System maintenance” classes to adapt the values of the reusability and the functionality attributes. The real values of the quality attributes after adaptation equal their simulated ones.

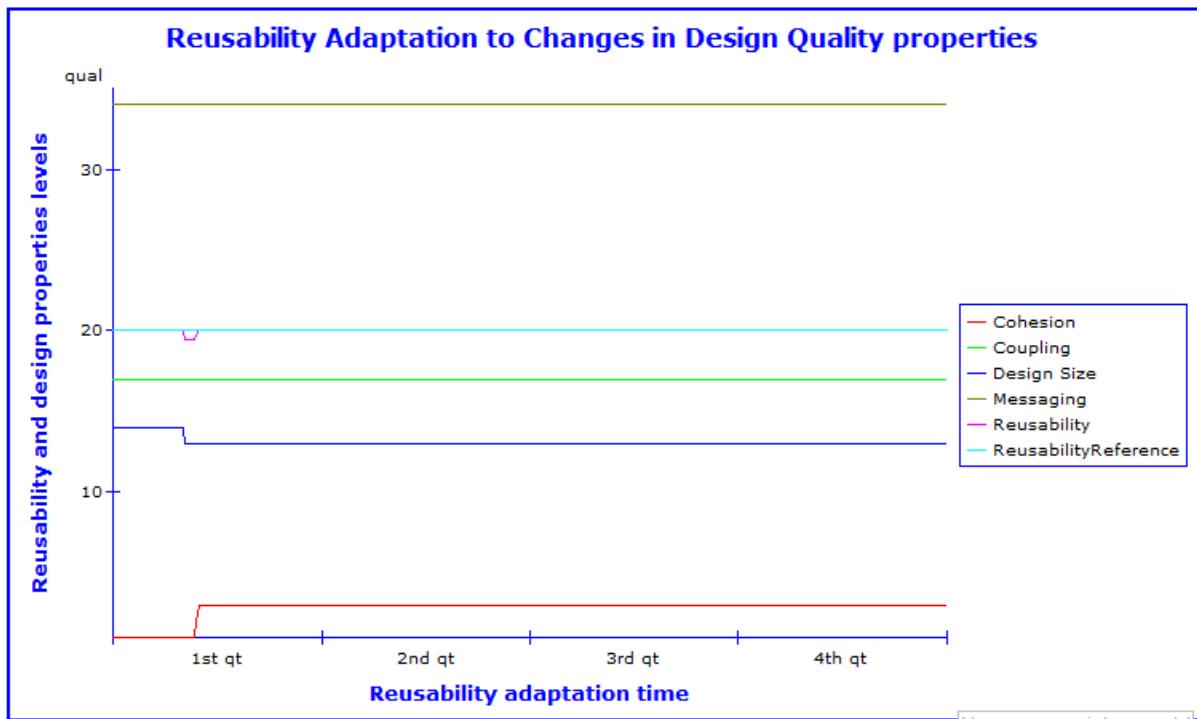


Figure 128: Reusability adaptation results of D8

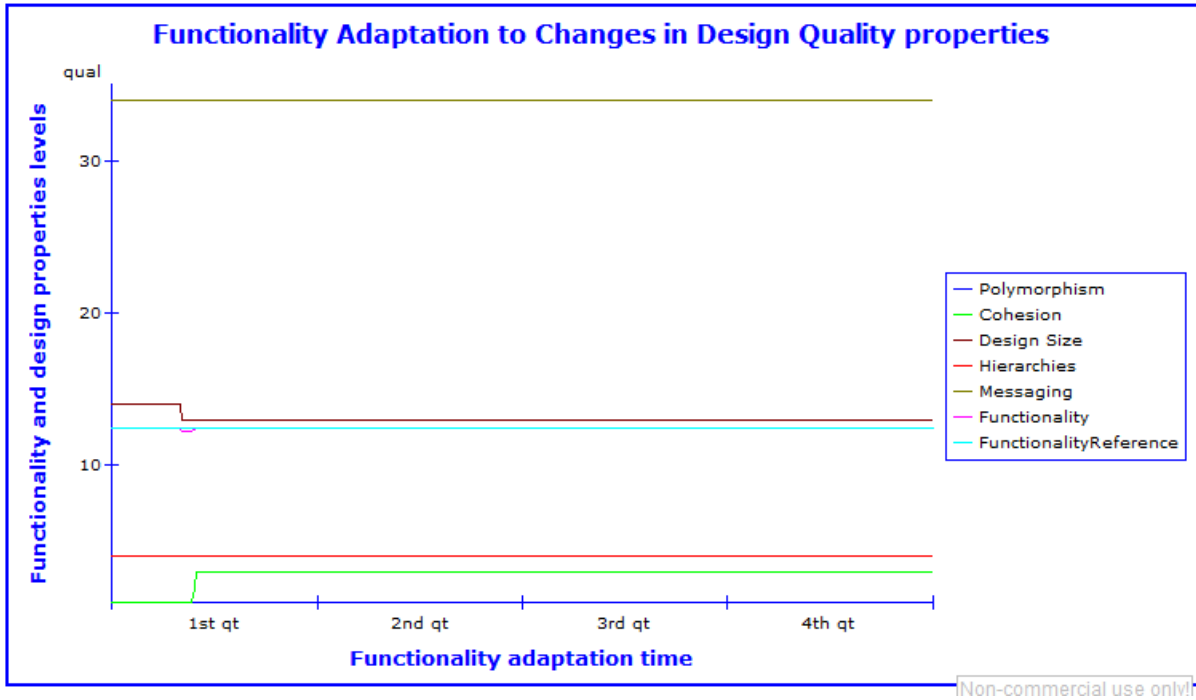
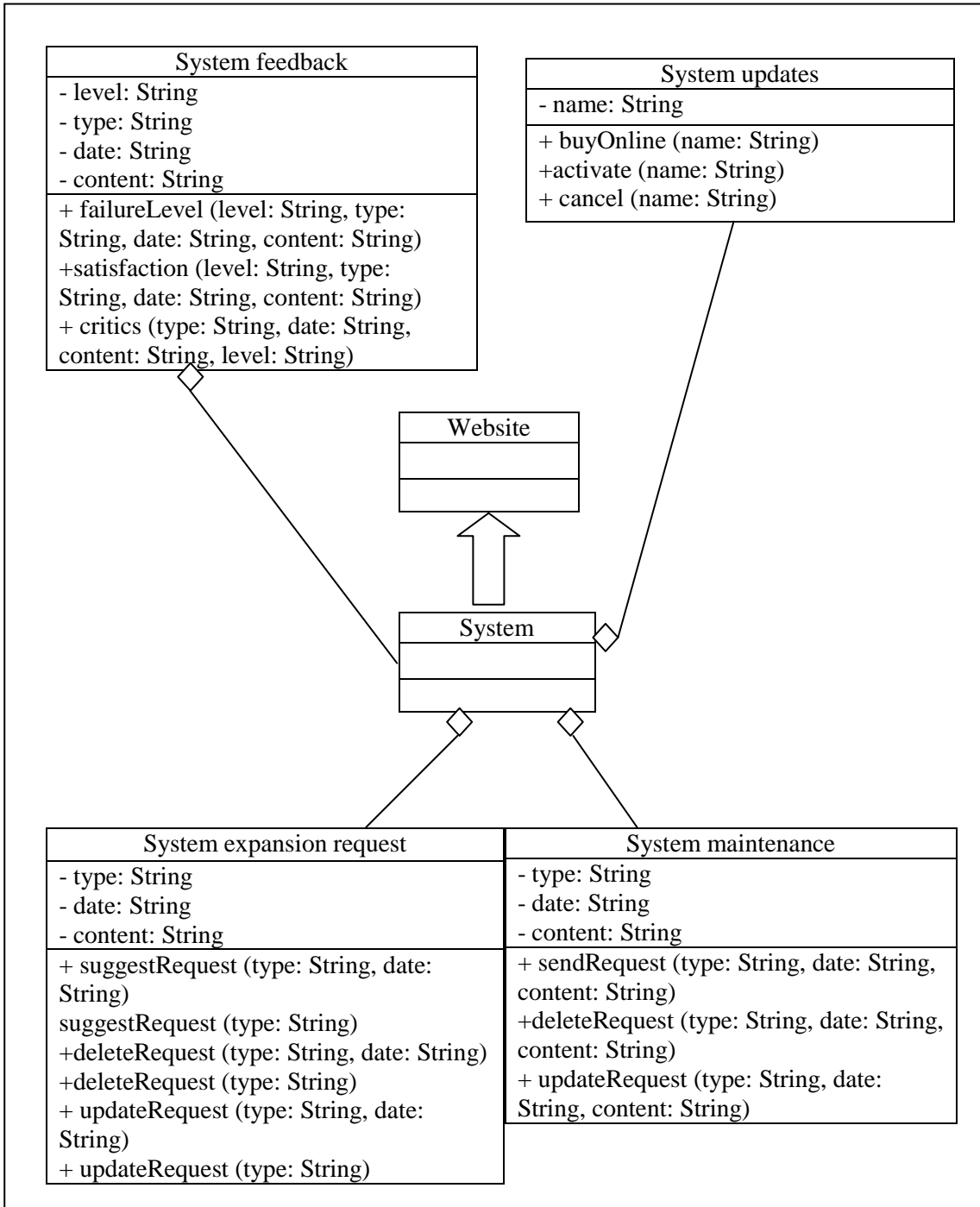


Figure 129: Functionality adaptation results of D8



**Figure 130: Reusability and functionality design changes and adaptations in D8**



### C.7.2.4 Design changes affecting the effectiveness quality attribute

The last change in D8 decreased the composition property and the effectiveness quality attribute. The “System expansion request” functionalities are merged in the “System feedback” class, which suppressed its aggregation relationship and the effectiveness property of D8 (figure 132). The impact of those changes and the applied polymorphism adaptation is illustrated in the simulated and the real results.

#### 1) Simulated results

Polymorphism should be increased by four to effectively adapt the value of effectiveness (figure 131).

#### 2) Real results

After adding four polymorphic equations to D8 such as “deleteRequest (type: String)”, the resulting effectiveness nearly equal its simulated value (figure 132).

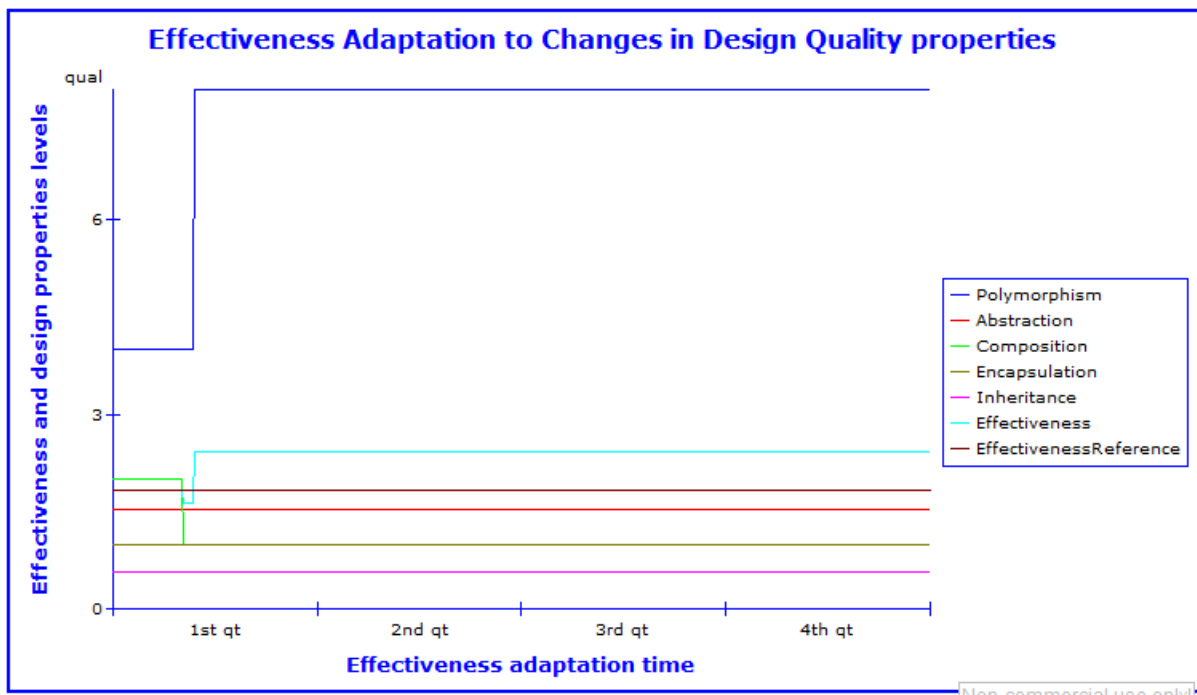
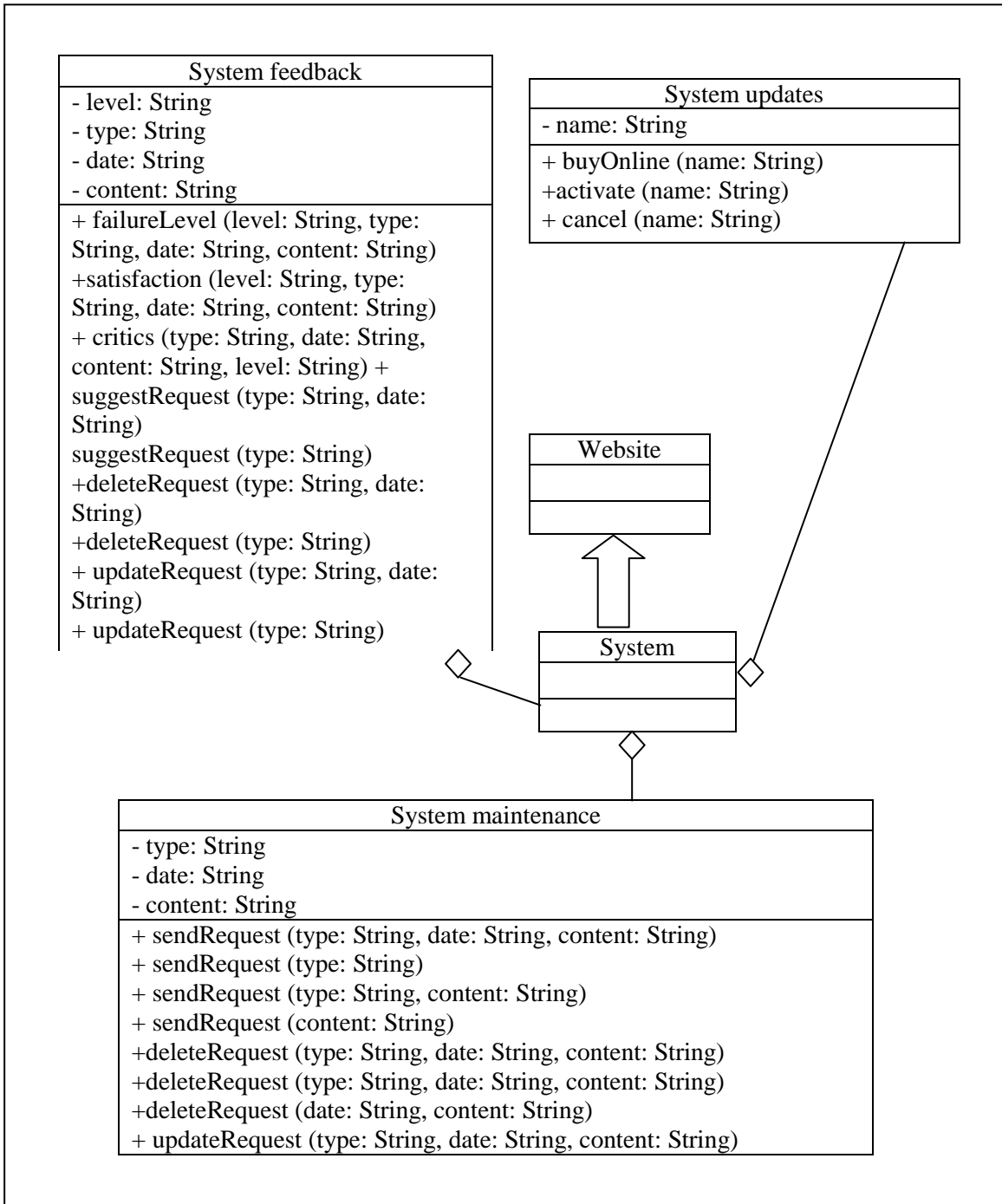


Figure 131: Effectiveness adaptation results of D8

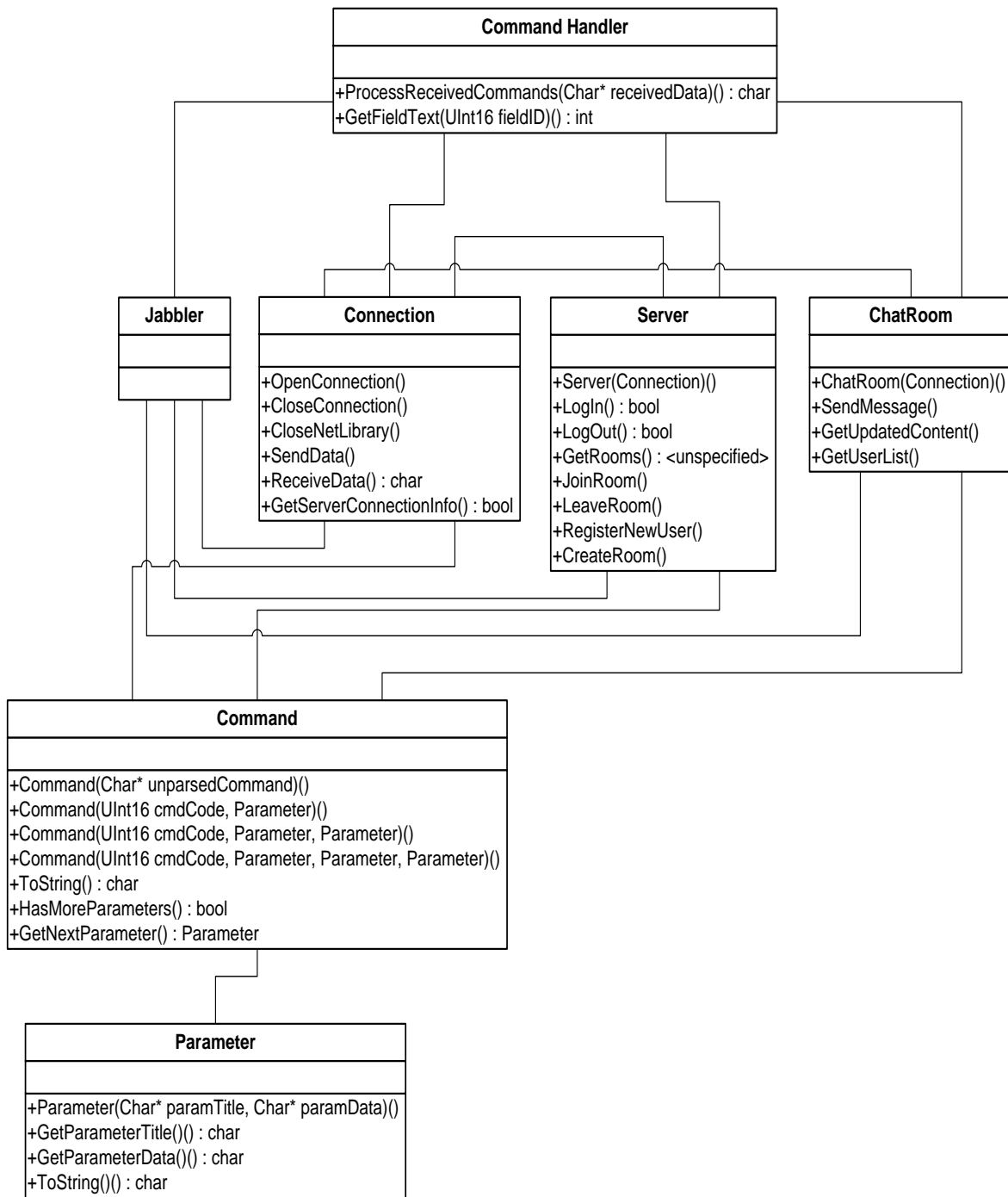


**Figure 132: Effectiveness design change and adaptation in D8**

## **C.8 Design 3 (D9): Jabbler chat system**

### **C.8.1 System description and reference quality values**

Jabblar is a text based online chat system that offers many options to users. Besides viewing a list of chat rooms, a user can join a room, view its messages, send messages, and even join multiple chat rooms simultaneously. Figure 133 illustrates the class diagram of D9. The reference quality values of D9 for all the quality attributes can be accessed in tables 17 and 18.



**Figure 133: D9 Class diagram**

## C.8.2 Design changes

### C.8.2.1 Design changes affecting the understandability quality attribute

D9 was extended by adding two new options that enable users to share videos and pictures as it is illustrated in the classes: “Video share” and “Picture share” (figure 135).

Increasing D9’s design size led to a decrease in understandability that was adapted by increasing encapsulation.

#### 1) Simulated results

The observed decrease in understandability after increasing design size is effectively adapted by increasing encapsulation by two (figure 134).

#### 2) Real results

Encapsulation was maximized in two classes of D9 as it is illustrated in figure 135. As a result, the obtained real understandability equals its simulated value.

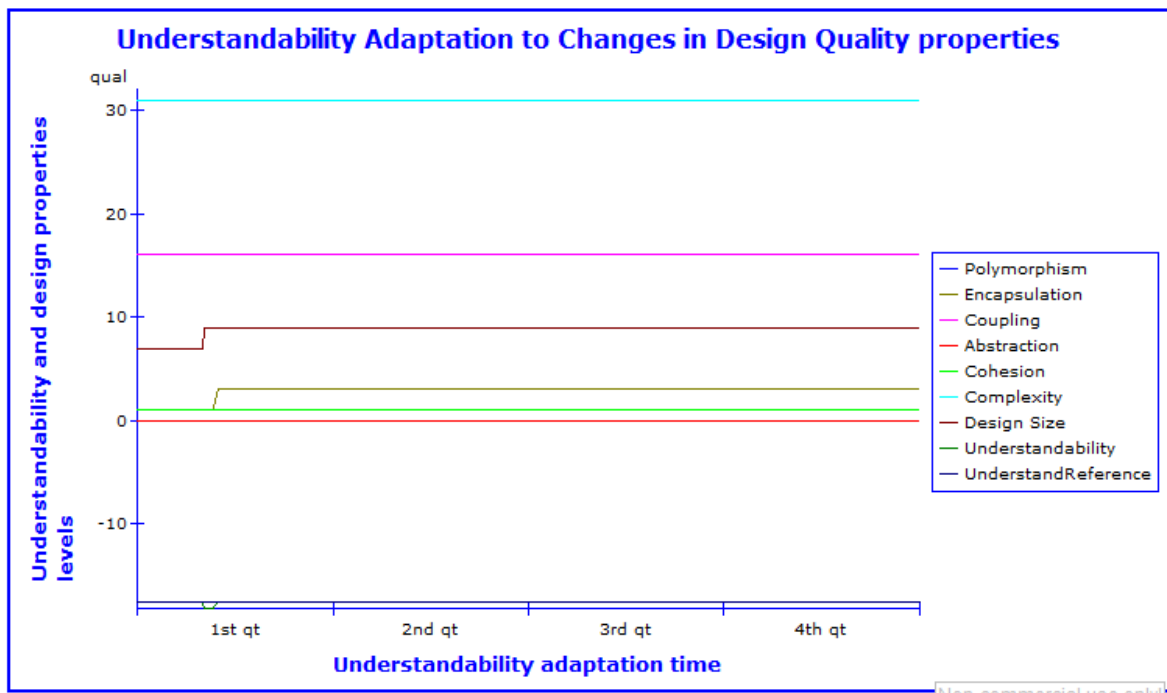
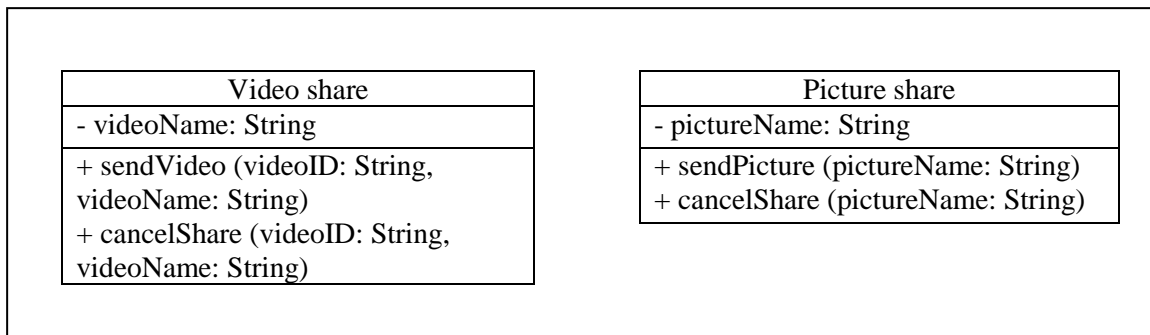


Figure 134: Understandability adaptation results of D9



**Figure 135: Understandability design change and adaptation in D9**

### **C.8.2.2 Design changes affecting the extendibility and the flexibility quality attributes**

The newly added classes were linked to the “ChatRoom” class through aggregation relationships, which increased the rate of coupling in D9 (figure 138). Consequently, Both the flexibility and the extendibility quality attributes dropped below their reference values as it is illustrated and adapted in the simulated and the real results.

#### **1) Simulated results**

To counterbalance the decrease in flexibility and extendibility, polymorphism should be increased by one (figures 136 and 137).

#### **2) Real results**

The class diagram of D9 was updated by adding one polymorphic method namely “sendVideo (videoID: String)”, which made the resulting real flexibility and extendibility attributes nearly equal their simulated values (figure 138).

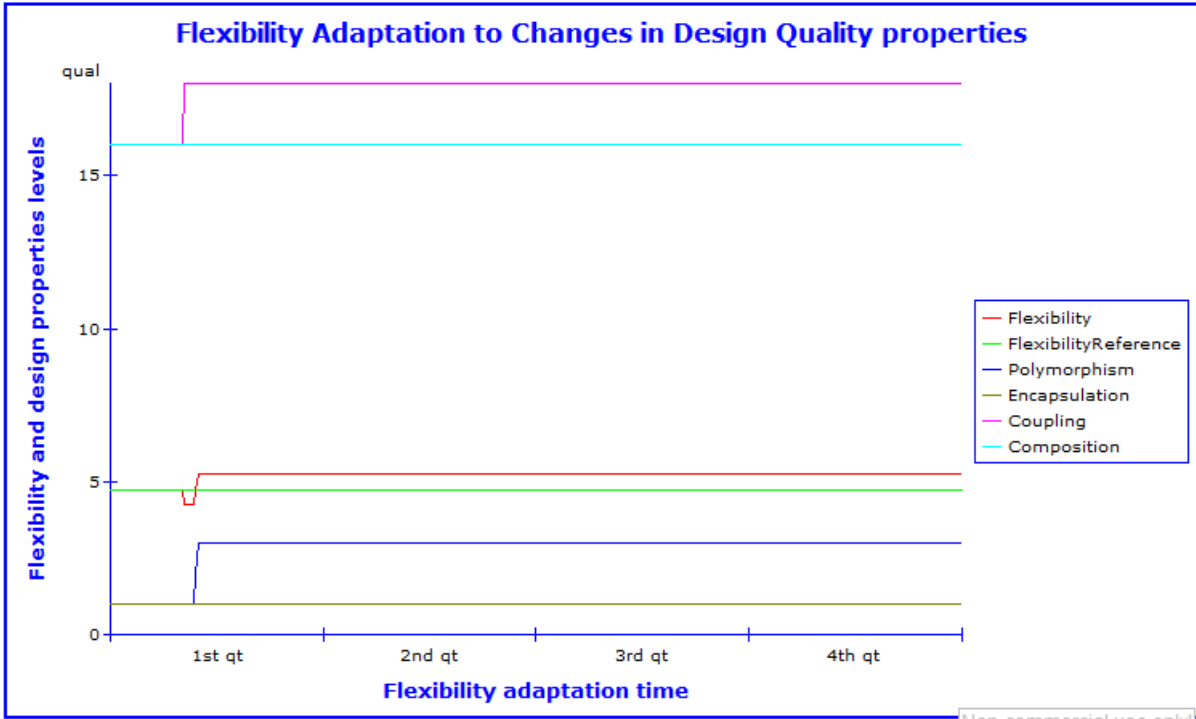


Figure 136: Flexibility adaptation results of D9

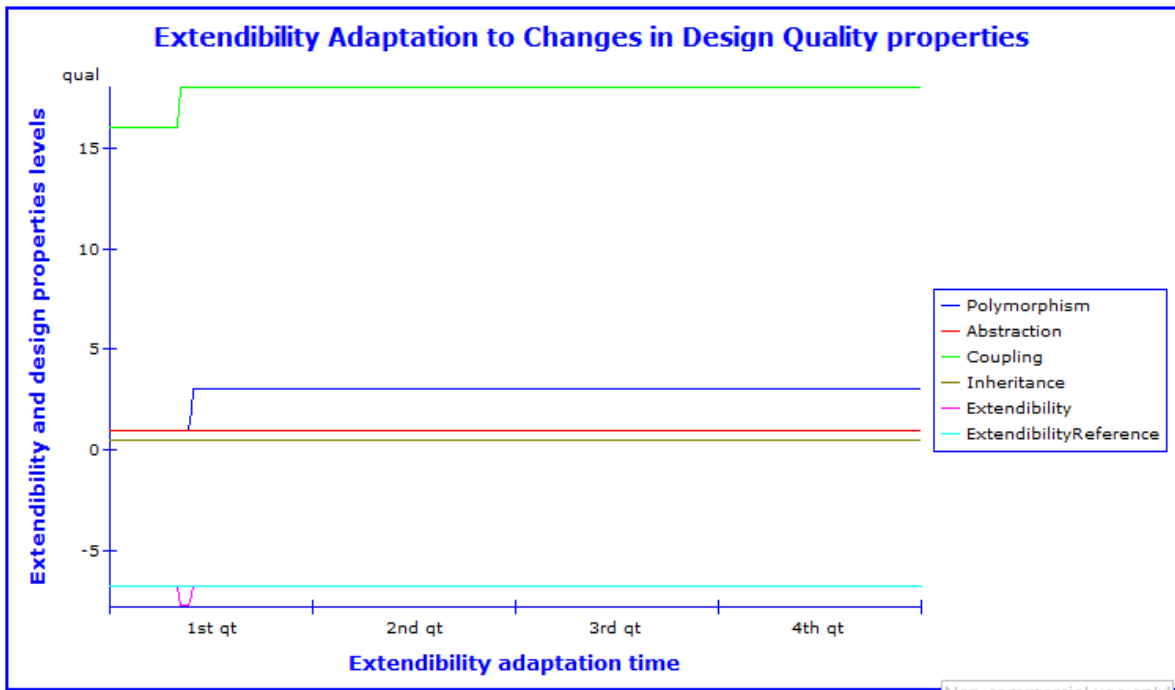
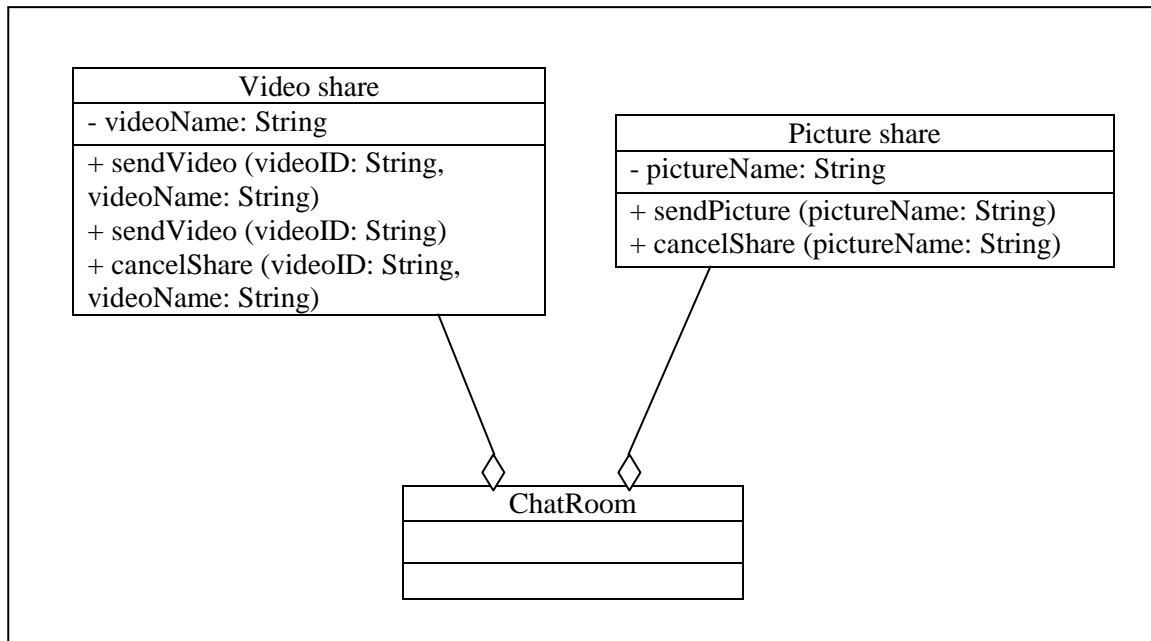


Figure 137: Extendibility adaptation results of D9



**Figure 138: Flexibility and extendibility design change and adaptation in D9**

### C.8.2.3 Design changes affecting the extendibility and the understandability quality attributes

The chat capabilities of “Jabber” were improved by adding the “Video chat” and the “Voice chat” classes (figure 141). Those newly added classes were linked to the existing classes of D9 through aggregation relationships. As a result, the understandability and the extendibility quality attributes dropped below their reference values as it is described in the simulated and real results.

#### 1) Simulated results

On the one hand, extendibility can be adapted by increasing polymorphism to six (figure 139). On the other hand, understandability can be adapted by maximizing encapsulation in the two newly added classes (figure 140).



## 2) Real results

From figure 141, encapsulation was maximized in the “Video chat” and the “Voice chat” classes. In addition, six polymorphic methods were added such as “sendVideo (videoID: String)”. Consequently, the resulting extendibility and understandability nearly equal their simulated values.

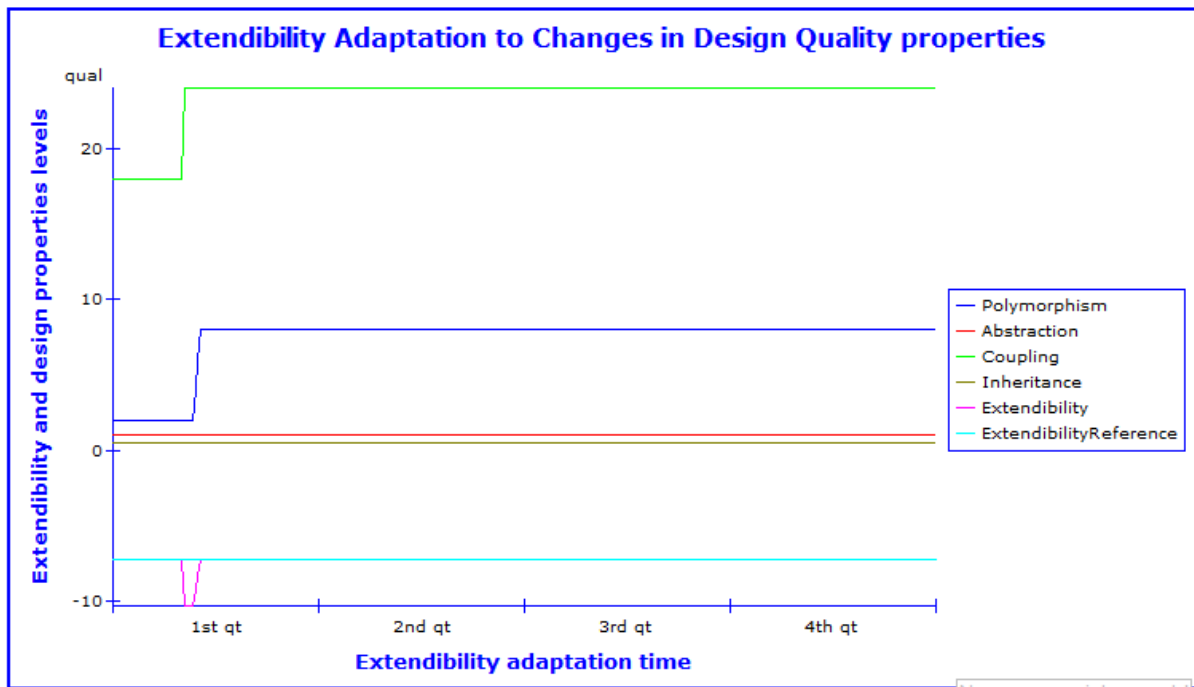


Figure 139: Second extendibility adaptation results of D9

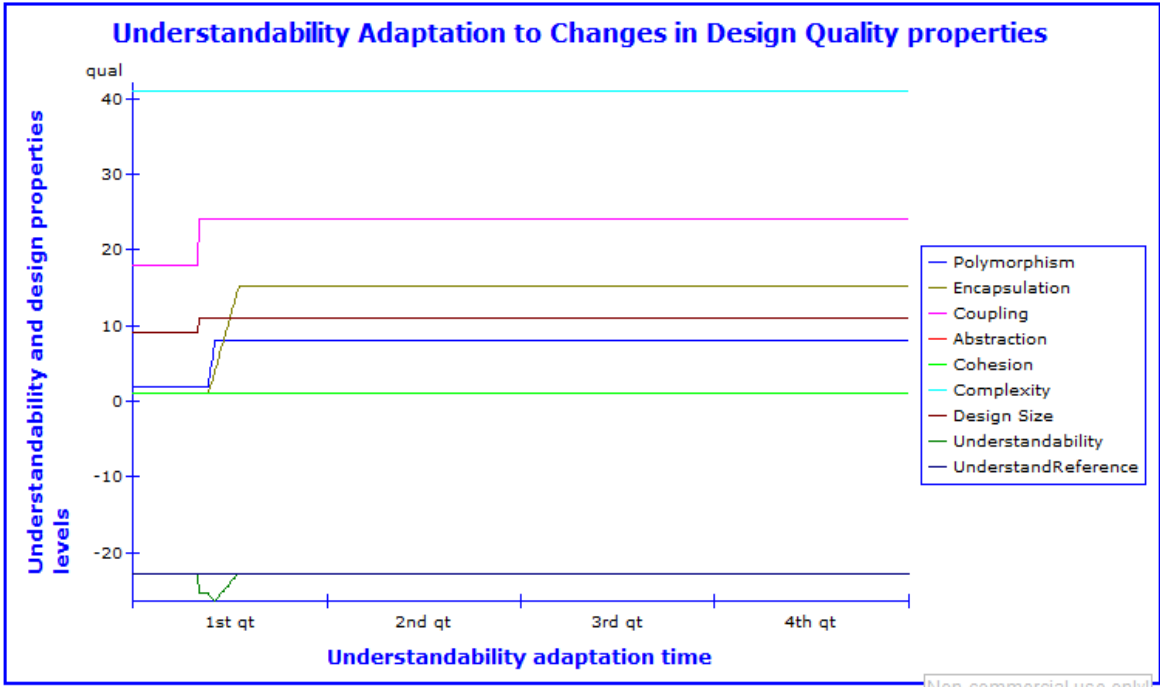
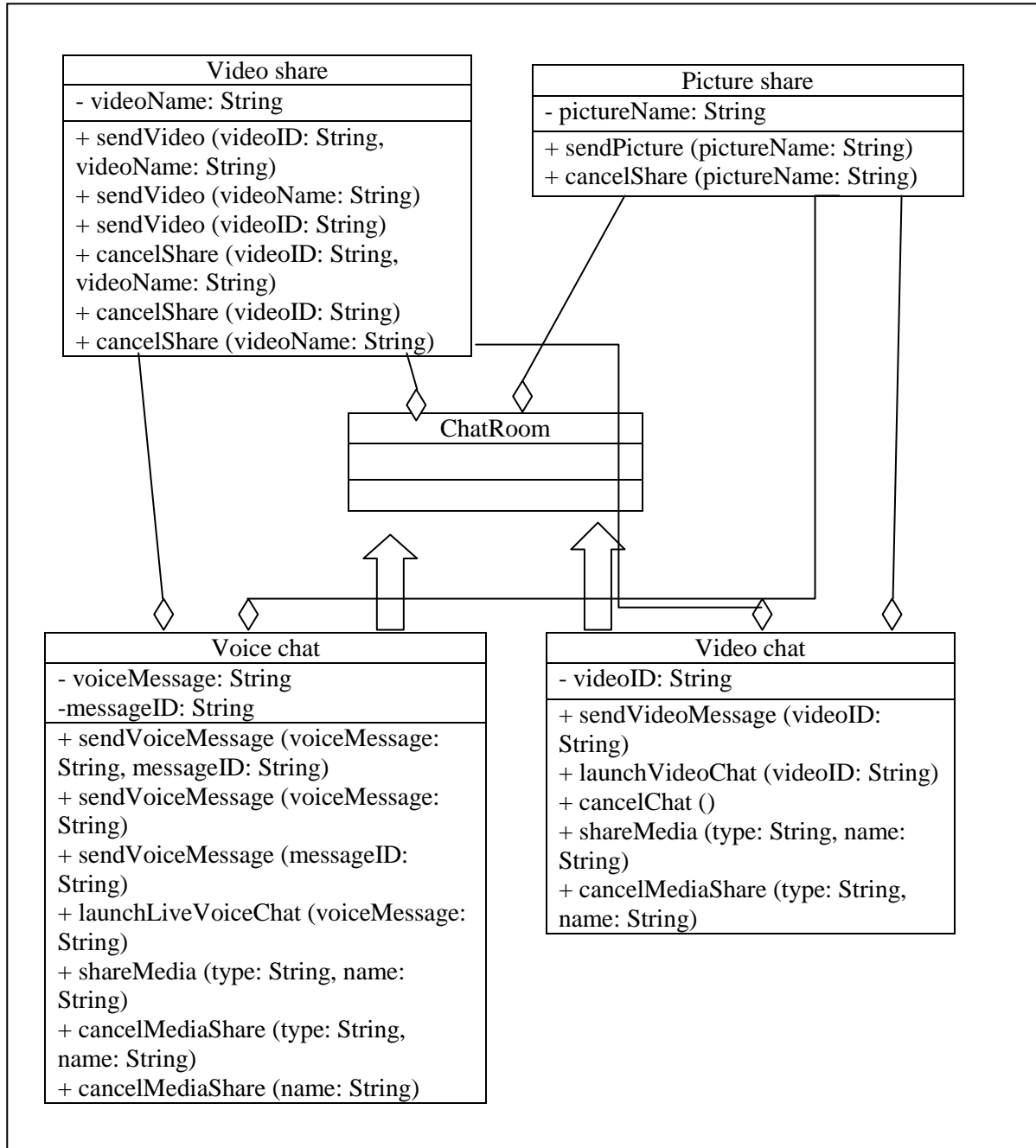


Figure 140: Second understandability adaptation results of D9



**Figure 141: Second extendibility and Understandability design change and adaptation in D9**

#### **C.8.2.4 Design changes affecting the reusability, the functionality and the effectiveness quality attributes**

The class diagrams “Video share” and “Picture share” were deleted from D9 as well as their corresponding aggregation relationships (figure 145). Thus, the reusability, the functionality, and the effectiveness of D9 decreased significantly. To face the impact of those design changes, the equations of cohesion and polymorphism were applied.

##### **1) Simulated results**

From figures 142 and 143, reusability and functionality can be adapted by maximizing cohesion in one class. Effectiveness can be adapted by increasing polymorphism to eight (figure 144).

##### **2) Real results**

Cohesion was maximized in the “Video chat” class, which adapted the values of reusability and functionality. Eight polymorphic methods such as “cancelShare (videoID: String)” were added to D9 to adapt the value of effectiveness (figure 145). The obtained real values of the quality attributes almost equal their simulated values.

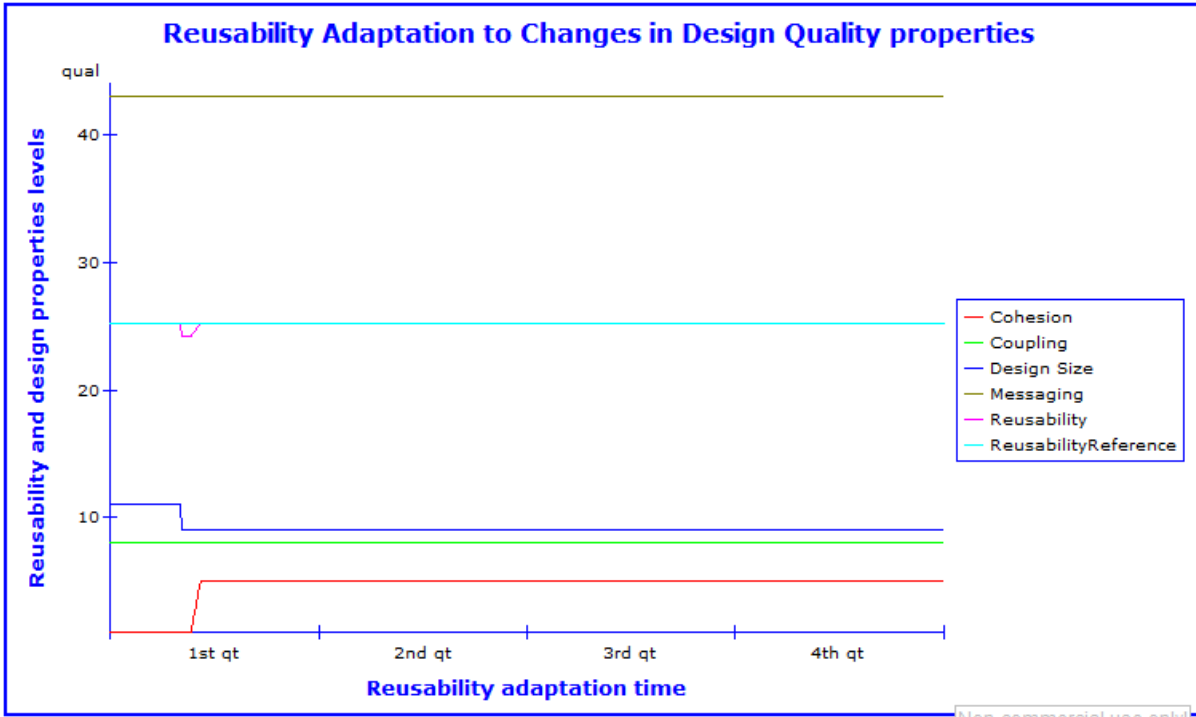


Figure 142: Reusability adaptation results of D9

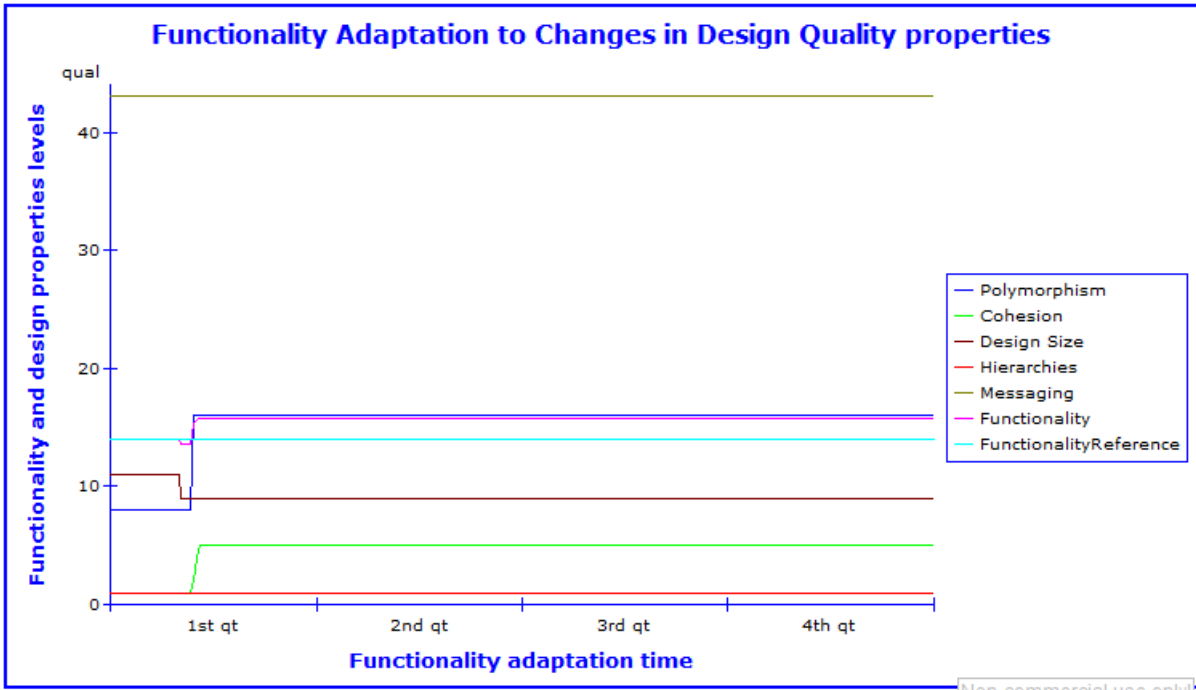


Figure 143: Functionality adaptation results of D9

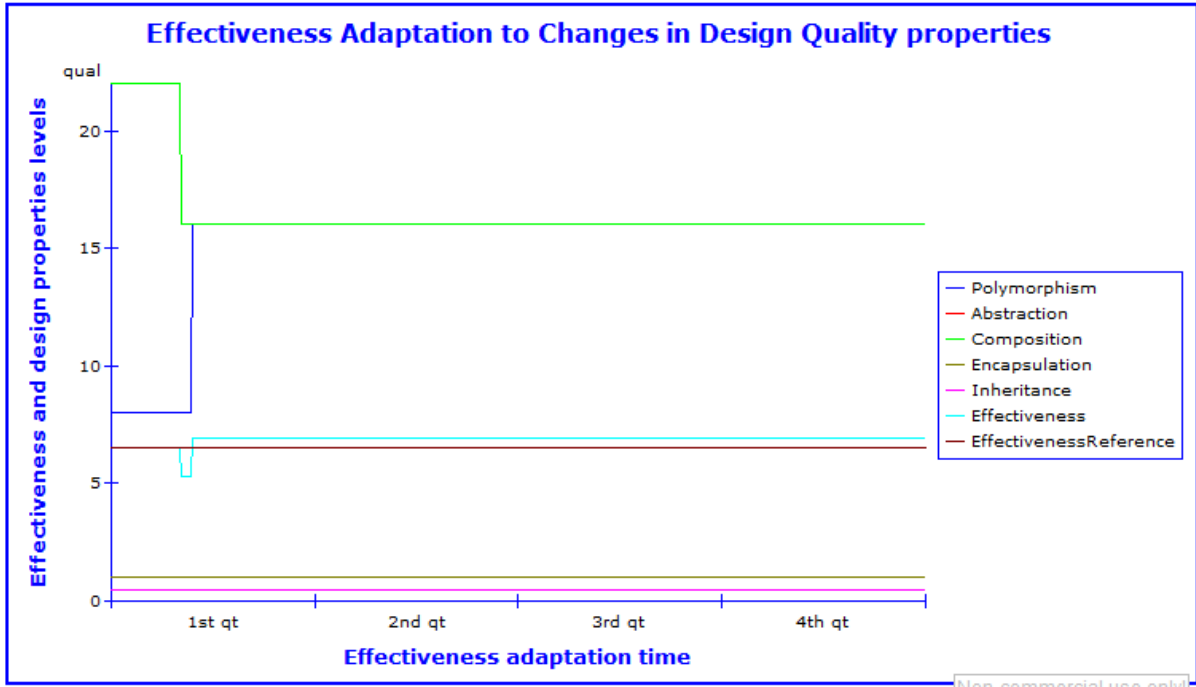
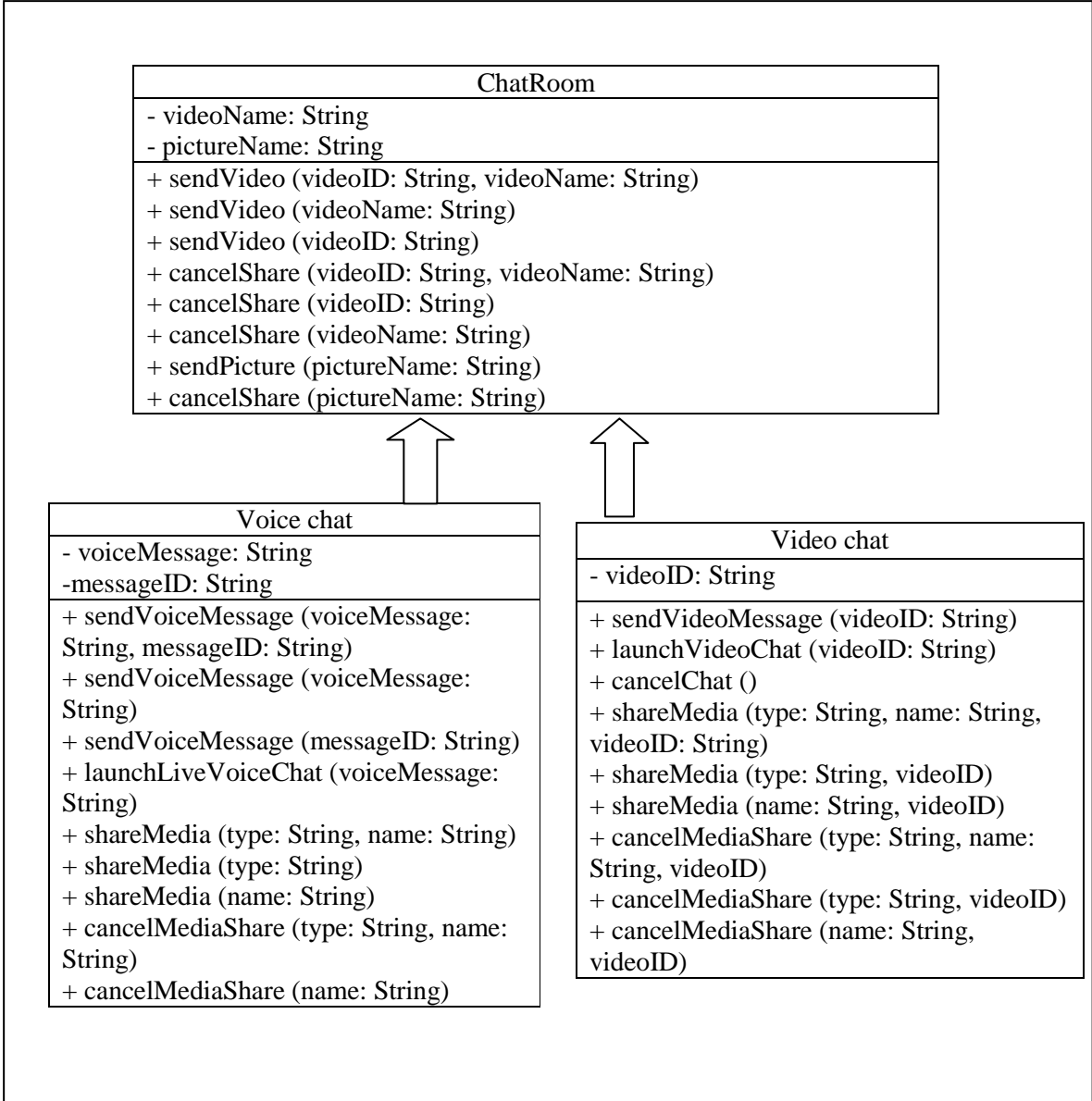


Figure 144: Effectiveness adaptation results of D9



**Figure 145: Reusability, functionality, and effectiveness design change and adaptation in D9**

## C.9 Design 10 (D10): Darden wellness center

### C.9.1 System description and reference quality values

The Darden wellness system allows nurses to input patients' information into electronic forms, add new patients as well as generate their records. The class diagram of D10 is represented in figures 146-150. The reference quality values are recorded in tables 17 and 18.

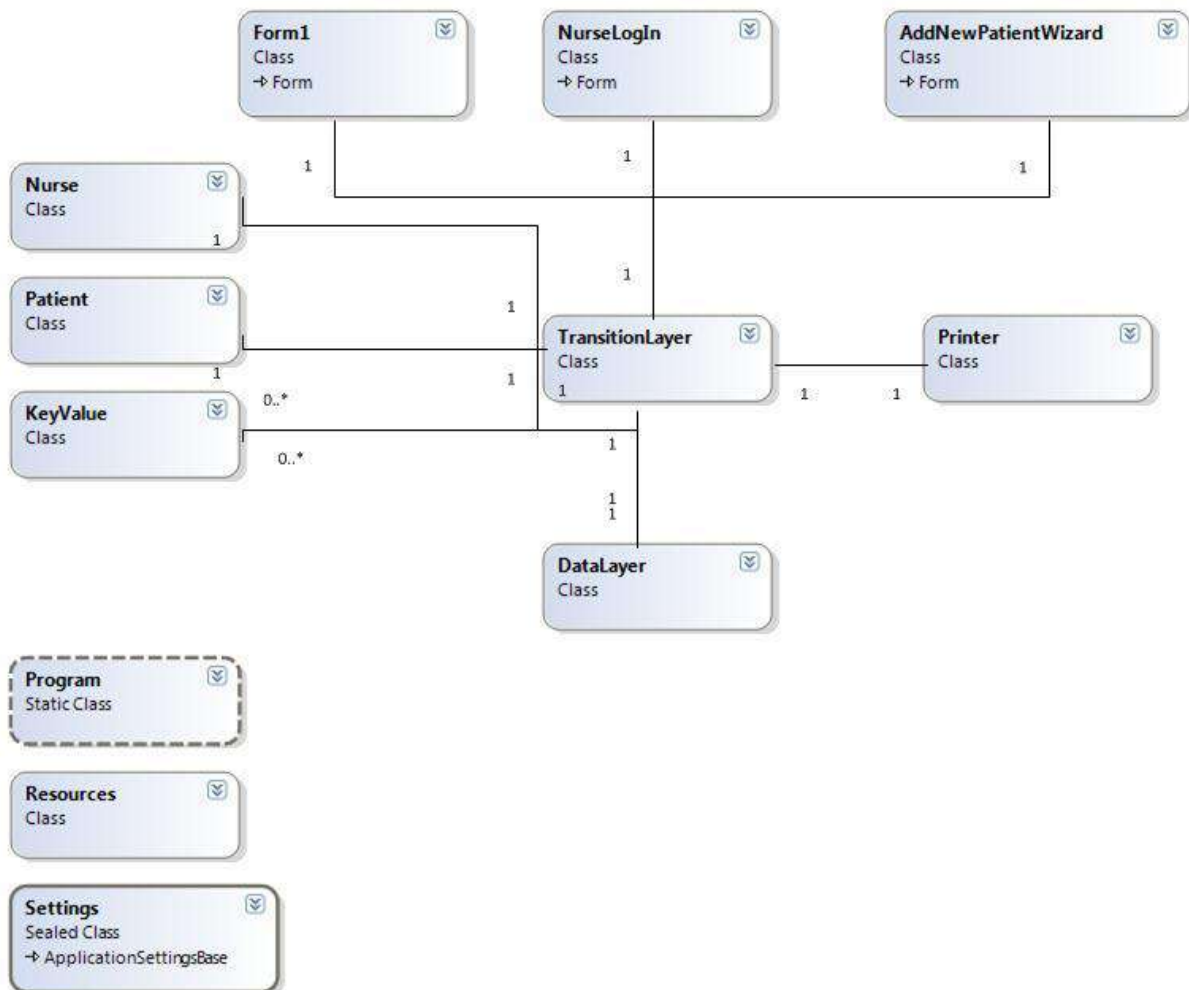


Figure 146: D10 class diagram (1)



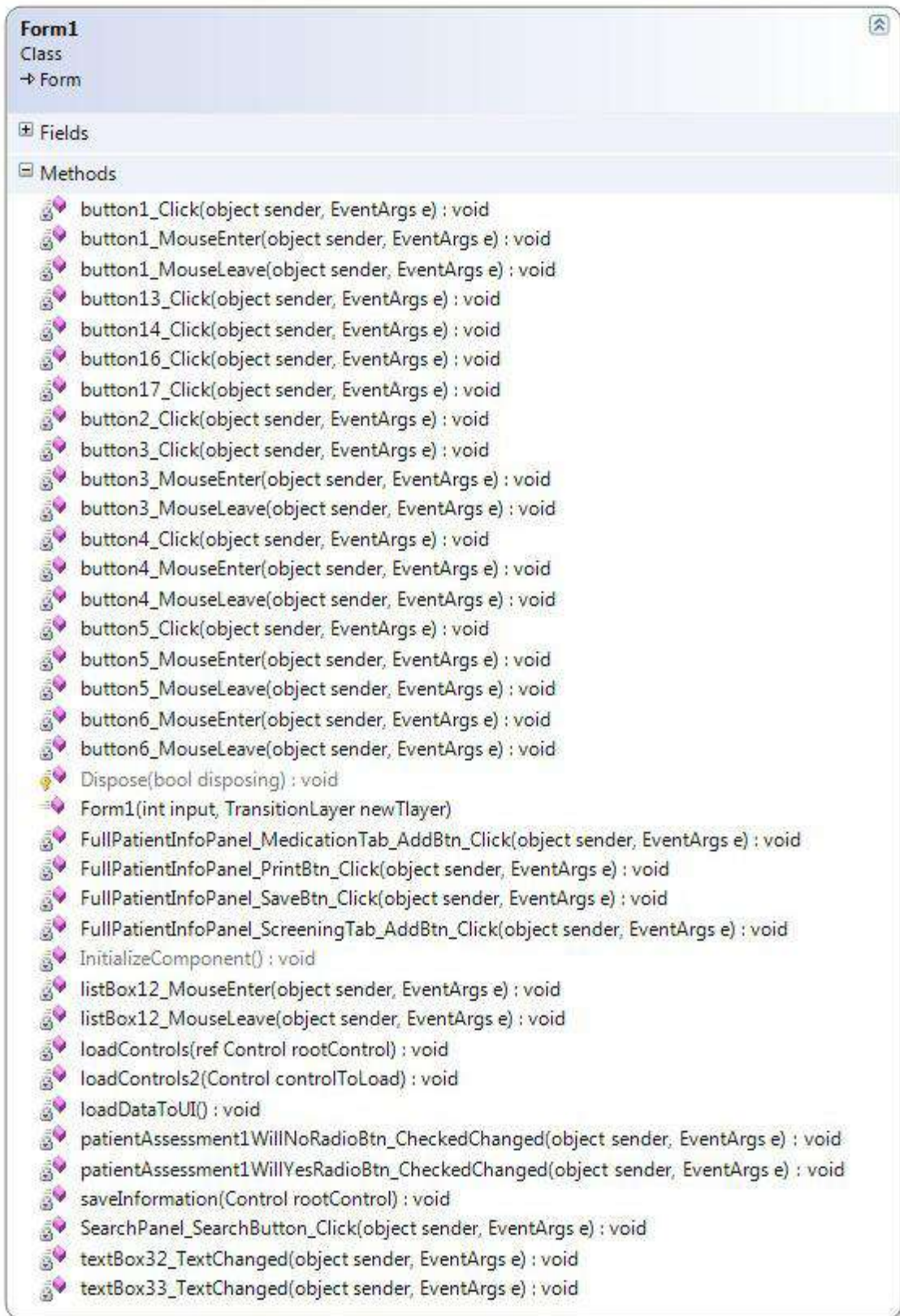
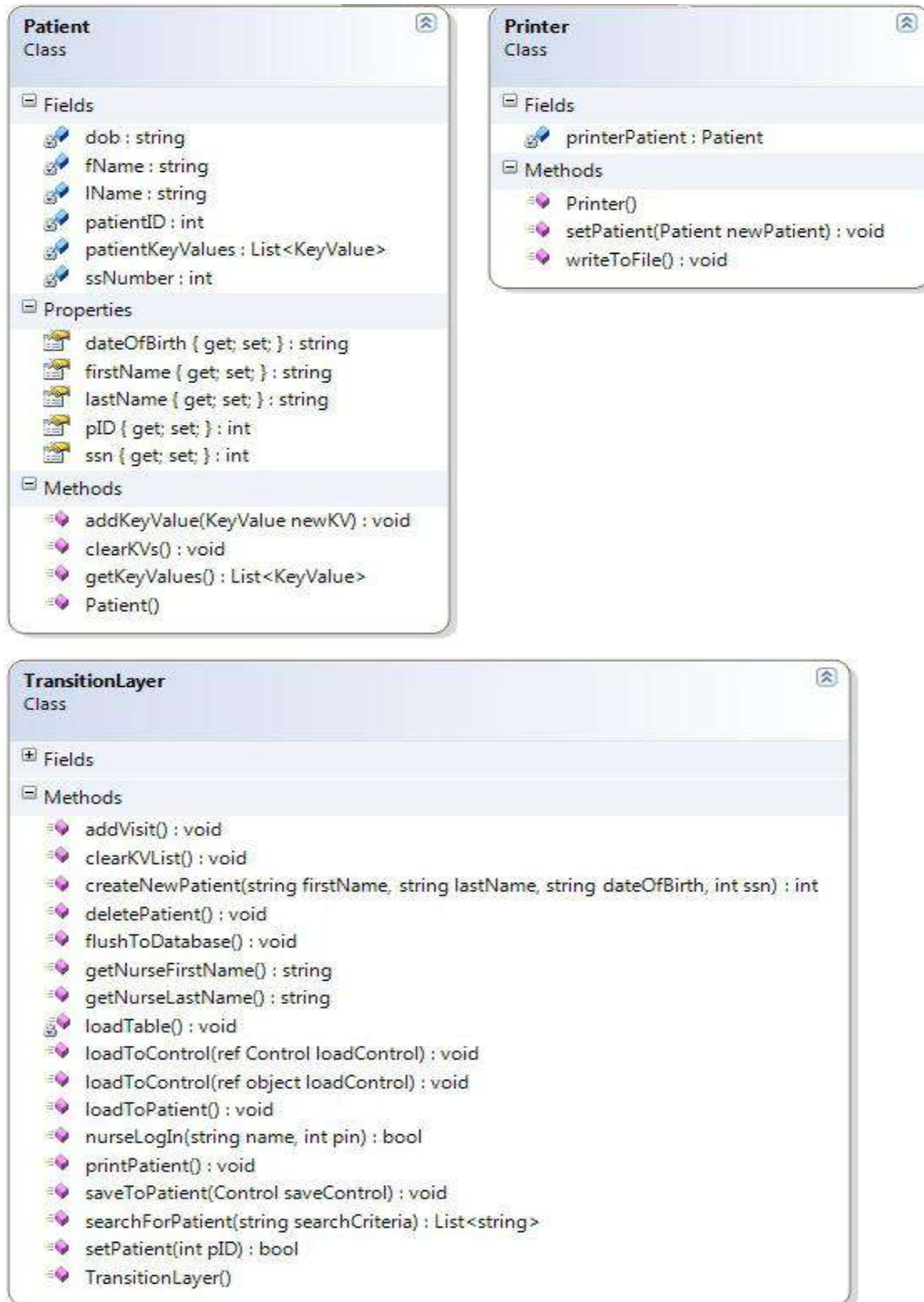


Figure 147: D10 class diagram (2)



**Figure 148: D10 class diagram (3)**



**AddNewPatientWizard**  
Class  
→ Form

Fields

Methods

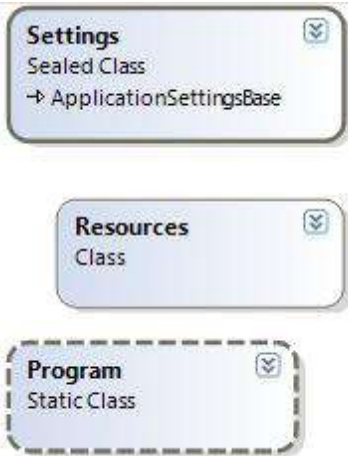
- ◆ AddNewPatientWizard()
- ◆ Back\_Btn\_Click(object sender, EventArgs e) : void
- ◆ Cancel\_Btn\_Click(object sender, EventArgs e) : void
- ◆ dateOfBirth\_MaskTB\_MouseUp(object sender, MouseEventArgs e) : void
- ◆ Dispose(bool disposing) : void
- ◆ emergencyContact\_phoneNumber\_MaskedTB\_MouseUp(object sender, MouseEventArgs e) : void
- ◆ ErrorMessage(Control errorTestedControl) : bool
- ◆ ErrorMessage(RadioButton errorTestedControl1, RadioButton errorTestedControl2) : bool
- ◆ firstName\_TB\_KeyPress(object sender, KeyPressEventArgs e) : void
- ◆ InitializeComponent() : void
- ◆ lastName\_TB\_KeyPress(object sender, KeyPressEventArgs e) : void
- ◆ LegalAndInsurance\_Groupbox\_Enter(object sender, EventArgs e) : void
- ◆ maskedTextBox1\_MouseUp(object sender, MouseEventArgs e) : void
- ◆ maskedTextBox4\_MouseUp(object sender, MouseEventArgs e) : void
- ◆ maskedTextBox5\_MouseUp(object sender, MouseEventArgs e) : void
- ◆ maskedTextBox6\_MouseUp(object sender, MouseEventArgs e) : void
- ◆ member1\_Birthday\_MaskTB\_MouseUp(object sender, MouseEventArgs e) : void
- ◆ member2\_Birthday\_MaskTB\_MouseUp(object sender, MouseEventArgs e) : void
- ◆ member3\_Birthday\_MaskTB\_MouseUp(object sender, MouseEventArgs e) : void
- ◆ member4\_Birthday\_MaskTB\_MouseUp(object sender, MouseEventArgs e) : void
- ◆ member5\_Birthday\_MaskTB\_MouseUp(object sender, MouseEventArgs e) : void
- ◆ member6\_Birthday\_MaskTB\_MouseUp(object sender, MouseEventArgs e) : void
- ◆ midName\_TB\_KeyPress(object sender, KeyPressEventArgs e) : void
- ◆ Next\_Btn\_Click(object sender, EventArgs e) : void
- ◆ phoneNumber\_maskedTB\_MouseUp(object sender, MouseEventArgs e) : void
- ◆ saveInformation(Control rootControl) : void

**DataLayer**  
Class

Methods

- ◆ addNewPatient(string firstName, string lastName, string dateOfBirth, int ssn) : int
- ◆ createDBConnection() : SqlConnection
- ◆ DataLayer()
- ◆ getNonUniqueAddString(string ParatableName, int id, int paraPid, string Parakey, string Paravalue) : string
- ◆ getTableLength(string tableName) : int
- ◆ getUniqueAddString(string ParatableName, int id, int paraPid, string Parakey, string Paravalue) : string
- ◆ grabKVFromTables(string tableName, int pID) : List<KeyValue>
- ◆ searchForPatient(int searchID) : List<string>
- ◆ searchForPatient(string searchInput) : List<string>
- ◆ validateNurse(string firstname, string lastname, int pin) : int
- ◆ writeKVToDatabase(KeyValue kvToWrite, int pID) : void

**Figure 149: D10 class diagram (4)**



**Figure 150: D10 class diagram (5)**

## **C.9.2 Design changes**

### **C.9.2.1 Design changes affecting the understandability quality attribute**

The computerized system of the Darden center was extended by adding three options dealing with insurance, vaccination, and pharmacy (figure 152). This increase in the design size of D10 led to a decrease in understandability that was adapted by increasing encapsulation.

#### **1) Simulated results**

From the simulation results in figure 151, encapsulation should be increased by three to counterbalance the decrease in understandability.

#### **2) Real results**

In figure 152, encapsulation was maximized in the three new added classes, which makes the value of the real understandability equal its simulated value.

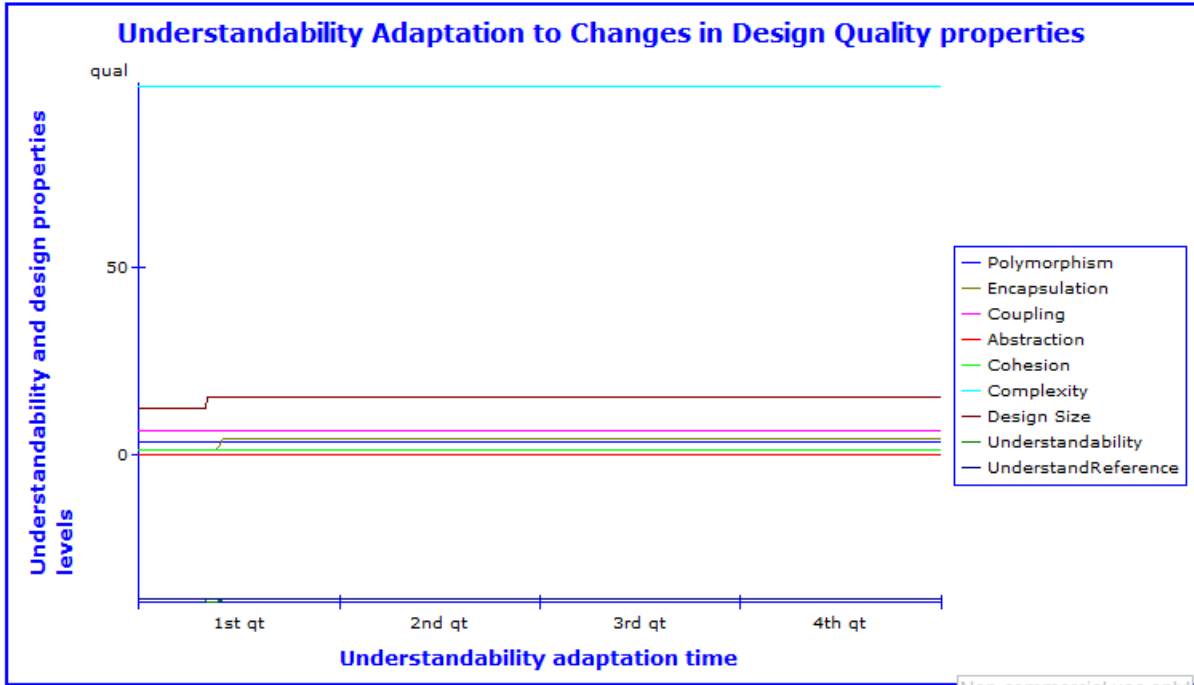


Figure 151: Understandability adaptation results of D10

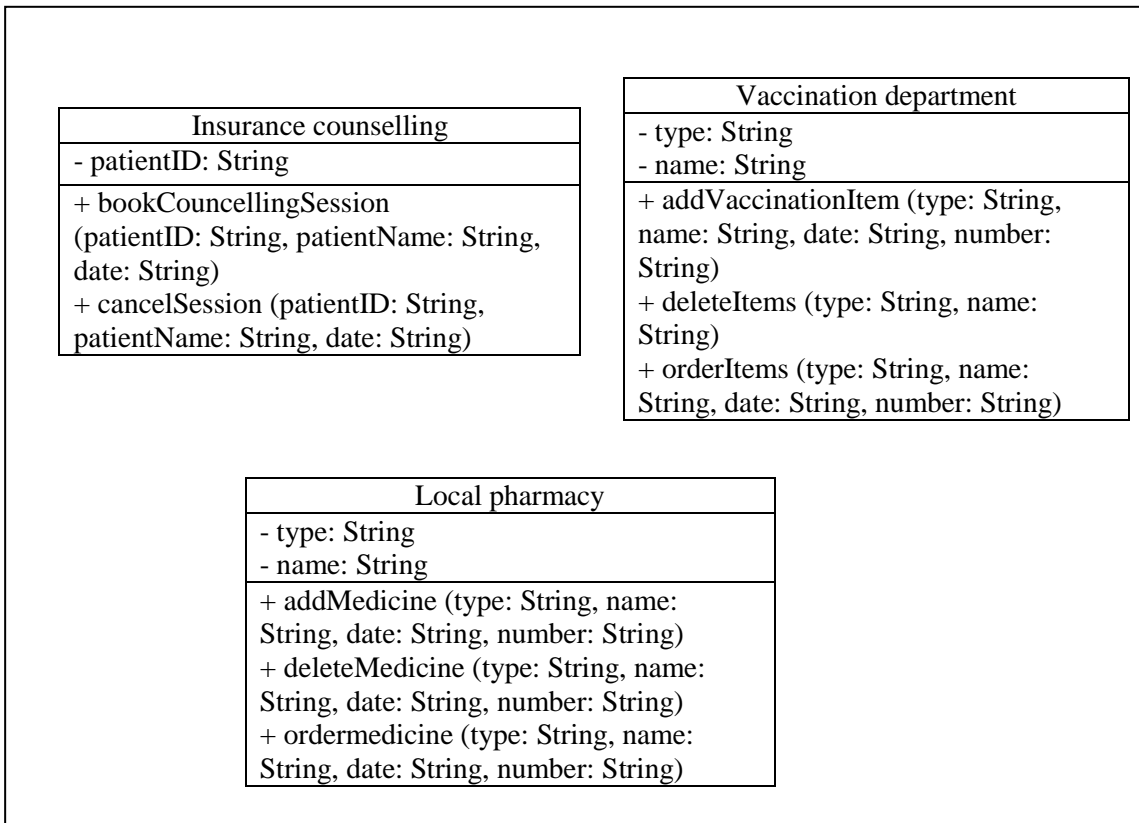


Figure 152: Understandability design change and adaptation of D10

### C.9.2.2 Design changes affecting the extendibility and the flexibility quality attributes

The coupling level of D10 was increased by linking the newly added classes to the existing classes through aggregation relationships (figure 155). As a result, extendibility and flexibility dropped below their reference values. Those design changes and the applied adaptation through polymorphism is illustrated in the real and the simulated results.

#### 1) Simulated results

Extendibility and flexibility are effectively adapted when polymorphism is increased by three (figures 153 and 154).

#### 2) Real results

After adding three polymorphic methods to D10 such as “bookCounsellingSession (patientID: String)”, the obtained extendibility and flexibility are similar to their simulated values (figure 155).

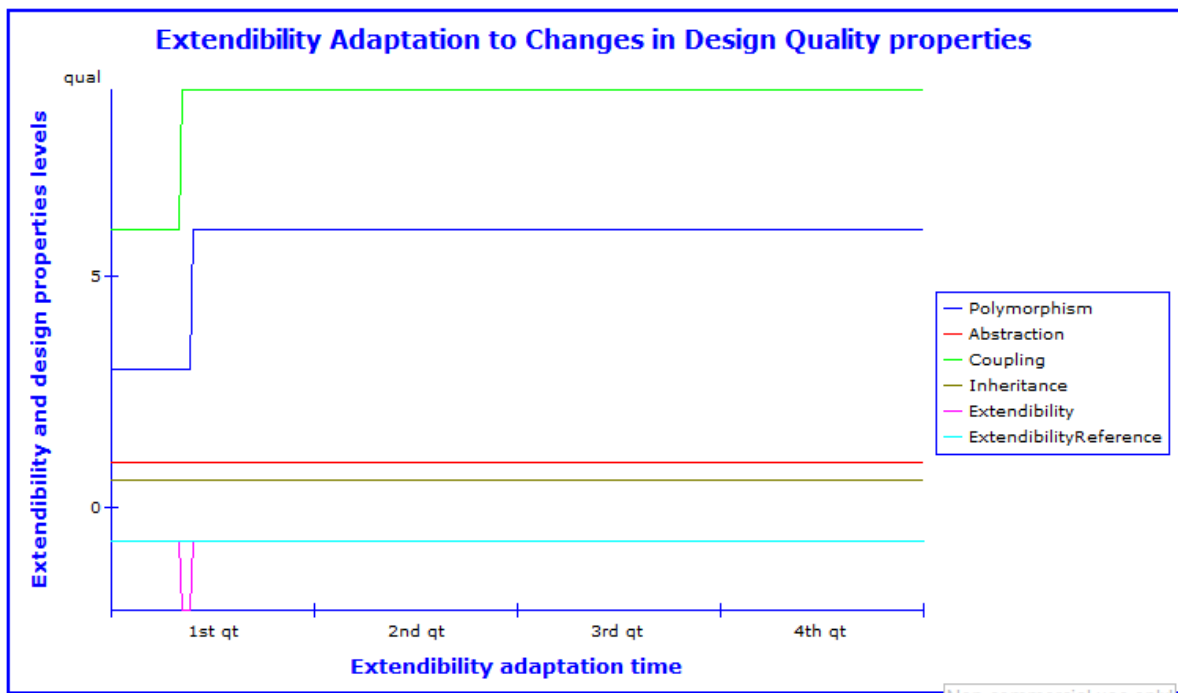


Figure 153: Extendibility adaptation results of D10

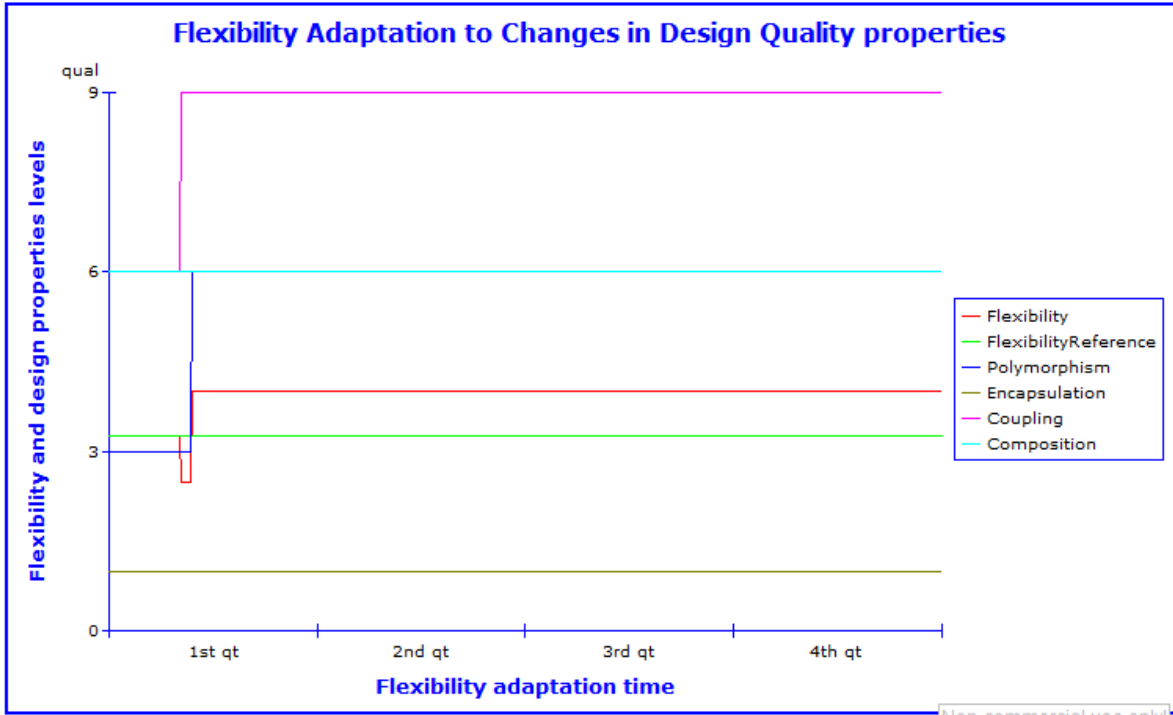
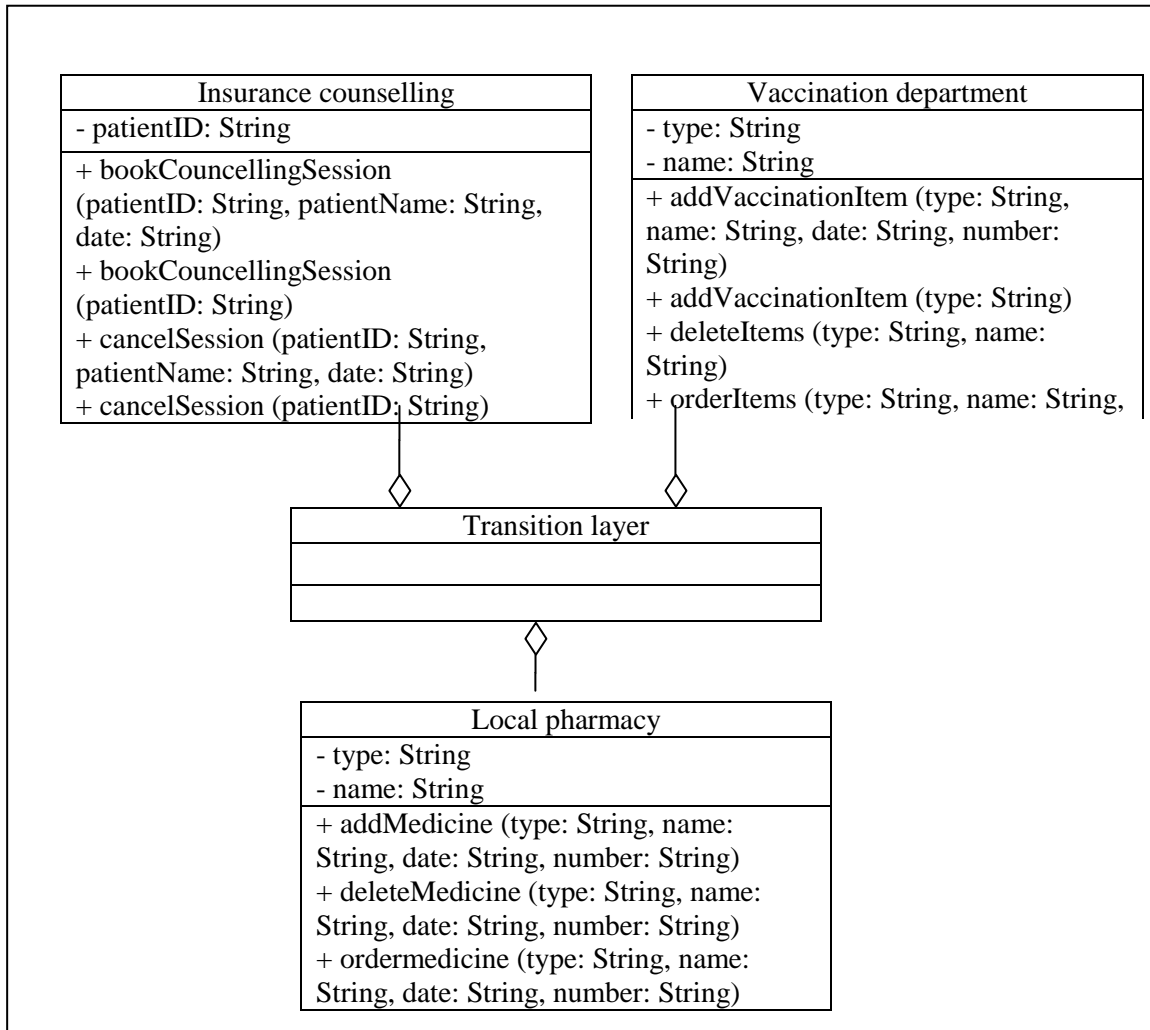


Figure 154: Flexibility adaptation results of D10



**Figure 155: Flexibility and extendibility design change and adaptation of D10**

### C.9.2.3 Design changes affecting the reusability, the functionality and the effectiveness quality attributes

The last design change in D10 dealt with deleting the class “Insurance counseling” as well as its corresponding aggregation relationship (figure 159). Thus, the reusability, the functionality, and the effectiveness quality attributes dropped below their reference values and were adapted through cohesion and polymorphism.



## 1) Simulated results

Reusability and functionality were adapted by maximizing cohesion in one class (figures 156 and 157). Effectiveness is adapted when polymorphism increases by six (figure 158).

## 2) Real results

The simulated design changes and adaptations were applied in D10's class diagram. Cohesion was maximized in the "Local pharmacy" class and six polymorphic methods were added to D10 such as "addVaccinationItem (type: String)". The computed quality attributes after adaptation almost equal their simulated values.

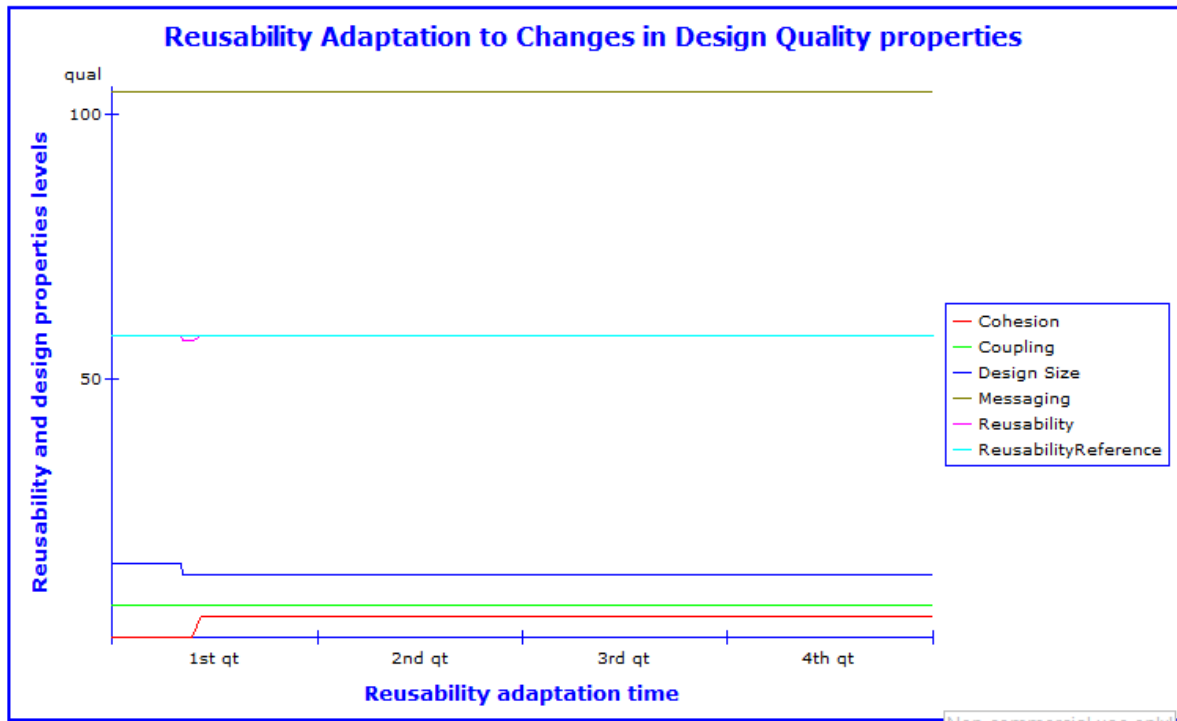


Figure 156: Reusability adaptation results of D10

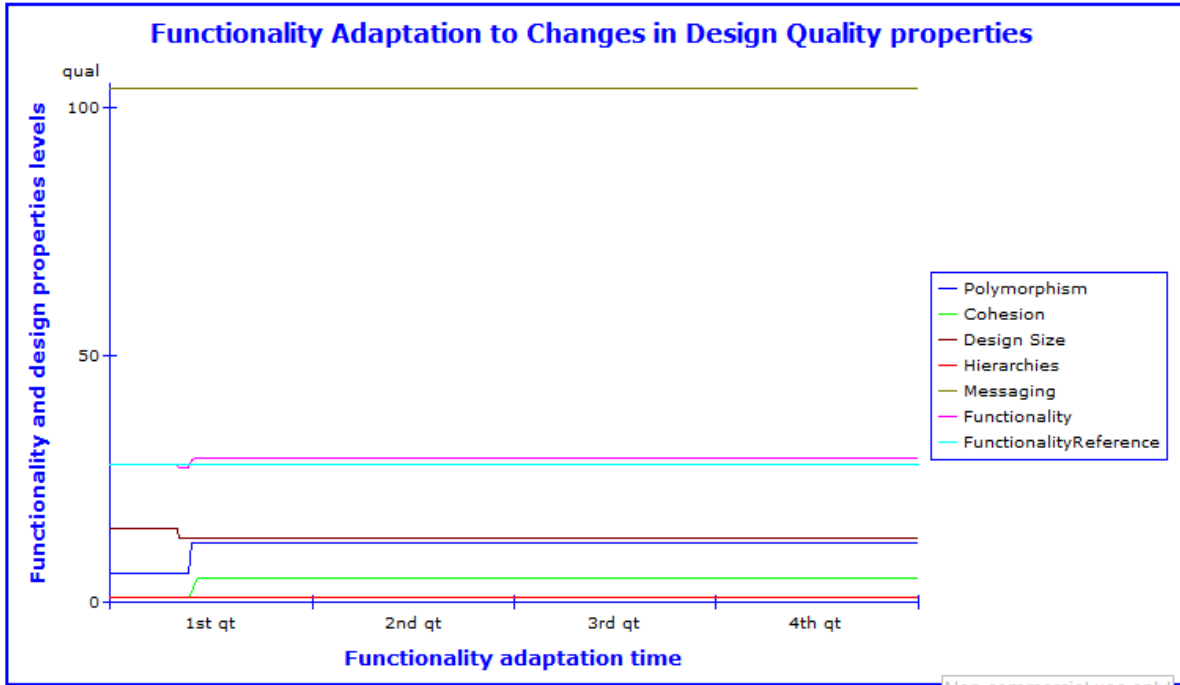


Figure 157: Functionality adaptation results of D10

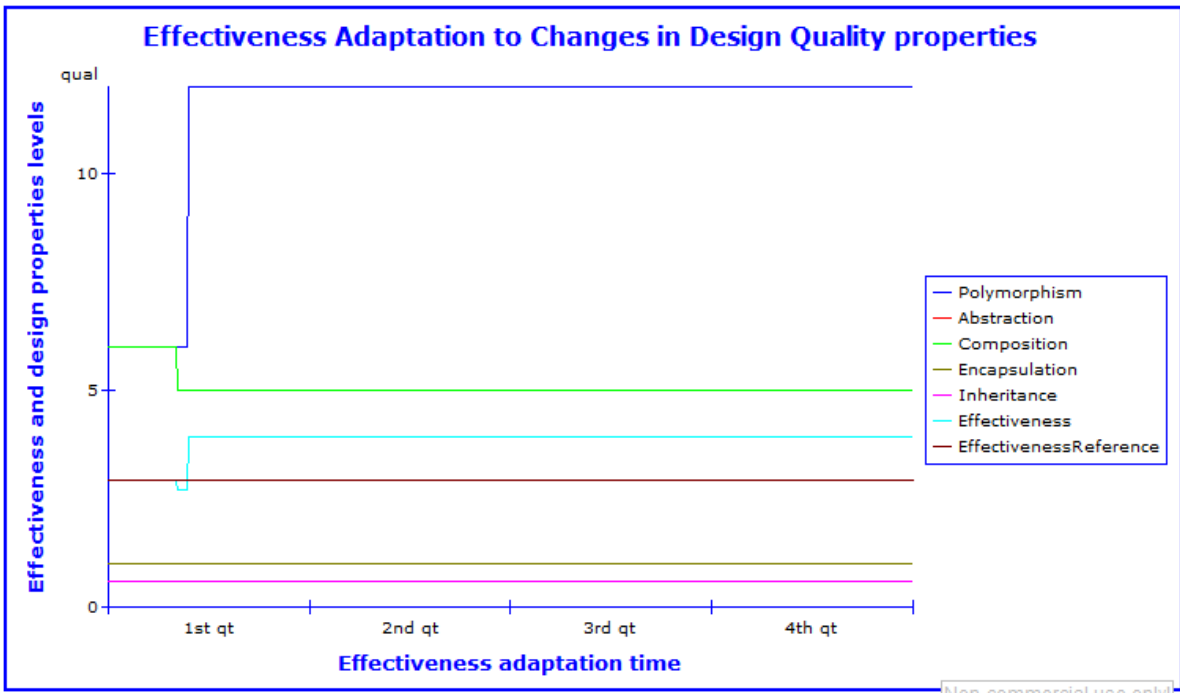
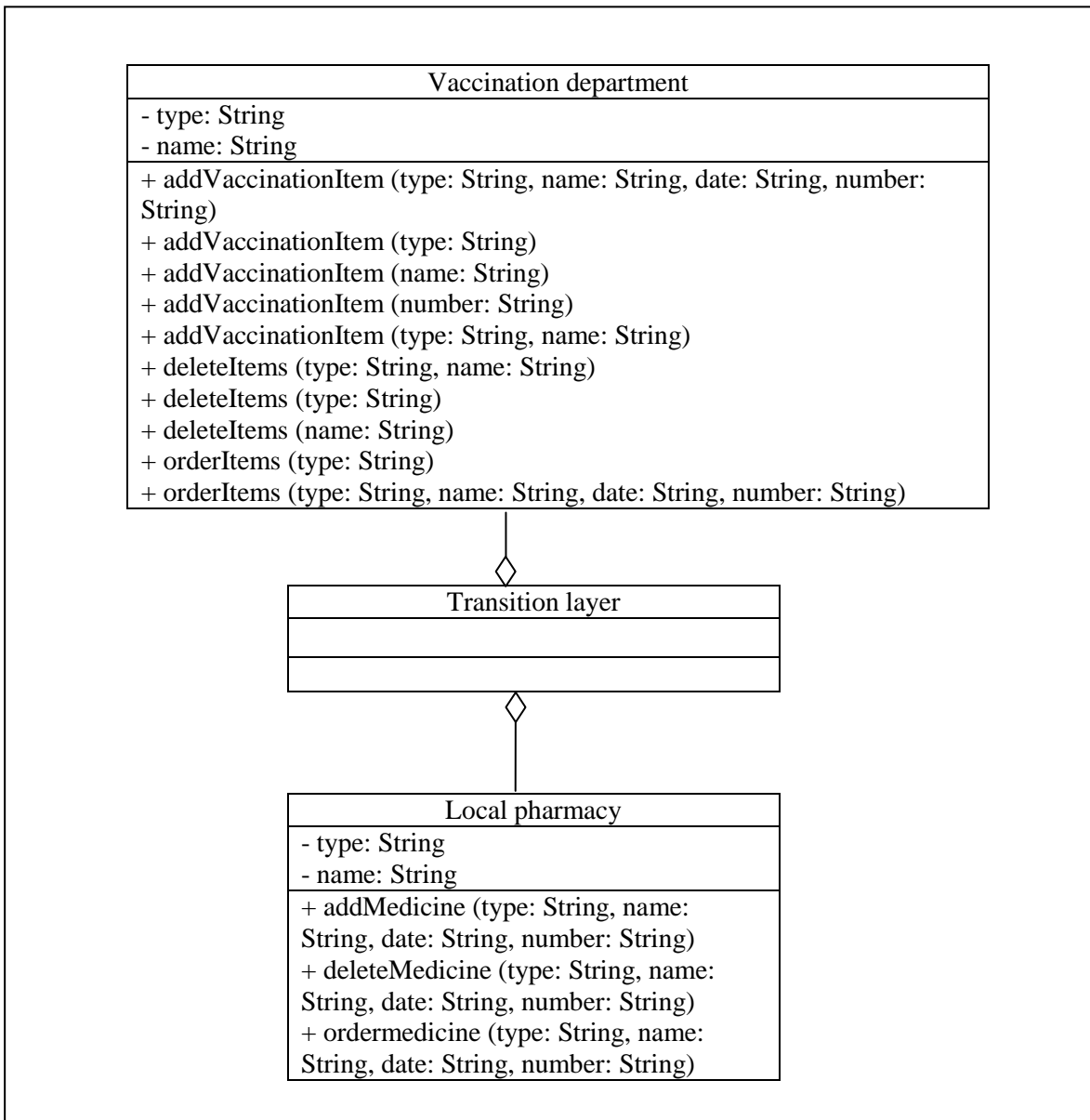


Figure 158: Effectiveness adaptation results of D10



**Figure 159: Reusability, functionality, and effectiveness design change and adaptation of D10**

As it was described in chapter 4, Pearson's r was also computed for the remaining QMOOD quality attributes namely reusability, flexibility, functionality, extendibility, and effectiveness in the following tables.

<b>X: Simulated extendibility in D1-D10</b>	<b>Y: Real extendibility in D1-D10</b>	<b>XY</b>	<b>X<sup>2</sup></b>	<b>Y<sup>2</sup></b>
-12.25	-11.80	144.55	150.06	139.24
-4.10	-4.10	16.81	16.81	16.81
-3.50	-3.50	12.25	12.25	12.25
-8.13	-8.13	66.09	66.09	66.09
-9.06	-9.06	82.08	82.08	82.08
-11.14	-11.14	124.09	124.09	124.09
-3.88	-3.88	15.05	15.05	15.05
-5.45	-5.45	29.70	29.7	29.70
-6.75	-6.75	45.56	45.56	45.56
-0.7	-0.7	0.49	0.49	0.49
<b>Σ X= -64.96</b>	<b>Σ Y= -64.51</b>	<b>Σ XY= 536.67</b>	<b>Σ X<sup>2</sup>= 542.18</b>	<b>Σ Y<sup>2</sup>= 531.36</b>
<b>n=10</b>				

**Table 22: Correlation computations of extendibility**

$$r_{xy} = \frac{10 (536.67) - (-64.96) (-64.51)}{\sqrt{|10 (542.18) - (-64.96)^2| * |10 (531.36) - (-64.51)^2|}}$$

$$= 0.99 \text{ very high correlation}$$

<b>X: Simulated flexibility in D1-D10</b>	<b>Y: Real flexibility in D1-D10</b>	<b>XY</b>	<b>X<sup>2</sup></b>	<b>Y<sup>2</sup></b>
-3	-0.75	2.25	9	0.562
-1.5	6.75	10.12	2.25	45.56
-1.5	0	0	2.25	0
-2.5	-0.5	1.25	6.25	0.25
-3.5	-2.75	9.62	12.25	7.56
-1.5	-1	1.5	2.25	1
0.25	1.25	0.31	0.06	1.56
-2.25	-1.5	3.37	5.06	2.25
4.75	5.25	24.93	22.56	27.56
3.25	4	13	10.56	16
<b>Σ X= -7.5</b>	<b>Σ Y= 10.75</b>	<b>Σ XY= 66.35</b>	<b>Σ X<sup>2</sup>= 72.49</b>	<b>Σ Y<sup>2</sup>= 102.3</b>
<b>n=10</b>				

**Table 23: Correlation computations of flexibility**

$$r_{xy} = \frac{10 (66.35) - (-7.5) (10.75)}{\sqrt{|10 (72.49) - (-7.5)^2| * |10 (102.3) - (10.75)^2|}}$$

= 0.95 very high correlation

<b>X: Simulated reusability</b>	<b>Y: Real reusability</b>	<b>XY</b>	<b>X<sup>2</sup></b>	<b>Y<sup>2</sup></b>
23.20	23.30	538.24	538.24	538.24
31.75	31.75	1008.06	1008.06	1008.06
51.80	51.82	2684.27	2683.24	2685.31
33	33	1089	1089	1089
38.05	37.05	1409.75	1447.80	1372.70
47.75	47.75	2280.06	2280.06	2280.06
61.75	61.75	3813.06	3813.06	3813.06
20	20	400	400	400
21.25	21.75	462.18	451.56	473.06
57.50	57.50	3306.25	3306.25	3306.25
<b>Σ X= 386.05</b>	<b>Σ Y= 385.57</b>	<b>Σ XY= 16990.87</b>	<b>Σ X<sup>2</sup>= 17017.27</b>	<b>Σ Y<sup>2</sup>= 16965.74</b>
<b>n =10</b>				

**Table 24: Correlation computations of reusability**

$$r_{xy} = \frac{10 (16990.87) - (386.05) (385.57)}{\sqrt{|10(17017.27) - (386.05)^2| * |10 (16965.74) - (385.57)^2|}}$$

= 0.99 very high correlation

<b>X: Simulated functionality</b>	<b>Y: Real functionality</b>	<b>XY</b>	<b>X<sup>2</sup></b>	<b>Y<sup>2</sup></b>
17.48	17.08	298.55	305.55	291.72
18.38	18.08	332.31	337.82	326.88
26.64	26.71	711.55	709.68	713.42
20.8	20.86	433.88	432.64	435.13
22.02	23.38	514.82	484.88	546.62
25.86	26.32	680.63	668.73	692.74
30.26	31.38	949.55	915.66	984.70
12.44	12.46	155	154.75	155.25
13.98	15.30	213.89	195.44	234.09
27.84	28.96	806.24	775.06	838.68
<b>Σ X= 215.7</b>	<b>Σ Y= 220.53</b>	<b>Σ XY= 5096.42</b>	<b>Σ X<sup>2</sup>= 4980.21</b>	<b>Σ Y<sup>2</sup>= 5219.23</b>
<b>n =10</b>				

**Table 25: Correlation computations of functionality**

$$r_{xy} = \frac{10 (5096.42) - (215.7) (220.53)}{\sqrt{|10(4980.21) - (215.7)^2| * |10 (5219.23) - (220.53)^2|}}$$

= 0.99 very high correlation

<b>X: Simulated effectiveness</b>	<b>Y: Real effectiveness</b>	<b>XY</b>	<b>X<sup>2</sup></b>	<b>Y<sup>2</sup></b>
3.68	5.38	19.79	13.54	28.94
4.12	5.52	22.74	16.97	30.47
0	0	0	0	0
0	0	0	0	0
2.18	3.18	6.93	4.75	10.11
3.34	3.54	11.82	11.15	12.53
2.45	3.25	7.96	6	10.56
1.82	2.42	4.40	3.31	5.85
6.5	6.9	44.85	42.25	47.61
2.92	3.92	11.44	8.52	15.36
<b>Σ X= 27.01</b>	<b>Σ Y= 34.11</b>	<b>Σ XY= 129.93</b>	<b>Σ X<sup>2</sup>= 106.49</b>	<b>Σ Y<sup>2</sup>= 161.43</b>
<b>n =10</b>				

**Table 26: Correlation computations of effectiveness**

$$r_{xy} = \frac{10 (129.93) - (27.01) (34.11)}{\sqrt{|10 (106.49) - (27.01)^2| * |10 (161.43) - (34.11)^2|}}$$

= 0.97 very high correlation