

L. Bratthall and C. Wohlin, "Is It Possible to Decorate Graphical Software Design and Architecture Models with Qualitative Information? - An Experiment", IEEE Transactions on Software Engineering, Vol. 28, No. 12, pp. 1181-1193, 2002.

# About this paper

**Title:** Is it Possible to Decorate Graphical Software Design and Architecture Models with Qualitative Information? – An Experiment

**Abstract:** *Software systems evolve over time and it is often difficult to maintain them. One reason for this is often that it is hard to understand the previous release. Further, even if architecture and design models are available and up to date, they primarily represent the functional behaviour of the system. To evaluate whether it is possible to also represent some non-functional aspects, an experiment has been conducted. The objective of the experiment is to evaluate the cognitive suitability of some visual representations that can be used to represent a control relation, software component size and component external and internal complexity. Ten different representations are evaluated in a controlled environment using 35 subjects. The results from the experiment show that it representations with low cognitive accessibility weight can be found. In an example, these representations are used to illustrate some qualities in an SDL block diagram. It is concluded that the incorporation of these representations in architecture and design descriptions is both easy and probably worthwhile. The incorporation of the representations should enhance the understanding of previous releases and hence help software developers in evolving and maintaining complex software systems.*

**Authors and affiliations:** Lars Bratthall, Claes Wohlin.

- Lars Bratthall: Dept. of Informatics, Oslo University, Norway.
- Claes Wohlin: Dept. of Software Eng. & Computer Sci., Blekinge Institute of Technology, Ronneby, Sweden.

**Contact author:** Lars Bratthall. lbr@ifi.uio.no. +47-22840060. Contact address: Lars Bratthall, PO Box 1080 Blindern, N-0316, Oslo, Norway. Fax: +47-22840072

# Is it Possible to Decorate Graphical Software Design and Architecture Models with Qualitative Information? – An Experiment

Lars Bratthall  
Dept. of Informatics, Oslo University  
Box 1080 Blindern, NO 0316 Oslo, Norway  
lbr@ifi.uio.no

Claes Wohlin  
Dept. of Software Eng. & Computer Sci.  
Blekinge Inst. of Technology  
Box 520, SE 372 25 Ronneby, Sweden  
claes.wohlin@bth.se

## Abstract

*Software systems evolve over time and it is often difficult to maintain them. One reason for this is often that it is hard to understand the previous release. Further, even if architecture and design models are available and up to date, they primarily represent the functional behaviour of the system. To evaluate whether it is possible to also represent some non-functional aspects, an experiment has been conducted. The objective of the experiment is to evaluate the cognitive suitability of some visual representations that can be used to represent a control relation, software component size and component external and internal complexity. Ten different representations are evaluated in a controlled environment using 35 subjects. The results from the experiment show that representations with low cognitive accessibility weight can be found. In an example, these representations are used to illustrate some qualities in an SDL block diagram. It is concluded that the incorporation of these representations in architecture and design descriptions is both easy and probably worthwhile. The incorporation of the representations should enhance the understanding of previous releases and hence help software developers in evolving and maintaining complex software systems.*

**Keywords:** software evolution, software maintenance, software quality representation, software quality aspects, software architecture models, software design models

## 1. Introduction

Software systems grow continuously, i.e. new versions of the software are released either to enhance, improve or correct its behaviour. Few software products are released once and then never updated. The

evolution of software systems calls for an understanding of previous versions and releases of the software system. Moreover, it is not possible to rely solely on the understanding of the code. The code is important, but other descriptions are needed to support the understanding of software, for example, architecture and design descriptions. The latter descriptions provide valuable tools on a higher abstraction level than the code. The value of architecture models has been indicated by e.g. [5, 9, 23, 25, 35, 37, 38].

The architecture and design models (e.g. UML class diagrams [35] and SDL block diagrams [24]) are, however, primarily formulated to express actual or estimated functional and static information, rather than information regarding issues that *only* can be gained after having developed the system. In other words, the models have no means of representing experiences from previous releases, for example, the number of user-reported failures. Thus, the models are mainly developed to form a basis for the continuation of the implementation within a project and not between successive projects. We believe, however, that there are simple ways of including experienced quality aspects in architecture and design descriptions. In particular, it should be possible to add information to architecture models during the coding and testing phase of one release to help in the understanding as new versions of the software are to be developed.

Code decay and identification of fault-proneness between releases have gained an increasing interest, for example, [14, 18, 30, 34]. These are important areas to help guide development activities in future releases or identify candidates for reengineering efforts. A problem encountered in these studies is that the information about non-functional aspects of the software has to be found by archival analysis of the software, testing records and problem reports. Four major reasons for fault-proneness are size, relationships, and internal and external complexity. See e.g. [4, 33].

The intention of this paper is to study whether these aspects easily can be included in graphical architecture and design models, and hence help in the understanding of some non-functional aspects between releases. A model is of little value unless it is understood. Once semantics and syntax have been

established, there is a basis for understanding a model. Is this enough or can we increase the *intuitive* understanding of a model, i.e. can we organize a model in such a way that our cognitive processes shaped by prior experience and knowledge help us interpret the model more easily? This is an important aspect, since engineers may prefer the information source with the lowest psychological cost, not the information source with the highest quality [19]. Thus, if a model is perceived as being inaccessible, it is possible that it will never be used.

This paper presents an experiment where the intuitive understanding of various graphic representations for size, relationships, internal and external complexity are evaluated. Intuitive representations of qualities have been suggested before, e.g. in [7, 8]. However, this work is related to cartography and other entities than software. Therefore, there is a value in studying intuitive representation in the context of software systems, with people who have software experience.

The basic hypothesis of this study is that it is possible to decorate or extend existing graphical architecture and design models (such as UML class diagrams and SDL block diagrams) fairly easily to include these four aspects related to quality. By quality, we refer to few experienced faults. By decorating graphical models instead of creating new models, we adhere to Baker's and Eick's proposition that a visualization should adhere to the structure of the software [2], which in this case is represented by an existing architecture model of the software. This differentiates our approach from e.g. [2, 3, 17], where special purpose graphical models are suggested that replace or complement existing models, rather than enhance existing models.

In the experiment different ways of representing four aspects are presented to subjects and the intuitive understanding of the representation is evaluated. From the experiment, it can be concluded that different individuals have the same intuitive understanding of the representations evaluated. This shows that it should be possible to include intuitive representation for these four aspects to help in the understanding of software systems between new versions of the software. An example suggests that it is feasible since it is

easy to add information about these aspects as the software is being released. The information provides then valuable input to forthcoming versions of the software.

This paper is organized as follows. Section 2 defines research questions, variables and hypothesis. Section 3 discusses the methods used to conduct the experiment. Section 4 presents an analysis of collected data. Section 5 shows some possible applications of the results, through applying the results of the experiment on an industrially operational car control software system. Finally, section 7 summarizes the paper.

## 2. Experiment definition

This experiment analyses how software developers can be supported during evolution of a software system. The objective of the experiment is to evaluate whether some relationships and quality related issues can be represented easily and intuitively in graphical software architecture or design models. Inspired by e.g. [13, 16], it is believed that different two-dimensional visual representations better express some software engineering concepts in terms of how intuitive the mapping between the representation and the concept is. The representation that is the most intuitive representation of a software engineering concept is said to have the minimum accessibility weight. For example, in Figure 1 there are two symbols, that both represent that smoking is not allowed. The left symbol is a more intuitive representation of the concept. Out of the two symbols, the left symbol can be said to have the minimum accessibility weight - no previous training or information is needed to understand what the symbol represents. By ensuring minimum accessibility weight in models, the need for training and manuals is minimized.



Fig. 1. Two symbols that represent the concept of no smoking.

Four research questions are formulated:

RQ1 How do we best show that component A controls the operation of component B? That a Component Controls another component is referred to as the ‘CC’ relationship.

RQ2 How do we best show that it is more complex to modify the internals of component A than it is to modify the internals of component B? This is referred to as the Internal Complexity relationship (‘IC’).

RQ3 How do we best show that the externally visible interface of component A is more complex than the externally visible interface of component B? This is referred to as the External Complexity relationship (‘EC’).

RQ4 How do we best show that the size of component A is larger than the size of component B? This is referred to as the Component Size relationship (‘CS’).

All of the research questions refer to an intuitive mapping between a concept used within software engineering and a visual representation. This paper does not take a stand on how the concepts should be measured, only how they should be represented. For example, internal complexity can be measured using some code metric such as cyclomatic complexity, or some process metric such as the number of faults reported for a component.

There are a large number of classes of spatial concepts used for visual representation, e.g. [20] lists 51 spatial concepts and indicates that there are many more. This experiment investigates only a few visual representations, that have been chosen because they share three properties: i) They are suitable both for paper and screen representation; ii) They can possibly be combined with many existing graphical architecture and design models, such as UML class diagrams and SDL block diagrams; and iii) The concepts can be used concurrently to decorate an existing diagram with several new properties. The representations studied for the CC relationship are shown in Figure 2. Representations 1 to 4 are assessed as good candidates for the CC relationship, i.e. possible indications that component A controls component B.

Representations 5 to 10, that are studied for the EC, IC and CS relations are shown in Figure 3. For

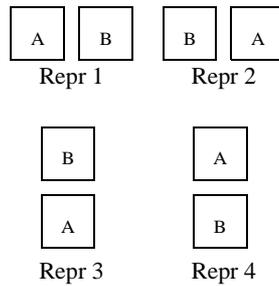


Fig. 2. Objects studied for the CC relationship

example, it is evaluated whether representation 6 is a better indicator of the fact that component A is larger (the CS relationship) than component B, than, for example, representation 9. All representations studied avoid unnecessary visual cluttering. Thus it is adhered to Tufte’s advice that “[A] large share of ink on a graphic should present data-information, the ink changing as the data change.” [42].

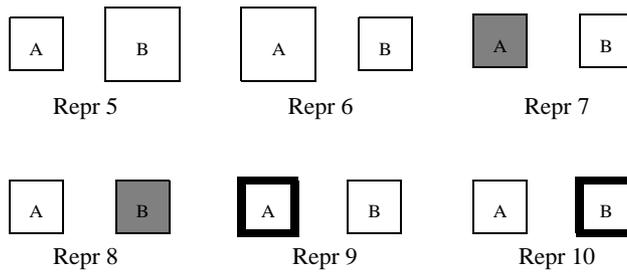


Fig. 3. Objects studied for the IC, EC and CS relationships

From the research questions, a number of hypotheses are derived. These are described in Table 1. The hypotheses all indicate that we believe that there is a representation that is better than the other ones.

Table 1. Hypotheses. All hypotheses are assessed at the  $p = 0,05$  significance level.  $m, n \in \{1, 2, 3, 4\}$ ,  $m \neq n$ ,  $p, q \in \{5, 6, 7, 8, 9, 10\}$ ,  $p \neq q$

RQ	Name	Definition
RQ1	$H_{CC,1}$	$Avg_{CC, Repr\ n} \neq Avg_{CC, Repr\ m}$ There is some representation $Repr_n$ that has minimum accessibility weight in terms of how well it represents that software component A controls software component B.
	$H_{CC,0}$	$Avg_{CC, Repr\ n} = Avg_{CC, Repr\ m}$ Null hypothesis: There is no statistical difference between the representations that may represent that software component A controls software component B. Explanation of variables: The average weight assigned to representation n regarding how well it represents the “A controls B” relationship (CC) is equal to the average weight assigned to representation m for the same relationship.
RQ2	$H_{IC,1}$	$Avg_{IC, Repr\ p} \neq Avg_{IC, Repr\ q}$ There is some representation $Repr_m$ that has minimum accessibility weight in terms of how well it represents that software component A is internally more complex than software component B. Null hypothesis: $H_{IC,0}$ .
RQ3	$H_{EC,1}$	$Avg_{EC, Repr\ p} \neq Avg_{EC, Repr\ q}$ There is some representation $Repr_m$ that has minimum accessibility weight in terms of how well it represents that software component A is internally more complex than software component B. Null hypothesis: $H_{EC,0}$ .
RQ4	$H_{CS,1}$	$Avg_{CS, Repr\ p} \neq Avg_{CS, Repr\ q}$ There is some representation $Repr_m$ that has minimum accessibility weight in terms of how well it represents that software component A is larger than software component B. Null hypothesis: $H_{CS,0}$ .

### 3. Method

#### 3.1 Introduction

The strategy used to answer the research questions is an experiment in a laboratory environment [44]. For data collection, questionnaires have been used. The method used for data collection and analysis is summarized in Figure 4 and described further below.

#### 3.2 Sampling and generalization

From [29] it is known that entry-level programmers work differently from more experienced

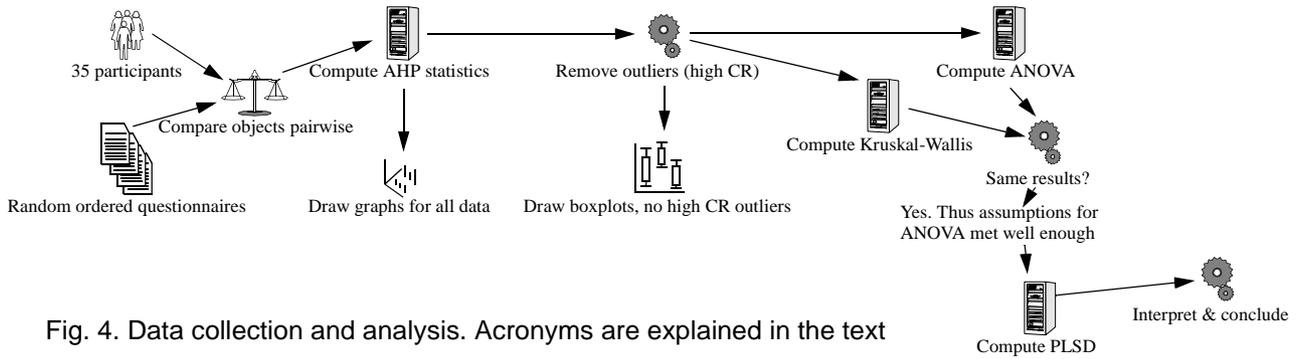


Fig. 4. Data collection and analysis. Acronyms are explained in the text

programmers. In particular, experienced programmers are able to concurrently draw on more sources of information such as different models. This experiment studies how entry-level programmers and part-time programmers can be aided. 35 subjects participated in the experiment. 26 of them are master students in computer science and electrical engineering. Nine have PhDs or are PhD candidates. ANOVA tests comparing the two types of participants show no significant difference in the results from the two types of participants. Therefore, all participants are treated as one homogeneous group, despite the participants' different exposure to various tools, technologies and methodologies. All participants have had some exposure to object-oriented methods prior to the experiment. We believe that the way the objects studied are perceived is not changed much by the increased time pressure and competitiveness that may exist in industry as compared to a student/university environment. Given the background of the participants, it should be possible to generalize to at least entry level programmers, and probably also to somewhat more experienced designers as there is no significant difference in the results between students and postdocs.

### 3.3 Data collection

Data collection has taken place in a laboratory setting. Data collection took place at three different occasions. The same introduction to the experiment was given to all participants. This introduction made sure that all participants knew how to fill in the questionnaires. It was also made sure that all participants knew that they were not to think more than 15 seconds about each question. This was to ensure that the initial understanding of each question was captured. No hypotheses regarding the outcome of the

experiment were presented to the participants.

Each of the research questions were addressed by a questionnaire. For each of these four questionnaires labelled RQ1, RQ2, RQ3 and RQ4, three different versions exist (A, B and C). The ordering of questions in each version was different, but the set of questions was equal across all versions. Participants were assigned one of each questionnaire in random order. The version used of each questionnaire was also randomly assigned. Two possible sequences of questionnaires are i) {RQ1:vA, RQ3:vC, RQ2:vA, RQ4:vA}; and ii) {RQ4:vA, RQ1:vC, RQ3:vB, RQ2:vB}. Thus, the order of questions answered is random for each participant, which should reduce the threat that the ordering of questions affect the outcome of the experiment.

Each question was applied on a number of pair-wise comparisons. For example, RQ1 was applied to six comparisons between representations Repr 1 to Repr 4, as the number of possible pair-wise comparisons is  $n(n-1)/2$  when  $n$  is the number of representations to be compared. Table 2 is an example of a single comparison for RQ1. To the left, there is one representation (here: Repr 1), and it is to be compared to Repr 2 (to the right). In between, there is a graphical scale with nine alternatives where it is possible for the participants to answer by checking the appropriate box. By checking the middle box, it is indicated that none of the representations is better. By checking the left-most box, the participant would have indicated that “Repr 1 is extremely better than Repr 2 when used to represent that component A controls component B”.

Table 2. Example of a single comparison

Which of representations 1 and 2 do you think better represents that component A controls component B?		
Repr 1		Repr 2
		

### 3.4 Data refinement

Data collected through the questionnaires is initially refined by using the Analytical Hierarchical Process (AHP) [27, 36]. The AHP was developed to improve decision-making through giving priority to items pair-wise, rather than prioritizing all items at once. Thus the method simplifies complex decision-making. The output from an AHP analysis is a value denoting the relative weight of items compared. In this paper, the AHP is used to compute the relative accessibility weight of the representations in Figure 2 for the CC relationship and those in Figure 3 for the IC, EC and CS relationships.

The original AHP method is intended to be used by one person or one group of people, who for every comparison decide one common answer based on consensus. In [15] it is shown how to instead aggregate the result of individual comparisons after the comparisons have been made. This procedure has been used in this study.

The AHP has an additional feature. Using the AHP, it is possible to objectively measure how consistent answers from a participant are. For example, if a participant claims that Repr 1 is a better representation of the CC relationship than Repr 2, and that Repr 2 is a better representation than Repr 3, and that Repr 1 is a much better representation than Repr 3, there is high consistency in the answers. On the other hand, if the participant claims that Repr 1 is a better representation than Repr 2 for the CC relationship, and that Repr 2 is a better representation than Repr 3, then answers are not consistent if the participant also

considers Repr 3 to be a better representation than Repr 1. This example of consistent and inconsistent answers is illustrated in Figure 5. This Figure uses the same nine point scale as Table 2.

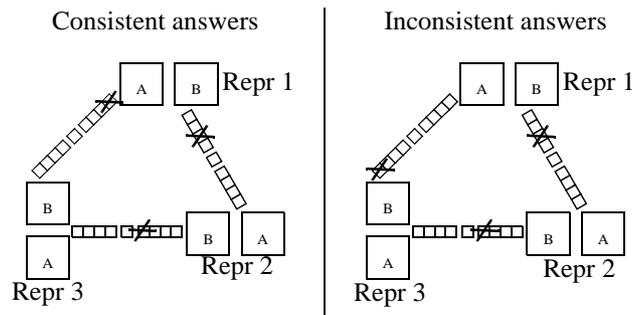


Fig. 5. Example of consistent and inconsistent answers

The consistency is measured by computing a consistency index (CI) and adjusting it according to the number of different items compared, to an adjusted index called the consistency ratio (CR). By looking at the CR, it is possible to objectively identify respondents who have either not put a lot of thought into the answers, or have not understood the questions, or have not enough knowledge to compare the items to prioritize. A low CR value is a sign of high consistency in answers given. As a rule of thumb, answers with a  $CR > 0.40$  are treated as outliers. For details in how to compute the relative weight and CI and CR, see e.g. [27]. In the software engineering field of research, the AHP has also been used, for example, to prioritize requirements [27], to estimate size of simple software modules [31], to measure organizational beliefs [26] and to estimate what consumes lead-time in the development and evolution of distributed real-time systems [10].

### 3.5 Statistical analysis

As shown in Figure 4, a number of statistical tests have been used<sup>1</sup> on the output of the AHP, notably the Kruskal-Wallis tests [28], Analysis of Variance (ANOVA) [32], and Protected Least Significant Difference tests (PLSD) [32]. The Kruskal-Wallis test is non-parametric and does not make any assumptions about the distribution or variance of data collected. This test is used to initially get a picture

1. SPSS 8.0 running on Windows NT has been used for all tests except the AHP.

of the data analysed. The ANOVA makes two assumptions: The variance in the groups being compared must be equal, and the data must be distributed according to a normal distribution. If both a Kruskal-Wallis test and an ANOVA show the same result, PLSD tests have been used as it is assumed that the preconditions for the ANOVA have been met well enough. The Kruskal-Wallis test and the ANOVA have been used to reject/support the hypotheses in Table 1. PLSD tests have been used to identify what contributes the most to the results of an ANOVA, i.e. to identify exactly which representation is considered the best. A PLSD is similar to a t-test, in that it identifies if there is a statistically significant difference between two sets of data.

### 3.6 Threats and validity

The validity of the findings reported depends to a large extent on how well threats have been handled. Four types of validity threats [12] are analysed: Conclusion validity, construct validity, internal validity and external validity. The relationship between these is illustrated in Figure 6.

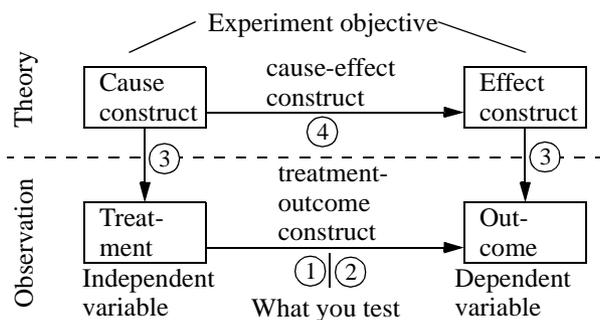


Fig. 6. Experiment principles as described by [44]

Conclusion validity (marked 1 in Figure 6) concerns the relationship between the treatment and the outcome. Care has been taken not to violate any assumptions made by the statistical tests. The questions used in the questionnaires have been easy to answer, and the participants were trained in advance in how to fill in the questionnaires. Therefore, the reliability of the measures is high. As the data collection took place at three different occasions, there has been a risk that the reliability of the implementation of the experiment could be degraded. This threat has been addressed by using a highly standardized experiment

introduction. By collecting data at different occasions, random irrelevancies in the experimental setting have been accounted for. The set of experiment participants is fairly homogenous, because of their common experiences in courses studied at the university. Additionally, there is no statistically significant difference between the student participants and the graduate participants. Therefore, the present random homogeneity (or heterogeneity) of participants should not affect the conclusion validity much.

Internal validity (2) concerns matters that may affect an independent variable's causality, without the knowledge of the researcher. History effects, i.e. how previous events affect participants, have been taken into account in several ways. First, there is some heterogeneity in the group of respondents. This balances long-term history effects. Secondly, the data collection has taken place at three different occasions. This accounts for effects of recent events. Thirdly, respondents answered questions in different order by using different versions of four questionnaires answered in random order. The latter ensures that history effects from the last few questions are accounted for. As data collection took no more than an hour, there should be no maturation effects. Testing effects have been minimized by making sure that there was no gain or loss in how questions were answered. No participants left the experiment, i.e. there has been no mortality.

Construct validity (3) concerns whether we measure what we believe we measure. Constructs used in the questionnaires have been well defined. The use of the scale used to compare objects has been explained to all participants. The use of the AHP and using the consistency ratio to identify outliers is a way of reducing the mono-method bias by making each participant compare each object studied to all other objects. Thus, the results from each respondent are not dependent on a single answer. There is always a risk that respondents guess the hypotheses at hand. By making sure questions are answered in different order, we balance both the threat of hypotheses guessing as well as the threat of interaction between different questions and objects.

External validity (4) concerns generalization of the findings to other contexts and environments than the one studied. All participants have been active at a university, either in teaching or studying. The

teachers have varying degree of industrial background. First, there is no statistically significant difference between the teachers and the students. Secondly, which may be more important for the external validity: Students leave the university for industry. Student participants are close to the end of their university education (M.Sc. level). It is believed that possible changes in competitiveness and time pressure that may be different in industry and a university setting does not affect the perception of the representations studied. Therefore, results should be valid at least for the first few years in industry. The difference between students and professionals as subjects in terms of perception of aspects that influence lead-time in software projects have been studied without finding any statistical significant difference [22]. This does not necessarily mean that there is not a difference in other studies, but it suggests that the difference may not be as large as one may anticipate. Recently, the use of students as subjects have also been discussed as part of the review process of empirical papers [40]. Comparative studies as the one presented here is a situation where the use of students is most easily motivated.

Another threat is that the experiment compares the objects in Figure 2 and Figure 3 in isolation, not in complex combinations such as shown in e.g. Figure 16. It is possible that this affects the outcome as perception is a function of “the entire visual field” [1]. Though further study is needed to verify that the understanding does not change in complex combinations, such as when the representations suggested are applied to e.g. UML class diagrams or SDL block diagrams, a limited experiment described in Section 6 suggests that understanding does not change in at least one complex diagram.

A threat that is of concern only to the representation of the CS relationship, is that we have used area to represent component size. Area is two-dimensional, while component size usually is one-dimensional (e.g. lines of code, number of bytes). This can possibly cause misinterpretation [42]. A threat related to the understanding of software, is the wide range of comprehension strategies used by different individuals [29, 39]. The objective of this study is not to generate a silver-bullet solution, rather a particular set of comprehension issues are investigated. The biggest threat to the external validity of this

study that we can think of, is the prior exposure by other populations to particular mappings between the software engineering concepts and the visual representations studied. This could possibly be the case if some tools have been used before.

#### 4. Results and analysis

In this section, results from the experiment are presented. First, the analysis for the CC relationship is presented in some detail in Section 4.1, to explain how analysis has taken place. The other relationships are briefly analysed in Section 4.2. In Table 3, results for all four relationships are summarized. Finally, in Figure 11 all results are graphically summarized.

Table 3. Summary of analysis for the four relationships studied

	CC	IC	EC	CS
Representations compared	1, 2, 3, 4	5, 6, 7, 8, 9, 10	5, 6, 7, 8, 9, 10	5, 6, 7, 8, 9, 10
Boxplots of all data	Figure 7	Figure 8	Figure 9	Figure 10
Outliers due to CR>40, participant(s) no.	None	2, 20	9	2
$H_x$ rejected by a Kruskal Wallis at a 0.05 significance level	$H_{CC,0}$ rejected	$H_{IC,0}$ rejected	$H_{EC,0}$ rejected	$H_{CS,0}$ rejected
$H_x$ rejected by an ANOVA test at a 0.05 significance level	$H_{CC,0}$ rejected	$H_{IC,0}$ rejected	$H_{EC,0}$ rejected	$H_{CS,0}$ rejected
Representation with best average minimum accessibility weight	Repr 4	Repr 7	Repr 9	Repr 6
Representation with next best minimum accessibility weight	Repr 1	Repr 9	Repr 7	Repr 9
PLSD indicates difference at 0.05 significance level between the two best representations	Yes	No	Yes	Yes
Suggested mapping with minimum accessibility weight	Repr 4	Repr 7 <sup>a</sup>	Repr 9	Repr 6
Answers research question	RQ1	RQ2	RQ3	RQ4

a. Representation 9 is significantly better at representing the EC relationship than representation 7. Therefore, representation 9 is chosen for the EC relationship. Representations 7 and 9 best represent the IC relationship. There is no statistically significant difference between representations 7 and 9 for the IC relationship, but since representation 9 already is accounted for, and representation 7 is the best on average, it is chosen to represent the IC relationship.

#### 4.1 Detailed analysis of the CC relationship

$H_{CC,1}$  is tested. The participants were asked (original grammatical error) “Does object 1 or object 2 indicate the stronger the fact that software component A controls software component B?”. The objects referred to are representations 1, 2, 3, and 4 in Figure 2. The individual participants’ results and a boxplot are shown in Figure 7. From Figure 7, it is seen that representation 4 seems to be the best and that the next best is representation 1 as they have the highest result values (i.e. minimum accessibility weight). Thus, the next step is to evaluate if the differences seen in Figure 7 are statistically significant. At this stage, it is also noted that no participants had a  $CR > 0.40$ , so there are no outliers.

Both a Kruskal-Wallis test and an ANOVA test show that there is a statistically significant difference at the 0.005 level in how designers rank the accessibility-weight of the four objects studied.

Repr4 is on average the best representation of the CC relationship. Since an PLSD shows that  $Avg_{CC,Repr4}$  is statistically different at the 0.005 level from the next best representation of the CC relationship, Repr1,  $H_{CC,0}$  can be rejected. The conclusion is that Repr4 is the representation of those studied that represents the controlling/controlled relationship with the smallest accessibility weight. This answers RQ1 and addresses the  $H_{CC}$  hypotheses.

#### 4.2 Analysis of the IC, EC and CS relationships

This analysis and the corresponding analysis for the other relationships are summarized in Table 3. In addition, box plots for the other three hypotheses are shown in Figures 8-10. The main result is that with

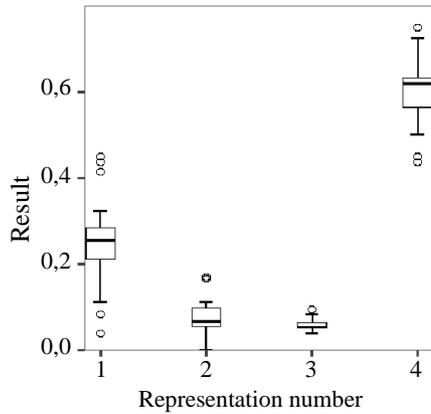


Fig. 7. Boxplot for the CC relationship

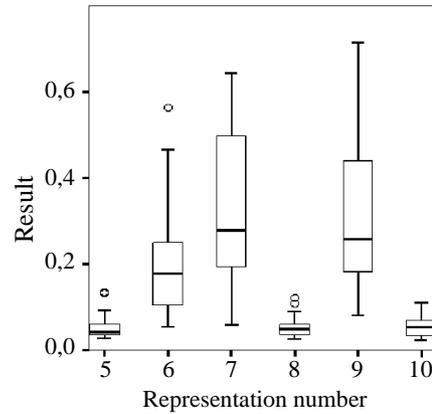


Fig. 8. Boxplot for the IC relationship

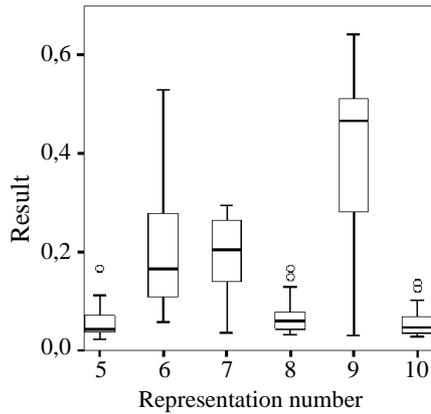


Fig. 9. Boxplot for the EC relationship

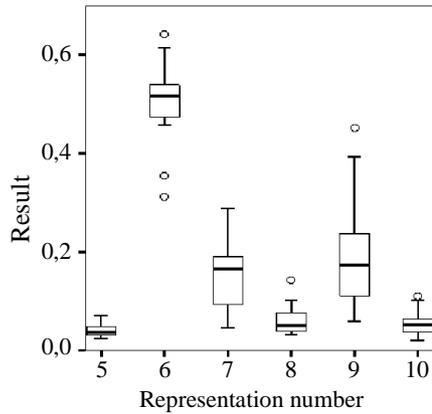


Fig. 10. Boxplot for the CS

statistical significance, the representations with minimum accessibility weight can be determined. The CC relationship is best represented by Repr 4, the IC relationship by Repr 7, the EC relationship by Repr 9 and the CS relationship by Repr 6.

These representations, with minimum accessibility weight, are illustrated in Figure 11. There is also a simple example of how the properties can be combined together. A larger example is provided in Section 5.

## 5. Application of results

The study in Section 4 has been conducted without any specific graphical notation in mind and hence the results have to be studied in more detail for each specific notation. Moreover, for a specific notation, other graphical representations may also be of interest.

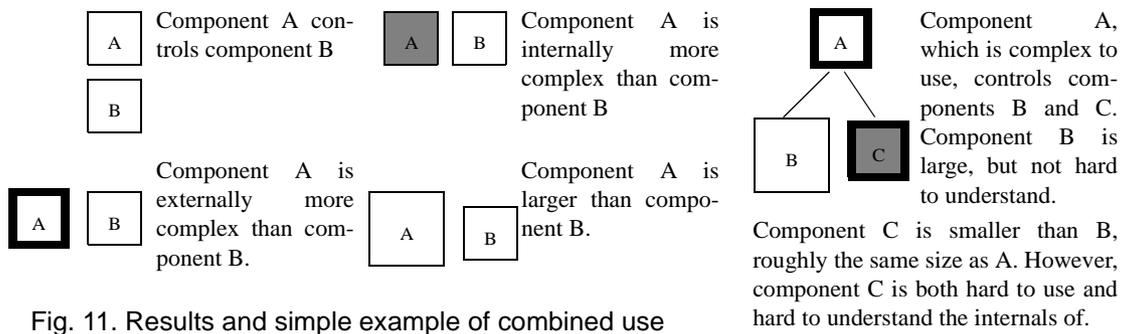


Fig. 11. Results and simple example of combined use

To illustrate how the results possibly can be applied in a more complex environment, with a specific notation, a scenario is presented below. As noted in Section 3.6, more research is needed to fully establish that the representations identified in the experiment also are valid in this more complex environment. In the scenario, we have applied the approach from the experiment to SDL block diagrams, where communication paths have been removed for clarity of figures. SDL is a formally defined language [24]. It is frequently used in embedded software, such as in the telecom domain, in medical equipment and in the car industry. The software illustrated is a part of an industrial currently operational car control software system. The relationships illustrated are the Component Size and Component Internal/External Complexity Relationships (CS, IC, EC). Since the approach is applied on *existing* software, the Component Controls relationship (CC) is not applied, since it is believed that this is best modelled at design time.

A Software Development Team (SDT) is given a static architecture level diagram (Figure 12). The system consists of two main components, that in their turn are aggregated from other components. The semantics of the aggregation relationship is formally defined by SDL [24]. The CarLogic software component is a representation of the car’s behaviour. The CarTestDriver is a software component that is used to test the CarLogic, without having to execute the software in an actual car. In fact, the road conditions, the driving context as well as the driver’s behaviour can be simulated. During simulation, the TestRecorder component monitors all actions as well as how the car reacts to each stimuli. Thus the behaviour of the car can be monitored and analysed also in potentially dangerous situations. The

CarTestDriver component is not a part of the car on-board embedded system, which is created by adding some hardware interface components to the CarLogic component.

The SDT's task is to extend both the CarLogic and the CarTestDriver with software that allows the software to handle two new video-cameras, that should be used to adjust the speed of the car in case there is a foreign object in front of the car. Examples of foreign objects are humans and other vehicles. This task has been selected to be used in this example because it represents the extension of the car control system with a feature described in a commercial patents database.

The SDT does not have any previous knowledge of the software, and starts by planning what should be done by looking at an architecture model of the system (Figure 12).

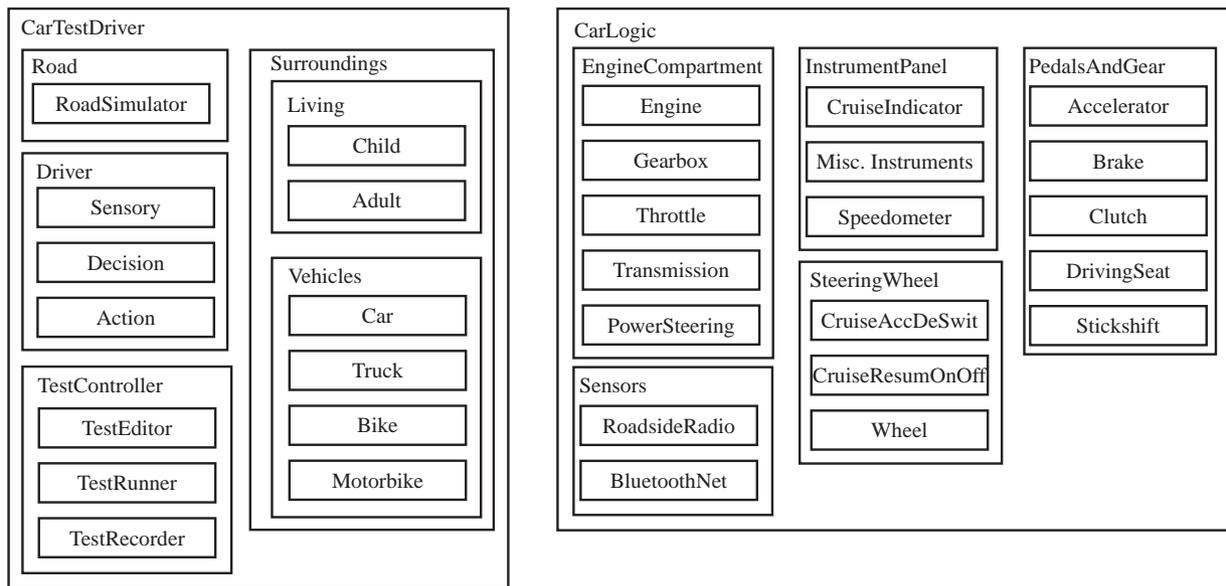


Fig. 12. Architecture of a part of the car control software

Seeing this model, the SDT draws a number of immediate conclusions. For example, the Sensors component in the CarLogic component should be expanded with camera functionality, the InstrumentPanel should be changed to accommodate a new switch that determines if the automatic speed adjustment is engaged or not. The SDT decides that this switch should be handled by the Misc. Instruments component. Also, there should be changes in the Surroundings component in the CarTestDriver to allow images to be constructed and fed to a new video analysis component in the

CarLogic component. There should also be changes to the Throttle component to handle new speed adjustment messages coming from a new component that analyses video images.

However, without prior experience of the car control system, it is hard for the SDT to estimate for example change effort for each component, and where the largest risks for faults are. This is important to know, since a small number of components are likely to account for a large amount of externally visible faults [4, 21]. Therefore, the SDT invokes the understandability mode of their design tool. In this example scenario, the number of problem reports and fixes for a software component is used to indicate internal complexity. These figures have been found to be a good indicator of component internal complexity in the SDT’s organization. The understandability mode is illustrated in Figure 13, which is an illustration of how the visual representation for the IC relationship identified in the experiment in this paper is applied in an SDL block diagram to leaf blocks. Shaded components are considered internally more complex, than not shaded components.

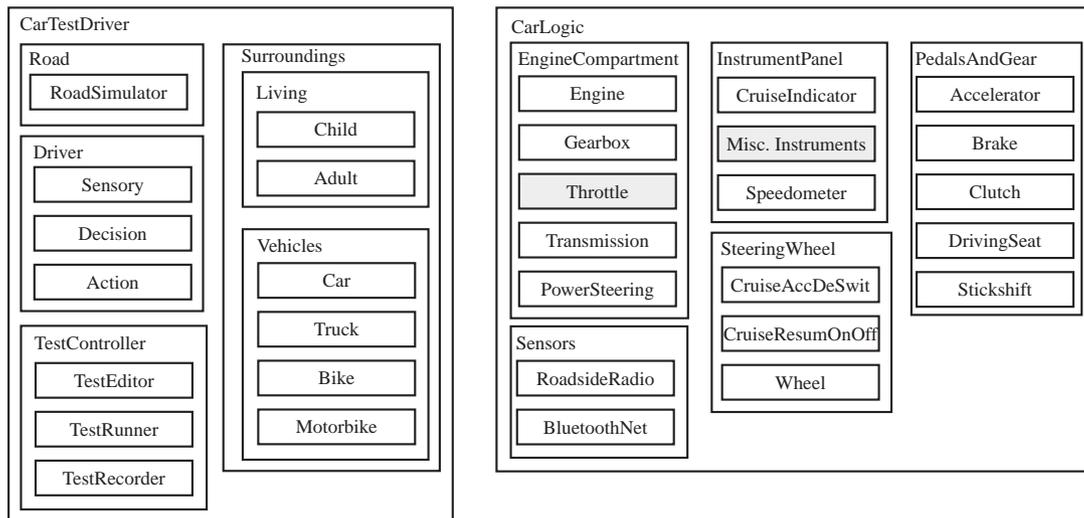


Fig. 13. Car control software, number of problem reports indicates internal complexity

Using the information in Figure 13, the SDT makes a number of new decisions. First, seeing that the Throttle component is fault-prone, the SDT decides to allocate extra time to the comprehension of this component, to avoid introducing new faults. Faults in software in embedded car systems are very expensive to fix, since on-board software usually is run on hardware that is physically glued to other parts

to handle the vibrations in a car. Therefore any software update can involve the costly replacement of a larger physical component.

Secondly, the component Misc. Instruments is prone to faults. Since this component has been identified as being likely to change due to the addition of the new functionality, the SDT changes the earlier decision and instead decides that the new instruments should not be allocated to the Misc. Instruments component, but to a new component: AutoSpeedAdjustIndicator.

Next, the SDT needs indications of the size of the components currently in place, in order to estimate the time it takes to understand each component. Size-related metrics has been identified as being related to effort. Therefore, the size-view is invoked as illustrated in Figure 13.

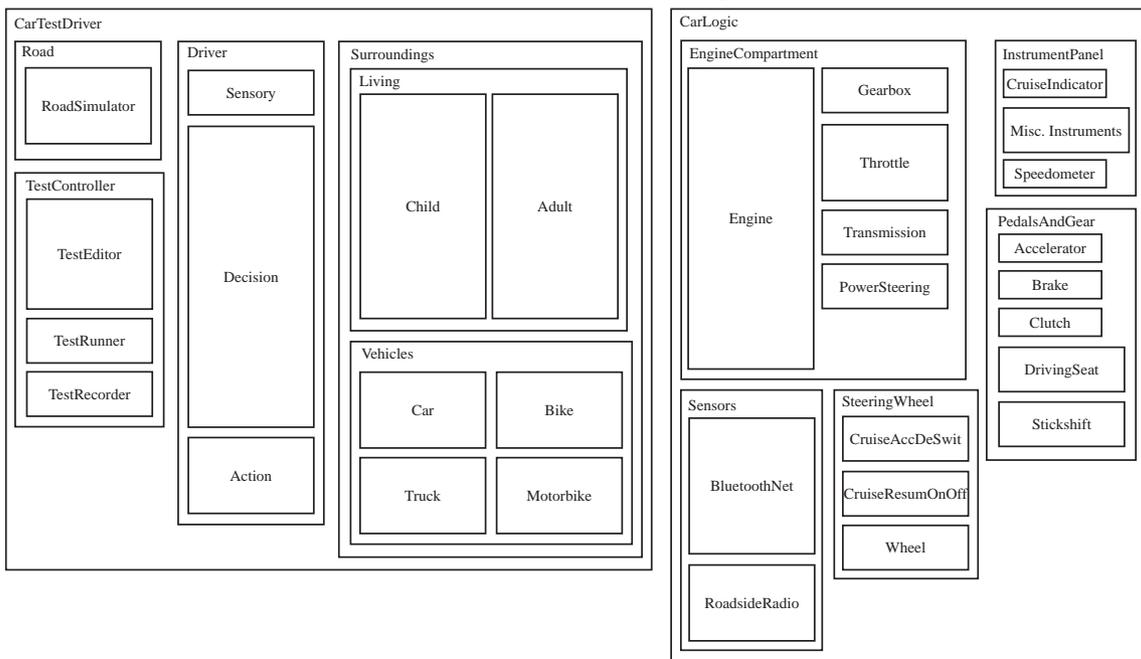


Fig. 14. Car control software, size view

The size view illustrates the source code volume of leaf components. The volume is important to know for two purposes: First, volume is a crucial factor in embedded systems. In a car, every single kilobyte of memory counts due to that the unit cost has a large impact on the total cost. Secondly, the SDT can use the information to adjust the standard development process: Extra time is assigned to understand and modify the Throttle component, since it is a relatively large component of those that are likely to change.

Next, it is known that effort is dependent on size measures that are affected by externally visible size measures such as the number of methods. Therefore, the system is shown with indications for externally visible complexity in Figure 15. Here the number of externally visible methods are used as an indicator of externally visible complexity.

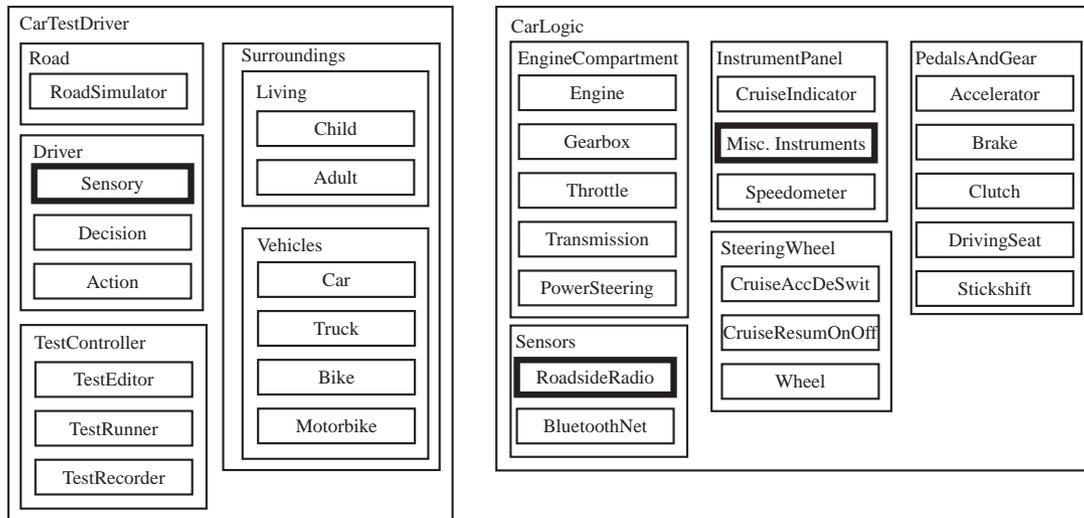


Fig. 15. Car control software, externally visible complexity

From this model, it is seen that the Misc. Instruments component is complex to use. Remembering that it was also fault-prone, the SDT changes its decision regarding this component: Instead of just adding a new component for the automatic speed adjustment instruments, the SDT decides to split the Misc. Instruments component into several components – one for each instrument.

Finally, the system is viewed with a concurrent combination of quality aspects represented, and an early estimation of the new parts are plotted as well. This is illustrated in Figure 16. Notice how fast it is possible to distinguish which components are large, which are internally complex, and which are externally complex. This is a sign of that the representations suggested have high selectivity [7], even when used in cluttered contexts. This is in line with [41], who found that simple shapes and colours can be as quickly found when embedded in a distracting environment, as when the environment is less distracting.

Figure 16 constitutes the base for further adjustments to the development process and the allocation of

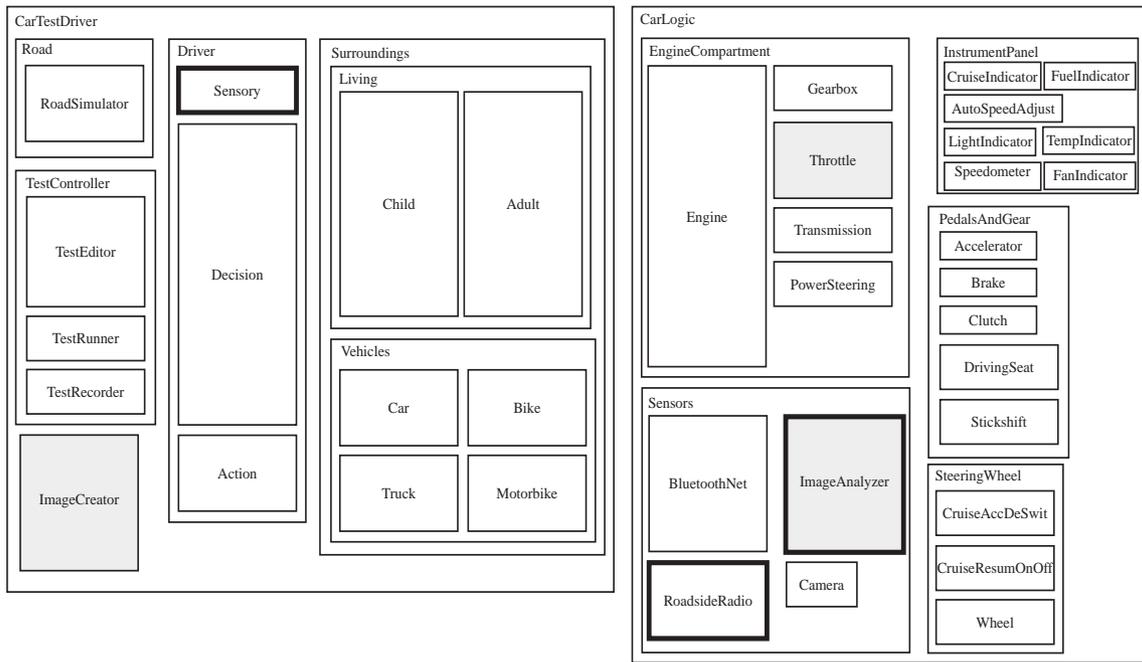


Fig. 16. Car control software, complete architecture including new components

developers. The SDT decides that their most experienced developers should be assigned to work mainly with the Image Analyser and the Image Creator components, due to their estimated internal complexity. The Image Analyser is expected to be complex to use, so extra effort is allocated to producing diagrams that help understand how to use the component. The new Camera component is estimated to be small and easy to design, so junior developers are assigned to work on this component.

## 6. Scenario validation

The scenario presented in Section 5 is one example of how the visualization technique proposed in this study can be used to help shape a development organization and its development process to produce higher quality software. The example system is a part of an industrial system and the change request, i.e. adding video-cameras to control speed relative objects in front of the car, is fetched from a commercial patents database. Thus it is ensured that the example scenario is industrially relevant.

In Section 3.6, it was identified that the simplicity of the experiment could be a threat to the validity of the results in more complex settings. In order to further establish trust in the findings, one more experiment has been conducted: Ten graduate software professionals were first introduced to a short text

explaining that in a graphical architecture model that they were about to see, there were representations of software leaf component size, internal complexity and external complexity. None of these participants had seen the representations discussed before. None of them had even heard about the first experiment, and none of them had any SDL experience. Thus the participants did not know how these qualities were represented.

After reading the short introduction text, the participants were given a copy of Figure 16. The participants were asked, in random order, to identify an example of a) one of the largest/smallest leaf components; b) one of the internally most/least complex leaf components; and c) one of the externally most/least complex leaf components. That is, in total each participant was asked to identify six leaf components.

All ten participants correctly identified examples of the largest/smallest leaf components. Seven participants correctly identified examples of all six types of components. The remaining three participants each identified one, two and four components incorrectly. That is, in total there were 53 correct identifications out of 60 possible, despite the brief explanation of the model. The single identification error was, according to the participant, a pure mistake. The other two participants explained that they had attempted to consider how they would have constructed the system in Figure 16. Instead of drawing conclusions from the model, they used their own experience to identify components that they believed would be internally/externally most/least complex.

Though this second experiment is very limited (e.g. only one diagram has been investigated, there are rather few participants and the participants did not know about the semantics of SDL block diagrams), it is believed that it strengthens the idea that the representations suggested are indeed intuitive, at least when applied to SDL block diagrams.

## **7. Summary**

The presented experiment is formulated based on the belief that it should be possible to represent some

software quality aspects clearly in graphical architecture and design models. For example, to the best of our knowledge, no UML [35] nor SDL [24] diagrams contain a graphical representation for e.g. component size and component external/internal complexity. Thus, the objective of the experiment was to evaluate whether it would be possible to augment some existing architecture and design models with this type of information, in an intuitive way.

Ten different representations have been evaluated independently of UML/SDL diagrams, and the results from the experiment have shown with statistical significance, in most cases, that it is indeed possible to intuitively represent non-functional aspects in a way that can decorate many UML/SDL architecture and design models. Exact rules for mapping complexity measures to decoration must be established for each model through future research. However, it is believed that the evaluated representations can easily be used to decorate existing graphical architecture and design models, which differ from for example [2, 3, 11, 17] that use special purpose models.

The main value of the results is in software evolution. Representations of the evaluated types can support architecture and design understanding during evolution and maintenance. In addition to the experiment, it has been illustrated through an example how tool support can be enhanced to include representations of non-functional aspects. For example, [2] and [3] use component area to represent component size. The findings in this study strengthen the idea that this is a sensible representation.

The representations in this study have a few weaknesses.

- Gray-scale is used to represent internal complexity. Color was avoided to decrease print costs, despite that color can be better at conveying value information [43]. As color printing becomes less costly, replacing the gray-scale with color representation may be beneficial.
- Human perception of gray differs depending on the amount of black and white surrounding the gray area [6]. Thus there is room for erroneous perception if the same internal complexity (same amount of gray) is used in one component with low external complexity (thin black border), and another with

high external complexity (thick black border). However, component size (area) should not affect the perception of the gray-scale [7].

It is our belief that these deficiencies are acceptable, given that the representations are used to decorate existing graphical design models.

In summary, the experiment indicates that representations can be found that can be used when software is evolving and we would like to stay in control of the evolution. The representations suggested can be viewed as warnings signs together with, for example, methods for classifying software components. Further work includes comparing the representations suggested by this study in more complex environments as well as the use in large scale software systems.

### References

- [1] Asch, S.E. *Social Psychology*. Prentice-Hall, USA. 1952. Reprinted, Cambridge University Press. 1987
- [2] Baker, M.J., and Eick, S.G. "Space-Filling Software Visualization", *Journal of Visual Languages and Computing* 6(2), p. 119-33. June 1995.
- [3] Ball, T.J., and Eick, S.G. "Software Visualization in the Large", *IEEE Computer* 29(4), p. 33-43, Apr. 1996.
- [4] Basili, V.R., and Perricone, B.T. "Software Errors and Complexity: An Empirical Investigation", *Comm. of the ACM*. Vol. 27, No. 1, p. 42-52, January 1984.
- [5] Bass, L., Clements, P., Kazman, R. *Software Architecture in Practise*. Addison-Wesley, 1998
- [6] Benary, W. "Boebachtung zu einem Experiment uber Helligkeitskontrast". *Psychologische Forschung*, 5, 131-142. 1924. Reprinted in Ellis, W. (Ed.) *A Source book of Gestalt Psychology*. Selection 8, The Humanities Press. 1950.
- [7] Bertin, J. *Semiology of Graphics*. The University of Wisconsin Press, 1983. Translation of Bertin, J.: *Sémiologie graphique*. 1967

- [8] Bertin, J. *Graphics and Graphic Information Processing*. Walter de Gruyter, Berlin, Germany, 1981.  
Translation of Bertin, J.: *La Graphique et le Traitement Graphique de l'Information*. 1977
- [9] Bosch, J. *Design & Use of Software Architectures: Adopting and Evolving a Product-line Approach*. ACM Press/Addison Wesley. 2000
- [10] Bratthall, L., Runeson, P., Adelswärd, K., Eriksson, W. "Lead-time Challenges in the Development and Evolution of Distributed Real-time Systems". *Information Systems and Technology*. Vol. 42, No. 13, pp. 947-58. Sep., 2000.
- [11] Chuah, M.C., and Eick, S.G. "Glyphs for Software Visualization". 5th Int'l Workshop on Program Comprehension, p. 183-91, IEEE Computer Society Press, Dearborn, Michigan, USA, May 1997.
- [12] Cook, T.D., and Campbell, D.T. *Quasi-Experimentation - Design and Analysis Issues for Field Settings*, Houghton Mifflin Company, 1979.
- [13] Egenhofer, M. J., and Mark, D.M. "Naïve Geography", In Proc. Spatial Information Theory: A Theoretical Basis for GIS COSIT'95. LNCS 988, p. 1-15. Springer Verlag, Germany. 1995.
- [14] Eick, S.G., Graves, T.L., Karr, A.F., Marron, J.S., Mockus, A. "Does Code Decay? Assessing the Evidence from Change Management Data". *IEEE Transactions on Software Engineering*, Vol. 27, No. 1, pp. 1-12. January, 2001.
- [15] Forman, E., Peniwati, K., "Aggregating Individual Judgements and Priorities with the Analytic Hierarchy Process". *European Journal of Operational Research*, 108, p. 165-9. 1998.
- [16] Freksa, C. "Spatial and Temporal Structures in Cognitive Processes", *Foundations of Computer Science: Potential - Theory - Cognition*. LNCS 1337. Springer Verlag, Germany, 1997.
- [17] Gall, H., Jazayeri, M., Riva, C. "Visualizing Software Release Histories: The Use of Color and Third Dimension". In Proc. International Conf. Software Maintenance, pp. 99-108. 1999.
- [18] Gall, H., Hajek, K., Jazayeri, M. "Detection of Logical Coupling Based on Product Release History". In Proc. International Conf. Software Maintenance, pp. 190-8. 1998.

- [19]Gertsberger, P.G., Allen, T.J. “Criteria Used by Research and Development Engineers in the Selection of Information Source”. *Journal of Applied Psychology*. Vol. 52, No. 4, pp. 272-279. 1968.
- [20]Habel, C., and Eschenbach, C. “Abstract Structures in Spatial Cognition”, *Foundations of Comp. Science. Potential -Theory - Cognition*, LNCS 1337. Springer Verlag, Berlin, Germany, 1997.
- [21]Haglund, S., and Persson, J. “A Process for Reverse Engineering of AXE 10 Software”, Proc. 6th Reengineering Forum, Firenze, Italy, Mar. 1998.
- [22]Höst, M., Regnell, B. and Wohlin, C. “Using Students as Subjects – A Comparative Study of Students and Professionals in Lead-Time Impact Assessment”. *Empirical Software Engineering: An International Journal*, Vol. 5, No. 3, pp. 201-214. 2000
- [23]IEEE. *IEEE Standard 1471-2000: Recommended Practice for Architectural Description of Software-Intensive Systems*. IEEE, New York, USA. September, 2000
- [24]International Telecommunication Union, Sector T (ITU-T). *Recommendation Z.100:1996—Specification and Description Language SDL*. ITU, Geneva, Switzerland. 1996
- [25]Jazayeri, M., Ran, A., van der Linden, F. *Software Architecture for Product Families: Principles and Practise*. Addison-Wesley, 2000
- [26]Johansson, E., Bratthall, L., Wesslén, A., Höst, M. “The Importance of Quality Requirements in Software Platform Development – A Survey”. In Proc. Hawaii International Conference on Systems and Science (HICS-34). Jan. 2001.
- [27]Karlsson, J., and Ryan, K. “A Cost-Value Approach for Prioritizing Requirements”, *IEEE Software*, Vol. 14, No. 5, p. 67-74, Sep./Oct. 1997.
- [28]Kruskal, W.H., and Wallis, W.A. “Use of Ranks on One Criterion Variance Analysis”, *Journal of the American Statistical Association*, Vol. 47. Corrections appear in Vol. 48, 1952.
- [29]von Mayrhauser, A., and Vans, A.M. “Comprehension Processes During Large Scale Maintenance”, Proc. 16th Intl. Conf. Software Engineering, p. 39-48. Sorrento, Italy, May 1994.

- [30]von Mayrhauser, A, Wang, J., Ohlsson, M.C., and Wohlin, C. “Deriving a Fault Architecture from Defect History”, Proc. Intl. Symposium on Software Reliability Engineering, p. 295-303. 1999
- [31]Miranda, E. “An Evaluation of the Paired Comparison Method for Software Sizing”. Proc. 22nd Int’l Conf. Software Engineering, p. 597-604. Limerick, Ireland. June 2000.
- [32]Montgomery, D.C. *Design and Analysis of Experiments, Third Edition*, John Wiley & Sons, New York, 1991.
- [33]Ohlsson, M.C., von Mayrhauser, A, McGuire, B., and Wohlin, C. “Code Decay Analysis of Legacy Software through Successive Releases”, Proc. IEEE Aerospace Conference, p. 69-81. Colorado, USA. March 1999.
- [34]Ohlsson, M.C., and Wohlin, C. “Identification of Green, Yellow and Red Legacy Components”, Proc. Int’l. Conf. on Software Maintenance, p. 6-15. Bethesda, Washington D.C., USA, Nov. 1998.
- [35]Rumbaugh, J., Jacobson, I., Booch, G. *The Unified Modeling Language Reference Manual*. Addison Wesley Longman, MA, USA. 1999
- [36]Saaty, T.L., *The Analytic Hierarchy Process*, McGraw-Hill, New York, USA, 1980.
- [37]Shaw, M., Garlan, D. *Software Architecture – Perspectives on an Emerging Discipline*. Prentice Hall, USA. 1996
- [38]Soni, D., Nord, R., and Hofmeister, C. “Software Architecture in Industrial Applications”, Proc. 17th Int’l. Conf. Software Eng., p. 196-207. Apr. 1995.
- [39]Storey, M.D., Wong, K., and Müller, H.A. “How do Program Understanding Tools Affect How Programmers Understand Programs?”, Proc. Fourth Working Conf. Reverse Engineering, p. 12-21. Amsterdam, The Netherlands, Oct. 1997.
- [40]Tichy, W. “Hints for Reviewing Empirical Work in Software Engineering”. *Empirical Software Engineering: An International Journal*, Vol. 5, No. 4, pp. 309-312. 2000

- [41]Treisman, A. “Features and Objects in Visual Processing”. *Scientific American*, Vol. 254, No. 11, pp. 114-125. 1996
- [42]Tufte, E.R. *The Visual Display of Quantitative Information*. Graphic Press, CN, USA. 1983
- [43]Ware, C. “Color Sequences for Univariate Maps: Theory, Experiments, and Principles”. *IEEE Computer Graphics and Applications*. Vol. 8, No. 5, pp. 41-49. September, 1988
- [44]Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., and Wesslén, A. *Experimentation in Software Engineering: An Introduction*, Kluwer Academic Publishers, Boston, MA, USA, 1999