Controlled Experiment on the Comprehension of Runtime Phenomena Using Models Created at Design Time

Michael Szvetits
Software Engineering Group
University of Applied Sciences Wiener Neustadt
Wiener Neustadt, Austria
michael.szvetits@fhwn.ac.at

Uwe Zdun
Software Architecture Research Group
University of Vienna
Vienna, Austria
uwe.zdun@univie.ac.at

ABSTRACT

Utilizing models for software construction is a well-studied research topic. Recent research investigates the integration of models into running systems to provide additional information about the system configuration at runtime. While this additional information enables innovative self-adaptive mechanisms, it is still an open research question if the information provided by models can also improve the analysis capabilities of human users where manual intervention is inevitable for investigating runtime phenomena. This paper contributes to fill this gap by conducting a controlled experiment where the correctness and completion time of tasks regarding runtime information are assessed. A control and experiment group had to analyze the output of a software system, and the experiment group additionally received traceability links between models and associated runtime records. The results show that improvements of the analysis can especially be observed where model elements emphasize relationships between system parts that are hardly recognizable in the implementation code.

CCS Concepts

•**Software and its engineering** → *Design languages; Unified Modeling Language (UML);*

Keywords

models, runtime, comprehension, experiment

1. INTRODUCTION

Modeling plays an important role in the whole software life cycle, for example to capture requirements, communicate architectural designs, emphasize a common understanding of a system, improve documentation and make decomposition and modularization explicit. Modeling usually comes in various forms and is often done in an informal way [1]. While modeling with a minimal degree of formality is perfectly suitable for communication and documentation purposes, it lacks the ability to serve as input for more advanced model-based techniques such as model driven engineering where executable software is directly generated from formal model specifications. With an appropriate organizational alignment, model driven engineering promises not only to improve communication and control, but also to help reacting quicker to changing requirements and boost productivity and maintainability [2].

The increasing need for dynamic adaptation to new requirements and environments leads to a blur between development time and runtime, meaning that modern software must inherently embrace changeability instead of being fitted to predefined requirements that are specified by various stakeholders [3]. To cope with such dynamism, recent research utilizes models not only for constructing software, but also integrates them at runtime to provide systems with additional capabilities to reflect on their own structure and react to changes in requirements and environments more effectively [4–7]. In this paradigm, models are created at runtime and are causally connected to the running system in a way that changes to the models are propagated to the running system, and vice versa. Using models in this way promises to enable analysis of running systems on an abstraction level which is closer to the problem space [6].

While such additional reflexive capabilities have been studied in various scenarios for automatic decisions and self-adaptive systems, situations where human intervention is necessary have seldom been addressed by existing approaches which utilize models that have a relationship to the running system or its ouput [8]. We argue that linking runtime data with associated models and their elements can provide human observers with contextual information to better understand occurring situations, for example by providing condensed views of a system, visualizing faulty processes and communication paths, presenting aggregated data for containment relationships or providing control mechanisms on the model level to make ad-hoc adaptations to configuration parameters. While such feedback-driven development can be used to measure performance, detect hotspots and find root causes of problems [9], some challenges regarding traceability, (semi-)automatic data extraction and intuitive navigation between models and associated runtime information remain. Furthermore, more empirical evidence is needed to assess the benefits for human observers when runtime information is related to associated design time artefacts.

This paper makes an initial step to fill this gap by conducting a controlled experiment where the correctness and

MODELS '16, October 02-07, 2016, Saint-Malo, France

DOI: http://dx.doi.org/10.1145/2976767.2976768

completion time of six questions concerning recorded runtime information is assessed. The goal of the experiment was to find out if model artefacts created at design time can support the comprehension of system behaviour if these artefacts are related to the logged runtime information. We preferred models created at design time over runtime models to better reflect modeling activities that are established in industry [2]. 39 students with medium programming experience were separated in two groups (one of size 20, one of size 19) and had to analyze the course of events of an open source game. One group, the control group, had access to the source code of the game and the produced runtime data via a browser-based interface with textual searching and sorting capabilities. The other half, the experiment group, additionally had access to models extracted from the project documentation and was able to utilize traceability links between these models (provided by the browser-based interface) and their associated runtime records (i.e., the browser-based interface connected the models to the runtime information). The results indicate that the presence of models created at design time and their corresponding traceability links during the analysis of runtime information can improve the quality of answers to questions that concern the course of events of an executing system. The results also indicate that the quality of answers increases where model elements emphasize relationships between system parts that are hardly recognizable in the implementation code.

This paper is organized as follows: In Section 2 we compare our approach to related work. In Section 3 we give a detailed explanation of our experiment including the hypotheses, the variables, its design and its execution. Section 4 discusses the statistical evaluation of the stated hypotheses. In Section 5 we interpret our observed results. In Section 6 we analyze possible threats to the validity of our results. We conclude and discuss future work in Section 7.

2. RELATED WORK

A common approach to realize feedback between a system and an external observer is an autonomic control loop where runtime data is measured, then analyzed, and corrective actions are planned and executed based on the gained knowledge [10]. For the domain of cloud applications, Cito et al. [9] propose to include development tools into such a feedback loop to cope with shortened requirements engineering and quality assurance phases and to exploit the amount of data which is easily available in cloud applications. While various use cases are demonstrated which show the potential of the feedback-driven approach, empirical evidence of integrating models for analyzing runtime data is not provided.

Several research works have been performed to investigate the impact of models on the comprehension of software architecture and source code. Haitzer and Zdun [11] measured the supportive effect of architectural components diagrams for design understanding of novice architects. They concluded that component diagrams are useful for system comprehension if direct links between components and source code parts of interest can be established. While the provided evidence indicates that traceability links can improve the comprehension of static architectural structures, the question arises if the results can be extended to dynamic runtime behaviour as well.

Javed and Zdun [12] performed two controlled experiments concerning the supportive effect of traceability links in architecture-level software understanding. The conclusion is that traceability links between architectural models and the source code leads to a better architectural understanding of the software under observation, regardless of the experience of the human analyst. While these results also indicate that traceability links provide valuable information for software comprehension, an analysis of runtime phenomena is not in focus of this work either.

Arisholm et al. [13] report controlled experiments on the impact of UML documentation on software maintenance and came to the conclusion that UML documentation is able to improve the functional correctness of changes. In contrast to their experiments, we argue that our system under observation is closer to real world systems and much more complex, but we also see conceptual differences: Our experiment focuses on the analysis of runtime behaviour and the produced runtime data, while their experiments mostly target the understanding of the design time artefacts themselves and on extending them appropriately.

In the work of Cornelissen et al. [14, 15] the impact of dynamic analysis and trace visualization on the comprehension of software has been analyzed. Dynamic analysis was investigated through a systematic survey which identified various methods to analyze runtime information like slicing, filtering, querying, metrics, pattern detection and static analyses of source code and documentation. However, relating modeling artefacts created at design time with actual runtime information seems under-represented in the identified literature. While the survey revealed related work in terms of model-based analysis of runtime data [16], the presented models are often generated post-mortem and used to gain insight into the produced data by innovative visualization.

In terms of trace visualization, Cornelissen et al. [15] conducted a controlled experiment with EXTRAVIS [17], a tool which provides a large-scale UML sequence diagram and a circular projection of structural entities as interactive visualizations. The goal of the experiment was to find out if the correctness of answers to system-related questions and the needed time improve by using the visualizations. The results show a decreased time and an increased correctness, but give no indication whether models created at design time can improve the comprehension as well.

Gravino et al. [18] investigated whether the comprehension of object-oriented source code increases when it is added with UML class and sequence diagrams produced in the software design phase. The results show an average improvement of 12% in solving different comprehension tasks when using class and sequence diagrams. On the other hand, they also concluded that only experienced participants could adequately take advantage of the provided models. While the experiment is strongly related to our approach, we focus on the comprehension of runtime phenomena and incorporate more UML models than class and sequence diagrams.

As a summary, there is a strong indication that models, visualizations and traceability information seem to improve overall system comprehension, but additional empirical evidence is necessary to assess the suitability of models for understanding the course of events of running systems. We believe that relating models with runtime information provides additional analysis capabilities, especially in the context of model driven engineering where some traceability information can directly be inferred from existing code generators and do not have to be specified manually [19].

3. EXPERIMENT DESCRIPTION

The design of the experiment is inspired by the guidelines of Kitchenham et al. [20] which concern the description of populations, sampling techniques, treatment allocations and bias minimization. For the statistical analysis, we followed the recommendations of Wohlin et al. [21] which describe the effects between independent and dependent variables, the presentation of descriptive statistics, the testing of hypotheses and the correct reporting of validity concerns. The resources of the experiment are available online[1].

3.1 Goal and Hypotheses

The goal of the experiment was to find out if the information provided by models created at design time can improve the analysis capabilities of human users if manual intervention is inevitable for investigating runtime phenomena. More specifically, the experiment should reveal if there is a significant improvement in the correctness of statements about the system behaviour if models are available and their elements can be traced to associated records of runtime information. We aimed to generalize our findings as effectively as possible by providing participants in the experiment group with UML models of different types and abstraction levels:

- A use case diagram with 39 model elements for analyzing high-level scenarios of the software under observation.
- A component diagram with 9 model elements for analyzing the high-level architecture of the software.
- A package diagram with 35 model elements to support a fine-grained analysis of one of the components.
- An activity diagram with 34 model elements showing a complex process in an implementation-agnostic way.
- Two sequence diagrams with 32 and 47 model elements showing two complex processes from a perspective which is close to the source code.
- Two class diagrams with 32 and 71 model elements showing the structure of essential parts from a perspective which is close to the source code.

We chose UML as modeling language because it is complete and the de-facto standard for modeling both structural and behavioural features of software systems on various abstraction levels. Furthermore, UML is the language that the participants of the experiment were capable of using and understanding.

Beside the correctness, another goal of the experiment was to determine if there is a significant difference between the control and experiment group regarding the time spent for answering the questions. The participants were instructed to write down the time spent for every question when they feel that their answer is complete.

3.1.1 Hypotheses

Based on the goals of the experiment, we formulated two null hypotheses and corresponding alternative hypotheses for the experiment:

- H_{01}: Models and traceability links between their elements and related runtime data *do not significantly improve* the correctness of given answers about the system behaviour.
- H_{A1}: Models and traceability links between their elements and related runtime data *significantly improve* the correctness of given answers about the system behaviour.
- H_{02}: The times spent for analyzing the system behaviour *do not significantly differ* if models and traceability links to related runtime data are provided.
- H_{A2}: The times spent for analyzing the system behaviour *significantly differ* if models and traceability links to related runtime data are provided.

3.1.2 Expectations

Regarding the correctness, we expect that the presence of models and their traceability links are especially useful for questions that target high-level system understanding, e.g. if specific user interactions should be analyzed based on the observed data. This expectation results from our impression that observed runtime data (e.g., using log files) is usually not directly connected to use case scenarios, and traceability links from high-level models to associated runtime data can provide a good starting place for further analysis. For such cases, we expect that the null hypothesis H_{01} can be rejected. In other cases where questions target behaviour which is closely related to the source code, we expect that the null hypothesis H_{01} cannot be rejected.

Similar to the correctness, we expect that the additional information provided by models and their traceability links do not hinder the completion time of questions, but instead improve the time for scenarios closer to the problem space. As a result, we expect that the null hypothesis H_{02} can be rejected for questions that target high-level scenarios. In other cases, we expect that the null hypothesis H_{02} cannot be rejected.

3.2 Parameters and Variables

Two dependent and four independent variables were observed during the experiment. Table 1 gives an overview of the six variables with their associated scale types, units and value ranges. Note that the independent variables were observed once per participant, while the dependent variables were observed once for each question per participant.

Table 1: Observed Variables of the Experiment

Description	Scale	Unit	Range
Dependent Variables			
Correctness	Interval	Points	[0, 1]
Time	Interval	Minutes	[0, 180]
Independent Variables			
Group affiliation	Nominal	N/A	Control group, Experiment group
Programming experience	Ordinal	Years	4 classes: 0, 1-3, 3-7, >8
Programming experience in industry	Ordinal	Years	4 classes: 0, 1-3, 3-7, >8
Software design experience	Ordinal	Years	4 classes: 0, 1-3, 3-7, >8

[1]see: http://jarvis.fhwn.ac.at/controlled-experiment-mars/

3.2.1 Dependent Variables

For measuring the correctness of an answer, we decided to abandon open-ended questions which are inherently sensitive to subjective bias of human analysts. Instead, we created six different questions for which the expected answers are always a list of distinct elements, which means that formulating free text by participants was unnecessary. Conducting the experiment in such a way allowed us to apply metrics from information retrieval systems which rely on the set of mentioned elements (the answer of the participant) and the set of expected elements (the preferred solution) per question [22]. If $R_{p,q}$ denotes the set of elements mentioned by participant p for question q, and C_q denotes the expected elements in the solution for question q, then these metrics are calculated in the following way:

$$Precision_{p,q} = \frac{|R_{p,q} \cap C_q|}{|R_{p,q}|} \qquad Recall_{p,q} = \frac{|R_{p,q} \cap C_q|}{|C_q|}$$

Precision is the fraction of mentioned elements that are correct, and recall is the fraction of expected elements that were actually found [22]. Since these metrics are hardly comparable, we unified them using the harmonic mean to compute the so-called F-measure, an indicator for the quality of the given answer with a value range of $[0, 1]$ where 0 denotes the worst and 1 the best quality. This F-measure is used as metric for the overall correctness of an answer given by participant p to question q and is computed as follows:

$$Correctness_{p,q} = F_{p,q} = 2 * \frac{Precision_{p,q} * Recall_{p,q}}{Precision_{p,q} + Recall_{p,q}}$$

For measuring the completion time of an answer, participants were instructed to write down the time spent for every question when they feel that their answer is complete (the maximum time for the overall experiment was 180 minutes). Participants were able to write down multiple time entries if they decided to return to a question later on. In such a case, the overall time spent for a question is the sum of all entries.

3.2.2 Independent Variables

According to Table 1, four different independent variables have been observed which can influence the results of the dependent variables. We tried to mitigate the influence of programming and design experience on the dependent variables by conducting the experiment with participants that have similar education and by choosing the questions and the system under observation in a manner so that experience provides no significant advantage.

3.3 Experiment Design

To test the hypotheses stated in Section 3.1, we conducted the experiment in the context of two courses focussing software architectures and adaptive software systems at the University of Applied Sciences Wiener Neustadt in the winter semester 2015.

3.3.1 Subjects

The subjects of the experiment are 39 students with medium programming experience and knowledge of software modeling and software architectures in general. The participants were randomly assigned into two groups, a control group of size 20 and an experiment group of size 19.

3.3.2 Object

The runtime information to be analyzed by the participants originates from the Mars Simulation Project[2] version 3.07, an open source social simulation of future human settlement of Mars. The game allows to model human behaviour and maintain settlements across the planet. The survival of the people depends on their social interactions, their collaboration in expanding their territory and improving their skills. The system was chosen for various reasons:

- The project is open source, thus allowing to distribute the source code to the participants.
- With a size of 213.794 lines of code, the project is small enough for participants to comprehend the overall structure, but at the same time big enough so that it is impossible for participants to learn the whole source code during the experiment (and thus leading to bias).
- Choosing a game as system under observation is motivating and a familiar domain for most of the participants.
- The game can easily be explained, but must not fully be understood to answer the questions in the experiment.
- The project is written in Java, a programming language the participants are familiar with.
- The source code seems to be in good quality with a deliberate use of best practices and design patterns.
- The project documentation already contains five UML diagrams which describe essential parts of the software. This is important because we wanted to experiment with models that are not created entirely by ourselves, thus reducing researcher bias as much as possible.

3.3.3 Instrumentation

The participants of both the experiment and the control group received browser-based access to the source code of the Mars Simulation Project, generated by Maven JXR[3]. Furthermore, participants gained browser-based access to the produced log output of the Mars Simulation Project in a table-based manner with the following columns:

- *Number:* This column is just a chronological enumeration of the log file entries.
- *Type:* This is an indicator if the log entry is a method call, method execution or an exception.
- *Source:* This is the fully qualified name of the source method of a call or exception. A click on the entry leads to the exact source code location of the call.
- *Target:* This is the fully qualified name of the target method of a call. A click on the entry leads to the exact source code location of the called method.
- *Message:* This column holds arbitrary information of the call, execution or exception. An example would be the string representations of the caller and callee objects.

[2]see: http://mars-sim.sourceforge.net/
[3]see: http://maven.apache.org/jxr/

Table 2: Traceability Information Provided for the Experiment Group

UML Element	Traced Log Entries	Traced Model Elements
Component	Calls within the component	Contained packages/classes
Package	Calls within the package	Contained packages/classes
Dependency	Calls between dependent components/packages	None
Class	Calls concerning class operations/properties	Contained classes, operations, properties
Operation	Calls of the associated method	Associated activity/action
Property	Calls concerning the property (getter, setter)	None
Lifeline	Calls concerning the represented object in the sequence	None
Activity	Calls during the represented operation of the activity	Contained operations/actions
Action	Calls of the associated method during the parent activity	Associated activity/operation
Control Flow	Associated method call or logger call, if available	None
Use Case	Calls concerning the realizing classifier of the use case	Realizing classifier

All of the columns could be searched textually by using textboxes on top of the columns. Participants were able to search for multiple terms in different columns, meaning that they could apply multiple column filters at once. Beside filtering, participants were also able to perform lexicographical sorting of the log entries in ascending and descending order by clicking on the respective column headers. Overall, the searchable log file consisted of 480.000 entries which resulted from recording a gaming session of 30 seconds. The large amount of log file entries ensures that the asked questions could not be answered by exhaustively going through the log file, but instead by cleverly applying the provided filtering, sorting and navigation capabilities.

In addition to the materials mentioned above, the experiment group also gained browser-based access to the eight UML diagrams mentioned in Section 3.1. Five of the eight diagrams were directly extracted from the project documentation, but had to be redrawn in Eclipse Papyrus[4] because they were only available as image and could not be linked to associated log entries. The other three diagrams (a use case diagram showing the possible user interactions, and two sequence diagrams showing two scenarios that were performed while the gaming session was recorded) were created by ourselves based on the information available in the project documentation. Table 2 shows the provided traceability links between model elements and log entries that were provided for the experiment group.

The navigation from log entries to their associated model elements was achieved by an additional column in the table-based log access which was only accessible for the experiment group. The links from model elements to associated log entries and other model elements could be utilized by the experiment group by clicking the respective elements in the browser.

Beside digital, browser-based material, both groups received a questionnaire to be answered during the experiment. The first page of the questionnaire contained questions regarding the independent variables (programming experience, programming experience in industry, and software design experience). The second and third pages contained the actual six questions to be assessed for correctness and completion time. Table 3 gives an overview of the six questions. Note that every question can only be answered by applying the correct filtering and sorting mechanisms. None of the questions could be answered by looking into the source code or the models alone since this would provide an advantage for either the control or the experiment group and would not actually measure the comprehension of runtime phenomena. Furthermore, some models and model elements provided for the experiment group were useless for answering the question, which is important to reflect a realistic scenario where design time artefacts are created without specific predetermined analyses in mind.

For question *Q1*, participants had to identify the concrete subtypes of the class *Unit* either in the source code or in the models (for the experiment group) and then filter the log according to constructor calls of those types. The challenge for this question was to filter only the non-abstract classes out of the log. There was especially a challenge for the experiment group since the class diagram provided by the project documentation depicts only a partial view of the whole class hierarchy. As a result, participants of the experiment group could not rely completely on the provided class diagram.

For questions *Q2-Q3*, participants had to analyze the amount of method calls between components and packages. The experiment group could utilize traceability information of model elements that represent dependency relationships.

For question *Q4*, participants had to list the names of all people on Mars that had at least once neither an active task (e.g., explore parts of Mars) nor an active mission (e.g., rescue people on Mars after a vehicle crash). The challenge was to identify the occurrence of unemployment either in the source code or in the provided diagrams (more specifically, in the activity diagram describing the mind of a person) and

Table 3: Questions of the Controlled Experiment

ID	Description
Q1	How often was each concrete subtype of *Unit* instantiated while the event log was recorded? Make a list of the subtypes with their respective instantiation counts.
Q2	Analyze the degree of coupling between the Mars simulation components by counting the method calls between them (if any). Make a list of component pairs with their respective call counts.
Q3	Analyze the degree of coupling between the following packages: *msp*, *person*, *mars*, *structure*, *science*. Make a list of package pairs with their respective call counts (in both directions, if any).
Q4	Name the people who – at least once – had neither an active task nor an active mission. The task/mission management is found in the class *Mind*.
Q5	Which classes were used at runtime to check the status of a health problem? Health problems are managed in the method *HealthProblem.timePassing*.
Q6	Which manual actions did the user perform during the session?

[4]see: http://www.eclipse.org/papyrus/

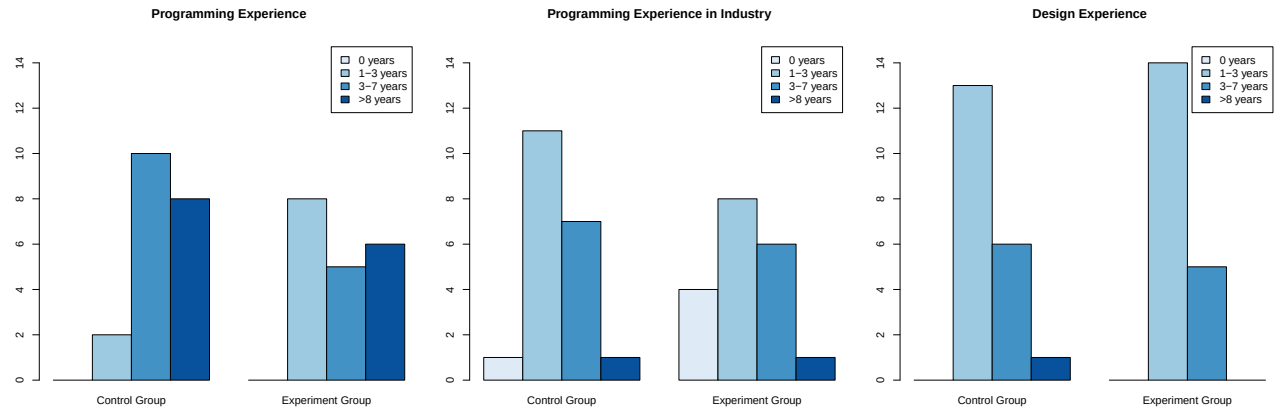

Figure 1: Participant Experience of the Controlled Experiment

then filter the log according to the identified method which indicates the unemployment of a person.

For question *Q5*, participants had to identify the classes that were actually used to check the health status of a person on Mars and write down the amount of calls these classes were receiving during the check. While potential classes can easily be identified in both the source code and the provided diagrams (in the sequence diagrams, more specifically), the question can only be answered by filtering the log to check which of the potential classes were used how often.

For question *Q6*, participants had to identify the user interactions which occurred during the game simulation. For both the experiment and control group, a list of possible user interactions was provided on the question sheet. Participants of the experiment group were additionally able to utilize the provided use case diagram to search for potential user interactions. Note that some of the recorded user interactions were intentionally performed in a way that the log had to be analyzed very carefully to identify if the user interaction actually happened. One example is the user interaction of changing a mission of a person on Mars: In the recorded session, the user opened the dialogue which allows to change the mission, but never actually confirmed the mission changes and instead cancelled the dialogue. Such distracting interactions ensure that the pure existence of traceability information between use cases and their realizing classifiers do not reveal the answer without the need for looking further into the filtered log entries.

3.4 Experiment Execution

At the beginning of the experiment, the participants were randomly assigned to the control and experiment group. After assignment, the groups consisted of 20 and 19 participants. The control and experiment group performed the experiment in two separate rooms, not knowing that there is an actual group affiliation involved.

Before the experiment began, the system under observation as well as the upcoming tasks were explained to the participants. The Mars Simulation Project was not completely new for some of the participants because it was used within a course at the University of Applied Sciences Wiener Neustadt, but none of the participants knew its implementation and none of them were aware of the upcoming tasks.

For exactly 15 minutes, participants were given the time to get familiar with the browser-based interface and its capabilities. After this initial phase, the paper materials described in Section 3.3 were handed to the participants and the questions were explained. Every participant seemed to understand the tasks and the participants were given 180 minutes to tackle the questions and fill out the questionnaire. All participants had exactly the same type of computer with exactly the same hardware specification. The usage of private computers was prohibited during the experiment to provide the same conditions for every participant. The questionnaire had to be filled out directly on the questionnaire paper.

The data collection was performed as planned in the design phase of the experiment. No participants prematurely quitted the experiment and no deviations from the study design occurred. The experiment itself took place in a controlled environment, more specifically within two lecture rooms equipped with the needed computers as mentioned above. At least one experimenter was present in each room during the whole experiment to prevent participants from using forbidden materials and to resolve potential questions by the participants.

After completing the experiment, the questionnaires were collected by the experimenters and a discussion about the difficulties during the experiment and the experiences of the participants was initiated. The outcomes of this discussion did not influence any results presented in this paper, but they were important for the experimenters to reveal any potential weaknesses in the experiment design and to understand the different strategies of answering the questions that participants used based on the available materials. Another reason for the discussion afterwards was simply to collect feedback and generate new ideas for ongoing research.

4. EXPERIMENT RESULTS

4.1 Descriptive Statistics

Figure 1 shows the programming and design experience as declared by the participants on the first sheet of the questionnaire. The figure shows that the control group has slightly more programming experience, but the differences in industrial programming experience between the control and the experiment group is negligible. Both groups have similar experiences when it comes to software design. Based on the profiles shown in Figure 1, we believe that neither one of the two groups had a significant advantage regarding the asked questions based on their personal experience.

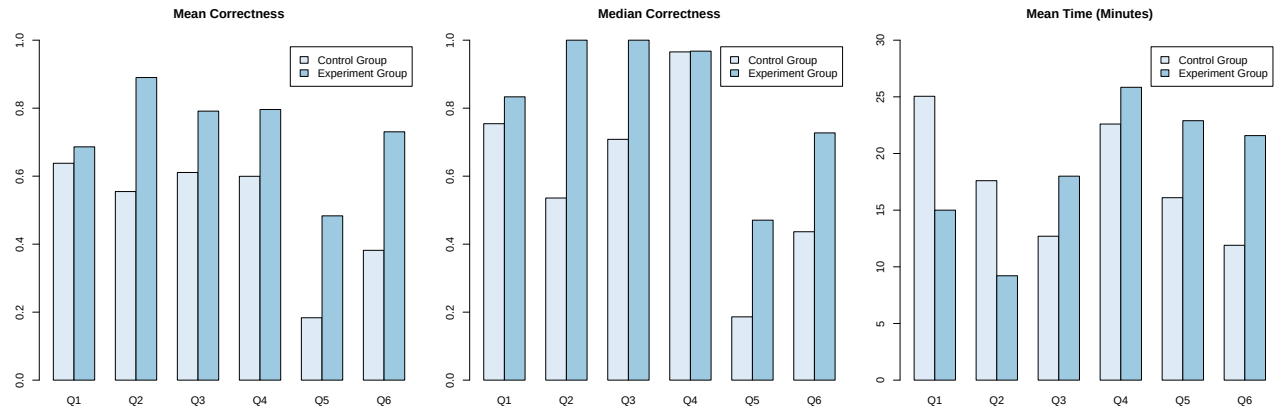

Figure 2: Descriptive Statistics for the Correctness and Time of the Controlled Experiment

Table 4: p-Values of the Shapiro-Wilk Normality Tests

Variable	Group	Size	Q1	Q2	Q3	Q4	Q5	Q6	All
Correctness	Control Group	20	**0.005860**	0.109503	**0.004913**	**6.130e-05**	**0.021665**	**0.015901**	**3.780e-07**
	Experiment Group	19	**3.784e-05**	**6.616e-05**	**0.000244**	**3.357e-05**	0.125054	0.878598	**5.008e-10**
Time	Control Group	20	0.575771	0.216652	0.367927	0.875501	0.773400	0.108051	**0.031904**
	Experiment Group	19	**0.033202**	0.056261	0.140212	0.813237	0.269014	0.168377	**0.000125**

Figure 2 shows the mean and median correctness of the answers and the mean completion time per question that resulted from the questionnaires. The figure shows that the experiment group achieved higher scores of correctness for the questions, regardless if the mean or the median correctness is considered. Although the experiment group achieves better results for the correctness, the differences between the experiment and the control group are not significant for all questions (e.g., when comparing the mean correctness of *Q1*). The figure also shows that the experiment group performed exceptionally well for the questions *Q2-Q3* since the median values are at maximum, meaning that more than half of the participants in the experiment group could find exactly the right answers.

The comparison of the completion time per question shows that the experiment group performed questions *Q1-Q2* faster, but questions *Q3*, *Q5* and *Q6* slower than the control group. Figure 2 indicates no significant difference in the completion time for question *Q4*. Note that the speed of completion is neither an indicator for the overall quality of an answer, nor is it an indicator if one of the groups performed better. The participants of the experiment were instructed to write down the completion time for each question when they are confident that their answer is complete. They were not instructed to complete the given tasks as fast as possible.

4.2 Handling of Outliers

While investigating the results of the questionnaires, we considered to exclude one participant of the control group who was not able to provide meaningful answers to the questions *Q3-Q5* and also spent rather little time on those questions. But as the participant was able to provide reasonable answers for the other three questions, we concluded that the participant understood the topic and was motivated, but simply skipped questions *Q3-Q5* to improve the results in the other questions. We performed all analyses without the participant, and excluding the participant would not have led to substantially different conclusions. For these reasons, we decided not to exclude the participant from further analyses. All other participants provided reasonable results for all of the questions, which gave us confidence that there is no flaw in the design of the experiment.

4.3 Hypothesis Testing

For testing the hypotheses formulated in Section 3.1, the initial step was to test the normality of the data by using the Shapiro-Wilk test of normality [23]. Based on the results of the Shapiro-Wilk normality test, further tests can either be parametric and rely on normality (e.g., the t-test) or be non-parametric without making assumptions of the underlying distribution (e.g., the Wilcoxon Rank-Sum test). The null hypothesis of the Shapiro-Wilk test states that the observed data is normally distributed. Based on a significance level of $\alpha = 0.05$, Table 4 shows the results of the Shapiro-Wilk tests for both the correctness and the completion time for each question and the overall experiment. Numbers in bold indicate that the p-value of a test is lower than the significance level of $\alpha = 0.05$, thus suggesting to reject the null hypothesis and assume that the observed data is not normally distributed. Since the data for correctness seems to be non-normally distributed in the majority of cases, we decided to make further analyses of the correctness based on tests that do not assume normality of the data. For the completion time, the results of the Shapiro-Wilk tests predominantly suggest normality, so we decided to make further analyses of the completion time based on parametric tests that rely on normality.

For comparing the correctness in a non-parametric way, we applied one-tailed Wilcoxon Rank-Sum Tests [24] as shown in Table 5. The null hypothesis of the Wilcoxon Rank-Sum Test states that the mean correctness achieved

Table 5: Wilcoxon Rank-Sum Tests for Correctness

ID	Control Group		Experiment Group		p-Value
	Mean	Std.Dev.	Mean	Std.Dev.	
Q1	0.63791	0.31101	0.68623	0.31871	0.31329
Q2	**0.55473**	0.33774	**0.89013**	0.15619	**5.62e-04**
Q3	**0.61083**	0.37020	**0.79131**	0.29845	**0.03694**
Q4	0.59959	0.45881	0.79613	0.32344	0.06149
Q5	**0.18349**	0.16650	**0.48325**	0.31586	**0.00139**
Q6	**0.38184**	0.19958	**0.73047**	0.08808	**1.33e-07**
All	**0.49473**	0.35570	**0.72959**	0.29032	**2.01e-07**

Table 6: t-Tests for Completion Time

ID	Control Group		Experiment Group		p-Value
	Mean	Std.Dev.	Mean	Std.Dev.	
Q1	**25.0500**	10.3490	**15.0000**	7.7172	**0.0015**
Q2	**17.6000**	8.7202	**9.2105**	4.7443	**0.0008**
Q3	**12.7000**	7.8947	**18.0000**	6.3944	**0.0268**
Q4	22.6000	10.3079	25.8421	9.1667	0.3055
Q5	16.1000	6.9578	22.8947	13.2870	0.0574
Q6	**11.9000**	7.4332	**21.5789**	11.7630	**0.0047**
All	17.6583	9.8035	18.7544	10.6576	0.4145

by the control group is stochastically larger than or equal to the mean correctness achieved by the experiment group. We applied unpaired tests for every question and one unpaired test to compare all questions together. Numbers in bold indicate that the p-value of a test is lower than the significance level of $\alpha = 0.05$, thus suggesting to reject the null hypothesis and assume that the mean correctness achieved by the control group is less than the mean correctness achieved by the experiment group. This is the case for questions *Q2*, *Q3*, *Q5*, *Q6* and for the overall comparison.

For comparing the completion time in a parametric way, we applied two-tailed t-tests [25] as shown in Table 6. The null hypothesis of the t-test states that the mean completion time for a question needed by the control group does not significantly differ from the completion time needed by the experiment group. We applied unpaired tests for every question and one unpaired test to compare all questions together. Numbers in bold indicate that the p-value of a test is lower than the significance level of $\alpha = 0.05$, thus suggesting to reject the null hypothesis and assume that the mean completion time achieved by the control group significantly differs from the mean completion time achieved by the experiment group. This is the case for *Q1*-*Q3* and *Q6*.

5. INTERPRETATION

As shown in Table 5, the majority of cases indicate that hypothesis H_{01} has to be rejected and the alternative hypothesis H_{A1} holds: Models and traceability links between their elements and related runtime data *significantly improve* the correctness of given answers about the system behaviour. Although a significant improvement of the correctness could not be shown for questions *Q1* and *Q4*, the results indicate that the experiment group performed slightly better than the control group for these two questions as well.

The reason for the close results for question *Q1* could be the well-structured source code of the Mars Simulation Project: The class diagram provided by the project documentation does not seem to provide better access to the class hierarchy of *Unit* than the source code itself. Nevertheless, there was only one participant (one of the control group) that achieved a perfect answer to question *Q1*. The reason that no one of the experiment group could achieve a perfect result to question *Q1* could be that participants considered the provided class diagram to be complete and did not look further into the source code to identify additional subtypes. The reason for the close results for question *Q4* could be similar: The hint that the task and mission management can be found in the class *Mind*, paired with the good code quality, seems to mitigate some of the advantages the provided activity diagram could provide.

On the other hand, it is noteworthy that the experiment group performed significantly better for questions *Q2*, *Q3*, *Q5* and *Q6* where related diagrams make concepts explicit (and thus, traceable) that are not easy to identify in the source code alone, for example component dependencies, package dependencies, message flows and use cases. The results suggest that models are especially useful if runtime information can be related to high-level architecture and design concepts for which no direct counterparts exist in the implementation code.

Another interesting picture arises if the completion time of Table 6 is taken into account. While the correctness does not differ significantly between the two groups for question *Q1*, it seems that the experiment group reached confidence about the completeness of the given answers much faster than the control group. This matches our impression that the experiment group considered the provided class diagram to be complete, thus coming faster to the conclusion that no further subtypes exist. Furthermore, the experiment group performed question *Q2* much faster, which may be due to the fact that dependencies between components (realized as Java projects) are made explicit in the provided component diagram, and can thus be traced to associated log entries more easily. On the other hand, it took the experiment group significantly longer to find confidence in the answers to question *Q6*. This could be because the traceability links from the use case diagram to the realizing classifiers may have provided more intuitive starting points for analyses than the source code and the textual filters. As a consequence, the experiment group had more potential places to look for user interactions, whereas the control group may have believed sooner that there are not any more relevant log entries to find.

Beside questions *Q1*-*Q3* and *Q6*, the results show no evidence that the time spent for analyzing the system behaviour differs significantly between the experiment and the control group. Especially the overall time spent per question is very similar between the two groups (see the last row of Table 6). This is a very mixed result, but since the overall completion time is almost identical, we argue that the null hypothesis H_{02} cannot be rejected: The times spent for analyzing the system behaviour *do not significantly differ* if models and traceability links to related runtime data are provided. Nevertheless, there are indicators that the null hypothesis must be rejected in cases where models are rather complete or emphasize relations that are not easy to discover in the source code alone (e.g., dependencies between components and packages, or links to associated use cases). However, it is noteworthy that for both correctness and completion time the quality of significance for question *Q3* is considerably

lower (i.e., the p-value is higher) than for other questions that show significant results. The difference in performance between the control and experiment group is smaller for this question. This could be because UML packages are directly translated to Java packages, thus mitigating the advantage of having the package diagrams at hand.

6. THREATS TO VALIDITY

According to Cook and Campbell [26], we analyze the validity of our results in four dimensions: *Construct validity* refers to the degree to which the applied experiment techniques are adequate to measure what was intended to be measured. *Internal validity* refers to the degree to which the observed results really follow from the collected data due to correct elimination of confounding variables. *External validity* refers to the degree to which the observed results are generalizable beyond the conducted experiment. *Conclusion validity* refers to the degree to which conclusions about the interactions of observed variables are statistically valid.

6.1 Construct Validity

One could argue that the provided materials (a searchable log file of method calls with filtering and sorting capabilities) do not necessarily reflect reality if runtime phenomena have to be analyzed. We argue that structured analysis of log files is still an important source of information which is widely adopted [27]. Furthermore, the instrumentation of source code with appropriate print statements is still an often used technique to inspect and debug running systems. We argue that this kind of instrumentation is very similar to providing the chronological history of method calls as it has been done in the experiment. Furthermore, providing more powerful searching mechanisms like complex event filters and query languages may be closer to real world scenarios, but would require a deeper level of preliminary knowledge for the participants which cannot be guaranteed easily.

Another threat is that the asked questions might not have enough similarity with actual real world situations. We argue that in the presence of unexpected behaviour, it is highly desirable to know which parts of the software actually qualify to be the root source of the problem. Such situations should be represented by the questions *Q2*, *Q3* and *Q5*. Another realistic scenario is the assessment of footprints and performance hot spots of running systems, which should be represented by the questions *Q1*, *Q2* and *Q3*. Independent of the domain, an understanding of the (usually erroneous) behaviour of a system is highly desirable if certain situations or user interactions occur, which should be represented by questions *Q4* and *Q6*. While this threat cannot be eliminated completely, we believe that our questions adequately represent scenarios similar to the real world while being feasible to be answered by participants in a controlled environment. There is obviously a trade-off to make in this respect.

The fact that only one object, the Mars Simulation Project, has been analyzed, introduces the risk that the observed results are specific to the examined case and domain. We think that this factor is not a strong threat to validity since the tasks performed by the participants are very generic and require no domain-specific knowledge. Furthermore, we argue that the object is an ideal surrogate for several real world projects that rely on modeling because of its existing modeling artefacts that were created at design time. It is common that design time modeling artefacts are not created with upcoming runtime analyses in mind.

A similar threat to validity is given by the fact that only one measure for correctness is recorded in our experiment. Although our assessment of correctness combines two widely adopted measures from information retrieval systems, further metrics for correctness would allow to cross-check our observed results more efficiently.

6.2 Internal Validity

Misbehaviour of participants during the experiment can never be prevented completely. An example would be that a participant writes down the start time of a task, but forgets to write down the end time when finishing the answer. We mitigated such risks by making sure that an experimenter is always present during the experiment. Especially the problem of noting incorrect times can be discovered quickly by the experimenter because there can never be two noted start times at once without one of them having a noted end time.

As shown in Figure 1, the control group seems to have slightly more programming experience than the experiment group. Such differences cannot be eliminated completely, but we consider the influence of the programming experience on the tasks rather low because none of the participants knew the implementation of the test object beforehand, no code had to be written manually during the experiment and the existing code of the Mars Simulation Project is very readable. Regarding other possible differences in the experience, all participants passed the same courses at the university that are relevant for the experiment, more specifically courses that target the understanding of programming, software architecture and software design. As a result, we consider that every participant was able to understand the questionnaire, especially the used terminology and concepts within the questions. We also distributed participants with pre-existing knowledge of the Mars Simulation Project evenly between the control and experiment group.

Another variation in human performance might result from fatigue effects due to the fact that the experiment lasted three hours. We do not consider this as a strong threat to validity because the questions were designed with a much lower completion time in mind and the reserved time was intentionally generous to reduce stress. In fact, as shown in Table 6, the average completion time per question is about 19 minutes, which extrapolates to an average completion time for the questionnaire of under two hours.

Another threat to validity is a potential bias introduced by the diagrams created by the experimentors. We believe that there is no strong bias, since the used models are exactly the same as in the project documentation. Exceptions are the use case and sequence diagrams, which were created based on the same documentation and semi-automatically with the help of Altova UModel[5], so with hardly any manual intrusion of the experimentors.

Another threat to validity is the incorrect assessment of answers and a potential bias towards the experiment group. We tried to mitigate that risk by the design of the questions and the questionnaire itself: Every question requires a list of distinct answers which can easily be verified objectively, and the real identities of participants were not transferred to the phase of assessing the correctness of answers.

[5]see: http://www.altova.com/umodel.html

6.3 External Validity

A possible threat to validity is that the Mars Simulation Project could be too simple, or as a game not suitable to be generalized to other business domains. There is obviously a trade-off to make between the motivation of the participants and the size of the test object that participants can handle in a limited time. Regarding the size, we argue that the project is small enough for participants to comprehend the overall structure in the given time, but at the same time big enough so that it is impossible for participants to learn the whole source code during the experiment. Regarding the complexity, we argue that the Mars Simulation Project, due to its complex simulation conditions and performance considerations, is equally complex as comparable business domains, maybe even more complex. However, the threat cannot be ignored completely, and experiments with systems and models of different sizes would strengthen the generalizability and validity of our results.

Another threat is that only a selected number of techniques was provided for the participants to answer the questions. As mentioned before, providing more powerful searching mechanisms like complex event filters and query languages may be closer to real world scenarios, but would require a deeper level of preliminary knowledge for the participants which cannot be guaranteed easily. We believe that a broad generalization to arbitrary tools and techniques is simply not possible, but argue that our web-based log access adequately allows the most important searching and filtering operations that are provided by a wide range of tools.

We used students with limited professional experience as participants for our experiment. Although we believe that differences in experience does not significantly influence the observed results because of our carefully selected questions, the potential threat to validity cannot be ignored completely. A replication of our study with experienced practitioners would provide additional insight into the validity and generalizability of our results.

6.4 Conclusion Validity

A threat to validity might result from too many or too little distracters (possible wrong answers) in the experiment design. We reduced this risk by providing a mix of models and model elements that either contribute to solve a task or do not help at finding the correct answer at all. Participants of the experiment group were not instructed to use certain models for certain questions, so every participant must have stumbled over several distracting elements while investigating the tasks. Regarding distracters on the log level, the recorded amount of 480.000 log entries made sure that relevant entries had to be actively searched and could not be found by pure chance.

Another threat to validity might result from guessing, meaning that participants may have simply tried to guess correct answers. Since the experiment was designed in a way that people only had vague directions to look for the answers (e.g., by mentioning specific classes) and the questions had no predefined answers, we believe that guessing was nearly impossible. In addition, when considering the large amount of log entries, it is highly doubtful that the statistical results were distorted by pure luck of the participants.

From the analysis point of view, we relied on objective techniques from information retrieval systems instead of subjective ad-hoc assessments by human analysts. Nevertheless, distortions may also result from the application of inappropriate statistical methods. We tried to mitigate that risk by using either the t-test or the Wilcoxon Rank-Sum Test based on the normality of the observed data, which was tested using the Shapiro-Wilk test of normality. However, we want to point out that every statistical test is inherently subject to probabilistic errors, so the Shapiro-Wilk test itself also introduces inaccuracies that cannot be eliminated completely.

The statistical validity might also be affected by the sample size of 39 participants. The size can be improved by replications of the study, ideally with experienced practitioners and systems of different domain and size.

7. CONCLUSIONS AND FUTURE WORK

In this paper we describe the results of a controlled experiment that was conducted to find out if models created at design time can improve the analysis capabilities of human users if manual intervention is inevitable for investigating runtime phenomena. In the experiment, the correctness and completion time of questions regarding recorded runtime information are assessed. While the results provide initial evidence that models help to give more correct answers to questions about system behaviour if they are linked to related runtime information, a difference in the completion time could only be observed for single cases.

We argue that the improvement of the correctness and the differences of the completion time are especially noticeable where model elements emphasize relationships between system parts that are hardly recognizable in the implementation code. This argumentation matches the fact that a significant difference of the correctness could not be identified for questions which target the understanding of runtime phenomena that are closely related to parts of the source code. For future work, additional research needs to be done to investigate this correlation between the explication of high-level relationships and the quality of manual system analysis.

8. REFERENCES

[1] M. Petre, "Uml in practice," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 722–731. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486883

[2] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen, "Empirical assessment of mde in industry," in *Software Engineering (ICSE), 2011 33rd International Conference on*, May 2011, pp. 471–480.

[3] C. Ghezzi, "The fading boundary between development time and run time," in *Web Services (ECOWS), 2011 Ninth IEEE European Conference on*, Lugano, Switzerland, 2011, pp. 11–11.

[4] N. Bencomo, "On the use of software models during software execution," in *Proceedings of the 2009 ICSE Workshop on Modeling in Software Engineering*, ser. MISE '09, Vancouver, Canada, 2009, pp. 62–67. [Online]. Available: http://dx.doi.org/10.1109/MISE.2009.5069899

[5] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg, "Models@ run.time to support dynamic adaptation," *Computer*, vol. 42, no. 10, pp. 44–51, Oct. 2009. [Online]. Available: http://dx.doi.org/10.1109/MC.2009.327

[6] G. Blair, N. Bencomo, and R. B. France, "Models@ run.time," *Computer*, vol. 42, no. 10, pp. 22–27, Oct. 2009. [Online]. Available: http://dx.doi.org/10.1109/MC.2009.326

[7] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven, "Using architecture models for runtime adaptability," *IEEE Softw.*, vol. 23, no. 2, pp. 62–70, Mar. 2006. [Online]. Available: http://dx.doi.org/10.1109/MS.2006.61

[8] M. Szvetits and U. Zdun, "Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime," *Software & Systems Modeling*, pp. 1–39, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10270-013-0394-9

[9] J. Cito, P. Leitner, H. C. Gall, A. Dadashi, A. Keller, and A. Roth, "Runtime metric meets developer: Building better cloud applications using feedback," in *2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)*, ser. Onward! 2015. New York, NY, USA: ACM, 2015, pp. 14–27. [Online]. Available: http://doi.acm.org/10.1145/2814228.2814232

[10] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003. [Online]. Available: http://dx.doi.org/10.1109/MC.2003.1160055

[11] T. Haitzer and U. Zdun, "Controlled experiment on the supportive effect of architectural component diagrams for design understanding of novice architects," in *Proceedings of the 7th European Conference on Software Architecture*, ser. ECSA'13. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 54–71. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-39031-9_6

[12] M. Javed and U. Zdun, "The supportive effect of traceability links in architecture-level software understanding: Two controlled experiments," in *Software Architecture (WICSA), 2014 IEEE/IFIP Conference on*, April 2014, pp. 215–224.

[13] E. Arisholm, L. C. Briand, S. E. Hove, and Y. Labiche, "The impact of uml documentation on software maintenance: an experimental evaluation," *IEEE Transactions on Software Engineering*, vol. 32, no. 6, pp. 365–381, June 2006.

[14] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A systematic survey of program comprehension through dynamic analysis," *Software Engineering, IEEE Transactions on*, vol. 35, no. 5, pp. 684–702, Sept 2009.

[15] B. Cornelissen, A. Zaidman, and A. van Deursen, "A controlled experiment for program comprehension through trace visualization," *Software Engineering, IEEE Transactions on*, vol. 37, no. 3, pp. 341–355, May 2011.

[16] R. J. Walker, G. C. Murphy, B. Freeman-Benson, D. Wright, D. Swanson, and J. Isaak, "Visualizing dynamic software system information through high-level models," in *Proceedings of the 13th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, ser. OOPSLA '98. New York, NY, USA: ACM, 1998, pp. 271–283. [Online]. Available: http://doi.acm.org/10.1145/286936.286966

[17] B. Cornelissen, D. Holten, A. Zaidman, L. Moonen, J. van Wijk, and A. van Deursen, "Understanding execution traces using massive sequence and circular bundle views," in *Program Comprehension, 2007. ICPC '07. 15th IEEE International Conference on*, June 2007, pp. 49–58.

[18] C. Gravino, G. Scanniello, and G. Tortora, "Source-code comprehension tasks supported by uml design models," *J. Vis. Lang. Comput.*, vol. 28, no. C, pp. 23–38, Jun. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.jvlc.2014.12.004

[19] ÃĄlvaro JimÃl'nez, J. M. Vara, V. A. Bollati, and E. Marcos, "Metagem-trace: Improving trace generation in model transformation by leveraging the role of transformation models," *Science of Computer Programming*, vol. 98, Part 1, pp. 3 – 27, 2015, fifth issue of Experimental Software and Toolkits (EST): A special issue on Academics Modelling with Eclipse (ACME2012). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167642314003700

[20] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering," *IEEE Trans. Softw. Eng.*, vol. 28, no. 8, pp. 721–734, Aug. 2002. [Online]. Available: http://dx.doi.org/10.1109/TSE.2002.1027796

[21] C. Wohlin, M. Höst, and K. Henningsson, "Empirical research methods in software engineering," in *Empirical Methods and Studies in Software Engineering*, ser. Lecture Notes in Computer Science, R. Conradi and A. Wang, Eds. Springer Berlin Heidelberg, 2003, vol. 2765, pp. 7–23. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-45143-3_2

[22] C. J. V. Rijsbergen, *Information Retrieval*, 2nd ed. Newton, MA, USA: Butterworth-Heinemann, 1979.

[23] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3-4, pp. 591–611, 1965. [Online]. Available: http://biomet.oxfordjournals.org/content/52/3-4/591

[24] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *Ann. Math. Statist.*, vol. 18, no. 1, pp. 50–60, 03 1947. [Online]. Available: http://dx.doi.org/10.1214/aoms/1177730491

[25] W. S. Gosset, "The probable error of a mean," *Biometrika*, vol. 6, no. 1, pp. 1–25, March 1908, originally published under the pseudonym "Student". [Online]. Available: http://dx.doi.org/10.2307/2331554

[26] T. Cook and D. Campbell, *Quasi-experimentation: design & analysis issues for field settings.* Rand McNally College, 1979. [Online]. Available: https://books.google.at/books?id=68HynQEACAAJ

[27] D. Jayathilake, "Towards structured log analysis," in *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*, May 2012, pp. 259–264.