Model-Integrated Queries for the Analysis of Runtime Events: A Controlled Experiment

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

Models describe a software system on an abstraction level higher than its actual implementation. Recent research results show that bringing models and a running system closer together by establishing traceability links between recorded runtime events and corresponding model elements improves the analysis performance of human observers when assessing the behaviour of the running system. Despite these results, common techniques for analyzing runtime events are rarely integrated into the models that are used for assessing the system behaviour from a high-level perspective. This paper presents a controlled experiment where model-integrated analysis facilities are compared with a more traditional analysis approach based on SQL queries to a system's database in terms of correctness and completion time of analysis tasks. The results show that model-integrated analyses allow analysts to give more correct answers to questions about the system behaviour, but provide no improvement of the time spent for completing the analysis tasks.

CCS CONCEPTS

• **General and reference** → *Empirical studies*; • **Software and its engineering** → *Model-driven software engineering*.

KEYWORDS

analysis, events, experiment, models, runtime

ACM Reference Format:
Michael Szvetits and Uwe Zdun. 2019. Model-Integrated Queries for the Analysis of Runtime Events: A Controlled Experiment. In *Evaluation and Assessment in Software Engineering (EASE '19), April 15–17, 2019, Copenhagen, Denmark.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3319008.3319010

1 INTRODUCTION

The level of abstraction in software engineering is steadily increasing, which can be observed in the adoption of model-driven techniques both in academa and industry [26]. These techniques promise to improve communication, control, reaction to changing requirements, productivity and maintainability [11].

Recent research utilizes models not only during the design phase of a software project, but also at runtime as additional input for the running system to allow the system to reflect on its own structure more effectively [1]. This enables the analysis of running systems on an abstraction level which is closer to the problem space [1].

A previously conducted experiment provides evidence that linking models to the output of a running system improves the analysis performance of human observers when assessing the behaviour of the system [24]: Two groups of people analyzed recorded runtime events with the help of simple textual search capabilities, whereas only one group was additionally able to utilize traceability links between the recorded runtime events and the model elements they originate from. The group using the traceability links achieved a higher correctness of answers to questions that target the comprehension of the system behaviour.

A post-experiment discussion revealed that the participants could imagine that they would achieve a higher correctness of the given answers if they would be able to utilize more powerful search and filter techniques than the ones provided in the experiment. In the experiment, using only simple search and filter techniques was essential in order to measure only the effects of the traceability links. However, a question that could not be answered is how providing more powerful search and filter techniques would influence the results. Hence, we investigate this question in this paper.

That is, this paper complements previous research results by conducting another controlled experiment where the correctness and completion time of eight questions concerning recorded runtime information are assessed. The goal of the experiment was to find out if traceability links between model elements and runtime events still improve the analysis capabilities of a human observer if well-established search and filter techniques are provided.

56 students with solid programming, modeling and design experience were separated in two groups of equal size and had to analyze runtime events yielded by a robot system. One group, the control group, was able to utilize SQL and its well-established filter and aggregation operations to analyze the runtime events. The other half of the participants, the experiment group, was able to annotate graphical model elements with queries and thus directly utilize the traceability links associated with an annotated model element for the analysis of its related runtime events. Such model-integrated queries were formulated in a custom query language inspired by stream-based processing features of mainstream programming languages [2, 12, 18]. The results show that the experiment group gives more correct answers to questions about the system behaviour, but the results show no significant difference between the two groups with respect to the completion time of the given analysis tasks.

2 BACKGROUND: MODEL-BASED ANALYSIS

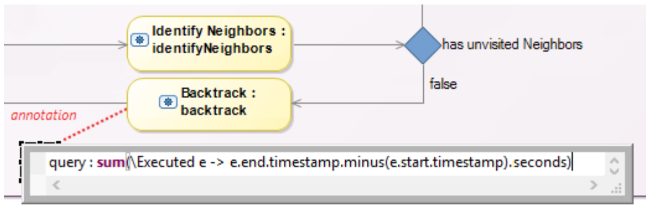
Consider a scenario where a user wants to analyze some high-level property of a software system, like the overall runtime that is spent within a modeled component. The goal of model-based analysis is to monitor runtime events that are relevant for the model elements of interest (e.g., the start and end events of operations in the component) and to perform the desired analysis based on those events (e.g., calculating the average runtime using the time stamps of the start and end events).

The actual filtering and aggregation of runtime events is often done by storing them in a database and using well-established query languages, most notably the Structured Query Language (SQL). An alternative is the adoption of complex event processing which is specialized in handling time series of events and data. Such complex event processing systems come in three flavours [5]: Publish-subscribe-based systems with high performance and scalability [6], stream databases [5, 19] and pattern matching based systems [28, 29]. These approaches have in common that their underlying query languages are dialects or extensions of SQL.

Although the runtime events are conceptually tightly integrated with the model elements they originate from, the mentioned query techniques to filter and aggregate them are usually not. In other words, they do not utilize traceability links between the model element of interest and its associated runtime events, and rather shift the burden of relating them to the analyst manually or in the formulated queries (e.g., by joining the relevant information in a SQL statement). The experiment presented in this paper addresses this fact: We provided SQL-based access to the runtime events for the control group, as requested in the discussion that followed our previous experiment [24], and equipped the experiment group with a custom language that allowed the participants to formulate equivalently powerful queries directly based on the models and utilize the traceability links in a more direct way. Since the common event analysis techniques are somehow related to SQL, it seemed only natural to compare the model-integrated query approach with the SQL-based analysis method.

The custom language used by the experiment group considers the events related to a certain model element as a stream and allows to filter and aggregate this stream in a concise way that can directly be integrated into graphical modeling environments. The provided filter and aggregation operations are inspired by the functional programming paradigm that is gradually integrated into mainstream object-oriented programming languages (e.g., LINQ in C# [2], streams and lambda expressions in Java [18] and closures in C++ [12]). Such a model-integrated query language can be used to analyze the behaviour of a running system on the model level independent from the concrete meta-model [23].

An example query formulated with the help of the model-based query language is shown in Figure 1. In the shown case, the analyst is interested in the events of type *Executed* that belong to the UML action named *Backtrack*. Events have certain properties that can be accessed and are used in this case to calculate the runtime of a single execution of the annotated UML action, measured in milliseconds. The runtime of all such events is summed up to obtain the overall runtime that was spent in the annotated UML action while the observed system was up and running. The result of the query is



```
query : sum(\Executed e -> e.end.timestamp.minus(e.start.timestamp).seconds)
```

Figure 1: Sum of events using a model-integrated query

shown next to the query (not depicted in Figure 1), so the analyst does not need to leave the modeling environment for the analysis.

The language provides various other operations to consecutively transform, filter, group and aggregate such event streams to obtain the desired result. Details of the language (e.g., its grammar definition) and the materials that were handed to the participants of the experiment (e.g., the database schema used by the control group) are openly available, as described in the next section.

3 EXPERIMENT DESCRIPTION

The setup of the experimental conditions is based on the experiment design guidelines of Kitchenham et al. [16] who describe recommended techniques for participant selection, allocation of participants to groups and minimization of bias throughout the experiment. For the statistical analysis, we relied on the recommendations of Wohlin et al. [27] to describe the involved independent and dependent variables, present the collected data, test our hypotheses and report potential validity concerns. However, we diverge from the exemplary statistical tests mentioned by Wohlin et al. [27] and instead apply more robust statistical methods to compare our results, as advised in the recent guidelines on robust statistical methods for empirical software engineering by Kitchenham et al [15]. The resources of the experiment are available online[1].

3.1 Goal and Hypotheses

The goal of the experiment was to find out if a direct integration of concise event query facilities into the models of a system (and thus, the utilization of its associated traceability links) can improve the analysis capabilities of human users if manual intervention is inevitable for investigating runtime phenomena. The improvement of the analysis capabilities was measured via the correctness of answers to questions about the observed system whose behaviour was captured using an event log. The participants of both the control group and the experiment group were provided with UML models of different types and abstraction levels of a robot system. UML was a natural choice for this experiment because it is the de-facto standard for modeling the structure and behaviour of a system and the participants were familiar in interpreting the various UML diagrams. More precisely, the following diagrams were provided:

- A use case diagram with 47 model elements for analyzing high-level scenarios of the software under observation.
- A component diagram with 19 model elements for analyzing the high-level architecture of the software.

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

- A package diagram with 8 model elements to support a fine-grained analysis of one of the components.
- A class diagram with 122 model elements (15 classes with an average of 6 child elements and 3 associations per class) showing the structure of a central part of the software.
- A state diagram with 49 model elements showing the states the system can remain in during execution.
- An activity diagram with 36 model elements showing a complex process in an implementation-agnostic way.

A second goal of the experiment was to measure the difference between the control group and experiment group with respect to the time they spent for analyzing the behaviour of the observed system. For each analysis task, participants were instructed to write down the time they spent for the respective task as soon as they perceive their answers to be complete.

3.1.1 Hypotheses. Based on the goals of the experiment, we formulate two null hypotheses and corresponding alternative hypotheses for the experiment:

- H_{01}: Model-integrated queries *do not significantly improve* the correctness of given answers about the system behaviour.
- H_{A1}: Model-integrated queries *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 queries are integrated into models.
- H_{A2}: The times spent for analyzing the system behaviour *significantly differ* if queries are integrated into models.

3.1.2 Expectations. We expected that the direct integration of concise query facilities into the models of the observed system enables the analyst a more direct way of selecting the right model elements and formulating the necessary query expressions for the analysis task at hand. As a consequence, we expected the experiment results to indicate the rejection of the null hypothesis H_{01} and thus the acceptance of the alternative hypothesis H_{A1}, i.e. model-integrated queries significantly improve the correctness of given answers about the system behaviour.

Moreover, we expected that the direct integration of queries into the models of the observed system allows the analysts of the experiment group to complete the analysis tasks faster than the analysts of the control group who utilize more traditional methods. As a consequence, we expected the experiment results to indicate the rejection of the null hypothesis H_{02} and thus the acceptance of the alternative hypothesis H_{A2}, i.e. the times spent for analyzing the system behaviour significantly differ if queries are integrated into models. Note that the second pair of hypotheses is formulated in a two-sided way, which means that the alternative hypothesis is also accepted if the experiment group actually performs worse than the control group with respect to the completion time of analysis tasks. This allows us to detect the effect if our provided query facilities are not as accessible as we thought they are.

3.2 Parameters and Variables

Table 1 gives an overview of the observed variables and describes their scales, units and value ranges. Independent variables were observed once per participant. 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, 150]
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

3.2.1 Dependent Variables. We created eight questions about the observed system for which the correctness of given answers is assessed. Each question instructs the participant to create a list of distinct elements (e.g., a list of error messages) which constitute the answer. We intentionally avoided open-ended questions because the assessment of free text answers inherently introduces a certain amount of bias based on the researcher who actually performs the assessment. Collecting a distinct list of elements per answer allowed us to objectively 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 [20]. Let $R_{p,q}$ be the set of elements mentioned by participant p for question q, and C_q be the expected elements in the solution for question q, then these metrics are:

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

Precision is the fraction of mentioned elements that are correct. Recall is the fraction of expected elements that were actually found [20]. These two metrics are combined using the harmonic mean to compute the so-called F-measure, our metric for the correctness whose value range is $[0, 1]$ where 0 denotes the worst and 1 the best quality of an answer:

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

Beside answering the questions, participants were instructed to write down the start time when beginning to tackle a question, as well as the end time when they feel that their answer is complete. They were allowed to write down multiple pairs of start and end times per questions so they can return to a question at a later time. The spent time per question can then be calculated by summing up the differences between such paired start and end time entries. The maximum time for the overall experiment was 150 minutes.

3.2.2 Independent Variables. Table 1 shows the four independent variables which were captured during the experiment and potentially influence the outcome of the dependent variables. We mitigated the influence of the different types of experiences on the dependent variables by conducting the experiment with participants that have similar education. We also formulated the questions in a way such that a potential domain experience of participants provides no significant advantage for answering the questions.

3.3 Experiment Design

We conducted the experiment in a course concerning software architectures and adaptive software systems at the University of Applied Sciences Wiener Neustadt in the winter semester 2018.

3.3.1 Subjects. The subjects of the experiment were 56 students with solid programming experience and knowledge in software and data(base) modeling. The subjects were randomly assigned to two groups of equal size, a control group and an experiment group.

3.3.2 Object. The participants of the experiments had to analyze the events of a robot system that was designed and implemented independently by five novice software architects two years before the idea of the experiment emerged. The robot is able to calibrate itself, follow a user-defined path, receive directions from an external operator, solve tasks with predefined strategies and discover its environment on its own to build a grid-based map of its surroundings. The researchers of the experiment were not directly involved in the construction of the system, but merely adjusted the formats and representations of the produced project artefacts so they can be used for the experiment (e.g., the models that were created during the realization of the system were converted into the format of the modeling environment where the query facilities are integrated). A more detailed description of the system and the produced artefacts can be found in the aforementioned online resources. The system was chosen for various reasons:

- We had access to the produced artefacts (including traceability links) of the project, thus allowing us to convert and distribute the required artefacts.
- With 4195 lines of code, the project is small enough for participants to comprehend in an experimental setting. However, the participants were not aware of the simple complexity because the source code was not provided to them.
- The system can easily be explained during the experiment.
- We considered a robot and its physical interactions as a motivating system under observation for the participants.
- The project provided six models created with UML, a modeling language the participants had experience with. Since the models were not created entirely by the researchers, potential bias could be reduced as much as possible.

3.3.3 Instrumentation. The participants of both the control and the experiment group received a prepared Eclipse instance showing the six UML models of the robot system using the Obeo UML Designer. Participants of the experiment group were able to directly annotate the elements of those models with the desired queries using the language presented in Section 2 and observe the results. This feature was disabled in the Eclipse instances of the control group. Instead, the control group was provided with a browser-based interface which allowed the typing of arbitrary SQL queries in a text box, submitting the queries and observing the results. The query editors of both groups provided similar features in terms of syntax highlighting, code completion and error reporting.

Before the experiment started, the participants were introduced for an hour to their available materials to perform queries on the events (including using the browser-based interface for the control group, and formulating model-integrated queries for the experiment group). Example queries presented during this training phase were not related to the upcoming experiment in any way.

The queries of both groups operated on the same set of runtime events. These events were recorded before the experiment by letting the robot perform its functions and writing 154.797 events to an event log. The large amount of log file entries ensures that the asked questions could not be answered by exhaustively going through the log entries, but instead by cleverly applying the provided filtering and aggregation capabilities. Regarding the types of recorded events, we relied on a set of reusable event types [23] and stored the events as serialized Java objects in the event log. We then inserted the recorded events into a MySQL database whose schema was a normalized relational equivalent to the object-oriented Java meta-model of the recorded events, which is shown in Figure 2. Figure 3 shows an example SQL query that demonstrates how some elements of the meta-model were mapped to database (cross-)tables who must be joined together to perform an analysis. Note that the demonstrated SQL query is equivalent to the model-integrated query shown in Figure 1. The exact database schema used by the control group can be found in the aforementioned online resources. The control group accessed the database for their queries, while the experiment group accessed the events of the event log directly.

Each participant received a questionnaire to be answered during the experiment. The first two pages of the questionnaire contained questions regarding the independent variables (programming experience, programming experience in industry, and software design experience) and a short summary of the provided query facilities. For the control group, this summary contained a database schema diagram of the stored events. Although the participants of the control group were proficient in SQL and other database-related technologies, they were permitted to make use of the online MySQL language reference to look up the available SQL expressions. For the experiment group, the summary contained the event meta-model and the grammar of the query language. The summaries contained no information that has not already been presented during the training phase and were merely designed as reminder for the participants of how to formulate queries. The third page of the questionnaire contained the actual questions to be assessed for correctness and completion time, as summarized in Table 2.

Question *Q1* requires the participants to count a specific set of events. The challenge for the control group was to join the required tables and filter only the calls that originate from the component *lego.robot* and target elements of the component *lego.path*. The challenge for the experiment group was to identify the dependency relationship between the communicating components in the component diagram and annotate the respective query.

Question *Q2* has similar complexity to question *Q1* but requires both groups to integrate arithmetic operations into their queries (more precisely, the difference of timestamps between start and end events of operations to calculate the average runtime). The annotated model element could be found in the package diagram.

Questions *Q3* requires the participants to concatenate data from multiple filtered events. While SQL provides concatenation out of

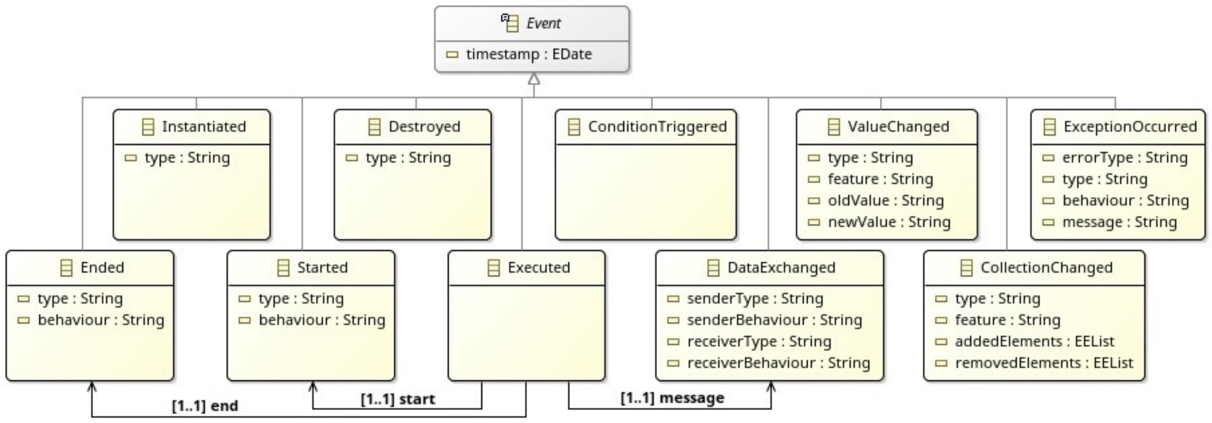


Figure 2: Meta-model of the events that were recorded and analyzed by the participants of the experiment

```sql
SELECT SUM(TIMESTAMPDIFF(SECOND, se.timestamp, ee.timestamp))
FROM EventModelElement eme
  INNER JOIN ModelElement me ON eme.modelElementID = me.ID
  INNER JOIN Event e ON eme.eventID = e.ID
  INNER JOIN Executed ex ON e.ID = ex.eventID
  INNER JOIN Started st ON ex.startedID = st.ID
  INNER JOIN Ended en ON ex.endedID = en.ID
  INNER JOIN Event se ON st.eventID = se.ID
  INNER JOIN Event ee ON en.eventID = ee.ID
WHERE me.name = "backtrack"
```

Figure 3: Example SQL query that is equivalent to Figure 1

Table 2: Self-developed Questions of the Experiment

ID	Description
Q1	Analyze the coupling of the components *lego.robot* and *lego.path* by measuring the count of calls between them.
Q2	Measure the average runtime of methods within package *lego.robot.behaviour*.
Q3	What exceptions did occur in package *lego.robot.behaviour* (give a comma-separated list of all the exception messages).
Q4	Calculate the average amount of cells per grid. You can achieve this by dividing the respective instantiation counts.
Q5	Measure how often the class *Arbiter* is accessing each subtype of the interface *Behaviour* (hint: use grouping).
Q6	What was the rightmost cell of a grid? Give the column index and the timestamp when the cell got its highest column value.
Q7	How many percent of the given tasks could the robot solve automatically?
Q8	Measure the percentage of time that backtracking takes in relation to the whole discovery activity.

the box, the experiment group had to fold over the filtered data and concatenate the error messages in a programmatic way.

Questions *Q4*, *Q7* and *Q8* are challenging for both groups since they require the participants to relate filtered events that belong to multiple model elements. For the experiment group, this means to annotate multiple model elements and carefully formulate the query on multiple streams of events.

Question *Q5* requires the participants to perform grouping in their queries according to the receiver type of data exchanges between two classes. Like in question *Q3*, participants of the experiment group must fold over the resulting groups to get the answer.

Question *Q6* requires the participants to filter events that describe the change of values and aggregate them according to the desired property that has been changed. The property must be identified using the class diagram that was provided to the participants.

3.4 Experiment Execution

The participants were randomly assigned to the control and experiment group. After assignment, the groups consisted of 28 participants each. The robot system, the available query techniques and the upcoming tasks (but not the experiment questions themselves) were explained to the participants in the aforementioned training phase. The training phase also included a time slot of 15 minutes for the participants to get familiar with the provided query facilities. None of the participants knew the observed system.

After the training phase, the questionnaires were handed to the participants and the questions explained. The time limit of 150 minutes was communicated to the participants after the explanation of the questionnaire. All participants had exactly the same type of computer on which the modeling environments were prepared before the experiment. The usage of private computers was prohibited during the experiment. The questionnaire had to be filled out directly on the questionnaire paper.

The experiment was completed without deviations from the experiment design. No participants dropped out prematurely before the end of the experiment. The experiment was conducted in two lecture rooms containing the prepared computer systems. One experimenter was stationed per room to prevent participants from using forbidden materials and resolve potential questions.

After the time limit, the experimenters collected the questionnaires and offered the participants an informal discussion about the conducted experiment to obtain additional feedback. The discussion had no influence on the results presented in this paper, but helped us to track down potential weaknesses in the experiment design and to understand the various solution strategies used by the participants which were not written down on the questionnaires.

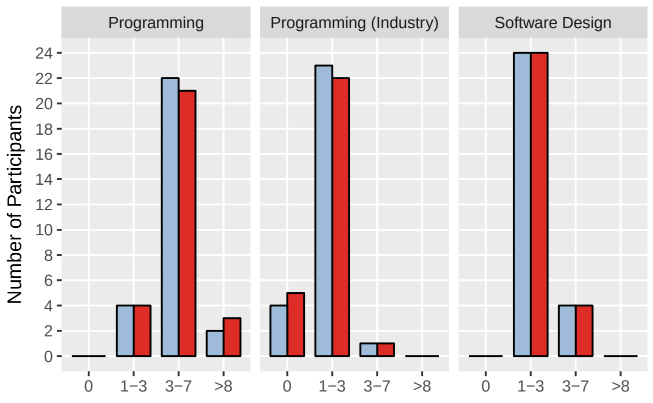


Figure 4: Experiences of the control group (left bars, blue) and the experiment group (right bars, red) in year intervals.

Table 3: Statistical Tests for the Correctness (F-measure)

ID	Control Group Correct	Control Group Incorrect	Experiment Group Correct	Experiment Group Incorrect	p-Value
Q1	18	10	25	3	**0.02751**
Q2	8	20	19	9	**0.00347**
Q4	17	11	26	2	**0.00477**
Q7	13	15	18	10	0.14112
Q8	7	21	15	13	**0.02716**
	Mean	Std.Dev.	Mean	Std.Dev.	
Q3	0.66045	0.38515	0.62644	0.40428	0.43703
Q5	0.35799	0.44687	0.50567	0.48719	0.08474
Q6	0.40476	0.48311	0.42857	0.50395	0.41166
All	0.45915	0.48226	0.65491	0.46344	**2.45e-06**

4 EXPERIMENT RESULTS

4.1 Descriptive Statistics

Figure 4 shows the level of experience as declared by the participants in the questionnaire. The random assignment of the participants to the control and experiment groups led to an almost perfect separation with respect to their experience levels. Based on this data, we believe that none of the groups could draw a significant advantage from the random assignment of the participants. Note that the software design experience also encompasses database design experience, which is important for the control group.

Figure 5 shows the correctness of answers and the completion times that were achieved by the participants. Since questions *Q1*, *Q2*, *Q4*, *Q7* and *Q8* required the participants to give a single answer, the F-measure for the correctness is effectively either 0 (wrong answer) or 1 (correct answer). As a consequence, we reported the number of correct answers per group instead of the mean F-measures for those questions. The figure shows that the experiment group achieved a higher correctness for all questions except *Q3*, although not significantly for all questions. The control group especially struggled with questions *Q2* and *Q8* compared to the experiment group.

A comparison of the completion times in Figure 5 shows a diverse picture. The control group needed more time to find the answers for questions *Q1* and *Q2*, but completed the questions *Q3* and *Q6* faster than the experiment group. Both groups achieved similar results for the rest of the questions. Note that the completion time is not an indicator for the quality of an answer: Participants were not instructed to answer the questions as fast as possible, but instead to write down the times if they feel that their answers are complete.

4.2 Handling of Outliers

All participants provided reasonable answers for the questions. If an answer to a question was completely missing, the correctness was assumed to be zero since the participant could not find the correct answer. Regarding the completion time, we could only assess time entries that were actually written down on the questionnaire (i.e., if a participant skipped a question and did not write any time down, we could not include it in our results). All in all, we excluded none of the 56 participants while compiling the results in this paper.

4.3 Hypothesis Testing

We tested the hypotheses formulated in Section 3.1.1 by using robust statistical methods, as advised in the recent guidelines by Kitchenham et al [15]. These methods do not assume a specific distribution of the data and which are solid alternatives to traditional methods of comparing means like the t-test [8] or the Wilcoxon Rank-Sum Test [17] which rely on a preceding step of analyzing the normality of data. We used a significance level of $\alpha = 0.05$ for all our tests. We used the R programming language and its accompanied packages *gdata*, *ggplot2*, *orddom* and *reshape2* to obtain the results shown in this paper. The R script can be found in the online resources.

For comparing the correctness between the control and the experiment groups, we applied two different methods: Since questions *Q1*, *Q2*, *Q4*, *Q7* and *Q8* only distinguish between correct and wrong answers, we applied one-tailed Fisher's exact tests [7] to compare the proportions of correct answers for those questions. The one-tailed Fisher's exact test works reasonable well for all sample sizes and states in its null hypothesis that the proportion of correct answers achieved by the control group is equal to or greater than the proportion achieved by the experiment group.

For all other questions, we relied on Cliff's δ [3], a non-parametric test with solid performance for all sample sizes whose null hypothesis states that the control group achieved a correctness that is equal to or greater than the correctness achieved by the experiment group. We also applied Cliff's δ when comparing the completion time, but as a two-tailed test to reflect the structure of the hypotheses H_{02} and H_{A2} presented in Section 3.1.1.

The results of comparing the correctness is shown in Table 3. Numbers written in bold indicate that the p-value of a test is lower than the significance level, thus suggesting to reject the null hypothesis and assume that the experiment group achieved a significantly higher correctness than the control group. This is the case for the questions *Q1*, *Q2*, *Q4* and *Q8*. We also applied Cliff's δ to all questions together to obtain a comparison of the correctness for the whole experiment. This result also indicates a significantly better performance of the experiment group.

The results of comparing the completion time is shown in Table 4. Numbers written in bold indicate that the p-value of a test is lower than the significance level, thus suggesting to reject the

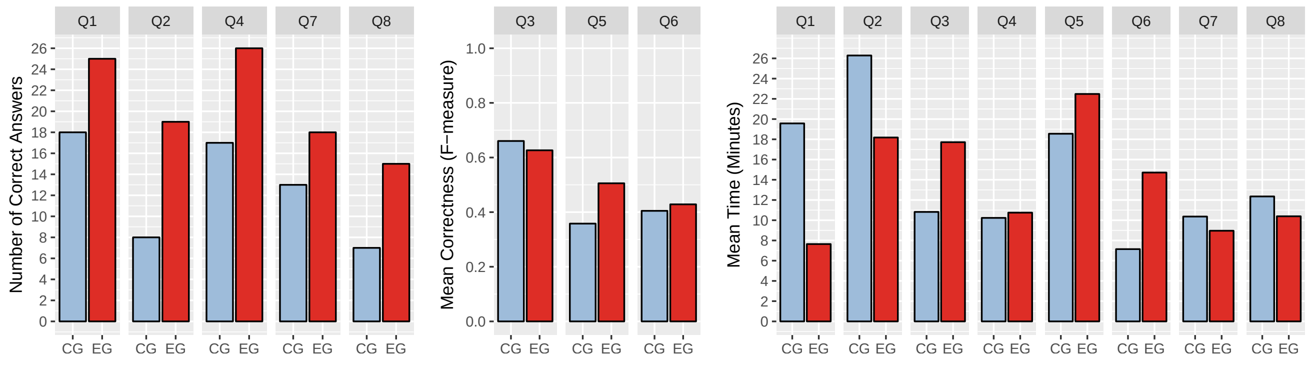


Figure 5: Correctness and mean time per question achieved by the control group (CG) and the experiment group (EG).

Table 4: Statistical Tests for the Time (Minutes)

ID	Control Group Mean	Control Group Std.Dev.	Experiment Group Mean	Experiment Group Std.Dev.	p-Value
Q1	19.57143	13.40378	7.64286	5.70389	**3.49e-08**
Q2	26.28571	11.28046	18.17857	9.87722	**0.00089**
Q3	10.82143	6.17074	17.71429	11.09173	**0.00504**
Q4	10.23077	5.66609	10.75000	9.00463	0.45337
Q5	18.55000	10.33377	22.48148	11.35568	0.19162
Q6	7.14286	3.23103	14.71429	7.23286	**5.32e-07**
Q7	10.36364	7.71236	8.96154	6.64217	0.36033
Q8	12.35294	6.20424	10.39130	7.01776	0.19303
All	15.09290	10.76355	13.91388	10.00540	0.26102

Table 5: Cliff's δ for the Correctness and the Time

	F_{CG} vs. F_{EG}	$Time_{CG}$ vs. $Time_{EG}$
$p_1 = PS(X > Y)$	0.18708	0.51476
$p_2 = PS(X = Y)$	0.40521	0.03595
$p_3 = PS(X < Y)$	0.40770	0.44929
δ	0.22062	-0.06547
s_δ	0.04770	0.05816
z	4.62571	-1.12561
CI (low)	0.14083	-0.17823
CI (high)	0.29757	0.04899
p	2.45e-06	0.26102

null hypothesis and assume that the experiment group achieved a significantly different completion time than the control group. This is the case for the questions *Q1*, *Q2*, *Q3* and *Q6*. We also applied Cliff's δ to all questions together to obtain a comparison of the completion time for the whole experiment. This result shows no significant difference in the completion time per question. The missing difference of the completion times between the two groups can also be observed when looking at the density of the time needed for answering a question, as shown in Figure 6. The curves of the two groups almost overlap each other, so they behaved very similarly in terms of getting confident that their answers are complete.

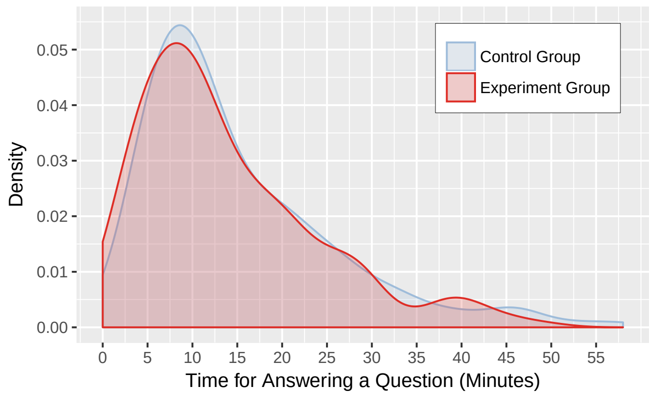


Figure 6: Density of the time for answering a question.

More insight into Cliff's δ for the correctness and the completion time over all questions is shown in Table 5 with the following values:

- p_1 is the probability that a participant randomly chosen from the control group has a higher score (i.e., correctness or completion time) than a randomly chosen participant from the experiment group.
- p_2 is the probability of equal scores between the two randomly chosen participants of the two groups.
- p_3 is the probability that the randomly chosen participant of the experiment group is superior.
- Cliff's δ, which is defined as $\delta = p_3 - p_1$.
- s_δ is the estimate of the standard deviation of Cliff's δ.
- z is the z-score of Cliff's δ.
- CI denotes the confidence interval of Cliff's δ.
- p is the probability of z, as already shown in Tables 3 and 4.

5 INTERPRETATION

Table 3 indicates that the experiment group performed significantly better than the control group for half of the questions. Although not significantly, the experiment group also performed better than the control group for the other questions as well, with question *Q3* being the exception. The experiment group also achieved a significantly better correctness than the control group if taking all questions together into account. As a result, we argue that the hypothesis

H_{01} has to be rejected and the alternative hypothesis H_{A1} holds: Model-integrated queries *significantly improve* the correctness of given answers about the system behaviour.

The control group especially struggled with finding the correct answers to the questions *Q2* and *Q8*. Question *Q2* required the control group to utilize specific SQL functions to obtain the time difference for the calculation of the average runtime. It is possible that participants applied the function in a wrong way, or struggled to look up the mechanics of the needed SQL functions in general, as indicated by the significant difference of the completion time for this question. Regarding question *Q8*, the control group might have struggled with formulating the joins needed for processing the events of multiple model elements. The experiment group could visually combine multiple event streams, which might have helped in this regard. This also applies to the questions *Q4* and *Q7* where the experiment group performed better (significantly for *Q4*).

Except the questions *Q2* and *Q8*, the results show no perfectly or exceptionally badly achieved correctness of one group compared to the other, so we think that the difficulty of the questions was feasible. Overall, it seems that the stream-based abstraction and its query language provides a better conceptual model for correctly assessing model-based events than its relational counterpart.

The comparison of the completion times shows that in the majority of cases there exists no significant difference between the control group and the experiment group. There are four questions that indicate a noticeable difference, but two of them are in favour of the control group and the other two in favour of the experiment group. This mixed result suggests that the null hypothesis H_{02} cannot be rejected: The times spent for analyzing the system behaviour *do not significantly differ* if queries are integrated into models.

This is against our expectation described in Section 3.1.2. It seems that the expected advantage of model-integrated queries is somewhat mitigated by the fact that participants had to apply filters and aggregations in an unfamiliar way. Interpreting the completion time of question *Q1* is rather vague, since the participants could have a familiarization phase for the first question beyond the regular training phase where they did not know the questionnaire beforehand (again, note that participants were not instructed to work as fast as they can). Nevertheless, the experiment group completed the question *Q1* significantly faster than the control group, likely because the annotation of the component dependency is faster than realizing how this model element manifests itself in the relational variant. As mentioned above, the difference for question *Q2* could result from the application of specific SQL functions.

For the questions *Q3*, *Q5* and *Q6*, it took the experiment group longer to gain confidence of their answers. The questions *Q3* and *Q5* require the experiment group to apply grouping and folding, two of the more complex operations to filter and aggregate events. Question *Q6* requires the experiment group to apply filtering, folding and a type conversion which must be formulated carefully. The solutions required by the control group for the questions *Q3*, *Q5* and *Q6* have predefined SQL counterparts for those operations. Overall, the control group and experiment group achieved similar results in terms of getting confident that their answers are complete.

6 THREATS TO VALIDITY

We analyze the validity of our research results in four dimensions [4]: *Construct validity* discusses the adequateness of the applied research techniques with respect to the goals of the experiment. *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* discusses the generalizability of the obtained research results to scenarios beyond the experimental setup. *Conclusion validity* refers to the statistical validity of conclusions that are drawn from the observed variables.

6.1 Construct Validity

A potential threat to validity is the selection of certain query mechanisms that were provided to the experiment participants. Regarding the experiment group, we argue that the provided query language meets the trend of integrating query features inspired by functional programming into mainstream programming languages (e.g., LINQ in C# [2], streams and lambda expressions in Java [18] and closures in C++ [12]) and that the language could be applied adequately using the programming knowledge of the participants. Regarding the control group, we argue that the capabilities of SQL are well-established for data analysis beyond the scope that was necessary to answer the questions in the experiment. SQL-based extensions like the ones provided by complex event processing approaches would require a deeper level of preliminary knowledge for the participants which cannot be guaranteed easily in an experimental setting.

In a realistic setup, the running system would provide a constant stream of events which are observed by the analyst. One could argue that the event log does not adequately represent such a scenario because it provides a fixed number of events that are available for the analysis. We argue that analyses of log files is still widely adopted [14] and that the difference between streaming and constantly writing to the event log is merely a question of performance. The questionnaire of the experiment was formulated in a way that is independent from the actual source of events, and the event recording performance was not one of the observed variables.

Another threat is that the questionnaire might not include questions that are relevant to real world situations. We argue that the questions are geared towards analysis tasks that often occur in practice, like fault analysis (see *Q3*), efficiency measurements (see *Q2*, *Q7* and *Q8*), dependability analysis (see *Q1* and *Q5*) and general domain analysis (see *Q4* and *Q6*). However, this threat cannot be eliminated completely since the questions in an experiment should be generalizable to realistic scenarios, but at the same time be constrained in a way that they can be answered by the participants under experimental conditions in a limited amount of time.

6.2 Internal Validity

A possible threat to internal validity is the missing compliance of participants with the rules of the experiment. We organized the experiment in a way such that at least one experimenter was always present while the experiment lasted, so we consider the threat of such misbehaviour going unnoticed as non-existent. Moreover, the experimenters regularly checked the questionnaires of the participants in an unobtrusive manner, e.g. to check if there are two noted start times without one of them having a noted end time.

Regarding the fact that the system under observation was implemented by software architects known to the university where the experiment took place, one could argue that there might have been some information leak between the developers of the observed system and the participants, or an introduction of bias by the researchers. This risk is negligible because the observed system was designed and implemented independently by the five novice software architects two years before the idea of the experiment even emerged. The researchers were not involved in the construction of the system, but merely adjusted the formats of the produced project artefacts so they can be used for the experiment. To further reduce researcher bias, we designed the anonymized questionnaire in a way such that every question requires a list of distinct answers which can easily be verified objectively with the help of the F-measure instead of a subjective grading system for the correctness.

One threat to validity that cannot be eliminated is the variation in human performance which results from fatigue effects while conducting the experiment. Since the experiment had a time limit of two and a half hours and the participants achieved a mean completion time of about 15 minutes per question (recall Table 4), we do not consider fatigue as a distorting factor: The extrapolated time for the eight questions is thus about two hours, meaning that the participants were able to complete the questionnaire without stress. Moreover, the participants assured in the post-experiment discussion that the time limit was not a hindering factor.

6.3 External Validity

Analyzing the events of only one system introduces the risk that the experiment results are specific to the examined case or the generalizability is limited to the examined domain. There is a trade-off between choosing a very generic system which hardly fits a domain and a system which covers a specific interesting domain that motivates the participants to carefully answer the given questions. We think that the robot system is motivating for the participants, while at the same time the majority of questions target analysis tasks which are not tied to the robotic domain. Moreover, the robot system is small enough for participants to comprehend in the given time, but at the same time produces enough events so that it is impossible for participants to answer the questions by just browsing through all the recorded events. However, a complete elimination of the threat can only be achieved by multiple experiments using systems of different sizes and domains.

Another threat is that only selected techniques were provided for the participants. We argue that the query language and its syntax used by the experiment group meets the trend of integrating functional programming features into mainstream languages and incorporates many features of other query languages like OCL and VQL. Regarding the control group, using SQL to query event data is a common approach in practice. The inclusion and comparison of SQL was actually suggested by the participants from our preceding experiment [24]. We believe that we achieved a fair comparison, however, a perfect generalization to arbitrary query tools and techniques is simply not possible in a single experiment.

The experiment participants were students with solid programming and design experience, but limited professional experience. We designed the questionnaire in a way such that the experience only plays a minimal role in answering the questions, but the potential threat to validity regarding the generalizability to analysts in general cannot be ignored. A replication of our study with different query mechanisms as well as analysts with varying levels of technical knowledge would provide additional insights, especially with respect to the adequateness of the proposed query language.

6.4 Conclusion Validity

A threat to validity might occur if the questionnaire can be answered very easily or is hardly impossible to solve by one of the groups. Our results show that the control and experiment groups neither achieved extraordinary good or bad results in relation to each other, which is a good sign that the questions enable a fair comparison between the groups. Distracting elements (i.e., possible wrong answers) in both the models and the recorded events ensured that the participants had to actively identify the required model elements and filter the 154.797 events to find the correct answers.

The amount of events also ensured that guessing or finding the right answer by chance was nearly impossible. The questions had no predefined answers to choose from, so participants were required to actively search for answers. It is highly doubtful that the statistical results were distorted by pure luck of the participants.

Regarding the statistical validity, we relied on objective techniques from information retrieval systems instead of subjective ad-hoc assessments by human analysts. Our selected statistical methods – Cliff's δ [3] and Fisher's exact test [7] – do not rely on specific assumptions of the data distributions, work for all sample sizes and are solid alternatives to comparing means using the t-test [8] or the Wilcoxon Rank-Sum Test [17]. However, although Fisher's exact test is applicable for all sample sizes, the statistical validity might be affected by the sample size of 56 participants. Replications of the study with practitioners and systems of different domain and size would strengthen the significance of our findings.

7 RELATED WORK

There exists various experimental evaluations that investigate the impact of traceability links on the comprehension of a software system. Shahin et al. [22] used architectural design decisions as traceability information from the problem space to the solution space. They performed a controlled experiment which indicated that visualizations of architectural design decisions (using the tool Compendium [21]) improve the correctness of understanding architecture design. Like in our experiment, they also concluded that the provided traceability mechanism does not increase the time needed to perform the analysis tasks. In contrast to our experiment, the analysis of the dynamic system behaviour is out of the scope.

Haitzer and Zdun [10] measured the supportive effect of architectural components diagrams for design understanding of novice architects. The results show that traceability links between components and their source code improve the comprehension of the system. Two other controlled experiments cocnluded that traceability links between architectural models and their corresponding source codes improved the comprehension of the observed system [13]. These experiments demonstrate the usefulness of traceability links for software comprehension, but the analysis of the dynamic system behaviour is not focused in these works either.

Tran et al. [25] propose a model-driven approach that supports traceability links between process development artefacts of service-oriented architectures at different levels of granularity and abstraction. The implemented framework promises improved change impact analysis, artefact understanding, change management and propagation, but does not utilize relationships between the model elements and the produced runtime information. Furthermore, their approach is fixed on the scope of service-oriented architectures.

However, there also exists research results in terms of understanding an observed system without the help of traceability links. Gravino et al. [9] assessed the comprehension of object-oriented source code if UML class and sequence diagrams are provided to the analyst. The results show an average improvement of 12% in solving the given comprehension tasks if the analyst is able to utilize the UML models that were created during the design phase of the system. Again, this result demonstrates that models help to analyze a software system, but does not indicate an improvement of the comprehension of the dynamic behaviour of a system.

8 CONCLUSIONS

In this paper we describe the design, execution and results of a controlled experiment whose goal was to find out if query facilities that are integrated into models can improve the analysis performance of human users when observing the behaviour of a system. The improvement of the performance was assessed by comparing the achieved correctness and completion time of a questionnaire between two groups, one using traditional database-based queries and one using the experimental model-integrated queries. The results provide evidence that model-integrated queries indeed allow analysts to give more correct answers to questions about the system behaviour, but provide no improvement of the time spent for completing the analysis tasks. We assume that the improvement of the completion times achieved by the experiment group is mitigated by the fact that filter and aggregation operations had to be applied in an unfamiliar way.

REFERENCES

[1] Gordon Blair, Nelly Bencomo, and Robert B. France. 2009. Models@ run.time. *Computer* 42, 10 (Oct. 2009), 22–27. https://doi.org/10.1109/MC.2009.326

[2] James Cheney, Sam Lindley, and Philip Wadler. 2013. A Practical Theory of Language-integrated Query. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming (ICFP '13)*. ACM, New York, NY, USA, 403–416. https://doi.org/10.1145/2500365.2500586

[3] Norman Cliff. 1996. *Ordinal Methods for Behavioral Data Analysis*. Erlbaum. https://books.google.at/books?id=bIJFvgAACAAJ

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

[5] Alan J. Demers, Johannes Gehrke, Biswanath Panda, Mirek Riedewald, Varun Sharma, and Walker M. White. 2007. Cayuga: A General Purpose Event Monitoring System. In *CIDR'07*. 412–422.

[6] Françoise Fabret, H. Arno Jacobsen, François Llirbat, João Pereira, Kenneth A. Ross, and Dennis Shasha. 2001. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems. *SIGMOD Rec.* 30, 2 (May 2001), 115–126. https://doi.org/10.1145/376284.375677

[7] Ronald Aylmer Fisher. 1925. *Statistical Methods For Research Workers*. Edinburgh Oliver & Boyd.

[8] William Sealy Gosset. 1908. The Probable Error of a Mean. *Biometrika* 6, 1 (March 1908), 1–25. http://dx.doi.org/10.2307/2331554 Originally published under the pseudonym "Student".

[9] Carmine Gravino, Giuseppe Scanniello, and Genoveffa Tortora. 2015. Source-code Comprehension Tasks Supported by UML Design Models. *J. Vis. Lang. Comput.* 28, C (June 2015), 23–38. https://doi.org/10.1016/j.jvlc.2014.12.004

[10] Thomas Haitzer and Uwe Zdun. 2013. 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 (ECSA'13)*. Springer-Verlag, Berlin, Heidelberg, 54–71. https://doi.org/10.1007/978-3-642-39031-9_6

[11] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. 2011. Empirical assessment of MDE in industry. In *Software Engineering (ICSE), 2011 33rd International Conference on*. 471–480. https://doi.org/10.1145/1985793.1985858

[12] Jaakko Järvi and John Freeman. 2010. C++ Lambda Expressions and Closures. *Sci. Comput. Program.* 75, 9 (Sept. 2010), 762–772. https://doi.org/10.1016/j.scico.2009.04.003

[13] Muhammad Atif Javed and Uwe Zdun. 2014. The Supportive Effect of Traceability Links in Architecture-Level Software Understanding: Two Controlled Experiments. In *Software Architecture (WICSA), 2014 IEEE/IFIP Conference on*. 215–224. https://doi.org/10.1109/WICSA.2014.43

[14] Dileepa Jayathilake. 2012. Towards structured log analysis. In *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*. 259–264. https://doi.org/10.1109/JCSSE.2012.6261962

[15] Barbara A. Kitchenham, Lech Madeyski, David Budgen, Jacky Keung, Pearl Brereton, Stuart Charters, Shirley Gibbs, and Amnart Pohthong. 2017. Robust Statistical Methods for Empirical Software Engineering. *Empirical Software Engineering* 22, 2 (01 Apr 2017), 579–630. https://doi.org/10.1007/s10664-016-9437-5

[16] Barbara A. Kitchenham, Shari Lawrence Pfleeger, Lesley M. Pickard, Peter W. Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. 2002. Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Trans. Softw. Eng.* 28, 8 (Aug. 2002), 721–734. https://doi.org/10.1109/TSE.2002.1027796

[17] Henry B. Mann and Donald R. Whitney. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *Ann. Math. Statist.* 18, 1 (03 1947), 50–60. https://doi.org/10.1214/aoms/1177730491

[18] HaiTao Mei, Ian Gray, and Andy Wellings. 2016. Real-Time Stream Processing in Java. In *Reliable Software Technologies – Ada-Europe 2016*, Marko Bertogna, Luis Miguel Pinho, and Eduardo Quiñones (Eds.). Springer International Publishing, Cham, 44–57.

[19] Rajeev Motwani, Jennifer Widom, Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Gurmeet Manku, Chris Olston, Justin Rosenstein, and Rohit Varma. 2002. *Query Processing, Resource Management, and Approximation in a Data Stream Management System*. Technical Report 2002-41. Stanford InfoLab. http://ilpubs.stanford.edu:8090/549/

[20] Cornelis Joost van Rijsbergen. 1979. *Information Retrieval* (2nd ed.). Butterworth-Heinemann, Newton, MA, USA.

[21] Mojtaba Shahin, Peng Liang, and Mohammad Reza Khayyambashi. 2010. Rationale Visualization of Software Architectural Design Decision Using Compendium. In *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC '10)*. ACM, New York, NY, USA, 2367–2368. https://doi.org/10.1145/1774088.1774577

[22] Mojtaba Shahin, Peng Liang, and Zengyang Li. 2011. Architectural Design Decision Visualization for Architecture Design: Preliminary Results of a Controlled Experiment. In *Proceedings of the 5th European Conference on Software Architecture: Companion Volume (ECSA '11)*. ACM, New York, NY, USA, Article 2, 8 pages. https://doi.org/10.1145/2031759.2031762

[23] Michael Szvetits and Uwe Zdun. 2015. Reusable event types for models at runtime to support the examination of runtime phenomena. In *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 4–13. https://doi.org/10.1109/MODELS.2015.7338230

[24] Michael Szvetits and Uwe Zdun. 2016. Controlled Experiment on the Comprehension of Runtime Phenomena Using Models Created at Design Time. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS '16)*. ACM, New York, NY, USA, 151–161. https://doi.org/10.1145/2976767.2976768

[25] Huy Tran, Uwe Zdun, and Schahram Dustdar. 2011. VbTrace: using view-based and model-driven development to support traceability in process-driven SOAs. *Software & Systems Modeling* 10, 1 (2011), 5–29. https://doi.org/10.1007/s10270-009-0137-0

[26] Jon Whittle, John Hutchinson, and Mark Rouncefield. 2014. The State of Practice in Model-Driven Engineering. *IEEE Software* 31, 3 (May 2014), 79–85. https://doi.org/10.1109/MS.2013.65

[27] Claes Wohlin, Martin Höst, and Kennet Henningsson. 2003. Empirical Research Methods in Software Engineering. In *Empirical Methods and Studies in Software Engineering*, Reidar Conradi and AlfInge Wang (Eds.). Lecture Notes in Computer Science, Vol. 2765. Springer Berlin Heidelberg, 7–23. https://doi.org/10.1007/978-3-540-45143-3_2

[28] Eugene Wu, Yanlei Diao, and Shariq Rizvi. 2006. High-performance Complex Event Processing over Streams. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD '06)*. ACM, New York, NY, USA, 407–418. https://doi.org/10.1145/1142473.1142520

[29] Detlef Zimmer and Rainer Unland. 1999. On the semantics of complex events in active database management systems. In *Data Engineering, 1999. Proceedings., 15th International Conference on*. 392–399. https://doi.org/10.1109/ICDE.1999.754955