Interactive Role Stereotype-Based Visualization To Comprehend Software Architecture

Truong Ho-Quang[1], Alexandre Bergel[2], Arif Nurwidyantoro[3], Rodi Jolak[1], Michel R. V. Chaudron[1]

[1] Chalmers & Gothenburg University, Sweden – ✉ {*truongh,jolak,chaudron*}@*chalmers.se*
[2] ISCLab, Department of Computer Science (DCC), University of Chile – *abergel@dcc.uchile.cl*
[3] Monash University, Australia – ✉ *arif.nurwidyantoro@monash.edu*

***Abstract*—*Motivation:* Software visualization can be helpful in comprehending the architecture of large software systems. Traditionally, software visualisation focuses on representing the structural perspectives of systems. In this paper we enrich this perspective by adding the notion of *role-stereotype*. This role-stereotype carries information about the type of functionality that a class has in the system as well as the types of collaborations with other classes that it typically has.**
***Objective:* We propose an interactive visualization called *RoleViz*, that visualizes system architectures in which architectural elements are annotated with their role-stereotypes.**
***Method:* We conducted a user-study in which developers use RoleViz and Softagram (a commercial tool for software architecture comprehension) to solve two separate comprehension tasks on a large open source system. We compared RoleViz against Softagram in terms of participant's: (i) perceived cognitive load, (ii) perceived usability, and (iii) understanding of the system.**
***Result:* In total, 16 developers participated in our study. Six of the participants explicitly indicated that visualizing roles helped them complete the assigned tasks. Our observations indicate significant differences in terms of participant's perceived usability and understanding scores.**
***Conclusion:* The participants achieved better scores on completing software understanding tasks with RoleViz without any cognitive-load penalty.**
***Demo:* https://youtu.be/HqCUAlai4qw?t=258**

I. Introduction

Software architecture visualization is a tool that can be used to understand complex software system. It can help developers maintain and further develop the system. In particular, it can be utilized to improve the search, navigation, and exploration of software architecture design [1][2].

In UML, stereotypes are a way to add complementary semantic information to the elements of a software design. Using such stereotypes in visualisation has been demonstrated to aid in the comprehension of software architectures. For instance, Genero et al. use object interaction stereotypes to improve the comprehension of UML sequence diagram [3]. Another example, Ricca et al. propose the use of web-specific notations to make UML applicable to model web application [4]. Beside those, a number of work focus on investigating the usefulness of class stereotype [5] to better understand UML class diagram [6][7][8][9].

The well-known class stereotypes, namely boundary, control, and entity, were introduced by Jacobson et al. as an extension to UML [5]. However, their definition of stereotypes is quite simple. Alternatively, Wirfs-Brock proposes role-stereotypes as the responsibilities that a class can have in an object-oriented system [10]. Some of both stereotypes are similar (e.g. *entity* and *information holder*), but Wirfs-Brock provides additional stereotypes beyond the class stereotypes. For example, a *service provider* is a class that performs work and offers services to others, which is not fit in any class stereotypes definition. To the best of our knowledge, no visualization tool has utilized role-stereotypes to help understand software architecture.

In this paper, we present RoleViz, a role-stereotypes-based visualization tool, and evaluate its usefulness to understand the architecture of an object-oriented system. We use the role-stereotypes [10] of a manually labeled ground-truth provided in our previous study [11]. The study presented in this paper shows the effectiveness of RoleViz to help developers in realistic software comprehension tasks.

Contributions. This paper makes the following contributions:

- We present RoleViz, an innovative visualization tool that overlay roles on top of a software architecture.
- We conduct a user study to investigate how RoleViz can help developers in real comprehension task, e.g. bug fixing.
- We compare the effectiveness of RoleViz against Softagram as our baseline. Softagram is a well-known software architecture visualization tool commonly used by software developers and architects.

Outline. The paper is structured as follows: Section II provides background of Wirfs-Brock's role stereotypes; Section III describes the RoleViz visualization; Section IV presents the research questions that leads our evaluation; Section V presents the user-study we conducted in order to answer the research questions; Section VI describes sources of data we collected and the methods for analysing the data; Section VII presents the result of our analysis; Section VIII discusses possible threats to the validity of this study; Section IX briefly presents the related work; Section X concludes and outlines our future work.

II. Role Stereotype

Our visualization is centered around the notion of *role* of an object-oriented class. Wirfs-Brock [10] identified six stereotypical role types that a class can play:

(CT) *Controller* makes decisions and control complex tasks;

(CO) *Coordinator* does not make many decisions, but in a rote or mechanical way, delegates work to other classes;
(IH) *Information holder* holds certain information and provides that information to others;
(IT) *Interfacer* transforms information and requests between distinct parts of a system. It can be a user interfacer class that interacts with users. An interfacer can communicate with external systems or between internal subsystems;
(SP) *Service provider* performs specific work and offers services to others on demand;
(ST) *Structurer* maintains relationships between classes and information about those relationships. Structurers might pool, collect, and maintain groups of classes.

It is noted that each class should play at least one role. There is a possibility where a class may carry more than one role. In this study we decided to only consider the primary responsibilities of the class as documented in the replication package of [11] where the authors attempted to classify role-stereotypes of a class automatically.

III. RoleViz

In this section, we use the K-9 Mail[1] application as the running example to illustrate RoleViz. K-9 Mail is an open source alternative mail application in Android. K-9 Mail is composed of 779 classes distributed in 52 different packages. K-9 Mail totals over 97 kLOC. Note that although K-9 Mail is written in Java, RoleViz is not tied to the Java programming language or Android platform.

A. RoleViz in a Nutshell

Figure 1 shows the use of RoleViz on K-9 Mail. RoleViz locates K-9 Mail's 52 packages in a circular fashion. Each package contains abstract classes, classes, enums, and interfaces. Each structural unit is colored according to the role it has.

Dependencies between two packages are represented with a *bimetric line* (number of dependencies are mapped to the size of the extremities, as described below). The package `k9` has classes heavily used in the system (indicated with tall inner colored boxes, marked with `A`), while `activities` has classes with outgoing dependencies variables (indicated with wide inner colored boxes, marked as `C`). Although, the application does not exhibit an architecture with crystal clear modularity boundaries, some tendencies may be visually inferred: for example, many packages depend on the package `k9`, while `k9` has relatively few external dependencies. Similarly, many packages depend on the package `mail`.

RoleViz is a polymetric view [12] in which software metrics are applied to visual dimensions, including height, width, and colors, as described below.

B. Compilation Unit

The source code in Java is organized as *compilation unit*, which is a technical jargon in Java to designate a definition contained in a `.java` file. We will, therefore, use this term along this paper to refer to a class, an enum, an abstract class, or an interface. Each unit is represented as a colored box, contained in a package.

Figure 2 details the visual representation of a compilation unit. The visual representation of a unit U uses two metrics:

1) the height of a unit represents the fan-in, *i.e.,* number of units that depends on U;
2) the width of a unit represents the fan-out, *i.e.,* number of units that U depends on.

To illustrate this visual technique, consider the following source code:

```
class Mail {
  Service c = new Service();
  Theme t = new DefaultTheme();
}
class Service { }
interface Theme {}
class DefaultTheme implements Theme {}
```

Four compilation units are defined, three classes and one interface. The class `Mail` depends on two other classes, `Service` and `DefaultTheme`. Assuming a closed-world assumption, the fan-out of `Mail` is therefore 2, and its fan-in is 0. The class `Service` has 1 as fan-in and 0 as fan-out. The classes `Mail` and `Service` are represented as two boxes as shown in Figure 2. The visual representation indicates that the class `Mail` has a fan-out greater than `Service` and has about the same the fan-in. This example is contrived, however it highlights the ability to visually describe some important aspects of the represented units.

The shape of the box is, therefore, an indicator for visually spotting *exceptional entities* [13]. For example, in the K-9 Mail example (Figure 1), one can recognize classes with a high fan-in value (marked as `A` and `B` in the figure) and a high fan-out (`C`). The visual shape is not meant to give an accurate value of the associated metrics, but instead, to give an idea of where significant visual differences lay in the visualization. As indicated below, in Section III-D, the visualization offers a number of interactions to obtain details about exact numerical values and offer numerous options to drill-down complementary information. A unit color indicates its role.

Edges between units indicates dependencies between these units. To not overload the visualization, edges are presented as bidirectional (*i.e.,* one cannot distinguish a caller from a callee). Hovering the mouse above a unit highlight callers and callees, as described below, as described in in Section III-D.

C. Package

Figure 3 details the representation of a package. A package is represented as a labeled gray box. The label, located above the gray box, is extracted from the name of the represented Java package.

The gray box contains inner colored boxes, representing the compilation units contained in the package. Units having dependencies between them are located on the right hand-side using a force-based layout (*i.e.,* units are assimilated as repulsing magnets and edges as springs, `D` in Figure 3). Note

[1]K-9 Mail's homepage: https://k9mail.app/

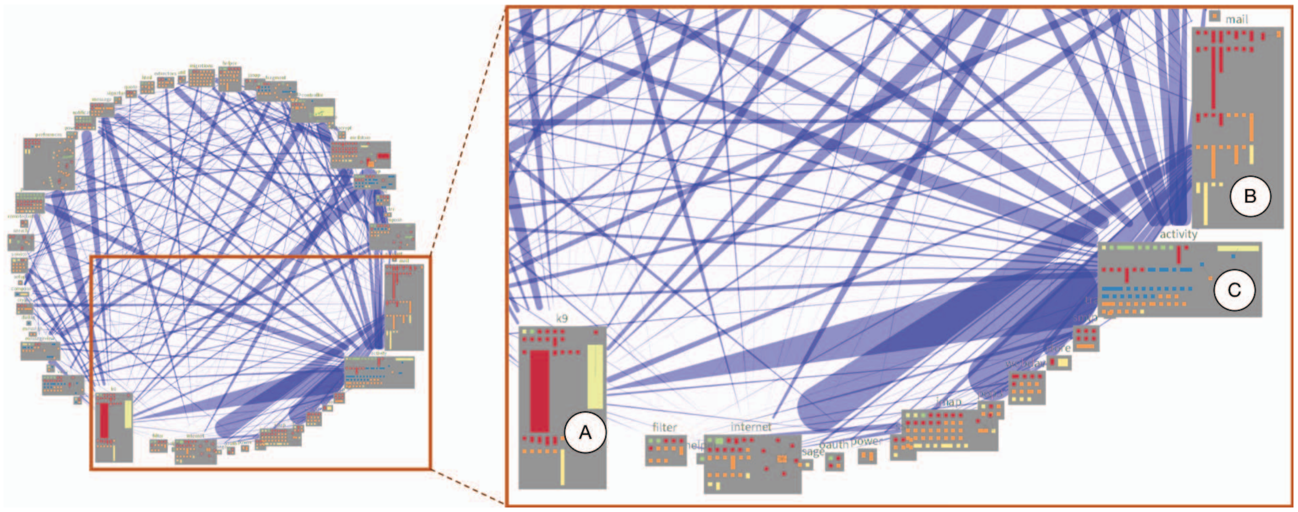


Fig. 1: Example of RoleViz

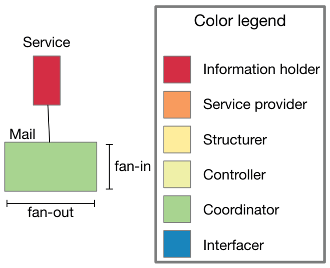


Fig. 2: Compilation unit detail

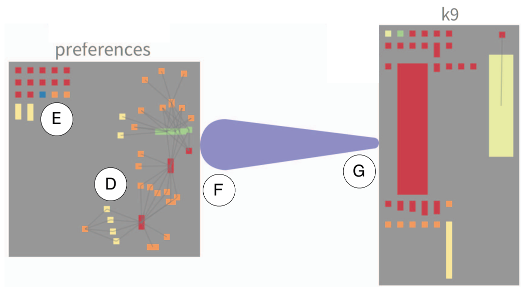


Fig. 3: Package detail

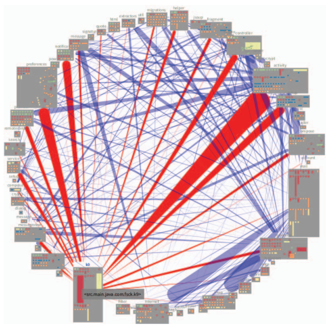


Fig. 4: Highlighting a package

that edges between units are scoped to the package, *i.e.,* only dependencies between units that belong to the same package are represented. Units not connected with other units within the same package are simply located as a grid and sorted by their role (`E`).

Dependencies between packages are deduced from the dependencies between units. Inter-package dependencies are represented using a *bimetric line*, in which the number of dependencies from the package `preferences` to `k9` is represented by the extremity size on the package `preferences` (`F`). Similarly, dependencies initiated in `k9` toward `preferences` are represented in the extremity size close to `k9` (`G`).

Such a bimetric line is adequate in presence of multiple birectional connections. Figure 3 clearly indicates that `preferences` heavily depends on `k9`, while `k9` depends little on `preferences`.

D. Interaction

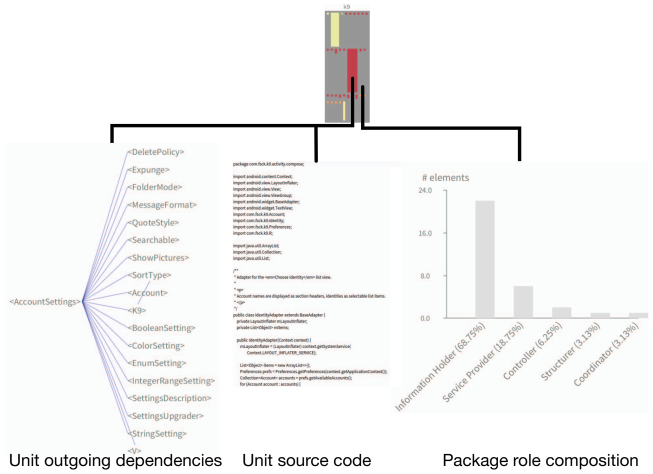
RoleViz offers a number interactions to ease the exploration of the software under analysis.

Mouse hovering. Hovering the mouse cursor above a package highlight in red dependencies between dependent and depending packages. Figure 4 illustrates the overall K-9 Mail application with `k9` highlighted. In addition, a popup appears to give the full package name of it. The figure shows that `k9` has little dependencies toward other packages however many are depending on `k9`.

When hovering the mouse cursor above a compilation unit, lines between the pointed unit toward all dependent other units appear (not shown in the figure). Lines are also colored according to the role of the dependent class.

Drill down. In a graphical environment, drill-down is an action to obtain detailed data about a particular visual element. Clicking on a package augments the main visualization with the *package role composition* histogram, indicating the proportion of different roles. In Figure 5, the histogram indicates that 68.75% of the compilation units contained in the `k9` package have the *Information Holder* role.

Clicking on a unit shows two views. *Unit outgoing dependencies* is a visualization that indicates the outgoing dependencies of the selected unit. *Unit source code* gives the source code, in which one can search using regular expressions. The view obtained when drilling down are displayed next to the main RoleViz visualization. For example, the source code may be shown all the time while using RoleViz.



```
package com.fsck.k9.activity.compose;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;
import com.fsck.k9.Account;
import com.fsck.k9.Identity;
import com.fsck.k9.Preferences;
import com.fsck.k9.R;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

/**
 * Adapter for the <em>Choose identity</em> list view.
 *
 * <p>
 * Account names are displayed as section headers, identities as selectable list items.
 * </p>
 */
public class IdentityAdapter extends BaseAdapter {
  private LayoutInflater mLayoutInflater;
  private List<Object> mItems;

  public IdentityAdapter(Context context) {
    mLayoutInflater = (LayoutInflater) context.getSystemService(
        Context.LAYOUT_INFLATER_SERVICE);

    List<Object> items = new ArrayList<>();
    Preferences prefs = Preferences.getPreferences(context.getApplicationContext());
    Collection<Account> accounts = prefs.getAvailableAccounts();
    for (Account account : accounts) {
```

Fig. 5: Drill down

Visualization Alteration. RoleViz offers five actions to alter the visualization. (i) First, packages and classes matching a provided a regular expression may be highlighted using a stark color. Such a feature is useful to highlight a particular cross cutting concern. The highlight remains until the user decides to explicitly remove it. (ii) Second, selected elements may be kept while all the others are removed. (iii) Third, selected elements may be removed. (iv) Fourth, the visualization can be reset to its original state, thus removing all the alterations. (v) Fifth, the visualization may be spawned into a new window, thus leading to a second instance of the visualization. This interaction allows for parallel unrelated system explorations. This can also be used in combination with the other alteration actions to produce a new visualization with smaller number of elements, e.g. the ones that match the search terms, thus allows users to focus on a specific parts of the system.

These interactions alter the visualization. As a consequence, they are likely to be triggered after a shallow exploration using mouse hovering and drill down.

IV. Research Questions

The research objective of our study is to determine whether RoleViz helps in enhancing the understandability of software architecture. We form two research questions to guide our study:

RQ1: *How does RoleViz compare to Softagram?* In particular, we compare the two visualisation tools in terms of:

- participant's perceived cognitive load,
- participant's perceived usability,
- participant's understanding of the software system regarding the tasks.

By "understanding", we refer to the participant's ability to: a) locate components/entities of the system relevant to the tasks, b) describe the responsibility of the located components/entities and relationship between them, and c) formulate a plan to solve the tasks.

RQ2: *What are the perceptions of the participants on the current features of RoleViz?*
Determining whether RoleViz meets the expectation of the participants is crucial to identify where exactly RoleViz falls short of feature. In addition, this research questions helps formulating the future direction of RoleViz.

V. User Study

To answer the research questions stated above, we designed and conducted a user study. The design of the user study involves the following five components.

A. Baseline

The performance of RoleViz has to be compared against a baseline visualization. Softagram, which is a commercial tool to visualize software system[2], was chosen to be the baseline tool for two main reasons.

Firstly, Softagram has been defined to address concrete problems of visualizing software architecture and it has been developed under a strong industrial influence. The visualization metaphor is UML-inspired: a software entity (*e.g.* a file, a package or a class) is represented as a node with attributes and links to other nodes. The associations between software entities are used to show various types of relationship (between the entities), such as inheritance, library usage, method-calls, etc. Figure 6 shows K-9 Mail with Softagram. At the center we see different packages, to which the red fading indicates a metric, number of lines of code in this example. Different layouts are accessible from the control panel located on the top of the window. On the right-hand side different properties to adjust the visualization are available.

Secondly, Softagram allows for a software exploration in an interactive fashion. In particular, users can drill down/up to navigate among levels of data ranging from the top package (up) to variables of a class (down). Mouse scroll can be used to zoom in/out at specific parts of the visualization canvas, thus allowing users to read details when the diagram is too large. Similarly to RoleViz, associations of an entity are highlighted when clicking on the entity. Moreover, Softagram also provides two search options which allow users to search globally in all entities of the studying system or locally within the entities showed on the main canvas.

Softagram can also be used to highlight architectural changes (such as new dependencies) introduced by the pull request author. Softagram does not offer source code view within the application but can direct users to the Github page of the source code file (via a web browser and the Internet).

B. Comprehension Tasks

We need to define two comprehension tasks. We started by defining a number of criteria (**C**) as the following:

C1: *Realistic.* The comprehension tasks should be derived from realistic software development or maintenance issues/tasks.

[2]https://softagram.com

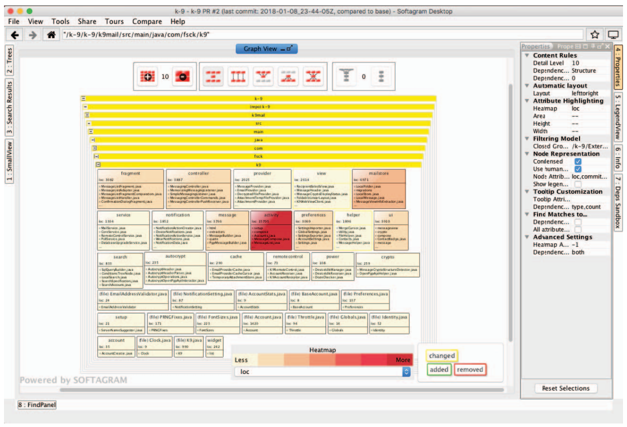

k-9 - k-9 PR #2 (last commit: 2018-01-08_23-44-05Z, compared to base) - Softagram Desktop
File View Tools Share Tours Compare Help
"/k-9/k-9/k9mail/src/main/java/com/fsck/k9"
Graph View
Heatmap
Less
More
loc
changed
added
removed
Powered by SOFTAGRAM
Reset Selections
8 : FindPanel

Fig. 6: Softagram main GUI - Structural View

C2: *Simple.* The tasks should be simple enough so that participants can complete them within the limited time of the study.

C3: *Independent.* The two tasks should not depend on each other and should not be semantically close. As we use a within-subject method, this criterion aims to mitigate the learning effects from solving one to another task.

C4: *Comparable.* The two tasks should be comparable in terms of complexity. With this criterion, we expect the differences between the tasks do not create any additional cognitive load or lead to any major changes in the performance of participants.

C5: *Verifiable.* We assess participant's understanding based on their solutions to the tasks. The assessment method should be built on top of a verified solution to the tasks. Therefore, it is important to find the tasks but also the solutions that are confirmed to solve the tasks.

Then, we looked into the issue tracking system of K-9 Mail to find realistic issues (**C1**). The issues were labeled by senior contributors of the project. We relied on these labels in order to filter relevant tasks for the study. In particular, we filtered those issues that were labeled as **good first issue** (for simplicity and comparability - **C2** & **C4**) and were solved/closed at the time of searching (so a working solution exists - **C5**). We found two issues namely "*Export/Import Settings*" (#2969) and "*Attachment Size Format*" (#3343) that satisfied the criteria.

Then, two comprehension tasks were built on the basis of the identified issues. Task *"Export/Import Settings"* (EXPORT/IMPORT) concerns with the problem that email settings (including preferences, contacts, etc.) do not display in the same order when being exported and imported from an old to a new Android device. Task *"Attachment Size Format"* (ATTACHMENT SIZE) aims at changing the size format of downloaded attachments from long number of bytes to a more human-readable form (e.g. in KB, MB, GB). The two tasks are independent and are not semantically close (**C3**).

It is noted that the main aim of the comprehension tasks is to locate and build up understanding around the part(s) of the software system that is(are) relevant to solving the given issues, not to implement or evaluate specific code changes. Details of the tasks can be found in the replication package of this study [14].

For each selected task, the following information was collected:

- The *description of the issue* is collected directly from the issue tracking system. The instructors do not modify or add any text to the description.
- The *discussion about the issue* includes messages regarding the issue and relevant/similar issues. Instructors of the user study focus on building up knowledge on the following two aspects when browsing the discussion: i) context/clarification of the issue; ii) solution(s) to resolve the task: This is the part of discussion where solutions are discussed.
- The *implementation of the solutions* is assessed against the code approved by the K-9 Mail community.

During the user study, participants are given the description of the issue only. The instructor of the user-study have access to all the information and used it to: i) build up understanding about the issue and the context where it arises. With this, the instructor is expected to be able to answer participant's questions regarding the task during the user study; ii) create a grading schema for assessing participant's understanding. We elaborate on the grading schema in Section VI-D.

C. Participants

The user study targets participants who have some kinds of experience with software development in Java programming language. The participants do not need to have prior understanding on the role-stereotypes or have any experience of using software comprehension tools.

We sent a call for voluntary participation to the user study via personal networks of the authors. In the call, the following information was clearly mentioned: i) a short description about the study; ii) requirements to the participants; and iii) expected time and duration of the study as well as expected amount of work from the participants. After two weeks, we received numerous responses and could finalize a list of 16 participants for the study. 2 weeks prior to the working session of the study (see Section V-E), an email with training materials (see Section V-D) and an URL to the online background form was sent to the participants. The instructor of the study also communicated with the participants in order to schedule time and location for the working session.

D. Training Period

The aim of the training period is to equip the participants with essential information to work effectively during the working session. By essential information, we refer the following pieces of knowledge: i) use of visualisation tools, i.e. functions and interaction mechanism of the visualisation tools; ii) role-stereotypes, i.e. what is the responsibility of each role.

We provided the participants with several training materials, including presentation slides about role-stereotypes and two self-designed tutorial videos on RoleViz [3] and Softagram [4]. These training materials were sent to the participants two weeks prior to the working session. The materials are included in the replication package of this study [14]. During this training period, the instructor was open to any questions regarding both of the tools and role-stereotypes.

E. Work Session

After the training period, we assume all participants have proper knowledge to start working on the comprehension tasks. The working sessions were designed to be 80 minutes long and were conducted on a desktop computer provided by the instructor in a scheduled time and room. All participants used the same screens and input devices. The activity (**A**) of a participant was structured as follow:

A1: *Introduction (5 mins)*: The instructor gave a brief introduction about the purpose and procedure of the session.

A2: *Warm-up (15 mins)*: During this time, the participant was allowed to actually use the visualisation tools. The main aim was for the participant to be more familiar with the control and interaction mechanism of the tools. The participant was also allowed to adjust the settings of the desktop computer and input/output devices (such as keyboard, mouse, screen) to fit his/her preferences.

A3: *Comprehension sessions (50 mins)*: Each participant performed two comprehension tasks (ATTACHMENT SIZE and EXPORT/IMPORT), each with help of a visualisation tool (RoleViz or Softagram). Each comprehension session was scheduled in 25 minutes with the following activities:

A3.1: Giving task description (3 min);

A3.2: Comprehending with a visualisation tool (15 mins);

A3.3: Answering post-task questionnaire (7 mins).

A4: *Post-study Questionnaire (10 mins)*: Participants were asked to answer open questions regarding their perceived benefit of using RoleViz and desired improvements of the tool.

VI. Data Collection & Analysis

We collected the following data (i) background information, (ii) NASA Task Load Index (TLX) Questionnaire, (iii) System Usability Scale (SUS) Questionnaire, (iv) Understanding Questionnaire, (v) Video Recording, (vi) Post-study Questionnaire. Next, we discuss how the data is collected and analyzed.

A. Background Questionnaire

Prior to the working session, participants were asked to fill in a background questionnaire. The questionnaire contains 10 questions regarding participant's experiences with Java programming language, Android and K-9 Mail system. If a participant answers that he knows/has experience with K-9 Mail, 2 extra questions are asked for clarification about this.

[3]https://youtu.be/HqCUAlai4qw
[4]https://youtu.be/YXizTrJ5j7I

B. TLX Questionnaire

Measurement. The NASA-TLX is a widely used technique for measuring subjective mental workload [15]. It relies on a multidimensional construct to derive an overall workload score based on six workload sources: mental demand, physical demand, temporal demand, performance, effort, and frustration level. There are two ways to compute the total workload score. One way, called *Weighted TLX*, involves a two-step process where participants first give rating for the six workload sources, then make a series of 15 pairwise comparisons between each pair of the sources as a basis for calculating weight of each source. The second way, called *Raw TLX*, is a light-weight approach in which the total mental workload score is simply calculated as the average of the 'raw ratings' of the six workload sources [16]. In this study, we chose to follow this light-weight approach to collect TLX data and calculate the total TLX score.

Data collection. After finishing a comprehension task (**A3.2**), participants were directly given a TLX rating sheet in paper form and a pen to mark on it. In total, each participant gave two rating sheets after the two comprehension sessions. We collected the sheets and transferred the result into a csv file for computational purpose. The instructor only gave explanation or clarification regarding the TLX scale based on NASA's TLX manual [17]. The instructor did not interfere or influence participant's ratings in any mean.

Data analysis. We compare the mean values of TLX scores between the two tasks in order to see the workload. Since our study is within-subject, we will use Wilcoxon signed-rank test to measure the differences.

C. SUS Questionnaire

Measurement. The System Usability Scale is an easy, standard way of evaluating the usability of a system [18]. It is a form containing ten statements, and users provide their feedback on a 5-point scale (1 is "strongly disagree" and 5 is "strongly agree"). It effectively differentiates between usable and unusable systems by giving a measure of the perceived usability of a system. It can be used on small sample sizes and be fairly confident of getting a good usability assessment [19].

Data collection. The ten SUS questions were integrated into the post-task questionnaire (**A3.3**). The participants were given the questionnaire after finishing with a comprehension task and the corresponding TLX ratings paper.

Data analysis. We follow the formula proposed by Brooke [18] to calculate the total SUS scores reported by the 16 participants. After that, we calculated the average of the usability values of all participants split by visualization tool to obtain the overall usability score of RoleViz and Softagram. We compare these values in order to examine the difference in usability of the two tools. In order to obtain a more detailed view of the difference (if any), we compare mean values of ratings to each of the 10 SUS questions between the two tools. We test the significance of the differences by using a Wilcoxon signed-rank test which is non-parametric and is often used in situations in which there are two sets of scores derived from same participants [20].

D. Understanding Questionnaire

Measurement. In order to measure participant's understanding of K-9 Mail system regarding to the tasks, firstly, we ask the participants to answers the following three questions (**Q**).

Q1. Can you name 5 elements (packages/classes/methods) that are the most relevant/important to the task?

Q2. What are the responsibilities of the elements chosen for the question above in performing the functionality related to the task?

Q3. Which changes of the elements chosen for question above are needed to complete the task? (Describe your plan/solution)

These three understanding questions aim to assess the three aspects of "understanding" (as defined in Section IV).

Next, we build and use a 11-point scale grading schema, *i.e.,* with the lowest score being 0 and the highest score being 10 points, to evaluate participant's answers. For each comprehension task, a grading schema is created by (same) one author of this paper based on the three sources of information regarding the task, including *description of the task*, *discussion about the task* and *approved implementation of solutions to the task* (as described in Section V-B). The grading schema consists of answers to the three understanding questions and criteria to judge the level of participant's understanding toward each questions. It is noted that different questions are given different maximum points based on our subjective judgment on their importance to forming participant's "understanding". In particular, answers to Q1, Q2 and Q3 could get maximum 5 points, 2 points and 3 points, respectively. More details about the grading schema can be found in the replication package of this paper [14].

Data collection. The 3 understanding questions are placed in the post-task questionnaire together with the 10 SUS questions (**A3.3**). During the comprehension time, participants were encouraged to take note about the relevant elements of the system to the tasks, thus they could quickly transfer their notes to the answer form. Their answers were then graded by two authors of this paper using the above-mentioned grading schema. In particular, the two graders graded all participants separately. After that, a meeting was held to discuss unclear cases, solve any disagreement, and give final scores. Total understanding score was calculated as a sum of the three component scores.

Data analysis. Similar to TLX and SUS score, we compute the total understanding scores of all participants and compare the mean values of understanding scores between the two visualization tools and the two tasks. In order to gain an insight about which aspect(s) of "understanding" contribute to the difference (if any), we also compare the scores between the understanding questions by visualization tools and tasks. We test the significance of the differences by using a Wilcoxon signed-rank method.

E. Post-study Questionnaire

Data collection. All participants were given a post-study questionnaire (**A4**) after they have finished with two comprehension sessions. The post-study questionnaire contains 7 open questions aiming at collecting participant's perceived benefit of using RoleViz in program comprehension and desired improvements of the tool.

Data analysis. We followed the Open Coding method [21] to analyze answers of the post-study questionnaire. We first identified concepts and key phrases are identified and moved into subcategories, and then grouped into categories.

VII. Result

In this section, we first present demographics of the participants of this study. Then, we explore the comparability of the two comprehension tasks used in the study. Lastly, we answer the two research questions of the study.

A. Demographics of Participants

In total, 16 people, with ages ranging from 23 to 36, participated in this study. These include 7 Master students, 4 Ph.D. candidates, 1 post-doc researcher and 4 software development engineers from 3 software companies. All of the participants have some experiences with the Java programming language, ranging from less than 1 year (2 participants) to more than 8 years (1 participant). The majority of the participants (10 out of 16) have 3-8 years experience with Java.

10 out of 16 participants reported to be familiar with the Android development framework. Among them, 5 participants have less than 1 year of experience, 2 participants have 1-2 years experience and 3 participants have 3-5 years of experience with the Android framework.

Only 5 out of 16 participants answered to know the K-9 Mail application and/or the K-9 Mail development project. 3 of them have been using the K-9 Mail application in a daily basis for managing emails on their Android devices. None of the participants reported to have comprehended the K-9 Mail system prior to the study.

14 participants watched the introduction videos of RoleViz and Softagram prior to the work session. For the two participants who did not watch the introduction videos, the instructor spent extra time (15 - 20 minutes) at the starting of the work session to guide them through important parts of the videos.

B. Are the Comprehension Tasks Comparable?

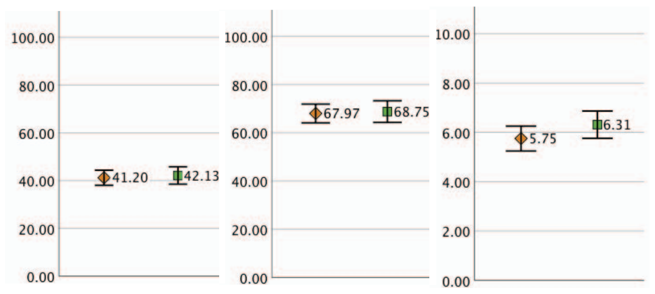
The two comprehension tasks used in our study were carefully selected (as described in Section V-B) with the expectation that the tasks are comparable. In this section, we examine this comparability in terms of participant's TLX, SUS and Understanding Questionnaire scores.

Figure 7 shows the mean values of participant's perceived TLX, SUS and Understanding scores for the two comprehension tasks. A Wilcoxon signed-rank test confirmed that the small differences are statistically insignificant, with *p-values* being *0.80*, *0.85* and *0.17* (all are well-above 0.05) for TLX, SUS and Understanding scores, respectively. We therefore conclude:

- The two comprehension tasks require similar cognitive load to solve (TLX score).

- Solving different tasks does not result in different perceived usability score (SUS score).
- Participants achieved comparable understanding scores after solving the two tasks (Understanding score).

The meaning of this result is two-fold. Firstly, it confirms that our task selection method is effective. Secondly, it suggests that we can ignore the factor of "task-difference" when analysing the difference between visualisation tools.



(a) TLX Score (b) SUS Score (c) Understanding Score

Fig. 7: Differences in mean values of (a) TLX, (b) SUS and (c) Understanding Scores (**+/-1 SD**) between two comprehension tasks: ATTACHMENT SIZE (◆) and EXPORT/IMPORT (■)

The two comprehension tasks are comparable in terms of complexity, required cognitive-load, usability score and understanding score. With this, we can eliminate the "task-difference" factor when analysing the difference between visualisation tools.

C. RQ1: Comparison between RoleViz and Softagram

1) TLX Task Load Score: Table I shows mean values of the overall- and component TLX scores across the two visualisation tools. The average task load index associated using RoleViz and Softagram in the comprehension tasks are 39.43±13.24 and 43.91±13.68, respectively. These scores indicate a low to moderate effort according [22].

Table I shows that the mean values of overall- and component task load associated with using RoleViz are always smaller, with the differences ranging from 0.63 to 12.19, compared to that of Softagram. The Wilcoxon signed rank test, however, shows that none of the differences are statistically significant (all p-values are above 0.05).

TABLE I: Comparison of average TLX scores by the two visualisation tools (N=16)

		RoleViz		**Softagram**		**Wilcoxon S.R.**	
		Mean (M1)	*SD*	*Mean (M2)*	*SD*	*M1-M2*	*p-value*
Overall TLX		39.43	13.24	43.91	13.68	-4.48	0.155
Task Load Sources	*Mental*	52.19	21.68	56.88	21.36	-4.69	0.347
	Physical	24.69	18.02	25.31	18.66	-0.63	0.857
	Temporal	50.63	28.63	54.69	26.86	-4.06	0.262
	Performance	33.75	15.33	36.25	24.87	-2.50	0.975
	Effort	42.19	17.89	54.38	20.65	-12.19	0.088
	Frustration	33.13	24.07	35.94	24.71	-2.81	0.371

The average task load associated with using RoleViz and Softagram for the comprehension tasks is comparable.

2) Usability Score: Table II shows the mean values of total SUS scores and component SUS scores associated with the two visualisation tools. RoleViz achieves an average of 72.43±14.93 in overall SUS score. This is significantly higher compared to that value of Softagram, which is 64.32±17.62 (p-value = 0.035). According [23], RoleViz is graded "C+" which indicates a *good* usability score, while Softagram is graded "C" which is considered as a *moderate* usability score.

In order to get an idea on which aspects of usability constitute the difference, we take a deeper look at the ten component SUS scores. We find that RoleViz tends to achieve higher (mean values of) rating to questions regarding the positive aspects of usability (i.e. Q1, Q3, Q5, Q7 and Q9). Meanwhile, Softagram seems to score "higher" for questions regarding the negative aspects of usability (i.e. Q2, Q4, Q6, Q8 and Q10). This plain comparison (of mean values) suggests that RoleViz achieved a "better" usability score for most of all component usability aspects (except for the required learning-effort where the mean values was equal).

A Wilcoxon signed rank test confirms that this difference is statistically significant. In the post-study questionnaire, one participant reported a comment that may explain this difference: "source code is not easily accessible using Softagram", whereas in RoleViz it is easy to navigate between design and source code perspectives.

TABLE II: Comparison of average SUS scores between RoleViz and Softagram (N=16)

		RoleViz		**Softagram**		**Wilcoxon S.R.**	
		Mean (M1)	*SD*	*Mean (M2)*	*SD*	*M1-M2*	*p-value*
Total SUS Score		72.34	14.93	64.38	17.62	7.97	0.035*
Usability Measurement	*Q1: Willing to use the system*	3.50	1.10	3.25	1.13	0.25	0.210
	Q2: Complexity of the system	2.00	1.03	2.56	0.63	-0.56	0.090
	Q3: Ease of use	3.75	0.93	3.38	1.15	0.38	0.110
	Q4: Need of support to use	1.88	0.81	2.25	1.13	-0.38	0.190
	Q5: Integrity of functions	3.69	0.87	3.13	0.96	0.56	0.020*
	Q6: Inconsistency	1.56	0.89	1.81	0.98	-0.25	0.250
	Q7: Intuitiveness	3.88	0.96	3.88	0.96	0.00	1.000
	Q8: Cumbersomeness to use	1.94	0.85	2.31	1.14	-0.38	0.080
	Q9: Feeling confident to use	3.44	0.81	3.00	1.21	0.44	0.080
	Q10: Required learning-effort	1.94	0.93	1.94	1.12	0.00	0.940

RoleViz is reported to have a significantly higher usability score compared to Softagram. Participants also valued the high level of integrity of available functions of RoleViz over Softagram.

3) Understanding Score: Table III shows the mean values of total Understanding scores and component Understanding scores associated with the two visualisation tools. Participants scored on average 6.56±1.82 points with RoleViz. This is significantly higher (by 10%) compared to an average of 5.50±2.28 points when using Softagram (p-value = 0.025).

To gain an insight into the participant's performance on different aspects of understanding, we calculate and analyse the component understanding scores. Table III shows that participants achieved higher scores for all three understanding questions. In particular, we observe a difference of 0.31, 0.25 and 0.50 points for the questions regarding *Identification* (of relevant components), *Responsibility* (between the identified components) and *Solution* formation, respectively.

The Wilcoxon signed rank test confirms that participants could indeed produce a better solution to the comprehension tasks with RoleViz compared to Softagram (p-value = 0.033).

TABLE III: Comparison of average Understanding scores between RoleViz and Softagram (N=16)

		RoleViz		**Softagram**		**Wilcoxon S.R.**	
		Mean (M1)	*SD*	*Mean (M2)*	*SD*	*M1-M2*	*p-value*
Understanding Score		6.56	1.82	5.50	2.28	1.06	0.025*
Comp.	*Identification*	2.69	1.20	2.38	1.26	0.31	0.353
	Responsibility	1.44	0.51	1.19	0.54	0.25	0.102
	Solution	2.44	0.63	1.94	1.00	0.50	0.033*

Participants achieved significantly higher understanding scores (by 10%) and produced better solutions when using RoleViz (for comprehension tasks) compared to using Softagram.

D. RQ2: Participant's perception on the features of RoleViz

The post-study questionnaire focused on collecting the perceptions and suggestions for improvement of RoleViz. The questions are open and need to be answered in plain English. We applied the Grounded Theory [21] to process the post-experiment feedback (Section VI-E)[5]. The analysis identified 10 general themes. Below, each general theme is annotated with the number of times it appears in the transcripts and the number of participants who explicitly expressed it. The general themes that we consider as positive are:

- *Usability and Efficiency* (50 occurrences / 14 participants): This theme covers the positive aspects of RoleViz regarding the efficiency (*e.g.,* accuracy of the provided information, helpful, searching, no need to read class names, support comprehension, identifying starting point) and usability (*e.g.,* clarity of the visualizations, narrowing down).
- *Role* (12 / 6): Overall, participants have positively perceived the way roles are presented by RoleViz (*e.g.,* "coordinator helpful to identify starting point", "used only the roles to complete the tasks").
- *Relevant view* (9 / 8): Participants have explicitly indicated that the main RoleViz visualization is helpful to complete the tasks. The possibility to have the source code always present is also reported as important.
- *Visual aspect* (5 / 5): Few participants have indicated some positive aspects of the visual cue. In particular, it was reported that the circle layout gives a good overview of the system. The highlighting and coloring are perceived as useful.

The general themes that we consider as negative toward RoleViz are:

- *Possibility for improvement* (34 / 15): We asked the participants to answer the question "Do you have any suggestions for improvements in RoleViz?". All participants but one made suggestions about various aspects of RoleViz. In particular, being able to navigate within the source code by only scrolling and textual searching is a limitation. One participant reported that the use of color is problematic (red usually refers to a problem and we use it to represent the information holder role, cf Figure 2).
- *Missing information* (7 / 7): Participants criticized missing information about methods and variable accesses. Currently, methods are listed within the source code, obtained by clicking on a compilation unit. Participants found this not convenient.
- *Issue when showing source code* (7 / 6): Source code is poorly supported by RoleViz.
- *Bugs* (5 / 5): A few bugs were reported, in particular that the legend is not always visible.
- *Issue with the experiment* (1 / 1): One participant found that not all the information provided by the visualization are necessary to solve the tasks.
- *Visual element not useful* (3 / 2): Two participants reported that the information about roles and dependencies between elements are not useful.

Overall, participants appreciate the usability and efficiency of RoleViz to complete the tasks. In total, 6 participants reported the importance of annotating the software architecture with roles information. Participants missed information about methods, in particular the need to inspect a method call graph was reported by 7 participants. Also, source code should be better supported with syntax highlighting and searching.

VIII. Threats to Validity

We identified a number of threats to our researchs validity and categorized them by using the validity terminology introduced by Wohlin et.al [24]. In this section, we discuss three types of threats to validity, i.e. threats to construction validity, internal validity and conclusion validity.

Threats to Construct Validity. Participants, who are in the network of the authors of this paper, might be biased towards the visualisation that the authors created. We mitigate this issue by not revealing the authorization of the two visualisation tools until the end of the study.

Threats to Internal Validity. All of the participants are not familiar with both Softagram and RoleViz prior to the study. The unfamiliarity might hinder participant's effective use of the tools for comprehension tasks, thus results in a low SUS/Understanding score and a high task load index. The training period and the warm-up sessions are involved as part of the study to mitigate this threat. In fact, participant's answers to the question 4 of the SUS form (Table I) indicates a small need of assistance when using the two tools.

Threats to Conclusion Validity. Our answers to RQ1 base mostly on statistical tests on a small sample size. Therefore, there is a threat that the conclusion might not be representative of our analysis. A mitigation strategy could be to involve more people to the next round of tool evaluation.

[5]The analysis can be found in the replication package of this paper

IX. Related Work

Several studies investigated the effect of using stereotypes on software comprehension tasks. Staron et al. [9] conducted a set of controlled experiments both in academia and in the industry to evaluate the effect of role-stereotypes on UML models comprehension. They found that stereotypes play a significant role in the comprehension of models. In particular, the participants who used stereotyped models scored more correct answers in tests checking the level of understanding. Moreover, these participants required less time to answer comprehension questions and identify the correct answers.

Genero et al. [3] conducted a controlled experiment to investigate the impact of using stereotypes on UML sequence diagrams comprehension. They analyzed the use of sequence diagrams with and without stereotypes. They found that there is a slight tendency in favor of the use of stereotypes in facilitating the comprehension of UML sequence diagrams.

Ricca et al. [4] run a series of experiments to test whether the use of the stereotyped UML diagrams supports the comprehension and maintenance activities of web applications with significant benefits. They compared the performances of subjects in comprehension tasks where they have the source code complemented either by standard UML diagrams or by stereotyped diagrams. They suggested that organizations can achieve a significant performance improvement by letting their less experienced developers (i.e., juniors) adopt stereotyped UML diagrams for comprehension tasks.

Sharif and Maletic [8] studied the effect of two different stereotyped layouts on the comprehension of UML class diagrams: *orthogonal* and *clustered*. The orthogonal layout minimizes edge crossing and bends and does not use information about the class stereotype in layout positioning. The clustered layout uses information about the class stereotype to position classes into multiple clusters in the diagram. They found that the use of stereotyped- clustered layouts demonstrates a significant improvement in subject accuracy and efficiency in solving problems in comprehension tasks.

In the same direction, Andriyevska et al. [6] designed and conducted a user study to evaluate the effect of using a stereotyped UML class diagram layout on diagram comprehension. Andriyevska and her colleagues suggested that stereotyped UML class diagrams support software comprehension tasks by letting developers build better mental models, hence gain more information about the considered software system.

Yusuf et al. [7] conducted a study to assess the effect of using layout, color, and stereotypes on UML class diagram comprehension. As a mean to achieve their goal, the authors used eye-tracking equipment to collect data on subjects' eye gaze which are then used to analyze the cognitive process involved in the visual data processing. They suggested that the use of class stereotypes plays a substantial role in the comprehension of UML class diagrams. Moreover, they suggested that the use of layouts with additional semantic information about the design is the most effective for diagrams comprehension.

Blouin et al. [2] proposed an interactive visualisation tool for comprehending large meta-models. Their tool, *Explen*, used a model slicing technique to allow users focus on subset of model elements of interest. A comparative evaluation of *Explen* with *EcoreTools*[6] showed that *Explen* outperforms in improving large meta-models understanding.

Unlike the previous work which focused on adding role as an extension of static UML models, we built an interactive polymetric view visualisation without referring to UML models. We also utilise the 6 role-stereotypes identified by Wirfs-Brock [10], which are at a different level of abstraction compared to the class-stereotypes used by other previous work [6][7][8][9]. Our visualisation tool also proven to be scalable, i.e. can visualise 700+ classes, compared to the previous work which focuses on assisting the comprehension task of a subset of the model of interest.

X. Conclusion and Future Work

In this paper we studied the visualisation of large software systems in order to aid comprehension of the design of the system. In particular, we contribute the use of role-stereotypes and a visualisation-tool called RoleViz. We compare our tool to an industrial tool Softagram via a user study with 16 people.

Results from the study indicate that RoleViz achieves a higher score on usability than Softagram. In particular, users like the integration features that enable exploring a software system at both the design and the source code levels of abstraction. According to Sauro's benchmark data, Roleviz achieves 'good' usability, and Softagram achieves 'ok' usability.

Performing comprehension tasks using RoleViz or Softagram is experienced as comparable with respect to the required cognitive effort. According to the benchmark data, the cognitive load of both tool is rated as 'low to moderate' effort.

Participants achieved about 10% better score on comprehension tasks when using RoleViz compared to using Softagram. Specifically, when using RoleViz, participants perform better in the ability to propose a solution to bug-fixing. The factor that may explain this is: RoleViz has as scoping mechanism that allows users to effectively focus on certain parts of the system that are relevant for the task at hand.

We found that some participants use classes with particular stereo-types as starting-points for particular understanding tasks: for example, for tasks that deal with 'user interface' issues, participants start their exploration of the system by looking at classes labelled as 'interface'-type.

For the future directions of the research, the participants in our study prominently pointed out at the need for also visualizing behavioral information (*e.g.,* call-graphs) at a level between source code and architecture level of abstraction.

Acknowledgments

Bergel is deeply grateful to Lam Research and FONDECYT Regular project 1200067 for partially sponsoring the work presented in this article.

[6]https://www.eclipse.org/ecoretools

References

[1] M. Shahin, P. Liang, M. A. Babar, A systematic review of software architecture visualization techniques, Journal of Systems and Software 94 (2014) 161 – 185. doi:10.1016/j.jss.2014.03.071.

[2] A. Blouin, N. Moha, B. Baudry, H. Sahraoui, J.-M. Jézéquel, Assessing the use of slicing-based visualizing techniques on the understanding of large metamodels, Inf. Softw. Technol. 62 (C) (2015) 124–142.

[3] M. Genero, J. A. Cruz-Lemus, D. Caivano, S. Abrahão, E. Insfran, J. A. Carsí, Does the use of stereotypes improve the comprehension of UML sequence diagrams?, in: 2nd Symposium on Empirical Software Engineering and Measurement, ESEM '08, ACM, USA, 2008, pp. 300–302.

[4] F. Ricca, M. Di Penta, M. Torchiano, P. Tonella, M. Ceccato, How developers' experience and ability influence web application comprehension tasks supported by UML stereotypes: A series of four experiments, IEEE Trans. Softw. Eng. 36 (1) (2010) 96–118. doi:10.1109/TSE.2009.69.

[5] I. Jacobson, G. Booch, J. Rumbaugh, The Unified Software Development Process, Addison-Wesley, Boston, MA, USA, 1999.

[6] O. Andriyevska, N. Dragan, B. Simoes, J. I. Maletic, Evaluating UML class diagram layout based on architectural importance, in: 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2005, pp. 1–6. doi:10.1109/VISSOF.2005.1684296.

[7] S. Yusuf, H. Kagdi, J. I. Maletic, Assessing the comprehension of UML class diagrams via eye tracking, in: 15th Int. Conf.on Program Comprehension. ICPC'07., IEEE, 2007, pp. 113–122.

[8] B. Sharif, J. I. Maletic, The effect of layout on the comprehension of UML class diagrams: A controlled experiment, in: 5th IEEE Int. WS. on Visualizing Software for Understanding and Analysis (VISSOFT 2009), IEEE, 2009, pp. 11–18.

[9] M. Staron, L. Kuzniarz, C. Wohlin, Empirical assessment of using stereotypes to improve comprehension of uml models: A set of experiments, Journal of Systems and Software 79 (5) (2006) 727–742.

[10] R. Wirfs-Brock, Characterizing classes, IEEE Software 23 (2) (2006) 9–11.

[11] A. Nurwidyantoro, T. Ho-Quang, M. R. V. Chaudron, Automated classification of class role-stereotypes via machine learning, in: Proceedings of the Evaluation and Assessment on Software Engineering, EASE '19, ACM, New York, NY, USA, 2019, pp. 79–88.

[12] M. Lanza, S. Ducasse, Polymetric views—a lightweight visual approach to reverse engineering, Transactions on Software Engineering (TSE) 29 (9) (2003) 782–795. doi:10.1109/TSE.2003.1232284.

[13] S. Demeyer, S. Ducasse, O. Nierstrasz, Object-Oriented Reengineering Patterns, Morgan Kaufmann, 2002.

[14] Replication package.
URL https://bit.ly/380PQb7

[15] S. G. Hart, L. E. Staveland, Development of nasa-tlx (task load index): Results of empirical and theoretical research, in: Advances in psychology, Vol. 52, Elsevier, 1988, pp. 139–183.

[16] E. A. Bustamante, R. D. Spain, Measurement invariance of the nasa tlx, in: Proceedings of the Human Factors and Ergonomics Society Annual Meeting, Vol. 52, SAGE Publications Sage CA: Los Angeles, CA, 2008, pp. 1522–1526.

[17] NASA, Nasa task load index (tlx) v.1.0 manual.
URL https://humansystems.arc.nasa.gov/groups/TLX/downloads/TLX.pdf

[18] J. Brooke, et al., Sus-a quick and dirty usability scale, Usability evaluation in industry 189 (194) (1996) 4–7.

[19] T. S. Tullis, J. N. Stetson, A comparison of questionnaires for assessing website usability, in: Usability professional association conference, Vol. 1, Minneapolis, USA, 2004.

[20] F. Wilcoxon, Individual comparisons by ranking methods, Biometrics bulletin 1 (6) (1945) 80–83.

[21] S. Anselm, J. Corbin, Basics of qualitative research: techniques and procedures for developing grounded theory, SAGE Publications, Thousand Oaks, USA, 1998.

[22] R. A. Grier, How high is high? a meta-analysis of nasa-tlx global workload scores, in: Proceedings of the Human Factors and Ergonomics Society Annual Meeting, Vol. 59, SAGE Publications Sage CA: Los Angeles, CA, 2015, pp. 1727–1731.

[23] J. Sauro, A practical guide to the system usability scale: Background, benchmarks & best practices, Measuring Usability LLC Denver, CO, 2011.

[24] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer Science & Business Media, 2012.