**Evaluations**

**Introduction**

According to Santos[1], visualization is a “human-in-the loop problem” which are complex by nature and those who want to evaluate visualizing tools still struggle with a lack of specific technique and methodologies to conduct the evaluation. In addition, as North states, visualization should not only provide insight to their users, but also do it in reasonable time with reasonable satisfaction. So we designed our own experiments to evaluate our tool by checking whether it generates insights and how well it is done qualitatively and quantitatively.

**Part 1: Empirical study**

We conducted empirical case study to evaluate our projects. In this stage, we designed questionnaires to explore what features are most desired by users, what’s the largest gap between services current visualization tool have provided and users’ needs, whether the features are useful for users, whether the interface is user-friendly for our own light-weighted visualization tool. Since users with various background may have quite different expectations for visualization tools and have diverse evaluation criterion, 2 different questionnaires are designed to collect maximal information from users.

Procedure of the survey:

We only select the users who use Java as their preferred language. We firstly explained the functions of our tool and then ask them to use it to visualize as unfamiliar project(…). In the survey, we first ask question 1, then based on the answer for question 1, we category the users into 2 types: infrequent users, and professional users, and send out questionnaire 1, questionnaire 2 respectively.

Question 1: Have you ever used any visualization tools?

Questionnaire 1:

1. What you do usually use it for?
2. What are the common methods or tools for visualizing, as far as you know?
3. How large are your projects when using the visualization tool?
4. What programming language do you prefer?
5. Which tool is your favorite one and which feature of it do you like most?
6. What is the feature you used mostly frequently?
7. Why do you prefer it over the others?
8. Is there any information you hope to get from this tool but it doesn’t provide?
9. Are you satisfied with the functions this tool has provided?
10. What do you like most about our tool?
11. What changes would most improve our tool?
12. Will you use our tool in the future? For what purpose?

Questionnaire 2:

1. What programming language do you prefer?
2. Will you use pen and pencil drawing UML when understanding unfamiliar codes?
3. What are the challenges you come across when reading unfamiliar codes?
4. Are you satisfied with the functions this tool has provided?
5. Do you think our tool has helped you understanding unfamiliar codes faster and better?
6. What do you like most about our tool?
7. What changes would most improve our tool?
8. Will you use our tool in the future? For what purpose?

**Part 2: Comparison with other tools**

According to Stephan Diehl[2], software visualization can be divided into three parts: Structure, behavior, and evolution. Unlike Code\_swarm which focuses on the evolution process, our software is designed to assist programmer in understanding code’s structure from given codes. To be more specific, it offers relationship (dependency and call) of a class and its methods with other classes visually. and shows the analysis of software systems and their anomalies by attaching additional attributes for nodes in the graph. It is designed for students who are newcomers for java and programming, who are in great need of assistance in understanding unfamiliar codes and setting up good habits in writing own projects, such as suitable percentage of comments among codes, readable size for methods.

Features of our tool:

1. Relationship between method and classes in the file is represented as edges between nodes
   1. Dependency : 0 or 1
   2. Coupling information: Afferent Coupling(Ca) and Efferent Coupling(Ce)
   3. TypeRank and MethodRank
   4. depth of inheritance Tree(DIT)
2. Attributes for each node:
   1. LOC (file, class, method)
   2. number of class, method, attributes
   3. Percentage of comments
3. interactive interface
   1. easy to query

Definition of attributes:

1. Here we use logical LOC which is the logical number of lines in the code computed from PDB’s sequence points instead of the physical LOC which directly computed from source code. In this way, coding style won’t make a difference on the output among same implementations of functions.
2. Percentage of Comment: It is the percentage of total number of lines for comments among total logical LOC. There should be a suitable range of percentage of comments, we assume it’s around 20%. Codes with percentage less than 20% should be more commented.
3. Afferent Coupling(Ca) and Efferent Coupling(Ce): Ca is the number of code elements that uses it and the Ce is the number of code elements that it uses. Ca equals 0 usually indicates a potential dead code except those will be called at runtime. Ce and ca can be referenced for evaluating the codes and further refactoring, design.
4. TypeRank and MethodRank: it implements PageRank algorithm to assign importance for each element (type or method) in the whole graph based on the connections among elements. It is especially helpful for new users by directing them to the most important or influential parts in the whole project.
5. The Depth of Inheritance Tree: the number of base classes for a class or structure. Types where DepthOfInheritance is higher or equal than 6 might be hard to maintain.
6. NbNamespaces. NbTypes.?????

Comparison:

Since each tool has its focus and visualizes different features, we compare our tool with those which contain same features but with different way of representation. Two main features we want to compare are inheritance and LOC. For inheritance, we compare with tool implementing Treemap representation. For LOC, we compare with ploymetric view?? or Seesoft?

In the experiment, participants are asked to use these two tools at the same time, which visualizing the same codes. However, only one picture can be shown at one time. Then they are asked to answer several questions about inheritance from these two pictures generated from two tools respectively. We video recording the whole process, then we calculate the usage of two pictures when asking questions. The one with the most frequent usage will be concluded as the most user-friendly one.

Questions for inheritance:

1. What are the classes class XXX depend on? (relative large one)
2. What are the classes class XXX depend on? (relative small one)
3. What is the largest class in the system?
4. What are the most important class inferred from the picture?

Questions for file attributes:

1. Which file is the largest in terms of number of lines?
2. Which file do you think contains fewer comments among all?

Add more?

**Part 3. Quantitative analysis with matrix**

We hope to evaluate our tool mainly from perspective of speed and accuracy.

Metric of run time behavior: Worst-case execution times, location of bottlenecks, processor utilization.

**Conclusion:**

Our tool is suitable for the following scenario compared to other visualization tools: 1. understand unfamiliar codes 2. reflect on own projects. Our unique contribution is that we select most important features which really matter to new users and combine those multiple perspectives within a single tool, conveying information they really need and as much as possible.

Additional features:

1. recommendation

Reference:

1. Santos, Beatriz Sousa. "Evaluating Visualization Techniques and Tools: What Are the Main Issues." *the 2008 AVI Workshop on Beyond Time and Errors: Novel Evaluation Methods For information Visualization (BELIV’08)*. 2008.
2. Diehl, Stephan. *Software visualization: visualizing the structure, behaviour, and evolution of software*. Springer Science & Business Media, 2007.
3. North, Chris. "Toward measuring visualization insight." *IEEE computer graphics and applications* 26.3 (2006): 6-9.