Canadian Journal of Pure and Applied Sciences
Vol. 9, No. 2, pp. 3431-3439, June 2015
Online ISSN: 1920-3853; Print ISSN: 1715-9997
Available online at www.cjpas.net

VISUALIZATION APPROACH FOR SOFTWARE PROJECTS

Mustafa Hammad
Department of Information Technology, Mutah University
Al-Karak, Mutah 61710, Jordan

ABSTRACT

Software visualization helps developers to understand, compare and browse large scale information. Software projects consist of a large number of packages, classes and methods. It is not an easy task for developers to browse source code and get information about the project's contents. This paper presents an approach to visualize the contents of a software project in summarized views named BookViews. Each project is modeled as a book in which its chapters represent packages, its sections represent classes and its pages represent methods. The visualization helps developers to understand the internal structure of java projects, as well as, compare between projects in simple and meaningful views. A pilot experimental study on an open source project has been conducted to evaluate BookViews. Results showed that BookViews supports program comprehension and helps developers in understanding the contents of software projects.

Keywords: Software visualization, program comprehension, software projects.

INTRODUCTION

Software projects compose of a large number of different artifacts. The main artifact of a software project is the source code. In object oriented programming, the source code consists of different components, such as packages, classes, and methods. A software project is implemented by many lines of code, and decomposed into a set of packages. Each package composed of a set of classes, and each class is responsible for specific and related activities that are implemented as methods.

Understanding and monitoring the current status of a project's components is very important for software developers and designers. In a software project, any code changing activity requires deep understanding of the project's internal elements. Therefore, software developers always need to browse the project's contents before modifying any part of the project. Moreover, to analyze a software project, developers need to explore huge information about code elements and components structure.

For large scale software projects, with large number of classes, browsing requires considerable amount of time and effort. It is not an easy task to explore hundreds or thousands lines of code to discover how these lines are divided into packages, classes, and methods. Furthermore, estimating the size of a software project is a time consuming task because the developer need to count and extract the internal contents of the project. In addition, to know the size of a project's packages or classes, developers need to investigate and count the internal components of each of them.

Another issue is in comparison between software projects. Developers may need to compare different project sizes. Comparison requires browsing all projects under consideration to count their structural contents. For instance, a developer may need to know which project has the largest number of methods, classes, or packages. There is a lack of tools that automatically extract all internal contents information for a project or a set of projects.

This paper proposes a methodology that utilizes the advantages of software visualization to help developers in program comprehension activities. The proposed visualization is set to apply for software projects written in Java programming language. To ease the process of browsing a project and comparing different projects, this paper proposes a visualization methodology named BookViews. In the proposed visualization, a software project is modeled and visualized as a book. The generated book view is considered as a summary of the target project. As a result, a set of different projects are visualized as a set of books. Each book is composed of a set of chapters and sections. The book's chapters and sections represent the project's packages and classes respectively. In addition, the visualized book's pages

*Corresponding author e-mail: hammad@mutah.edu.jo represent project's methods.

The proposed visualization has two different views for developers. One view is called close-book view and the second one is called open-book view. In the close-book view, the target project is visualized as a closed book titled with the project name. The open-book view visualizes a summary of the project's contents as a table of contents.

The BookViews visualization solves the problem of automatic generation of visual summaries for software projects. It saves time and effort for developers who want to browse the contents of their projects. It also provides summarized information about the contents of software projects in one view. Moreover, developers can visually compare different project sizes.

The remainder of the paper is organized as follows; the following subsections discuss the important of software visualization and related work. The next section presents the proposed visualization with an example. After that, the evaluation of the proposed visualizations is discussed, followed by conclusion and future work.

Why software visualization?

In large scale software projects, there is a huge information that are related to system development and evolution, which is hard to collect, measure, and analyze (Sommerville *et al.*, 2012). Moreover, extracting useful information from a set of software data is a challenging task and it is inherently costly. For example, to extract relationships among a project's components or artifacts, developers need to explore and measure each component in the target project to understand which component is communicating with which. Furthermore, to compare two sets of information, there is a need to develop mathematical representations. However, these mathematical representation need to be analyzed and proved carefully. Software visualization is one way to overcome these issues.

Visualizing any information related to software projects is becoming more popular way to understand large scale software systems (Maletic *et al.*, 2002). Software visualization is used to provide a visual representation of software information. By providing visual objects, software projects' designers or managers can easily get too much detailed information. In addition, software developers can discover related information and clusters without any formal equations or statistical data. This is because the generated visual objects are easy to understand, follow, and compare.

Software visualization is used to provide a visual representation for different aspects of software projects evolution (Novais *et al.*, 2013). For instance, visualization techniques can be used to visualize source code, documentations, software archives, code changes, project's developers, and bugs. The visualization process is a methodology to visualize a software project's aspects in interactive or animated visual representations. The methodology is set to find relationships among different information of the development process. The visualized information can be static, such as project's structure and size, or dynamic, such as execution behaviour and evolution. This paper focuses on visualizing static information of software projects.

The goal of software visualization is to support program comprehension. For instance, understanding software projects is easier when visualizing its structure and internal parts in comprehensive and representative one, two, or three dimensional views. Usually, software visualization is used as a tool or technique to explore and analyze software system information. This paper proposes a visualization methodology, called BookViews that visualize the contents of software projects. The generated views of the proposed visualization are easy to understand and provide rich, useful, and detailed information in a graphical way.

Related work

Software visualization can be applied in different methodologies. For example, Lanza and Ducasse (2001) presented a novel categorization of classes and a visualization of the classes, which called the class blueprint. The class blueprint visualizes the internal structure of classes. The focus of the visualization is on the static structure of classes and the way they make use of inheritance and leave out other collaboration aspects that is addressed by BookViews.

Maletic *et al.* (2001) described a system for visualizing object-oriented software in a Virtual Reality Environment. They defined a visualization language (COOL) to map C++ source code to a visual representation. Wettel and Lanza (2008) presented a 3D visualization approach that depicts object oriented software systems as cities that can be explored. The buildings in the city, which represent software artifacts, are located according to a set of predefined rules, and thus facilitate the establishing of visual orientation to gain familiarity with the system.

Another visualization approach is presented by Kobayashi *et al.* (2013). They developed SArF Map technique that visualizes software architecture based on its feature. In this visualization, each feature is visualized as a block in a city. All classes that implement a feature are laid out as buildings. Moreover, in a recent study, Hammad and Rawashdeh (2014) proposed a framework to visualize class coupling as bar charts. A single class is modeled as a block section in the visualized bar chart. The height of the bar represents the coupling degree, and the width of the bar represents the size of the class. BookViews differ from these tools by providing the ability to compare the internal contents not the logical structure of software projects.

In 2009, Ogawa and Ma presented a visualization technique using a method called code_swarm. The visualization shows the efforts of developers on software projects. Hammad *et al.* (2014) presented a visualization approach for bug reports in software repositories. The visualization goal is to display the different status of a bug report, as well as, the bug reports/developers relationships by using combinations of colored shapes and arrows.

Ma (2008) developed a visualization system named StarGate. The system visualizes the code repository and social network of developers associated with a software project in one integrated representation. Developers are grouped visually into clusters corresponding to the areas of the repository they work on the most. Links are generated between people who communicate via email.

Alam and Dugerdil (2007) developed a visualization tool named EvoSpaces. It represents the architecture and metrics of complex software systems as 3D software cities. Files and classes are represented as buildings of a 3D virtual city arranged in districts. The user can travel through the city by flying over the buildings or walking among them like in some video games. In 2011, Caserta and Zendra surveyed many other 3D and 2D visualization techniques and tools.

Some visualization techniques are used to help analyzing software development process. For instance, DEVis tool (Zhi and Ruhe, 2013) is an interactive tool to generate software documentations that are related to project evolution. Moreover, SourceVis (Anslow *et al.*, 2013) is a collaborative tool to visualize software artifact in multi-touch tables and user interfaces.

In 2010 Beck and Diehl proposed a visualization technique to represent software architecture from descriptions that are generated from reverse engineering. The visualization is based on a scalable adjacency matrix representation. CoCA-Ex (Holy *et al.*, 2013) is another example tool that is used during reverse engineering to visualize components with large inner connections in a separate level by using clustering.

MATERIALS AND METHODS

Overview of the Proposed Visualization

BookViews models a software project and visualize it as a book. The generated book graph provides a general summary of the project's contents. By visualizing many projects, the final views act as a library with different books. By using this library view, software developers can easily compare project sizes and contents.

Usually, a book consists of a set of chapters, and each chapter consists of a set of sections. A specific number of pages are considered as a section length. BookViews visualization uses this known structure of books to model software projects. The proposed visualization models the project's contents and size for the developers. Structural contents of a software project that are considered in BookViews visualization are:

- Project title
- All package names in the target project
- All class names in each package
- Number of methods in each class

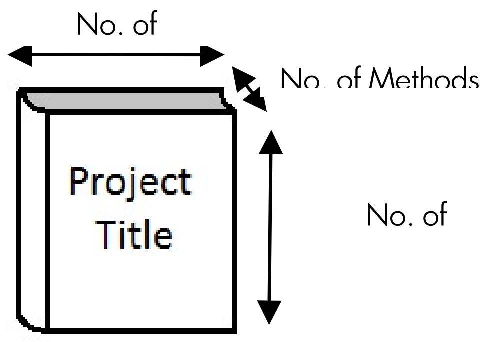


Fig. 1. Close-book View of the Proposed Visualization.

In the proposed visualization model, the contents of a project are modeled as follows:

- The name of the target project is the title of the generated book.
- Each package in the project is represented as a chapter in the book.
- Each class is represented as a section in a chapter.
- Each method is represented as a single page.

The proposed visualization provides two different views; close-book and open-book views. In close-book view, the target project is visualized as a closed book titled with the project name. The size of the book depends on the total number of methods defined in the project. In open-book view, a table of contents is generated to summarize the contents of the project. These views are illustrated in the following subsections.

The Close-book View

The close-book view visualizes a software project as a closed book. This view helps to identify the size of the target project. Figure 1 shows the proposed close-book view structure. The size of the visualized book depends on the number of pages, which represents the total number of methods in the project. Usually, large projects have large number of functionalities implemented by many methods, while small projects have limited number of methods. Moreover, the height and the width of the visualized book represent the number of packages in the project. Generally, developers divide a project into multiple packages in order to make it easy to understand and maintain. A height and width unite in the rendered picture of a book represents a single package in the target project. This is useful to let the developers know how wider their projects are.

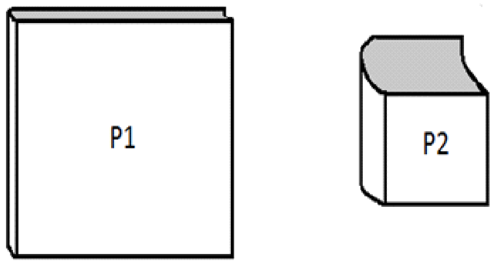


Fig. 2. Two close-book View Examples for Two Sample Projects Named P1 and P2.

Close-book view can be used to compare between different projects in terms of main structure and total number of functionalities provided in a project. The user of the BookViews tool can visualize a list of books together to generate a library of many books view. In this library, each book corresponds to a project with its own size. In one view, the user can quickly compare between projects to answer designing questions, such as:

I. What is the biggest project in term of number of methods?
II. Which of them has the smallest number of packages?
III. How many projects have equal number of packages?

Another benefit for the close-book view is that it can give the developer an overview about the relationship between number of methods and number of packages in the project. For example, Figure 2 shows two different examples of close-book view for two sample projects named P1 and P2. We can understand from Figure 2 that first project, P1, has large number of packages and relatively small number of methods. Therefore, developers may be encouraged to merge some packages together in P1 project. On the other hand, P2 project has many methods with few packages. For the second project, designers may start thinking of splitting some project's packages after seeing this close-book view generated from BookViews tool.

The Open-book View

The second view in the proposed visualization is the open-book view. This view visualizes the contents of a single project. The contents of the target project are represented as a table of contents for the visualized book. A table of contents for a visualized book is generated by the BookViews tool to summarize and organize the internal structure of the target project. Usually, a table of contents in a book consists of a set of chapters, and each chapter contains multiple sections. In the proposed visualization, each chapter represents a package. The title of the chapter is the name of the package. Chapters are ordered and numbered alphabetically. For example, a book with 10 chapters means a project with 10 packages.

Figure 3 shows the proposed table of contents for a visualized book. Each chapter is partitioned into a set of sections. Sections represent classes that are defined in that package. Sections are ordered alphabetically and given a sequence number based on the chapter's number. For example, Section 3.1 means the first class in the third package. Therefore, the total number of sections reflects the size of a package. Project's methods are represented as the book's pages. For instance, a class with four methods is considered as a section that has four pages. Each section (class) has its own pages (methods), which are listed after pp. symbol next to the section name. In Figure 3, α_{ij} is the cumulative summation of total number of pages (methods) from Section1.1 to Section i.j, where *i* is the current package/chapter, and *j* is the current class/section. Formally, α_{ij} and can be described as:

$$\alpha_{ij} = \sum_{k=1}^{i} \sum_{j=1}^{S(k)} \textbf{no. of pages in } \boldsymbol{Section}_{kj} \qquad (1)$$

Where S(k) is number of sections in Chapter k. On other words, α_{ij} also can be described as:

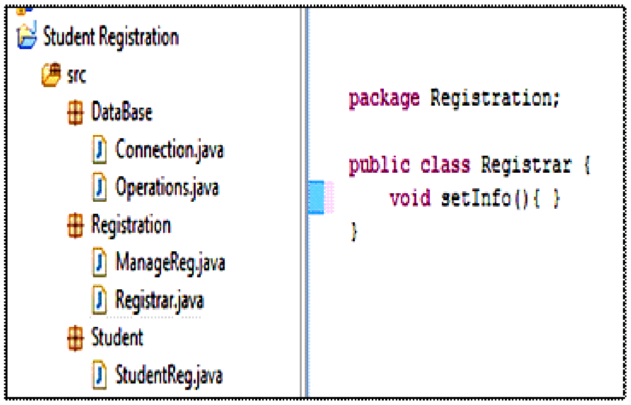
$$\alpha_{ij} = \sum_{k=1}^{i} \sum_{j=1}^{C(k)} \textbf{no. of methods in } \boldsymbol{Class}_{kj} \qquad (2)$$

Where C(k) is number of classes in Package k.

Table of Contents

- …
- Chapter i: Package$_i$ name
  - …
  - Section i.j: Class$_{ij}$ name (pp. (a_{ij} +1) - (a_{ij} + No. of methods in Class$_{ij}$)
  - Section i.j+1: Class$_{ij+1}$ name (pp. (a_{ij+1} +1) - (a_{ij+1} + No. of methods in Class$_{ij+1}$)
  - …

Fig. 3. The Proposed Structure of the Table of Contents in the Open-book View.



```
package Registration;

public class Registrar {
    void setInfo(){ }
}
```

Fig. 4. Snapshot from Eclipse for Student Registration project.

Table 1. Statistics for the sample project.

Class No.	Class Name	No. of Methods
(1)	Connection.java	3
(2)	Operations.java	4
(3)	ManageReg.java	3
(4)	ManageReg.java	3
(5)	Registrar.java	1
(6)	StudentReg.java	5

The visualized book's pages are numbered sequentially starting from the first method in the first class that is defined in the first package. Therefore, the first page is in Section 1.1. A method number depends on the order in which it appear in the body of the class.

The open-book view that is generated by the proposed visualization helps developers in many things. For instance, the table of contents view is used to view names of all packages and their classes in an ordered list. Furthermore, developer can see names of all classes and their sizes in one view. Moreover, developer can use the proposed views to compare between packages and classes based on their sizes.

A Visualization Example

This section details BookViews via a java source code example. Consider a java project named "Student Registration". The project has three packages; Student, Registration, and Database. Figure 4 shows a snapshot from the Eclipse interface of the Student Registration project. The project has 16 methods that are distributed over five classes as shown in table 1.

The visualization of the project using BookViews is shown in figure 5. The close-book view is shown in (a) and the open-book view is shown in (b). The title of the book is the name of the project. The table of contents shows the detailed contents of the book. The book has three chapters that correspond to the three packages in the project. Also, the visualized book has five sections, one section for each class. The first chapters are ordered alphabetically. The Database package is given the sequence number one, followed by Registration and Student packages.

In each chapter, the sections are also ordered by their names. For the Database chapter, Connection section is given the sequence number 1.1 followed by Operations section with sequence number 1.2. The same applies for the Registration chapter. ManageReg section is numbered 2.1 and Registrar section is given 2.2 number.

Each method in the project is given a unique sequence number. The numbering is given based on the sequence number of the section and the order in which the method appears in the body of the class. Page numbers are shown with the names of sections (pp.). These page numbers represents the number of methods defined in the class. The Connection section is marked with (pp. 1-3). This means that the Connection.java class has three methods numbered from 1 to 3. In this example, the numbering of methods starts from the first method defined in the body of class Connection. The Registrar section has page numbers (pp. 11-11), which means that Registrar.java class has only one method.

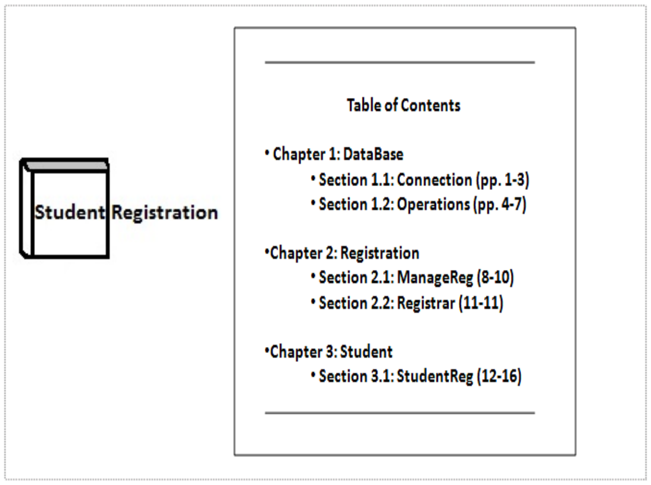


Table of Contents

- Chapter 1: DataBase
  - Section 1.1: Connection (pp. 1-3)
  - Section 1.2: Operations (pp. 4-7)
- Chapter 2: Registration
  - Section 2.1: ManageReg (8-10)
  - Section 2.2: Registrar (11-11)
- Chapter 3: Student
  - Section 3.1: StudentReg (12-16)

(a) Close-book view (b) Open-book view

Fig. 5. BookViews Snapshots for the Student Registration Project.

The total number of pages is the total number of methods defined in the project. The open-book view enables developers to determine which class has the largest number of methods by looking to page numbers of sections. From the example, class StudentReg is the largest class because it has five methods (pp. 12-16). They also can compare packages based on their sizes. In this example, Student package is the smallest package because it has only one section.

The Approach

The models of the BookViews visualization are automatically generated by a tool. The tool reads a java project and generates its corresponding models that are used in the visualization. BookViews generates the models for both open and close book views for the target project automatically. Then, the close-book view is visualized on the screen for the user. The open-book view is generated after clicking on the closed book. The proposed visualization can be used as a plug-in for Eclipse IDE, and it is implemented in two phases.

In the first phase, information about the main structural contents of each project is extracted. This process is done by generating the corresponding Javadoc for the target project. Then, statically analyzed the generated Javadoc files by a set of proposed techniques to automatically extract and explore structural information of the project's contents. The generated Javadoc files, which are written in HTML, are parsed to identify the appropriate HTML tags that contain the project's information.

The tool automatically browses the contents of the HTML code for the generated Javadoc files to extract names of packages. After that, the source file of each package is parsed to extract the names of classes. For each class, a list of method names is extracted, and each method is numbered based on the order of its definition in the project.

The second phase in the proposed approach is visualizing the generated information. The proposed visualization gives a summary about the contents of a software project or a set of projects for developers. In the proposed visualization, each project is summarized and visualized as a virtual book in two dimensional views. Each project is rendered as a book that contains chapters, sections, and pages.

The main structure of the proposed visualization approach is shown in Figure 6. As shown in Figure 6, the Javadoc files, which are generated from the source code of the target project, are processed by a set of processes to automatically extract useful information for the second phase in the approach, *Project Modeler*. These processes are packages, classes, and methods extractors, which implement the first phase in the approach.

First of all, *Package Extractor* traces the generated project's Javadoc files to know how many packages in the project, as well as, the names of these packages. As a result, a list of packages is generated in this process. Secondly, *Class Extractor* traces the Javadoc code for each package in the generated package list to explore its classes. Then, *Class Extractor* generates a list of class names for each package in the project. After finding number of classes in each package, *Method Extractor* calculate number of methods in each class in the generated classes list. Finally, *Project Modeler* process all generated lists to model the target project as a book.

RESULTS AND DISCUSSION

The goal of the proposed BookViews visualization is to support program comprehension for java software projects. Developers who view BookViews are assumed

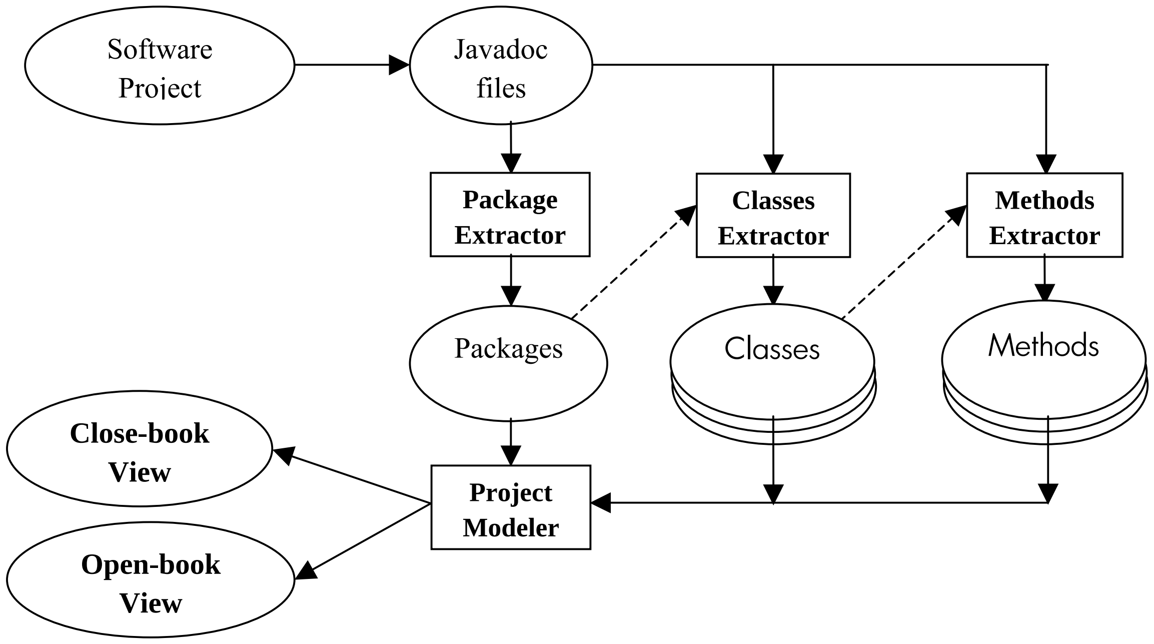


Fig. 6. Main Components of the Proposed Visualization.

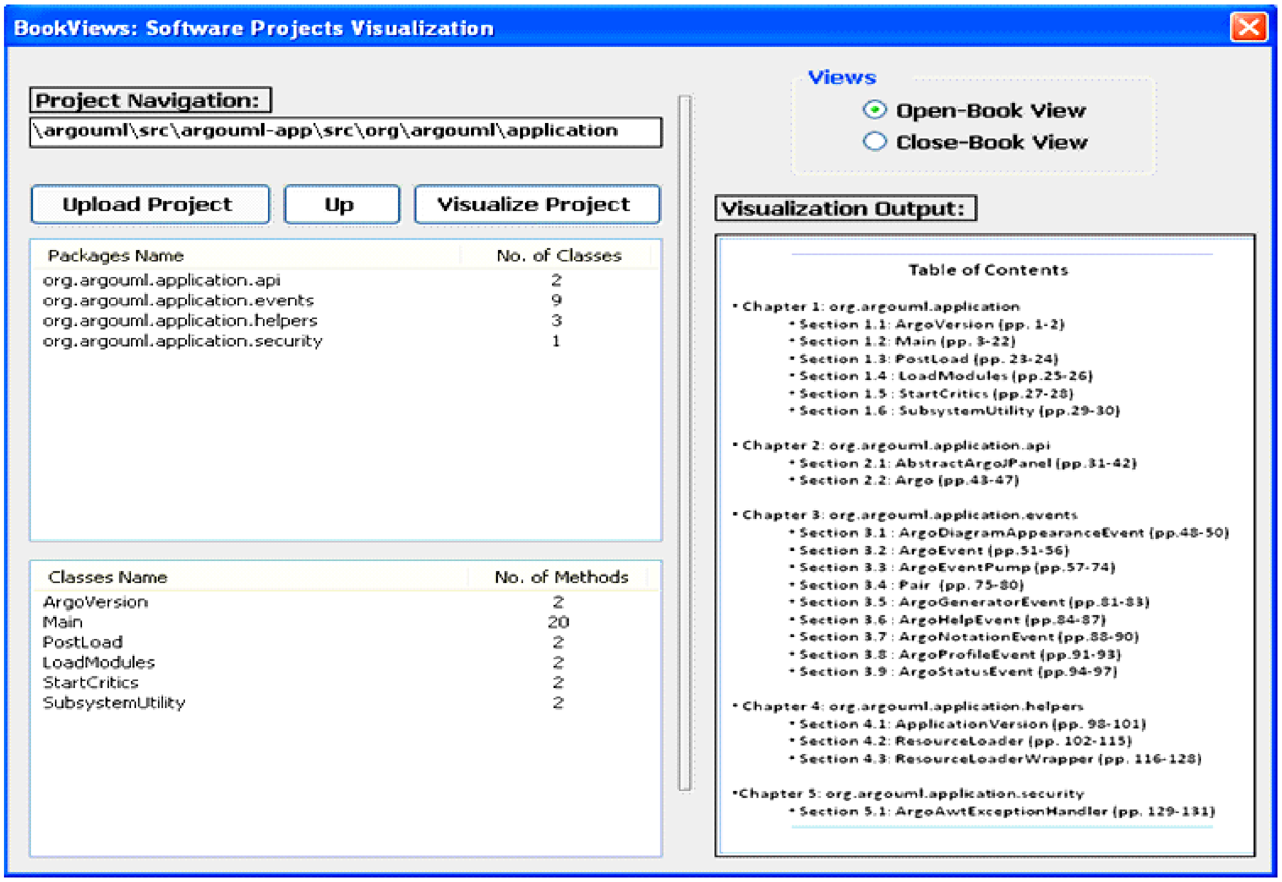
to understand the structure of the project more quickly and more clearly. To evaluate the effectiveness of BookViews, we did an experimental study on using BookViews. BookViews is used to visualize selected parts, chosen randomly, from the open source project ArgoUML (http://argouml.tigris.org/). ArgoUML is a tool, written in java, for modeling UML diagrams. The generated views are checked manually and compared with the source code to be sure about the correctness of the generated views.

Figure 7 shows a snapshot for the BookViews interface that show the open-book view for some contents of package *argouml\src\argouml-app\src\org\argouml\application* in ArgoUML open source project. Developers can explore any part of the project by clicking on a specific package on the project. In Figure 7, The Application package is considered as a separate project that is visualized using the open-book view as a complete project.

Two groups of developers are used in the experiment. All developers have a good knowledge in java and object oriented programming. Each group consists of ten members. The first group was given only the source code from the ArgoUML project. The second group of developers was given the same source code with the BookViews tool.

The two groups were given the same set of questions that need to be answered. The questions are about the contents of the ArgoUML project. For each developer, the time required to answer each question was recorded. The questions that given to each developer are:
Q1) How many classes and packages in the given code?
Q2) How many methods in the given code?
Q3) Which class has the largest number of methods?
Q4) Which package has the largest number of classes?

The time needed to answer each question for each developer was recorded. Then, the average time for each developers group was generated. Figure 8 shows average time needed to answer each question by the two groups. Collected results, as shown in figure 8, showed a big difference between the recorded times for the two developer groups. Results show that by using BookViews, developers can clearly answer the experiment questions much faster.



\argouml\src\argouml-app\src\org\argouml\application

[✓ Open-Book View] Close-Book View

Packages Name	No. of Classes
org.argouml.application.api	2
org.argouml.application.events	9
org.argouml.application.helpers	3
org.argouml.application.security	1

Classes Name	No. of Methods
ArgoVersion	2
Main	20
PostLoad	2
LoadModules	2
StartCritics	2
SubsystemUtility	2

Table of Contents

- Chapter 1: org.argouml.application
  - Section 1.1: ArgoVersion (pp. 1-2)
  - Section 1.2: Main (pp. 3-22)
  - Section 1.3: PostLoad (pp. 23-24)
  - Section 1.4 : LoadModules (pp.25-26)
  - Section 1.5 : StartCritics (pp.27-28)
  - Section 1.6 : SubsystemUtility (pp.29-30)
- Chapter 2: org.argouml.application.api
  - Section 2.1: AbstractArgoJPanel (pp.31-42)
  - Section 2.2: Argo (pp.43-47)
- Chapter 3: org.argouml.application.events
  - Section 3.1 : ArgoDiagramAppearanceEvent (pp.48-50)
  - Section 3.2 : ArgoEvent (pp.51-56)
  - Section 3.3 : ArgoEventPump (pp.57-74)
  - Section 3.4 : Pair (pp. 75-80)
  - Section 3.5 : ArgoGeneratorEvent (pp.81-83)
  - Section 3.6 : ArgoHelpEvent (pp.84-87)
  - Section 3.7 : ArgoNotationEvent (pp.88-90)
  - Section 3.8 : ArgoProfileEvent (pp.91-93)
  - Section 3.9 : ArgoStatusEvent (pp.94-97)
- Chapter 4: org.argouml.application.helpers
  - Section 4.1: ApplicationVersion (pp. 98-101)
  - Section 4.2: ResourceLoader (pp. 102-115)
  - Section 4.3: ResourceLoaderWrapper (pp. 116-128)
- Chapter 5: org.argouml.application.security
  - Section 5.1: ArgoAwtExceptionHandler (pp. 129-131)

Fig. 7. Snapshot from the Visualization tool shows BookViews with Open-book View for Part of ArgoUML Open Source Project.

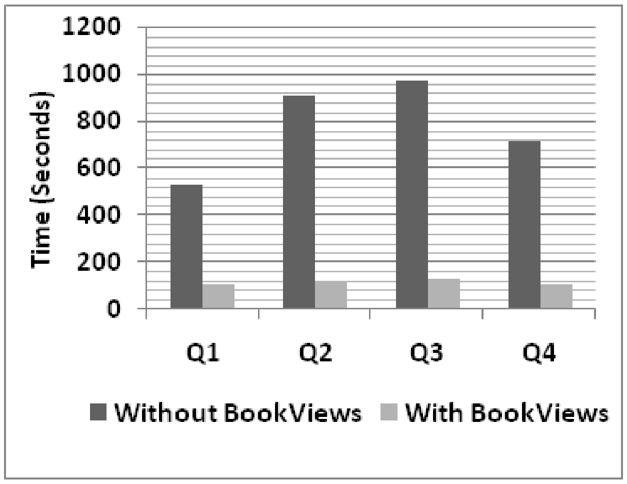


Fig. 8. Average Time (in Seconds) needed to Answer Each Question by the Two Developer Groups.

Additionally, for developers who used BookViews we asked them to evaluate the generated views. We asked them if BookViews helps in understanding the contents of software projects ("very useful", "useful", "not useful"). Eight out of ten developers found the views "very useful". The views were found just "useful" by two developers.

The results of the experiment showed that BookViews visualization save time and efforts of developers who want to understand or browse the internal contents of a project. It is also shown that developers considered BookViews useful in helping them to understand a project's contents.

CONCLUSION

A visualization methodology has been proposed to provide a visual summary for the contents of software projects. The visualization helps developers to quickly understand the contents of software projects in terms of packages, classes and methods. A visualization tool, called BookViews, has been developed to automatically generate the models that are used for visualization. The generated views provide useful summaries for software projects and are used to visually compare between projects based on their sizes. BookViews visualization has been evaluated by a preliminary experimental study. The results of the study showed the effectiveness and the usefulness of the views for software browsers.

The work can be extended in many ways and these extensions are under consideration. One direct extension is to include further information about methods in book pages. Information about methods names, parameters and comments can be included in pages. Another extension is to include more information in the book. Testing files, documentations and configuration files are examples for such information. We also aim to include names of programmers who participated in the development process.

REFERENCES

Alam, S. and Dugerdil, P. 2007. Evospaces visualization tool: Exploring software architecture in 3D. Proc. IEEE 14th Working Conference on Reverse Engineering (WCRE'07). 269-270.

Anslow, C., Marshall, S., Noble, J. and Biddle, R. 2013. SourceVis: Collaborative software visualization for co-located environments, Proc. First IEEE Working Conference on Software Visualization (VISSOFT'13). 1-10.

Beck, F. and Diehl, S. 2010. Visual comparison of software architectures, Proc. 5th International Symposium on Software visualization (SOFTVIS '10). 183-192.

Caserta P. and Zendra, O. 2011. Visualization of the Static Aspects of Software: A Survey Transactions on Visualization and Computer Graphics. 2011. IEEE. 17(7):913-933.

Hammad, M., Abufakher, S. and Hammad, M. 2014. A Visualization Approach for Bug Reports in Software Systems. International Journal of Software Engineering and Its Applications. 8(10):37-45

Hammad, M. and Rawashdeh, A. 2014. A Framework to Measure and Visualize Class Coupling. International Journal of Software Engineering & Its Applications. 8(4):137-146.

Holy, L., Snajberk, J., Brada, P. and Jezek, K. 2013. A Visualization Tool for Reverse-Engineering of Complex Component Applications, Proc. IEEE International Conference on Software Maintenance (ICSM'13). 500-503.

Kobayashi, K., Kamimura, M., Yano, K., Kato, K. and Matsuo, A. 2013. SArF Map: Visualizing Software Architecture from Feature and Layer Viewpoints, Proc. IEEE 21st International Conference on Program Comprehension (ICPC 2013). 43-52.

Lanza, M. and Ducasse, S. 2001. A Categorization of Classes based on the Visualization of their Internal Structure: the Class Blueprint, Proc. 16th International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'01). 300-311.

Ma, KL. 2008. StarGate: A Unified, Interactive Visualization of Software Projects, Proc. IEEE Pacific Visualization Symposium (PacificVIS'08). 191 - 198.

Maletic, J., Leigh, J., Marcus A. and Dunlap, G. 2001. Visualizing object-oriented software in virtual reality, Proc. IEEE 9th International Workshop on Program Comprehension (IWPC'01). 26-35.

Maletic, J., Marcus, A. and Collard, M. 2002. ATask Oriented View of Software Visualization. In Proceedings of the 1st International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT '02). IEEE Computer Society, Washington, DC, USA. pp32.

Novais, R., Torres, A., Mendes, T., Mendonça, M. and Zazworka, N. 2013. Software evolution visualization: A systematic mapping study, Information and Software Technology. 55(11):1860-1883.

Ogawa, M. and Ma, KL. 2009. code_swarm: A design study in organic software visualization. IEEE Transactions on Visualization and Computer Graphics. 15(6):1097-1104.

Sommerville, I., Cliff, D., Calinescu, R., Keen, J., Kelly, T., Kwiatkowska, M., Mcdermid, J. and Paige. R. 2012. Large-scale complex IT systems. Communications of the ACM. 55(7):71-77.

Wettel, R. and Lanza, M. 2008. Visual exploration of large-scale system evolution, Proc. IEEE 15th Working Conference on Reverse Engineering (WCRE'08). 219-228.

Zhi, J. and Ruhe, UDE. 2013. A tool for visualizing software document evolution. Proc. 2013 First IEEE Working Conference on Software Visualization (VISSOFT '13). 1-4.

Received: Jan 12, 2015; Revised: Feb 21, 2015;
Accepted: March 12, 2015