Identifying Designers and their Design Knowledge

Maen Hammad¹, Mustafa Hammad² and Hani Bani-Salameh³

¹Department of Software Engineering, The Hashemite University, Zarqa, Jordan
²Department of Information Technology, Mu'tah University, AL Karak, Jordan
³Department of Software Engineering, The Hashemite University, Zarqa, Jordan
¹mhammad@hu.edu.jo, ²hammad@mutah.edu.jo, ³hani@hu.edu.jo

Abstract

This paper presents an approach to identify designers who participated in shaping the design of software systems. Designers are identified as well as their design knowledge. Identified designers are recommended to handle a high level change request. High level requests require changing the design of software systems. The identified design knowledge of designers is used to find the appropriate designer to handle a change request. Designers are developers who made design changes to the software. Identifying designers and their design knowledge is based on mining software repositories to extract design changes from commits. Developers who committed any design change are considered designers. Type and amount of committed design changes are used to identify the knowledge of designers. A tool has been developed to automatically recommend a set of designers to handle a high level change request. Recommended designers are ranked based on their knowledge that is identified from cumulative design activities for designers. The tool also extracts useful information and statistics about designers and their activities. A case study of two open source projects is presented to show the benefits of the proposed approach and tool. Results showed that most committers are not designers. It also shows that designers can be ranked based on their design knowledge.

Keywords: software design, change request, software maintenance

1. Introduction

Software systems are subject to change for different reasons. In general, each change is a response to a change request. Change requests are analyzed and then assigned to a developer or group of developers to handle and implement the request. Examples of change requests are bug fixes, adding new features, adding new functionalities, refactoring etc. Usually, project managers or team leaders are responsible for assigning change requests to appropriate developers. Finding the correct developer, who can perform the request, helps in applying the change quickly and correctly.

It is not an easy task to identify the appropriate developer, or group of developers, to implement the change request. This is because the knowledge or experience of developers is not well defined and change requests have different types. In open source projects, the problem is harder to solve. This is due to the development nature of open source projects. Open source projects are usually maintained by voluntary developers who are distributed over remote geographical areas with different expertise. Moreover, there is no central management to control the development process as closed projects. In such development environment, it is difficult to find the appropriate developer for a change request. Automatic identification of designers, who could update the design in order to implement the new requests, saves time...
and efforts. It saves time and effort for team leaders who manage the requests and the task assigning process especially in open source projects.

In this paper, the focus is on high level change requests that directly impact the design of software systems. These high level changes require more time and efforts to be implemented. For example, adding a new class to the source code needs more efforts than just adding a loop or a conditional statement. Suppose there are 60 developers that are involved in maintaining a large software project. Not all of the 60 developers are able to update the design in order to handle the high level change request. The problem rises in locating designers who can update the design to implement high level change requests such as adding features. So, finding the appropriate designer automatically saves the time that is used in the searching process.

The high level change requests, which are under our consideration in this paper, are the addition/deletion of features, addition/deletion of functionalities, and applying design enhancements. Adding new features or functionalities requires changing the design of the system by adding classes and methods. Design enhancement changes also impact design by merging classes, changing relationships, extract method etc.

In this paper we present an approach supported by a tool to automatically assign a high level change request to an appropriate designers. The developed tool recommends a set of designers ranked by their amount of design knowledge they have. The tool also provides useful statistics about designers and their design activities. Another feature of the tool is the building of a design knowledge graph. The graph models the relationships between designers and classes they used during their design changes activities.

The main research contributions addressed in this paper are:

1. An approach to identify designers by mining archives of software repositories.
2. An approach to identify the type of design knowledge for designers.
3. A tool to automatically:
   - Recommend list of designers to deal with a high level change request.
   - Provide statistics about designers and their activities.
   - Build a knowledge graph for designers.
4. A case study to investigate the proposed approaches and the developed tool.

The approach is based on mining software repositories that are managed by subversion systems. The unit of change is the commit. Identification of commits with design impact provides valuable information about identifying developers with design knowledge. Detailed design changes committed by designers can be used to identify the design knowledge of designers. For example, a developer who added a new class to the source code has more design knowledge than a developer who added a loop or a conditional statement.

The rest of this paper is organized as follows. Section 2 presents our approach of identifying designers. Identification knowledge of designers is detailed in Section 3. Section 4 details a case study on two open source projects. Related work is discussed in Section 5 followed by conclusions and our future work.

2. Identifying Designers

Some developers do not update the design of software systems. For example, adding a parameter to a method or adding a conditional statement does not impact the design. On the other hand, adding a new class or a method are considered design changes. We define a designer as the developer who committed at least one commit that impacted the design. This developer is categorized as a designer. The unit of change that we consider is the commit. Any commit, that causes addition or removal of a class, a
method, or a class relationship (i.e., generalization, association, or dependency) will be considered as a design impact commit. In other words, code changes that make changes to UML class diagram of the source code are considered design changes.

Design changes are identified by the design change identification tool srcTracer [1, 2]. The tool works by representing both versions (old and new) of the source code in the XML representation srcML [3]. Then both versions are represented in the XML differencing format srcDiff [4]. Design changes are then extracted from srcDiff by using a set of XPath queries. The process of identifying if the developer is a designer is summarized as follows:

1. Extract group of commits S from a software repository.
2. For each extracted commit, analyze code changes by the srcTracer tool to identify design changes.
3. For all committers of S, if at least one committer had a commit with a design change then that developer is a designer.

The approach is based on analyzing histories of code changes. Suppose we want to identify designers who participated in shaping the current design of a software system in the last year. A design change is defined as the addition or deletion of a class, a method, or a relationship (i.e., generalization, association, dependency) in the corresponding UML class diagram of the source code. First, all commits, of the last year, are extracted from the repository of code changes. In the next step, all extracted commits are analyzed by the srcTracer tool to categorize them as commits with design changes or commits with no design changes. Finally, the committers of the commits with design changes are reported as designers.

### 3. Identifying Knowledge of Designers

Designers vary in their knowledge in the evolved design of the system. We need to identify designers with specific type of knowledge. This knowledge or expertise can be identified by examining the type of design changes committed by designers. The cumulative design changes activities, during time period, are used to identify level of design knowledge of designers. We distinguish between three levels of knowledge in design; the depth knowledge in design, the broad knowledge in design, and the architecture knowledge in design. Three historic based measures are proposed to categorize designers into one of the three levels of knowledge. These measures are:

1. The total number of added/deleted methods by designer.
2. The total number of added/deleted classes by designer.
3. The total number of unique classes used in design changes by designer.

The number of added/deleted methods is used to measure the depth knowledge. Designers who added/deleted the largest number of methods have the highest depth knowledge. The amount of broad knowledge is measured by the number of added/deleted classes. Finally, the number of classes used by designers during their design activities is used to measure the broad knowledge.

The identification of design knowledge is based on cumulative historical design changes of designers. So, results may vary based on the time duration and number of extracted commits.

#### 3.1. Depth Knowledge in Design

The number of added/deleted methods determines the depth knowledge of designers. Designers who mainly participated in adding or deleting methods over time have good knowledge in the detailed functionality of the system.
For example, a developer added new methods to a class. So, this developer has to know the
detailed functional requirements for the class and the system. As a result, this developer will
get more knowledge in the detailed functionality of the system.

Our premise is that designers who have high depth knowledge can be chosen to apply
perfective maintenance tasks that include adding new functionality. Adding new functionality
to the system requires adding new methods. Developers with highest depth knowledge could
be the best choice to do the tasks.

3.2. Architectural Knowledge in Design

The number of added/deleted classes measure is used to determine the architectural
knowledge of designers. These designers have knowledge about the big picture of the design.
They participated in organizing and controlling the number of classes in the design.

For example, consider a designer who contributed in adding three new classes during
specific time duration. So, he has to know the architecture of the software in order to place
the new classes in the correct position (subsystem, package, inheritance tree etc.). As a result,
this designer has a good knowledge about the architecture of the software.

Our premise is that designers with high architectural knowledge can help in developing
new features in the system. Adding new features mostly requires adding new classes. So,
any change request with requires adding new features, designers with highest architectur al
knowledge could be the right developers who can handle the request.

3.3. Broad Knowledge in Design

More classes used by the designer mean more broad knowledge in design for that designer.
The number of different (unique) classes a designer used during his design changes activities
reflects the amount of knowledge he gained. Knowing classes by designers is the broad
knowledge they have.

For example, a developer added a method but he called five methods in five different
classes in his implementation to the new method. So, this designer knows the structure and
functionality of the five classes he used. This means he has a broad knowledge across many
and different classes.

Designers with high broad knowledge may be the appropriate designers who can help in
solving high level design problems as design enhancements. Reducing the coupling between
classes is an example. The designer who performs this maintenance activity should have a
good knowledge about most of the classes and their relationships. Knowing most of the
classes means high broad knowledge.

3.4. Handling Change Requests

A tool has been developed to automatically handle high level change requests. The name
of the tool is DesignHandler. The tool prompts the user to determine the type of change
request. The options are; add/delete feature, add/delete functionality, or design enhancement.
Based on the selected option, a list of developers is recommended to the user. For the feature
option, designers with highest architecture knowledge are recommended. Designers with
highest deep knowledge are recommended for the functionality change request. Design
enhancement tasks require knowledge in many or most classes. So, designers with the highest
broad knowledge are recommended to handle design enhancement change requests. The
process of handling a change request by the DesignHandler tool is summarized as follows:

1. The amount of historical code changes is determined by the user.
2. The type of change request is determined by the user as add/delete a feature, add/delete functionality, or do design enhancement.

3. The tool analyses the history to recommend a list of designers as follows:
   - For add/delete features change requests: designers with top architectural knowledge are recommended.
   - For add/delete functionalities change requests: designers with top deep knowledge are recommended.
   - For do design enhancements change requests: designers with top broad knowledge are recommended.

The maximum number of recommended designers is determined by the user of DesignHandler. The tool also provides the following information and statistics to the user.
   - Determining if a developer D is a designer or not.
   - Providing a list of all designers ranked based on their design knowledge.
   - Identifying the number of designers among developers.
   - Identifying the designer who added/deleted a specific design element.
   - Building a design knowledge graph for designers.

The above information and statistics are useful for project managers. For example, if a problem is detected in a method M, the designer who added the method M may be contacted by the project manager to fix that problem. The DesignHandler tool can identify the designer who added method M. The DesignHandler tool analyses design knowledge of designers to extract common design knowledge of designers. The analyzed design knowledge are used to build a design knowledge graph. The graph helps to identify designers who have common design knowledge. It also helps to extract other useful information that will be discussed in Section 4.

The DesignHandler tool is implemented as an Eclipse plug-in. It mines the history of code changes to identify designers, their design knowledge, and to build the design knowledge graph. The time period and amount of history is determined by the user. Larger time duration means larger history to mine by the tool and hence more representatives results. Figure 1 shows the structure and the main components of the DesignHandler tool. First of all, users have to determine the URL of the repository under consideration. The time duration, start and end dates, for the history need to be also determined. The Commit Extractor component is responsible for extracting all commits between the start and end dates determined by the user. The component mainly access repositories of open source projects that are under the control of subversions systems. All extracted commits are analyzed by the srcTracer tool to identify design changes from code changes. As a result, design changes are identified and developers responsible for those design changes are also identified as designers. Users then use the interface of the DesignHandler tool to determine the type of change request as one of the followings:
   - Recommend designers for a design change request
   - Build design knowledge graph for designers.
   - Find statistics about designers.

The above requests are handled by the Request Handler component. Both identified designers and their design changes are used by the component to handle user’s requests. For recommendations, a list of designers is viewed to the user based on the type of design change request. The component also generates a model for the design knowledge graph for designers.
It also provides some statistics about identified designers as the number of total designers, list of all designers ranked by their design knowledge, and the number of added/deleted design elements by a specific designer.

4. Case Study

We analyzed code changes from repositories on two C++ open source projects. The two open source projects are; the KDE editor Kate (http://kate-editor.org), and the KOffice spreadsheet KSpread (www.koffice.org/kspread). A subset of commits from these two projects have been extracted and analyzed. The extracted commits cover three years time duration. The extracted commits were analyzed by the DesignHandler tool to identify designers and to rank them based on their design knowledge. The reported results are analyzed and discussed in the following subsections.

4.1. Identifying Designers

As explained in Section 2, a designer is any developer who committed at least one design change. Based on this assumption, the extracted commits are analyzed by DesignHandler. We need to identify who are the designers and what is their percentage among developers. Designers vs. non-designers results of the two studied projects are shown in Table 1. The first column is the name of the open source project. The second column represents the total number of developers (committers) for the extracted commits during the three years. The (#Designers) column shows the total number of designers identified by the method described in Section 2.

<table>
<thead>
<tr>
<th>#Developers</th>
<th>#Designers</th>
</tr>
</thead>
<tbody>
<tr>
<td>KDE-Kate</td>
<td>104</td>
</tr>
<tr>
<td>KDE-KSpread</td>
<td>62</td>
</tr>
<tr>
<td>Total</td>
<td>166</td>
</tr>
</tbody>
</table>

For Kate and KSpread designers are less than 50% of all developers, 47% and 38% respectively (43% in average). This means most of the committers do not make any change to the design. So, their contribution is limited to simple code changes that do not impact the design. In this case, developers (not designers) could be excluded from the candidate list of handling a high level change requests. Designers may be the right persons to be assigned high level change requests. In this case, the list of candidates is shorter than a list of candidates that includes all developers.
In summary, results show that most maintainers usually do not commit design changes and hence most of them are not designers.

4.2. Knowledge of Designers

Section 3 discussed three proposed historic-based measures to identify and rank designers based on their knowledge in design. Based on these measures, designers can be ranked according to their knowledge in design. The developed DesignHandler tool has been applied on the extracted commits from Kate and KSpread. Table 2 shows the top 12 designers of Kate ranked by the three different design knowledge measures. For example, designer cullmann added and deleted 58 classes. So, he has the highest architectural knowledge among other designers and his rank is the first. He also is ranked the first based on the deep knowledge and the broad knowledge measures.

Table 2. Top 12 Designers of Kate Ranked by the Three Design Level Knowledge Measures

<table>
<thead>
<tr>
<th>Designer</th>
<th>Architectural Knowledge</th>
<th>Deep Knowledge</th>
<th>Broad Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+/- Classes</td>
<td>Rank</td>
<td>+/- Methods</td>
</tr>
<tr>
<td>Cullmann</td>
<td>58</td>
<td>1</td>
<td>432</td>
</tr>
<tr>
<td>Rodda</td>
<td>18</td>
<td>2</td>
<td>399</td>
</tr>
<tr>
<td>Dhaumann</td>
<td>17</td>
<td>3</td>
<td>179</td>
</tr>
<tr>
<td>Ehamberg</td>
<td>11</td>
<td>4</td>
<td>214</td>
</tr>
<tr>
<td>Ereslibre</td>
<td>7</td>
<td>5</td>
<td>18</td>
</tr>
<tr>
<td>Savernik</td>
<td>7</td>
<td>6</td>
<td>23</td>
</tr>
<tr>
<td>Zwabel</td>
<td>6</td>
<td>7</td>
<td>138</td>
</tr>
<tr>
<td>Jowenn</td>
<td>4</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td>Kling</td>
<td>2</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>Sping</td>
<td>1</td>
<td>10</td>
<td>69</td>
</tr>
<tr>
<td>Alund</td>
<td>1</td>
<td>11</td>
<td>19</td>
</tr>
<tr>
<td>Dimsuz</td>
<td>1</td>
<td>12</td>
<td>20</td>
</tr>
</tbody>
</table>

The rank of designers based on their design knowledge varies. For example, ereslibre is ranked fifth based on the architectural knowledge and the broad knowledge measures. But he is the 11th based on the deep knowledge measure. It means that this designer has more knowledge about the big picture of the design (classes) than the functionality of the system (methods).

The ranking is useful to identify designers to perform various maintenance tasks. For example, to implement a new functionality, designers with top deep knowledge are considered first. To add new features, designers with top architectural knowledge are recommended. Designers with high broad knowledge are considered to solve high level design issues as coupling and system decomposition.

DesignHandler recommends designers based on their rank. For example, consider a change request of type add/delete a feature for Kate project. Suppose that the maximum number of recommended designers is set to three. The DesignHandler tool recommends, in order, designers: cullmann, rodda, dhaumann. But, if the change request is about adding functionality, designers: cullmann, rodda, ehamberg are recommended. Both recommendations differ in the third designers. Designers cullmann, dhaumann, rodda are recommended in case a design enhancement is requested.
The same applies for the results obtained from KSpread project. Designers are ranked based on their design knowledge. They also have different ranks based on the type of their design knowledge.

In summary, results show that designers have different design knowledge and hence they can be ranked based on the level of that design knowledge.

4.3. Design Knowledge Graph

Relationships between designers and classes are used to build a graph. The graph is called a design knowledge graph. The design knowledge graph connects designers with classes they used during design changes activities. The designer-class relationship represents the knowledge of the designer (i.e., class name). Frequently used classes during design evolution by a specific designer indicate the knowledge area of that designer. This knowledge area is determined by these classes.

Figure 2 shows a design knowledge graph for a subset from Kate’s designers. DesignHandler generated the representation of the graph. Then, the generated representation has been used by the Pajek software (http://pajek.imfm.si/doku.php) to draw the graph. Each designer is represented by a red node. Classes are represented by black nodes. Edges between designers and classes represent the using relationship. Each designer is connected to classes that he used during various design changes activities he committed over time. The graph in figure 1 represents three years time period for shown designers.

Visualizing the graph helps to extract many useful information about designers and classes. The graph helps to visualize the design knowledge of each designer by looking to the classes connected to him. It also helps in locating classes that are maintained or used by only one designer over a long period of time. This type of classes should be documented carefully because only one designer knows about them. In case new designers want to use or understand such a class, it will not be an easy task without documentation. For example designer kling is the only developer who maintained or used KateSearchBar class. Some classes are clustered around one designer only. For example, class KatePythonIndent is used by only one designer; bram. This locates the design knowledge of bram to these classes. Some designers have common design knowledge. Hausman and Ervin used classes KDataToolLuginView and KAction. The graph also helps to identify classes that have been used by many designers. KateView is an example of such a class. It was used by three designers; kling, mbroast, and staikos.

As a result, the design knowledge graph provides useful information about designers and their design knowledge. It is also useful to identify relationships between designers based on their design knowledge.

![Design Knowledge Graph for some Designers of Kate](image)
5. Related Work

Kagdi et al., [5] presented an approach with a tool named xFinder. The approach recommends expert developers by mining version archives of a software system. The basic premise of this approach is that the developers who contributed substantial changes to a specific part of source code in the past are likely to best assist in its current or future changes. The approach is used to recommend experts for files. The approach has been used in [6, 7] to recommend a ranked list of expert developers to assist in the implementation of software change requests. An Information Retrieval based concept location technique is first used to locate source code entities, relevant to a given textual description of a change request. Our approach is a light weight approach that does not use IR techniques. It also does not involve feature location methods. We recommend a list of designers to handle specific type of change requests determined by the user.

McDonald and Ackerman [8] developed a heuristic-based recommendation system called the Expertise Recommender (ER) to identify experts at the module level. Developers are ranked according to the most recent modification date. Mino and Murphy [9] presented a tool called Emergent Expertise Locator (EEL). Their tool helps to find the developers who can assist in solving a particular problem. Expertise Browser (ExB) [10] is another tool to locate people with the specific expertise based on their previous code changes. Surian et al., [11] presented an approach to recommend a list of developers that are most compatible with an input developer. The approach is based in graph analysis. It extracts a graph from past history of projects and developers working on them. The graph contains three types of nodes: developers, projects, and project properties. A detailed analytical study about mining collaboration patterns from large developer network is presented in [12]. Bird et al., [13] mined e-mail archives to analyze the communication and co-ordination activities of the participants. Del Rosso [14] used collaborations and interactions between knowledge-intensive software developers to build a social network to locate developers with a wide expertise on the project.

Anvik, et al., [15] presented an approach to help in process of assignment bug reports to a developer. The approach applies a machine learning algorithm to the open bug repository to learn the kinds of reports each developer resolves. Another bug report assignment approach is discussed in [16]. The approach is based on modeling developer expertise using the vocabulary found in their source code and compares this vocabulary to the vocabulary of bug reports.

In the field of measuring and identifying developers’ contributions, Gousios et al., [17] proposed a model for evaluating developer contribution. The model creates clusters of similar projects to extract weights that are then applied to the actions a developer performed. Many studies as in [18, 19] showed that most of code changes are contributed by small number of developers. Ben et al., [20] examined the contribution characteristics of developers in open source environment based on visual analysis, and presented approaches from three aspects— influencing factors, time characteristics and region characteristics.

Our works differs in focusing on developers who mainly involved in updating the design of software systems. The focus is on their type of design knowledge and handling change requests based on this knowledge.

6. Conclusions and Future Work

This paper presented an approach and a tool to identify designers who could handle a high level change requests. The developed tool automatically mines software archives to identify designers and their knowledge in design. The tool is useful to handle requests in large
software projects with a large number of developers. Applying the approach on two open source projects resulted in many observations. Most committers do not change the design. Also, designers can be ranked based on their design knowledge. A design knowledge graph has been proposed to visualize the relationships between designers and classes they used during design changes activities.

Our future work aims to enhance the performance of the DesignHandler tool by handling other change requests as bug fixes. Another extension of the work is the identification of design changes that are required to implement a high level change request.

References


Authors

Maen Hammad is an Assistant Professor in Software Engineering Department at The Hashemite University, Jordan. He completed his Ph.D. in computer science at Kent State University, USA in 2010. He received his Master in computer science from Al-Yarmouk University—Jordan and his B.S. in computer science from The Hashemite University—Jordan. His research interest is Software Engineering with focus on software evolution and maintenance, program comprehension and mining software repositories.

Mustafa Hammad is an Assistant Professor at Information Technology department in Mu’tah University, Al Karak - Jordan. He received his PhD. in computer science from New Mexico State University, USA in 2010. He received his Masters degree in computer science from Al-Balqa Applied University, Jordan in 2005 and his B.Sc. in computer science from The Hashemite University, Jordan in 2002. His research interest is Software Engineering with focus on static and dynamic analysis and software evolution.

Hani Bani-Salameh is an Assistant Professor in the Software Engineering Department at The Hashemite University since 2011. He holds a B.Sc. in Computer Science (1995), a M.Sc. in Computer Science (2007) from New Mexico State University (NMSU), and a Ph.D. in Computer Science (2011) from the University of Idaho (UI). His research interests include software engineering, computer supported cooperative work (CSCW), software development environments, collaborative software development in virtual environments, and social networking and social media. He studies social interactions in social networks and online environments, including Facebook and SourceForge.