Università
della
Svizzera
italiana

Faculty
of Informatics

Bachelor Thesis

June 3, 2016

CSI:Cube8

Augmenting Software System Representation with Corollary Information

Marco Bedulli

Abstract

The information not strictly related to a software system, likes forum discussions and code documentation, can be useful to understand how much knowledge is available about the source code. Using an augmented city metaphor as visualisation method we allow the developer to evaluate the information coverage. A developer is thus able to visualise which part needs more documentation and also directly access the online information related to it.

Advisor
Prof. Michele Lanza
Assistant
Phd. Luca Ponzanelli

Advisor's approval (Prof. Michele Lanza): Date:

Contents

List of Figures

1 Introduction

The main purpose of this paper is offering to anyone a way to get an impression at first glance about the information coverage of a software in an immediate and intuitively way. It can be seen as the combination of the needs to get a better understanding of the backbones in a project and the needs to find all the available information related to it rendered in a easy and fast system.

Since the web community has plenty of features questions and answers on a wide range of topics in computer programming, having a pre-built application able to show the most popular discussion tightly focused on a specific problem could undoubtedly reduce the amount of time spent on learning all the functionality of the project.

1.1 City metaphor as visualisation method

The city is create using a mix of information related and not to the code that are mapped to construct the building of the city. The use of a metaphor from the physical world is the key point that makes this system particularly intuitive and effective. In fact, it allows the viewer to transfer existing perceptual abilities to the comprehension of the visualisation.
R. P. Gabriel [**?**] said that "Habitability is the characteristic of source code that enables programmer, code ,bag-fixer, and people coming to the code later in life to understand his construction and his intentions[..]".

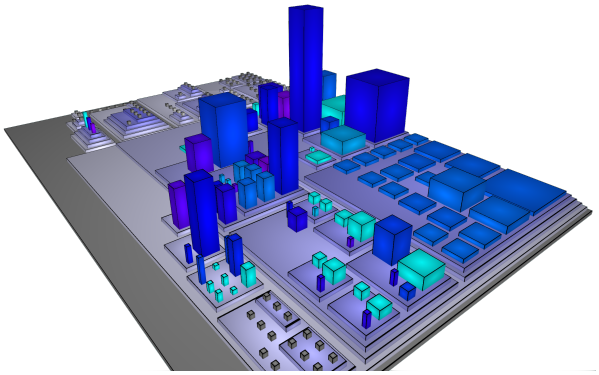


Figure 1. A first example of a city

Starting from this concept, we try to improve this idea of habitability as explained in [**?**] Visualising Software System as City, where this metaphor is used as a way to allows the developers to get a better understanding of a specific software.
As main aim, we thought to help all the people that, coming to the code late during its implementation and development, need to by filled in quickly about all the reference and the information available on it.
By doing that, the user can navigate and interact with all the city's components, from the folder (shown as the basement in the render) to all the file that compose the project (shown as building in the render).

1.2 Corollary Information

Proposition B is a corollary of proposition A if B can be readily deduced from A or is self-evident from its proof, but the meaning of readily or self-evident varies depending upon the author and context. The importance of the corollary is often considered secondary to that of the initial theorem; B is unlikely to be termed a corollary if its mathematical consequences are as significant as those of A. Sometimes a corollary has a proof that explains the derivation; sometimes the derivation is considered self-evident. [**?**]

For instance, the number of class or the Interface in a file, are information that modifies the structure of the whole project because they are the fiscal parts that compose a system; so they all may be considered corollary information. Instead, on the contrary, the comment has not influenced on the result of a project; it means that they can't be considered corollary information. In other words, we can refer to the fiscal part of a project to all the information that you could draw a UML diagram, and the corollary information as all the component that has no design influence on the project, and so are not representable on a UML.
The java doc can be used as a simple example of Corollary information because it is not important for the design proposed, but is extremely useful to understand what the code does. We could also think about the information that is not present, but could be found by using your code. Usually, a software is compose using a different third part

library, and it could be helpful to know how much information are available about a call of that particular library because this information gives a better understanding of the code that we are working on. The corollary information isn't essentially useful to understand the struct of a system but can give a "normal human readable information" related to its structure.

1.3 Importance of code related information

We can't remove the information related to the code during the visualisation process because they give as an intuitively way to understand the topology of the project, and are useful to get a metric unit to better understand what the information coverage means.
It's cool to know how much java doc you have respect to a file, but if you don't know the characteristics of that file, you can't say if the documentation is enough or not. Is also useful to use the purely code relation information to have a main idea about the struct of the system, in which package there are more concentration of classes or methods and for highlight design problem.

1.4 Document Structure

In section 2 we present the related work. We are going to analyse the functionality of system like Cube8 and we explain briefly how stormed works since is use in this project. Then we explain the approach used and the different metrics that we used in section 3. In section 4 we show two different projects and how to use out system and which kind of informations are possible to retrieves. Finally we conclude with some improvement that will be interesting to be implemented.

2 Related Works

Cube8 is a mix of two different sectors and works. One is the visualisation method for a software system and the other one is the way to get the corollary information.

Program Comprehension through Software Habitability [**?**] propose a city metaphor in which there are a fix number of building type such as Skyscraper, Office building, Apartment Block,mansion and House. They propose two mapping: Boxplot-based Mapping and Threshold-based Mapping. Also is using a box-packing algorithm to visualise the city. We are using the same idea of box-packing to organise the city. We also apply the same city metaphor: classes are representing as building located in city districts which in turn represent packages.
The colour meaning is completely different colour meaning since we have to visualise different information. In those paper they are concentrate about the structure of a software, here we would like to visualise the coverage information. We still allows the developer to get an idea about other software propriety like classes and interface apply different metrics and therefore generate a new city.The size of the building are code dependent, this allows a better understanding about the system.

Visualise Software system as Cities [**?**] is also propose a 3d environment in which the software system is represent as a city, whit different class of buildings. It's also implements a way to navigate and interact with the system.is possible to select any artefact and interact with them, spawning complementary views, a tagging system and a query system.
In Cube8 we have only some of this feature like a basic query system that allows to search for file name and perform different actions. Is also possible to read the code and navigate through the information found on the web related to a particular building.

The StORMeD [**?**] gives a dataset of JSON files, one for each discussion that contain an H-AST about the discussions. The discussion parsing happens in two different step:the former consist into HTML tag rules to extract the information unit. The latter concern the effective use of the heterogeneous island grammar.This approach is an extension of Bacchelli [**?**]. We are using simply this dataset to compute the information coverage of the system.

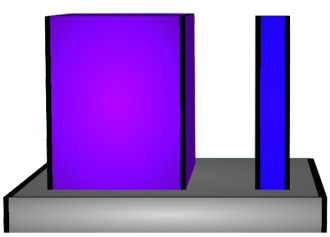
3 Approach

3.1 Introduction

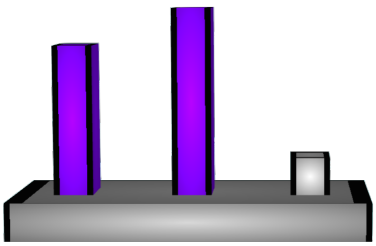
In this section we are going to analyse the approach used in our tool and explain why we are made same decision. The system language that we analyse Java, so a Object Oriented language. This imply that we are going to speak about classes, interfaces methods and fields. The container, or building, are the files and not the classes so on building can have more classes and interfaces.

3.2 Information Strictly related to the code

As describe in the introduction we are used code related information to give to the user a better understanding about the locality of your code. To demonstrate this concept you can see two render about the same code. For simplicity we used a huge system that consist of two classes. ClassA has 4 methods and ClassB has 4 method and the same number of fields. We have to find which class need more documentation. The Java Doc, is express in percentage respect the number of method. The colour goes from light blue to purple: light blue represent the minimum and purple the maximum. In the figure **??** , we can see that the big box at left as full documentation, instead, the right one has less. Therefore we found the class that need more documentation (classA) in an easy way. Suppose you have to use the figure **??**, it has more information not related to the code and has only the method count as code related information. The city become unusable since you can not determinate which is the class A and the class B since the only difference is on the number of fields! Later in this chapter we are going to analyse more in detail colour system and the meaning of the used metrics.



(a) Mapping as Width:N of method, height: Number of field, colour: javaDoc



(b) Mapping as Width:Discussion count, height: Java doc, color: N of method

Figure 2. Information Strictly related to the code

3.2.1 Class and Interface

The classes and interface are another metrics that we add to our tools. Remember that the basic building represented is the source file. By the Java Code Conventions [**?**] "Each Java source file contains a single public class or interface. When private classes and interfaces are associated with a public class, you can put them in the same source file as the public class". This mean that we could have more classes in a single source file and therefore could be useful to have a metrics that give to the analyser the ability to find this relations.

This is an example, figure **??**, where we analyse this two concept in a big project. As you can see there are a few classes that could need some check to make sure that this design principle is respected.

3.2.2 Methods and Fields

Method and field are the main component that compose a class or interface. In the tools we are using this two measure to map the size of the buildings. The reason is that this two concepts give the correct granularity to have a better understand about the system.
We can also identify a potential God Class that has a hight number of method or a Data Class that has a hight number of field and a few methods. Let's get an example. We using the same project as before.

It's easy to see that there is a flat an big building that could represent a Data Class and there are two thigh and height building that could represent a God Class. In this case the both candidate for the god Class are tests. Instead the Data Class has really 686 fields.

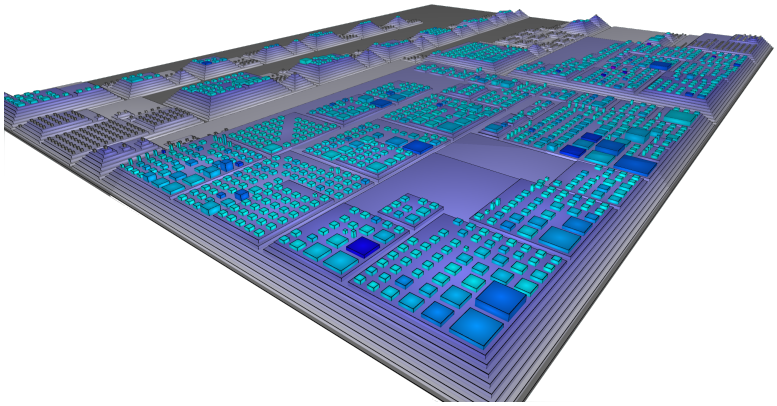


Figure 3. Mapping as Width:N of Class, height: Number of interface

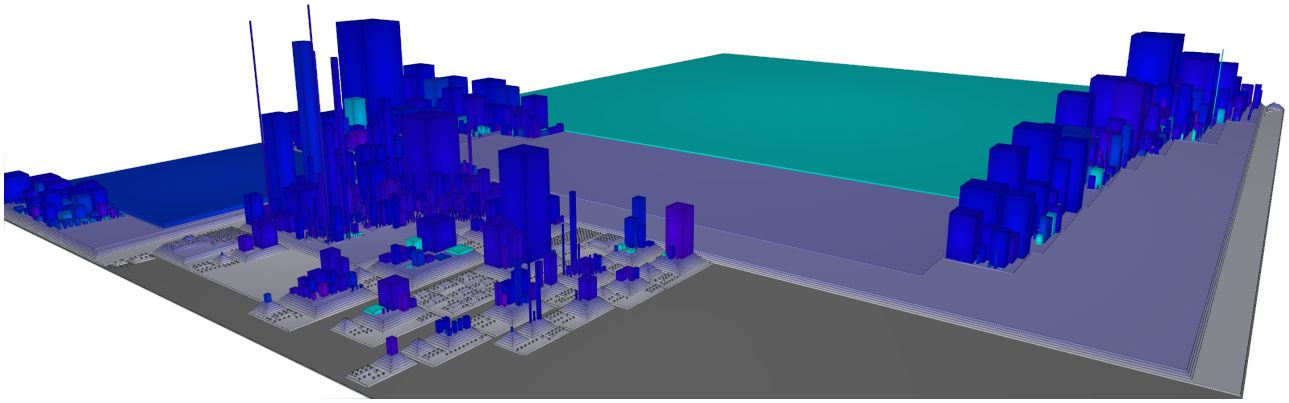


Figure 4. Mapping as Width:N of Fields, height: Number of Method

3.2.3 Identity Harmony

Design disharmonies are formalised design shortcomings to detonate pieces of system that exhibit design problem. With our tool we can only identify some of the identity harmony. Every entity in the system must justify his existence. Does it implement a specific concept or is it doing to many things or does it nothing?

We can identify 3 kind of disharmonies:

- God Class:is a class that does too much . In our representation appear like a big box.
- Brain Class:is a class that accumulate an excessive amount of intelligence,usually has a lot of methods: it's look like an antenna
- Data Class:is a class that hold a lot of data and doesn't perform any operation:it is appear to a be a big and thin box.

3.3 Information Not Strictly related to the code

The information not strictly related to code are the core of this paper. As we sow before, there already exist tools that allows the visualisation of a system as a city, and they does a lot of computation around strict related information. What we interesting, instead, is the amount ok knowledge that are available about a given system. This knowledge are meaningful to get an idea about the complexity of understanding a software system and where it should be use more effort.At the same time it could be use as a monitor for the developers to understand which part of their code need more information.

3.3.1 Java Documentation

Collecting and visualising the java doc was the first step of the process to collecting the coverage information since it is integrated on the code and he doest require any particular computations. It cover an important role in the process of understanding the functionality of a given code since is written directly by the developers and should be use in each method, field and class definitions. With this computation unit is possible to visualise the documentation state of a project. We usually map the java doc using the colour; it is collect only the method documentation since we claim that was a good level of granularity. The class documentation was not enough, it gives only a global view of the functionalities.

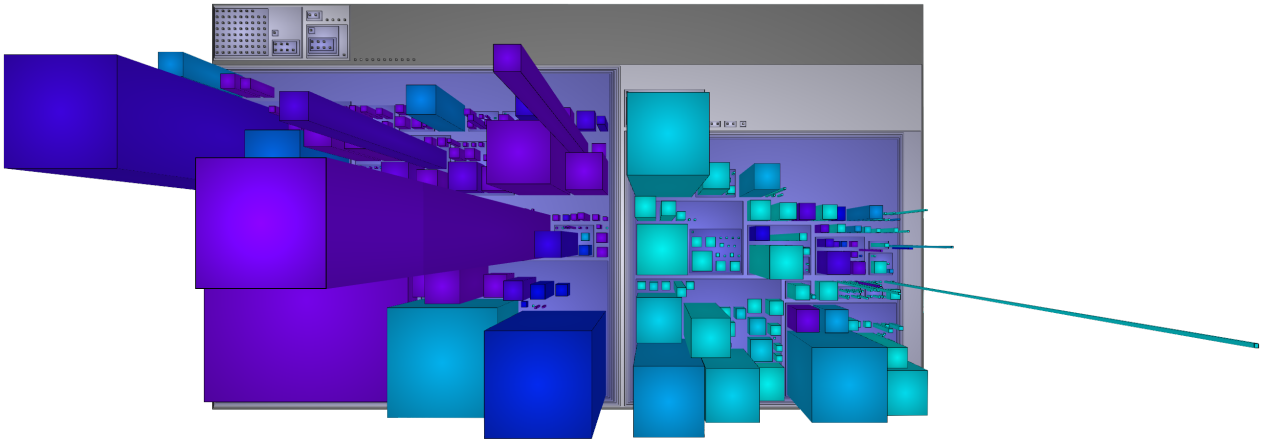


Figure 5. Mapping as Width:N of Fields, height: Number of Method,Colour: Percentage of documented methods

Figure **??** is example of a city in which the colour represent the percentage of documented methods.The project is the apache common-lang . Is very interesting to see that half of the project has a documentation coverage greaten that the 80% and in the other one is very rare. In reality this is a common case since a lot of project does't document the tests. To help the analyser to understand the documentation coverage we also provide a package base colouring system. In which the colour of each package is the average of the child component. In this case it look like figure **??**

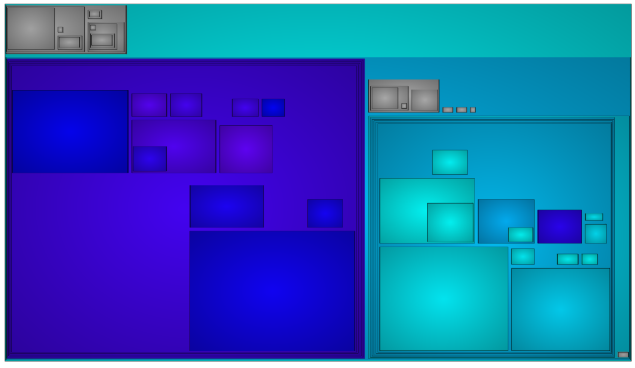


Figure 6. Mapping as Width:N of Fields, height: Number of Method,Colour: Percentage of documented methods

3.3.2 Stack Overflow Discussion

Stack Overflow is one of the most popular developer's forum. It contains a lot of code snippet and text related to the code. What we try to do using this visualisation method is to show to the user all the available discussion related to each method call. As you could observe the granularity is different respect the java doc metric. That allows to understand the complexity to read and understand the methods code not what the method itself does.We get he dataset update to august 2015. It contains ruffly 490000 discussions with more than 20000 different imports declaration and 100000 methods call.
In this stage we have all the repository code and all the discussion information (method call and import) from the stormed Dataset how we can merge to get a good approximation? We didn't use any type resolution system and this is a future improvement of the system. Now we are going to analyse the way that our tools match this information. Java imports are match easily by using a string matching. We ignore the asterisk at the end of the import if is present. We also analyse only the external import and not the import of the system.
The Method is a bit more complex since we use the name of the method call and the numbers of args. In this way we have a better matching.Remember that the discussion contains code that is not complete so we have to make an approximation to retrieve the data. The metrics that results is a sum over the total information that should be found. By interact on each building is possible to see the link to the stack overflow discussions.
The figure **??** is an example of the discussion found in respect to a building. The colour represent the number of discussion in absolute way. We see later what this means. As you can see there are two classes that as more discussion that the others . Of course classes that has more field are light blue coloured.

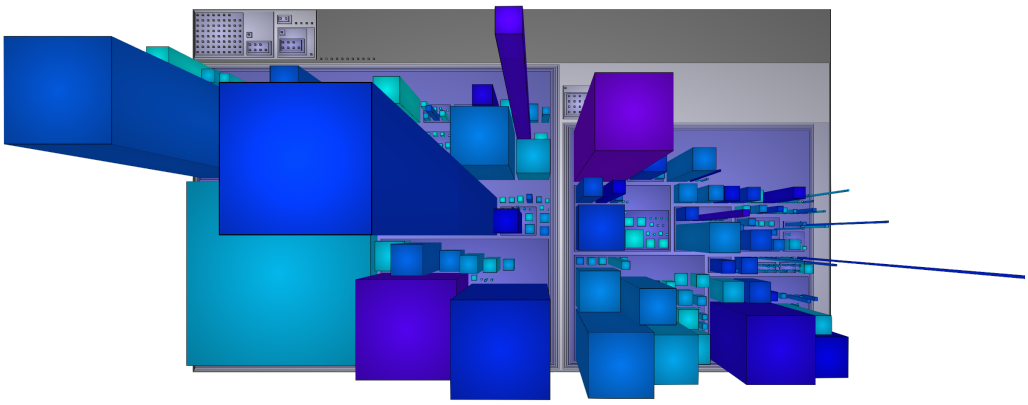


Figure 7. Mapping as Width:N of Fields, height: Number of Method,Colour: Absolute number of discussions

In figure **??** you can see the list of link for each method call or import declaration:

Discussion URL	
java.util.Arrays	http://stackoverflow.com/questions/25680943 http://stackoverflow.com/questions/21888201 http://stackoverflow.com/questions/2426380 http://stackoverflow.com/questions/17848025 http://stackoverflow.com/questions/20005140 http://stackoverflow.com/questions/27324495 http://stackoverflow.com/questions/26288671 http://stackoverflow.com/questions/23049314 http://stackoverflow.com/questions/27587522 http://stackoverflow.com/questions/23010150
length(0)	http://stackoverflow.com/questions/27223172 http://stackoverflow.com/questions/23820026 http://stackoverflow.com/questions/23530127 http://stackoverflow.com/questions/3699141 http://stackoverflow.com/questions/16570195 http://stackoverflow.com/questions/26089316 http://stackoverflow.com/questions/10541245 http://stackoverflow.com/questions/21764334 http://stackoverflow.com/questions/22419285 http://stackoverflow.com/questions/10564867
java.util.Iterator	http://stackoverflow.com/questions/3939447 http://stackoverflow.com/questions/29472128 http://stackoverflow.com/questions/20344851 http://stackoverflow.com/questions/23049314 http://stackoverflow.com/questions/4385003 http://stackoverflow.com/questions/13870322 http://stackoverflow.com/questions/25366639 http://stackoverflow.com/questions/17729475 http://stackoverflow.com/questions/10596744 http://stackoverflow.com/questions/27366643

Figure 8. List Of Discussions

3.4 Merge Code Related information with Corollary Information

The Code related information helps to identify the different component of the city and also helps to found design problem over the application. The corollary information, instead gives an idea about the information coverage.How we can mix together to get a global overview of the entire system? As you can see in the picture before we are using the information related to the code to give a size of the building, and we use the colour to represent the information coverage. In this way we improve the concept of locality, since a developer should remember a file not for the number of documented method but for the number of methods or fields.

3.4.1 Percentage and Absolute Numbers of informations

The corollary information could be computed in an absolute or in percentage. In the former way we count the number of information available and is possible to see which file contains more documentations. The percentage, instead, is computed over the total amount of information that it could be found. This metric is useful to spot which files have more documentation and which are not documented. We also decide to give 0% of documentation were we can't find anything because either are all fields or all the import are from local package.

3.4.2 Using Java Doc and Discussion together

To get a better understanding about the information coverage we have to mix up the documentation and the information related to code. We made an average of both since they correspond to two difference level of granularity. The java documentation refer to a method and the discussion refers to either import or method calls. We can show the result in both way: percentage and absolute.
In the former case, the developer can get a better understanding about the the percentage of the information available. This is useful to guess the effort require to understand a code. The latter, instead, is used to see were there are more concentration of information and where are not. It could be useful to identify package bad documented.

3.5 Colors

The colours is used to show another metrics. They goes from light blue to purple. They are very useful to give to the developer a quick impression about the system in the next section is used to map corollary information and for identify some code anomaly.

3.5.1 Corollary Information Colour meaning

The colour used to compute the corollary information has two different mining either if it compute as absolute or percentage. In the former case they show which is the most documented or which one has more discussion in purple and the minimum discussed if light blue.In the latter case we see the percentage over the methods. This give a local view about the percentage respect the file itself. So colours depend to the file not the whole project like in absolute view.

3.5.2 Code related Colour meaning

The colour used to compute the code related information it represent easily the number of methods,fields,class or interfaces in a files.It useful as we are going to see later, to check the code style, for example the number of classes or interfaces into a file or to get an idea where the majority of the methods, fields are concentrated. The scale is the same as above: purple mean full information and light blue no documentation available.

3.6 System Architecture

4 Evaluation

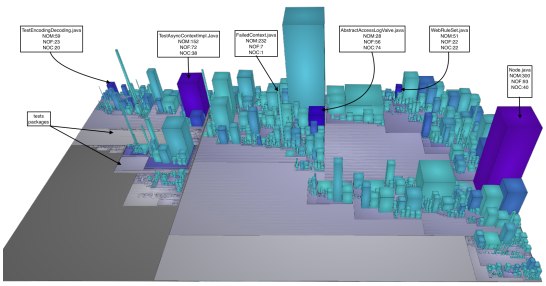
4.1 Introduction

In this section we are going to analyse two project by using our tool. The analysis of this project is split in two parts. The former part speak about the structure, we are looking for code identity harmony see **??**. The latter part we are going to analyse the information cover. For the former part we can't say that a particular design is wrong, we could only give a monitor to the developer to check some port and understand if it's correct. You can navigate and play with this two project on `http://rio.inf.usi.ch:51001/`.

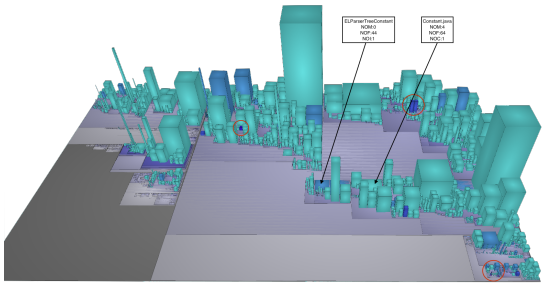
4.2 Tomcat

The Apache Tomcat software is an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies. The Apache Tomcat software is developed in an open and participatory environment. The Apache Tomcat project is intended to be a collaboration of the best-of-breed developers from around the world.

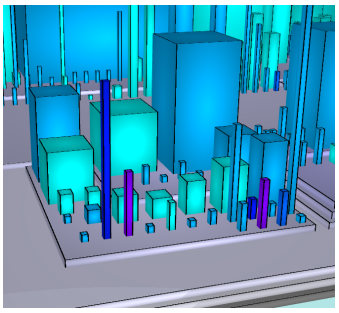
4.2.1 Code related analysis



(a) Tomcat: Classes



(b) Tomcat: Interfaces



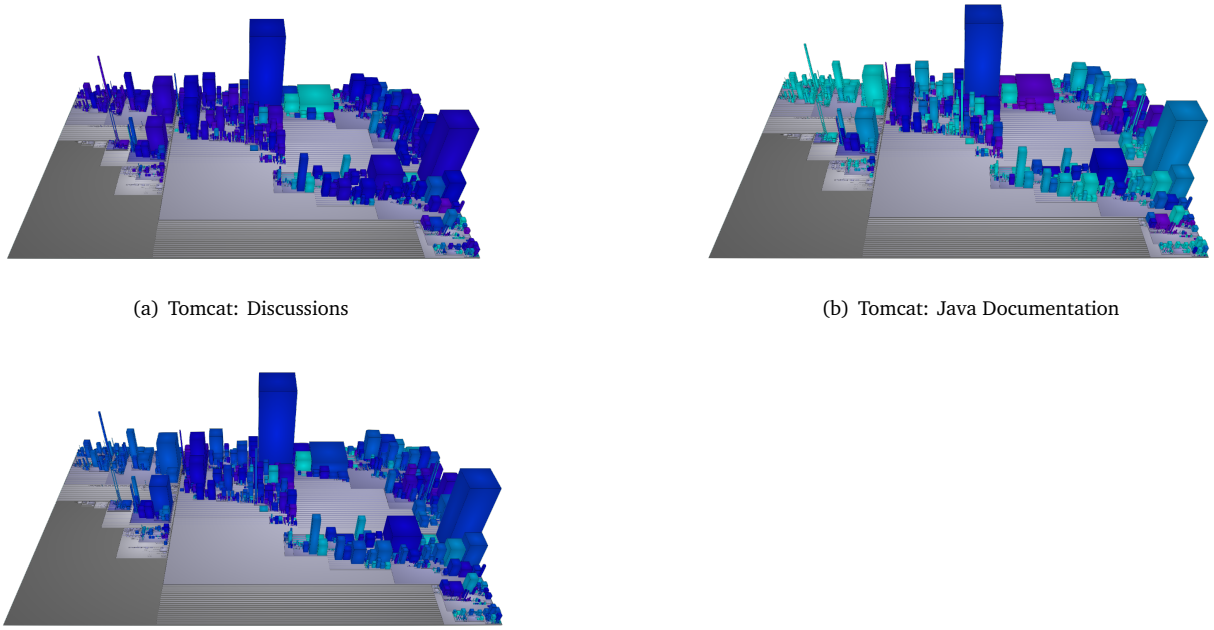
(c) Tomcat: Zoom Interfaces

Figure 9. Tomcat

Figure **??** depict tomcat's code related infrormations. The building represent the java files post on top of his package. The height of a building is the number of method and the width is the number of field. The colour represent in Figure **??** the number of interface and in **??** the number of classes.
Generally we have an equal distribution of class and interface for each file, there are only a few anomaly. Whit this software we can not understand if this anomaly are an error or not, we can only give to the developer a monitor to check class that looks strange.
In Figure **??** appear clear that there are some classes that has a height concentration of inner classes. Infect if you open the file you'll see a huge number of private static class. This is not a problem for the Java Code Conventions [**?**] infect there are not more then one public class and all the other inner class are inside the public class. The name and the characteristics are written on the image. As you can see, 2 of them are test classes the other 3 are not.The test class are fine.Tho other three classes could have a high level of coupling that is an hint to check the design. Regarding the interface, we have some files that contains more then once. In this case we have not big number of interface so it could be a design chose and not a problem.
Now we take a look on the method and fields of a class. As we can aspect the class Node.java has a lot of methods

but at the same time have a huge number of classes as sow before. It is a good candidate for a God Class. As well as Node.java also StandardContex.java has the potential to be a god class either, since he has only thee classes and a huge number of methods and fields. In both class we can incur in a high level of coupling.
There are a few buildings that looks like a brain class. We don't care about test class since are, by definition a list of methods. The fist one is FailledContext.java, that as a huge amount of methods and no too much fields. As you can see from image there are other classes of this type.
The Data class are not too much, once is call, Constant.java and there are others on the figure **??**.

4.2.2 Corollary Information analysis



(a) Tomcat: Discussions

(b) Tomcat: Java Documentation

(c) Tomcat: Discussion and Java Doc

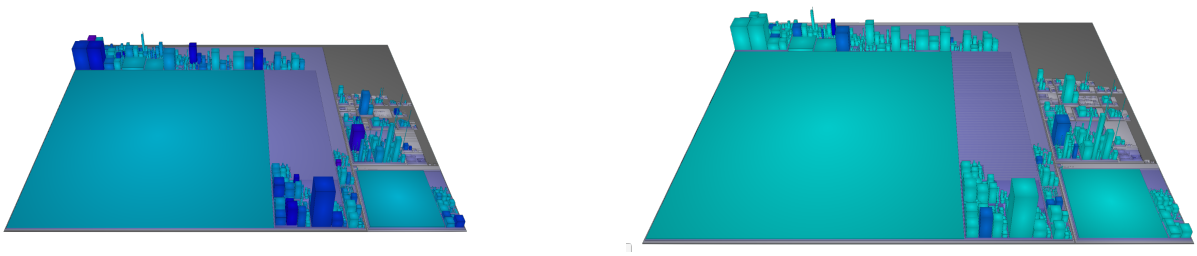
Figure 10. Tomcat Corollary Informations

Figure **??** depict tomcat's code corollary informations.The building represent the java files post on top of his package. The height of a building is the number of method and the width is the number of field. The colour represent in Figure **??** the number of discussion over methods, in **??** the number of java documentation over methods and in **??** the information coverage.
Let's start analysed the java documentation: is possible to see there are classes completely documented and others that has not documentation at all. The tests are completely not documented and some of the class that as more methods has a lower percentage of documentation, this is bad.
Instead the discussion coverage is pretty good. The colour of the city in average is dark blue and there are a lot of building purple.
Now that we have the result of both metrics we can merge it together an we have the **??**. Thanks to the discussion found online and the documentation, in general, the source code should not be too hard to understanding it.

4.3 JGit

Jgit is an implementation of the Git version control system for java.We analyse the system in the same way as Tomcat.

4.3.1 Code related analysis



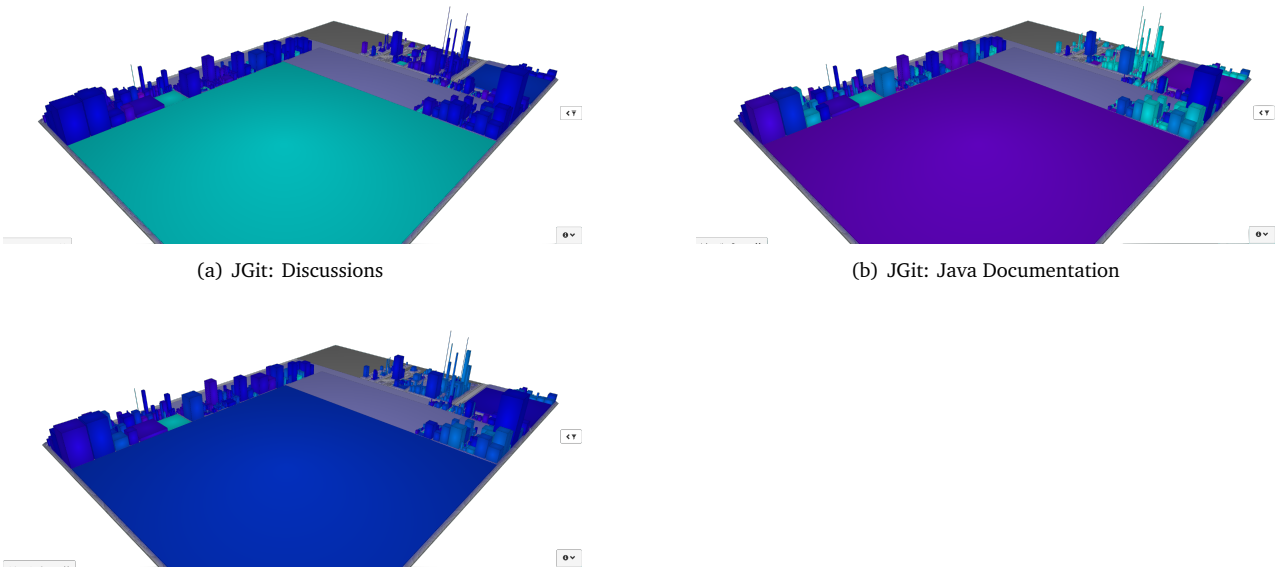
(a) JGit: Classes

(b) Jgit: Java Interfaces

Figure 11. Jgit Code Related Informations

Figure **??** depict jgit's code related informations.The building represent the java files post on top of his package. The height of a building is the number of method and the width is the number of field. The colour represent in Figure **??** the number of classes, in **??** the number of java interfaces.
The class number view shows that there are some files that have more then one class. Is important to note that the maximum amount of class is 10. Also the interface distribution appear to be well spread, there are only a few occurrence of multiple interface on the same class. Just remind that is not a crime to have more interface or class in the same file. The problem is that if you have too much you could have a small coupling degree that is bad!
Regarding the methods and fields. Differently as aspected the file that has more class has not more field and methods.We can see a god class call PackWriter.java that has 48 fields and 121 methods. There are also 2 big data class:CLIText and JGitText. At last but not least there is a brain class call RepositoryState.java that has 90 methods.

4.3.2 Corollary Information analysis



(a) JGit: Discussions

(b) JGit: Java Documentation

(c) JGit: Discussion and Java Doc

Figure 12. JGit Corollary Informations

5 Conclusion

References

[1] A. Bacchelli, A. Cleve, M. Lanza, and A. Mocci. Extracting structured data from natural language documents with island parsing. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, pages 476–479, Nov 2011.

[2] R. P. Gabriel. *Patterns of software*, volume 62. Oxford University Press New York, 1996.

[3] Oracle. Corollary.

[4] L. Ponzanelli, A. Mocci, and M. Lanza. Stormed: Stack overflow ready made data. In *Proceedings of MSR 2015 (12th Working Conference on Mining Software Repositories)*, page to appear. ACM Press, 2015.

[5] R. Wettel and M. Lanza. Program comprehension through software habitability. In *15th IEEE International Conference on Program Comprehension (ICPC '07)*, pages 231–240, June 2007.

[6] R. Wettel and M. Lanza. Visualizing software systems as cities. In *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 92–99, June 2007.

[7] Wikipedia. Corollary.