Towards the Visualization of Software Systems as 3D Forests: the CodeTrees Environment

Ugo Erra and Giuseppe Scanniello
Dipartimento di Matematica e Informatica
Università della Basilicata
Potenza, Italy
{ugo.erra, giuseppe.scanniello}@unibas.it

ABSTRACT

We present an approach based on a forest metaphor to ease the comprehension of object oriented software systems. Software systems are represented as forests of trees that users can navigate and interact with. We also describe here the mapping of the information of the source code in meaningful ways to take advantages of familiar concepts such as agglomerates of trees (or sub-forest), trunk, branches, leaves, and color of the leaves. The approach has been implemented in a prototype of a 3D environment, namely CodeTrees. To assess the validity of the approach and environment, we have also conducted a preliminary empirical evaluation on three open source software systems implemented in the programming languages Java and C++.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: Graphical User Interface; I.3.6 [**Computer Graphics**]: Methodology and Techniques—*Interaction Techniques*

Keywords

Software visualization, Software maintenance, forest metaphor

1. INTRODUCTION

Software maintenance is essential in the evolution of software systems and represents one of the most expensive, time consuming, and challenging phase of the development process. Maintenance starts after the delivery of the first version of the system and lasts much longer than the initial development process [1]. As shown in the survey by Erlikh [2], the costs needed to perform maintenance operations range from 85% to 90% of the total cost of a software project.

During the maintenance phase, a software system is continuously changed and enhanced because of the execution of maintenance operations that are carried out for several reasons (e.g., to correct faults or to improve quality requirements) [3]. Whatever is the maintenance operation, a maintainer has to comprehend source code implemented by others [4]. Therefore, it is easy to imagine that the greater part of the cost and effort for accomplishing maintenance tasks is due to the comprehension of the source code. There are several reasons that make the comprehension even more costly and complex, they range from the size of the software to its overall quality.

Software visualization [5], [6] has been being extensively and successfully explored and used in the software maintenance and program comprehension, in particular. Researchers have proposed metaphors and supporting tools based on 2D and 3D techniques [7], [8], [9]. The proposed metaphors and tools often fail to show significant information of a software system to improve its comprehension (e.g., public attributes and methods).

In this paper[1], we propose a 3D visualization metaphor for depicting object oriented software system and a supporting tool, named CodeTrees. In particular, a given software software is visualized as a forest of trees that users (also maintainers in the following) can navigate and interact with. Visual properties of trees (e.g., trunks, branches, and leaves) are mapped according to well defined rules with the metrics extracted from source code.

The metaphor is new from the point of view of software visualization and, compared to previous metaphors, provides a fine-grained representation of the entire software system and a large-grained representation of the classes and the packages they reside in. Therefore, we display classes as trees and packages as agglomerates of trees (from here on, sub-forests or simply forests). The shape (trunk and foliage) of each tree is characterized by source code metrics of the class it represents.

To validate the metaphor and CodeTrees, we have also conducted a preliminary case study on three well known open source software systems implemented in Java and C++: JEdit, ArtOfIllusion, and FileZilla.

The main contributions of the paper can be summarized as follows:

- A visualization metaphor based on forests of trees;
- A prototype of software tool (i.e., CodeTrees) implementing the metaphor;
- A preliminary case study on small/medium open source software systems implemented in Java and C++.

[1]Please read the paper on-screen or as a color-printed paper version, we make extensive use of color pictures.

SAC '12 March 25-29, 2012, Riva del Garda (Trento), Italy

The remainder of the paper is organized as follows: in Section 2, we review previous software visualization metaphors based on natural environments (e.g., the solar system) or real objects (e.g., a jigsaw puzzle). In Section 3, we describe the proposed metaphor. In Section 4, we present the preliminary results of the conducted empirical evaluation. Finally, Section 5 concludes and discusses possible future directions for our research.

2. RELATED WORK

In the literature there are some metaphors to visually represent software systems as natural environments. Among these, the city metaphor is one of the most explored (e.g., [10], [11], [12], [13]). For example, Wettel and Lanza [14] propose a city metaphor for the the comprehension of object oriented software systems. Similar to our proposal, the metaphor proposes a large and low -scale understanding of software. Classes are represented as buildings and packages as districts. The metaphor is implemented in the CodeCity tool and to improve the realistic aspect of the city, authors focused on the design of urban domain. Our metaphor is different because the fine-grained representation of classes takes into account a larger number of metrics (e.g., number of public methods). However, we plan users' study to understand whether the use of more information results in a deeper understanding of the system.

In a recent paper, the same authors [15] present a controlled experiment with professionals to assess the validity of their metaphor and supporting tool. The results show that the CodeCity tool leads to a statistically significant improvement in terms of task correctness. The results also indicate that the use of the tool statistically decreases the task completion time.

Graham *et al.* [16] propose a solar system metaphor where suns represent a package, planets are classes, and orbits represent the inheritance level within the package. Such metaphor is used as a means to analyze either static or evolving code to perceive in real time suspected areas of risk within the code base.

Martínez *et al.* [17] suggest a metaphor based on landscape, whose main objective is to visualize the integrated representation of software development processes. The metaphor is conceived to describe several aspects of development process. Differently from our proposal, the metaphor does not provide means to visualize source code.

Ghandar *et al.* propose a jigsaw puzzle metaphor in [18], where each component of a system is modeled as a piece of a jigsaw puzzle. The most remarkable characteristic of the metaphor concerns the visualization of the system complexity that is represented as a pattern on the surface of the piece. The metaphor does not provide a view at class granularity level. This is one of the most remarkable differences with respect to our proposal.

A botanical tree metaphor is proposed in [19]. Differently from us, the authors suggest forests of trees for the visualization of huge hierarchical structures and apply the proposed metaphor to the visualization of directory structures. Directories, files, and their relations are visualized using trees. The approach is basically a natural visual metaphor for information hierarchically structured. Several are the differences with respect to our proposal. For example, we propose here a metaphor for the visualization of software systems, while in [19] the authors propose trees for depicting hierarchical structures in file systems. Further, we use trees as fine-grained representation of classes.

Several are the differences with respect to the approaches discussed above. The most remarkable one is that our proposal offers a proper representation for methods, attributes, and source code comment. A possible drawback that affects our metaphor is that it could result complex in case the maintainer is not properly trained. This issue is directly connected to the considerable amount of information our metaphor is able to visually summarize.

3. THE METAPHOR

We propose a 3D visualization metaphor that depicts object oriented software systems as forests of trees. Maintainers can freely navigate and interact with the forests to improve the comprehension of the systems. The reasons for defining this metaphor can be summarized as follows:

- A tree has a familiar shape. We use trees to visually summarize complex information in a natural way.
- A tree is a complex structure composed of unmistakable elements such as trunk, branches, leaves, and color. Several aspects of a software system can be represented through the visual properties of a tree.
- The class proliferation problem is a major cause of failure in object oriented development. A forest of trees provides developers with a large-scale understanding of the design and the proliferation of classes avoiding exponential explosion in the number of classes.

Our main goal is a large-scale understanding of a software system because the overall result is a visualization of all the contained classes. The metaphor also provides a fine-grained representation of the components of a software in terms of individual classes. This is because a visualization of each class is possible zooming on the corresponding tree.

3.1 The Rendering of the Trees

The model upon which we based the creation and the rendering of the trees is based on the Weber and Penn approach [20]. This approach uses an intuitive model to design the geometrical structure of trees. It handles properties/parameters to modify the shape of a tree. There are general parameters to control: the height of the tree, the width of the trunk, the level of recursion of the branches, and so on. Further, there are also parameters that control more specific aspects of a tree (e.g., leaves orientation).

The approach needs no knowledge of botany and complex mathematical principles. For example, each branch may have similarity with its parent and inherit attributes from it. Due to this concern, all the branches are influenced by the primary branches that mainly depend from the tree height. In our case, this is not an actual drawback because a primary aim of the Weber and Penn approach is to get a plausible result with a few parameters. However, we consider a subset of the parameters needed to render a tree, thus affecting its shape (trunk and foliage) according to the main characteristics of the class the tree represents. The interested reader can read the paper by Weber and Penn [20] to get a deep description of the algorithm and to knw how each parameter affects the creation and the shape of a tree.

3.2 Mapping Between Tree Properties and Source Metrics

We identified the following set of visual properties of trees that can give information about the system to analyze (or to comprehend in case of maintenance tasks): height, branch number, branch direction, leaf number, leave color, leaves size, and base size (i.e., the part of trunk whithout branches).

The considered metrics used to influence the appearance of a tree are: lines of code (LOCs), lines of comment (CLOCs), number of attributes (NOAs), number of public methods (NEMs), number of private methods (NOMs), and the total number of methods (NEOMs). Although there many other design metrics available in the literature [21], we considered only the most widely known (e.g., [21]). The rationale for this choice relies on the fact that more complex and badly known metrics may complicate the metaphor. However, the use of different metrics is subject of future work.

The mappings between the properties of a tree and the selected metrics were based on the experience we gained in the program comprehension, software visualization, and development. We tried as much as possible to find intuitive mappings to ease the comprehension of a class.

Similarly to [14], we mapped the height of a tree with the LOCs metric to denote the size of a class (without take into account CLOCs). The number of branches that sprout from the trunk corresponds to NEOMs. A tree with few branches represents a class with few methods. To highlight the number of public methods we use branch orientation. In case the value NEMs/NEOMs is close to one, the class has many public methods and its tree has branches pointing out. Conversely, if NEMs/NEOMs tends to zero, the class has many private methods and its tree has branches oriented parallel to ground. We mapped the total number of leaves with NOAs and the size of the leaves results from 1/NOAs. In such way, a class with few attributes is represented as a tree with large leaves, so resulting more visible. While, a dense tree will represent a class with many attributes. We map the color of leaves as the ratio between the LOCs and the CLOCs. The color ranges from the green to brown. Thus, a class with lines of comments greater than lines of code is represented as a tree with a green foliage, brown otherwise. Finally, to visualize the amount of private methods we used the ratio NOMs/NEOMs that is visually represented as the base size of a tree. Then, if the value NOMs/NEOMs is close to one, the class has many private methods and the tree has a large base size. Conversely, if NOMs/NEOMs tends to zero, the class has many public methods and its tree has a small base size.

In Table 1, we summarize the mapping between metrics and visual properties. It is worth mentioning that some visual properties highlight the local characteristic of a given class (e.g., NOMs/NEOMs), while others make sense only in case trees are analyzed together (e.g. LOCs).

To illustrate how the mapping changes the appearance of a tree, we illustrate some sample classes. Figure 1 shows a tree of a possible class where the number of line of codes is large (the foliage is stretched out), there are huge number of line of comments, and the total number of methods (both public and private) can be considered large. We show in Figure 2 a tree of a class with: a small lines of codes (see the foliage), a number of lines of comments much smaller than the lines of source codes, a small number of methods, and a few attributes. Figure 3 depicts a tree of a class where the number of lines of code is high (see the foliage), the number of lines of comment is high, there are a considerable number of methods both public and private, but public methods are prevalent. Figure 4 shows a class with a high number of public methods and few attributes. This mostly happens when we are dealing with façade patters. Finally, Figure 5 depicts a tree of a class with a huge number of methods. Many of these methods are public. The class also contains a large number of lines of comment.

Figure 1: jEdit 4.3::JarBundler. The tree represents a class with: a high number of lines of code, a huge number of lines of comment, a high number of methods (both public and private).

Figure 2: jEdit 4.3::BrowserIORequest. The tree represents a class with: a low number of lines of code, a small number of lines of comment, a small number of total methods, and few attributes.

The metrics extracted from the sample classes are shown in Table 2. Although the variability of the values for each metric, the tree of of each class maintains a good visual plausibility and naturalism.

3.3 A Forest of Trees

We represent a package as a sub-forest of the whole forest of trees. All the trees which represent classes inside a package are placed in a land square whose size is function of number of trees. To provide visibility to the smaller trees, we place theme in front of the highest trees using a spiral pattern. In this way, highest trees will be placed in the cen-

Figure 3: FileZilla 3.0::sshbn. The tree represents a class with: a high number of lines of code, a high number of lines of comment, a number of public methods greater that private methods.

Figure 4: ArtOfIllusion 2.4.1::CustomDistortionTrack. The tree represents a class with: a large number of public methods, few private methods, and few attributes.

Metrics	Visual Properties
LOCs	Tall
NEOMs	Branches
NOAs	Number of leaves
NEMs/NEOMs	Branches orientation
1/NOAs	Size of leaves
LOCs/CLOCs	Color of the leaves (from green to brown)
NOMs/NEOMs	Base size

LOCs = Lines of Code
CLOCs = Lines of Comment
NOAs = Number of Attributes
NEOMs = Total Number of Methods
NEMs = Number of Public Methods
NOMs = Nmber of Private Methods

Table 1: Metrics and visual properties mapping.

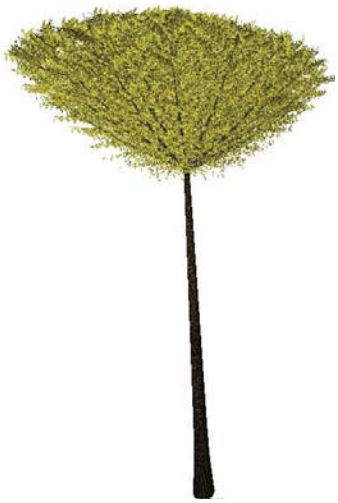

Figure 5: jEdit 4.2::jEdit. The tree represents a class with: a high number of public methods greater than private methods, a large number of lines of comment, a large number of attributes, and few private methods.

	LOCs	CLOCs	NOAs	NEMs	NOMs	NEOMs
JarBundler	713	465	35	50	22	72
BrowserIORequest	248	71	11	2	4	6
sshbn	732	246	75	3	11	14
CustomDistortionTrack	362	51	0	34	1	35
jEdit	2488	1043	28	94	24	118

Table 2: Metrics of the sample classes.

ter of land square and they will be always visible regardless the user's point of view. It is also important to point out that we have considered the packages structure in terms of contained classes. The use of relationships between classes (e.g., inheritances and method invocations) in the visualization of a forest is not considered. This could be subject of future work.

An example of forest as depicted within CodeTrees is shown in Figure 6. The forest represents a software system composed of 7 packages. Some trees have no leaves, thus indicating that no attributes are present. Some trees have the branches that point out. When trees do not have leaves and the branches point out it is possible that we are in the case of a façade pattern. The leaves of some trees are brown, while others are green. In the first case, we can deduce that these classes are not properly commented, while the source code is commented in the latter case. The forest also contains a few trees whose branches are parallel to the ground. These trees have brown leaves, thus suggesting that the classes have a large number of private methods and are scarcely commented. Therefore, we can deduce that the greater part of the functionality implemented by these classes is only accessible through a few public methods.

3.4 Navigating within a Forest

The forest is visualized in real-time and the maintainer can navigate around the forest using a free-fly 3D camera. The maintainer can move inside the forest without limited movement capabilities and then it is possible also to pass through trees. The maintainer can then enlarges the tree

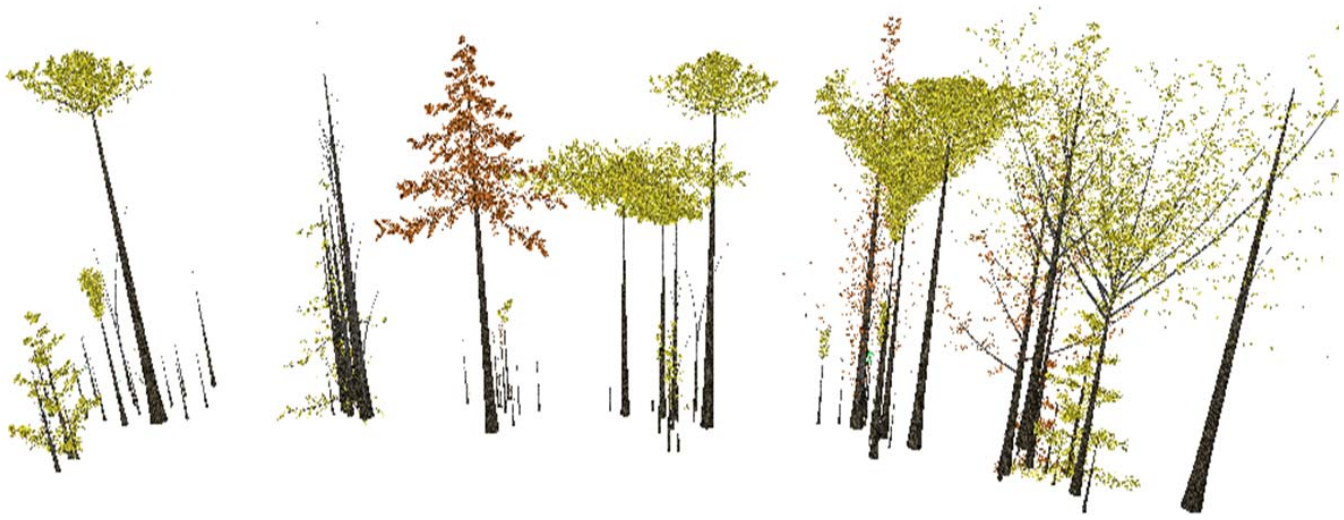

Figure 6: A Sample Forest composed of 7 packages.

of interest. The name of the class is also associated to the corresponding tree and visualized if required.

3.5 CodeTrees Implementation

The metaphor has been implemented in a prototype of a supporting system, named CodeTrees[2]. The prototype is composed of three main software components. The former extracts the metrics for a given software system and then produces an XML file with all the information needed for the visualization. The design rationale for this choice relies on the fact that the extraction of the measures can be performed once for all. Further, this makes independent the extraction of the measures from the rendering of a forest and its trees. This component is implemented in Java.

The second component is aimed at graphically maps the tree parameters defined in the algorithm proposed in [20] and the extracted metrics. The component is implemented in Java and produces an XML file that is then used to make the rendering of a forest by our 3D engine.

To implement the 3D engine, we used the OpenTree library. This library provides 3D tree generation for real time applications such as games and visualization software. OpenTree is a cross-platform, engine-independent library written in C++. It implements the tree generation algorithm described in [20]. The library uses an array of vertex to generate mesh data necessary to render trees. Vertex can be used by any graphics library. We used in our implementation the OpenGL [22] graphics library.

4. EVALUATION

To validate CodeTrees and the underlying metaphor, we conducted a preliminarly case study on three object oriented software systems: JEdit, Art of Illusion, and FileZilla. The former two systems were implemented in Java, while the third in C++. We selected these systems: *(i)* to verify whether the validity of our proposal is affected by the programming language; *(ii)* because they have been widely used in the past to assess the validity of tools for supporting maintenance tasks. Another motivation for selecting these systems relies on the fact that they are very different in terms of implemented functionality:

- **JEdit** is a programmer's text editor with an extensible plug-in architectures;
- **Art of Illusion** is a 3D modeling and rendering software system;
- **FileZilla** is a cross platform client for FTP, FTPS, and SFTP.

	JEdit	Art of Illusion	FileZilla
# Classes	532	453	153
# Packages	42	26	0
LOCs	102680	103404	71510
CLOCs	49743	18367	9976
NEMs	4920	5110	884
NOMs	1001	1010	332
NOAs	2994	2353	2299
Language	JAVA	JAVA	C++
Software Version	4.3	2.4.1	3.0

Table 3: Metrics of the three software systems.

Some statistics of these systems are reported in Table 3. In particular, the table shows the numbers of classes and packages for JEdit, Art of Illusion, and FileZilla. For each system, the number of line of code and comment is shown as well. The latter rows report the number of methods (total, public, and private), the number of attributes, and the programming language used to implement each system, and the analyzed version.

Figure 7 shows the forests of the three software systems. On the top there is the JEdit forest, while in the middle the one for Art of Illusion. The forest of FileZilla is on the bottom. In the following, we discuss the findings achieved by applying the metaphor and the tool only from a quantitative perspective. In fact, the metaphor is based on the properties of trees and the metrics for each class in the forest. A qualitative analysis of the results is subject of future work.

[2]A sample video of the application of our metaphor and the prototype is available at `http://www.unibas.it/utenti/erra/sac2012.wmv`

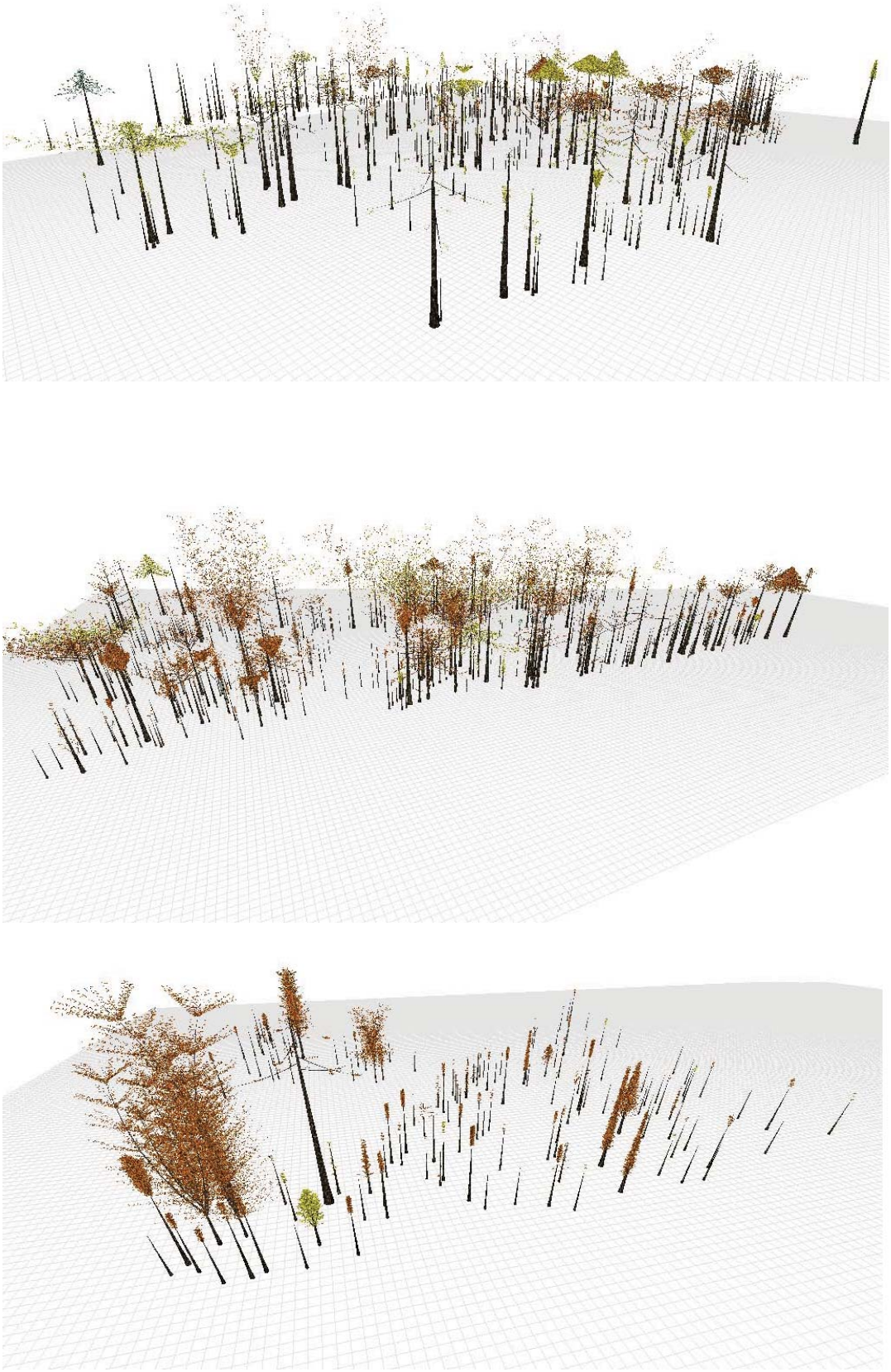

Figure 7: From up to bottom the forest of JEdit, Art of Illusion, and FileZilla.

The big picture of the three forests shows that among the tree systems FileZilla has a smaller number of lines of comments. Further, this system contains the lowest number of classes and the majority of these classes contain a low number of lines of code. Further, the trees are sparse (i.e., sub-forests are not present) since the classes are not organized in packages being this system implemented in C++.

With regards to JEdit, we can observe that there are some classes that contained a high number of lines of comments. Among these classes, there are some classes with a larger number of public methods and with a few attributes. On the other hand, there are few classes with a large number of attributes and a few number of public methods. In the first case, the classes provide services to other classes and to make easy how to use these services the developers commented the code. In the latter case, the trees represent property classes.

The forest of Art of Illusion contain a big class with a large number of lines of code, attributes, and methods. A few lines of comment are present. Each sub-forest contains a tree without leaves and with branches that point out.

4.1 Further Results

The use of CodeTrees on the systems JEdit, Art of Illusion, and FileZilla leads also to the following considerations:

- **Scalability**. CodeTrees scales up well in terms of the size of the software systems to be visualized. However, for very large software systems (e.g., Eclipse 3.0) the tool slowed down, thus affecting the interactivity and the navigability of the forest. Several directions for our future work have been planned to improve scalability. For example, we will implement a GPU (Graphics Processing Unit) based version of CodeTrees.

- **Navigation and Interactivity**. Similarly to many 3D environments [14], CodeTrees allows the maintainer to move back or forward, hover left or right, orbit around the city, and change altitude. CodeTrees also enables the visualization of the name of each class through a label associated to the corresponding tree. A further feature to make the forest more realistic concerns the ground. In particular, the type (e.g., grass or stone) and the color (e.g., green or brown) of the ground can be changed if needed.

- **Completeness**. The defined metaphor and implemented 3D environment provide a fair amount of information for an overview of the system. They also offer a proper representation for methods, attributes, and comments. These characteristics make our approach different from the ones available in the literature (e.g., [14], [10]).

5. CONCLUSION AND FUTURE WORK

In this paper, we presented a 3D software visualization approach based on a forest of trees. The classes are represented as trees and the packages as sub-forests. The approach provides a general overview of the software giving a maintainer the possibility of carrying out a large-scale reasoning on it. Further, the classes within packages are represented as trees. The visual properties of each tree (e.g., trunks, branches, and leaves) are used to represents metrics of the class it represents. In this way, the maintainer has a large-grained representation of each class.

The metaphor has been implemented in prototype of a supporting tool, named CodeTrees. It provides features to navigate in the forest as a free fly 3D virtual camera. CodeTrees also implements zoom features and visualizes the names of the classes to promote the comprehension at fine-grained level. Although the metaphor is for any object oriented programming language, the current implementation of the prototype supports Java and C++. The prototype can be however easily extended to visualize software systems implemented with different programming languages. It is only needed to implement a tool for extracting the considered metrics from code implemented with those programming languages.

In order to assess the validity of the approach and CodeTrees, we have conducted a preliminary empirical investigation as a case study. We used medium open source software systems. Two of these systems were implemented in Java and one in C++. The results seem encouraging, but due to the preliminary nature of our investigation caution is needed and replications are required to increase our awareness on the achieved findings.

There are several future directions for our work. We plan to conduct users' studies to evaluate the effectiveness of our proposal in the execution of maintenance or software comprehension tasks. We will use recent guidelines [23] for such a kind of studies. We also plan to improve the tool support in terms of interactions between the maintainers and the virtual natural environment using the idea of Visual Information Seeking Mantra "Overview, zoom and filter, details-on-demand" [24]. We are also going to investigate our metaphor in the visualization of evolving software systems as growing forests. Finally, future work will be devoted to increase the realism of the forest and to assess whether users perform maintenance tasks in a more efficient fashion in case of more realistic forests. The assessment of efficiency will represent another key factor in that investigation.

Acknowledgment

We would like to thank the Manuela Viggiani and Nicola Capece, who developed some of the software modules of the prototype implementing the metaphor presented here.

6. REFERENCES

[1] M. V. Zelkowitz, A. C. Shaw, and J. D. Gannon. Principles of software engineering and design. 1979.

[2] Len Erlikh. Leveraging legacy system dollars for e-business. *IT Professional*, 2:17–23, May 2000.

[3] M. M. Lehman. Program evolution. 19(1):19–36, 1984.

[4] Anneliese Von Mayrhauser. Program comprehension during software maintenance and evolution. *IEEE Computer*, 28:44–55, 1995.

[5] John Stasko, John Domingue, Blaine A. Price, and H Marc. Brown: software visualization: programming as a multimedia experience. 1998.

[6] Stephan Diehl, editor. *Software Visualization, International Seminar Dagstuhl Castle, Germany, May 20-25, 2001, Revised Lectures*, volume 2269 of *Lecture Notes in Computer Science*. Springer, 2002.

[7] Richard Wettel and Michele Lanza. Program comprehension through software habitability. In *ICPC*, pages 231–240, 2007.

[8] Stéphane Ducasse and Michele Lanza. The class blueprint: Visually supporting the understanding of classes. *IEEE Trans. Software Eng.*, 31(1):75–90, 2005.

[9] Andrian Marcus, Louis Feng, and Jonathan I. Maletic. Comprehension of software analysis data using 3d visualization. In *IWPC*, pages 105–114, 2003.

[10] C. Knight and M. Munro. Virtual but visible software. In *Information Visualization, 2000. Proceedings. IEEE International Conference on*, pages 198 –205, 2000.

[11] Stuart M. Charters, Claire Knight, Nigel Thomas, and Malcolm Munro. Visualisation for informed decision making; from code to components. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, SEKE '02, pages 765–772, New York, NY, USA, 2002. ACM.

[12] Thomas Panas, Rebecca Berrigan, and John Grundy. A 3d metaphor for software production visualization. In *Proceedings of the Seventh International Conference on Information Visualization*, pages 314–, Washington, DC, USA, 2003. IEEE Computer Society.

[13] Blazej Kot, Burkhard Wuensche, John Grundy, and John Hosking. Information visualisation utilising 3d computer game engines case study: a source code comprehension tool. In *Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: making CHI natural*, CHINZ '05, pages 53–60, New York, NY, USA, 2005. ACM.

[14] Richard Wettel and Michele Lanza. Codecity: 3d visualization of large-scale software. In *Companion of the 30th international conference on Software engineering*, ICSE Companion '08, pages 921–922, New York, NY, USA, 2008. ACM.

[15] Richard Wettel, Michele Lanza, and Romain Robbes. Software systems as cities: A controlled experiment. In *Proceedings of ICSE 2011 (33rd International Conference on Software Engineeering)*, page to be published, 2011.

[16] Hamish Graham, Hong Yul Yang, and Rebecca Berrigan. A solar system metaphor for 3d visualisation of object oriented software metrics. In *Proceedings of the 2004 Australasian symposium on Information Visualisation - Volume 35*, APVis '04, pages 53–59, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[17] Amaia Aguirregoitia Martínez, J. Javier Dolado Cosín, and Concepción Presedo García. A landscape metaphor for visualization of software projects. In *Proceedings of the 4th ACM symposium on Software visualization*, SoftVis '08, pages 197–198, New York, NY, USA, 2008. ACM.

[18] Adam Ghandar, A. S. M. Sajeev, and Xiaodi Huang. Pattern puzzle: a metaphor for visualizing software complexity measures. In *Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation - Volume 60*, APVis '06, pages 221–224, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[19] Ernst Kleiberg, Huub van de Wetering, and Jarke J. Van Wijk. Botanical visualization of huge hierarchies. In *Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*, pages 87–, Washington, DC, USA, 2001. IEEE Computer Society.

[20] Jason Weber and Joseph Penn. Creation and rendering of realistic trees. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 119–128, New York, NY, USA, 1995. ACM.

[21] Michele Lanza and Radu Marinescu. *Object-Oriented Metrics in Practice. Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-oriented Systems*. Springer Verlag, 2010.

[22] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1999.

[23] Mariam Sensalire, Patrick Ogao, and Alexandru Telea. Evaluation of software visualization tools: Lessons learned. In *VISSOFT*, pages 19–26, 2009.

[24] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, page 336, Washington, DC, USA, 1996. IEEE Computer Society.