A 3D Metaphor for Software Production Visualization

Thomas Panas
Software Technology Group
Växjö University, Växjö, Sweden
Thomas.Panas@msi.vxu.se

Rebecca Berrigan
Peace Software
Auckland, New Zealand
rebecca.berrigan@peace.com

John Grundy
Department of Computer Science
Auckland University, Auckland, New Zealand
john-g@cs.auckland.ac.nz

ABSTRACT
Software development is difficult because software is complex, the software production process is complex and understanding of software systems is a challenge. In this paper we propose a 3D visual approach to depict software production cost related program information to support software maintenance. The information helps us to reduce software maintenance costs, to plan the use of personnel wisely, to appoint experts efficiently and to detect system problems early.

KEY WORDS
Program Visualization, Information Visualization, Program Understanding, Reverse Engineering, Software Maintenance.

1 Introduction

Software engineers who are to maintain, extend or reuse a software system must understand it in the first place. Obtaining this understanding of an industrial size system is often a time consuming process since most legacy systems are usually sparsely (or inadequately) documented. Estimates of the proportion of resources and time devoted to maintenance range from 50% to 75% [1, 2]. The greatest part of the software maintenance process, in turn, is devoted to understanding the system being maintained. Fjeldstad and Hamlen report that 47% and 62% of time spent on actual enhancement and correction tasks, respectively, are devoted to comprehension activities [3].

Due to the above reasons, reverse engineering of legacy systems has become a vital matter in today's software industry. However, increasing program understandability is not a simple task due to the complexity of software systems. It is assumed that software complexity is related to the commonly accepted empirically derived features of software like code size, coupling, depth of inheritance, McCabe complexity related to logical paths and data flow, amongst others. It is also likely that the *cost* of related aspects in a system such as defect levels or the effort required to design, develop, test and maintain a software product gather more attention in a business context focused on software production.

Therefore, to simplify software complexity and increase program understandability while correlating with cost profiles across the product, reverse engineered programs must be effectively analyzed and visualized. Since developers think and perceive information differently, a tool must support user specific metaphors and views. Currently, metaphors and views within reverse engineering are restricted to depict program specific data only, i.e. they show merely information retrieved from code or data analysis. However, it is important and necessary to establish cost related analyses and views, which give answers to questions like: Which components are never executed? Which components are often changed? On which components do my developers currently work? Is the architectural structure of my system obsolete and hence needs to be refactored? Therefore, within this paper we present an idea that helps system maintainers and managers to e.g. decide, which components in a system are superfluous, which ones cause high cost due to frequent changes and whether the entire system needs to be revised.

In Section 2 we give a brief introduction to program visualization for software maintenance. In Section 3 we present an example, showing a 3D city metaphor for program visualization. On top of this picture, we present a code production focussed metaphor in Section 4. Related work is summarised in Section 5. A discussion and future work is presented in Section 6. Finally, the paper is concluded in Section 7.

Proceedings of the Seventh International Conference on Information Visualization (IV'03)
1093-9547/03 $17.00 © 2003 IEEE

2 Program Visualization

Visualization is the presentation of pictures, where each picture presents an amount of easy distinguishable artifacts that are connected through some well defined relations. Visualization itself has a number of specialized foci [4]. However, in this paper we are merely interested in program visualization, dealing with static and dynamic code visualizations.

Within program visualization, the scale and complexity of software are pressing issues, as is the associated information overload problem that this brings. In an attempt to address this problem the following concepts are considered to be important [5]:

- **Abstractions** Program representations resulting from program analysis contain already for middle size programs enormous quantities of information. Consequently, to be able to adequately represent and comprehend the most relevant program information, the information assembled needs to be focused. There are basically three techniques to focus information [6]: Information abstraction, compression and fusion.

- **Metaphors** The mapping from a program model (lower level of abstraction) to an image (higher level of abstraction) is defined through a metaphor, specifying the type of visualization. Most visualization techniques and tools are based on the graph metaphor (including the extensive research on graph layout algorithms). Other initiatives are the representation of programs as 3D city notations [7], solar systems [8], video games [7, 9], nested boxes [10, 11], 3D Space [12], etc.

- **Visualizations** It is not feasible to depict all kinds of program model phenomena in just one picture when the model carries too much information. Therefore each program model is depicted through various views, guaranteeing that the right subsets of objects and their relations are depicted and understood. Most effort to solve software complexity was put into different visualization forms, mainly the graph metaphor, including UML diagrams, to depict various program class and architecture views [4, 13, 14, 15].

The success and quality of any visualization depends on many vital features [4]: Animation, Metaphors, Interconnection, Interaction, Dynamic Scale. However, most vital for successful program visualization is the retrieval of necessary data for visualization and the availability of a suitable metaphor.

3 A 3D City Example for Program Development Visualization

Metaphors, when depicting real worlds and establishing social interaction [16], especially in virtual reality [17, 18], become very important. Essential is therefore the choice of metaphor to improve the usability of a system. One fundamental problem with many graphic designs is that they have no intuitive interpretation, and the user must be trained in order to understand them. Metaphors found in nature or in the real world avoid this by providing a graphic design that the user already understands. When illustrating a reverse engineered architecture, it is important for the understanding of a program that the final picture is adjusted for the individual [19, 20]. Therefore, we are currently developing a visualization architecture that allows all kind of metaphors to increase individual program understandability.

Within our unified recovery architecture [6, 21], the selection of metaphor can be undertaken depending on the user's requirements and focus in visualizing program information. Currently, we are implementing a 3D City metaphor to our architecture to support program understanding within three dimensions - this could be changed for an alternate world in a user dependent fashion. Figure 1 shows a screenshot of the running example, where buildings denote components (mainly Java classes) and the city itself represents a package. Different metaphors between the source code and the visualization are possible, i.e. components must not always be mapped to buildings and packages to cities. Other compilations are thinkable, where e.g. buildings are mapped to methods. To support the user with an intuitive interpretation of a software city and to increase the overall realism of the metaphor, we added trees, streets and street lamps to the figure.

Further, the figure illustrates both, static as well as dynamic information about a program. From a static point of view, the size of the buildings give the system maintainers an idea about the amount of lines of code of the different components. The density of buildings in an certain area shows the amount of coupling between components, where the information for this can easily be retrieved from metric analysis. The quality of the systems implementation within the various components is visualized through the buildings structures, i.e. old and collapsed buildings indicate source code that needs to be refactored.

From the dynamic standpoint, cars moving through the city indicate a program run. Cars originating from different components leave traces in different colors, so that their origin and destination can easily be determined. Dense traffic indicates heavy communication between various components. Performance and priority are depicted through the speed and type of vehicles. Occasionally exceptions occur, where cars collide with other cars or buildings, leading to

Proceedings of the Seventh International Conference on Information Visualization (IV'03)
1093-9547/03 $17.00 © 2003 IEEE

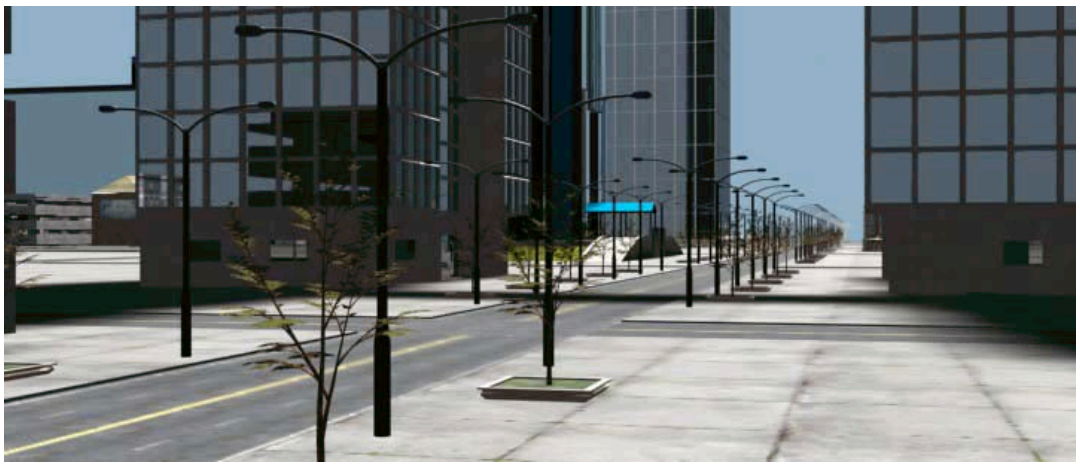

Figure 1. 3D City

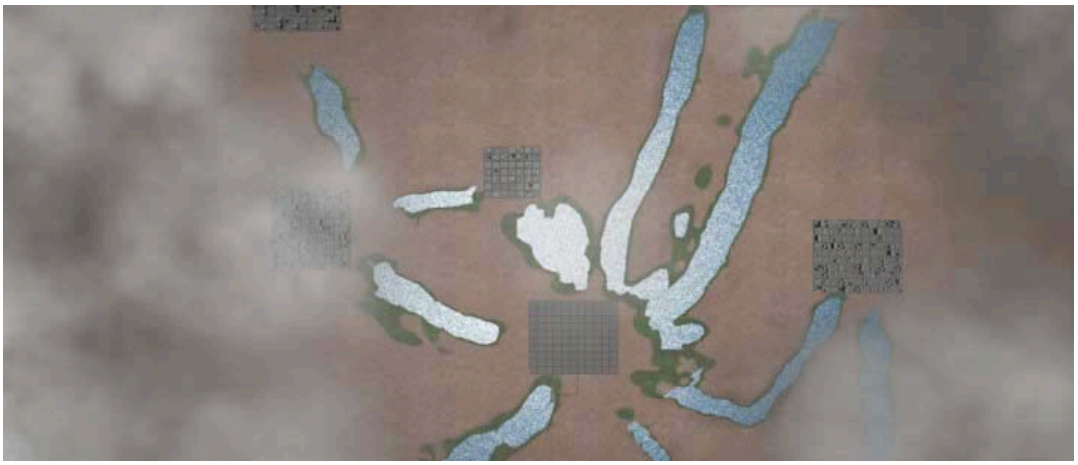

Figure 2. 3D City from Top

explosions.

A satellite view of our analyzed system can be seen in Figure 2, where cities (packages) within the architecture are connected via streets (two-directional calls) or water (uni-directional calls). Information between cities (packages) is passed via boats and vehicles. Again, dynamic as well as static information is illustrated. Clouds in that figure cover cities that are not of current interest to the user and hence hidden.

The general idea is to fly interactively through a reverse engineered software system and depict it within a 3D city. The user must have full freedom in zooming and navigating through the system, and even be able to perceive the system not only on the usual computer monitor, but also within a virtual environment. However, the figures represent, so far, only static and dynamic information about a reverse engineered software system, which is received from the programs source code. For maintenance, however, we wish for more information to answer cost related questions, like where does the main maintenance costs occur in the system. In order to answer this questions in a picture, we present next a software production cost related 3D city metaphor.

4 A Metaphor Designed to Highlight Production Information

Program visualization can provide a large reduction in effort associated with program understanding. However, in order to increase the reactivity of a development environment with respect to business realities, highly specialized, visual information should be delivered in a timely fashion to those people that can effect greatest impact.

Proceedings of the Seventh International Conference on Information Visualization (IV'03)
1093-9547/03 $17.00 © 2003 IEEE

Figure 3. 3D City from Top without Business Info

Figure 4. 3D City from Top with Business Info

Developers are mainly interested in information about functional and non-functional issues of a program, e.g. which components do I have to modify in order to change the security aspect of my system. Project managers, need to know to which parts of the program his team is allocated to, answering questions like, can we meet the next deadline? Additionally, managers and vendors are interested in hot spots, which means components that have been modified frequently, indicating the high cost areas of the systems maintenance. Designers and maintainers are more interested in the overall structure of the system, indicating places, which are of heavy or little use. This helps to decide which components to remove or how to restructure the system in order to achieve a higher performance or reliability.

Although, Figure 2 and Figure 3 help to visualize various static as well as dynamic aspects of a reverse engineered system and hence increase the understandability of the system, they do not provide support for the various additional demands that developers, designers, vendors and project managers have. Therefore, we apply a cost focused metaphor over the 3D city metaphor, to visualize additional business related information of a system. Figure 4 shows our approach.

In the figure we illustrate various additional aspects relevant for different types of stakeholders.

- *Work Distribution.* The components currently being modified by the staff are indicated in yellow with the respective names. This gives an idea about the progress of the maintenance or development team and can help to estimate the total time needed.

- *Recycling.* In brown, those parts of the system are

Proceedings of the Seventh International Conference on Information Visualization (IV'03)
1093-9547/03 $17.00 © 2003 IEEE

illustrated, which are no longer needed. These parts have not been used for quite a while and increase only the total complexity of the system. Those parts should be further investigated for removal.

- *Hot Execution Spots.* Components with frequent execution are indicated by a surrounding fire. In order to increase the reliability and performance of a system it is beneficial to investigate these components carefully.

- *Aspects.* Aspects, from Aspect-Oriented Programming [22, 23], can be depicted in various colors. In the figure here, distribution is shown in blue and synchronization in green. These spots help out to point out functional or non-functional cross-cuttings that can be investigated and further changed by experts.

- *High Costs.* Buildings with flashes indicate frequent component modifications. The result is an increased maintenance cost and hence also an increased cost for the entire software project.

The list above is not complete. Many more production related issues exist that could be depicted. Vendors and managers might see the quality of a software system at once, by having a short look at the 3D city. Lots of fire, flashes and mud indicate high cost areas of code and unacceptably high risk regarding the ongoing health of the system.

Developers might benefit from business process, use-case, control flow, data-flow, and event-based visualizations. For example, a dynamic picture of events occurring on various positions in the city and being handled on other positions (indicated by e.g. colors) can aid the understanding of event-based software development. Another example might be an animation of cars driving between the different cities, marking them in color, to indicate which components are used when and for which use case.

Further, static and dynamic source code analysis can be illustrated, showing a changing city, depending on the type of analysis applied. For example, when applying the concept analysis, which computes maximum sets of objects using the same set of attributes, different components can be formed, depending on the user interaction on the concept lattice. For reverse engineering, component detection can be optimized by interactively illustrating the changing picture of the city while the concept lattice of the analysis is being changed.

Finally, the navigation within the city is important in order to present the 3D city to the user in the most natural way. Software maintenance should not necessarily require reading millions of lines of code. Navigating through the city from top or even from inside a car (program counter), might make maintenance a game. Complex, huge software systems could be comprehended much faster.

5 Related Work

At present there exists some work on visualizing source code in three dimensions, however little effort was made to depict production relevant information in 3D. Knight and Munro [7] describe what they call the 'software world' - a system being visualized in the world. Cities represent source files that that may contain one or more classes which themselves are represented as districts and finally methods are shown as buildings. Here colors are used on buildings to encode the type of methods: public or private. However, the focus of this paper is the application of virtual reality technology to the problem of visualizing data artifacts and not production related visualization.

In [24] the authors present a case study, which shows the possibility of deploying 3D visualizations in a business context. In the case study a reusable collection of behaviors and visualization primitives is presented (3D gadgets), with the intention to deploy visualization components in a software architecture. However, the paper focuses on business process visualization to validate requirements and to acquire feedback on design-tradeoffs and not on program understanding or software maintenance.

6 Discussion and Future Work

As mentioned above, the 3D city is an example metaphor for our unified visualization architecture, that is currently being developed. The architecture supports any kind of reverse engineering, following three basic steps: analysis of a program, focusing (which means filtering, folding, fusing, etc. of program information), and the visualization of retrieved source code information. For the visualization, a mapping between the data representations and the final picture is needed. Hence, we need metaphors.

Currently, we are about to finish our unified recovery architecture that allows us to plug in various visualizations, including metaphors and layout algorithms, providing the best fit system view for an individual user. The step to follow is to implement the 3D city metaphor in e.g. OpenGL4Java, in order to perform tests and evaluations. The present study about the 3D city is based on our 3DStudio Max implementation.

The unified recovery architecture permits the visual correlation of code structure information and development house operational information in a single viewscreen. The choice of metaphor can also be altered depending on the relative importance of each aspect of the code (or cost related to the code) to the viewer at hand. Once these tools are robust and readily configurable the test will be to undertake a medium scale industrial trial where scalability, performance and time constraints will test the ability of the visualization tool to provide valuable information to the organization.

Proceedings of the Seventh International Conference on Information Visualization (IV'03)
1093-9547/03 $17.00 © 2003 IEEE

7 Conclusion

In this paper, we have illustrated our 3D city metaphor that we are currently developing for our unified recovery architecture. Further, we have discovered the need to depict more than just static or dynamic program information for increasing a programs understandability and maintainability. We need to depict production cost related information of a system that shows the right stakeholder the right information using minimal time and effort. In general we have tried to present a vision where software maintenance must not always be labor intensive, trying to understand and alter undocumented legacy code, but also fun, when navigating through a software city in 3D.

References

[1] B. W. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.

[2] B. Lientz, E. Swanson, and G. E. Tompkins. Characteristics of application software maintenance. *Communications of the ACM*, 21(6), June 1978.

[3] R. K. Fjeldstad and W. T. Hamlen. Application program maintenance study: Report to our respondents. In *Proceedings GUIDE 48, Philadelphia*, April 1983.

[4] J. Stasko, J. Domingue, M. H. Brown, and B. A. Price, editors. *Software Visualization*. MIT Press, 1998.

[5] C. Knight. *Visual Software in Reality*. PhD thesis, University of Durham, 2000.

[6] T. Panas, W. Löwe, and U. Aßmann. Towards the unified recovery architecture for reverse engineering. In *Int. Conf. on Software Engineering Research and Practice, Las Vegas*, June 2003.

[7] C. Knight and M. C. Munro. Virtual but visible software. In *IV00*, pages 198–205, 2000.

[8] P. Damien. Building program metaphors. In *PPIG'96 Post-Graduate Students Workshop at Matlock, UK*, September 1996.

[9] K. Kahn. Drawing on napkins, video-game animation, and other ways to program computers. *Communications of the ACM*, 39(8):49–59, 1996.

[10] J. Rekimoto and M. Green. The information cube: Using transparency in 3d information visualization, 1993.

[11] S. P. Reiss. An engine for the 3d visualization of program information. *Visual Languages and Computing (special issue on Graph Visualization)*, 6(3), 1995.

[12] J. Rilling and S.P. Mudur. On the use of metaballs to visually map source code structures and analysis results onto 3d space. In *Ninth Working Conference on Reverse Engineering (WCRE'02)*. IEEE, October 2002.

[13] T. Ball and S. G. Eick. Software visualization in the large. *IEEE Computer*, 29(4), 1996.

[14] C. Lewerentz and F. Simon. Metrics-based 3d visualization of large object-oriented programs. In *1st Int. Workshop on Visualizing Software for Understanding and Analysis*, June 2002.

[15] P. Eades. *Software Visualization*. World Scientific Pub Co, 1996.

[16] C. Russo dos Santos, P. Gros, P. Abel, D. Loisel, N. Trichaud, and J.P. Paris. Metaphor-aware 3d navigation. In *IEEE Symposium on Information Visualization*, pages 155–65. Los Alamitos, CA, USA, IEEE Comput. Soc., 2000.

[17] G. Fitzpatrick, S. Kaplan, and T. Mansfield. Physical spaces, virtual places and social worlds: A study of work in the virtual. In *CSCW'96. ACM Press*, 1996.

[18] K. Vaananen and J. Schmidt. User interfaces for hypermedia: how to find good metaphors? In *CHI '94 conference companion on Human factors in computing systems*, pages 263–264. ACM Press, 1994.

[19] J.F. Hopkins and P.A. Fishwick. The rube framework for personalized 3d software visualization. In *Software Visualization. International Seminar. Revised Papers (Lecture Notes in Computer Science Vol.2269). Springer-Verlag*, pages 368–380. Berlin, Germany, 2002.

[20] S. North. Procession: using intelligent 3d information visualization to support client understanding during construction projects. In *Proceedings of Spie - the International Society for Optical Engineering*, volume 3960, pages 356–64. USA, 2000.

[21] VizzAnalyzer. `http://www.msi.vxu.se/~tps/VizzAnalyzer`, 2003.

[22] G. Kiczales, K. Lieberherr, H. Ossher, M. Aksit, and T. Elrad. Discussing Aspects of AOP. *Communications of the ACM*, 44(10), October 2001.

[23] T. Panas, J. Andersson, and U. Aßmann. The editing aspect of aspects. In I. Hussain, editor, *Software Engineering and Applications (SEA 2002)*, Cambridge, November 2002. ACTA Press.

[24] B. Schönhage, A. van Ballegooij, and A. Eliens. 3d gadgets for business process visualization - a case study.

Proceedings of the Seventh International Conference on Information Visualization (IV'03)
1093-9547/03 $17.00 © 2003 IEEE