Organisational Trails through Software Systems

Claire Knight and Malcolm Munro
Visualisation Research Group
Research Institute in Software Evolution
Department of Computer Science, University of Durham,
South Road, Durham, DH1 3LE, England.
+44 191 374 2554
{C.R.Knight, Malcolm.Munro}@durham.ac.uk

ABSTRACT
The use of three dimensional software visualisation techniques has the power to transform the way that tacit knowledge gathering and retrieval takes place during many software maintenance activities. The focus of this work is the understanding and management of software from various perspectives and therefore it is the visualisation of these software artefacts that is of greatest interest. This work moves towards providing a generic visualisation that is applicable to the management of many software systems within the bounds of the organisation, with the ability to view various artefacts over time (and thus their evolution) and also to incorporate lower level visualisations in three dimensions.

Keywords
Software Visualisation, Program Comprehension, Software {Evolution, Maintenance, Engineering, Management}

1 INTRODUCTION

It is a well-known and documented fact that software is large and complex and that despite many software engineering advances this is still the case. For many this size and complexity is seen as a good barrier to hide behind with comments such as "inherent in software" and "unvisualisable" thrown out. Whilst software may remain complex, and indeed this may be unavoidable with the demands placed on the systems of today, the hiding behind these sorts of comments is not a good thing for the field.

The work documented in this paper was originally driven, and to some extent still is, by the need to find alternative ways of visualising software. The relationship between graphics and the program comprehension field, indeed the software engineering field, is a long one but it is precisely these fields that have been slow to embrace new and novel techniques. This is despite documented shortcomings in the oft-used representations that centre on two-dimensional graphs. Software visualisation has to overcome many problems, not least in that there is usually a large amount of data to be presented and that those data items are then related in many complex and varied ways. Software is also inherently intangible and this therefore requires that the field consider representational issues, a situation that does not hold true for all visualisations [1, 2, 3]. As an added problem, software systems evolve over time [4] and this affects the visualisations that can be considered useful and remain useful over that time.

Program comprehension is the process of understanding an existing piece of code [5]. It is a gradual process of building up the necessary understanding by examining sections of the source code. Using the knowledge gained from the source code explanations and understanding can be built and refined. These process can be applied to the individual pieces of software that go some way towards composing the whole system, but it is also necessary to consider the management and organisational issues which tend to have profound and far reaching implications on the software. This level of association is at best referred to in passing by non-technical managers, but is often not well appreciated by those who have to manage the technology. There is also the same problem of loss of tacit knowledge at a system level, just as can happen at the code level with development and then maintenance activities.

This paper is organised by presenting the visualisation that is proposed to address the problems related to the management and understanding of software systems over time. A discussion of some of the many future directions that could be followed is provided as a route to further work and to move the software engineering visualisation field to the state that the information visualisation one already finds itself in.

2 SYSTEM VISUALISATIONS

Software Visualisation is a powerful way of viewing the many and varied artefacts that composed together form a computer system. It also has much power and use in the

representation of the complex interrelationships between these artefacts. The addition of interaction, annotation, and detail control greatly adds to the possibilities that visualisations have as useful and flexible tools.

There are still problems and issues to be tackled when visualising any data set in an acceptable and reasonable way. The first of these is to make available ways for the user to be able to explore and navigate the complex environments in which the visualisations are located. There is also the issue of converting suitable data to well-founded information (for the task in hand); metaphors. Although there are many more issues, the final one presented here is the management of increasingly abstract problems and being able to communicate to users (and managers) the visualisation visions accurately.

In order to be able to view and manage a variety of software systems (and their component artefacts) all with the same aim of integrating organisational and system knowledge was a motivating factor. This led to the creation of a visualisation metaphor and mapping that aimed to present many dimensions of this data to the user at once. It also had the added intention of allowing the important aspects of time and evolution to be tracked, managed, and even possibly predicted making use of visual display as well as the underlying data.

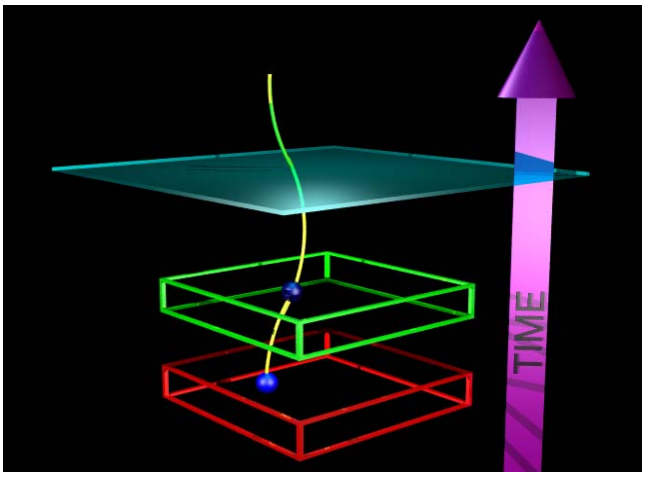

Figure 1

A good reason for visualising this information is that it allows for the integration of various filtering and overview mechanisms to be used to be able to grasp limited portions of the data. It also allows for the detail to be seen when necessary. An added benefit is that visualisation allows these filtering/detail decisions and the data focus to be user driven. Other techniques, for example data mining, seek to find what the algorithms think of as interesting patterns in the data and then present these to the user. For management of complex systems such as software this may not be adequate as the underlying data factors may not accurately reflect the management needs. Further data work and mining algorithms may obviously contradict this and could well be a future enhancement to such visualisation techniques.

System Data

As well as the basic source code, possibly the only definitive view of the current system and what it does, there is the organisational knowledge about the software. At a programmer level some of this may be shared, and in some visualisation systems the storage of this on an ad-hoc basis attached to the code is advocated. The problem with this knowledge is that is often only passed on by word of mouth, and the nature of it means that is in a form that does not suit formal documentation (something many programmers dislike anyway).

Moving up from the source code there is higher level information about the architecture and patterns employed in the software. Some of this may have been extracted from the code, or the only source may be the original documentation, from which the current version may have deviated. Either way, this information can be utilised in the process of knowledge discovery and refinement to aid other decision making. If it is known that the current system does not match the original design then the requirements (both original and updated if available) can be used to see to guess intents, to work out the amount of diversion, etc.

Another useful log of both the source code and the system as a whole evolving are the change requests, released versions, bug reports and bug fixes. These provide a form of traceability through the life of the software, and even if the original code lines have changed and are no longer recognisable these provide a source of evidence for when, where and why alterations were made.

Tied in with the versioning of the software is the market data (even if the product is only used internally within one company). The use of, and uses to which the product is put can be logged and this information can be integrated into the views, as can standard sales data (if this is applicable). This information can also be of use with bug reports/fixes since the use may influence the problems found, either with erroneous code or missing functionality.

Graphics Overview

The history of “objects” in this type of data set is considered important, not only because of the timeline that can be generated from such information but also because of the various requirements that can be attached to the different component parts of the system. These “objects” are also the focus of any number of projected events; for example, the release of a new version. The amalgamation of all of this data (regardless of any other data or attributes) creates a complex inter-related mass, which to be understood in both summary and detail requires some form of display and filtering mechanism to reduce information overload problems.

It should come as no surprise that this is the end result of

collating facts about software systems; software is purely a representation of complex thought patterns that have been employed to solve possibly complex real world problems. Software is an embodiment of an idea; it is the solution in a real form although the paradox is that the software is then intangible, at least at the code level. There is no inherent form beyond that code which presents and allows interaction with the user interface, or the clusters taken up on whatever storage medium that the source and executable files are stored.

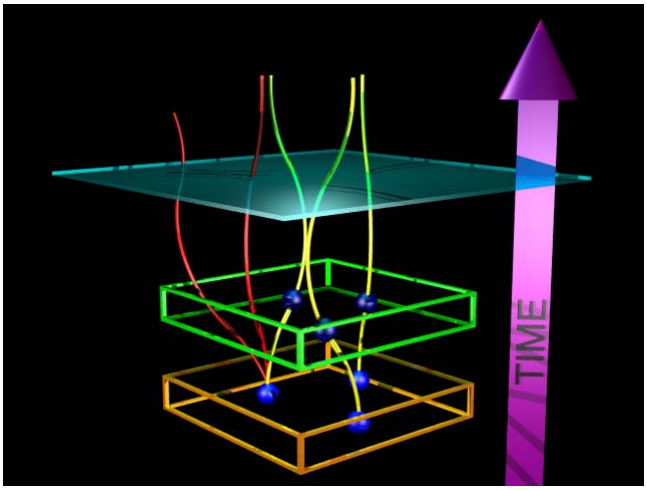

Figure 2

So there is a situation of complex information with no inherent form, and this information needs to be not only understood but also used as the basis for analysis (which some suggest requires a deeper level of understanding). There are some existing software representations that could be used for the visual presentation of the source code files and some attributes relating to those. These lack something in that they ignore the relationships between the objects, the history of that object and any one of the projected futures, plus the provision of deadlines.

An important aspect of the data under consideration here is the consequence of time. In this case an object (any component part of the software system including management influences) has a history through time and it also has any number of possible predicted futures at for any given point in time. These predicted futures are used for management and impact analysis based on set deadlines such as releases of software and system changeovers. Even third party software that is a component of the software system is involved. It would have a history trace of releases with future traces for known (or predicted) upgrades. Whilst the software unit itself is outside of the control of the management of this system, it changing has an impact on any aspect of the system that uses or relies on it.

Since much of the focus (at this stage) is based around the passing of time, the correlation of predicted and actual future, and the use of previous data for extrapolation to aid decision making, then the main focus of the visualisation is that of time. Of the three dimensions available, the up-down, vertical dimension is used for the passing of time. There is a potentially infinite stack of events and representations of the system at distinct times that are arrayed in this vertical space. Also running up through this space, passing through planes and representations are time ribbons of predicted (and actual) time to show the progress of individual objects and their predicted futures. This can be seen in Figure 1. The view at any one time is purely a window onto this "virtual" stack of system visualisations, events and time traces. This view provides a high level overview of the system.

A more detailed view is available should the user choose to explore a section of the time stack in any detail. The system representations (as yet not described) contain a visual representation of the system at that moment in time. These objects may be of interest in themselves and the user has the freedom to explore them. Once looking at this level of detail of the vertically arranged landscape, there may be variations in the vertical placement of objects as the representation demands. It is for this reason that the use of bounding boxes is used in the higher level views. This is evident in Figure 2 where three objects of interest can be seen through time, but held within bounding boxes. Once working within this "new" three dimensional landscape of the system the user has the freedom to bring in aspects that are more evident at the higher level view; that of time and impact. For any given object it is possible to project onto a semi-transparent plane (to maintain overall context) extra data about the chosen object. Figure 3 shows the insertion of an extra plane into the visualisation. This intersecting plane can then provide (amongst other things) some of the time information that pertains directly to this object. Furthermore, time information or predicted releases of dependent software can then be included in much more detail. This then allows trains of thoughts to follow object dependencies.

A possible criticism of the time ribbons is the complexity that may exist if they were all permanently visible. Such a criticism is well justified and to combat this, they can be selectively turned on and off by the user as their investigation demands. The other side of this is that they can show a lot of predicted futures from an object and therefore indicate in one glance that a particular object is the subject of much interest. Once in the detailed view there is the same control over what time ribbons are displayed, both on surrounding objects and the one being investigated. The extra planes of information that show detail are also configurable in this way. There is also control of information and analysis available by inserting query planes into the stack, which creates a new (predicted) representation of the system at that given time based on the predicted change and time histories of each object at that stage.

An additional level of tracking that is not covered in the visualisation as described so far is the individual evolution of objects rather than the system as a whole. There may be some indicated between the system snapshots on the stack if the lower level representation uses placement to indicate this but it cannot be relied on. To some extent system releases will highlight possible points of major evolution but the detail is available upon further investigation. An object may have undergone much change between the last two releases but none before that, whilst something with continual minor change between every release has cumulatively less change. Traditional colour scaling does not distinguish between this, but by stacking a representation of the system this information can be better displayed if it is captured.

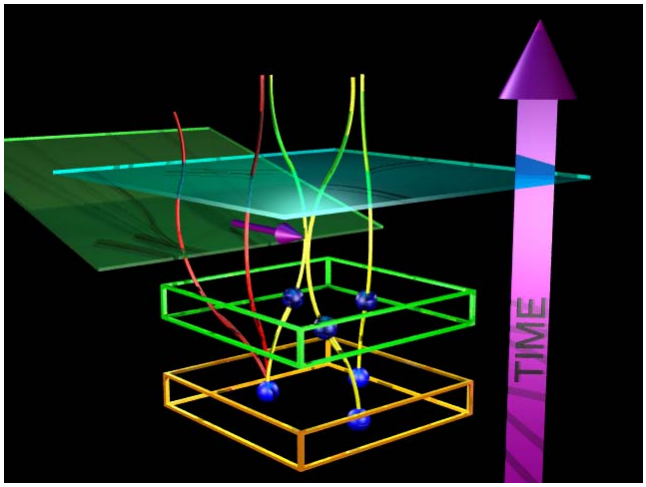

Figure 3

The system representations are not of great importance for the visualisations at this stage as the main concepts relate to the representation and management of the time and histories associated with the system at different stages of its existence. It is for this reason that they have not been described. This is not to say that the representations themselves are not important; indeed to be able to deal with spatial placing and evolution (to name but two issues) requires much work if the visual display is to be effective and useful.

3 CONCLUSIONS

Programming is a creative task; the artistry and wide range of applicable ways of implementing a solution to the problem to which the finished program is to be applied is a skill. Process management and design methods both help and hinder this process, differently at different times. What is important is that the creative aspect is not lost; that those skilled in this form of problem visualisation and solution creation are not simply permitted to be managers or to move elsewhere. These people embody much of an organisation's system knowledge; despite the management attitude and press coverage this knowledge is both powerful and extremely important to an organisation.

One way of achieving this is through the use of system visualisation. Visual representation of data provides another way for programmers and managers to interact with same data at different levels of abstraction. This combination of views, and therefore the combination of management minds with those that know the system best, provides the best potential for dealing with the maintenance and running of software systems. The code views (an integral part of any system) are still understood and controlled by those best qualified to deal with them. The external influences, changes in release dates or suppliers updating components, organisational changes, etc. can then be integrated with the code objects in the visualisation. This is a powerful combined representation of data that is often lost or mismanaged due to the vast amounts capable of being generated for any system that is of reasonable size.

This paper has provided a glimpse of the future of research in software and system visualisation. There are many areas where work is still to be done both with issues identified and presented in the paper and with the many others that exist, but it is an area in which much benefit can and should be expected for the research effort put in now. The ability to enhance a users own cognition is a powerful notion, and can lead to powerful use, which in turn should benefit the work being done with visualisation tools.

ACKNOWLEDGEMENTS

This work has been done with the support of an EPSRC ROPA award: VVSRE; Visualising Software in Virtual Reality Environments.

REFERENCES

1. Knight, C., and Munro, M. Comprehension with[in] Virtual Environment Visualisations. In *Proceedings of the IEEE 7th International Workshop on Program Comprehension*, Pittsburgh, PA, (1999), 4-11.

2. Eick, S. G. Engineering Perceptually Effective Visualizations for Abstract Data. In *Scientific Visualization Overviews, Methodologies and Techniques*, G. M. Nielson, H. Mueller, H, Hagen (Editors). IEEE Computer Science Press, (February 1997) 191-210.

3. Feijs, L., and De Jong, R. 3D Visualization of Software Architectures. *Communications of the ACM, Vol. 41, No. 12*, (December 1998), 72-78.

4. Lehman, M. M., Perry, D. E., Ramil, J. F., Turski, W. M., and Wernick, P., Metrics and Laws of Software Evolution – The Nineties View, *Proceedings of the 4th International Symposium on Software Metrics*, Albuquerque, New Mexico, (November 1997).

5. Von Mayrhauser, A., and Vans, A. M. Program Comprehension During Software Maintenance and Evolution, *IEEE Computer*, (August 1995) 44-55.