DUBLIN CITY UNIVERSITY
SCHOOL OF ELECTRONIC ENGINEERING

Visualisation Tools for Decision Support in Complex Software Projects

Final Report

Niall Whelan

August 2013

Acknowledgements

I would like to thank my supervisor Dr. Derek Molloy for his support in my project selection and encouragement in defining a challenging scope of work. I would also like to thank Ariba Inc. for their support throughout this course. Finally, I would like to thank my family for their patience and support over the last two years.

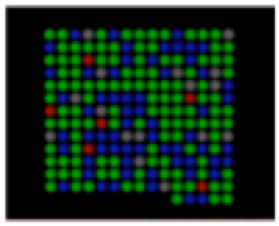
Declaration

I hereby declare that, except where otherwise indicated, this document is entirely my own work and has not been submitted in whole or in part to any other university.

Signed: ..... Date: .....

Table Of Contents

Visualisation Tools for Decision Support in Complex Software Projects



Niall Whelan

***Abstract*—Agile development for Cloud based applications creates many new challenges for software development, maintenance activities and downstream processes such as Documentation, Training, Support, and Localisation. The aim of Information Visualisation is to use "computer-supported interactive visual representations of abstract data to amplify cognition" [1]. The Software Engineering process now more than ever requires data driven decision support and knowledge management specifically focused on content change management. This project proposes to employ well founded visualisation techniques to design a decision support toolset to harness data from live software project artefacts to accelerate daily decision making in various software project phases.**

***Index Terms*—Visualisation Tools, Agile Software Development, Heatmaps, Node-Link Graphs, Temporal Cubes, Decision Support, ISO 9241-11.**

I. Introduction

This project aims to employ appropriate visualisation designs to support all stages of the decision making process of specific software project tasks.

Central to Software Development is constant change; source file changes, requirements changes, test infrastructure changes, terminology changes, design changes, translation changes, etc. Software Projects often are quite complex [2, 3] due not only to the nature of the systems being developed but also due to market complexity, organization complexity, and process complexity[4]. Thus the challenge for software project managers is to deal with this complexity in two main ways: 1. Complexity reduction in the design of the system, 2. Complexity control related to product and process, communication among all interested parties, and change management. It is the area of software change management which this project will focus on and explore how visualization tools can be employed to positive effect on this process.

Change management typically results in a number of basic questions about software such as:

- At a given point in time, how much are software features changing?
- Which files are changing?
- Which components are changing?
- Is the change frequency increasing, decreasing, constant?
- Is there a pattern of stabilization in the file set?
- Are changes occurring after a component was deemed complete?

The degree to which these questions are answered accurately greatly affects subsequent decisions and ultimately affects software quality, release schedule, and content of that release. There are many tools available to software practitioners to manage and track quality, schedule, and content. However many of these system are driven by human data entry and filtering, and not necessarily automatically data driven by the software development process. For example, defect tracking systems generally rely on user data entry. Accuracy of entry and duplication of reported defects results in a necessary filtering process in which valid defects can often be filtered out of a particular release. Requirements management tools also rely on user input and are not data driven by the software development artifacts. As a result the decision making process as a whole is less data driven and more driven by the experience of the practitioners. Having worked in the software industry for the past 25 years it is clear there is a growing need for data driven visualisation tools to step into the gap where daily change affects daily decisions, but the decision makers may not have the appropriate information to make the best decisions possible.

II. Prior Work

In the literary review [5] for this project a combination of works are referenced related to Data Visualisation in Software Engineering [6] and Visualisation Tools for Decision Support [7-9]. However, what this research exposed was the lack of commercially available or open-source domain specific tools for visualizing software change and dealing with its challenges outlined above.

Much work has been done in researching the efficacy of various 2D and 3D visualisations [10-12]. Many interactive 2D representations have found their way into various commercial Business Intelligence applications, but less so for 3D. However, for this project both 2D and 3D solutions are considered since temporal multivariate data [13] and semantic associations [14] are well represented in 3D, despite the added complexity of design and implementation.

III. Technical Description

In order to realize a new visualization toolset for software change management it was necessary to consider the existing tools and development environments which would be most suitable to realize the desired design. The choice of 2D or 3D visualisations and degree of user immersion and interaction with the visualization were key selection criteria. For this project the aim was to develop prototype visualization tools and test the efficacy of those visualizations. Therefore the time constraint of the project also influenced the platform choice for development.

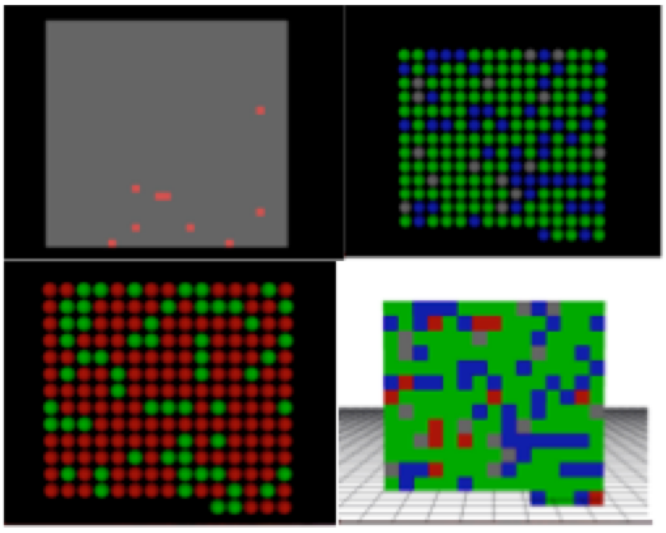
A. Visualisations

The visualisation design follows a well established series of steps to journey the user from an overview of a data set to the final reduced data set selection for a particular task:

- 2D Overview Map
- Zoom and Detail
- Focus and Context
- Final Data Selection and Information Display

The added goal of this project was to try provide an immersive user experience which moved the user through a series of visual queries in a logical manner for each particular task so as to minimise distraction while focusing on the required data selection decisions to reach the desired data set as efficiently as possible.

The concept of volatility is used to describe the frequency of change to files and various colours are chosen to highlight various states of volatility. Testing of the effectiveness of these chosen colours is addressed in section IV below.



Although the initial overview is essentially a 2D representation of the changing file set, a 3D framework was chosen to render the 2D view in order to facilitate seamless transition from a 2D representation to subsequent 3D exploration of a reduced data set, employing semantic information to visualize relationships of various files to their parent components. It also allowed exploring temporal cube representation of the data as an alternate visualization approach.

B. Software Design

The chosen development platform for this project was Qt using C++, OpenGL and Qt3D.

A number of platforms considered for this project include: Java3D, Mathlab, Qt and Qt3D & openGL.

Other commercial Business Intelligence tools like Business Objects, Tableau, ClikView were discounted for two reasons: a. lack of immersive 2D and 3D experience, b. Cost.

The two reasons MathLab was not selected were lack of custom 2D and 3D object creation and interaction, and cost. Although it is widely used in academic and scientific communities its barrier to adoption in the commercial software industry is perhaps the combination of cost and lack of data analysis generally employed in the software development community.

Visualisation Functionality	Java, Java3D	Mathlab	C++, Qt & Qt3D
Raw Data Load (CSV)	Yes	Yes	Yes
Live Database connectivity	Yes	Yes	Yes
Graphic Object Interactions	Yes	Yes	Yes
Brushing	Possible to create	Yes	Possible to create
Annotation	Possible to create	Yes	Possible to create
Hover Tips	Possible to create	Yes	Possible to create
3D Features			
3D Nodelink Graphs	Possible to create	Not available	Possible to create
3D Picking	Yes	Yes	Yes
Other Considerations			
Low Cost	Yes	No	Yes
Rapid Development	No	No	Yes
Short Learning Time	No	No	No
Cross Platform Development	Yes	Yes	Yes

C. Qt, Qt3D and OpenGL

Qt is a rich C++ UI framework for Desktop, Embedded, and Mobile OS cross-platform application development [15]. It provides Open Source and Commercial license versions and enjoys an active community of over 500,000 developers worldwide.

During initial evaluation of the Qt framework, its support of openGL was investigated. Qt processes an OpenGL library – QtOpenGL, but also supports 3D development specifically with Qt3D [16] which extend the QML declarative language to provide 3D rendering, Scene management, asset loading, and geometry building capabilities.

Rendering large numbers of primitive objects such as spheres can quickly overwhelm processing capabilities unless optimization of the rendering process occurs. OpenGL implements a rendering pipeline which is a sequence of processing steps to deal with all the calculations for vertex shading, geometry shading, primitive assembly to prepare for clipping objects outside the viewport, before final pixel rasterization and final fragment shading to determine which pixels are drawn and with which final colour values.

Qt3D provides levels of detail to be selected when defining primitives such as spheres. Level 1 uses 64 triangles, level 2 uses 128, up to level 10 which uses 32,768 triangles to produce very smooth object surface. Generally, level 3 (256 triangles) was used in this project, except for temporal cube rendering where level 1 was selected for 3rd and subsequent depth objects to minimize computations. Since the data sets being used in this project have up to 1,200 objects, even rendering at level 3 produced 307,200 triangles to be drawn and shaded for each frame rendering.

D. Class Design for the system

Management of large numbers of objects in 2D or 3D is greatly simplified when a Scenegraph data model is employed. Qt provides two classes, QGraphicsScene for managing 2D graphics and QGLAbstractScene and QGLSceneNode for managing 3D scene objects. Rather than employing both 2D and 3D models, this project uses 3D scene management exclusively.

The data model design consists of a SourceHistory class which acts as the container for each denormalised record loaded into the system. This data is then normalized and visual details for colour and position added in the VisualDataModel class. File changes are stored in the FileChange object.

Qt provides Model View Controller separation for UI and Data within its class hierarchy. This is maintained using the MainWindow class and CubeView class whereby visual data is loaded, stored and modified in MainWindow, but UI changes and visual object management is implemented in CubeView.

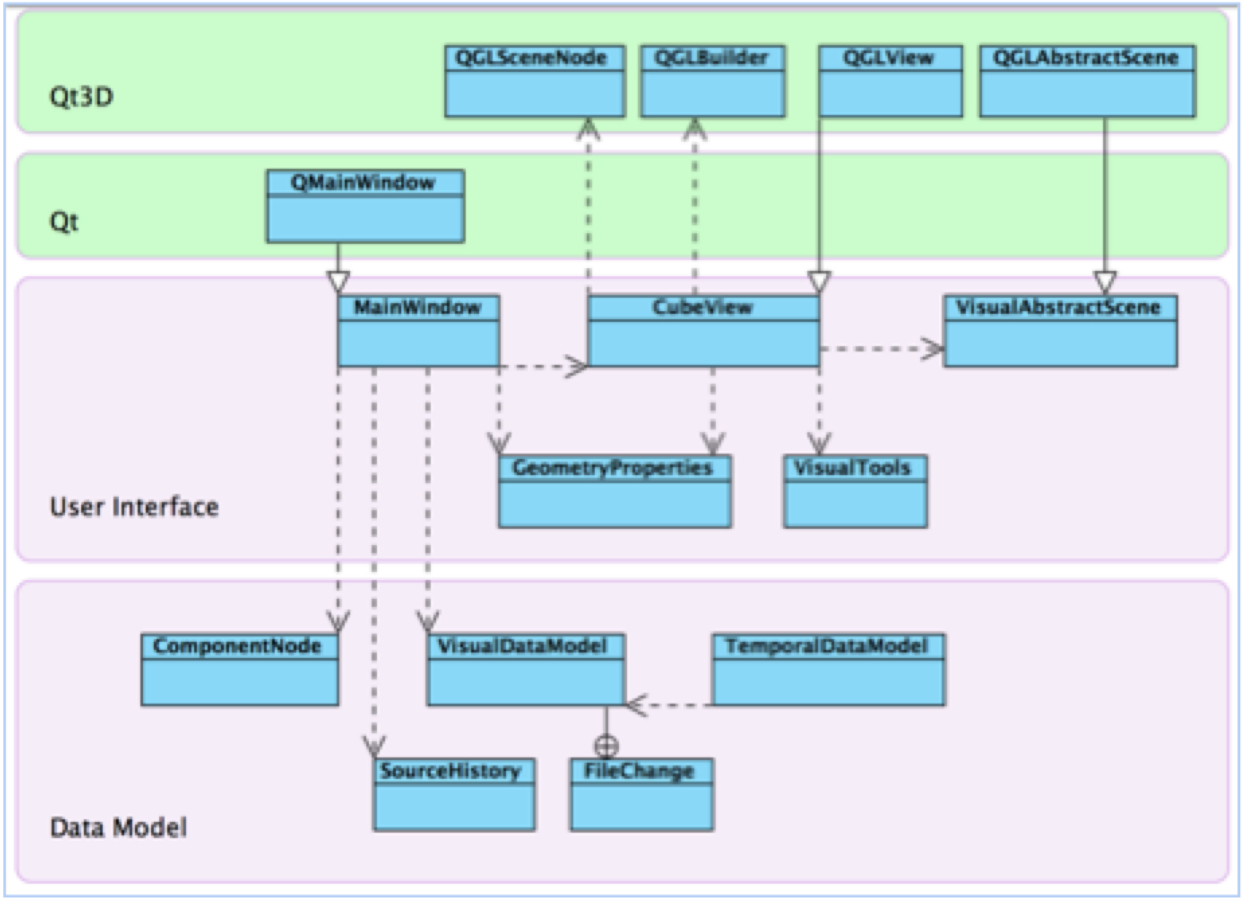
E. Implementation

The development platform chosen was a Macbook Pro running Mac OS X 10.8.3, with 8GB 1600 MHz RAM, 2.6GHz Intel Core i7 processor.

Qt installation was straightforward and testing the installation was done by compiling and running a number of sample Qt application from the Qt website.

1) Challenges

Qt3D was more challenging to install as the source code had first to be downloaded, compiled, and linked to the Qt installation. Specifying library paths in a new project to link Qt3D was more trial and error since the documentation was unclear on this process, and error messages were equally cryptic. Successful setup was again verifed using sample applications. All Application Source code is listed in [17]



IV. Results Obtained

A. Project Goals realized

This project was planned in a number of feature development and validation milestones. After the initial preparation phase of the Qt and Qt3D environment the first milestone identified four objectives which were achieved:

- Defining a Visual Data Model
- Creating an appropriate Scenegraph
- Loading data
- Creating a planar 2D view in a 3D world

The second milestone proved more time consuming in its initial objective, resulting in further work being left for future investigation. Milestone 2 consists of:

- Defining a set of Visualisations
- Defining a Decision Framework
- User Interaction Design
- Initial Workflow creation
- Testing and refactoring

A number of challenges related to the Qt Platform on the MacOS impacted 3D visualisation creation including:

- Retina display causing half size viewport rendering. This was subsequently overcome when connecting the Mac to an external display, at which point a 3rd set of screen settings were selectable. Using external screen setting the viewport match the full window.
- Picking based in full viewport mis-matched rendered objects, correct coordinates were not returned. This resulted in Brushing with mouse selection not being achieved.
- Lines used for a 3D Grid and node-link edges could only be rendered in black, standard glColor() method had no effect.

B. Usability Testing

1) Test approach

The standard chosen for usability testing was ISO 9241-11 as this method was also shown to be appropriate for prototype testing of Time-Cube visualization in [18].

Since the User Interaction Design was not realized in this project, the 2D visualisations were tested for effectiveness and satisfaction where the main interest was user feedback on background colour, primitive object selection, and their combination impacted by scale.

2) Test Data Generation

The initial prototype was developed using real project data exported from a Perforce version control system. The SourceHistory class encapsulated the file format. However, one project was not sufficient to adequately test the variation of changing data on the visualisations, so a program was developed [19] to generate nine variations of data files for small, medium, large file sets with low, medium, and high number of changes to each file.

# of Files	Shape	Colour	Background	Changes
30	Cube / Sphere	RGB / RGBa	Black / White / Grey	Low / Medium / High
200	Cube / Sphere	RGB / RGBa	Black / White / Grey	Low / Medium / High
1200	Cube / Sphere	RGB / RGBa	Black / White / Grey	Low / Medium / High
Subtotal	6	12	36	**108**

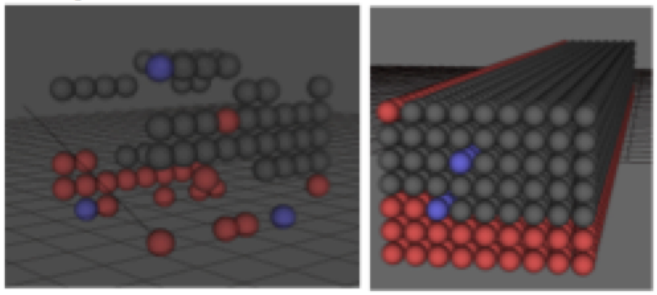
The 108 images were captured and stored in a hosted test database [19] in which users were able to log in and inspect all 108 images giving ratings of 1 to 5 for both effectiveness and satisfaction. Effectiveness was the degree to which the user felt the image reflected change to the file-set. Satisfaction was the degree to which the user liked or disliked looking at the image.

Six software engineers and project managers volunteered for the usability test and 650 individual rating were recorded against the image set. The findings are discussed in section VI.

C. Performance Testing

Performance testing was carried out on the rendering time for each dataset visualization, planar view, with component depth added, and finally temporal cube view.

Examples of component depth and cube view are shown below. Component depth is where each file relates to a parent class and many files can have the same parent. This semantic information is used to eventually build the node-link graph view. The Cube view creates a full set of objects for each file and also each build of the file set over time. In other words if the data set has 30 files from 5 builds of those files, 150 individual objects will be drawn 30 at each Z co-ordinate to a depth of 5.



File size (rows)	Planar (ms) Cube/Sphere	Depth (ms) Cube/Sphere	T-Cube (ms)
30	2/10	42/10	41/194
200	39/228	42/230	42/230
1200	61/343	61/351	Failure

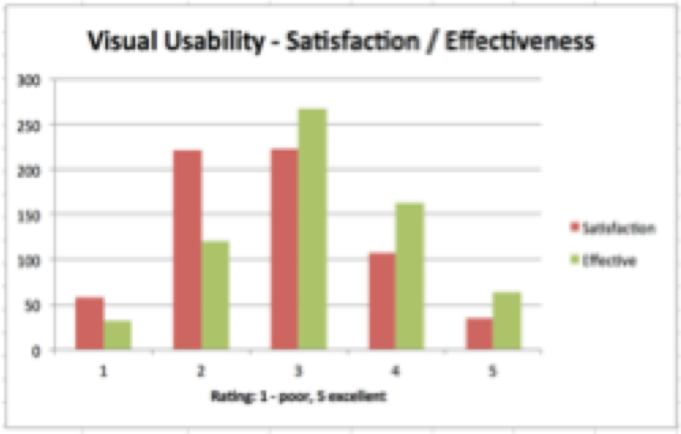
V. Analysis & Interpretation

A. Usability Analysis

Since the overall system prototype was not realized, the usability testing focused on the generated images. As well as numerical results below, some users also offered comments and suggestions for alternate visualization approaches.

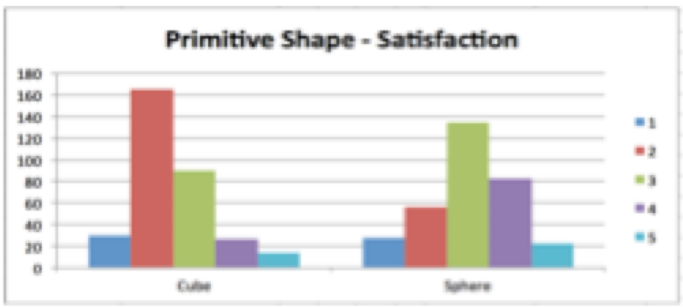
1) Overall usability

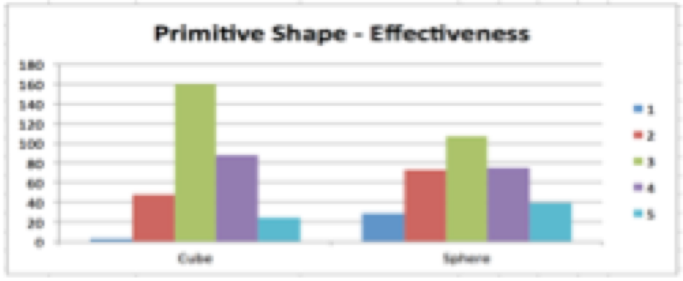
Image effectiveness followed a normal distribution from 1 to 5, where as satisfaction was evaluated lower with 100% more 2 ratings than effectiveness. This may indicate that the chosen colour combinations and backgrounds did not appeal naturally to the observer. Suggestions to cluster the colour to reduce the mixed pattern would also indicate that this was the case.



2) Shape selection

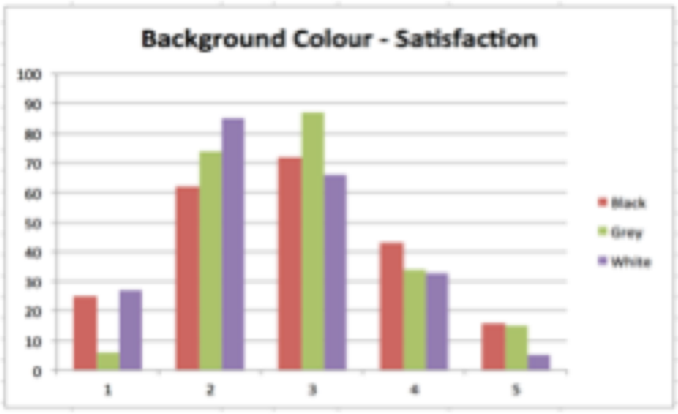
Spheres where preferred over cubes. This may be due to two factors: 1. The sphere edges were gradient to black thus enhancing their appearance, particularly on darker backgrounds, 2. Each sphere has a small separation due to their shape, giving an ability of the user to more easily distinguish individual objects. Cubes had no gaps between them and instead appeared as a solid block.

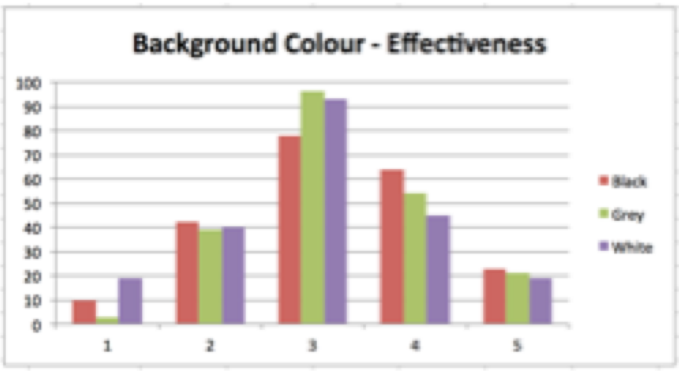




3) Background colour

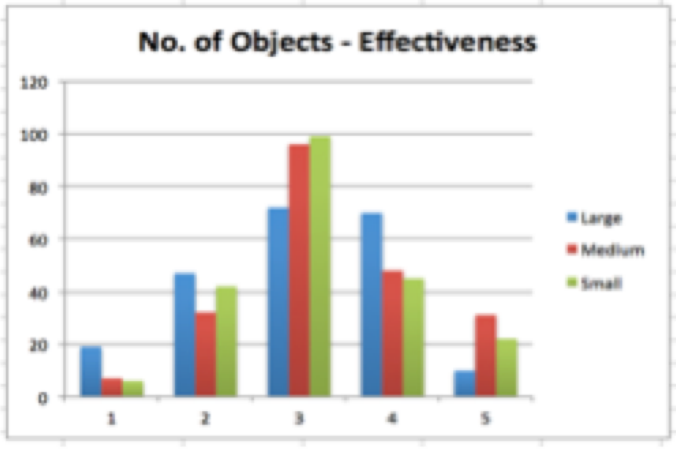
The rendering of spheres had gradient dark edges which greatly enhanced their look on black or grey backgrounds, but created too much contrast against a white background.





4) File set size

Larger file set (1200 objects) made many images difficult to distinguish individual colours given the reduced object scale and apparent colour blending.



B. Performance Test Analysis

There was a clear relationship between number of objects and rendering time, to the point that very large numbers of objects cause a general error in the application. The scene building algorithms were not optimized and would need to be refactored if larger file sets were to be rendered with the selected primitives.

VI. Conclusions

A. *Project achievements*

The motivation for this project was to investigate creating visualation tools to help make sense of software changes in a timely manner to assist decisions making. The initial views created go a long way to providing an effective overview visualisation for change across a large file set over many points in time. Initial work on data set reduction included creating a semantic mapping by means of the component hierarchy, and modifying objects in the scenegraph accordingly. The performance testing indicated large number of objects need optimal rendering. Since the eventual use of 3D is intended for a reduced data set this is not seen as an obstacle to further work.

Usability testing indicated the visualsations are effective but may need refined colour definitions.

B. *Future work*

- The 3D views explored did not realize node-link graph edges.
- Reducing the initial overview data set to which ever rate of change is of interest requires brushing or object selection in some way.
- This may be combined with change clustering based on change type as well as semantic clustering such as the component hierarchy implemented in this project.
- Seamless transition from overview to selected reduced data set needs to be designed.
- User Interface cues to encourage completion of the required visual queries for a specific decision task need to be defined.

References

[1] S. K. Card, J. MacKinlay and B. Shneiderman, ***Readings in Information Visualization: Using Vision to Think***, 1999,

[2] J. Pelrine. Is software development complex?[Online]. http://cognitive-edge.com/blog/entry/4597/is-software-development-complex/ [accessed: August, 2013]

[3] G. Rzevski. Modelling large complex systems using multi-agent technology. Presented at Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS International Conference On.

[4] H. Daniilidis, W. Bauer, K. Eben and U. Lindemann. Systematic goal definition for complexity management projects. Presented at Systems Conference (SysCon), 2012 IEEE International.

[5] N. Whelan, "Visualisation Tools for Decision Support in Complex Software Project, Final Report, Appendix A."

[6] Xin Dong, Qiu-Song Yang, Qing Wang, Jian Zhai and G. Ruhe. Value-risk trade-off analysis for iteration planning in extreme programming. Presented at Software Engineering Conference (APSEC), 2011 18th Asia Pacific.

[7] T. Hanratty, R. J. Hammeil II, J. Yen, M. McNeese, S. Oh, H. -. Kim, D. Minotra, L. Strater, H. Cuevas and D. Colombo. Knowledge visualization to enhance human-agent situation awareness within a computational recognition-primed decision system. Presented at Proceedings - IEEE Military Communications Conference MILCOM.

[8] E. Koyuncu, U. Ozdemir, G. Tokadli, Z. Bahcivan and G. Inalhan. Design of a pilot-centered visual decision-support system for airborne collision avoidance. Presented at Digital Avionics Systems Conference (DASC), 2012 IEEE/AIAA 31st.

[9] M. M. Marefat, A. F. Varecka and J. Yost. (1997 An intelligent visualization agent for simulation-based decision support. *Computational Science & Engineering, IEEE 4(3),* pp. 72-82.

[10] S. Few. **Criteria for Evaluating Visual EDA Tools** [Online]. http://www.perceptualedge.com/library.php#Articles [accessed: 02/17/2013]

[11] C. Ware., *Information Visualization: Perception for Design,* ,3rd ed., vol. 1, MA 02451, USA: Morgan Kaufmann, 2013, pp. 512.

[12] B. Johnson and B. Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. Presented at Visualization, 1991. Visualization '91, Proceedings., IEEE Conference On.

[13] Xia Li and M. -. Kraak. A temporal visualization concept: A new theoretical analytical approach for the visualization of multivariable spatio-temporal data. Presented at Geoinformatics, 2010 18th International Conference On.

[14] E. Kalogerakis, S. Christodoulakis and N. Moumoutzis. Coupling ontologies with graphics content for knowledge driven visualization. Presented at Virtual Reality Conference, 2006.

[15] Qt Digia [Online]. http://qt.digia.com/ [accessed: August, 2013]

[16] Qt3D Reference Documentation [Online]. http://doc-snapshot.qt-project.org/qt3d-1.0/index.html [accessed: August, 2013]

[17] N. Whelan, "Visualisation Tools for Decision Support in Complex Software Project, Final Report, Appendix B."

[18] L. Xia. New Methods of Visualization of Multivariable Spatio-temporal Data: PCP-Time-Cube and Mulitvaiable-Time-Cube [Online]. www.itc.nl/library/papers_2005/msc/gfm/xia.pdf [accessed: August, 2013]

[19] N. Whelan, "Visualisation Tools for Decision Support in Complex Software Project, Final Report, Appendix C,"

Appendix A

Visualisation Tools for Decision Support in Complex Software Projects
Literature Survey

DUBLIN CITY UNIVERSITY
SCHOOL OF ELECTRONIC ENGINEERING

Visualisation Tools for Decision Support in Complex Software Projects

Literature Survey

Niall Whelan

April 2013

MASTER OF ENGINEERING

IN

ELECTRONIC SYSTEMS

Supervised by Dr. Derek Molloy

Acknowledgements

I would like to thank my supervisor Dr. Derek Molloy for his support in my project selection and encouragement in defining a challenging scope of work.

Declaration

I hereby declare that, except where otherwise indicated, this document is entirely my own work and has not been submitted in whole or in part to any other university.

Signed: ..... Date: .....

Abstract

Agile or incremental development for SaaS / Cloud based applications creates many new challenges for software development, maintenance activities and downstream processes such as Documentation, Training, Support, and Localisation.

The aim of Information Visualisation is to use "computer-supported interactive visual representations of abstract data to amplify cognition" [1]. The Software Engineering process now more than ever requires data driven decision support and knowledge management specifically focused on content change management. This project proposes to employ well founded visualisation techniques to design a decision support toolset to harness data from live software project artefacts to accelerate daily decision making in various software project phases.

By identifying the most common and important questions being asked by stakeholders and decision makers in daily planning, tracking and review meetings I propose to define a decision tree and associated visual queries to accelerate the cognitive process by following the most important question/decision pathways.

Since visual working memory is limited, and a decision pathway may require many visual queries, designing interactive visualisation tools that engage and seamlessly guide the decision makers through the visual decision pathway is a key challenge. This project aims to employ appropriate visualisation designs to support all stages of the decision making process. It also defines a decision framework in which a number of typical industrial software project decision tasks are used to exemplify rapid decision support through the data-driven visualisation process.

Table Of Contents

Table of Figures

Chapter 1 - Introduction

1.1 State of the art Visualisation Tools for Decision Support

This chapter begins by separating the definition of visualisation used in InfoViz versus Data Analytics, outlines Visualisation usage for Decision Support, Visualisation usage in Software Engineering and briefly discusses the proposed solution to create visualisation tools for decision support in complex software projects.

1.1.1 InfoViz and Visual Data Analytics

Recent growing popularity of InfoViz applications to deal with the growing volumes of data being produced by the growing number of users generating data in ever expanding forms on more and more devices, shows the general appetite for knowledge and understanding of the information being generated. However, even though many novel visualisation techniques are being created, there is still many more areas in which information visualisation can improve to meet the needs of its users. InfoViz tends to deal with well-defined data sets and novel interactive visualisation of these. Data Analysts on the other hand often deal with more complex data and very domain specific analysis and visualisation needs. Generic tools for Data Analysis Visualisation tend to be expensive tools created for Data Analysts only and not the information consumers [2]. As a result practitioners in many domains often lack the tools to provide insight into their daily operations because of the high cost barrier to data analysis specific to their domain.

1.1.2 Data Visualisation in Software Engineering

Software Engineering is one such domain where highly complex projects with very diverse related information is produced on a daily basis usually without information management tools to consume all the data in a meaningful way to allow all facets of a software project to be measured and tracked effectively. Project Managers face a huge daily challenge in knowing if a project is on track, or needs immediate corrective action to meet it's many, often competing goals. To further exacerbate the problem, agile development with globally disperse teams and tighter release deadlines and release criteria elevate the problem to the realm of management decision support [3] where many dependant stakeholders have a vested interest in the daily updates on project progress, risks, and mitigation plans. Tools needed to manage all these demands are not 'off the shelf' tools, but domain specific yet

interdisciplinary. Traditional Software tools for Project Planning and Tracking, Defect Management, CRM, Requirements Planning, etc. do not provide an effective decision support platform for software project management.

1.1.3 Visualisation Tools for Decision Support

Decision Support Systems (DSS) are used in countless diverse human activities from industrial, governmental, medical, military, and emergency response. Like Visualisation systems, DSS have been developed in many cases on a domain specific basis. Decision support does not necessarily require visualisations, and many good decisions have been made by inspecting numbers on a spreadsheet. However, as has been discussed already, visualisations do reduce the cognitive load on the user, and in the case of someone trying to make an informed decision from a large multivariate data set using appropriate visualisations can and has been shown to improve speed and effectiveness of decision making.

Complex real-time DSS heavily rely on visualisations to give rapid situational awareness to decision makers. Hanratty et al [4] show a 20% improvement in military simulation experiments where human decision makers are relying on remote human or agent information from a potentially hostile remote environment. This work relies on a well defined, remote decision space visualisation to give an instant overview of the remote environment. Another example of critical use of visual decision support is in an Aircraft flight deck. In [5] Koyuncu et. al. present a conceptual design of the pilot-centred visual decision support system for flight critical collision avoidance using "dynamic 4D trajectory management". The motivation is to enable enhanced situational awareness in the flight deck, and also reduce the cognitive load on the controller by delegating flight separation responsibility to appropriately equipped flight decks. The system proposes to a) provide the pilots with alternative flight paths via visual 'tunnels-in-the-sky' on various displays in real-time, b) provide the flight crew with quantified and visual understanding of collision risks in terms of time, directions, and countermeasures, and c) provide autonomous conflict resolution as an autopilot mode.

In [6] Marefat et. al. introduce an intelligent visualisation agent for simulation based decision support. This approach is useful as it allow the underlying data types determine the appropriate visualisations which are possible. It also introduces semantic mapping and

combines data types and data semantics to drive the visualisation process thus removing some of the cognitive load on the user for reaching appropriate visualisation quickly.

1.2 Solution approach

1.2.1 Let the data do the talking

This project proposes to define a decision framework for specific software engineering activities which allow daily software project data acquisition to be the input into a visualisation application to drive daily decisions and produce actionable results. By defining an appropriate sequence of visualisations in short succession the efficacy of using a directed visualisation experience to accelerate and improve decision making outcomes specific to the software engineering domain can be investigated.

The notion of content volatility is used as a metric for software component task choice. For example code files or translateable content files which change on a daily basis can be considered volatile, whilst files which have not changed in weeks are stable. Depending on the state of change a different set of tasks will be valid for any subset of software files.

By keeping the user focused on the decision making process through a sequential set of visualisations the user may refine a file set until a final set of 'work product' is achieved for which an action or process is defined. These actions can be tasks such as perform code review on volatile content; define test cases for partially stable content; start translating stable User Interface content. The file set refinement process will employ semantic data related to all content under consideration. This categorical information will enable automatic clustering or thresholding to be performed based on the users desired task decisions.

1.2.2 Don't get lost

A measure of success for this project will be if a users progression from an overview visualisation to a final decision through various visualisation transitions can be performed in a matter of seconds without loosing the mental map or being distracted from the task decision goal. Another measure of success will be in providing a measure of effectiveness or improvement of each actual decision.

1.2 Summary

This chapter provides an overview of the starte of the art in Visualisation tools with specific reference to how such tools are used in a decision support context and also in the software engineering domain. The proposed solution approach was then discussed at a high level in order to give context for the next topic in chapter two where a review to the options for visualisation techniques are discussed. This will be followed in chapter three with a discussion of the visualation evaluation techniques available.

Chapter 2 - Technical Background

Visual queries[7] form the basis of hypothesis formulation by humans in resolving a cognitive task by the presence or absence of some visual pattern during initial feature detection. When a person is reading a piece of text or a table of figures, although the task may seem to be a linguistic task, in fact the most basic task is searching for text patterns or number patterns on a page, which is in itself a visual task. The old adage 'a picture speaks a thousand words' is very relevant in data analysis and reminds us that the information content of a picture can have orders of magnitude higher content than a page of text of numbers.

The focus of this research is visualising software change in a way which allows rapid identification of changes throughout a code base, but also relating tasks to levels of software volatility to allow decision makers to reach decisions rapidly and take appropriate well defined actions. This chapter outlines the research and challenges in good visualisation design related to software artefact hierarchies and decision support problem areas. Issues related to defining a decision support framework for domain knowledge workers are then highlighted. Finally, Software Change Management task areas which will be the focus of this project are identified.

2.1 Visualisation Design

Data Science defines a seven step process to allow practitioners discover new insight from raw data. These steps can be summarised as follows:

- Acquire
- Parse
- Filter
- Mine
- Represent
- Refine
- Interact

Data mining and it's extract, transform, and load (ETL) processes are well matured processes at this point. This project interest is in the represent, refine and interact steps of the process. Therefore the visualisation design discussion will not address the preceding data acquisition steps.

2.1.1 Choosing appropriate visualisations

Choosing visualisations is wholly dependant on the application and desired outcome. There is no single off the shelf application available to provide users with all visualisation needs as highlighted by Few in[2]. For the pusposes of this project the focus is on key visualisation areas: 2D Overview Maps, Zoom+Detail, Focus+Context, and finally displaying details on reduced data sets in 3D. Employing visualisation interaction techniques enhances the user's perception of information when exploring a data set and enables the user to over come such issues as object occlusion or visual clutter [8]. In this discussion the overall task of maintaining the users goal and mental map to reach a final decision is considered.

2.1.1.1 Overview Maps

There are many variations of overview maps and their usage. Often overview and detail interfaces go hand in hand. A major disadvantage of overview and detail is the additional use of screen real estate (which may be more effectively used for details) and the mental effort and time required to integrate the distinct views. This section highlights some of these overview maps and improvements that have been added to overcome their various shortcomings.

Heatmaps

Heatmaps [9] are a useful 2D visualisation technique because their high information density allows a whole data population to be viewed at once. Walker et. al address in[10] a common problem with heatmap visualisation, that the often arbitrary ordering of rows and columns renders the heatmap unclear. They show that using spectral *seriation* to rearrange the solutions and objectives can enhance the clarity of the heatmap.

Treemaps

The Treemap introduced by Johnson and Shneiderman in[11] is an appropriate visualization for hierarchical data and could naturally be used as an overview visualization. Treemaps were developed for disk directory browsing, which had meaningful variables that could govern rectangle size (file size) and color coding (file type or age). Wattenburg enhanced the treemap in [12] by adding colour to represent real-time changing Stock Market data, and also provided alternate colour schemes for both black and white printing and Red/Green coloured blind users.

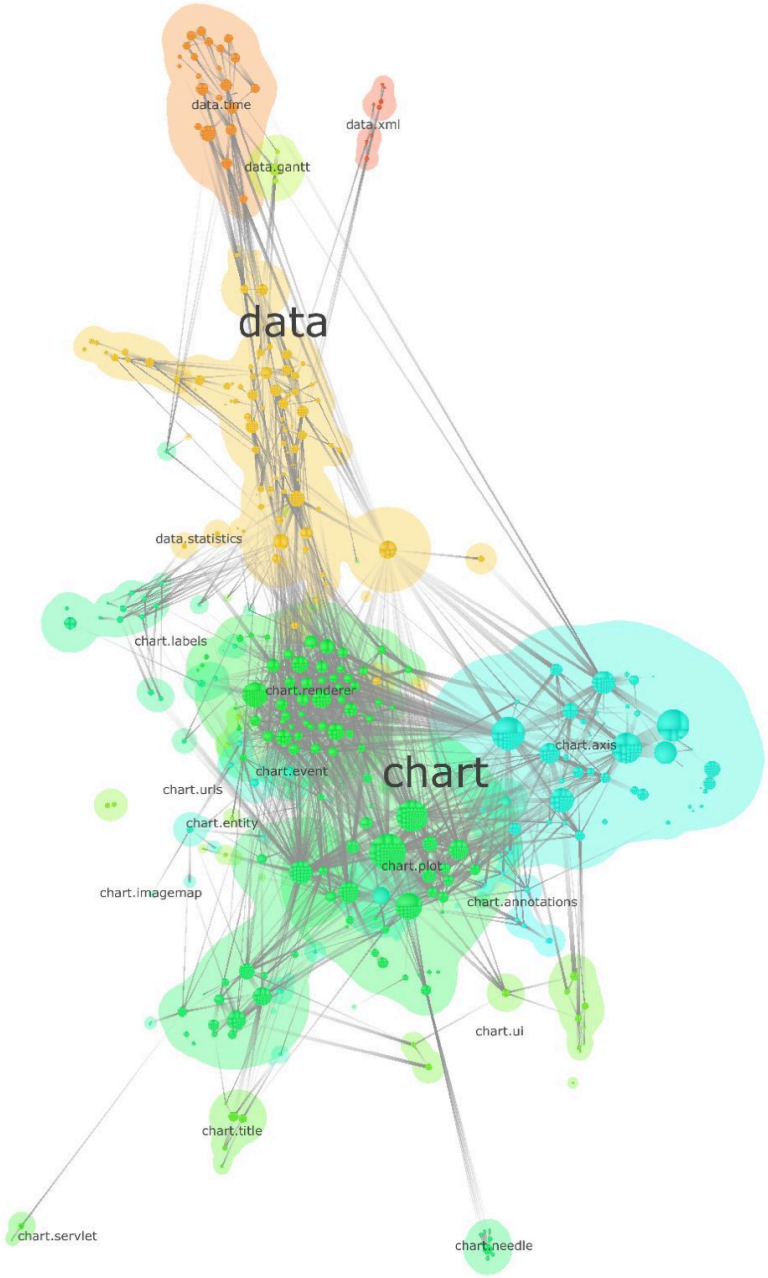


Figure 1 Call Graph between classes of two main packages in 'JFree' [18].

To explore relations between software entities in[13] Balzer et. al aggregated parts of the software system represented as treemaps that visualize the structure of the contained software entities. Rather than depicting the standard rectangle based treemap layout they an alternative method was introduced for the horizontal and vertical subdivision of convex polygons. This allowed differentiation of entity types with the hierarchical structure.

Node-link Graphs

Trees are a basic and intuitive organising structure and are often used to represent hierarchical data sets such as a file system. However, by their very nature, tree size can grow exponentially. This produces many challenges from a screen real estate point of view, or rapid or easy browsing of a large hierarchy. Reingold et. al. discuss this problem in[14]

and proposed an improved aesthetic algorithm to improve hierarchy drawing and use of available space.

Elastic Hierarchies.

In [15] Zhao et. al. propose combining the hierarchical representation advantage of Node-Link structures with the data density advantage of Treemaps by introducing an Elastic Hierarchy. Although visually intuitive, their work does not delve into the impact on cognitive load or performance comparison with other visualisations.

Self Organising Maps and Category Maps

An interesting idea of Self Organising Maps was introduced by Kohonen[16] and applied to large document database with similarity mappings as a 2D Category Map. A document dataset of over one million documents containing 245 million words in total were analysed and mapped. The processing at the time (1997) took 9 weeks. Since we are looking for near real time mapping, this approach suffers from computational limitations. Also, the larger the dataset the more visual load was placed on the fixed size 2D map space. Elastic Graphs and Metro Maps discussed by Gorban et al. [17] show the use of principal component analysis (PCA) can be employed for dimensionality reduction as needed in rendering complex graphs. Their application on 2D Metro Maps could perhaps be extended to 3D hierarchical data sets, thus open an opportunity to explore 3D Elastic Nodes within the context of my project for focus+context and detail viewing stages of the visualisation process.

Fish Eye and Fractal Views

To address some of the limitations of the Category Map, Yang et al. [18] introduce the combination of Fisheye and Fractal views. Fisheye views and fractal views support the visualization of large category maps by two different approaches. Fisheye views use the distortion approach to emphasize view on the data in focus, while fractal views employ information reduction with a scale factor controlled by the user to limit the data set. The performance comparison of both techniques concludes distortion has a negative impact on user experience, whilst fractal views do not distract the user but increase focus. Since our interest involves keeping a clear mental map through multiple visualisations, fractal views appear to be a more favourable approach than fisheye views for overview 2D maps.

Zoom + Filter, without lossing mental map

Temporal separation of visualisations can negatively impact cognitive load for users in assimilating the relationship between pre and post zoom states. Animating the transition between zoom levels can dramatically reduce the cognitive load. There is also evidence that users can optimize their performance with zooming interfaces when concurrent controls for pan and zoom are supported.

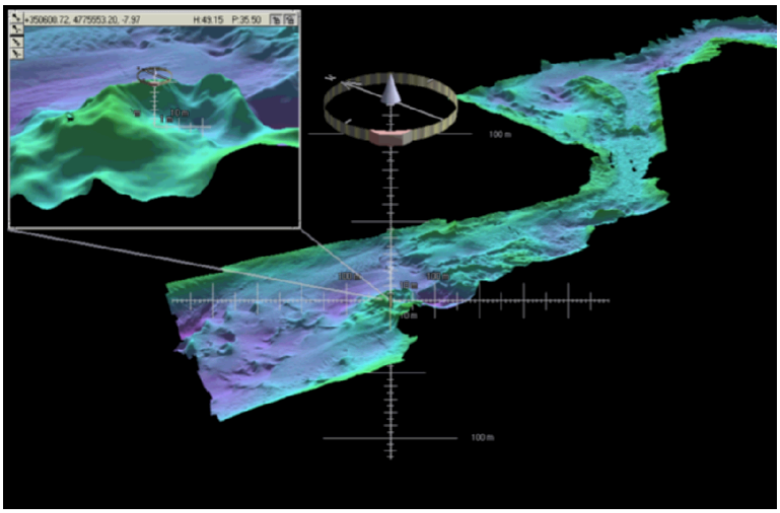
In[7] p.408-412, Ware discusses the options for zooming and when it is best to use mutlitple windows in order to maintain the mental map and global view whilst focusing on detail. Ware shows there is an optimum number of three visual chunks which can be held in visual working memory before performance of mulitple windows begins to outperform a single zooming interface. The term chunk is taken from cognitive psychology where Anderson and Milson [19] show chunks of information are constantly being prioritised and reorganised in the brain based on current cognitive needs.

Visual spacial redunancy

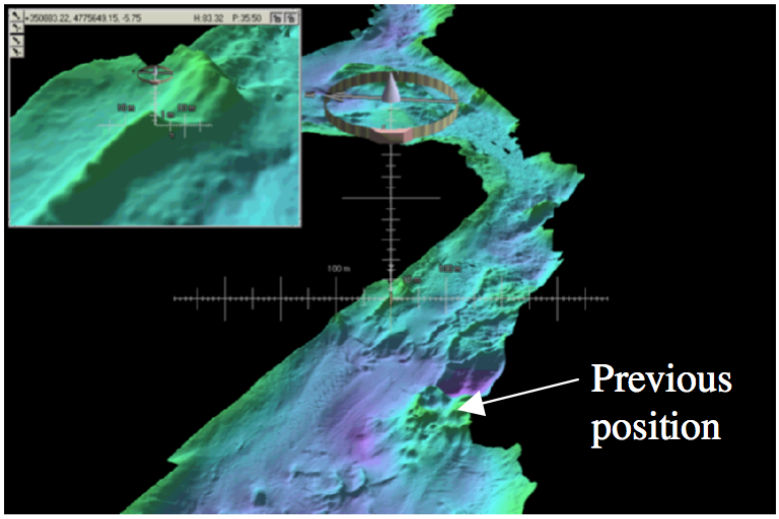
A basic bar chart can be a powerful way to make a list of number instantly understandable in a visual form. However, a bar chart typically has relatively few data points required compared to the number of pixels of screen space it occupies. This is a an example of spatial reducnancy and is an important consideration in designing visualisation containing large data setw and multiple dimensions, particularly when trying to provide overview visualisations. Sparklines discussed by Tufte in [20] show a simple way to reduce the space needs of a line graph to a simple graphic word concept which is much more spatially efficient.

2.1.1.3 Focus + Context

When navigating a 3D space it is easy to zoom into detail (focus) and effectively loose your way in relation to the overall landscape (context) [7] p366-373. When analysing a data set it is often important to seek patterns in detail and across a spatial scale concurrently. The scale may not just be spatial but also temporal or structural depending on the data. A number of techniques have been designed to over come this problem: Distortion, used in Hyperbolic trees and Fisheye Views; Rapid Zooming; Hidden Structure; Multiple Simultaneous Views.



(a)



(b)

Figure 2 Zoomports coupled in a magnified view arrangement. Any movement in one is matched in the other. [7]

As this project will be looking at volume of content in various states of volatility the notion of distorting the overview map as in a hyperbolic tree does not appeal. Instead the latter three techniques will have a role in detail viewing in a 3D space.

2.1.1.4 Displaying details on reduced data sets

It may be possible to visually depict all levels of detail needed to traverse a decision pathway in 2D, but given the complex relations in software systems it is perhaps useful to also explore detail views of these relations in a 3D setting. Intuitively, creating a 3D hierachical view of the problem space may be difficult for overview maps as data density is likely to mean each object has only 1 pixel in which to render it's 3D reprentation. Level of Detail (LOD) techniques could clearly be used to cluster neghbouring nodes in a hierarchy but then the volatility metric of the overview map is being compromised as neighbouring nodes may have opposing volatillty values.

2.1.3 Maintaining the mental map and exploiting Visual Working Memory effectively

In [7] Ware outlines the main properties of Visual Working Memory (VWM) which is the limiting factor on cognitive load as follows:

- VWM has a small limit (3 to 5) of visual objects or patterns it can retain concurrently.
- VWM exhibits a rough visual spatial map that can hold residual information on recently attended objects.
- A persons current focus of attention determines what visual information is held in VWM.
- Focus of attention can change as quickly as 100ms, yet the sematic meaning or gist of the visual object or scene can be acquired in 100ms.
- A visual query pattern can be held in VWM to begin an active search based on the focus of attention.

2.1.4 Data Clustering appropraite to task execution decisions

At different levels of details in the visualisation process there is a need to order data in some meaningful way for the application. For example in my solution I will want an overview map which can reveal perhaps three levels of variation in volatility across a large data set as an initial clustering strategy. In [21] Melcher et al. show a clustering approach in a heatmap for software processes. The hierarchical structure is indicated as well as the process metrics above and below the image respectively. This visualisation is useful for many application but may fall short when the hierarchy has multuple realtions per node. Since I am interested in following a decision pathway based on one value of volatility at a time, thresholding such an image to the desired value would free up space to allow a complex hierarchy (such as software components) to be perhaps explored in other ways.

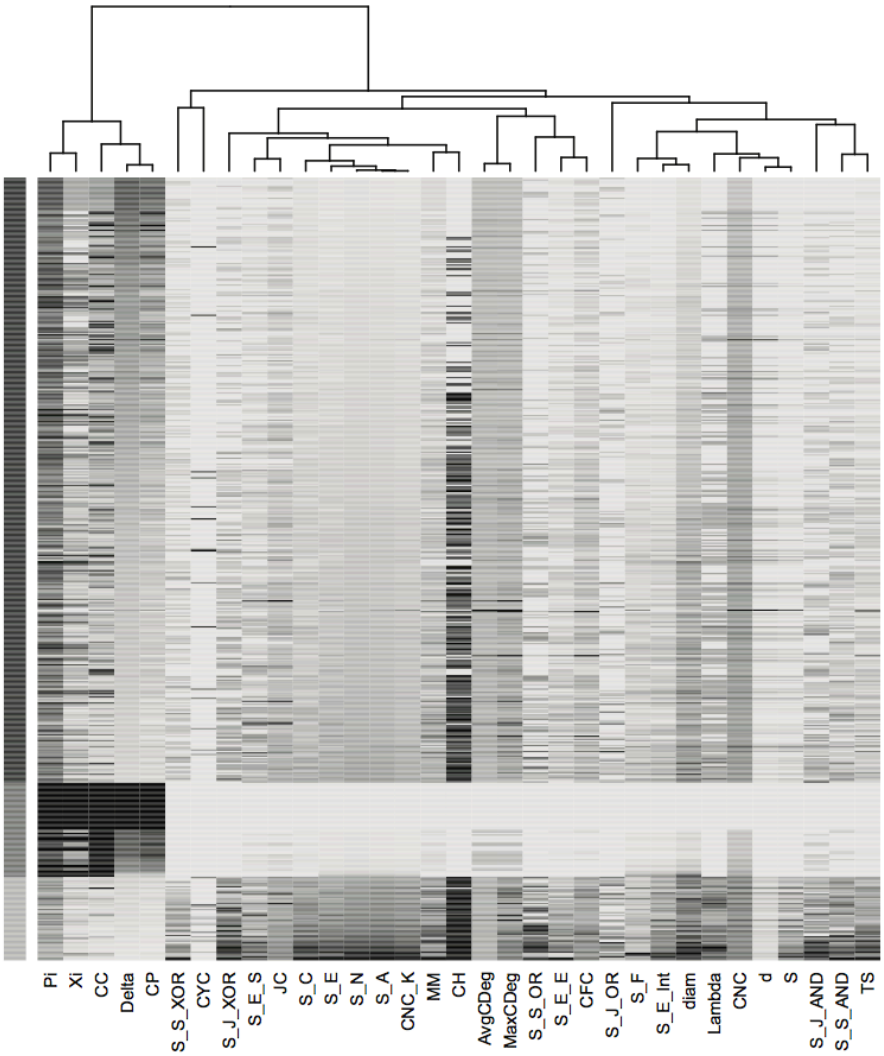


Figure 3 Clustered Heatmap showing 33 process metric values for 515 processes. The rows are separated in three clusters (see column with grey scale values on left side) of figure. The columns are hierarchically clustered using the Spearman's rank correlation coefficient as a distance measure [21].

2.2 Defining Decision Support Framework

2.2.1 Using Decision Grids to decompose the decision making process.

In [22]Wang introduces a decision and operation theory known as the decision grid for modeling and supporting dynamic and serial decision-making. The mathematical model and properties of the decision grid allow efficiency to be empirically calculated. Decision grids can be applied in a wide range of serial and dynamic decision making situations and could provide the basis for decision framework and feedback process for any DSS system.

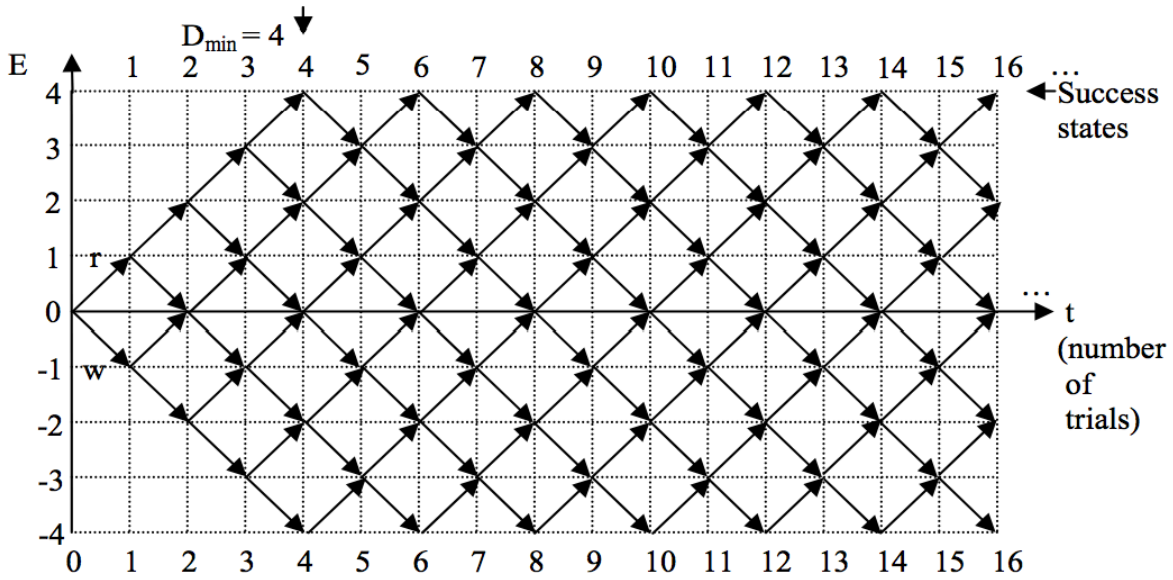


Figure 4 A decision Grid with Dmin = 4. [22]

A formal model of a decision grid DG is a 4-tuple, i.e.:

DG = (T, D, E, S) (2)

Where

• T is a finite or infinite set of trials $T = \{t_1, t_2, ..., t_n\}$, and n is the number of trials where n may be infinite.

• D is the decision distance of a series of trials, $D=t_i - t_0 = t_i, D_{min} \le i \le n$, where D_{min} is the minimum distance towards the success state.

• E is the effort of a specific trial series towards the success state in the grid, $D_{min} \le E \le n$.

• S is a finite or infinite set of success states of the grid, $S=\{s_k | D_{min} \le k \le n\}$.

By modelling a decision framework on this grid each grid location or decision point will become a specific visualisation in the visual decision pathway. So based on the defined number of decision points in the grid, that same number of appropriate visualisations are needed to encompass the entire decision space. This model also raises the question of what happens is you do not achieve Dmin; what king of visualisations should be shown if based on past visualisations the user is arriving at the decision endpoint much later? Should a decision feedback proactively compensate in some way to ensure subsequent visualisation get the user back on track?

2.3 Defining the software change management tasks

In [23] Rilling describes a process centered visualization tool to help software maintenance process owners improve effectiveness in managing the many maintenance tasks related to on-going software improvement. The system employs a formal process ontology and an automated reasoning system. Although this research covers some of the challenges to be addressed, the actual visualisations appear cluttered and unintuitive for an uninitiated user perspective. However, the need for ontologies to automatically add meaning to large data sets and many artifact types is evident.

Another approach to ontologies in 3D space [24] is presented by Kalogerakis et al. where a generalized model and methodology to incorporate domain knowledge into 3D scenes in web-based environments is outlined. The methodology uses W3C Ontology Web Language (OWL) and raises the description of graphics and virtual reality content to the ontologies layer of the semantic web providing the ability to manipulate and query the scenes.

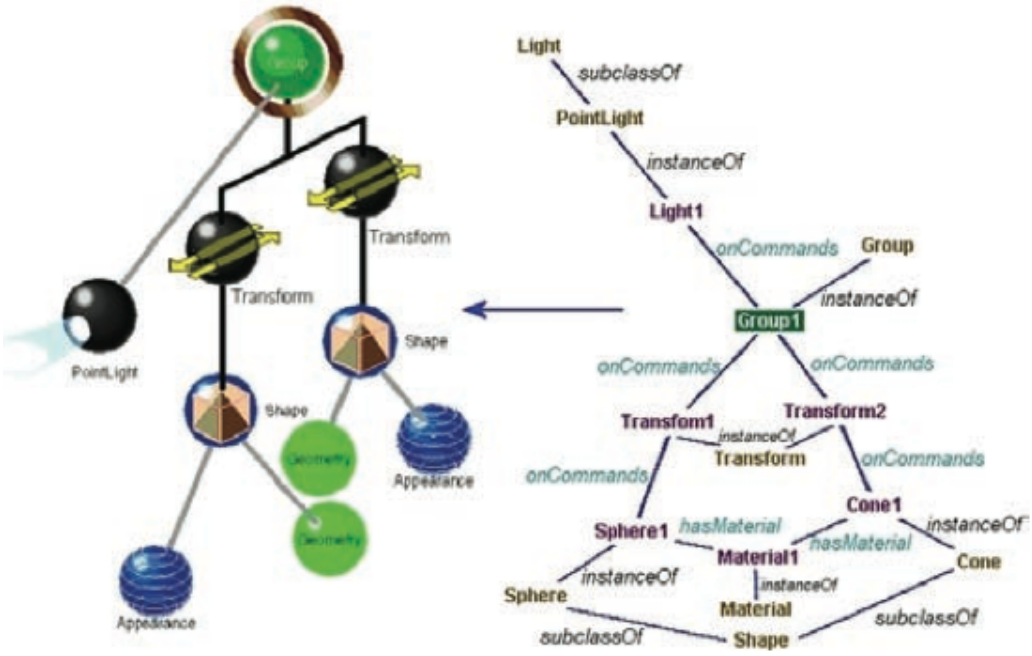


Figure 5 A triple based OWL graph describing a scene graph. [24]

Defining the set of software project artefact attributes which need to be used within the visualisation process semantic map will directly depend on which software project tasks are to be included and how those task execution decisions are made.

2.4 Summary

This chapter provides an outline of the main visualisation options and challenges to be faced in providing visual tools for decision support in complex software projects. The following chapter will discuss ways to evaluate the effectiveness of a visualisation system design.

Chapter 3 – Visualisation Evaluation

In his book [7] Ware gives an overview of some of the empirical research methods that can be applied to evaluating visualisations. Because of the nature of visualisation, many of the evalusation techniques are borrowed from displiplines such as psychophysics or cognitive psychology. In this chapter I will summarise some of these evaluation methods which can be used in evaluating visualisations.

3.1 Psychophysics

Psycophysics is a set of methods based on physics to measure human sensation, for example how a person will perceive flickering light or steady light depending on the speed of the light pulse. Psychophysics is also applied to fundamental sensory properties such as colour or visual texture detection. A recent extension of this disipline is into information psychophysics which applies classical methods to information structures such as flow patterns, surface shape or paths in graphs.

3.1.1 Measurements

For measurements of graphics objects the possible units are pixels or pixel size, centimetres, visual angle. For colour measrurement it is important to use colour calibrated monitors using a standard such as CIE XYZ. For movement, monitor refresh rate in Hz and graphics update rate on screen need to be known in order to known the lower bound measurement time.

3.1.2 Detection Methods

Error rates are used to identify when a user makes a mistake performing some task. In visualisation evaluation error rates can be used to discovery a users sensitivity threshold is reached for some visual feature or system. Being able to find this threshold quickly and efficiently in investigated in [25] by Wetherill et al. where in the ‘staircase procedure’ is introduced for speeding up threshold detection using error rates. Each time a user successfully performs a task, the sensitivity of the system is lowered until the user fails at task, when the sensitivity is raised. This process quickly hones in on the optimal sensitivity for a given user.

Another threshold detection method borrowed from signal detection theory using the receiver operating characteristics (ROC) curve [26] to present results. In this model a target signal pattern is assumed to produce a neutral signal with a normal distribution in the presence of noise. In this model it is possible due to noise to obtain false positive or false negative results. Using the ROC curve the observations optimal outcome will be just underneath the curve, and allow modelling of sensitivity vs. specificity of a system.

3.1.3 Method of Adjustment

This technique can be used for tuning a visualisation system based on individual user perception. By providing users with some parameter to adjust the behaviour of the visualsiation, different users may find best performance for themselves at different paramter settings. This also leads on the notion of calibrating or baslining user results. For example Garaas and Pomplum show in [27] that inspection time (IT) is not equal for all users. Therefore some base-lining of users employed for visualisation usability studies needs to be employed.

Their study highlights that 20% of the variance in individual mental capcaity can be attributed to inspection time. In their study with 34 participants performing a set of visual search tasks, two groups emerged with low IT and high IT. The low-IT group had ITs that ranged from 33.3 ms to 73.9 ms with a mean of 62.0 ms and a standard deviation of 9.0ms. The high-IT group had ITs that ranged from 81.9 ms to 158.3 ms with a mean of 98.0 ms and a standard deviation of 21.4 ms. If such a base-lining process is not used it is clear from these results that the comparison of two marginally different visualisations may be entirely misrepresentative of the overall performance of one system over another.

3.2 Cognitive Physcology

In this discipline the brain is modelled as a set of linked processing areas, for example short term and long term memory areas. This approach uses measurement of reaction time or measuring errors to test a specific cognitive task model. For a Decision support visualisation tool this method can be used for example to measure the timing of users to execute a series of visualisation-decision tasks and the error rate in reaching a desired or known outcome. In fact the research into properties of visual working memory which show the speed at which our minds can acquire the gist of an image and the fact that we can only

retain three to five objects motivates the proposed solution to break down the decision making tasks into a series of visualisations to avoid overloading our working memory and loosing our focus on the decision goal.

3.2.1 Structural Analysis

Structuralism is a cognitive psychology approach which gives emphasis to formulating hypothesis and then testing users against each hypothesis using testbench applications, semi-structural interviews and rating scales.

A testbench application is built to allow a research experiment with variations of parameters of a system to discovery best settings, unexpected properties. For example, to test colour mapping for a heatmap or pixel map it would be useful to be able to swap palettes easily and use black and white, grey scales, or colour blind friendly palettes when running tests.

Structured interviews assist in initial understanding of a problem and it requirements, as well as individual task analysis. By ensuring the right questions are being asked the structured interviews ensure all user requirements are being understood. Evaluation of system performance can also use structured interviews to solicit feedback consistently for a number of users and without significant time investment.

Finally rating scales can be used to give an overall numerical result for relative preferences of users. If a scale gives ratings for Good or Bad on a scale like: (GOOD) 1, 2, 3, 4, 5: (BAD), then even though different users may give different numeric answers, they will tend to rate good and bad based on their observations. A set of many user observations can be normalised to give the overall preference for one system over another without worrying about the fact that some users rated all good systems as 2 while others rated good systems as 1.

3.2.2 Statistical Exploration

Often the statistical analysis may be used to classify data into user expected or understood ranges of classification. For example asking a set of users to define the set of textures they know they all may come up either different descriptions or sets, but the union of their responses gives a basic user categorisation of textures. The problem then is given a data set, how do you ensure all elements fall neatly into the various categories. Perhaps there are more categories not mentioned by users and the number of dimensions is considerably

larger than originally thought. Applying statistical methods to the data can reveal it's underlying properties.

Principal Component Analysis (PCA) can be used to transform a set of variables from one domain to another where all variables are uncorrelated using eigenvalue decomposition. This can help reduce a high dimension data set to a lower order data set as the majority of the information within the data is stored with a relative low number of eigenvectors.

Clustering analysis can be performed in any dimensional space to find clusters of values for categorisation. The main forms are hierarchical and k-means. The hierarchical method recursive processes the data point to produce a tree structure that can be useful to build a hierarchical taxonomy such as a software component structure. K-mean requires a user defined set of clusters that the algorithm then mapped all data points to using a sum of least squares distance metric between points. This method may prove useful in this project if LOD technique is being applied in the 3D space where actual LOD classification could be measured against the k-means calculation for a given set of data.

3.2.3 User Study issues

Experimenter Bias

Researchers face the challenge of objectively evaluating their own work in their treatment of results and comparison to similar solutions proposed by other. Some important questions to answer in evaluating this work include:

- Is the task clearly defined?
- Has the experiement actually addressed the problem?
- Are there appropriate control conditions in place?
- Has the stated hypothese actually been tested?
- How significant are the results?
- Are there outside variables influencing the claimed results?

Number of test subjects to use?

For a specif domain problem it may be difficult to identify a statistically sufficient number of users to test a system. The more variable the user outcomes the more users are needed to statistically model the results. By breaking down the problem space into domain specific and more fundamental visualisation tests is may be possible to evaluate parts of a system for larger or smaller pools of valid test users.

Task Identification

Clearly identifying the task a user is being measured on is important when designing an experiment. For example, in a software engineering visualisation, it is not just the decision goal task which needs to to be measured. If the visualisation of the software components includes a 3D node link structure, the users ability to identify links between nodes is a specific test which can be used for example to model the best colour or shading setting or if occlusion is a problem.

Controls

A new visualisation technique needs to be compared to some existing process, an existing visualisation or a process without a visualisation employed. In order for the control to be valid it needs to be un-biased and best on ideally based on best practise current process.

Chapter 4 - Conclusions and Project Outline

In this final chapter some conclusions are drawn about the usefulness or otherwise of the areas researched in this survey as they relate to the problem. Based on these finding an outline solution proposal is described below.

4.1 Conclusion

Growing research in visualisation for large data sets shows many solutions to many problems can be derived in a 2D or a 3D space. The choice of visualisation depends on the application. Real-time Decision support puts it's own constraints of visualisations and demands clear separation between decision choices and the underlying classification of data. A formal decision framework is an important construct to drive multistage decisions and incorporate some level of feedback for decision learning. Overview visualisations need to take into account two competing facts. Complex hierarchies are challenging to design in 2D without some level of dimension reduction; decision support needs clear representation of maximum data under consideration.

4.2 Solution overview

- Combining a heatmap or pixel map with fractal view overview from a planar view of a 3D node-link graph.
    - The concept of a heatmap appeals for full data set overview, but the rigid strucure does not. Pixel maps and thresholding with fractal views will be explored to investigate the effectiveness for initial gist of overall software volatility and also to what level of detail can be manipulated with this view before the next visualisation technique is required in the decision making process.
- Combining Zoom and Focus through animation of selected data subset in 2D.
- **Exploring 3D Elastic nodes**
    - One way to explore the software hierarchy in 3D maybe using a node-link hierarchy. It may be possible to use an initial single plane view of the 3D space as the initial overview map. Once volatility thresholding has occurred the user can zoom and explore the next stage in the the decision pathway in an orthographic projection view. Zoom rate would need to be set carefully so this view transition from overview to detail and 2D to 3D is performed without loosing the mental focus on the next decision step.
- Detail view by energizing the subset 3D node-link graph and adding semantic visual cues.
- Driving decision process by adding decision cues to each visualisation.

Implementation details

- Using C++ UI framework called QT and OpenGL for 2D and 3D visualisations.
- Investigate Elastic Nodes in 3D model as planar view for initial overview map thereby supporting smooth direct transition from 2D to 3D view.
- Focusing in a 3D view through animation from planar view to orthonormal view
- Adding Decision Cues

Solution validation and evaluation

- Baselining user visualisation testers
- Measuring decision task timings without prosposed solution
- Measuring decision task timings with solution
- User Training impact on solution user test results
- Performance tests for system usability

References

[1] S. K. Card, J. MacKinlay and B. Shneiderman, ***Readings in Information Visualization: Using Vision to Think,*** 1999,

[2] S. Few. **Criteria for Evaluating Visual EDA Tools** [Online]. http://www.perceptualedge.com/library.php#Articles [accessed: 02/17/2013]

[3] Xin Dong, Qiu-Song Yang, Qing Wang, Jian Zhai and G. Ruhe. Value-risk trade-off analysis for iteration planning in extreme programming. Presented at Software Engineering Conference (APSEC), 2011 18th Asia Pacific.

[4] T. Hanratty, R. J. Hammeil II, J. Yen, M. McNeese, S. Oh, H. -. Kim, D. Minotra, L. Strater, H. Cuevas and D. Colombo. Knowledge visualization to enhance human-agent situation awareness within a computational recognition-primed decision system. Presented at Proceedings - IEEE Military Communications Conference MILCOM.

[5] E. Koyuncu, U. Ozdemir, G. Tokadli, Z. Bahcivan and G. Inalhan. Design of a pilot-centered visual decision-support system for airborne collision avoidance. Presented at Digital Avionics Systems Conference (DASC), 2012 IEEE/AIAA 31st.

[6] M. M. Marefat, A. F. Varecka and J. Yost. (1997 An intelligent visualization agent for simulation-based decision support. *Computational Science & Engineering, IEEE 4(3),* pp. 72-82.

[7] C. Ware., *Information Visualization: Perception for Design,* ,3rd ed., vol. 1, MA 02451, USA: Morgan Kaufmann, 2013, pp. 512.

[8] M. C. F. de Oliveira and H. Levkowitz. (2003 From visual data exploration to visual data mining: A survey. *Visualization and Computer Graphics, IEEE Transactions On 9(3),* pp. 378-394.

[9] A. Pryke, S. Mostaghim and A. Nazemi. (2007, 5 March 2007 through 8 March 2007). Heatmap visualization of population based multi objective algorithms. *Lect. Notes Comput. Sci. 4403 LNCS*pp. 361-375.

[10] D. J. Walker, R. M. Everson and J. E. Fieldsend. (2012 Visualising mutually non-dominating solution sets in many-objective optimisation. *Evolutionary Computation, IEEE Transactions On PP(99),* pp. 1-1.

[11] B. Johnson and B. Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. Presented at Visualization, 1991. Visualization '91, Proceedings., IEEE Conference On.

[12] M. Wattenberg. Visualizing the stock market. Presented at Conference on Human Factors in Computing Systems, CHI EA 1999.

[13] M. Balzer and O. Deussen. Exploring relations within software systems using treemap enhanced hierarchical graphs. Presented at Visualizing Software for Understanding and Analysis, 2005. VISSOFT 2005. 3rd IEEE International Workshop On.

[14] E. M. Reingold and J. S. Tilford. (1981 Tidier drawings of trees. *Software Engineering, IEEE Transactions On SE-7(2),* pp. 223-228.

[15] Shengdong Zhao, M. J. McGuffin and M. H. Chignell. Elastic hierarchies: Combining treemaps and node-link diagrams. Presented at Information Visualization, 2005. INFOVIS 2005. IEEE Symposium On.

[16] T. Kohonen, et al. (1996 Engineering applications of the self-organizing map. *Proceedings of the IEEE 84(10),* pp. 1358-1384.

[17] A. N. Gorban, N. R. Sumner and A. Y. Zinovyev. Branching principal components: Elastic graphs, topological grammars and metro maps. Presented at Neural Networks, 2007. IJCNN 2007. International Joint Conference On.

[18] C. C. Yang, Hsinchun Chen and K. Hong. Internet browsing: Visualizing category map by fisheye and fractal views. Presented at Information Technology: Coding and Computing, 2002. Proceedings. International Conference On.

[19] J. R. Anderson and R. Milson. (1989 Human memory: An adaptive perspective. *Psychol. Rev. 96(4),* pp. 703-719.

[20] E. R. Tufte., *Beautiful Evidence,* Graphics Press, 2006,

[21] J. Melcher and D. Seese. Visualization and clustering of business process collections based on process metric values. Presented at Symbolic and Numeric Algorithms for Scientific Computing, 2008. SYNASC '08. 10th International Symposium On.

[22] Y. Wang. A novel decision grid theory for dynamic decision making. Presented at Fourth IEEE Conference on Cognitive Informatics 2005, ICCI 2005.

[23] J. Rilling, Wen Jun Meng, Fuzhi Chen and P. Charland. Software visualization - A process perspective. Presented at Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop On.

[24] E. Kalogerakis, S. Christodoulakis and N. Moumoutzis. Coupling ontologies with graphics content for knowledge driven visualization. Presented at Virtual Reality Conference, 2006.

[25] G. WETHERILL and H. LEVITT. (1965 Sequential estimation of points on a psychometric function. *Br. J. Math. Stat. Psychol. 18(1),* pp. 1-10.

[26] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters 27 (2006) 861–874,* 2005.

[27] T. Garaas and M. Pomplun. The visual implications of inspection time. Presented at Cognitive Informatics, 6th IEEE International Conference On.

Appendix B

Project Source code. C-1

Main.CPP

```
#include <QApplication>
#include "mainwindow.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    MainWindow main;

    main.show();

    return app.exec();
}
```

Mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QDebug>
#include "cubeview.h"
#include <QFile>
#include <QFileDialog>
#include "qglbuilder.h"
#include <qmath.h>
#include <visualtools.h>
#include <QScreen>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    numFiles = 0;
    numComponents = 0;
    totalFilechanges = 0;
    counter = 0;
    percent = 0;
    nodeWeight = 6.0;
    defaultColour = QColor(120, 120, 120);

    ui->setupUi(this);
    this->move(800,660);
    ui->progressBar->hide();
    ui->Loading->hide();


    view = new CubeView();
    view->setVisualDataModel( VisualData );
    qDebug() << "View Geometry: " << view->frameGeometry();
    on_pushButton_3_clicked();

    tools = new VisualTools();
    tools->setGeometry(1000, 10, 200, 400);
    tools->hide();
    tools->setView( view );  // Pass handle to view to tools dialog
for updates.
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::addDynamicColour()
{
    // override default colour setting to account for file changes.

    foreach ( VisualDataModel *entry, VisualData )
    {
        if ( entry->getTotalChanges() != 0 )
            entry->setFinalColour(entry->getDynamicColour());
    }
}

void MainWindow::resetParameters()
```

```
{
    numFiles = 0;
    numComponents = 0;
    totalFilechanges = 0;
    counter = 0;
    percent = 0;
    tools->resetDepthButton( false );
    tools->resetToCubeButton();
}

void MainWindow::on_loadSource_clicked()
{
    QString fileName = QFileDialog::getOpenFileName(this, tr("Open
Source History"),
                                                    ".",tr("Comma
Separated files (*.csv);;Other Files (*.*)"));
    if (!fileName.isEmpty())
    {
        resetParameters();
        loadFile(fileName);
    }

    // Once file is loaded into SourceData, we can create a
VisualDataModel (of normalised data) quickly.

    if ( !SourceData.isEmpty() )
    {
        // foreach SourceData item,
        qDebug() << "Loading Source data..." << fileName;

        VisualData.clear();  //  Clear our model for each load.

        foreach ( SourceHistory source, SourceData )
        {
            VisualDataModel *nominalData = new VisualDataModel();
            nominalData->setComponentName( source.getComponentName()
);
            nominalData->setComponentPath( source.getComponentPath()
);
            nominalData->setFileName( source.getFileName() );
            nominalData->setFilePath( source.getPath() );
            nominalData->setBuildDate( source.getLabelDate());

            // if not in VisualDataModel, add it.

            if ( VisualData.size() == 0 )
                VisualData.append( nominalData );
            else
            {
                bool entered = false;
                foreach ( VisualDataModel *entry, VisualData )
                {
                    // Check if nominal data in it, the long way,
since .contains() does not like my operator==()
                    if ( ( QString::compare( entry-
>getFileName(),nominalData->getFileName(),Qt::CaseInsensitive ) == 0 )
&&
                         ( QString::compare( entry-
>getComponentName(),nominalData->getComponentName(),
Qt::CaseInsensitive ) == 0 ) )
                        entered = true;
```

```
                }
                if ( entered == false)
                    VisualData.append( nominalData );
            }


            if ( source.getSizeChange() != 0 )
                {
                // Check file change != 0, can be positive or
negative.
                // Add FileChange for current SourceData item
                    FileChange *change = new FileChange(
source.getSize(),

source.getSizeChange(),

source.getSize() );  //  Source data needs to show delta duration.

  //                  int i = VisualData.indexOf(nominalData);  NOT
WORKING!!!
                    int i=0;
                    int index=0;
                    bool indexFound = false;
                    foreach ( VisualDataModel *entry, VisualData )
                    {
                        i++;
                        // Check if nominal data in it, the long way,
since .contains() does not like my operator==()
                        if ( ( QString::compare( entry-
>getFileName(),nominalData->getFileName(),Qt::CaseInsensitive ) == 0 )
&&
                             ( QString::compare( entry-
>getComponentName(),nominalData->getComponentName(),
Qt::CaseInsensitive ) == 0 ) )
                        {
                            indexFound = true;
                            index = i-1;
                        }
                    }
                    if ( indexFound )
                    {
                        qDebug() << "Calling addFileChange...";
                        VisualData.at(index)->addFileChange( change );
                    }
                // For each filename, store every instance of a
FileChange where size != 0.
                }
            // We should now have a unique list of every file in
VisualData.
            // For each file we should have a list of file changes !=
0 in VisualData.DynamicData.
            }

        // Count how many files, count how many components; update UI
to show numbers.

        updateMainWindowStats();

        // Create 3D geometries for each file and Initialise location
for each.
```

```
        initVisualDataGeometry();

        // Update geometry to include colour based on DynamicData.

        addDynamicColour();

        // Attach geometries to scenegraph.

        buildScene();

        // Display loaded scenegraph

        view->update();

        // Hide the Loading progrss bar.
        ui->progressBar->hide();
        ui->Loading->hide();

        // now use VisualData to build temporalDataModel.

        tdm = new TemporalDataModel();
    }
}

void MainWindow::updateMainWindowStats()
{
    // Count how many files, count how many components; update UI to
show numbers.

    numFiles = VisualData.size();
    calcTotalFileChanges();
    ui->TotalFiles->setText( QString::number(numFiles));
    ui->TotalChanges->setText(
QString::number(getTotalFileChanges()));
    MainWindow::update();
}

void MainWindow::on_pushButton_3_clicked()
{
    view->resize(800, 600);
    view->size();
    view->show();
}

void MainWindow::processLineFromCSV( QString line )
{
    QStringList fields = line.split(",");
    SourceHistory tempSrcRow;

    tempSrcRow.setLabelName(fields.value(0));
    tempSrcRow.setComponentPath(fields.value(1));
    tempSrcRow.setRevision(fields.value(2).toInt());
    tempSrcRow.setSize(fields.value(3).toInt());
    tempSrcRow.setRevisionDate(fields.value(4) );
    tempSrcRow.setLabelDate(fields.value(5));
    tempSrcRow.setSizeChange(fields.value(6).toInt());
    tempSrcRow.setComponentName(fields.value(7));
    tempSrcRow.setFileName(fields.value(8));

    SourceData.append(tempSrcRow);
}
```

```
void MainWindow::loadFile( QString fileName )
{
    // load data straight from csv file for now...

    SourceData.clear();  // Wipe current contents of SourceData.

    QFile file(fileName);
    if (file.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        // Display the Loading progress bar.
        MainWindow::setFocus();

        MainWindow::setCounter(file.size());
        percent = counter / 100;
        ui->Loading->show();
        ui->progressBar->setValue(0);
        ui->progressBar->setMaximum(counter);
        ui->progressBar->show();
        qApp->processEvents();

        QTextStream in(&file);

        // Throw away header; ideally should keep in structure if
generalising data sources.

        QString line = in.readLine();

        while (!in.atEnd())
        {
            line = in.readLine();
            processLineFromCSV(line);
        }
        file.close();
    }

    // Give view pointer to Source Data.
    view->setSourceData( &SourceData );
}


void MainWindow::calcTotalFileChanges()
{
    // Traverse VisualData and add up all Dynamic Data entries.
    totalFilechanges = 0;
    int totalFiles = 0;
    foreach ( VisualDataModel *entry, VisualData )
    {
        totalFiles += entry->getTotalChanges();
    }
    setTotalFileChanges( totalFiles );
}


int MainWindow::getTotalFileChanges()
{
    return totalFilechanges;
}

void MainWindow::setTotalFileChanges( int total )
{
```

```
    totalFilechanges = total;
}

QString MainWindow::flattenList( QStringList l)
{
    QString tmpStr;
    if ( !l.isEmpty())
        foreach ( QString s, l )
        {
            tmpStr.append(s);
            tmpStr.append(".");
        }
    tmpStr.chop(1); // remove last period.
    qDebug() << "flattened list: " << tmpStr;
    return tmpStr;
}

bool MainWindow::isParent( QString parent, QString child )
{
    // compares parent string as immediate preceding string for child
name.

    // Loose the final child sub-element and compare to full parent
string.

    QStringList fields = child.split(".");
    fields.removeLast();
    QString tmp = flattenList(fields);

    qDebug() << "Parent: " << parent << " Child: " << child;
    qDebug()  << "tmp: " << tmp;

    if (tmp.compare(parent))
    {
        // Edge exists between parent and child.
        qDebug() << "Parent is true!";
        return true;
    }
    else
        return false;
}

ComponentNode *MainWindow::findComponent( QString name )
{
    foreach ( ComponentNode *node, Components )
    {
        if ( node->ifComponentNode( name ) )
        {
            qDebug() << "Parent node found: " << name;
            return node;
        }
    }
    return NULL;
}

int MainWindow::getDepth( ComponentNode *node, int i )
{
    // Recurse through parents counting depth.
    if ( node->getComponentParent() != NULL )
    {
        i = getDepth( node->getComponentParent(), ++i);
```

```
    }
    return i;
}

void MainWindow::addComponents()
{
    // Make up a unique list of component names in Components.

    QString childName;
    QList<QString> tmpLst;
    foreach ( VisualDataModel *vdm, VisualData )
    {
        childName = vdm->getComponentName();
        if ( !tmpLst.contains(childName) )  // Work Around 'contains'
not working for Components class.
        {
            qDebug() << "Adding component to list: " << childName;
            tmpLst.append(childName);
            ComponentNode *newNode = new ComponentNode();
            newNode->setComponentName(childName);
            Components.append(newNode);
        }
    }
}

void MainWindow::addParentComponents()
{
    // Make parent components, most will be 'abstract'

    foreach ( ComponentNode *node, Components )
    {
        // for each child, make parents from component name elements
        QString child, parent;
        QStringList parsedName;
        ComponentNode *parentNode;
        int treeDepth;

        child = node->getComponentName();
        parsedName = child.split('.');
        parsedName.removeLast();
        treeDepth = parsedName.size();

        // Loop and make parents
        for ( int j=0; j < treeDepth; j++)
        {
            parent = flattenList(parsedName);
            parentNode = findComponent( parent );
            if ( parentNode == NULL )
            {
                ComponentNode *newComponent = new ComponentNode();
                newComponent->setComponentName(parent);
                Components.append(newComponent);
                qDebug() << "Parent added: " << parent;
            }
            parsedName = parent.split('.');
        }
    }
}

void MainWindow::calcNodeDepth()
{
```

```
    // Set the tree depth for each node in hierarchy.

    foreach ( ComponentNode *node, Components )
    {
        int i = 0;
        // check dept, mul depth by nodeWeight to apply to yPos.
        int depth = getDepth( node, i );
        node->setDepth( depth );
        qDebug() << "node depth: " << node->getTreeDepth();
    }
}

void MainWindow::setComponentDelta()
{
    // y pos is multiple of Depth x NodeWeight by CurrentValue
    foreach ( ComponentNode *node, Components )
    {
        QVector3D vector(0.0, 0.0, 0.0);
        vector.setZ(node->getDeltaPos().z()+(node-
>getTreeDepth()*nodeWeight));
        vector.setX(node->getDeltaPos().x());
        vector.setY(node->getDeltaPos().y());
        node->setDeltaPos(vector);
    }

}

void MainWindow::setVisualDataModelComponents()
{
    foreach ( VisualDataModel *entry, VisualData )
    {
        // Find a matching Component in Components to associate with.
        foreach ( ComponentNode *node, Components )
        {
            if ( node->ifComponentNode(entry->getComponentName()) )
            {
                entry->setTreeNode(node);
                qDebug() << "VDM Component TreeNode set to: " << node-
>getComponentName();
            }
        }
    }
}

void MainWindow::buildComponentTree()
{
    // Make up a unique list of component names.

    addComponents();

    // Make parent components, most will be 'abstract'

    addParentComponents();

    //find a parent, store ptr, or leave as null.
    foreach ( ComponentNode *node, Components )
    {
        // loop to find and assign each parent
        QString child, parent;
        QStringList parsedName;
        ComponentNode *parentNode;
```

```
        child = node->getComponentName();
        parsedName = child.split('.');
        parsedName.removeLast();
        parent = flattenList(parsedName);

        parentNode = findComponent( parent );

        if ( parentNode != NULL )
        {
            node->setComponentParent( parentNode );
            qDebug() << "Parent set: " << parentNode-
>getComponentName();
        }
    }

    // Set the tree depth for each node in hierarchy.

    calcNodeDepth();

    // Apply node weights to y coord.

    setComponentDelta();

    // assign relevant Component to VDM elements.

    setVisualDataModelComponents();
}


void MainWindow::addComponentDelta()
{
    foreach ( VisualDataModel *vdm, VisualData )
    {
        QVector3D updatePos, delta;
        updatePos = vdm->getLocation();
        delta = vdm->getTreeNode()->getDeltaPos();

        updatePos.setX(updatePos.x() + delta.x() );
        updatePos.setY(updatePos.y() + delta.y() );
        updatePos.setZ(updatePos.z() + delta.z() );

        vdm->setLocation(updatePos);
    }
}

void MainWindow::setDynamicColour()
{
    foreach ( VisualDataModel *vdm, VisualData )
    {
        int changes = vdm->getTotalChanges();
        // check for dynamic Data, pickup colour change to set vdm
FinalColour.
        if ( changes != 0 )
        {
            if ( changes == 1 )
            {
                // low change
                vdm->setDynamicColour(QColor(25,25,200));
            }
            else if ( changes < 4 )
```

```
            {
                // medium change
                vdm->setDynamicColour(QColor(25,200,25));
            }
            else
            {
                // large changes
                vdm->setDynamicColour(QColor(200,25,25));
            }
        }
    }
}


void MainWindow::initVisualDataGeometry()
{
    // Create 3D geometries for each file and Initialise location for
each.

    float x, y, z;

    x = 0;
    y = 0;
    z = 0;

    qreal width = qSqrt((qreal)numFiles);  // X x Y = TotalFiles

    foreach ( VisualDataModel *vdm, VisualData )
    {
        // set x, y, z incrementing x, y only, leaving z at 0 for all.
        vdm->setLocation(QVector3D( x, y, z));

        // Make a width x height plane of objects calc from total
number of files.
        // After each row of obecjts, reset X and increment Y
position.
        if (x < ((int)width) )
            x += 1;     // Assume geometry centres are 2 x X apart.
        else
            {
                x = 0;
                y -= 1;
            }
        // Set inital colour - neutral.
        vdm->setFinalColour(defaultColour);
        vdm->setVolatility();  // Integer representation of final
Colour value.
    }

    // Change default colour based on volatility.

    if (tools->getEnableColour() )
    {
        setDynamicColour();
    }

    //  Now create Component hierarchy from which to modify relative
positions of objects.

    buildComponentTree();
```

```
    // Use component tree to update coordinate in for for objects

    if ( tools->getEnableDepth() )
    {
        addComponentDelta();
    }
}


void MainWindow::buildScene()
{
    // Save pointer to VisualData in View
    view->setVisualDataModel( VisualData );

    int count = 0;

    // Attach geometries to scenegraph.
    foreach ( VisualDataModel *vdm, VisualData )
    {
        // make a GeometryProperty for each node.
        qDebug() << "Adding file geometry..." << ++count;

        GeometryProperties gp;
        gp.setColour(vdm->getFinalColour());
        gp.setPosition(vdm->getLocation());

        // attach geometry to scene and return the scenenode ptr to
vdm.

        vdm->setNode( view->setNodeGeometry( gp ));

        // Update Progress Bar...
        if (counter % 200 == 0)
        {
          ui->progressBar->setValue(percent);
          percent += percent/2;
          qApp->processEvents();
        }
        counter--;

    }

    if ( view->isPicking())
    {
        // Once the scene is built we can register all nodes pickable
( OR NOT ).
        view->registerPickableNodes();
    }
}

void MainWindow::setCounter( int count )
{
    this->counter = count;
}

void MainWindow::setPercent( int percent )
{
    this->percent = percent;
}

int MainWindow::getCounter()
```

```
{
    return counter;
}

int MainWindow::getPercent()
{
    return percent;
}

void MainWindow::on_ShowTools_clicked()
{
    if (ui->ShowTools->isChecked())
    {
        tools->show();
    }
    else
        tools->hide();
}

MainWindow *MainWindow::getMainWindow()
{
    return this;
}
```

MainWindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "cubeview.h"
#include "sourcehistory.h"
#include "visualdatamodel.h"
#include <componentnode.h>
#include <visualtools.h>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    void loadFile( QString fileName );
    void processLineFromCSV( QString line );
    void calcTotalFileChanges();
    int getTotalFileChanges();
    void setTotalFileChanges( int );
    void updateMainWindowStats();
    void initVisualDataGeometry();
    void buildScene();
    void setCounter( int count );
    void setPercent( int percent );
    int getCounter();
    int getPercent();
    MainWindow *getMainWindow();
    void resetParameters();

private slots:
    void on_loadSource_clicked();

    void on_pushButton_3_clicked();

    void on_ShowTools_clicked();

private:
    Ui::MainWindow *ui;
    CubeView *view;
    VisualTools *tools;
    int numFiles;
    int numComponents;
    int totalFilechanges;
    int counter;        // Progress bar counter
    int percent;        // Progress bar percent

    QList<SourceHistory> SourceData;  // Holds raw csv data, can be
dropped after initialisation.
    QList<VisualDataModel *> VisualData;  // Visual model of data held
throughtout session.
```

```
    QList<ComponentNode *> Components;    // Categorical / Semantic
position info
    TemporalDataModel *tdm;     // Temporal data model held throughout
session for loaded data.

    int nodeWeight;
    QColor defaultColour;

protected:
    void buildComponentTree();
    QString flattenList( QStringList l);
    bool isParent( QString parent, QString child );
    ComponentNode *findComponent( QString name );
    int getDepth( ComponentNode *node, int i );
    void addComponents();
    void addParentComponents();
    void calcNodeDepth();
    void setComponentDelta();
    void setVisualDataModelComponents();
    void addComponentDelta();
    void setDynamicColour();
    void addDynamicColour();

public:

};

#endif // MAINWINDOW_H
```

Sourcehistory.h

```
/*
 *  SourceHistory.h
 *
 *  Defines the class use to import raw Perforce project data
 *
 *  Owner:          Niall Whelan
 *
 *  Date Modified:  20/06/2013
 *
 **/

#ifndef SOURCEHISTORY_H
#define SOURCEHISTORY_H

#include <QObject>
#include <QDate>

class SourceHistory
{
public:
    SourceHistory();

    void setLabelName(QString labelName);
    void setPath(QString path);
    void setRevision(int revision);
    void setSize(int size);
    void setRevisionDate(QString revisionDate);
    void setLabelDate(QString labelDateStr );
    void setSizeChange(int sizeChange);
    void setComponentPath(QString componentPath);
    void setComponentName(QString componentName);
    void setFileName(QString fileName);

    QString getLabelName();
    QString getPath();
    int     getRevision();
    int     getSize();
    QString   getRevisionDate();
    QString   getLabelDate();
    int     getSizeChange();
    QString getComponentPath();
    QString getComponentName();
    QString getFileName();

private:
    QString labelName;
    QString path;
    int     revision;
    int     size;
    QString revisionDate;
    QString labelDate;
    int     sizeChange;
    QString componentPath;
    QString componentName;
    QString fileName;


signals:
```

```
public slots:

};

#endif // SOURCEHISTORY_H
```

Sourcehistory.cpp

```
#include "sourcehistory.h"

SourceHistory::SourceHistory()
{
}



QString SourceHistory::getLabelName()
{
    return labelName;
}

QString SourceHistory::getPath()
{
    return path;
}

int     SourceHistory::getRevision()
{
    return revision;
}

int     SourceHistory::getSize()
{
    return size;
}

QString   SourceHistory::getRevisionDate()
{
    return revisionDate;
}

QString   SourceHistory::getLabelDate()
{
    return labelDate;
}

int     SourceHistory::getSizeChange()
{
    return sizeChange;
}

QString SourceHistory::getComponentPath()
{
    return componentPath;
}
```

```
QString SourceHistory::getComponentName()
{
    return componentName;
}

QString SourceHistory::getFileName()
{
    return fileName;
}

void SourceHistory::setLabelName(QString labelName)
{
    this->labelName = labelName;
}

void SourceHistory::setPath(QString path)
{
    this->path = path;
}

void SourceHistory::setRevision(int revision)
{
    this->revision = revision;
}

void SourceHistory::setSize(int size)
{
    this->size = size;
}

void SourceHistory::setRevisionDate(QString revisionDate)
{
    this->revisionDate = revisionDate;
}

void SourceHistory::setLabelDate(QString labelDateStr)
{
    this->labelDate = revisionDate;
}


void SourceHistory::setSizeChange(int sizeChange)
{
    this->sizeChange = sizeChange;
}

void SourceHistory::setComponentPath(QString componentPath)
{
    this->componentPath = componentPath;
}


void SourceHistory::setComponentName(QString componentName)
{
    this->componentName = componentName;
}

void SourceHistory::setFileName(QString fileName)
{
    this->fileName = fileName;
}
```

Visualdatamodel.h

```
#ifndef VISUALDATAMODEL_H
#define VISUALDATAMODEL_H

#include <QString>
#include <QDate>
#include <QList>
#include <filechange.h>
#include <QGLBuilder>
#include <componentnode.h>

class VisualDataModel
{
private:
    // Categorical - Nominal Source Data
    // Each File has 1..m relation to Dynamic Data below.
    QString fileName;
    QString filePath;
    QString componentName;
    QString componentPath;
    QDate buildDate;

    // Categorical - Ordinal and Interval - Dynamic Data
    QList<FileChange *> DynamicData;
    int volatility;     //  Sum R, G, B values.

    // Geometry data
    ComponentNode *treeNode;  // Binding to Components hierarchy for
this file.
    QVector3D location;
    QGLSceneNode *node;
    QColor finalColour, dynamicColour;

    bool transparent;   // Enable / Disable transparency of individual
visual object.

public:
    VisualDataModel();
    QString getFileName();
    QString getFilePath();
    QString getComponentName();
    QString getComponentPath();
    QDate getBuildDate();
    void setBuildDate( QString dateString );
    void setFileName( QString name );
    void setFilePath( QString path );
    void setComponentName( QString name );
    void setComponentPath( QString path );
    void addFileChange( FileChange *change );
    bool operator==( VisualDataModel *);
    bool operator<( VisualDataModel *v );      // Used for Qsort

    int getTotalChanges();
    void setTreeNode( ComponentNode *node );
    ComponentNode *getTreeNode();
    void setVolatility();
    int getVolatility();

    // Scene related methods
```

```
    QVector3D getLocation();
    void setLocation( QVector3D locate );
    void setNode( QGLSceneNode *node );
    QGLSceneNode *getNode();
    void setFinalColour( QColor colour );
    QColor getFinalColour();
    void setDynamicColour( QColor colour );
    QColor getDynamicColour();

    bool isTransparent();
    void setTransparency( bool flag );
};

#endif // VISUALDATAMODEL_H
```

Visualdatamodel.cpp

```
#include "visualdatamodel.h"
#include <QDebug>
#include <geometryproperties.h>

VisualDataModel::VisualDataModel()
{
    transparent = false;
    dynamicColour = QColor(0,0,0);
    volatility = 0;
}

QString VisualDataModel::getFileName()
{
    return fileName;
}

QString VisualDataModel::getFilePath()
{
    return filePath;
}

QString VisualDataModel::getComponentName()
{
    return componentName;
}

QString VisualDataModel::getComponentPath()
{
    return componentPath;
}

void VisualDataModel::setFileName( QString name )
{
    this->fileName = name;
}

void VisualDataModel::setFilePath( QString path )
{
    this->filePath = path;
}

void VisualDataModel::setComponentName( QString name )
{
    this->componentName = name;
}

void VisualDataModel::setComponentPath( QString path )
{
    this->componentPath = path;
}

void VisualDataModel::addFileChange( FileChange *change )
{
    // Add the filechange object to the QList for current VisualData
entry.
    DynamicData.append( change );
    qDebug() << "Adding file change to DynamicData... " << change-
>getSize();
```

```
    // This would be a good place to update Dynamic Colour
incrementally.

   volatility = 1; // Once first change is made we should set initial
value to 1.

    // Accumulate changes to assign colour within set palette.

//    if (volatility == 1 )
//        setDynamicColour(QColor(0,125,250));
//        else if (volatility == 2 )
//            setDynamicColour(QColor(50,250,100));
//        else
//            setDynamicColour(QColor(250,100,100));
}

bool VisualDataModel::operator==( VisualDataModel *v )
{
    if ( ( QString::compare( v->fileName,this-
>fileName,Qt::CaseInsensitive ) == 0 )
         && ( QString::compare( v->componentName,this->componentName,
Qt::CaseInsensitive) == 0 ) )
         return true;
    else
         return false;
}

int VisualDataModel::getTotalChanges()
{
    return DynamicData.size();
}

QVector3D VisualDataModel::getLocation()
{
    return location;
}

void VisualDataModel::setLocation( QVector3D locate )
{
    location = locate;
}

void VisualDataModel::setNode( QGLSceneNode *node )
{
    // Set transform and geometry properties for
    this->node = node;
}


QGLSceneNode *VisualDataModel::getNode()
{
    return node;
}


void VisualDataModel::setFinalColour( QColor colour )
{
    finalColour = colour;
}
```

```
QColor VisualDataModel::getFinalColour()
{
    return finalColour;
}

void VisualDataModel::setTreeNode( ComponentNode *node )
{
    treeNode = node;
}

ComponentNode *VisualDataModel::getTreeNode()
{
    return treeNode;
}


void VisualDataModel::setDynamicColour( QColor colour )
{
    dynamicColour = colour;
}


QColor VisualDataModel::getDynamicColour()
{
    return dynamicColour;
}

void VisualDataModel::setVolatility()
{
    int changes = getTotalChanges();

    if (changes == 1)
        volatility = 1;
    else if ( changes < 3 )
        volatility = 2;
    else
        volatility = 3;

}

int VisualDataModel::getVolatility()
{
    return volatility;
}

bool VisualDataModel::operator<( VisualDataModel *v )
{
    // If volatility of this is less that v then return true;
    if (this->volatility << v->getVolatility() )
        return true;
    else
        return false;
}

bool VisualDataModel::isTransparent()
{
    return transparent;
}

void VisualDataModel::setTransparency( bool flag )
```

```
{
    this->transparent = flag;
}

QDate VisualDataModel::getBuildDate()
{
    return this->buildDate;
}

void VisualDataModel::setBuildDate( QString dateString )
{
    qDebug() << "setBuildDate using: " << dateString;

    QDate newDate(QDate::fromString(dateString, "MM/dd/yyyy"));

    // parse string for day, month and year.
    this->buildDate.setDate( newDate.year(), newDate.month(),
newDate.day());
    qDebug() << "VDM - Build date added: " << buildDate;
}
```

Cubeview.h

```
#ifndef CUBEVIEW_H
#define CUBEVIEW_H

#include <qt3d/qglview.h>
#include <qt3d/QGLTexture2D>
#include <QGLFormat>
#include <visualdatamodel.h>
#include <geometryproperties.h>
#include <sourcehistory.h>
#include <temporaldatamodel.h>

QT_BEGIN_NAMESPACE
class QGLSceneNode;
QT_END_NAMESPACE

class VisualAbstractScene;

//! [1]
class CubeView : public QGLView
{
    Q_OBJECT
public:
    CubeView( QWindow *parent = 0 );
    ~CubeView();

    enum primativeShape { cube, sphere };

    QGLSceneNode *setNodeGeometry( GeometryProperties properties );
    void setVisualDataModel( QList<VisualDataModel *> v );
    void setCamera( QGLCamera *camera );
    QGLCamera *getCamera();
    CubeView *getView();

    void buildScene();
    void addComponentDelta();
    void removeComponentDelta();

    void setTranslation( QVector3D translate );
    QVector3D getTranslation();
    void setPrimativeShape( primativeShape shape );

    void Message(const QString& string);
    void registerPickableNodes();
    void setPicking( bool flag );
    bool isPicking();

    //    void addEdge( QGLSceneNode *node, vdm *VisData );

    bool isTransparent();
    void setTransparency( bool flag );
    QGLSceneNode *addGridGeometry();

    void drawGrid();
    void setGrid( bool flag );
    int getExtentX();
    int getExtentY();

    void setTemporal( bool flag );
```

```
    void setSourceData( QList<SourceHistory> *source );
    QList<SourceHistory> *getSourceHistory();

    void plotLine( QVector3D start, QVector3D end );

    void drawVisualModel(QGLPainter *painter);
    void drawTemporalModel(QGLPainter *painter);
    void buildTemporalScene();
    void setTemporalData( TemporalDataModel *tdm );
    TemporalDataModel *getTemporalDataModel();
    QGLSceneNode *setTemporalNodeGeometry( GeometryProperties
properties, int depth );

protected:
    void paintGL( QGLPainter *painter );
    void resizeGL(int w, int h);
    void initializeGL(QGLPainter *painter);

private:

    VisualAbstractScene *pPickingScene; // used for scene picking
    QGLSceneNode *topNode, *vdmNode, *temporalNode;

    QGLTexture2D logo;
    QList<VisualDataModel *> vdm;
    QList<SourceHistory> *SourceData;  // Holds raw csv data, can be
dropped after initialisation.
    TemporalDataModel *tdm;

    QGLCamera *myCamera;
    QVector3D userTranslation;
    primativeShape currentShape;

    bool transparency;  // enable / disable viewing transparency in
view.
    bool picking;       // enable / disable picking in the view.
    bool grid;          // enable / disable grid in view.
    bool temporal;      // enable / disable temporal cube view, false
mean vdm view.

public slots:
    void MouseClicked();

};
//! [1]

#endif
```

Cubeview.cpp

```
#include "cubeview.h"
#include "qglbuilder.h"
#include "qglcube.h"
#include <qt3d/QGLSphere>
#include <QPoint>
#include <QGLTeapot>
#include <QGLCylinder>
#include <QtAlgorithms>
#include <visualabstractscene.h>
#include <QGLPickNode>
#include <QMessageBox>
#include <QRect>
#include <QTime>

CubeView::CubeView( QWindow *parent )
    : QGLView(parent)
    , pPickingScene(0)
{
    // here a top level node for the app is created, and parented to
the view
    //    topNode = new QGLSceneNode(this);

    // setup Picking for Scene.
    pPickingScene = new VisualAbstractScene(this);
    topNode = pPickingScene->mainNode();
    pPickingScene->setPickable(true);
    picking = true;

    // Setup 2 child nodes for 2 visualisation types: vdm, temporal

    vdmNode = new QGLSceneNode(topNode);
    temporalNode = new QGLSceneNode(topNode);

    // By defualt vdmNode will only be visible.

    temporalNode->setOption(QGLSceneNode::HideNode, false );
    temporal = false;

    currentShape = cube;
    userTranslation = QVector3D(0.0,0.0,0.0);

    // Setup initial camera and view position.
    camera()->setEye(QVector3D(0.0f, 2.0f, -40.0f));

    // We can attach other nodes once the source data has been loaded.

    // Associate VisualDataModel to TemporalDataModel for later.

    tdm = new TemporalDataModel();  // Will not be initialised fully
at this point.
}

CubeView::~CubeView()
{
    delete topNode;
}

void CubeView::setCamera( QGLCamera *camera )
```

```
{
    this->myCamera = camera;
}

QGLCamera *CubeView::getCamera()
{
    return this->myCamera;
}

//void CubeView::addEdge( QGLSceneNode *node, QList<VisualDataModel *>
*vdm )
//{
//    // Add edge to parent node is necessary.


////    node->

//}


QGLSceneNode *CubeView::addGridGeometry()
{
    // build geometry, size should also be added to properties,
later...
    QGLBuilder builder;
    builder << QGL::Faceted;

    // Add vertices on X and Z axises from -50 to +50 for X and -20 to
+50 for Z

    float vertices[12] =
     {
         -20.0, -20.0, -10.0,   // A
         20.0, -20.0, -10.0,    // B
         20.0, 20.0, 10.0,      // C
         -20.0, 20.0, 10.0      // D
     };
     QGeometryData data;
     data.appendVertexArray(QArray<QVector3D>::fromRawData
        (reinterpret_cast<const QVector3D*>(vertices), 4));
     builder.addQuads(data);

     // Finalise geometry and get QGLSceneNode pointer.
    QGLSceneNode *thisNode = builder.finalizedSceneNode();

    thisNode->setParent(topNode);

    topNode->addNode(thisNode);  // Test for picking.

    return thisNode;
}


QGLSceneNode *CubeView::setNodeGeometry( GeometryProperties properties
)
{
    // build geometry, size should also be added to properties,
later...
    QGLBuilder builder;
    builder << QGL::Faceted;
```

```
    if ( currentShape == sphere )
        builder << QGLSphere(0.99,3);  // size of 1, smooth
    else if ( currentShape == cube )
        builder << QGLCube(1.0);  // size of 1, smooth

    // Finalise geometry and get QGLSceneNode pointer.
    QGLSceneNode *thisNode = builder.finalizedSceneNode();
    thisNode->setParent(vdmNode);

    vdmNode->addNode(thisNode);  // Test for picking.

    // Position Geometry based on GeometryProperties

    qDebug() << "Set Position: x:" << properties.getPosition().x() <<
" y:" <<
                properties.getPosition().y() << " z:" <<
properties.getPosition().z();

    thisNode->setPosition( properties.getPosition());

    // Add edge to parent node is necessary.
//    addEdge( thisnode, vdm );

    return thisNode;
}


QGLSceneNode *CubeView::setTemporalNodeGeometry( GeometryProperties
properties, int depth )
{
    // build geometry, size should also be added to properties,
later...
    QGLBuilder builder;
    builder << QGL::Faceted;

    if ( currentShape == sphere )
    {
        if ( depth < 2 )
        {
            builder << QGLSphere(0.99,3);  // size of 1, smooth
        }
        else
            builder << QGLSphere(0.99,1);  // size of 1, minimun
geometry
    }
    else if ( currentShape == cube )
        builder << QGLCube(1.0);  // size of 1, smooth

    // Finalise geometry and get QGLSceneNode pointer.
    QGLSceneNode *thisNode = builder.finalizedSceneNode();
    thisNode->setParent(temporalNode);

    temporalNode->addNode(thisNode);  // Test for picking.

    // Position Geometry based on GeometryProperties

    qDebug() << "Set Position: x:" << properties.getPosition().x() <<
" y:" <<
                properties.getPosition().y() << " z:" <<
properties.getPosition().z();
```

```
    thisNode->setPosition( properties.getPosition());

    // Add edge to parent node is necessary.
//    addEdge( thisnode, vdm );

    return thisNode;
}
void CubeView::setVisualDataModel( QList<VisualDataModel *> v )
{
    vdm = v;
}

void CubeView::initializeGL(QGLPainter *painter)
{
    glEnable(GL_BLEND);
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glColor3f(1.0, 1.0, 1.0);

    camera()->setEye(QVector3D(0.0f, 2.0f, -40.0f));
}

int CubeView::getExtentX()
{
    // return width of world occupied by objects.

    int minX = 0;
    int maxX = 0;

    foreach(  VisualDataModel *VisualDataItem, vdm )
    {
        // get min x;
        if (VisualDataItem->getLocation().x() < minX)
            minX = VisualDataItem->getLocation().x();
        // get max x;
        if (VisualDataItem->getLocation().x() > maxX)
            maxX = VisualDataItem->getLocation().x();
    }
    qDebug() << "MinX is: " << minX << " Max X is: " << maxX;
    return maxX - minX;
}


int CubeView::getExtentY()
{
    // return height of world occupied by objects.

    int minY = 0;
    int maxY = 0;

    foreach(  VisualDataModel *VisualDataItem, vdm )
    {
        // get min y;
        if (VisualDataItem->getLocation().y() < minY)
            minY = VisualDataItem->getLocation().y();
        // get max y;
        if (VisualDataItem->getLocation().y() > maxY)
            maxY = VisualDataItem->getLocation().y();
    }
    qDebug() << "MinY is: " << minY << " Max Y is: " << maxY;

    return maxY - minY;
```

```
}

void CubeView::drawVisualModel( QGLPainter *painter)
{
    // Draw the set of objects in the scene.
    if ( !vdm.isEmpty())
    {
        // Need to traverse all the SceneNodes and draw them...
        foreach(  VisualDataModel *VisualDataItem, vdm )
        {
            // Combine object Colour and Opacity

            QColor colour = QColor(VisualDataItem->getFinalColour());

            if ( isTransparent() )
                glEnable(GL_BLEND);
            else
                glDisable(GL_BLEND);

            colour.setAlpha(150);
            painter->setFaceColor(QGL::AllFaces, colour);

            VisualDataItem->getNode()->draw(painter);
        }
    }
}


void CubeView::drawTemporalModel( QGLPainter *painter)
{
    // Draw the cube of objects in the scene.
    if ( tdm->isInitialised() )
    {
        QList<TemporalPlane *> p = tdm->getPlanes();

        qDebug() << "planes..." << p;

        foreach(  TemporalPlane *temporalItem, p )
        {
            // Draw each node from each plane.
            qDebug() << "Drawing Temporal node: for: " <<
temporalItem->getBuild();

            // Combine object Colour and Opacity

            QColor colour = QColor(temporalItem->getFinalColour());

            if ( isTransparent() )
                glEnable(GL_BLEND);
            else
                glDisable(GL_BLEND);

            colour.setAlpha(150);
            painter->setFaceColor(QGL::AllFaces, colour);

            temporalItem->getNode()->draw( painter );
        }

    }
}
```

```
void CubeView::paintGL( QGLPainter *painter )
{
    // Set initial global settings for primatives.  Will modify based
on object dynamic data later.

    int width = getExtentX();
    int height = getExtentY();

    float x, y, z;
    x = userTranslation.x() - (width/2);
    y = userTranslation.y() + (height/2);

    // set camera back far enough to see extent of objects.
    if (width < 30)
        z = userTranslation.z() -10.0;
    else
        z = userTranslation.z() +45.0;

    painter->setStandardEffect(QGL::LitMaterial);
    painter->modelViewMatrix().translate(x, y, z);

    drawVisualModel( painter );      // Standard Visual Model

    drawTemporalModel( painter );    // Temporal Cube Model

    if (this->grid)
    {
        drawGrid();
    }

    // Test drawing general lines...

//    plotLine(QVector3D(0.0f, 0.0f, 0.0f), QVector3D(5.0f, 5.0f,
0.0f) );
}


void CubeView::drawGrid()
{
    int loop1, loop2;
    const int gridSize = 81;
//    glClearColor(1.0, 1.0, 1.0, 1.0); // White background

    glClearColor(0.40, 0.40, 0.40, 1.0);    // Grey background

    glColor3f(1.0,0.0,0.0);   // Set my lines colour...
    glLineWidth(2.0);
    glDisable(GL_LINE_SMOOTH);

    for ( loop1 =0; loop1<gridSize; loop1++)
    {
        for ( loop2=0; loop2<gridSize; loop2++)
        {
            // Draw Horizontal lines...
            if ( loop1<gridSize)
            {
                glBegin(GL_LINES);
                glVertex3f(-gridSize,0.0,-gridSize+(loop2*2));
                glVertex3f(gridSize,0.0,-gridSize+(loop2*2));
                glEnd();
            }
```

```
            // Draw vertical lines...
            if (loop2<gridSize)
            {
                glBegin(GL_LINES);
                glVertex3f(-gridSize+(loop1*2),0.0,-gridSize);
                glVertex3f(-gridSize+(loop1*2),0.0,gridSize);
                glEnd();
            }
        }
    }
    glFlush();
}


void CubeView::resizeGL(int w, int h)
{
    qDebug() << "resizeGL... Width: " << w << "  Height: " << h;

    int side = qMin(w, h);
    glViewport(((w - side)/2), ((h -side)/2), side, side);
}

CubeView *CubeView::getView()
{
    return this;
}

void CubeView::buildScene()
{
    int count = 0;

    // Record start and end time for rendering.
    QTime startTime = QTime::currentTime();

    // Attach geometries to scenegraph.
    foreach ( VisualDataModel *entry, vdm )
    {
        // make a GeometryProperty for each node.
        qDebug() << "Adding file geometry..." << ++count;

        GeometryProperties gp;
        gp.setColour(entry->getFinalColour());
        gp.setPosition(entry->getLocation());

        // Test Transparancy for objects.
        if ( transparency )
        {

            qDebug() << "enabling transparency in scene...";
            gp.setColour( QColor(entry->getFinalColour().red(),
                                 entry->getFinalColour().green(),
                                 entry->getFinalColour().blue(),
                                 0.95));
        }
        else
        {
            qDebug() << "Disabling transparency in scene...";

            gp.setColour(entry->getFinalColour());
        }
```

```
        // attach geometry to scene and return the scenenode ptr to
vdm.

        entry->setNode( setNodeGeometry( gp ));
    }

    if ( this->isPicking())
    {
        // Once the scene is built we can register all nodes pickable
( OR NOT ).
        registerPickableNodes();
        qDumpScene(topNode);
    }

    if ( temporal )
    {
        // Now initialise the temporalDataModel from the VDM.
        tdm->initialise( vdm );

        // Now build the temporal cube.
        buildTemporalScene();
    }

    if ( temporal )
    {
        // Show cube, hide VDM
        temporalNode->setOption(QGLSceneNode::HideNode, false );
        vdmNode->setOption(QGLSceneNode::HideNode, true);
    }
    else
    {
        // Show VDM, hide Cube
        temporalNode->setOption(QGLSceneNode::HideNode, true );
        vdmNode->setOption(QGLSceneNode::HideNode, false);
    }

    QTime endTime = QTime::currentTime();

    qint32 duration = endTime.msec() - startTime.msec();

    qDebug() << "Build time: " << duration;
}

void CubeView::addComponentDelta()
{
    foreach ( VisualDataModel *entry, vdm )
    {
        QVector3D updatePos, delta;
        updatePos = entry->getLocation();
        delta = entry->getTreeNode()->getDeltaPos();

        updatePos.setX(updatePos.x() + delta.x() );
        updatePos.setY(updatePos.y() + delta.y() );
        updatePos.setZ(updatePos.z() + delta.z() );

        entry->setLocation(updatePos);
    }
}

void CubeView::setTranslation( QVector3D translate )
{
```

```
    userTranslation = translate;
}

QVector3D CubeView::getTranslation()
{
    return userTranslation;
}

void CubeView::removeComponentDelta()
{
    foreach ( VisualDataModel *entry, vdm )
    {
        QVector3D updatePos, delta;
        updatePos = entry->getLocation();
        delta = entry->getTreeNode()->getDeltaPos();

        updatePos.setX(updatePos.x() - delta.x() );
        updatePos.setY(updatePos.y() - delta.y() );
        updatePos.setZ(updatePos.z() - delta.z() );

        entry->setLocation(updatePos);
    }
}

void CubeView::setPrimativeShape( primativeShape shape )
{
    this->currentShape = shape;
}

void CubeView::registerPickableNodes()
{
    // Enable picking in our QGLView.
    setOption(QGLView::ObjectPicking, true );
//    setOption(QGLView::ShowPicking, true );

    // Every node in our VisualAbstractScene class is potentially set
to pickable.
    pPickingScene->setPickable(true);
    pPickingScene->generatePickNodes();

    QList<QGLPickNode*> pickList = pPickingScene->pickNodes();

    QList<QGLPickNode*>::const_iterator it = pickList.constBegin();

    for (int i=0; it != pickList.constEnd(); ++it, i++)
    {
        QGLPickNode *pQGLPickNode = *it;
        pQGLPickNode->disconnect(this);

        // Criteria for object being pickable?  Start by setting all
pickable for now.

        registerObject(pQGLPickNode->id(), pQGLPickNode);
        QObject::connect(pQGLPickNode, SIGNAL(clicked()), this,
SLOT(MouseClicked()));
    }
}

void CubeView::Message(const QString& string)
{
 QMessageBox::information( 0, "System Message", string);
```

```
}

void CubeView::MouseClicked()
{
    QGLPickNode *pn = qobject_cast<QGLPickNode*>(sender());
    Q_ASSERT(pn);
    QGLSceneNode *n = pn->target();
    QString SceneNodeName = "";
    QVector3D pos = n->position();
    QString position = QString::number(pos.x(),'f',5);
    position = position + QString::number(pos.y(),'f',5) +
QString::number(pos.z(),'f',5);
    QString PickNodeName = pn->objectName();

    if(SceneNodeName.isEmpty())
     SceneNodeName = position;

    if(PickNodeName.isEmpty())
     PickNodeName = "Name Missing";

    int id = pn->id();
//    Message( QString("MouseClicked on SceneNodeName:
%1.\r\nPickNodeName:
%2\r\nID:%3").arg(SceneNodeName).arg(PickNodeName).arg(id) );
    qDebug() << "Picking: Mouseclicked...";
}

void CubeView::setTransparency( bool flag )
{
    this->transparency = flag;
}

bool CubeView::isTransparent()
{
    return this->transparency;
}

void CubeView::setPicking( bool flag )
{
    this->picking = flag;

    qDebug() << "Picking set to: " << flag;

    if ( picking )
        pPickingScene->setPickable(true);
    else
        pPickingScene->setPickable(false);
}

bool CubeView::isPicking()
{
    return this->picking;
}


void CubeView::setGrid( bool flag )
{
    this->grid = flag;
    if ( !grid )
        glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
```

```
}

void CubeView::setSourceData( QList<SourceHistory> *source )
{
    this->SourceData = source;
}

QList<SourceHistory> *CubeView::getSourceHistory()
{
    return this->SourceData;
}

void CubeView::setTemporal( bool flag )
{
    this->temporal = flag;
}

void CubeView::plotLine( QVector3D start, QVector3D end )
{
    glColor3f(1.0f,1.0f,1.0f);   // Set my lines colour...
    glLineWidth(3.0);
    glEnable(GL_LINE_SMOOTH);

    glBegin(GL_LINES);
    glVertex3f(start.x(), start.y(), start.z());
    glVertex3f(end.x(), end.y(), end.z());
    glEnd();

    glFlush();
}

void CubeView::setTemporalData( TemporalDataModel *tdm )
{
    this->tdm = tdm;
}


void CubeView::buildTemporalScene()
{
    // Temporal Scene contains a cube of primitives, one sqr section
per build date.

    qDebug() << "Adding TemporalCubeData...";

    int depth = 0;
    QList<QString> b = tdm->getBuilds();

    foreach ( QString build, b )
    {
        // Add a set of nodes for each build. +Z for older builds.

        foreach ( VisualDataModel *vdmEntry, vdm  )
        {
            // copy vdm node co-ords, add depth for each build.

            QGLSceneNode *node = vdmEntry->getNode();
            GeometryProperties gp;
            gp.setColour(vdmEntry->getFinalColour());

            // add depth to position Z.
            QVector3D pos = vdmEntry->getLocation();
```

```
            pos.setZ(pos.z() + depth);
            gp.setPosition(pos);

            // Add new node to cubeNodes QList.
            QGLSceneNode *cubeNode = setTemporalNodeGeometry( gp,
depth );

            // Store ref to node in TemporalData Planes QList.
            tdm->addPlaneNode(  cubeNode, gp, build );
        }
        qDebug() << "Adding Build level: " << depth;
        depth++;
    }
    tdm->setInitialised( true );
    // cubeNodes now has depth * num(vdm nodes) in a cube formation.
}

TemporalDataModel *CubeView::getTemporalDataModel()
{
    return this->tdm;
}
```

Filechange.h

```
#ifndef FILECHANGE_H
#define FILECHANGE_H

#include <QDate>
#include <QString>
#include <QColor>
#include <QRgb>

class FileChange
{
private:
    // Each file change details here, not nominal info.
    int revision;
    QDate revisionDate;
    QDate buildDate;
    QString buildNo;

    // Quantatative data

    int fileSize;
    int changeSize;
    int deltaChangeDays;
    int percentChange;

public:
    FileChange();
    FileChange( int size, int change, int delta);
    void setRevision( int revision );
    void setRevisionDate( QDate date );
    void setBuildDate( QDate date );
    void setBuildNo( int buildNo );

    int getRevision();
    QDate getRevisionDate();
    QDate getBuildDate();
    QString getBuildNo();
    int getSize();
    int getChange();
    int getDelta();
};

#endif // FILECHANGE_H
```

Filechange.cpp

```
#include "filechange.h"
#include <QDebug>

FileChange::FileChange()
{
}

FileChange::FileChange( int size, int change, int delta)
{
    this->fileSize = size;
    this->changeSize = change;
    this->deltaChangeDays = delta;

    // Initially RGBA will be:

    // R: Relative Change (255/size)*change
    // G:
    // B:
    // A: Rate of change - 0

    if ( size != 0)
        this->percentChange = (change / size ) * 100;

    qDebug() << "FileChange create: " << size;
}

void FileChange::setRevision( int revision )
{
    this->revision = revision;
}

void FileChange::setRevisionDate( QDate date )
{
    this->revisionDate = date;
}

void FileChange::setBuildDate( QDate date )
{
    this->buildDate = date;
}

void FileChange::setBuildNo( int buildNo )
{
    this->buildNo = buildNo;
}

int FileChange::getRevision()
{
    return revision;
}

QDate FileChange::getRevisionDate()
{
    return revisionDate;
}

QDate FileChange::getBuildDate()
{
```

```
    return buildDate;
}

QString FileChange::getBuildNo()
{
    return buildNo;
}
int FileChange::getSize()
{
    return fileSize;
}

int FileChange::getChange()
{
    return changeSize;
}

int FileChange::getDelta()
{
    return deltaChangeDays;
}
```

Geometryproperties.h

```
#ifndef GEOMETRYPROPERTIES_H
#define GEOMETRYPROPERTIES_H

#include <QVector3D>
#include <QColor>

class GeometryProperties
{
public:
    GeometryProperties();
    void setPosition( QVector3D locate );
    void setColour( QColor colour );
    QVector3D getPosition();
    QColor getColour();

private:
    QVector3D position;
    QColor colour;

};

#endif // GEOMETRYPROPERTIES_H
```

Geometryproperties.cpp

```
#include "geometryproperties.h"

GeometryProperties::GeometryProperties()
{
}

QColor GeometryProperties::getColour()
{
    return this->colour;
}

void GeometryProperties::setColour(QColor colour )
{
    this->colour = colour;
}

QVector3D GeometryProperties::getPosition()
{
    return position;
}

void GeometryProperties::setPosition(QVector3D pos )
{
    position = pos;
}
```

Componentnode.h

```
#ifndef COMPONENTNODES_H
#define COMPONENTNODES_H

#include <QString>
#include <QVector3D>

class ComponentNode
{
public:
    ComponentNode();
    ComponentNode( QString name, QVector3D delta );

    void setComponentName( QString name );
    QString getComponentName();

    void setDeltaPos( QVector3D pos );
    QVector3D getDeltaPos();
    bool operator==( ComponentNode * );

    void setComponentParent( ComponentNode *parent );
    ComponentNode *getComponentParent();
    bool ifComponentNode( QString name );
    void setDepth( int depth );
    int getTreeDepth();

private:
    QString componentName;
    QVector3D deltaPos;
    ComponentNode *parent;
    int treeDepth;
};

#endif // COMPONENTNODES_H
```

Componentnode.cpp

```
#include "componentnode.h"
#include <QDebug>

ComponentNode::ComponentNode()
{
    parent = NULL;
    treeDepth = 0;
    deltaPos = QVector3D(0.0,0.0,0.0);
}

ComponentNode::ComponentNode( QString name, QVector3D delta )
{
    this->componentName = name;
    this->deltaPos = delta;
    parent = NULL;
    treeDepth = 0;
}

void ComponentNode::setComponentName( QString name )
{
    this->componentName = name;
}

void ComponentNode::setDeltaPos( QVector3D pos )
{
    this->deltaPos = pos;
}

QString ComponentNode::getComponentName()
{
    return this->componentName;
}

QVector3D ComponentNode::getDeltaPos()
{
    return this->deltaPos;
}

bool ComponentNode::operator ==( ComponentNode *c )
{
    qDebug() << "comparing... " << c->getComponentName() << " and " <<
componentName;

    if ( c->getComponentName().compare(this->componentName) )
        {
            return true;
        }
    else
         return false;
}

bool ComponentNode::ifComponentNode( QString name )
{
    if ( componentName.compare(name) == 0 )
        return true;
    else
        return false;
}
```

```
void ComponentNode::setComponentParent( ComponentNode *parent )
{
    this->parent = parent;
}

ComponentNode * ComponentNode::getComponentParent( )
{
    return parent;
}

void ComponentNode::setDepth( int depth )
{
    this->treeDepth = depth;
}

int ComponentNode::getTreeDepth()
{
    return treeDepth;
}
```

Temporaldatamodel.h

```
#ifndef TEMPORALDATAMODEL_H
#define TEMPORALDATAMODEL_H

#include <QString>
#include <QDate>
#include <QList>
#include <QGLBuilder>
#include <visualdatamodel.h>
#include <temporalplane.h>
#include <geometryproperties.h>
class TemporalDataModel
{
private:
    QList<VisualDataModel *> vdm; // use vdm for normalised source
data.
    QList<QString> builds;          // Use each build date as temporal
reference.
    QList<TemporalPlane *> planes;  // one plane of nodes per build.
    bool transparent;
    bool initialised;

public:
    TemporalDataModel();
    void initialise(QList<VisualDataModel *> vdm );
    void setVisualDataModel( QList<VisualDataModel *> v);
    bool isTransparent();
    void setTransparency( bool flag );
    QList<VisualDataModel *> getVisualDataModel();
    QList<QString> getBuilds();
    void addPlaneNode( QGLSceneNode *node, GeometryProperties gp,
QString build );
    QList<TemporalPlane *> getPlanes();
    void setBuilds( QList<VisualDataModel *> vdm );

    void setInitialised( bool flag );
    bool isInitialised();
};

#endif // TEMPORALDATAMODEL_H
```

Temporaldatamodel.cpp

```
#include "temporaldatamodel.h"

TemporalDataModel::TemporalDataModel()
{
    this->initialised = false;
}

void TemporalDataModel::initialise( QList<VisualDataModel *> vdm )
{
    qDebug() << "Initialising TemporalDataModel...";

    // Associate VisualData to TemporalDataModel
    setVisualDataModel( vdm );

    // Populate the build list from vdm.
    setBuilds( vdm );

    this->initialised = true;  // Only true is planes added.
}

void TemporalDataModel::setBuilds( QList<VisualDataModel *> vdm )
{
    // Populate the build list from vdm.
    foreach ( VisualDataModel *entry, vdm )
    {
        QString dateString = entry-
>getBuildDate().toString("MM/dd/yyyy");
        builds.append(dateString);
    }
    qDebug() << "Load build dates:" << builds;
}

void TemporalDataModel::setVisualDataModel( QList<VisualDataModel *>
v)
{
    this->vdm = v;
}


bool TemporalDataModel::isTransparent()
{
    return this->transparent;
}

void TemporalDataModel::setTransparency( bool flag )
{
    this->transparent = flag;
}

QList<VisualDataModel *> TemporalDataModel::getVisualDataModel()
{
    return this->vdm;
}

QList<QString> TemporalDataModel::getBuilds()
{
    return this->builds;
}
```

```
void TemporalDataModel::addPlaneNode( QGLSceneNode *node,
GeometryProperties gp, QString build )
{
    qDebug() << "TemporalDataModel:addPlaneNode for build: " << build;
    TemporalPlane *tp = new TemporalPlane();
    tp->setBuild( build );
    tp->setNode( node );
    tp->setFinalColour( gp.getColour());
    tp->setLocation( gp.getPosition());
    planes.append( tp );
}

QList<TemporalPlane *> TemporalDataModel::getPlanes()
{
    return this->planes;
}


void TemporalDataModel::setInitialised( bool flag )
{
    this->initialised = flag;
}

bool TemporalDataModel::isInitialised()
{
    return this->initialised;
}
```

Temporalplane.h

```
#ifndef TEMPORALPLANE_H
#define TEMPORALPLANE_H

#include <QString>
#include <QList>
#include <QGLSceneNode>

class TemporalPlane
{
private:
    QString build;
    QGLSceneNode *planeNode;
    QVector3D location;
    QColor finalColour;

public:
    TemporalPlane();

    void setBuild( QString build );
    QString getBuild();

    // Scene related methods
    QVector3D getLocation();
    void setLocation( QVector3D locate );
    void setNode( QGLSceneNode *node );
    QGLSceneNode *getNode();
    void setFinalColour( QColor colour );
    QColor getFinalColour();

};

#endif // TEMPORALPLANE_H
```

Temporalplane.cpp

```
#include "temporalplane.h"

TemporalPlane::TemporalPlane()
{
}


// Scene related methods
QVector3D TemporalPlane::getLocation()
{
    return this->location;
}

void TemporalPlane::setLocation( QVector3D locate )
{
    this->location = locate;
}

void TemporalPlane::setNode( QGLSceneNode *node )
{
    this->planeNode = node;
}

QGLSceneNode *TemporalPlane::getNode()
{
    return this->planeNode;
}

void TemporalPlane::setFinalColour( QColor colour )
{
    this->finalColour = colour;
}

QColor TemporalPlane::getFinalColour()
{
    return this->finalColour;
}

void TemporalPlane::setBuild( QString build )
{
    this->build = build;
}

QString TemporalPlane::getBuild()
{
    return this->build;
}
```

visualabstractscene.h

```
#ifndef VISUALABSTRACTSCENE_H
#define VISUALABSTRACTSCENE_H

#include <QObject>
#include "qglabstractscene.h"

class QGLSceneNode;

class VisualAbstractScene : public QGLAbstractScene
{
    Q_OBJECT
public:
    explicit VisualAbstractScene(QObject *parent = 0);
    QGLSceneNode *mainNode() const { return pRootNode; }
    virtual QList<QObject *> objects() const;
    virtual void GeneratePickNodes() const;

private:
    QGLSceneNode *pRootNode;
signals:

public slots:

};

#endif // VISUALABSTRACTSCENE_H
```

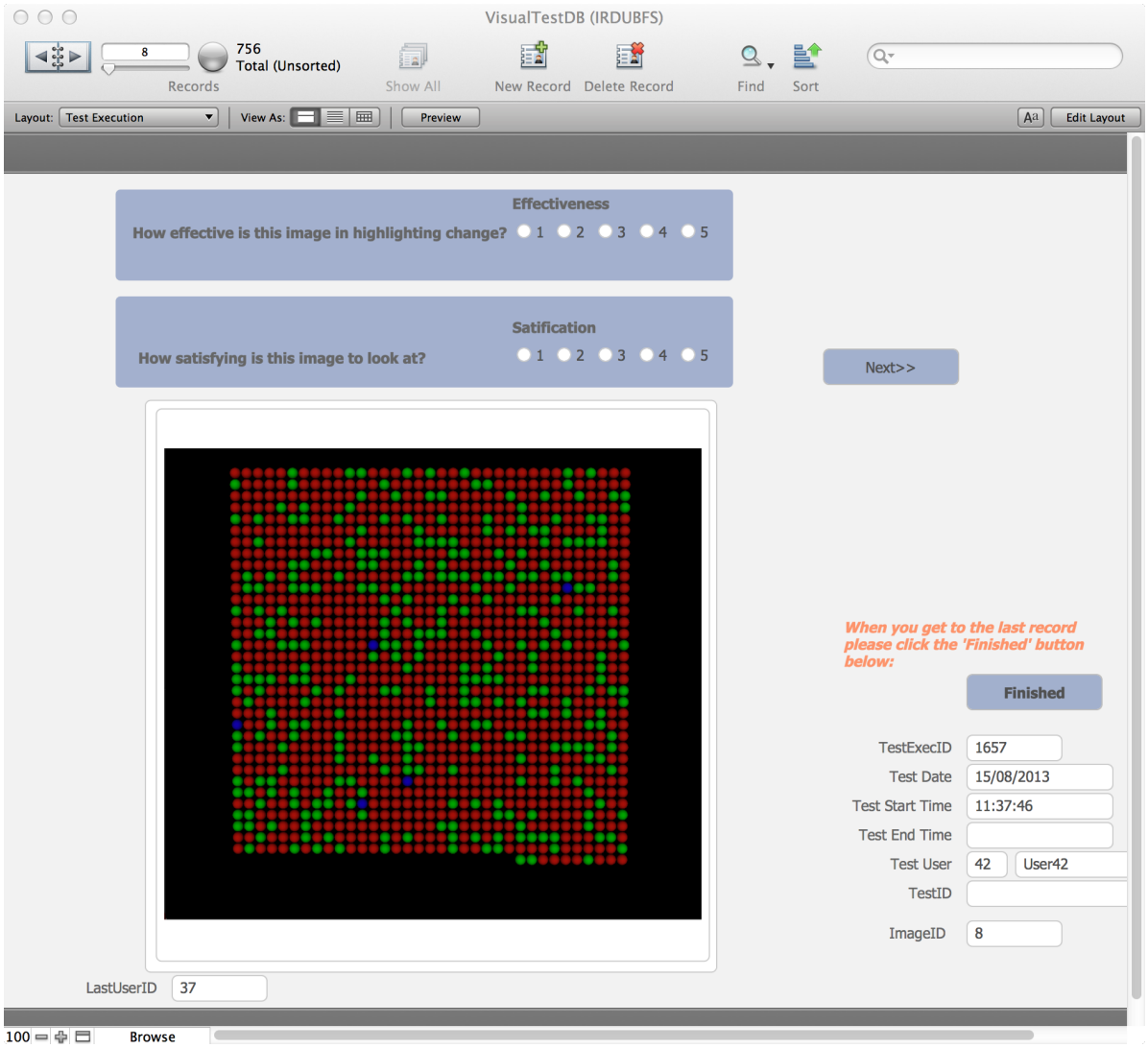
visualabstractscene.cpp

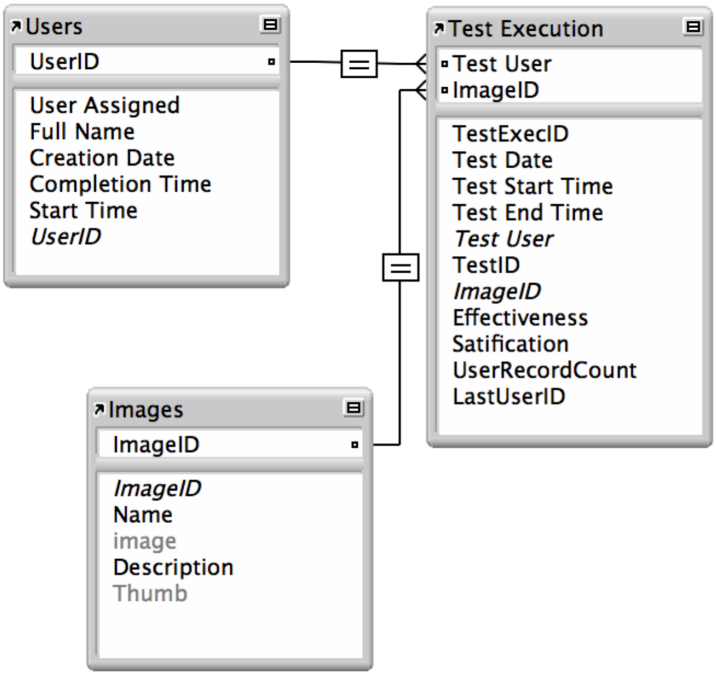
```
#include "visualabstractscene.h"
#include <QGLSceneNode>
#include <QGLPickNode>

VisualAbstractScene::VisualAbstractScene(QObject *parent) :
    QGLAbstractScene(parent)
    ,pRootNode(new QGLSceneNode(this))
{
}

QList<QObject *> VisualAbstractScene::objects() const
{
    QList<QGLSceneNode *> children = pRootNode->allChildren();
    QList<QObject *> objects;

    for (int index = 0; index < children.size(); ++index)
        objects.append(children.at(index));

    return objects;
}


void VisualAbstractScene::GeneratePickNodes() const
{
    QList<QObject *> objs = objects();
    QList<QObject *>::iterator it = objs.begin();

    for ( ; it != objs.end(); ++it)
    {
        QGLSceneNode *n = qobject_cast<QGLSceneNode *>(*it);
        QString name = n->objectName();

        QString it;
        n->setPickNode(new QGLPickNode());
    }
}
```

APPENDIX – C

Test Information

Screen and Schema from the VisualTestDB used for Usabillity testing.





Main.CPP

```
#include <QCoreApplication>
#include <QDebug>
#include <sourcehistory.h>
#include <QFile>
#include <QTextStream>
#include <QStringList>
#include <qnumeric.h>
#include <qglobal.h>
#include <QTime>
#include <iostream>

int randInt( int low, int high )
{
    return qrand() % ((high+1) - low) + low;
}

//enum testType { low, medium, high };

const int xfiles = 50;
const int xbuilds = 6;
const int xcomponentDepth = 3;

//testType x = low;

QList<SourceHistory> SourceData;


QString getComponentLevel( int seed, const QString levels[10], int
depth, QString filename )
{

    // Component depth needs to be the same for a file every time.

    QString component = "";

    component = levels[seed % 9];

    return component;
}


int getFileChange( int amount)
{
    int seed = randInt(0,12);
    int size = 0;

    switch (amount) {
        case 1 :
            {
            int seedLow = randInt(0,1);
            size = seedLow * 5;
            break;
           }
        case 2 :
            {
            int seedMed = randInt(0, seed / 4 );
            size = seedMed * 8;
            break;
```

```
            }
        default :
            size = seed * 12;
            break;
    }

    qDebug() << "making file change size: " << size;
    return size;
}

void generateRecords( int numfiles, int numBuilds, int componentDepth,
int type )
{

    const QString dates[10] = { "1/1/2012","1/2/2012", "1/3/2012",
"1/4/2012", "1/5/2012", "1/6/2012",
                        "1/7/2012", "1/8/2012", "1/9/2012",
"1/10/2012" };

    const QString levels[10] = { "CName1", "CName2", "CName3",
"CName4", "CName5",
                            "CName6", "CName7", "CName8", "CName9",
"CName10" };

    qDebug() << "generating records... ";

    for ( int b = 0; b < numBuilds; b++)
    {
        // Add Builds

        int any = randInt(0,9);

        QString date = dates[any];

        qDebug() << "Date: " << date;

        for ( int i = 0; i < numfiles; i++ )
        {
            // Add files per build
            QString filename = "file" + QString::number(i);
            QString buildname = "build" + QString::number(b);
            qDebug() << "Filename: " << filename << "Build: " <<
buildname;

            SourceHistory rec;

            rec.setLabelDate(date);
            rec.setRevisionDate(date);
            rec.setFileName( filename );
            rec.setLabelName(buildname);
            rec.setComponentName(getComponentLevel(i, levels,
componentDepth, filename ));
            rec.setComponentPath("blank");
            rec.setLabelName("blank");
            rec.setPath("blank");
            rec.setRevision(any);
            rec.setSize( any * 103 );
            rec.setSizeChange( getFileChange(type) );

            SourceData.append(rec);
        }
```

```
    }
}

void writeFile( QString filename, QList<SourceHistory> data )
{
    QFile mFile(filename );

    if (!mFile.open(QFile::WriteOnly | QFile::Text))
    {
        qDebug() << "Could not open file for write... ";
        return;
    }

    QTextStream out(&mFile);

    foreach( SourceHistory row, data )
    {
        qDebug() << "writing row from sourceData...";
        out << row.getLabelName()   << "," <<   row.getPath() << ","
<<
               row.getRevision()    << "," <<   row.getSize() << ","
<<
               row.getRevisionDate() << "," <<  row.getLabelDate() <<
"," <<
               row.getSizeChange()  << "," <<   row.getComponentName()
<< "," <<
               row.getFileName()    << "\r\n";
    }
    mFile.flush();
    mFile.close();
}


void readFile( QString filename )
{
    QFile mFile(filename );

    if (!mFile.open(QFile::ReadOnly | QFile::Text))
    {
        qDebug() << "Could not open file for read... ";
        return;
    }

    qDebug() << "Reading file.. " << filename;

    QTextStream in(&mFile);
    QString mText = in.readAll();

    qDebug() << mText;

    mFile.close();
}


int main(int argc, char *argv[])
{

    QCoreApplication a(argc, argv);


    QTime time = QTime::currentTime();
```

```
    qsrand((uint)time.msec());

    std::cout << "Console app VisualTest running..." <<
a.arguments().size();

    int files, builds, depth, amount;
    QString outfile;
    SourceData.clear();

    if ( a.arguments().size() == 7)
    {
        files = a.arguments().at(2).toInt();
        builds = a.arguments().at(3).toInt();
        depth = a.arguments().at(4).toInt();
        amount = a.arguments().at(5).toInt();
        outfile = a.arguments().at(6);
    }
    else
        exit(1);

    qDebug() << "Calling generateRecords with parameters: " << files
<< ", " << builds << ", " << depth << ", " << amount;
    generateRecords( files, builds, depth, amount );

    writeFile( outfile, SourceData );
    readFile( outfile );

    return a.exec();
}
```

SourceHistory.H

```
/*
 *  SourceHistory.h
 *
 *  Defines the class use to import raw Perforce project data
 *
 *  Owner:          Niall Whelan
 *
 *  Date Modified:  20/06/2013
 *
 **/

#ifndef SOURCEHISTORY_H
#define SOURCEHISTORY_H

#include <QObject>
#include <QDate>

class SourceHistory
{
public:
    SourceHistory();

    void setLabelName(QString labelName);
    void setPath(QString path);
    void setRevision(int revision);
    void setSize(int size);
    void setRevisionDate(QString revisionDate);
    void setLabelDate(QString labelDateStr );
    void setSizeChange(int sizeChange);
    void setComponentPath(QString componentPath);
    void setComponentName(QString componentName);
    void setFileName(QString fileName);

    QString getLabelName();
    QString getPath();
    int     getRevision();
    int     getSize();
    QString   getRevisionDate();
    QString   getLabelDate();
    int     getSizeChange();
    QString getComponentPath();
    QString getComponentName();
    QString getFileName();

private:
    QString labelName;
    QString path;
    int     revision;
    int     size;
    QString revisionDate;
    QString labelDate;
    int     sizeChange;
    QString componentPath;
    QString componentName;
    QString fileName;


signals:
```

```
public slots:

};

#endif // SOURCEHISTORY_H
```

SourceHistory.CPP

```
#include "sourcehistory.h"

SourceHistory::SourceHistory()
{
}

QString SourceHistory::getLabelName()
{
    return labelName;
}

QString SourceHistory::getPath()
{
    return path;
}

int     SourceHistory::getRevision()
{
    return revision;
}

int     SourceHistory::getSize()
{
    return size;
}

QString   SourceHistory::getRevisionDate()
{
    return revisionDate;
}

QString   SourceHistory::getLabelDate()
{
    return labelDate;
}

int     SourceHistory::getSizeChange()
{
    return sizeChange;
}

QString SourceHistory::getComponentPath()
{
    return componentPath;
}

QString SourceHistory::getComponentName()
{
    return componentName;
}

QString SourceHistory::getFileName()
{
    return fileName;
}

void SourceHistory::setLabelName(QString labelName)
{
```

```
    this->labelName = labelName;
}

void SourceHistory::setPath(QString path)
{
    this->path = path;
}

void SourceHistory::setRevision(int revision)
{
    this->revision = revision;
}

void SourceHistory::setSize(int size)
{
    this->size = size;
}

void SourceHistory::setRevisionDate(QString revisionDate)
{
    this->revisionDate = revisionDate;
}

void SourceHistory::setLabelDate(QString labelDateStr)
{
    this->labelDate = labelDateStr;
}


void SourceHistory::setSizeChange(int sizeChange)
{
    this->sizeChange = sizeChange;
}

void SourceHistory::setComponentPath(QString componentPath)
{
    this->componentPath = componentPath;
}


void SourceHistory::setComponentName(QString componentName)
{
    this->componentName = componentName;
}

void SourceHistory::setFileName(QString fileName)
{
    this->fileName = fileName;
}
```

Test Data file: Med_Small.CSV

```
blank,blank,1,103,1/2/2012,1/2/2012,8,CName1,file0
blank,blank,1,103,1/2/2012,1/2/2012,0,CName2,file1
blank,blank,1,103,1/2/2012,1/2/2012,0,CName3,file2
blank,blank,1,103,1/2/2012,1/2/2012,0,CName4,file3
blank,blank,1,103,1/2/2012,1/2/2012,16,CName5,file4
blank,blank,1,103,1/2/2012,1/2/2012,8,CName6,file5
blank,blank,1,103,1/2/2012,1/2/2012,0,CName7,file6
blank,blank,1,103,1/2/2012,1/2/2012,0,CName8,file7
blank,blank,1,103,1/2/2012,1/2/2012,0,CName9,file8
blank,blank,1,103,1/2/2012,1/2/2012,0,CName1,file9
blank,blank,1,103,1/2/2012,1/2/2012,0,CName2,file10
blank,blank,1,103,1/2/2012,1/2/2012,0,CName3,file11
blank,blank,1,103,1/2/2012,1/2/2012,0,CName4,file12
blank,blank,1,103,1/2/2012,1/2/2012,8,CName5,file13
blank,blank,1,103,1/2/2012,1/2/2012,0,CName6,file14
blank,blank,1,103,1/2/2012,1/2/2012,8,CName7,file15
blank,blank,1,103,1/2/2012,1/2/2012,0,CName8,file16
blank,blank,1,103,1/2/2012,1/2/2012,0,CName9,file17
blank,blank,1,103,1/2/2012,1/2/2012,16,CName1,file18
blank,blank,1,103,1/2/2012,1/2/2012,16,CName2,file19
blank,blank,1,103,1/2/2012,1/2/2012,16,CName3,file20
blank,blank,1,103,1/2/2012,1/2/2012,8,CName4,file21
blank,blank,1,103,1/2/2012,1/2/2012,0,CName5,file22
blank,blank,1,103,1/2/2012,1/2/2012,8,CName6,file23
blank,blank,1,103,1/2/2012,1/2/2012,0,CName7,file24
blank,blank,1,103,1/2/2012,1/2/2012,0,CName8,file25
blank,blank,1,103,1/2/2012,1/2/2012,0,CName9,file26
blank,blank,1,103,1/2/2012,1/2/2012,0,CName1,file27
blank,blank,1,103,1/2/2012,1/2/2012,0,CName2,file28
blank,blank,1,103,1/2/2012,1/2/2012,16,CName3,file29
blank,blank,9,927,1/10/2012,1/10/2012,8,CName1,file0
blank,blank,9,927,1/10/2012,1/10/2012,0,CName2,file1
blank,blank,9,927,1/10/2012,1/10/2012,0,CName3,file2
blank,blank,9,927,1/10/2012,1/10/2012,0,CName4,file3
blank,blank,9,927,1/10/2012,1/10/2012,8,CName5,file4
blank,blank,9,927,1/10/2012,1/10/2012,0,CName6,file5
blank,blank,9,927,1/10/2012,1/10/2012,0,CName7,file6
blank,blank,9,927,1/10/2012,1/10/2012,0,CName8,file7
blank,blank,9,927,1/10/2012,1/10/2012,0,CName9,file8
blank,blank,9,927,1/10/2012,1/10/2012,8,CName1,file9
blank,blank,9,927,1/10/2012,1/10/2012,0,CName2,file10
blank,blank,9,927,1/10/2012,1/10/2012,0,CName3,file11
blank,blank,9,927,1/10/2012,1/10/2012,16,CName4,file12
blank,blank,9,927,1/10/2012,1/10/2012,0,CName5,file13
blank,blank,9,927,1/10/2012,1/10/2012,8,CName6,file14
blank,blank,9,927,1/10/2012,1/10/2012,0,CName7,file15
blank,blank,9,927,1/10/2012,1/10/2012,0,CName8,file16
blank,blank,9,927,1/10/2012,1/10/2012,16,CName9,file17
blank,blank,9,927,1/10/2012,1/10/2012,8,CName1,file18
blank,blank,9,927,1/10/2012,1/10/2012,8,CName2,file19
blank,blank,9,927,1/10/2012,1/10/2012,16,CName3,file20
blank,blank,9,927,1/10/2012,1/10/2012,0,CName4,file21
blank,blank,9,927,1/10/2012,1/10/2012,0,CName5,file22
blank,blank,9,927,1/10/2012,1/10/2012,0,CName6,file23
blank,blank,9,927,1/10/2012,1/10/2012,0,CName7,file24
blank,blank,9,927,1/10/2012,1/10/2012,0,CName8,file25
blank,blank,9,927,1/10/2012,1/10/2012,8,CName9,file26
blank,blank,9,927,1/10/2012,1/10/2012,8,CName1,file27
```

```
blank,blank,9,927,1/10/2012,1/10/2012,0,CName2,file28
blank,blank,9,927,1/10/2012,1/10/2012,0,CName3,file29
blank,blank,2,206,1/3/2012,1/3/2012,8,CName1,file0
blank,blank,2,206,1/3/2012,1/3/2012,24,CName2,file1
blank,blank,2,206,1/3/2012,1/3/2012,0,CName3,file2
blank,blank,2,206,1/3/2012,1/3/2012,0,CName4,file3
blank,blank,2,206,1/3/2012,1/3/2012,0,CName5,file4
blank,blank,2,206,1/3/2012,1/3/2012,0,CName6,file5
blank,blank,2,206,1/3/2012,1/3/2012,8,CName7,file6
blank,blank,2,206,1/3/2012,1/3/2012,0,CName8,file7
blank,blank,2,206,1/3/2012,1/3/2012,0,CName9,file8
blank,blank,2,206,1/3/2012,1/3/2012,0,CName1,file9
blank,blank,2,206,1/3/2012,1/3/2012,0,CName2,file10
blank,blank,2,206,1/3/2012,1/3/2012,0,CName3,file11
blank,blank,2,206,1/3/2012,1/3/2012,0,CName4,file12
blank,blank,2,206,1/3/2012,1/3/2012,0,CName5,file13
blank,blank,2,206,1/3/2012,1/3/2012,0,CName6,file14
blank,blank,2,206,1/3/2012,1/3/2012,16,CName7,file15
blank,blank,2,206,1/3/2012,1/3/2012,0,CName8,file16
blank,blank,2,206,1/3/2012,1/3/2012,0,CName9,file17
blank,blank,2,206,1/3/2012,1/3/2012,0,CName1,file18
blank,blank,2,206,1/3/2012,1/3/2012,0,CName2,file19
blank,blank,2,206,1/3/2012,1/3/2012,16,CName3,file20
blank,blank,2,206,1/3/2012,1/3/2012,0,CName4,file21
blank,blank,2,206,1/3/2012,1/3/2012,0,CName5,file22
blank,blank,2,206,1/3/2012,1/3/2012,0,CName6,file23
blank,blank,2,206,1/3/2012,1/3/2012,16,CName7,file24
blank,blank,2,206,1/3/2012,1/3/2012,16,CName8,file25
blank,blank,2,206,1/3/2012,1/3/2012,8,CName9,file26
blank,blank,2,206,1/3/2012,1/3/2012,16,CName1,file27
blank,blank,2,206,1/3/2012,1/3/2012,8,CName2,file28
blank,blank,2,206,1/3/2012,1/3/2012,0,CName3,file29
blank,blank,0,0,1/1/2012,1/1/2012,0,CName1,file0
blank,blank,0,0,1/1/2012,1/1/2012,0,CName2,file1
blank,blank,0,0,1/1/2012,1/1/2012,8,CName3,file2
blank,blank,0,0,1/1/2012,1/1/2012,8,CName4,file3
blank,blank,0,0,1/1/2012,1/1/2012,0,CName5,file4
blank,blank,0,0,1/1/2012,1/1/2012,0,CName6,file5
blank,blank,0,0,1/1/2012,1/1/2012,0,CName7,file6
blank,blank,0,0,1/1/2012,1/1/2012,16,CName8,file7
blank,blank,0,0,1/1/2012,1/1/2012,8,CName9,file8
blank,blank,0,0,1/1/2012,1/1/2012,8,CName1,file9
blank,blank,0,0,1/1/2012,1/1/2012,8,CName2,file10
blank,blank,0,0,1/1/2012,1/1/2012,0,CName3,file11
blank,blank,0,0,1/1/2012,1/1/2012,0,CName4,file12
blank,blank,0,0,1/1/2012,1/1/2012,0,CName5,file13
blank,blank,0,0,1/1/2012,1/1/2012,8,CName6,file14
blank,blank,0,0,1/1/2012,1/1/2012,0,CName7,file15
blank,blank,0,0,1/1/2012,1/1/2012,8,CName8,file16
blank,blank,0,0,1/1/2012,1/1/2012,0,CName9,file17
blank,blank,0,0,1/1/2012,1/1/2012,8,CName1,file18
blank,blank,0,0,1/1/2012,1/1/2012,8,CName2,file19
blank,blank,0,0,1/1/2012,1/1/2012,16,CName3,file20
blank,blank,0,0,1/1/2012,1/1/2012,0,CName4,file21
blank,blank,0,0,1/1/2012,1/1/2012,8,CName5,file22
blank,blank,0,0,1/1/2012,1/1/2012,8,CName6,file23
blank,blank,0,0,1/1/2012,1/1/2012,0,CName7,file24
blank,blank,0,0,1/1/2012,1/1/2012,0,CName8,file25
blank,blank,0,0,1/1/2012,1/1/2012,0,CName9,file26
blank,blank,0,0,1/1/2012,1/1/2012,0,CName1,file27
blank,blank,0,0,1/1/2012,1/1/2012,0,CName2,file28
```

```
blank,blank,0,0,1/1/2012,1/1/2012,8,CName3,file29
```