

Visualization and Analysis in Automated Trace Retrieval

Chuan Duan and Jane Cleland-Huang

Center for Requirements Engineering

School of Computer Science, Telecommunications, and Information Systems

DePaul University

{duanchuan, jhuang}@cti.depaul.edu

Abstract

This paper describes a new visualization technique for helping analysts to understand the potential impact of changing requirements, and for providing valuable early feedback on the quality of a software design. Based on the candidate links that are automatically generated by a trace retrieval tool, VisMatrix creates a graphical representation of the requirements trace matrix showing not only where candidate links exist, but also the strength of those links. New metrics derived from the trace visualization, and its underlying trace matrix, are introduced. These include ‘trace clustering’ for evaluating modularity, and ‘trace scope’ for analyzing fan-in and fan-out behavior of traces, where fan-in represents the number of requirements influencing a design artifact, and fan-out represents the degree of influence a single requirement has on the set of design artifacts. Examples are drawn from the automatically generated trace matrices of four different data sets to illustrate the visualization techniques and analyze the proposed metrics.

1. Automated Trace Retrieval

Requirements traceability supports numerous software engineering activities such as requirements validation, compliance verification, and change management. Unfortunately, typical practices in which a requirements trace matrix (RTM) is manually constructed tend to be costly to implement and extremely hard to maintain over the long-term [1, 2]. To solve this problem researchers have developed methods for automating the process of generating traceability links [3, 4, 5, 6]. These approaches typically use information retrieval techniques to link artifacts based on shared terms that occur in both the requirement and the set of searchable documents or by considering the hierarchical and grammatical structuring of the artifacts [7]. A *term* is defined as a single word or phrase that may occur in either a query or a document. A *query* typically represents a single requirement, while a traceable artifact known as a *document* constitutes any artifact that is part of the search space. Examples of

traceable artifacts include design elements, code modules, UML components, or test cases. As part of the retrieval process, text within the query and documents is preprocessed to remove stop words, which are common words, such as conjunctions and articles that appear too frequently across multiple documents to be useful in the trace generation process. The remaining words are then stemmed to remove prefixes and suffixes in order to reduce them to their common grammatical roots, so that traces between artifacts containing syntactic variations of the same words can be correctly retrieved.

In the probabilistic approach described by Cleland-Huang et al [8], first the frequency of every term t_i appearing in the query q , $\text{freq}(q, t_i)$, and the frequency of every term t_i appearing in document d_j , $\text{freq}(d_j, t_i)$, are computed, then the probability of a link between each potential query-document pair d_j and q is calculated

as
$$\text{pr}(d_j | q) = \left[\sum_{i=1}^k \text{pr}(d_j | t_i) \text{pr}(q, t_i) \right] / \text{pr}(q) \quad \text{where}$$

$$\text{pr}(d_j | t_i) = \frac{\text{freq}(d_j, t_i)}{\sum_k \text{freq}(d_j, t_k)}$$
 represents the degree of

information provided by term t_i to describe the concept d_j ;

$$\text{pr}(q, t_i) = \frac{\text{freq}(q, t_i)}{n_i}$$
, in which n_i represents the number

of documents in the collection containing the term t_i , $\text{pr}(q, t_i)$ represents the degree of relevance of the term t_i to

describe the concept of the query; and
$$\text{pr}(q) = \sum_i \text{pr}(q, t_i)$$
 denotes the sum of degree of

relevance for each term t_i according to query q . After computing all of the probabilities, the results are ranked according to probability scores and a threshold value is used to categorize them as candidate (likely) or unlikely links. The performance of the algorithm is primarily evaluated using the two metrics of recall and precision, where recall is defined as the number of correct candidate links over the total number of true links, and precision is defined as the number of correct candidate links over the number of candidate links. Supporting data, such as use of hierarchical information, logical groupings of artifacts, and semi-automated pruning, can be incorporated into the

probabilistic retrieval algorithm in order to improve the results returned by an automated requirements traceability tool [9].

Experimental results indicate that it is feasible to replace traditional trace methods with dynamic ones, and that precision ranging from 20-30% can be achieved at recall values of 90-95% [9]. Visualization increases the usefulness of trace retrieval tools by providing developers with a more intuitive understanding of the results of a single trace query as well as the general traceability of a system. For example, the visualized matrix could show whether a requirement has localized or far reaching impact across the design. It could also help analysts to understand the extent to which an individual artifact is impacted by one or more requirements.

This paper therefore describes a traceability visualization technique and tool that supports three specific objectives. First it provides a visual representation of the candidate links, in order to graphically display the traces generated between system artifacts. Secondly, it provides visual feedback about the correctness of the generated links which is useful if an organization wishes to conduct a pilot study to evaluate the potential use of automated trace methods, or to monitor its effectiveness in an ongoing project. Finally, it combines architectural complexity metrics, such as coupling and cohesion, with knowledge of the generated traceability links in order to detect potential risks in the design and identify requirements that may be of high architectural significance.

VisMatrix, a trace matrix visualization tool, was designed with these purposes in mind. This paper describes the tool and the associated traceability-based metrics. The approach is illustrated and evaluated through four data sets: the Ice-Breaker system (IBS), Event Based Traceability (EBT), Light Control System (LC), and Zodiac. The IBS system was designed to manage road de-icing, work schedules, inventory, and truck maintenance for a public works department. It consists of 180 functional requirements, 19 use cases, and 75 java classes. The EBT system is a prototype system for supporting traceability in a distributed environment. It includes 37 functional requirements and 50 java classes with a total of 11,870 lines of code. The LC system was developed for the University of Kaiserslautern [2], to control lights in a building based upon current external lighting conditions, building usage, and user defined schemes. Our version of this system consists of 36 requirements, 25 classes, and 5 packages. Additional details describing these three data sets can be found in [9]. Zodiac is a collaborative tool for supporting the elicitation and visual modeling of non-functional goals. Developed as part of a 22 person, two quarter software engineering studio course at DePaul University using an agile approach, it contains 20,904 lines of java code, 204 classes, and 108 user stories.

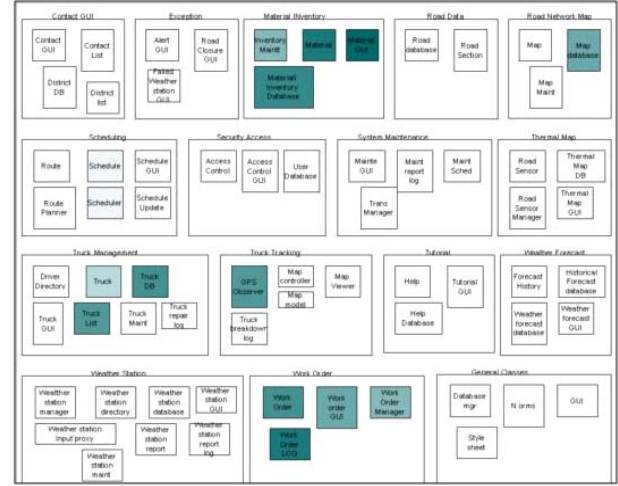


Figure 1. Poirot visualization of a trace query

The remainder of the paper is organized as follows. Section 2 surveys and discusses prior work in trace visualization and section 3 introduces the trace visualization tool named VisMatrix. Section 4 describes the application of VisMatrix to help evaluate the performance of an automated traceability tool. Section 5 then introduces and evaluates several ideas for analyzing the quality of a design based on the confidence of generated links and a set of standard Object-Oriented metrics. Finally section 6 concludes with a summary of our findings and suggestions for future work.

2. Related Visualization Work

Several researchers have investigated traceability visualization techniques. For example the trace tool *Poirot*, which implements the probabilistic algorithm described in Section 1, includes a feature for visualizing traces from an individual requirement to all classes in a system. As depicted in Figure 1, classes are grouped into packages, and shaded according to the strength of their probability links. Darker shading represents links with stronger probability values, while lighter shading represents lower values. The visualization clearly highlights the clustering effect of a query; however the method is unable to support simultaneous visualization of traces from multiple queries. It also does not incorporate additional information about the design that could help an analyst to perform a more meaningful impact analysis. Similar weaknesses are found in TraceVis developed by Marcus et al. [10], which utilizes a similar technique to provide visualization support during traceability recovery, maintenance, and browsing.

Antoniol et al. implemented traceability visualization to support reverse engineering and software maintenance

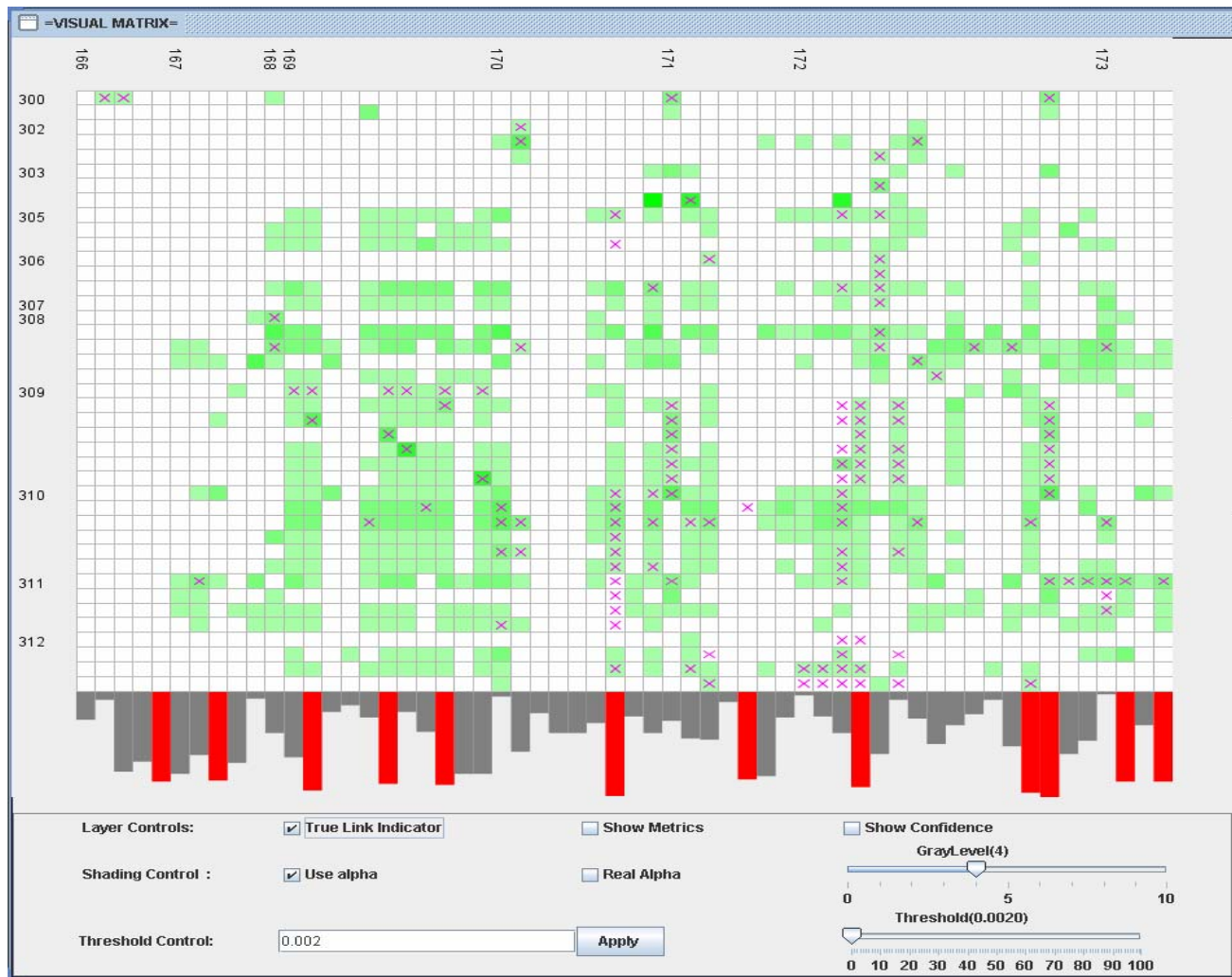


Figure 2. The trace matrix visualization of EBT data

[11]. In their approach, recovered features, which they describe as, micro-architectures responsible for specific functionality, are first extracted through static or dynamic analysis. The features are then rendered using 3D visualization techniques in which selected metric values of the classes within each feature are encoded by means of height, color, and twist. The paper also briefly discusses visualization depicting evolution of the architecture and constituents of features contributing to multiple functionalities; however it does not discuss how the extracted features relate to design quality of the classes, which will be analyzed in this paper.

3. Trace matrix visualization

Within the context of automated traceability, a user issues a query against a set of traceable artifacts and a set

of candidate links are generated. Each of these candidate links is represented by a probability score depicting the likelihood that the query and document are linked. Raw probabilities are then translated into confidence scores (ranging from 1% to 99%) representing the belief that a link is in fact a true link [12].

To visualize these links, VisMatrix adopts a 2D coordinate scheme to depict the relationship between queries and documents. The horizontal axis represents the set of traceable documents, the vertical axis represents the set of queries, and the degree of shading in each cell reflects the confidence value of the link. A cell is shaded more darkly to represent higher confidence levels, and more lightly for lower confidence levels.

Figure 2 shows the main interface of VisMatrix in which an RTM is visualized. The main view consists of three parts. The matrix view, which occupies the major

part of the screen, is comprised of shaded cells representing links between requirements on the vertical axis, and design/implementation artifacts on the horizontal axis. If a results set is available, either from a previously developed RTM or one that has been constructed dynamically through capturing user feedback during ongoing use of the tool, then the user can toggle a switch to display the correct links from the results set. VisMatrix superimposes correct links over the existing RTM and represents them as check marks. Check marks appearing on shaded cells represent links that have been correctly retrieved by the tool, while those on non-shaded cells represent true links missed by the tool. Furthermore, shaded cells with no check marks indicate falsely retrieved links. To improve visibility into large and complex systems, VisMatrix provides a zooming feature that enables the user to zoom into interesting parts of the RTM.

The labels shown on both axes represent the names of meaningful groups such as packages or IDs of requirements subtitles. Each of these groups, including the parent artifact, is termed a *family*.

A histogram is displayed under the matrix depicting a design metric selected by the user. The choice of metrics depends partially on what type of artifact is being traced. Throughout the remainder of this paper the visualization process is illustrated using the typical scenario of tracing requirements (or user stories) to design components. For this reason the metrics represent coupling and cohesion of elements in the design, e.g., LCOM (Lack of Cohesion in Methods) or RFC (Response for a Class). Finally, in the lower section of the screen a control panel allows the user to manipulate several aspects of the display. For example the user could create a lower confidence threshold to eliminate less likely links, or could select one of the available metrics to be displayed in the histogram.

4. Trace Performance Evaluation

Trace performance, defined as the ability to generate correct links, can be evaluated during an initial pilot study, in which the adoption of automated traceability techniques is being considered. It should also be evaluated on an ongoing basis to monitor the quality of traces being generated. Generally, trace performance is measured using the standard metrics of recall and precision, where recall measures the percentage of true links that are retrieved, and precision measures the percentage of retrieved links that are correct. During a process adoption initiative an organization may utilize an existing, manually constructed trace matrix for comparison purposes. Similarly during ongoing monitoring of the trace tool's performance, an RTM result set is constructed and validated through user feedback, and this can be compared with the set of automatically generated links.

As illustrated in Figure 2, the result set is transposed over the automated results. Problematic areas such as groups of correct links that are not retrieved by the tool or groups of falsely retrieved links can be easily identified. For example, in the EBT data at the bottom right of Figure 2, the retrieval algorithm missed half of the true links between the requirement family "Notification processing" (ID: 312) and the package "Subscriber" (ID: 172). An analysis of the data indicated that although the requirements were well-defined, the package contained poor domain information. For example, the class "Subscriber Listener" contained words such as *result*, *set*, *connection*, *socket*, *subscriber* and *manager*, most of which are related to the implementation of communication sockets and database objects. The lack of comments in the code or domain information embedded into variable and method names meant that the automated retrieval algorithm was unable to retrieve the right links.

The visualization of generated vs. correct links also is useful for helping the analyst to prune unwanted links that decrease the precision of the results. A pruned link generally reduces the probability of links occurring between artifacts in the two families [9]. Although *Poirot* can automatically prune links based on a partial RTM result set or on feedback collected from the user during the course of using the tool, the visualization process can improve the pruning results through eliciting input directly from the analyst. For example, in Figure 2, it is clear that false links are generated between the group of requirements under "Speculative events" (ID: 308), and the package labeled "Trigger" (ID: 169). These false links occurred because the same term was used to mean two entirely different things.

Problems identified through the trace visualization could lead to several process improvement initiatives such as increased use of a project glossary to minimize ambiguities; improved use of comments in code; or strengthening of hierarchical information through the use of more meaningful package names, titles, and subtitles etc. In this way, the visualization not only helps an organization evaluate a project's traceability, but also helps it to improve and monitor the traceability of its artifacts.

5. Trace Analysis

By considering the intersection between the generated traces and the design components, it is possible to evaluate several aspects related to the quality of the design.

5.1. Requirements Scope

In the trace matrix, each row represents the traceability scope of a single requirement in respect to

each of the design artifacts. Certain requirements exhibit focused traces into clusters of closely related classes while others tend to have a more dispersed impact. For example, in the Zodiac data set, the user story entitled to “Add a New User” (ID: 64) traces only into the package “ProjectControllerAction Strategies” (ID: 1217), while “View SIG” (ID: 56) traces to 26 out of the 37 possible packages. To differentiate between requirements with focused versus broad scope, a metric was defined called “Trace Scope” (TS) that captures the ratio of traced packages out of all packages. Furthermore to record the fact that in this case, the traces represent fan-out links, the metric is named TS-Out. TS-Out is determined by calculating the ratio of all packages linked from a requirement (or user story) over all possible packages. “Add a New User” therefore has a TS-Out score of $1/37$ (0.027), whereas “View SIG” has a TS-Out score of $26/37$ (0.703).

High TS-Out scores mean that the requirement impacts multiple packages across the architecture and could potentially either represent architecturally significant requirements exercising many dispersed classes, or cross-cutting concerns representing candidate aspects [13, 14]. Early identification of architecturally significant requirements can lead to better designs in which the architecture is carefully constructed to meet the stated requirements. Similarly early identification of aspects can result in greater system modularity which in turn can reduce the cost and effort involved in building and maintaining a system.

It should not be overlooked however that the generated traces represent candidate links, some of which might in fact be false. To minimize the potential impact of false positives, a higher confidence level cutoff can be established, so that only higher confidence links are considered during design evaluation.

The ten Zodiac user stories (requirements) scoring the highest TS scores are depicted in Table 1. Each of these was analyzed to determine if they did in fact

represent either architecturally significant requirements or potential aspects. Analysis suggested that all of these requirements were too high-level to be considered as aspects, however they all exercised multiple server side classes and often persistence subroutines, and could therefore be considered as being architecturally significant.

One of the many advantages of automated traceability over traditional approaches is that it supports traceability of new requirements or even of requests for change. If a new requirement were to be introduced to the system, and the automated trace query returned a high TS-Out score, then our observations concerning the correlation between high TS-Out values and architecturally significant requirements suggest that the new requirement should be managed carefully. Before real conclusions can be drawn here, additional work is still needed to evaluate the architectural significance of complete sets of requirements. Each requirement in these data sets will be manually evaluated for its architectural significance so that a full correlation analysis can be performed. We also need to analyze TS-Out scores for lower level derived requirements in order to look for patterns related to early aspects.

5.2. Design Modularity

The previous section discussed “fan-out” traces from the requirements side and their implications for discovering the scope of a requirement. In contrast, this section focuses on “fan-in” traces on the design side and explores two different ideas. The first idea is based on the concept that a well-modularized system exhibits qualities of high cohesion and low coupling [15, 16]. It investigates whether strong trace clustering is correlated with good modularity. The second explores whether a single class with multiple incoming links that originate from primarily unrelated requirements is likely to also exhibit low cohesion as measured by a traditional cohesion metric [15]. The real question here is whether patterns of generated traceability links provide indicators of risk in the design.

5.2.1. Trace Clustering. In a well-modularized system we would expect to see groups of functionally related requirements traced to one or two groups of design elements. For example, a set of requirements describing the functionality of a specific feature might primarily link to a small subset of classes that implement that feature. However, requirements that play architecturally significant roles, or represent cross-cutting concerns would not be expected to exhibit this same level of modularity in the trace matrix. To evaluate the modularity of a system, a new metric for measuring clustering among artifact families called “Trace Clustering” (TC) is proposed.

Table 1. User stories with high trace scope

User story	TS_Out
View SIG	0.703
Add user to project	0.405
Add/remove users to project	0.351
Retrieve a SIG from local workstation	0.324
Check Out SIG GUI	0.297
Check In SIG GUI	0.297
Evaluate a SIG goal node	0.297
Create a new SIG goal node	0.216
Create a new SIG operationalization goal	0.216
Open SIG	0.189

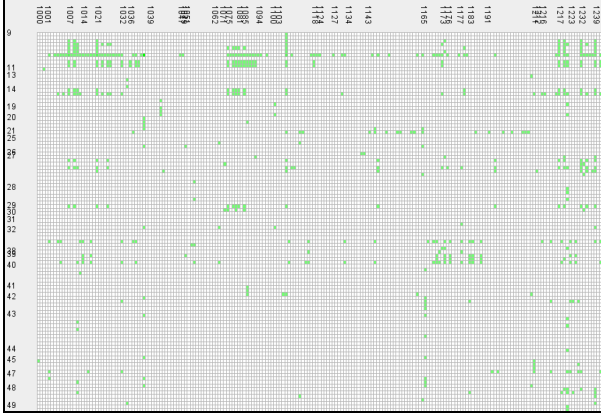


Figure 3. Zodiac data shows non-developed user stories

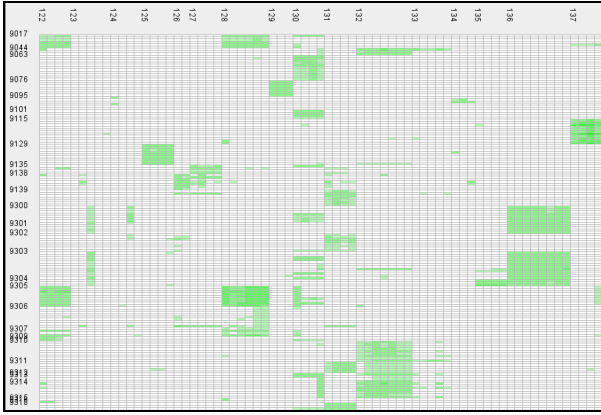


Figure 4. IBS data shows strong clustering

Formally, the Cartesian product of M requirement families $R = \{r_i \mid i=1,2,\dots,m\}$ and N design / implementation families $D = \{d_j \mid j=1,2,\dots,n\}$ produces $M*N$ blocks $B = \{b_{ij} \mid i=1,2,\dots,m; j=1,2,\dots,n\}$. Each block b_{ij} contains the links set $r_i \times d_j$ and the ratio of candidate links out of total possible links within it is defined as $C(b_{ij})$. The TC metric measures the number of blocks that exhibit cluster-like behavior over the number of blocks that contain any candidate links. Entirely empty blocks which could either represent unimplemented requirements, initial skeletal designs, or good clustering are not incorporated into this metric.

First a lower threshold t_1 is set to filter out empty (or close to empty) blocks. An upper threshold t_2 is then set to select blocks with high $C(b_{ij})$ values. Then a binary function to measure the clustering of a single block is

$$\text{defined as: } S(b_{ij}) = \begin{cases} 1 & \text{if } C(b_{ij}) \geq t_2 \\ 0 & \text{if } C(b_{ij}) < t_2 \end{cases}$$

and another to determine whether the block should be included in the calculation is defined

$$\text{as } T(b_{ij}) = \begin{cases} 1 & \text{if } C(b_{ij}) \geq t_1 \\ 0 & \text{if } C(b_{ij}) < t_1 \end{cases}$$

Finally the clustering measure of the modularity of the

$$\text{entire data set is represented as } TC = \frac{\sum_{1 \leq i \leq m, 1 \leq j \leq n} S(b_{ij})}{\sum_{1 \leq i \leq m, 1 \leq j \leq n} T(b_{ij})}.$$

Table 2 records the TC values for IBS, Zodiac, EBT, and LC data under a range of lower t_1 and upper t_2 threshold values. IBS data clearly returned the highest values at all reported threshold ranges, and this is supported by examining its trace matrix shown in Figure 4. In this matrix, it is clear that most of the links between requirement families and design families exhibit high clustering. This contrasts notably with the EBT and Zodiac matrices depicted in Figures 1 and 3 respectively. Our expert knowledge of the four data sets indicates that IBS is the most well organized and modularly designed system. This suggests that the visualized matrix and the TC metric together provide a useful measure of the modularity of a data set.

5.2.2. Lack of Cohesion. Cohesion describes the degree to which the different actions performed by a module contribute to a unified function [17]. We hypothesized that a class with low cohesion, measured using the Low Cohesion of Methods (LCOM) metric would tend to exhibit traces from multiple unrelated requirements, and would therefore be depicted in the visualized matrix as a column with multiple links originating from disparate requirements families. If such a correlation exists, the trace matrix could provide a simple and early technique for detecting cohesion problems as they develop in a design.

A class exhibiting high “spread” of incoming links represents a class with high TS-In scores. Just as high TS-Out scores are depicted in the matrix by a row of

Table 2. Trace Cluster (TC) metrics for four data sets

t_1	t_2	IBS	Zodiac	EBT	LC
0.1	0.6	0.45	0.18	0.21	0.21
0.1	0.7	0.41	0.14	0.16	0.18
0.1	0.8	0.38	0.09	0.04	0.18
0.2	0.6	0.55	0.24	0.24	0.25
0.2	0.7	0.51	0.18	0.18	0.20
0.2	0.8	0.47	0.12	0.04	0.13
Average		0.46	0.16	0.15	0.19

dispersed links, an element with a high TS-In value is depicted as a column with dispersed links. Although there are multiple reasons why a class might exhibit high TS-In values, one possibility is that it could exhibit low cohesion.

Finding an optimal LCOM metric is a controversial issue in the object oriented design metrics community. However, as the theoretical difference between LCOM metrics is not the focus of this paper, the traditional C&K LCOM metric proposed by Chidamber and Kemerer [15] was used. This metric is defined as the number of pairs of methods in a class that have no data fields in common, minus the number of pairs of methods in the class that share at least one data field. Higher metric values represent lower cohesion in the class. In cases of high cohesion where the value is negative, the value is set to 0.

To evaluate the hypothesis that low class cohesion correlates with high TS-In values, a correlation analysis was performed on TS-In and LCOM metrics for the Zodiac dataset. No significant correlation was found; moreover, the graph depicting TS-In versus LCOM metrics shown in Figure 5 suggested that classes with the highest LCOM values (ie poorly cohesive classes) clearly exhibited low TS-In values, while several of the classes with higher cohesion exhibited higher TS-In values. This was contradictory to our hypothesis.

However, further analysis of the classes with the highest LCOM values suggested that TS-In values might be useful in filtering out the clearly inaccurate results we observed from using the LCOM metric. Classes with high LCOM values yet low TS-In values may not in fact exhibit low cohesion. For example the class “QETMainWindow” scored a high LCOM value of 435 yet a relatively low TS-In score of 0.19. An observation of the class indicated that it contained a long list of GUI elements declarations and much skeletal code such as methods containing only signatures and no concrete body. The class was judged to be fairly cohesive. In contrast, QetAppClientManager scored high LCOM values and high TS-In values and contained disparate functionality representing socket communication, management of input objects, and manipulation of output objects.

These results are by no means conclusive, and clearly additional work is needed to understand if TS-In values can be used to filter cohesive classes out of a set of classes exhibiting high LCOM values. In fact the Zodiac system, which was used for this initial analysis, was relatively well designed with a thoroughly analyzed architecture and clearly defined functional modules. Additional experiments must be conducted using datasets exhibiting potentially greater problems with cohesion.

5.2.3. High risk classes. The VisMatrix tool provides the option of displaying standard complexity metrics such as Weighted Methods per Class (WMC), which typically sums the number of methods per class weighted by a

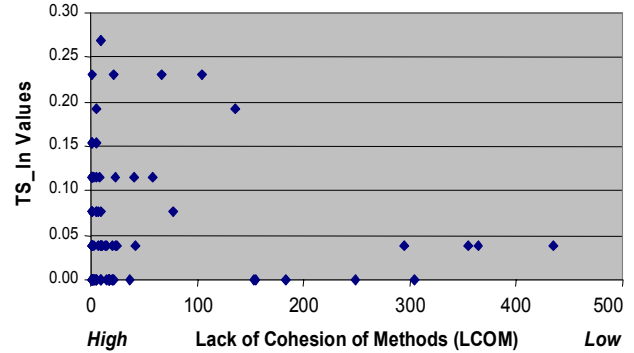


Figure 5. TS-In metric versus Lack of Cohesion

factor such as its Cyclomatic Complexity [18]. This is illustrated in the visualized matrix shown in Figure 6, which displays the complexity of classes within the Zodiac dataset.

The complexity control window allows the user to control the complexity or cohesion threshold for identifying risky classes. Classes above or below a user-defined threshold are identified as risky, and are shaded red in the histogram. Their associated columns are also outlined in red. Classes with high TS-In values that also exhibit high complexity can be earmarked as potentially risky classes. These classes are impacted by multiple requirements and therefore potentially subject to more changes. At the same time, their complexity suggests that implementing those changes may be more difficult.

The top 10 most complex classes and their TS-In values are shown in Table 3. The fact that none of them exhibit high TS-In values suggest that Zodiac may be relatively free of classes in this category.

5.3. Implementation status

The visualized traceability matrix also provides support for project management. When a requirement returns only low confidence links or few links in the matrix visualization (ie exhibits very low or zero TS-Out values), it may be an indication that the requirement has not yet been implemented or fully satisfied. For example, in the Zodiac data shown in Figures 3 and 6, there are several primarily blank rows that occur in the lower part of the matrix, and these in fact represent user stories that were not implemented at the time of this analysis.

6. Conclusions

Trace matrix visualization, used in conjunction with standard design metrics portrays useful information to an analyst during many different stages of the automated traceability life-cycle model. First it supports an organization in evaluating the effectiveness of automated traceability methods during either a pilot study or as part

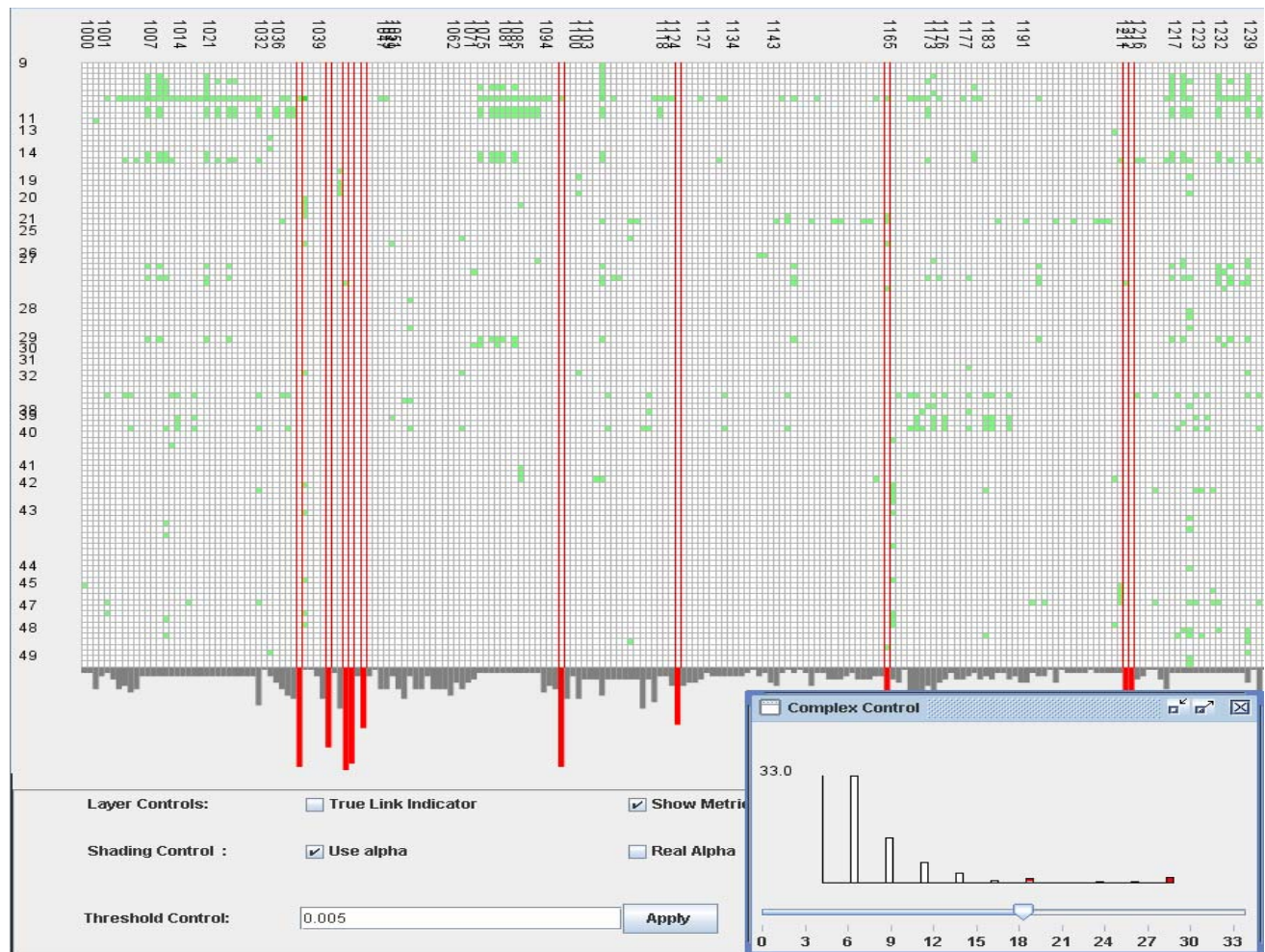


Figure 6. Zodiac matrix with exploratory histogram and complexity metrics marking

of an ongoing process monitoring effort. Secondly, it provides useful insight into the traceability relationships within a system, and can potentially warn developers of risky areas of the design. Finally it can be used to help managers more effectively assess the state of a project and understand the effort that might be involved in introducing a new requirement.

This paper represents an initial exploration of the use of trace visualization, and the related analysis of generated trace patterns. Several interesting hypotheses have been proposed and an initial examination of the results is reported in this paper. Future work will include a more extensive quantitative analysis of the results and a deeper analysis of the qualitative reasoning supportable by the visualization technique. Our work will also consider visualization scalability issues including the possible use of fisheye views [19].

6. Acknowledgements

The work described in this paper was partially funded by NSF grant CCR- 0306303.

7. References

- [1] O. Gotel, and A. Finkelstein, "An Analysis of the Requirements Traceability Problem", *Proceedings First Int'l Conf. Requirements Eng.*, 1994, pp. 94-101.
- [2] B. Ramesh, and M. Jarke, "Toward Reference Models for Requirements Traceability", *IEEE Trans. on Software Engineering*, Vol. 27, No. 1, Jan 2001, pp. 58-92.
- [3] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering Traceability Links between Code and Documentation", *IEEE Transactions on Software Engineering*, Vol. 28(10), 2002, pp. 970-983.

Table 3. Ten Zodiac classes scoring highest WMC Values

Class	WMC	TS-In
DrawServerCloud	33	0.0385
DrawCloud	32	0.0385
QETMainWindow	31	0.0385
DrawLine	31	0.0000
SigClientView	31	0.0385
DrawServerLine	28	0.0000
CloudName	25	0.0000
LineName	19	0.0000
CollaborationActionRequestFactory	18	0.0000
RemoteClientProxy	17	0.1923

[4] J. Huffman Hayes, A. Dekhtyar, and J. Osborne, "Improving Requirements Tracing via Information Retrieval", *11th IEEE International Requirements Engineering Conference*, Monterey, CA, Sept. 2003, pp. 138-150.

[5] J. Huffman-Hayes, A. Dekhtyar, S. Karthikeyan Sundaram, S. Howard, "Helping Analysts Trace Requirements: An Objective Look", *12th International Requirements Engineering Conference*, 2004, Kyoto, Japan, pp. 249-259.

[6] R. Settimi, J. Cleland-Huang, O. BenKhadra, J. Mody, W. Lukasik, and C. DePalma, C., "Supporting Change in Evolving Software Systems through Dynamic Traces to UML", *IEEE International Workshop on Principles of Software Evolution*, Kyoto, Japan, Sept. 2004, pp. 49-54.

[7] G. Spanoudakis, "Plausible and Adaptive Requirement Traceability Structures", *Proc. of the 14th International Conference on Software Engineering and Knowledge Engineering*, Ischia, Italy, 2002, pp. 135-142.

[8] J. Cleland-Huang, R. Settimi, O. BenKhadra, E. Berezanskaya, S. Christina, "Goal-Centric Traceability for Managing Non-Functional Requirements", *Intn'l Conference on Software Engineering*, St. Louis, USA, May 2005, pp. 362-371.

[9] J. Cleland-Huang, R. Settimi, C. Duan, X. Zou, "Utilizing Supporting Evidence to Improve Dynamic Requirements

Traceability", *Proceeding of the 13th IEEE International Requirements Engineering Conference*, Paris, France, Aug 2005, pp. 135-144.

[10] Andrian Marcus, Xinrong Xie, and Denys Poshyvanyk, "When and how to visualize traceability links?" *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, Long Beach, California, pp. 56-61.

[11] Giuliano Antoniol *et al.*, "On Feature Traceability in Object Oriented Programs", *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, Long Beach, California, pp. 73-78.

[12] X. Zou, R. Settimi, J. Cleland-Huang, C S. Miller, "Supporting Trace Evaluation with Confidence Scores", *Proceeding of REDECS05, workshop on Requirements Engineering Decision Support*, Paris, France, August 2005, pp. 1-6.

[13] A. Rashid, P. Sawyer, A. Moreira and J. Araujo, "Early Aspects: A Model for Aspect-Oriented Requirements Engineering", *IEEE Joint International Conference on Requirements Engineering*, 2002, pp. 199-202.

[14] A. Rashid, A. Moreira, and J. Araujo, "Modularisation and Composition of Aspectual Requirements", *2nd Intn'l Conf on Aspect-Oriented Software Development*, Boston, Massachusetts, 2003, pp. 11-20.

[15] Shyam R. Chidamber and Chris F. Kemerer, "Towards a Metrics Suite for Object Oriented Design", *OOPSLA '91*, pp. 197-211.

[16] Lee, Y., B. Liang, and F. Wang, "Some Complexity Metrics for Object-Oriented Programs Based on Information Flow", *Proceedings of CompEuro '93*, March 1993, pp. 302-310.

[17] Constantine Larry, and Edward Yourdon, *Structured Design*, Englewood Cliffs, N. J., Prentice-Hall, 1979.

[18] Thomas J. McCabe, and Arthur H. Watson, "Software Complexity", *Crosstalk, Journal of Defense Software Engineering* 7, 12, December 1994, pp. 5-9.

[19] Furnas, G. W., "Generalized fisheye views", *Human Factors in Computing Systems CHI '86 Conference Proceedings*, Boston, Massachusetts, 1986, pp. 16-23.