Extending Software Change Impact Analysis into COTS Components

Shawn A. Bohner
Virginia Tech, Dept. of Computer Science, Falls Church VA 22043
sbohner@vt.edu

Abstract

As software components and middleware occupy more and more of the software engineering landscape, interoperability relationships point to increasingly relevant software change impacts. Packaged software now represents over thirty-two percent of the software in most organizations. While traceability and dependency analysis frameworks have effectively supported software impact analysis in the past, they do not adequately addressed this trend. As software systems grow in size and complexity, their dependency webs of information also extend beyond most software engineers ability to comprehend them. This paper examines research for extending current software change impact analysis to incorporate interoperability dependency relationships for addressing distributed applications and explores three dimensional (3D) visualization techniques for more effective navigation of software changes.

1 Introduction

As software engineering practice responds to demands for web applications on heterogeneous platforms, software change is more and more influenced by middleware and Commercial-Off-The-Shelf (COTS) components [WWB01]. Interoperability dependency relationships now point to increasingly relevant software change impacts and necessarily drive the analysis. Software changes to systems that incorporate middleware components (e.g., web services) expose these systems and the organizations they serve to unforeseen ripple effects that frequently result in serious software quality issues [BOH99].

Since software functionality is routinely distributed across heterogeneous software and hardware platforms, complex interoperability issues necessarily govern any analysis of software changes. For example, when a custom software component is replaced by a COTS component, the internal complexity of the component (i.e., internal data and control dependencies) is replaced by a focus on interface and interoperability complexities (i.e., ability to exchange information, compatibility, and conformance to standards). Web services in today's distributed applications provide an excellent research platform to explore these shifts in complexity.

Existing software change impact analysis models have not adequately addressed this emergent trend towards middleware and COTS software. Software projects increasingly use middleware and gain important advantages including reduction in overall complexity. This gain is partially displaced by increased complexity for handling software changes as more interoperability dependencies are introduced. Moreover, as software systems grow in size and complexity, the dependency webs extend beyond most software engineers ability to comprehend them [HUT98]. Blinded by the barrage of information, many ripple effects of software change go unnoticed until they are manifest in fragile designs or system failures. Impacts range from software defects inadvertently injected during maintenance activities to more sinister information assurance vulnerabilities (e.g., tampering, denial of service).

More generally, the software community faces growing needs for technology to better predict and control the effects of software changes, particularly in the area of web development. Not only are software systems growing in size and complexity, they are increasingly incorporating COTS software components to deliver requisite functionality to the user via web and other application services [WWB01] [BOH98]. These trends tax software engineers' abilities to respond to software change and lead to vulnerable information, less disciplined development, shorter operational life, and higher total cost of critical software products.

This research couples proven software change impact analysis and three dimensional (3D) visualization approaches on this critical problem. Software change impact analysis technology must be expanded to address middleware and the integration of COTS software components. It is essential that mechanisms be developed to convey impact information more effectively for navigating software changes.

With both component service support and an extensible markup language (XML) that incorporates declarative statements about interoperability dependencies, this platform embodies all of the factors that face the impact analysis challenge. There is an opportunity to use the formalisms present in web service representations (i.e., XML) to extract and analyze interoperability dependencies, which will in turn provide requisite impact information for middleware and COTS components. By extending the analysis to this growing area, we enable software engineers, architects, and project managers to make better decisions about software changes, preserve the software quality, and increase the life of systems while lowering the total costs over their operational life.

Software change impact analysis determines possible effects of proposed software changes [ARN93] [LEE00]. While software change impact analysis is not yet formally a part of many software processes, after Year 2000 date conversion efforts, most software maintainers now employ some means of identifying impacts prior to making extensive software changes [ARN95] [BOH99].

Impact analysis for most large software systems is a labor-intensive, often manual process, performed only when absolutely necessary due to the cost involved – effectively limiting the quality, consistency, and number of changes that can be made to a software system [BOH95][LIN98]. Since software functionality is routinely distributed across heterogeneous software and hardware platforms, complex interoperability issues necessarily govern any analysis of software changes.

1.1 Software Change

The key to effective software change is software comprehension. This applies equally to adaptive, perfective, and corrective maintenance activities [SWA89]. Unlike many other types of products, a software product is intended to be malleable. A solution is implemented in software when it is expected to evolve or change periodically; software can be changed incrementally and adapted as the environment changes around it. Although software neither deteriorates nor changes with age, most of software maintenance involves change that potentially degrades the software unless it is proactively controlled.

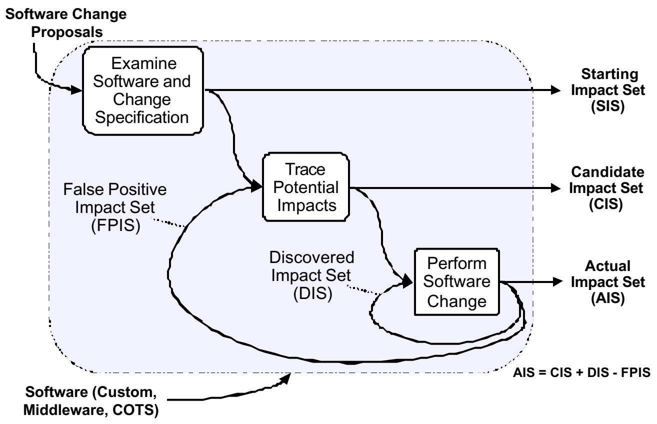
Today, much of the concern has more to do with the complexity and shear size of current applications than it has to do with change [KEM99]. As we developed larger and larger software systems incorporating more features and newer technology, the need for new impact analysis technology has emerged [BOH95] [ARN95].

Changing requirements are endemic to software [DAV89][LEH94]. Continuously changing requirements represent a key management challenge [SWA89]. Final requirements seldom exist for software systems since they are continually being augmented to accommodate changes in user expectations, operational environment, and the like. Therefore, many software systems are never really complete until their function in the organization becomes obsolete.

Basic software change activities are: understanding software with respect to the change, implementing the change within the existing system, and retesting the newly modified system. Each of these activities has some element of impact determination. To understand the software with respect to the change, we must ascertain parts of the system that will be affected by the change and examine them for possible further impacts. While implementing the change within the existing system, we need to be aware of ripple-effects caused by the change and record them so that nothing is overlooked. Once the change has been designed and implemented, we need to find existing test cases for regression testing and test cases that may need to be re-examined for redesign based on new requirements. Without requisite change impact analysis and management mechanisms, software changes during maintenance can have unpredictable consequences that often delay their implementation.

2 Software Change Impact Analysis

Software Change Impact Analysis is defined as "The determination of potential effects to a subject system resulting from a proposed software change" [ARN93] [BOH95]. Figure 1 depicts the basic software change impact analysis process and its iterative nature.



Software Change Proposals
Examine Software and Change Specification
Trace Potential Impacts
Perform Software Change
Starting Impact Set (SIS)
Candidate Impact Set (CIS)
Actual Impact Set (AIS)
False Positive Impact Set (FPIS)
Discovered Impact Set (DIS)
AIS = CIS + DIS - FPIS
Software (Custom, Middleware, COTS)

Figure 1: Software Change Impact Analysis

The starting impact set (SIS) is the initial set of objects thought to be affected by a change. The SIS is normally determined while examining the change specification. The candidate impact set (CIS) is the set of objects estimated to be affected. The CIS is produced while conducting the impact analysis. The actual impact set (AIS) is the set of objects actually modified. The AIS is not necessarily unique, as a change can be implemented in several ways.

The impact analysis process is iterative and discovery in nature. While a change is being performed, there are likely to be more impacts discovered. The discovered impact set (DIS) represents an under-estimate of impacts. The false-positive impact set (FPIS) represent the over-estimate of impacts in the analysis. The CIS plus additions of the DIS and minus deletions of FBIS should represent the AIS. The accuracy (error) is obtained by adding the number of impacts in the DIS and FPIS then dividing them by the number of impacts in the CIS. This represents the number of errors (DIS and FBIS) divided by the estimate. The objective of the analysis is to have the Candidate Impact Set produced from tracing potential impacts (manually or with automation) as close to the Actual Impact Set as possible by identifying true impacts while eliminating false-positives.

The basic principle underlying the need for impact analysis is that a small change in a software system may affect many other parts of the system, thus causing a ripple-effect. Other parts of the system which are affected often are not obvious or easy to detect. Ripple-effects may cause direct or indirect impacts based on other software life-cycle objects (SLOs) that may depend upon it.

- A direct impact – occurs when the object affected is related by one of the dependencies that fan-in/out directly to/from the SLO. This type of impact is also called a first level impact and can be obtained from the connectivity graph of a dependency matrix.
- An indirect impact – occurs when the object affected is related by the set of dependencies representing an acyclic path between the SLO and effected object. This type of impact is also referred to as an N-level impact where N is the number of intermediate relationships between the SLO and the effected object. This can be obtained from the reachability graph of a dependency matrix.

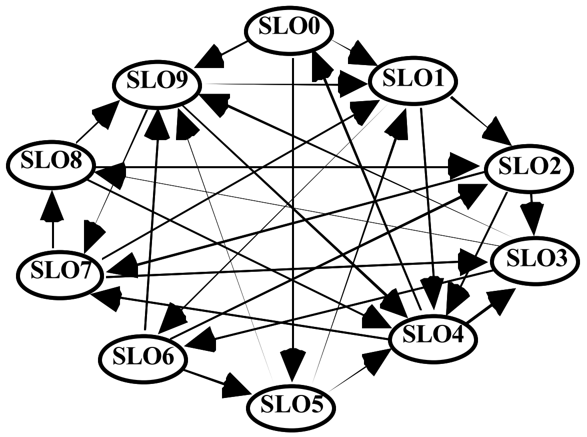


Figure 2: Simple Directed Graph of SLOs

Software Change impact analysis approaches focus on both structure and semantics. First, taking the analysis down to its essential structural level, Figure 2 illustrates a simple directed graph of ten SLOs numbered 0 to 9. Each of these SLOs represents a software artifact connected to other SLOs by unlabeled relationships (reflecting no semantic information). These artifacts can be anything from a variable in a program module to a requirements statement. Each of the arrows represent some dependency relationship between two SLOs. For purposes of this initial discussion, we will assume no semantic information on these relationships.

2.1 Structural Analysis

Note that SLO1 has a "direct" impact from SLO9 and that SLO8 has an "indirect" impact. The fan-in or in-degree of an SLO node indicates the number of known SLOs that depend on that particular SLO. For example, the in-degree of SLO1 is 4. This is a relatively high in-degree when compared to SLO2 with an in-degree of 2 but not as high as SLO4's in-degree of 5.

In this contrived case, SLO1 has an out-degree of 3 and is dependent on SLO2, SLO4, and SLO6. For purposes of this example, all the SLOs have an out-degree of 3 (unlikely under normal circumstances).

Taking the SLOs and the relationships depicted in Figure 2, we can construct the connectivity matrix described in Table 1. Reachability graphs can indicate potential impacts to a SLO but they are prone to over-estimating the impact. That is, reachability graphs used in software contexts tend to indicate that all objects are related to each other unless some constraining mechanism is employed. For example, transforming the connectivity matrix in Table 1 to the reachability matrix in Table 2 gains no additional information since the matrix is completely filled in. Therefore, techniques to guide detection of impacts to more likely choices are needed.

SLO=>	0	1	2	3	4	5	6	7	8	9
SLO0		X				X				X
SLO1			X		X		X			
SLO2				X	X			X		
SLO3							X		X	X
SLO4	X			X				X		
SLO5		X			X					X
SLO6			X			X				X
SLO7		X		X					X	
SLO8			X		X					X
SLO9		X			X			X		

Table 1: Connectivity Matrix of Relationships

This sparse matrix of connections can be transformed into reachability graphs using transitive closure algorithms such as Warshall's algorithm. These reachability graphs indicate the objects that can potentially be affected by a change to a particular SLO. Further still, they offer some indication of the paths necessary to navigate the change. However, simply detecting all potential relationships can generate a great deal of false-positive identifications.

SLO=>	0	1	2	3	4	5	6	7	8	9
SLO0		X	X	X	X	X	X	X	X	X
SLO1	X		X	X	X	X	X	X	X	X
SLO2	X	X		X	X	X	X	X	X	X
SLO3	X	X	X		X	X	X	X	X	X
SLO4	X	X	X	X		X	X	X	X	X
SLO5	X	X	X	X	X		X	X	X	X
SLO6	X	X	X	X	X	X		X	X	X
SLO7	X	X	X	X	X	X	X		X	X
SLO8	X	X	X	X	X	X	X	X		X
SLO9	X	X	X	X	X	X	X	X	X	

Table 2: Reachability Matrix of Relationships

One way to constructively limit the detection of potential impacts is to add another degree of structural information to the analysis. Indirect impacts suggests that there exists a distance between SLOs. For example, in Table 3 we introduce the notion of distance to the analysis. Notice that SLO0 (first row of the matrix) has a 3 in the SLO8 column. This means that SLO8 is three SLOs from SLO1 based on the available relationships.

SLO=>	0	1	2	3	4	5	6	7	8	9
SLO0		1	2	3	2	1	2	2	3	1
SLO1	2		1	2	1	2	1	2	3	2
SLO2	2	2		1	1	3	2	1	2	2
SLO3	3	2	2		2	2	1	2	1	1
SLO4	1	2	3	1		2	2	1	2	2
SLO5	2	1	2	2	1		2	2	3	1
SLO6	3	2	1	2	2	1		2	3	1
SLO7	3	1	2	1	2	3	2		1	2
SLO8	3	2	1	2	1	3	3	2		1
SLO9	2	1	2	2	1	3	2	1	2	

Table 3: Relationships with Distance Indicators

This distance concept enables us to describe the explosion of impacts problem without some level of search constraints. Figure 3 illustrates how the impacts number goes up dramatically.

Starting with only 3 impacts at a distance of one, jumping then to 10, 115, 1132, and 46203. This is only with a fan-out of approximately nine relationships per SLO. Moreover, the distance threshold technique is predicated on the weak assumption that if direct impacts have high potential for being true, then those farther away will be less likely. This leaves a large range of variability in the analysis.

2.2 Semantics in Impact Analysis

With only so much leverage with structural information, impact analysis must necessarily employ additional semantic information to increase the accuracy by finding more valid impacts and reducing the number of false-positives. Since object-oriented translates real-world situations into software design and implementation representations, it makes sense to exploit this aspect in identifying software impacts in object-oriented systems.

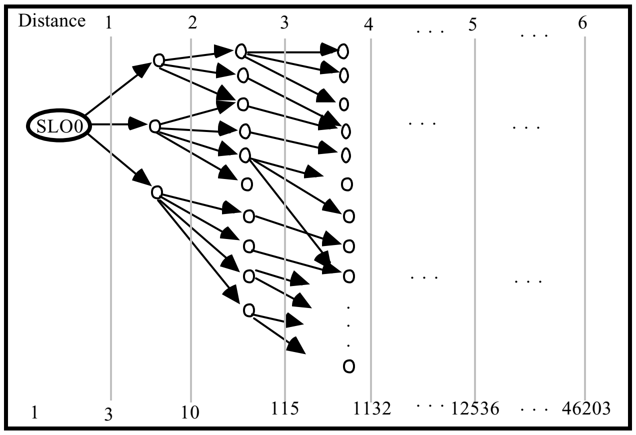


Figure 3: Impacts Explosion without Semantics

Basically, semantics are addressed in two forms. The first is using the design semantics embedded in the dependency or traceability relationships to augment the constraints provided by the graph analysis. There can be advantage in selecting a view of only design relationships or decomposition relationships if it has something to do with the type of change that is being analyzed. The second is to take advantage of the labels associated with the relationships (links) and SLOs (nodes). With classes, objects, messages, operations, and the like, there are hosts of semantics that can be used to guide the search for candidate impacts. By simply extracting the nouns (data objects) and the verbs (functions or messages) the analysis can be focused down to a more specific level.

There are three challenges that make software change impact analysis difficult today. *Information source volume* – there are a great number of relate software information sources to analyze. *Change semantic* – methods for describing meaningful software change relationships are limited for the range of software artifacts. *Analysis methods* – methods of analyzing the software workproduct dependency and traceability structures have not been fully explored for the growing areas of software workproducts (i.e., web services and COTS components). Impact analysis methods are based on search algorithms and can be classified as follows:

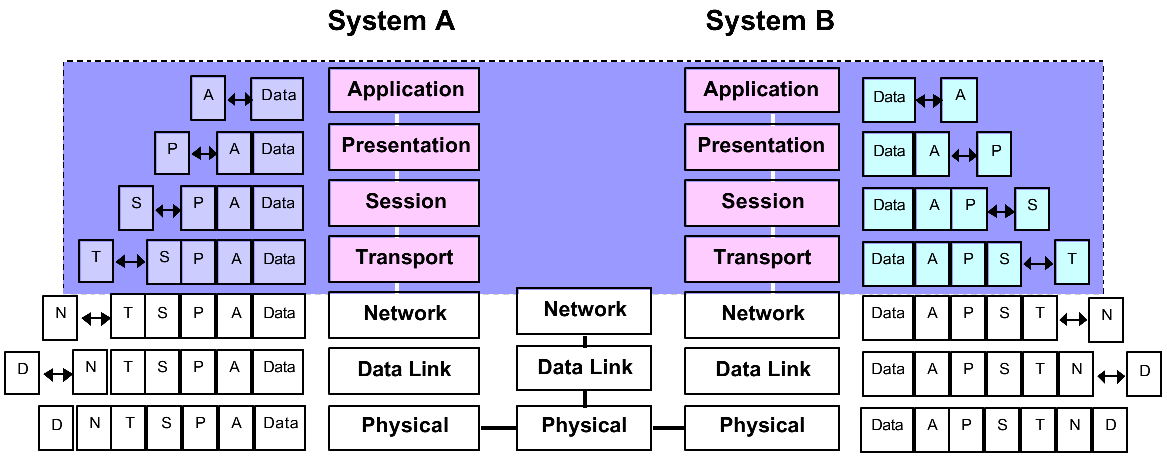


Figure 4: Software Change Impact Analysis in OSI Interoperability Model

- Unguided/Exhaustive: identify impacts in a "brute force" manner (e.g., transitive closure algorithms).
- Semantically Guided: impact analysis directed by predetermined semantics of objects and relationships – semantic inferencing [MOR90] and program dependence graphs [POD].
- Heuristically Guided: impact identification is directed by predetermined rules or heuristics to suggest possible paths to examine or dubious ones to avoid– truth maintenance [DHA88] and program dependence graphs [LOY93] [HAR93].
- Stochastically Guided: impact determination is guided by probabilities derived for the situation – classification patterns [PAL92] [GOT94].
- Hybrid Guidance: identify impacts using a combination of the above – program slicing [HUT98] and ripple-effect analysis [TUR94].

Each of these categories of approaches has their merit and warrants a more comprehensive treatment than this paper can offer in the space provided. However, suffice to say, the most promising impact analysis approaches have employed both structural and semantic techniques together to yield the best results [BOH95] [BOH96a][LEE00]. The natural next step is to incorporate middleware and COTS components.

3 Extending the Analysis to Middleware

From the software change impact analysis perspective, interoperability refers to the ability of two or more components to interact and exchange data according to a prescribed method with predictable results. Levels of interoperability range from no interoperation all the way up to a formal protocol for interaction. Figure 4 illustrates all of the interoperability levels of the OSI Interoperability model. However, for purposes of software change impact analysis, the best leverage can be gained from application, presentation, session, and transport levels since it is at these levels that middleware efforts like XML, Web Services, and COTS components are integrating.

Middleware can be defined as a distributed system service that includes standard programming interfaces and protocols [BER96]. Web services exemplify this by providing a middle layer between the operating system (and network) and the application software. Focusing impact analysis on web services, there is a range of interoperability dependencies that are not apparent during software changes. For example, an information exchange component (e.g., an XML to PDF translator) used in a web application may have version compatibility dependencies that conflict as web applications are integrated [BOH01]. Simple Object Access Protocol (SOAP) and Universal Description Discovery Integration (UDDI) along with their underlying Extendible Markup Language (XML) representations provide a wealth of the relevant interoperability dependence information that can be analyzed to provide better visibility into potential ripple effects [SCH02]. Figure 5 depicts the key levels of interoperability within today's web services.

Web services represent a key component of emerging system development strategies as companies attempt to achieve reasonable reuse levels of canonical services and contain costs of developing web-enabled applications. Web services is becoming a dominant information infrastructure approach for planned systems, despite the large base of Electronic Data Interchange (EDI) based systems [BOH01]. While the benefits of web-enabled EDI are certainly appealing, there is even more promise as industry and the public sector organizations use XML for integrating standards-based components and component interfaces.

These compelling trends provide evidence that this research has a growing constituency. Moreover, as the levels of interoperability increase, the opportunity to use impact analysis techniques to support changes also increases. The fact that interoperability dependencies are recorded in a formal representation [SCH01] that can be analyzed with traditional parsing technology makes possible an effective impact analysis. This public interoperability framework should serve to allow current impact analysis models to be augmented, extending the types of impacts that can be made visible. Figure 5 illustrates known representations for obtaining interoperability dependencies [UDD00].

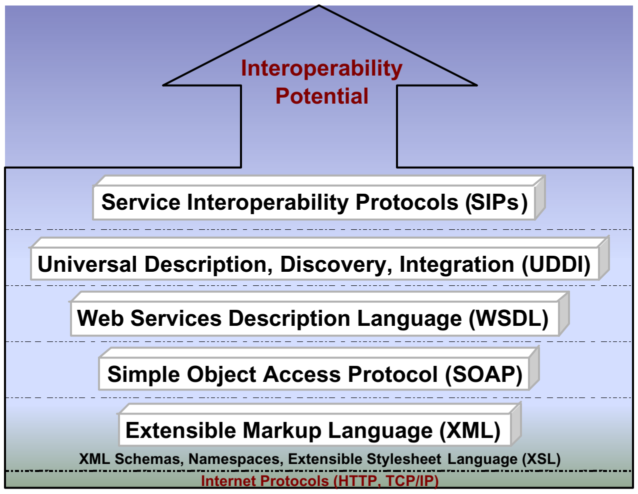

Interoperability Potential
Service Interoperability Protocols (SIPs)
Universal Description, Discovery, Integration (UDDI)
Web Services Description Language (WSDL)
Simple Object Access Protocol (SOAP)
Extensible Markup Language (XML)
XML Schemas, Namespaces, Extensible Stylesheet Language (XSL)
Internet Protocols (HTTP, TCP/IP)

Figure 5: Interoperability Dependency Levels (Interoperability Stack)

Since web services incorporate and integrate COTS components, dependencies must be defined for both integration and compatibility. Since web services are based on XML, a declarative language, with supporting representations for XML Schema and the Extensible Style Sheet Language (XSL). Moreover, the next generation web services will increasingly incorporate Service Interoperability Protocols for facilities like Globally Available Services. All of this information and more must be incorporated into an impact analysis model to provide effect visibility into the impacts of change.

While this research into extending the software change impact analysis model to incorporate middleware and components is preliminary, we are already observing some promising patterns. For example, with all of the XML based language and facilities, parsers are readily available and considerable experience has been reported that indicates that XML dependency analysis will mirror work that has already been completed with other languages. We are experimenting with some of the open source products from the Apache XML Project [APA02] for analyzing XML related products for interoperability dependencies. The downside of the plethora of information that can be generated for interoperability dependencies is that it will further complicate an already complex dependency model for current impact analysis.

4 Visualizing Impacts

The goal of this research is to provide support to improve the performance of software engineers making changes to web-enabled software by providing better visibility into the dependencies among software life cycle objects encountered in the software change process. Our technical approach has two key objectives.

1. To extend existing software change impact analysis research from source code dependencies and traceability relationships, to incorporate interoperability information for web services, COTS components, and packaged software applications.

2. To provide effective mechanisms for navigating software changes using promising 3D visualization techniques currently employed on the World Wide Web [MUN95] [BEN99].

Extending the software change impact analysis model introduces more dependency relationships – complicating an already complicated process [HUT98]. The second prong not only compensates for this additional load, it provides a more effective platform for software architects, software designers, and software engineers engaged in software change activities to visualize software from the dependency perspective.

The 3D visualization will be implemented using a virtual environment where software components and their dependencies will be handled similar to displaying WWW components in a virtual environment [GRA99]. We will incorporate the software dependency information into a virtual environment space to represent the various features of the software in a spatial environment. The representation will take advantage of the added dimension of the 3D space, allowing the user to easily visualize the information in much the same what as the technology is being used to visualize and navigate the WWW [BEN99].

5 Conclusions

While systems increasingly use web services, integrate COTS components and packaged applications, current software change impact analysis approaches have not addressed this key concern. Moreover, as software systems grow in size and complexity, the dependency webs of information extend beyond most software engineers' ability to comprehend them.

This research triangulates proven software change impact analysis with 3D visualization techniques to address this emerging middleware problem. We are expanding software change impact analysis technology to address the web services and the integration of COTS software components through interoperability dependency relationships.

Unfortunately, the extended model adds more complexity to an already complex analysis for software changes. Therefore, it is also essential that technology be developed to convey impact information more effectively for navigating software changes. This research addresses these concerns by developing a series of models and prototype impact analysis environments focused on increasingly more difficult portions of the impact analyses within web service supported applications. By applying 3D visualization techniques successfully employed on World Wide Web (which is analogous to the software change impacts visualization problem), this approach will clarify the volumes of dependency relationships that are endemic to the software change process.

While this research serves to advance software change impact analysis in the middleware arena, the techniques developed should also support impact analysis and software engineering community at large. By providing better visibility that reveals software change impacts and eliminates false positives, this research will support software project estimation, software risk analysis, and software architecture trade-off analyses.

References

[APA02] Apache Project Group, “The Apache XML Project,” Apache XML Project Team website, http://xml.apache.org/, February 2002.

[ARN93] Arnold, R. S., and Bohner, S. A., "Impact Analysis - Towards A Framework for Comparison," *Proc. of the Conf. on Software Maint.*, pp. 292-301, Sept. 1993.

[ARN95] Arnold R. S., "Millennium Now: Solutions for Century Data Change Impact", *Application Development Trends*, January 1995, pp.60-66.

[BEN99] Benford, S., Taylor, I., Brailsford, D., Koleva, B., Craven, M., Fraser, M., Reynard, G., and Greenhalgh, C., “Three Dimensional Visualization of the World Wide Web,” ACM Computing Surveys, Vol. 31, No. 4, December 1999.

[BER96] Bernstein, P., “Middleware: A Model for Distributed Systems Services,” Communications of the ACM, Feb. 1996.

[BOH95] Bohner, S. A., "A Graph Traceability Approach to Software Change Impact Analysis," Ph.D. Dissertation George Mason University, Fairfax, VA, 1995.

[BOH96a] Bohner, S. and Arnold, R., book entitled, "Software Change Impact Analysis", for the IEEE Computer Society Publications Tutorial Series, 1996.

[BOH96b] Bohner, S., "Impact Analysis in the Software Process: A Year 2000 Perspective," International Conference on Software Maintenance, 1996.

[BOH98] Bohner, S., “Packaged Applications: Off the Rack or Pay the Tailor?” META Group Research Publication, Stamford, CT, April 1998.

[BOH99] Bohner, S. A., “Year 2000 Speak Easy – Software Failures in Hiding,” META Group Research Publication, Stamford, CT, November 1999.

[BOH01] Bohner, S., “DFAS on the XML Road to Better Interoperability,” White paper for Defense Finance and Accounting Service, Crystal City, VA, August 2001.

[DHA88] Dhar V. and Jarke M., "Dependency Directed Reasoning and Learning in Systems Maintenance Support," IEEE Transactions on Software Engineering, Vol. 14, No. 2, February 1988, pp. 211-227.

[DAV89] Davis, A., Software Requirements: Analysis and Specification, Prentice-Hall, New Jersey, 1989.

[GOT94] Gotel O. C. and Finkelstein A. C., "An Analysis of the Requirements Traceability Problem," Proceedings of First Conference on Requirements Engineering," April 1994.

[GRA99] Gracanin, D. and K. E. Wright, “Virtual Reality Interface for the World Wide Web.” In Proceedings of the 15th Twente Workshop on Language Technology: Interactions in Virtual Worlds, University of Twente, Enschede, The Netherlands, pp. 59-68, 1999.

[HAR90] Hartmann, J. and D. Robson, ."Techniques for Selective Revalidation," *IEEE Software*, January 1990.

[HUT98] Hutchins, M. and Gallagher, K. B., "Improved Visual Impact Analysis," International Conference on Software Maintenance, 1998.

[KEM99] Kemerer, C. and Slaughter, S., "An Empirical Approach to Studying Software Evolution," *IEEE Trans. on Software Engineering*, Vol. 25, No. 4, July/Aug. 1999.

[LEE00] Lee, M., Offutt,J., and Alexander, R., “Algorithmic Analysis of the Impacts of Changes to Object-Oriented Software,” Technical Report ISSE-TR-00-02, George Mason, University, Department of Information and Software Engineering, May 2000.

[LEH94] Lehman M. M., "Software Evolution," *Encyclopedia of Software Engineering*, 1994, pp. 1202-1208.

[LIN98] Lindvall, M. and Sandahl, K., “How well do Experienced Software Developers Predict Software Change?”, Journal of Systems and Software, Vol. 43, No.10, 1998.

[LOY93] Loyall J. P. and Mathisen S. A., "Using Dependence Analysis to Support Software Maintenance," Proceedings of Conference on Software Maintenance, September 1993.

[MOR90] Moriconi M. and Winkler T. C., "Approximate Reasoning About the Semantic Effects of Program Changes," IEEE Transactions on Software Engineering, Vol. 16, No. 9, September 1990, pp. 980-992.

[MUN95] Munzner, T. and Burchard, P., “Visualizing the Structure of the World Wide Web in 3D Hyperbolic Space,” Proceedings of the VRML’95, San Diego, CA, pgs. 33-38, December 1995.

[PAL92] Palmer, J. D., and Liang, Y., "Indexing and Clustering of Software Requirements," Information and Decision Technologies, Vol. 18, No. 4, April 1992.

[POD90] Podgurski, A. and L. A. Clarke, "A Formal Model of Programming Dependencies and its Implications for Software Testing, Debugging, and Maintenance," IEEE Transactions on Software Engineering, Vol. 16, No. 9, September 1990.

[SCH02] Schlimmer, J. et al., “Web Service Description Requirements,” Web Service Description Working Group report, http://www.w3.org/TR/ws-desc-reqs/, February 2002.

[SWA89] Swanson E.R. and Beath C., "Maintaining Information Systems Organizations," Willey Publishing Company, 1989.

[TUR94] Turver, R. J. and Munro M., "An Early Impact Analysis Technique for Software Maintenance," Journal of Software Maintenance: Research and Practice, Volume 6, pp. 35-52, 1994.

[UDD00] UDDI Project Team, “UDDI Technical White Paper,” Universal Description, Discovery, and Integration project, http://www.uddi.org/pubs /, September 2000.

[WWB01] 2001 World Wide Benchmark by Rubin, H.,, META Group, Inc. Stamford, CT, 2001.