

An Agile Approach to Capturing Requirements and Traceability

Christopher Lee, Luigi Guadagno, Xiaoping Jia
School of Computer Science, Telecommunications, and Information Sciences
DePaul University
Chicago, IL
cle3@students.depaul.edu, lguadagno@cti.depaul.edu, xjia@cti.depaul.edu

Abstract

Agile methodologies are gaining popularity quickly, receiving increasing support from the software development community. Current requirements engineering practices have addressed traceability approaches for well defined phase-driven development models. Echo is a tool-based approach that provides for the implicit recording and management of relationships between conversations about requirements, specifications, and subsequent design decisions. By providing a means to capture requirements in an informal manner and later restructure the information to suit formal requirements specifications, Echo aims to solve the problems of applying traditional requirements engineering practices to agile development methods making available the demonstrated benefits of requirements traceability – a key enabler for large-scale change management.

1. Introduction

A study by the Standish Group showed that thirty-one percent of software development projects are cancelled before they are completed [7]. Many development methodologies have suggested strategies for on-time software delivery, but due to a variety of issues [8], projects are often late, over-budget, or cancelled. Agile methods aim to improve the software development process by promoting the use of the industry's best practices designed to embrace change and promote continuous delivery. To keep up with the fast pace of business, software projects must handle the frequently changing goals and needs of the customer. To do so implies echoing the voice of the customer through all phases of development, making design decisions traceable to their requirements specifications and ultimately the conversations that created them. Therefore, traceability is a key enabler in maintaining the customer focus during the software development process. Furthermore, traceability enables the rapid assessment and impact of change. Again, agile development strives to embrace change, which can

become challenging when dealing with large scale or distributed software development efforts.

For traceability to be successful in agile projects, it must start as early as possible. The method for creating and maintaining traceability must be non-intrusive to the development team by leveraging work already in progress during the requirements elicitation process. Echo is a tool-based approach to requirements engineering and traceability in agile projects. Echo creates the traceability web between customer needs and specified solutions by providing a means to capture conversations with customers and structuring them into requirements artifacts.

This paper will first describe the background of the work, and the need to remedy specific pains of requirements gathering. A description of the problem will then discuss key elements that struggle to integrate with agile development. Following the description of the problem, the paper will propose an approach and key enabling elements for gathering requirements and maintaining traceability leveraging the agile best practices. The paper will then present a summary of the current status of the Echo implementation, as well as plans for future work and validation of the approach. Finally, the paper will conclude by relating the approach back to agile requirements engineering and restate the drawn conclusions.

2. Background

2.1. Requirements Engineering

The software development industry has increasingly placed greater importance on ISO 9001 certification and process maturity models such as the SEI maturity model [25]. The ability to trace through the artifacts of a product lifecycle, source code, acceptance tests, requirements, and design rationale, is critical to the success of large complex projects. Requirements engineering attempts to communicate the ideas and needs of the product's stakeholders to the engineers and developers who ultimately build the product. While stakeholders generally know the functions and features the software should include, they have difficulty

quantifying the fine grain details and behavior of a system. They make the mistaken assumption that the high-level description of the features implies the detailed procedural steps [26]. The details of the system can often be elicited through interviews and discussions between the stakeholders and the software development team [9, 27].

The details of software development techniques and requirements ranges greatly from project to project, and there is not a single requirements engineering specification that covers each project perfectly. Each company, development team, and software developer needs to tailor the requirements engineering process to suit their needs [19]. However, the premise behind requirements engineering remains the same, to document stakeholder requirements, design rationale, and traceability to improve the quality of software development and the efficiency of software upgrade and maintenance.

Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction [24]. Gotel and Finkelstein show that while many techniques have been presented to solve traceability issues, the problem itself has different definitions and fundamental conflicts among practitioners. Application of requirements traceability practices towards agile development practices must begin with an initial effort to capture the critical conversations between developers and stakeholders, driving the creation of relationships between design rationale and requirements.

2.2 Agile Requirements Gathering

Agile methods have emerged around the belief that change is constant and hence promote software development practices that support frequently changing requirements, plans, and deliverables. As an example, Extreme Programming employs the use of unstructured requirements gathering techniques referred to as User Stories. Stories are customer written descriptions of system functionality and behavior. Some teams may group stories into features to identify larger clusters of functionality [11].

Agile requirements, features and stories, are generally written on index cards. When the team has the real estate to do so, the cards are pinned to a wall in some arrangement. The cards may be organized in some fashion, for example, by priority, risk, cost, or dependencies. The main purpose of the cards is to serve as vehicles, driving discussion with the customer about the system functionality and expected behavior. Often referred to as “promises for conversation”, stories, periodically revisited by the team, bring the focus back to the customer’s needs. Class Responsibility Collaborator (CRC) cards and Customer Acceptance

Tests (CAT) are the resulting artifacts of these conversations, which are then attached to the story card.

However, the conversations occurring as a result of the stories are seldom captured – they are retained in the collective memory of the team. This behavior brings out the most commonly discussed weakness of agile practices, a natural limit to scale for large and distributed projects [3]. In large, complex projects, developers often need access to the same information concurrently. In distributed projects, distribution of the information becomes difficult without the assistance of technology [13]. Communication and interaction lies at the heart of agile practices, however, as projects become increasingly larger and distributed, maintaining effective and up to date communication becomes error prone, work intensive and ultimately unmanageable.

3. Challenges of Requirements Engineering and Traceability in Agile Development

From requirements to artifacts, there is an elicitation and elaboration process that is often overlooked in the requirements gathering and development processes and critical to the scalability of large software projects. During this requirements elicitation process, collaboration and communication lead to an increase in understanding of the problem-space and the formulation of insights often critical to the success of the project. Without a proper mechanism to capture and organize this influx of information and its elaboration into insights, design rationale and traceability models cannot attain their full potential impact in the delivery of customer satisfying solutions.

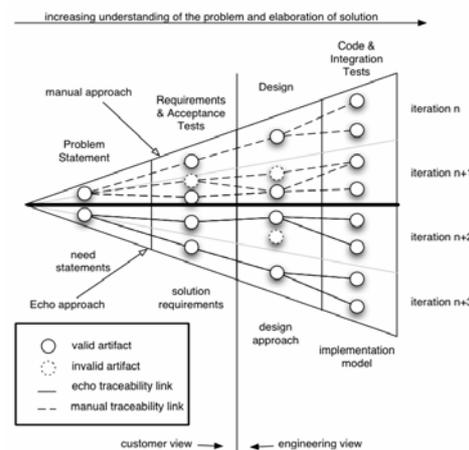


Figure 1: Maintaining traceability manually in an agile project space

Figure 1 shows how an agile project works from stakeholder need statements to iteration deliverables. Maintaining traceability in the standard fashion would

require manual tracking and maintenance from artifact to artifact, across the different stages. In complex projects, management of the links becomes burdensome, and accuracy depends on the frequency of updates. If not maintained correctly, links from one artifact to another may be incorrect, as requirements or other requirements documents are disposed of. The traceability is therefore rarely end-to-end, as the time and effort required is more than the perceived worth.

3.1. The Shortcomings of User Stories

Many requirements elicitation strategies leverage informal techniques for capturing information exchanges with customers. Extreme Programming involves the user story best practice. Stories are short descriptions of features or behaviors of a system, written by the customer generally on index cards. Stories are the vehicles that then drive conversations, which uncover details about the story's described task, possibly including an in-depth description of the task, the risk, the cost, the priority, and outstanding technical or design issues. The actual information varies from project to project, and team to team, as the agile methodology stresses the importance of using practices that fit the team. Requirements are often expressed in documents, such as RFP's, project briefs, business process models, and procedural manuals.



Figure 2: A sample information flow based on XP.

Many requirements result from the gained insights that surface as the development team carries out analysis of the gathered documents and notes. These requirements are represented in the form of a structured specification, usually aligned with an overall development approach, such as a use case, user story, or feature description. By providing a means to contribute the reference and excerpts of the content source to overall requirement specifications enables the linkage between the various models.

3.2. The Deficiency of the Traceability Model

The relationship between problem and solution model are rather fluid as the understanding of the problem evolves and is redefined. The nature of agile development artifacts makes it difficult for static, document-centric models to fulfill these needs. Recording links between changing requirements and design constructs is only successful when there is a clear understanding of how changing customer needs are mapped onto requirements specifications. For

traceability to succeed in agile development models, the relationship between conversations about the problem and refinement into requirements specifications must first be addressed.

4. The Echo Approach

Software engineering and development of new software systems is an exploratory process that needs to manage change. Managing change to maintain predictability and quality requires rigorous application of best practices, such as just-in-time planning, traceability, and continuous requirements gathering and validation [20].

Solutions for agile practices must closely mimic and support these human-centric practices and adopt the principles promoted by collaborative authoring where content progressively contributes structures like plans, stories, and tasks, which effortlessly trace to changing content. Requirements elicitation takes place in the form of conversations, where insights into the problem space, clarification of assumptions, and deeper understanding take place. While these conversations are rarely captured because of their ad-hoc nature, tools should provide a mechanism to record unstructured requirements elicitation and transcribe them to a more structured model when appropriate.

4.1. A Tool-based Approach for Requirements Traceability

Echo is a tool-based approach designed to leverage the activities carried out by agile development teams to deliver the benefits of requirements engineering and traceability practices. Rather than requiring changes in development processes and practices, Echo takes advantage of the elaboration activities carried out when detailing user stories (or other agile requirements constructs) into other project artifacts (e.g. CRC, CAT, Plans, etc). The team, following agile best practices, is often engaged in the refinement, exploration, synthesis of the initial user story into a more precise set of specifications representing the customer needs and the solution to be developed. Echo, by providing a tool-based mechanism to facilitate such activities, enables the development team to progressively refine needs statements into solution requirements as the project carries on. More importantly, the transposition of structure onto unstructured content, as a result of user-driven elaboration, enables the recording of software development artifacts like requirements specifications. Additionally, the set of relationships that document design rationale are also captured. Figure 3 illustrates the approach where initial story cards are ultimately elaborated into formal requirements specifications (e.g. use cases). As conversations with customers,

stakeholders, and team members occur, providing supporting content and insight into the problem and the solution spaces, structure progressively shapes the collected information into the set of specifications that will drive development activities. By facilitating the natural elaboration activities, Echo first provides an effective means to record and share information gathered during requirements analysis, and secondly transparently delivers the benefits of requirements engineering and design rationale traceability, while maintaining true to the principles of agile methods.

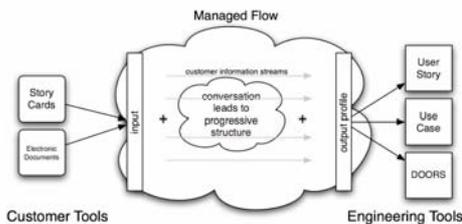


Figure 3: Echo Pipeline - Information input, creation, and refinement.

4.2. Adherence to Agile Development Principles

For an approach to be successful at capturing traceability in an agile project, it must account for the principles and practices commonly espoused by agile development teams. Projects, while still producing many artifacts, often do not deliver documents as traditional development processes do. The set of artifacts produced is continuously created or updated (just-in-time) as development proceeds. Story cards are often ripped-up and re-written, affecting their relationships with CRCs, and tests. Finally, customers and developers contribute and review content in various forms like customer meeting notes, design sessions, and other various communication exchanges. It is also important to note the variability of the information model used by each team to capture requirements specifications, including the rules and structure. Use case models for one team are different than use case models for another team.

The tool-based approach proposed, Echo, accounts for the previously detailed constraints critical to agile development teams. It begins by creating a conversation-centric model for recording exchanges with customers and team members. Each conversation contributes to the overall unstructured set of information. Conversations can record exchanges with customers or can be used to contribute available project documentation (e.g. RFPs). The approach does not enforce any specific organization model for the conversation space. Users are free to organize the contributed content according to a project-specific model.

Secondly, Echo allows users to progressively structure the content into specifications of the solution. In face-to-face analysis meetings, where discussions reach new insights, the discussion is captured and ultimately recorded as part of a specification. Via context-specific annotations, users are able to extract critical snippets on information from the conversation space, contributing them to structured representations of the solution space, such as a use case specification. The context for the structuring options is provided by the subject information. For example, when annotating a use case, a user is provided with the means to create further structures like CRCs. The structures and annotation rules are configurable to match the approach used by the team.

Thirdly, the approach provides for both unstructured and structured view of the information together with indicators to help users assess the completeness of the specifications. Much like a story wall, a tool commonly used by agile development team to track requirements and visualize dependencies, Echo provides visual hints that enable users to identify areas where further refinement is required. Echo also helps assess the impact of changes as the implicitly captured links allow forward and backward traceability between structured and unstructured content.

4.3. Prototype Architecture and Implementation

The proposed tool-based approach has been implemented as a plug-in for the Eclipse Integrated Development Environment. The implementation leverages the facilities provided by Eclipse to support presentation (SWT), role-oriented views (perspectives) and team development.

As illustrated in figure 4, users interact with Echo via a set of navigation and annotation views. Navigation views are designed to provide the user various access to both structured and unstructured content. The annotation views provide in context access to the various structuring mechanisms that will give rise to the information artifacts.

The annotation and navigation views are supported by corresponding facilities responsible for mediating access to the repositories and applying the rule-sets defined by the project artifact and annotation schemas. The navigation facility dynamically builds views of the structured and unstructured repositories and provides the traceability-based access of the information. As users interact with the views, the facility extracts and formats the content to suit the user's needs. The annotation facility, using rules specified by a project's schema set, provides access to the available annotation constructs. For example, as the user highlights content in a conversation, the annotation facility (and views) makes the "Use Case" annotation available. The user could

then select any of the schema-based annotation tags (e.g. “use case name”) to markup the content and contribute it to a structured representation, an artifact.

An XML-based repository provides for the storage of conversations, artifacts, and schemas used by Echo. The Conversation Store (CS) is responsible for persisting unstructured content contributed to the project like meeting notes, documents, and emails. The Artifact Store (AS) provides persistence for the structured content created by users. Both the CS and AS store the relationships resulting from the annotation and hence, the traceability links. The Schema Store (SS) persists project specific definitions for artifacts, annotation, and relationship rules. The definitions specify the properties that characterize an artifact (e.g. a use case has a use case id), the annotation rules specify how content is contributed to an artifact (e.g. there is only one value associated with a property like risk) and relationship rules constrain source and target artifact types (e.g. a feature leads to a story).

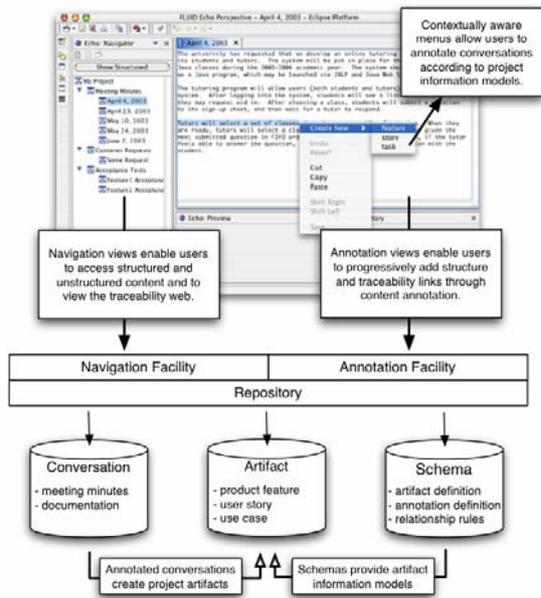


Figure 4: Prototype Architecture

As previously mentioned, the first version of Echo is available as a plug-in to IBM’s Eclipse Integrated Development Environment. However, it has been designed to easily port to other SWING-based development environments, such as NetBeans, Together Control Center, etc. It is also integrated with other tools in Project FLUID to support planning, visualization, and impact analysis.

4.4. Usage Scenario

For the Echo approach to be most effective, it should be used as early as possible in the design and

development process. The development team may gather requirements initially through meetings with the customer, or with a requirements document. Conversation with the customer should be captured, possibly in the form of meeting minutes.

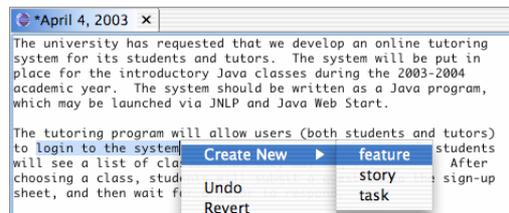


Figure 5: Annotating meeting minutes to create a new feature

From the meeting minutes, system features and functionality may be annotated as shown in figure 5, creating the beginnings of requirements artifacts. As future design sessions are captured, additional artifacts may be created or information may be appended to existing artifacts as in figure 6.

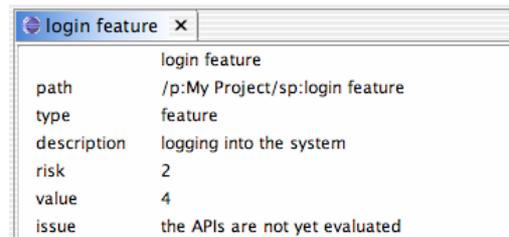


Figure 6: Detailed view of a feature

While providing the ability to annotate requirements into artifacts, the Echo approach also provides traceability from the artifacts back to the original requirements.

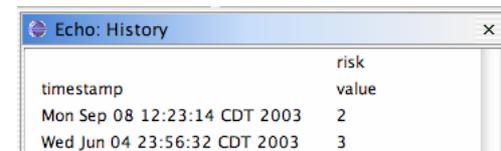


Figure 7: History and versioning of an Annotation

Information in a requirements artifact may be traced back to its design rationale allowing for the user to view important contextual information and possibly overlooked design assumptions not included in the artifact. Figure 8 shows the traceability of a risk value back to the conversation it was annotated from, highlighting the value in the conversation window.

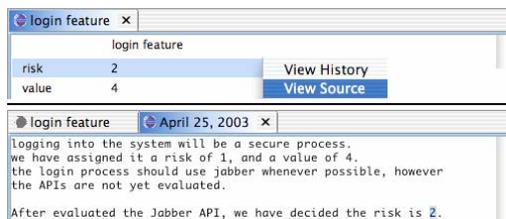


Figure 8: Traceability back to design rationale

The usage scenario shows how the Echo approach creates artifacts from design rationale, and provides the traceability from the artifacts back to the design rationale. The requirements artifacts may vary from project to project, and is not limited to Features. For example, design rationale may be annotated to create a Use Case scenario or a User Story.

5. Future Work

A community of agile practitioners in Chicago supported the design and development of the approach. To validate the approach, and its implementation, usability testing has begun. The various practitioners have volunteered to adopt and provide feedback on the approach. Results of these evaluations will be the subject of subsequent papers. Concurrently, the approach will be applied to develop further extensions to the tool.

Future development efforts for Echo include extensions of the approach to support a multi-user environment to enable distributed collaboration. Additional wizard support will be added for schema creation and manipulation to assist users in developing structured models and rules for the project. There are also plans to extend the approach to allow the capturing of other input mediums as well as exploring annotation mechanisms as a means to support traceability recording in other areas of development (e.g. design, coding, and testing.) Finally, the Echo approach should be integrated with other tool-based approaches to support the scalability of other agile practices such as Test Driven Design, and Continuous Integration.

6. Conclusion

This paper has described a tool-based approach to enable the scalability of agile requirements gathering practice. Echo provides a mechanism that allows for flexible and dynamic creation of content as well as the supporting traceability structure. Therefore, in the spirit of agility, the focus on customer needs and software development is maintained during requirements elicitation activity. Traceability is beneficial to the software development process, but is difficult and complex to manage accurately. Echo delivers a transparent model in which traceability is implicitly

maintained while the content is created, and thus brings about the benefits of requirements engineering practices to agile development methods.

Acknowledgements

The authors would like to thank the Chicago Agile Development community for its support and feedback as well as the Institute for Software Engineering at DePaul University.

References

- [1] Beck, K, *Extreme Programming Explained: Embrace Change*, 2001, Addison Wesley.
- [2] Cockburn, A., *Agile Software Development*, 2002, Pearson Education.
- [3] Crocker, R., "The 5 reasons XP can't scale and what to do about them".
- [4] Oppenheimer, H., and Mancl, D., "An Architecture for Collaboration: a Case Study", Lucent Technologies.
- [5] Gottesdiener, E., "Specifying Requirements With a Wall of Wonder", *The Rational Edge*, 2001 Nov.
- [6] Leon, M., "Staying on Track", *Intelligence Enterprise*, 2000 Sept, pp 54-57.
- [7] Standish Group, "Chaos Report", 1995.
- [8] Boehm, B., "Project Termination Doesn't Equal Project Failure", *Computer*, 2000 Sept, vol. 33, no. 9, pp 94-96.
- [9] Potts, C., Takahashi, K. and Anton, A.I., "Inquiry-Based Requirements Analysis", *IEEE Software*, vol. 11, Issue 2, 1994, pp.21-32.
- [10] Jorgensen, H. D., and Carlsen, S., "Emergent Workflow: Planning and Performance of Process Instances."
- [11] Schwaber, K., and Beedle, M., *Agile Software Development with Scrum*, 2001, Prentice Hall.
- [12] Peng, X., and Balasubramaniam, R., "Supporting Workflow Management Systems with Traceability", *Proceedings of the 35th Hawaii International Conference on System Sciences*, 2002, pp 91c.
- [13] Rees, M. J., "A Feasible User Story Tool for Agile Software Development?", *Proceedings of the Ninth Asia-Pacific Software Engineering Conference*, 2002 Dec, pp 22-31.
- [14] van Deursen, A., and Visser, E., "The Reengineering Wiki", *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*, 2002 Mar, pp 217-221.

- [15] Lindvall, M., and Sandhal, K., "How well do experienced developers predict software change?", *The Journal of Systems and Software*, vol. 43, 1998, pp 19-27.
- [16] Jeffries, R., Anderson, A. and Hendrickson, C., *Extreme Programming: Installed*, 2001, Addison Wesley.
- [17] Anecdotal evidence from Chicago Agile Developers (ChAD) group discussion.
- [18] Carpenter, P. B., "Verification of Requirements For Safety-Critical Software", *The Engineering of Industrial Strength REAL-TIME Software & Distributed Systems Using Ada and Related Technologies*, 1999.
- [19] Bentley, R. and Dourish, P., "Medium versus mechanism: Supporting collaboration through customization", *Proceedings of the European Conference on Computer-Supported Cooperative Work*, 1995.
- [20] Rasmusson, J., "Introducing XP into Greenfield Projects: Lessons Learned", *IEEE Software*, 2003 May, vol. 20, no. 3, pp 21-28.
- [21] Bianchi, A., Fasolino, A., and Visaggio, G., "An Exploratory Case Study of the Maintenance Effectiveness of Traceability Models", *Proceedings of the Eighth International Workshop on Program Comprehension*, 2000 Jun, pp 149-159.
- [22] Rational Software Corporation, "Rational Unified Process", Version 2001.3, CD-ROM, Rational Software, Cupertino, CA, 1999.
- [23] Ramesh, B., Powers, T. and Stubbs, C., "Implementing Requirements Traceability: A Case Study", *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, 1995, pp 89-95.
- [24] Gotel, O., and Finkelstein, A., "An Analysis of the Requirements Traceability Problem", *Proceedings of the IEEE International Conference on Requirements Engineering*, 1994 Apr, pp 94-101.
- [25] Macfarlane, I., and Reilly, I., "Requirements Traceability in an Integrated Development Environment", *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, 1995, pp 116-123.
- [26] Moore, J. Michael, and Shipman III, Frank M., "A Comparison of Questionnaire-Based and GUI-Based Requirements Gathering", *The Fifteenth IEEE International Conference on Automated Software Engineering*, 2000.
- [27] Carroll J.M., Rosson, M.B., Chin, G. and Koenemann, J. "Requirements Development: Stages of opportunity for collaboration needs discovery", *Proceedings of Designing Interactive Systems: Processes, Practices, Methods, & Techniques*, 1997, pp. 55-64.
- [28] Herzum, P. and Sims, O., *Business Component Factory*, Wiley and Sons, 1999.
- [29] Lee, C., and Guadagno, L., "FLUID:Echo – Agile Requirements Authoring and Traceability", *Proceedings of the Midwest Software Engineering Conference 2003*, June 2003, pp. 50-61.
- [30] B. Ramesh, and Jarke, M., "Toward Reference Models for Requirements Traceability", *IEEE Trans. on Software Engineering*, Vol. 27, No. 1, Jan 2001, pp. 58-92.