Software Qual J (2013) 21:23–38
DOI 10.1007/s11219-011-9166-3

A streamlined, cost-effective database approach to manage requirements traceability

Hossein Saiedian · Andrew Kannenberg · Serhiy Morozov

Published online: 5 October 2011

Abstract Requirements traceability offers many benefits to software projects, and it has been identified as critical for successful development. However, numerous challenges exist in the implementation of traceability in the software engineering industry. Some of these challenges can be overcome through organizational policy and procedure changes, but the lack of cost-effective traceability models and tools remains an open problem. A novel, cost-effective solution for the traceability tool problem is proposed, prototyped, and tested in a case study using an actual software project. Metrics from the case study are presented to demonstrate the viability of the proposed solution for the traceability tool problem. The results show that the proposed method offers significant advantages over implementing traceability manually or using existing commercial traceability approaches.

Keywords Requirements traceability · Software requirements management · Requirement engineering · Software requirements · Traceability tools

1 Introduction

The importance of requirements traceability has been well-established throughout the software engineering industry. Young (2006) identified traceability as one of twelve requirements basics that is important for project success. Common guidelines for software development such as the Software Engineering Institute's Capability Maturity Model Integration (CMMI), ISO's 9001:2000 for software development, and IEEE's J-STD-016

H. Saiedian (✉)
Electrical Engineering and Computer Science, University of Kansas, Lawrence, KS 66049, USA
e-mail: saiedian@eecs.ku.edu
URL: http://saiedian.com

A. Kannenberg
Innovative Systems, Mitchell, SD 57301, USA

S. Morozov
Department of Computer Science and Software Engineering, University of Detroit Mercy, Detroit, MI 48221, USA

all include requirements traceability practices. Even the United States government has recognized the importance of traceability by including it in governmental standards for software development such as DOD-STD-2167A (superseded by MIL-STD-498) for government contractors and DO-178B for aviation products.

Because of this, one would expect quality traceability practices to be firmly ingrained throughout the software engineering industry. Unfortunately, this is not the case. Research has shown that numerous organizations are struggling with implementing traceability practices (Jarke 1998; Ramesh 1998; Ramesh and Jarke 2001; Egyed and Grunbacher 2002). Many organizations do not even attempt to implement traceability, while others only do so in a haphazard manner.

Why is this? Perhaps, it is because manual methods for implementing traceability are time-consuming and error-prone. However, this cannot be the only reason because alternatives to manual traceability methods exist. The International Council on Systems Engineering (2008) has identified 31 different tools that claim to provide full traceability support. In spite of the large number of available traceability tools, the adoption rate throughout industry is surprisingly low. A general study of the software engineering industry performed by Gills (2005) found that less than half of the organizations utilized any tools to assist with requirements traceability. Even an aviation software-specific study, a field where traceability is mandated by governmental regulations, discovered that only half of the organizations surveyed use specialized tools to assist with traceability (Lempia and Miller 2006).

If there are 31 tools that provide full support for traceability, then why are they not more widely deployed throughout industry and why are quality traceability practices not more prevalent? Research has indicated that poor support for traceability in existing tools is the root cause for its lack of implementation (Spanoudakis et al. 2004) and that traceability practices will only improve when tools are deployed that reduce the effort required to implement and maintain traceability information (Munson and Nguyen 2005). It follows that if currently existing traceability tools were adequate for the needs of the industry, then it would be reasonable to expect that their adoption rate would be much closer to 100%, especially in the area of aviation software due to its mandates for traceability practices.

2 The traceability tool problem

Poor tool support is perhaps the biggest challenge to the implementation of traceability today. Spanoudakis et al. (2004) argues that currently existing traceability methods and tools are inadequate for the needs of the software engineering industry. Neither manual traceability methods nor existing commercial traceability tools provide a streamlined, cost-effective approach to managing requirements traceability.

2.1 Problems with manual traceability methods

Cleland-Huang et al. (2003) found that the number of traceability links that need to be captured grows exponentially with the size and complexity of the software system. This means that manually capturing traceability data for large software projects require an extreme amount of time and effort.

Manual traceability methods are vulnerable to changes in the system. If changes occur to any elements captured in the traceability data, the affected portions of the traceability data must be updated manually. This requires discipline and a significant amount of time

and effort spent on link checking throughout the traceability data. Because of this, it is easy for manually created traceability data to become out of sync with the current set of requirements, design, code, and test cases.

Manual traceability methods are also prone to errors, which are not easy to catch. Errors can arise from simple typographic mistakes, from inadvertently overlooking a portion of the traceability data such as an individual requirement or from carelessness by the individual capturing the traceability data. Because traceability artifacts for large projects are often hundreds or even thousands of pages in length, such errors are difficult to detect when depending on manual methods for error checking.

Due to these disadvantages, manual traceability methods are not suitable for anything other than small software projects. Young (2006) stated "in my judgment, an automated requirements tool is required for any project except tiny ones." Similarly, Cleland-Huang (2006) found that traceability is error-prone, time-consuming, and arduous without the use of automated tools. In spite of this, two recent surveys of software companies found that approximately 50% of software companies are using manual traceability methods (Gills 2005; Lempia and Miller 2006). In 1994, Gotel and Finkelstein found that manual traceability methods were preferred in industry due to shortcomings in available traceability tools. It is apparent that this problem still exists today because manual traceability methods are still preferred by a significant percentage of software organizations (Gills 2005; Lempia and Miller 2006).

2.2 Problems with commercial off the shelf (COTS) traceability tools

Regrettably, currently existing COTS traceability tools are not adequate for the needs of the software engineering industry either. Research has indicated that the simplistic support for traceability provided in existing traceability tools is the root cause for the lack of implementation of traceability (Spanoudakis et al. 2004).

COTS tools are typically marketed as complete requirements management packages, which means that traceability is only one added feature. The traceability features usually only work if the project methodology is based around the tool itself. Unless the project is developed from the ground up using a particular tool, the tool is unable to provide much benefit without significant rework.

Although some COTS traceability tools do support the identification of impacted artifacts when changes occur, they typically do not provide assistance with updating the traceability links or ensuring that the links and affected artifacts are updated in a timely manner (Cleland-Huang et al. 2003). This means that even when tools are used, the traceability information is not always maintained, nor can it always be trusted to be up to date and accurate.

COTS traceability tools often suffer problems with poor integration and inflexibility. This has led one researcher to conclude that existing traceability tools have been developed mostly for research purposes, and that many projects are still waiting for tools that do not require a particular development or testing methodology (Gills 2005).

Cost is another major disadvantage. Although the licensing fees vary per tool, the price tends to be thousands of dollars up-front per license in addition to yearly maintenance fees. Because of this, the cost of using COTS tools is often prohibitive, even for fairly small teams. Such tools are also decoupled from the development environment, meaning that important traceability information such as code modules that implement requirements may not be available (Naslavsky et al. 2005). This is a significant problem with popular commercial traceability tools such as Telelogic's DOORS and Requisite Pro.

Few solutions are available for the problem of poor tool support for traceability. The adoption of COTS tools is hampered by shortcomings such as inflexibility and poor integration (Lungu and Muvuti 2004), which leads organizations to instead utilize manual methods such as traceability matrices (Gills 2005; Lempia and Miller 2006). Some organizations concerned with high-quality traceability information develop their own in-house tools and utilities to implement traceability (Lempia and Miller 2006). Unfortunately, this approach is not always feasible because many organizations do not have the manpower or the knowledge necessary to develop such tools. Therefore, poor tool support for traceability remains an open problem at this time.

3 A proposed solution for the traceability tool problem

The development of improved tools for implementing traceability is not an insurmountable problem. The solution is simply the creation of traceability tools usable for software projects that do not share the limitations of currently available tools and that are available for a reasonable cost. To accomplish this, a proposal for a novel, cost-effective traceability tool that improves upon the capabilities of existing tools is presented.

3.1 A proposal for a database-based approach to traceability

The main idea behind the proposal is to use a database to store all traceability information and to include a mechanism supporting the generation of a set of complete traceability artifacts. Identifiers for each traceability element such as requirement identifiers, code module names, and test cases would need to be stored within the database, but the elements themselves could be maintained outside of the database to reduce the impact on existing project artifacts.

Identifiers for requirements and other project artifacts would need to be imported into the database for it to be used with an already existing project. A software wrapper around the database would be included to parse existing requirements documents and other project artifacts. After the initial set of records containing identifiers in the database was created, it could be kept up-to-date by regular usage of the importation features of the tool. Depending on the needs of the project, this process could occur automatically at periodic intervals or it could require human intervention to trigger the updates.

Traceability would be maintained through the use of link fields for each requirement record. These fields would specify other requirements, design, source code modules, and test cases that each requirement traces to. This allows the tool to provide complete traceability for all project artifacts, a feature that is lacking in many popular traceability tools (Naslavsky et al. 2005).

Filling out the link fields would be where the human interaction in this traceability method would take place. Requiring human interaction to create traceability links is a reasonable decision because it is impossible to completely remove human interaction from the traceability process (Hayes and Dekhtyar 2005) and because the reason for adding a traceability element is nearly always known by the person adding that element.

Depending on the format of the project, portions of the link creation could be automated. For example, test cases typically identify the requirements and source code that they test. The software wrapper for the database tool could parse this information from test cases and use it to create links between the test case and the requirements

and source code identified in the test case. The database would then be capable of generating a complete traceability artifact based on the stored identifiers and the link fields for each element.

The database would make use of referential integrity to ensure that all links stored within the database are valid. If a requirement or other data element is deleted, the database would be able to detect and flag any traceability links that become invalid. Flagged links would need to be corrected to satisfy the constraints of referential integrity, thereby ensuring that any invalid links are corrected before the traceability artifact can be generated. Similarly, the database would be able to detect and prevent any attempts to create links between invalid project elements using referential integrity.

3.2 Prototyping the database-based approach to traceability

Developing a prototype for the proposed database-based traceability approach required three main activities: identifying the necessary traceability data, designing the database, and creating a software wrapper around the database to provide the user interface, automation, and error-checking capabilities.

3.2.1 Identifying the necessary traceability data

The first step toward creating an improved traceability tool was to identify the data that needed to be traced. Because the database-based traceability tool was planned for use in a case study in the aviation software industry, it needed to be able to meet the governmental traceability mandates for aviation software projects specified by DO-178B. These mandates include the following:

- Traceability between system requirements and software design data
- Traceability between system requirements and software requirements
- Traceability between software requirements and source code
- Traceability between software requirements and test cases
- Traceability between source code and test cases

To fulfill these requirements, the traceability tool needed to track links for all of the mandated traceability data.

3.2.2 Designing the database

The next step was to design a database capable of storing traceability information for the identified traceability elements. The database design began with an entity-relationship diagram relating the entities that needed to be traced to fulfill the DO-178B traceability mandates. For other applications, the database design could easily be adapted to include other traceability information by customizing the elements included in the entity-relationship diagram. The resulting entity-relationship diagram is shown in Fig. 1. It should be noted that according to DO-178B, unary relationships between system requirements or software requirements are optional depending on how the requirements are broken down into high- and low-level requirements. We have chosen to show such unary relationships.

The next step was to create an efficient database schema based on the entity-relationship diagram. The database schema was normalized into Boyce–Codd Normal Form (BCNF) to provide protection from redundancy and logical anomalies.

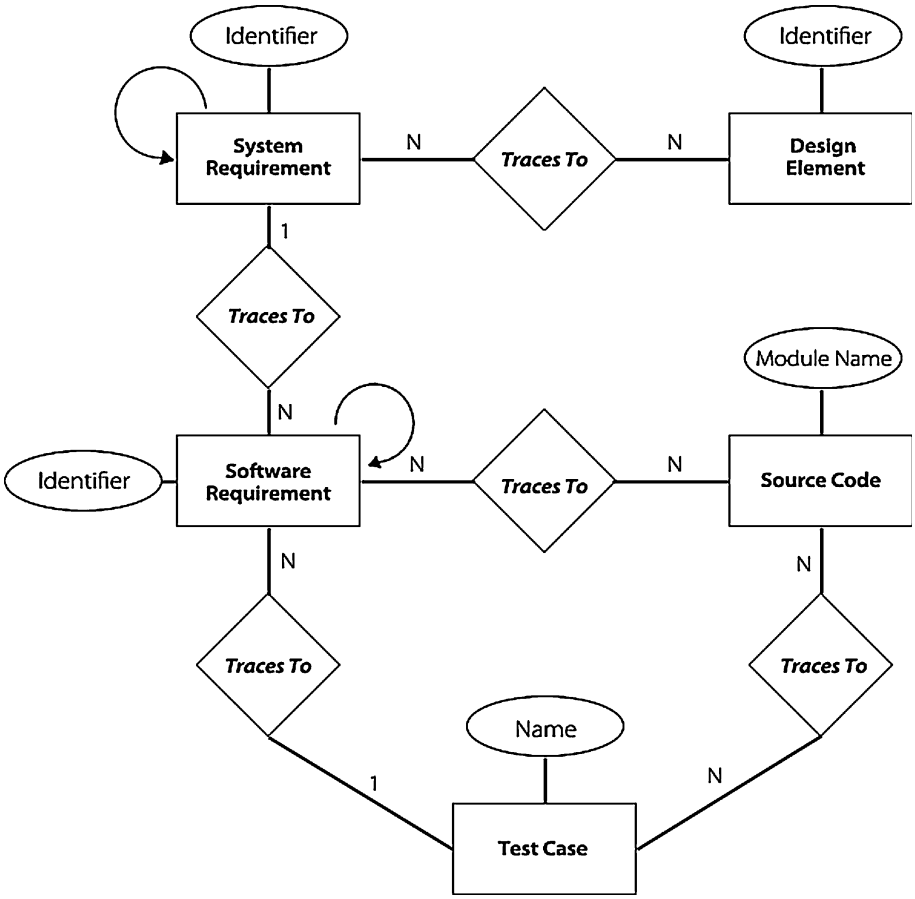

Fig. 1 Entity-relationship diagram for the database

3.2.3 Creating the software wrapper for the database

After the database design was complete, software mechanisms for automatically populating the database relations needed to be written. A software wrapper for the database was written to automatically populate the system requirements, design data, software requirements, source code, and test cases relations in the database. This involved writing code to parse the requirements and design documents for requirements and design identifiers and to store them in the appropriate relations in the database. Although this does require project artifacts to be stored in a text-parsable manner, since the majority of projects using manual traceability methods use text-parsable artifacts (Lempia and Miller 2006), this approach is effective for most projects. For the source code and test cases, the importation software was set up to simply read the directories where all of the source code and test cases for the project were stored and to enter the name of each source code module and test case into the database.

Traceability links need to be recorded by entering them into the database. This is where the human interaction in the traceability process occurs. Requiring human interaction to

create traceability links is a reasonable decision because it is impossible to completely remove human interaction from the traceability process (Hayes and Dekhtyar 2005) and because the reason for adding a traceability element is nearly always known by the person adding that element.

Test cases are an exception to this process because test cases already identify the requirements and source code that they test. Therefore, traceability links involving test cases can be automatically populated by code written to parse each test case for the requirement identifiers and source code modules that they identify. This leaves only the traces between requirements, design elements, and source code as items requiring human interaction.

Validity of the traceability links is enforced through referential integrity. Only links between valid elements are allowed because the use of referential integrity disallows the ability to create links to non-existent items. Each attribute in the link relations in the database is a foreign key that references the key attribute in the relation maintaining data for that particular traceability element. This reduces the potential for human error through typographical mistakes. The tool also includes a reporting feature that detects any missing traceability information.

To make the database tool easy to use, a custom menu was created to appear when the database tool is started. Buttons on the menu simplify traceability tasks. There are buttons to update the system requirements, design data, software requirements, source code, and test cases relations. There are also buttons to automate the test case traces and to manually enter traces between requirements, design, and source code elements. A button for detecting missing traceability links is also included as well as a button for generating a complete traceability artifact in a traditional traceability matrix format.

4 A practical case study

After the proposed database-based traceability tool was prototyped, it was tested in a case study using an actual software project in the aviation software industry. A description of this case study is provided, and metrics are presented to demonstrate the viability of the database-based traceability approach.

4.1 Software project background

The software project used for the case study is an iterative, incremental project, where versioned builds of the software are delivered periodically. Each succeeding build of the software is based upon the previous build, but it adds significant new functionality. The project is used in the aviation industry and is therefore subject to the governmental mandates specified by DO-178B. The initial build of the project took place in 2002, and the project has continued to grow in size and complexity since that date. Today, the development team for the project includes 45 software engineers.

Little thought was given to traceability prior to the completion of the first build of the software project. The lack of planning for traceability meant that it was difficult to implement, making it a time-consuming activity that provided little benefit to the project apart from meeting governmental mandates.

Traceability information was recorded in a traceability matrix contained in a single spreadsheet shared among the software engineers working on the project. This was not a very efficient mechanism because all of the traceability data were gathered manually, and it

needed to be entered into the spreadsheet manually by each software engineer. Having multiple engineers work in parallel was a challenge because only one person could enter data into the spreadsheet at a time. Multiple individuals could work in parallel using a temporary copy of the spreadsheet on their own computer, but there was no foolproof method to ensure that work was not duplicated, and merging each person's changes into the spreadsheet was a time-consuming and potentially error-prone process.

The information recorded in the spreadsheet traced system requirements to software design data, system requirements to software requirements, software requirements to source code, software requirements to test cases, and source code to test cases in order to meet the traceability mandates of DO-178B. The source of this information was special knowledge either recollected or researched by specific engineers working on the traceability artifact since most of the information had not been previously documented. This meant that finding traceability data for items that none of the engineers had a clear recollection of was difficult and time-consuming.

Overall, the creation of the traceability artifact required input from 23 software engineers and took 5 weeks to create. When the initial version of the traceability matrix was subjected to a review, it took another full day to correct all of the problems found during that review. In the end, the lack of forethought regarding traceability meant that the initial delivery of the software was delayed by nearly 6 weeks after the software build itself was complete. The significant delays introduced by the creation of the traceability artifact after the completion of the first software build made it obvious that better methods were necessary for implementing traceability in the future, which made the project an ideal candidate for a case study for the database-based traceability tool.

The database-based traceability tool was introduced as a replacement for the manually created traceability matrix, which had been used to document traceability information for the project in the past. The conversion occurred right after the release of a build of the software so that it would not cause a disruption right in the middle of a software release. From that point on, the engineers working on the project used the database tool to record traceability information for the project. Instead of waiting until the end of the software release to document traceability links, use of the tool to capture traceability links for project elements when they were created was added to the process of adding new elements to the project. Because the requirements for aviation software mandated by DO-178B necessitate reviews for all project elements, this was easily accomplished by including checks for appropriate traceability in the review forms for each project element.

4.2 Metrics

Metrics were collected throughout the duration of the case study in order to determine whether the database-based traceability method provides a viable alternative to existing traceability methods. Quantitative metrics from the case study are presented, and a cost comparison with alternative traceability methods is performed.

4.2.1 Comparison with past project results using manual methods

This section quantitatively compares the results of using the database-based traceability tool with the manual traceability methods used on the case study project in the past. Bar graphs are used to detail the number of man-hours required for activities such as preparation for use (Fig. 2), time spent while working on a software release (Fig. 3), and time spent at the end of a software release (Fig. 4) for each method. The number of errors found

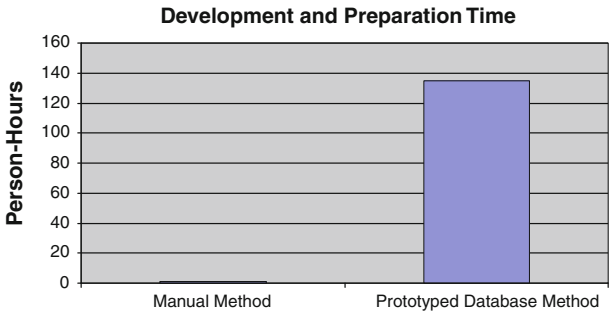

Fig. 2 Development time required for traceability methods

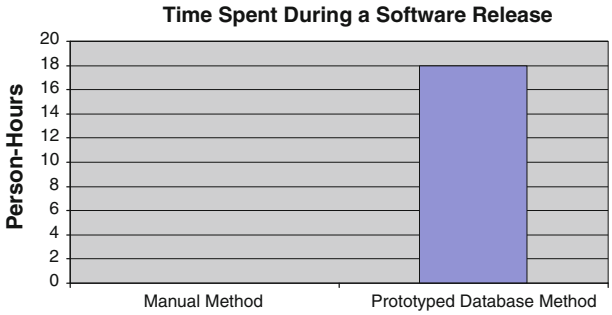

Fig. 3 Amount of time spent on traceability activities during a release

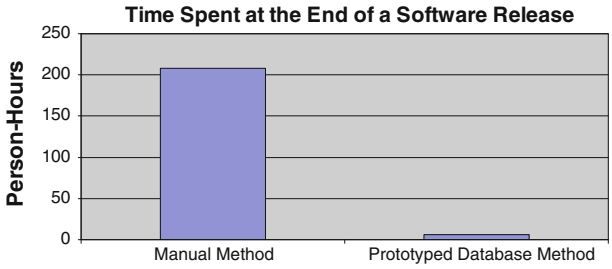

Fig. 4 Time spent on traceability activities at the end of a software release

after the initial release of the traceability data for each method is also compared (Fig. 5). These results are reasonable to compare because, for each method, the results were collected using software releases that added similar amounts of functionality to the system and because the database-based traceability tool utilized a completely different methodology for recording traceability information, thus minimizing the impact of organizational learning.

The database method of implementing traceability required significantly more development and preparation time than the manual method. This is because the database tool required a significant amount of complex custom code to be written for the automation, error-checking, and data output capabilities. By comparison, manual traceability methods

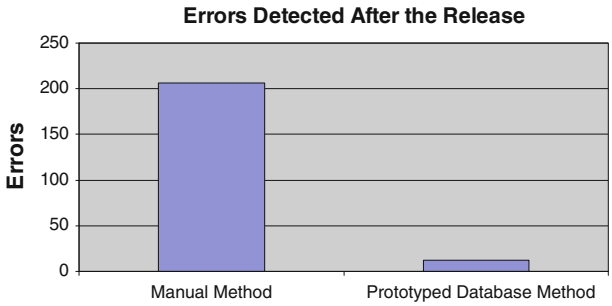

Fig. 5 Number of errors detected in the traceability data

require very little preparation time. The creation of a spreadsheet or a document with tables to record the data is sufficient. However, the extra development time required for the automated database traceability method pays off later through improved quality of the results (see Fig. 5) and time saved later on in the process (see Fig. 4). Because the development time is a one-time cost, it can be viewed as an up-front sacrifice resulting in faster, higher-quality results later. In addition, if the tool was reused for other projects, the development time would not need to be repeated for each project, thereby making it a start-up cost only.

Use of the database traceability method did require more time than manual methods while working on a software release due to the need to create traceability links as elements were added to the project. However, the extra amount of time required for the database method was a small price to pay for the time savings later as shown in Fig. 4 and better quality of the results as shown in Fig. 5.

Figure 4 clearly shows that the payoff for using the database traceability method comes at the end of a software release. Although some time is still required to generate the data and have the traceability information reviewed, the total time required is insignificant compared with the amount of time required to gather traceability data manually. In fact, it would be virtually impossible to reduce the amount of time required for traceability activities at the end of a software release because of the need for reviews.

The significant time savings at the end of a software release provided by the database-based tool is important because it meant that the software could be released to market approximately 4.5 weeks sooner than it could in the past when manual traceability methods were used. An earlier time to market results in additional sales which means that higher profits are realized.

Using the database method of implementing traceability greatly reduced the number of errors that were later detected in the released traceability artifact. Due to the robust error-checking features built into the database tool, only two errors were found after the release of the traceability data generated by the tool. These errors were human errors where incorrect links between requirements were manually entered into the database. The reason that so many errors were detected in the results from the manual method was because many requirements were overlooked in the manually created traceability matrix due to human error. While it is true that the amount of human error can vary based on staff experience and other factors, research has shown that human error can be expected wherever the opportunity exists in technical systems (Brown 2004). Since the manually created traceability matrix offered many more opportunities for human error than the database-based tool, it is not surprising that the number of human errors was much larger.

Fewer errors in the traceability results are significant because not only does it prevent the possibility of errors propagating later, but it also reduces the potential for errors to be uncovered during a Federal Aviation Administration (FAA) audit. The last time that errors were uncovered during an FAA audit on the project used for the case study resulted in two extra months of effort on the next software release to correct the errors and to put additional processes in place to prevent similar errors in the future.

4.2.2 Cost comparison with existing traceability tool alternatives

This section compares the cost of using the database-based traceability tool with the cost of using other traceability alternatives including manual methods and Telelogic's DOORS, which is the most popular COTS traceability tool used in the aviation software industry (Lempia and Miller 2006). Figure 6 compares the start-up costs for each traceability method, and Fig. 7 compares the cost of using each method for each software release.

The development and other necessary start-up efforts required for using the prototype for the database-based tool required approximately 995 man-hours of effort. Assuming an average salary of $75,000.00, this translates into a start-up cost of approximately $35,877.40. If Telelogic's DOORS (2008) had been selected for use on the project, the licensing cost for the 45 software engineers assigned to the project would have been $180,000.00 in addition to a $36,000.00 yearly maintenance fee. Converting to DOORS

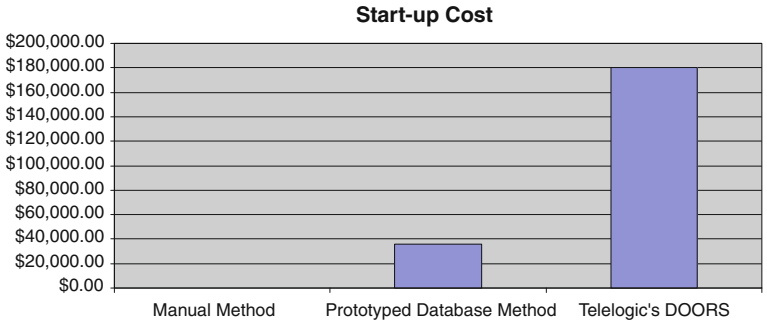

Fig. 6 Start-up cost comparison

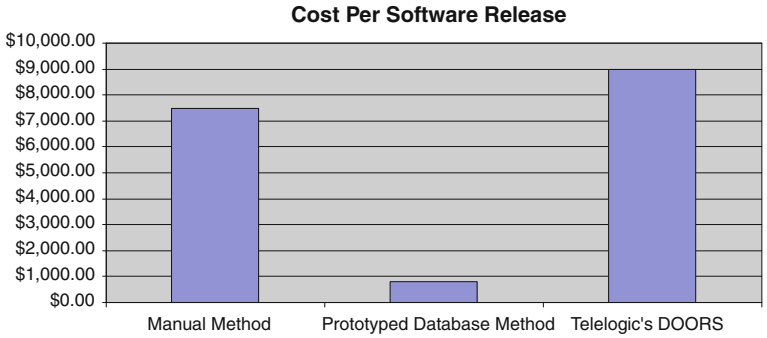

Fig. 7 Cost comparison per software release

would also incur a signification start-up cost in addition to the licensing fees because it would require both time and resources to convert the project over to the DOORS system.

It should be noted that DOORS does provide requirements management features in addition to traceability; thus, if an organization is able to make use of these features, the cost comparison may not be appropriate. However, many organizations are not able to utilize these features as they may need to maintain their project artifacts using standalone documents (Armbrust and Soto 2005). In these cases, the cost comparison is appropriate since only the traceability features of the tool are needed.

Manual methods require very little in terms of start-up costs because they can make use of a simple spreadsheet or table in a document. However, manual methods become more costly after a project is started due to the amount of time required to use them. This is shown in Fig. 7.

The high cost of using manual traceability methods is clearly shown in Fig. 7. Due to the large amount of time and effort required to implement traceability manually for each software release, manual methods incurred a cost of $7,500.00 per software release. In comparison, the prototyped database method only cost $793.27 because most traceability tasks were automated and did not require significant human interaction. The cost estimate of $9,000.00 for Telelogic's DOORS came from dividing the yearly maintenance fee of $36,000.00 by the average number of software releases per year (four) for the software project. In practice, the actual costs would be higher because time would need to be spent on traceability activities within the DOORS system for each software release.

Overall, use of the database-based tool for traceability is favorable in terms of cost in comparison with both Telelogic's DOORS and manual methods. Because implementing traceability using manual methods required 186 additional man-hours of work per software release, this translates into an extra cost of approximately $6,706.73 per software release. At that rate, only six software releases would be required to completely offset the initial development cost of the database-based tool. Because the software project used for the case study averages four software releases per year, the initial cost of development for the database-based tool would be offset in only 1.5 years. In addition, the estimate of the extra cost for using manual methods is a very conservative one, as neither the potential for extra sales resulting from releasing the product to market sooner nor the benefits from the higher-quality results provided by the database-based tool were taken into account. If the tool were used for additional projects, the overall costs would be even lower because the initial development costs could be spread among multiple projects.

Use of the database-based tool is also favorable in terms of cost when compared with using Telelogic's DOORS. The initial costs for developing the database-based tool were $144,122.60 less than licensing Telelogic's DOORS, and the cost per software release was $8,206.73 less because the database tool did not have yearly maintenance fees. This is a conservative estimate as the cost per software release for Telelogic's DOORS does not include the cost of the time that would need to be spent on traceability activities using the DOORS interface because these data were not available for the project for which the case study was performed.

5 Conclusions

Traceability offers many benefits to software projects, and it has been identified as being critical for their success (Young 2006). Unfortunately, many organizations struggle to understand and implement traceability which means that these benefits can go unrealized.

Many methodologies exist for implementing traceability; however, each existing methodology has important weaknesses that hinder the implementation of traceability. Most of these methods require a significant amount of manual work to create and maintain. Commercial tools exist that attempt to automate some aspects of the traceability process, but they are expensive and have their own set of limitations. Because of this, quality tool support for traceability activities in the software engineering industry has remained an open problem.

For this reason, this article proposed a streamlined, cost-effective method of automating traceability activities using a database-based tool. The proposed method was described, prototyped, and tested in a case study using an actual software project. Metrics from the case study were presented, and the results serve to demonstrate the viability of the proposed method for implementing traceability for software projects. Not only did the new method save time in comparison with manual methods of implementing traceability, but the resulting output also contained far fewer errors.

Existing commercial traceability tools such as Telelogic's DOORS and Requisite Pro have limitations including the lack of traceability information for source code (Naslavsky et al. 2005), a requirement that project artifact methodology be based around the tool itself (Gills 2005), which may not be feasible for some projects (Armbrust and Soto 2005), and, in many cases, high licensing fees. The new method did not share in these weaknesses. Not only was source code traceability information included, but the tool was also able to integrate with external project artifacts, thereby not requiring the project methodology to be based around the tool itself. As demonstrated through a case study, for projects in need of a standalone traceability tool, the new method was considerably more cost-effective than licensing a commercial tool like DOORS.

Traditional solutions focus on wide range of requirements management features, with traceability being far from the focal point of the system. Our proposed system has a simple, open architecture that allows easy customization to satisfy many traceability needs. It is a suitable solution for the requirements traceability problem because it provides automatic referential integrity assurance, supports customization, and reduces time to market at a fraction of the cost of the traditional solutions.

Furthermore, since the data are kept separate from the user interface, it may be accessed independently. This allows custom reports that would be difficult to implement in a proprietary system. Additionally, the data separation allows custom import/automation software to be integrated into our system. Data separation may enhance and simplify user experience, eliminating many human errors that were previously inevitable. Finally, our system keeps track of software artifacts that are managed outside of it. This makes our system language independent, so it may be used in any environment without interfering with the development process.

5.1 Opportunities for future research

It is worth noting that since the database tool was intended for projects currently using either manual traceability methods or no traceability at all, it does expect project artifacts to be stored in a text-parsable format. Since the majority of manual traceability methods utilize text-parsable formats for project artifacts (Lempia and Miller 2006), this methodology should be viable for most projects, but it may not be effective for projects using proprietary methods for artifact storage. Also, since the tool development was performed alongside the project for which it was tested in the case study, there is a potential that some project-specific customization may have occurred. An interesting area of further research

and development would be to perform additional case studies with other projects and identify challenges and complexities in adapting the tool for other types of projects.

The proposed system is applicable to a great number of software development domains as well as methodologies. Even though it was originally built to meet the requirements of the DO-178B standard, it offers much of the traceability capabilities necessary for a typical software project. In the industries where tracking of additional artifacts is necessary, our solution may be easily customized with minor changes to the database. For instance, if a company needs to keep track of the software engineers responsible for implementing a piece of code, a table for people and an intersection table for people-code relationship may be added. Future research into useful customizations for software projects in other industries and additional work toward simplifying tool customization as much as possible would be beneficial.

Our proposed traceability tool as well as many of the existing traceability tools focus primarily on requirements traceability or traceability among the various artifacts of a software product. But many organizations invest a great deal on their processes and maintain and improve it independently from their product development. End-to-end traceability refers to efforts in streamlining and combining process and product traceability activities. A major research area is to (1) consider how to analyze the challenges and problems in end-to-end traceability and (2) enhance the proposed tool to provide for this kind of traceability.

References

Armbrust, O., Ocampo A., & Soto, M. (2005). Tracing process model evaluation: A semi-formal process modeling approach. In *ECMDA traceability workshop (ECMDA-TW) 2005 proceedings* (pp. 57–66). Nuremberg, Germany.

Brown, A. (2004). Oops! Coping with human error in IT systems. *Queue, 2*(8), 34–41.

Cleland-Huang, J. (2006), Just enough requirements traceability. In *Proceedings of the 30th annual international computer software and applications conference (COMPSAC'06)* (pp. 41–42).

Cleland-Huang, J., Chang, C., & Christensen, M. (2003). Event-based traceability for managing evolutionary change. *IEEE Transactions on Software Engineering, 29*(9), 796–810.

Egyed, A., & Grunbacher, P. (2002). Automating requirements traceability: beyond the record and replay paradigm. In *Proceedings of the 17th IEEE international conference on automated software engineering* (pp. 163–171). Edinburgh, United Kingdom.

Gills, M. (2005). *Software Testing and Traceability*. University of Latvia. http://www3.acadlib.lv/greydoc/Gilla_disertacija/MGills_ang.doc.

Gotel, O., & Finkelstein, A. (1994). An analysis of the requirements traceability problem. In *Proceedings of the first international conference on requirements engineering* (pp. 94–101). Colorado Springs, CO.

Hayes, J., & Dekhtyar, A. (2005). Humans in the traceability loop: Can't live with 'Em, can't live without 'Em. In *Proceedings of the 3rd international workshop on traceability in emerging forms of software engineering,* (pp. 20–23). Long Beach, CA.

International Council on Systems Engineering. (2008). INCOSE requirements management tools survey. http://www.paper-review.com/tools/rms/read.php.

Jarke, M. (1998). Requirements tracing. *Communications of the ACM, 41*(12), 32–36.

Lempia, D., & Miller, S. (2006). Requirements engineering management, presented at the 2006 National Software and Complex Electronic Hardware Standardization Conference, Atlanta, GA.

Lungu, N., & Muvuti, F. (2004).Service oriented architecture for a software traceability system. Technical Report CS04-14-00, Department of Computer Science, University of Cape Town.

Munson, E., & Nguyen, T. (2005). Concordance, conformance, versions, and traceability. In *Proceedings of the third international workshop on traceability in emerging forms of software engineering* (pp. 62–66). Long Beach, CA.

Naslavsky, L., Alspaugh, T., Richardson, D., & Ziv, H. (2005). Using scenarios to support traceability. In *Proceedings of the third international workshop on traceability in emerging forms of software engineering* (pp. 25–30). Long Beach, CA.

Ramesh, B. (1998). Factors influencing requirements traceability practice. *Communications of the ACM, 41*(12), 37–44.
Ramesh, B., & Jarke, M. (2001). Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering, 27*(1), 58–93.
Spanoudakis, G., Zisman, A., Perez-Minana, E., & Krause, P. (2004). Rule-based generation of requirements traceability relations. *Journal of Systems and Software, 72*(2), 105–127.
Telelogic. (2008). Telelogic DOORS—requirements management for advanced systems and software development. http://www.telelogic.com/products/doors/doors/index.cfm.
Young, R. (2006). Twelve requirement basics for project success. *CrossTalk The Journal of Defense Software Engineering, 19*(12), 4–8.

Author Biographies

Hossein Saiedian received a PhD in Computing and Information Sciences from Kansas State University in 1989 and is currently a professor of software engineering at the University of Kansas. Professor Saiedian's primary area of research is software engineering and in particular models for quality software development, both technical and managerial ones. Professor Saiedian has over 100 publications in a variety of topics in software engineering and computer science. His research in the past has been supported by the NSF as well as regional organizations. Professor Saiedian is a senior member of the IEEE.

Andrew Kannenberg is a software engineer at Innovative Systems in Mitchell, South Dakota. He received a bachelor's degree in computer science from the South Dakota School of Mines and Technology in 2004 and his master's degree in computer science from the University of Kansas in 2008.

Serhiy Morozov received his Ph.D. in Computer Science from the University of Kansas in 2011 and is an assistant professor of computer science at the University of Detroit Mercy. He completed his MS in Computer Science in 2007. His primary research area is in recommender systems, data retrieval, knowledge discovery systems, and software engineering.