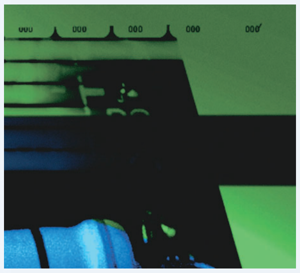
Best Practices for Automated Traceability

Automated traceability applies information-retrieval techniques to generate candidate links, sharply reducing the effort of manual approaches to build and maintain a requirements trace matrix as well as providing after-the-fact traceability in legacy documents. The authors describe nine best practices for implementing effective automated traceability.



Jane Cleland-Huang, Raffaella Settimi, and Eli Romanova
DePaul University

Brian Berenbach
Siemens Corporate Research

Stephen Clark
iRise

Requirements traceability, defined as the ability to "follow the life of a requirement in both a forward and backward direction,"[1] provides critical support for software engineers as they develop and maintain software systems. Traceability helps determine that researchers have refined requirements into lower-level design components, built them into the executable system, and tested them effectively. It further helps analysts understand the implications of a proposed change and ensures that no extraneous code exists.

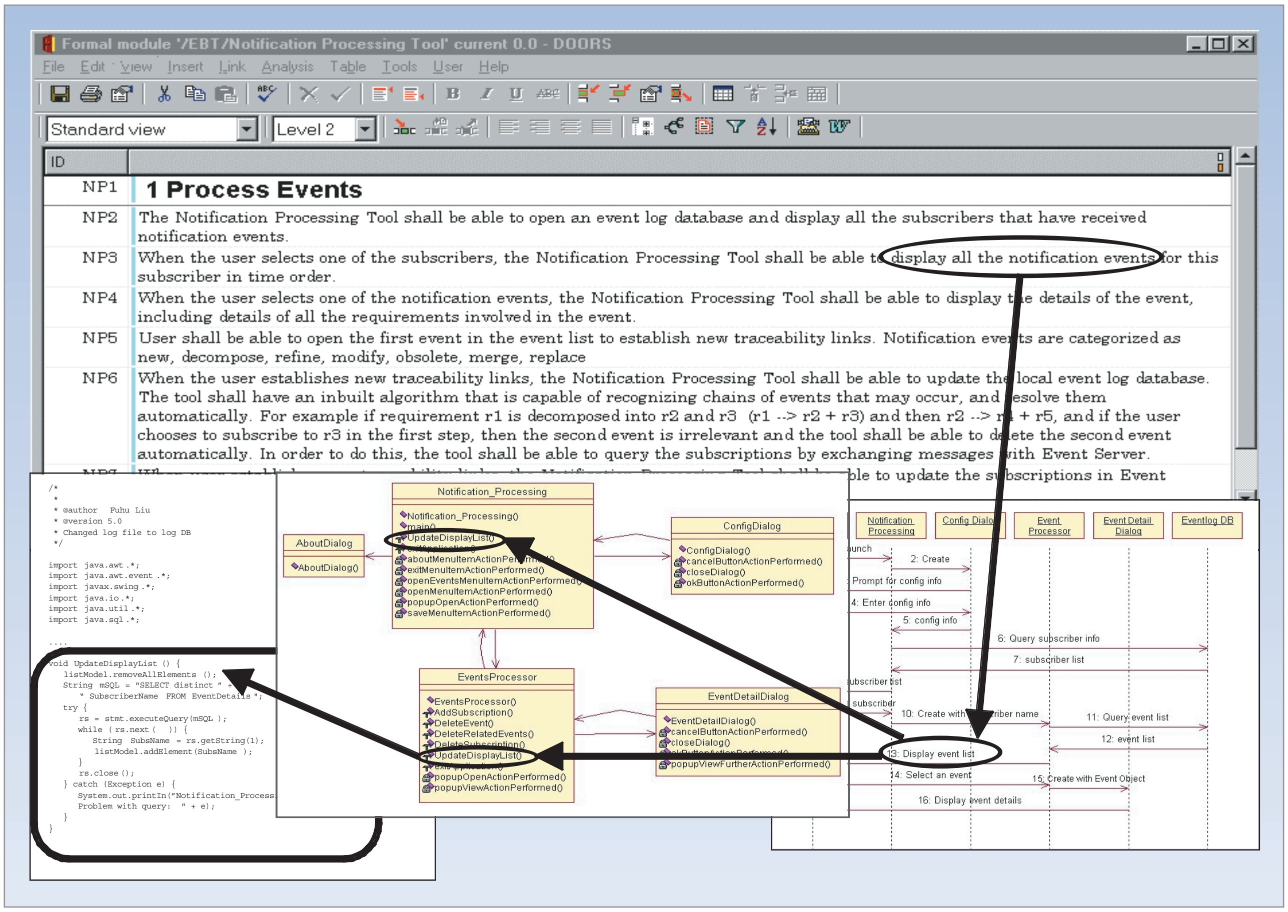
Numerous standards include traceability as a recommended or legally required activity. For example, *IEEE Std. 830-1998* states that a software requirements specification must be traceable. The standard defines an SRS as traceable "if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation." Backward traceability is required from requirements to "previous stages of development," while forward traceability proceeds from requirements to "all documents spawned by the SRS."[2]

Further, organizations building safety-critical systems are often legally required to demonstrate that all parts of the code trace back to valid requirements. Laws such as the US Sarbanes-Oxley Act of 2002 require organizations to implement change-management processes with explicit traceability coverage for any parts of a software product that potentially impact the balance sheet.

Unfortunately, many organizations fail to implement effective traceability practices either due to difficulties in creating, assessing, using, and maintaining traceability links or because they succumb to the misconception that traceability practices return little value for the effort involved.[3]

Traditionally, traceability links are physically stored in spreadsheets, text files, databases, or requirements management (RM) tools such as Telelogic's DOORS or IBM's Rational RequisitePro, and such links tend to deteriorate during a project as time-pressured team members fail to update them.[1] Offshoring and outsourcing exacerbate this problem by creating temporal and physical distance between subject-matter experts and developers.

Because manual traces are often created in an ad hoc fashion according to the developers' whim, they tend to be inconsistent and often incomplete. Current RM tools provide limited support for link creation and maintenance, offering features


Formal module '/EBT/Notification Processing Tool' current 0.0 - DOORS

ID	
NP1	1 Process Events
NP2	The Notification Processing Tool shall be able to open an event log database and display all the subscribers that have received notification events.
NP3	When the user selects one of the subscribers, the Notification Processing Tool shall be able to display all the notification events for this subscriber in time order.
NP4	When the user selects one of the notification events, the Notification Processing Tool shall be able to display the details of the event, including details of all the requirements involved in the event.
NP5	User shall be able to open the first event in the event list to establish new traceability links. Notification events are categorized as new, decompose, refine, modify, obsolete, merge, replace
NP6	When the user establishes new traceability links, the Notification Processing Tool shall be able to update the local event log database. The tool shall have an inbuilt algorithm that is capable of recognizing chains of events that may occur, and resolve them automatically. For example if requirement r1 is decomposed into r2 and r3 (r1 --> r2 + r3) and then r2 --> r4 + r5, and if the user chooses to subscribe to r3 in the first step, then the second event is irrelevant and the tool shall be able to delete the second event automatically. In order to do this, the tool shall be able to query the subscriptions by exchanging messages with Event Server.

```
/*
 *
 * @author   Fuhu Liu
 * @version 5.0
 * Changed log file to log DB
 */

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.util.*;
import java.sql.*;

....

void UpdateDisplayList () {
   listModel.removeAllElements ();
   String mSQL = "SELECT distinct " +
      " SubscriberName  FROM EventDetails ";
   try {
      rs = stmt.executeQuery(mSQL );
      while ( rs.next (   )) {
         String  SubsName = rs.getString(1);
         listModel.addElement(SubsName );
      }
      rs.close ();
   } catch (Exception e) {
      System.out.printIn("Notification_Process
      Problem with query:  " + e);
   }
}
```

Figure 1. Automated traceability solutions retrieve links based on the similarity of terms across documents.

such as drag-and-drop techniques for creating links, simple visualization for depicting traces, and flagging of "suspect" links—those for which one of the associated artifacts has been modified, suggesting that the links might have become outdated. Unfortunately, this type of support does not sufficiently ease the burden of managing traceability links, and practitioners often find that increasingly larger percentages of links become suspect.

Further, there is almost universal failure to establish traceability between requirements and supplemental documents such as stakeholders' rationales, vision documents, or other free-form textual documents. For many organizations seeking to attain a corporate-wide standard such as Capability Maturity Model Integration (CMMI) Level 3, achieving traceability in projects for which it was not initially a systematic part of the development process can seem like an arduous and often unachievable task.

Automated traceability methods aggressively tackle these problems by decreasing the effort needed to construct and maintain a set of traceability links and by providing traceability across a much broader set of documents.

THE AUTOMATED SOLUTION

Automated traceability relies on various information-retrieval algorithms[4-8] that use techniques such as the *vector-space model*[4] or the *probabilistic network* model[6,7] to compute the likelihood of a link based on the occurrence and distribution of terms, as Figure 1 shows. During the preparsing phase, VSM- and PN-based algorithms remove stop words such as "this" and "the" representing extremely common terms from the artifact text and stem remaining terms to their root forms so that similar words such as "registers" and "registered" are considered equivalent. For code and Unified Modeling Language (UML) documents, the algorithms separate method names that follow standard conventions of uppercase and lowercase usage and appear in forms such as "getLastName()" into individual words such as "get last name." They also remove program-related keywords such as "while" and "for."

VSM- and PN-based algorithms calculate a similarity score based upon the frequency and distribution of terms in the artifacts. Both types of algorithms belong to the tf-idf (term frequency-inverse document frequency) family of information-retrieval methods, which consider rarer terms to be stronger indicators of a potential link than more common ones.

Probabilistic network model

To better understand how these algorithms work, consider the PN-based approach. In this model, the basic probability of a link between a query q and a traceable artifact a_j is defined as

$$pr(a_j|q) = \left[\sum_{i=1}^{k} pr(a_j|t_i) pr(q,t_i) \right] \Big/ pr(q).$$

The first two parts of the formula, $pr(a_j|t_i)$ and $pr(q,t_i)$, represent the dispersion of the term t_i within the artifact a_j and query q, respectively. These are estimated by computing the normalized frequency of terms. For example, $pr(a_j|t_i)$ is calculated by considering the frequency with which t_i occurs in the artifact, normalized over the total number of words in the artifact. This is represented as

$$pr(a_j|t_i) = \frac{\text{freq}(a_j,t_i)}{\sum_k \text{freq}(a_j,t_k)}.$$

Similarly, $pr(q,t_i)$ is calculated by considering the frequency with which term t_i occurs in query q, normalized over the total number of potential queries in which t_i occurs.

In the third part of the formula, $pr(q)$ is computed as

$$pr(q) = \sum_i pr(q,t_i).$$

using simple marginalization techniques and represents the relevance of the term t_i to describe the query concept q. The resulting probability is inversely proportional to the number of artifacts containing the index term, reflecting the assumption that rarer index terms are more relevant than common ones in detecting potential links.[4,5]

PN-based algorithms translate raw probability scores into confidence levels that are more intuitive to a human analyst and depict the likelihood of each link being correct.[9] Once the algorithm calculates the confidence scores, it returns a set of candidate links to the analyst for assessment. Our experience indicates that with adequate supporting documentation, an analyst can use this information to quickly make a decision about the correctness of a link.[9]

A trace query

Figure 2 shows the results returned by our Poirot: TraceMaker tool for the query "All trucks shall display a map of the de-icing route." Of the top 10 results displayed on the screen, five were actually correct; the analyst accepted these and rejected the other five. Several of the incorrect links were retrieved because the term "map" was used broadly across a number of requirements. A filtering feature (not shown) can be used by an analyst to help remove some of these extraneous links.

Experimental results

Researchers have conducted extensive experiments to assess the effectiveness of automated traceability methods. Results are typically evaluated using two metrics.[10] The *recall* metric measures the extent to which all of the desired links are retrieved:

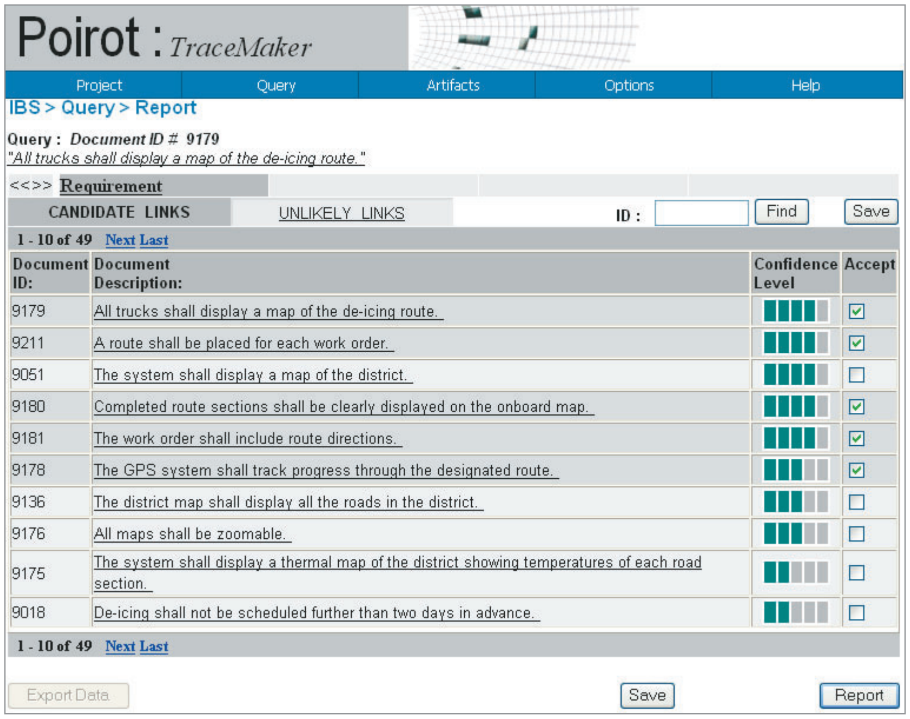
$$\frac{\text{Correct links} \cap \text{Retrieved links}}{\text{Correct links}}$$

The *precision* metric measures the percentage of retrieved links that are relevant:

$$\frac{\text{Correct links} \cap \text{Retrieved links}}{\text{Retrieved links}}$$

There is typically a clear tradeoff between recall and precision, so that as one increases the other decreases.

A third metric of particular importance in automated traceability measures the retrieval algorithm's ability to place more good links at the top of the can-



Query : Document ID # 9179
"All trucks shall display a map of the de-icing route."

1 - 10 of 49 Next Last

Document ID:	Document Description:	Confidence Level	Accept
9179	All trucks shall display a map of the de-icing route.	▮▮▮▮▯	✓
9211	A route shall be placed for each work order.	▮▮▮▮▯	✓
9051	The system shall display a map of the district.	▮▮▮▮▯	☐
9180	Completed route sections shall be clearly displayed on the onboard map.	▮▮▮▮▯	✓
9181	The work order shall include route directions.	▮▮▮▮▯	✓
9178	The GPS system shall track progress through the designated route.	▮▮▮▯▯	✓
9136	The district map shall display all the roads in the district.	▮▮▮▯▯	☐
9176	All maps shall be zoomable.	▮▮▮▯▯	☐
9175	The system shall display a thermal map of the district showing temperatures of each road section.	▮▮▯▯▯	☐
9018	De-icing shall not be scheduled further than two days in advance.	▮▮▯▯▯	☐

1 - 10 of 49 Next Last

Figure 2. Results from a trace query in Poirot: TraceMaker. Precision in the top 10 results was 50 percent.

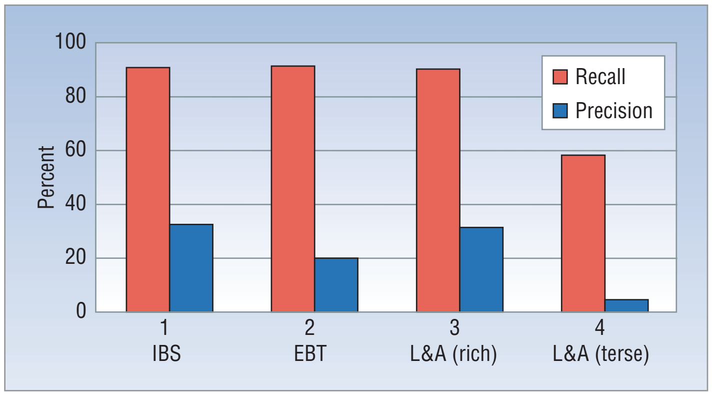


Figure 3. Trace retrieval results for several data sets. Data sets 1-3 all returned acceptable precision results ranging from 19 to 32 percent when recall was fixed close to 90 percent, while data set 4 returned an abysmal precision rate of 4 percent at recall levels of 58 percent.

didate list and to place bad links at the bottom. This can be calculated either by considering precision at various levels of recall or by utilizing the *lag* metric, defined as the average number of false links that occur in an ordered list of candidate links above each good link.[7]

Figure 3 shows experimental results for several data sets using the PN method. Because industrial applications of automated traceability are only considered successful when they achieve high recall levels, precision results are reported at a fixed recall level close to 90 percent.

The first data set represents the requirements and UML classes of a system for managing de-icing of roads in a public works department. The Ice-Breaker System (IBS) includes 180 functional requirements and 75 primary classes.[6] The second data set represents an event-based traceability (EBT) system that includes 54 requirements and 60 classes.[5] The third data set, provided by Siemens Logistics and Automation, represents a system for automating the layout of product lines in a factory. The artifacts used in the L&A experiment included business use cases (BUCs) and system use cases (SUCs), both represented as relatively terse requirements in RequisitePro. A subset of SUCs, labeled as "rich" in Figure 3, had additional associated information modeled in UML activity diagrams.

Data sets 1-3 all returned acceptable precision results ranging from 19 to 32 percent when recall was fixed close to 90 percent. In contrast, data set 4 performed poorly, returning an abysmal precision rate of 4 percent at recall levels of 58 percent.

These results suggest that not every data set will perform well. In fact, we have found that although automated traceability can provide a viable and economically beneficial solution, returning a good set of candidate traces requires properly structuring and specifying the artifacts.

BEST PRACTICES

Best practices for automated traceability fall into three categories. The first category describes best practices for establishing a traceability environment, the second describes the structuring and content of the artifacts, and the third describes a process for introducing automated traceability into an organization.

Traceability environment

The first three best practices relate to establishing the traceability environment.

Trace for a purpose. During the software development life cycle, project stakeholders create numerous work products that introduce the potential for a vast number of links. Although automated methods support after-the-fact traceability of legacy documents, stakeholders can perform day-to-day trace tasks more seamlessly if they determine in advance which types of artifacts they will trace and which types of artifacts they will trace to. For each identified traceability path, stakeholders must identify the artifact type at the origin and destination of each link, establish where each artifact is physically located, and determine in what format or third-party case tool it is stored.[3]

As an example, consider the traceability framework established for the L&A project's requirements. This project captured several different types of requirements during the project's life cycle from several unique role-oriented viewpoints. These included business goals, stakeholder requests, minimal marketable features (MMFs) that defined units of business value, MMF groups, BUCs, BUCs grouped by life-cycle stages, SUCs, and concrete system capabilities (CSCs), which were further refined into concrete system components.

A guiding principle during the requirements process was that it would express domain-specific requirements in the vocabulary of that domain's participants. As a result, stakeholders were interested in different subsets of artifacts and had different traceability needs. For example, business end users would need to answer questions such as "Do all MMFs trace to one or more business use cases?" or "Are all system requirements derived from one or more MMFs, and through which BUCs do they trace?" In contrast, as developers would need to know how accurately and completely their system requirements were aligned with business priorities, they would need support for queries such as "Do all CSCs trace to one or more SUCs?" Figure 4 shows the artifacts, traceability paths, and rationales for each link.

Making up-front decisions and explicitly modeling the traceability needs of the project stakeholders provide the necessary physical infrastructure to support automated traceability.

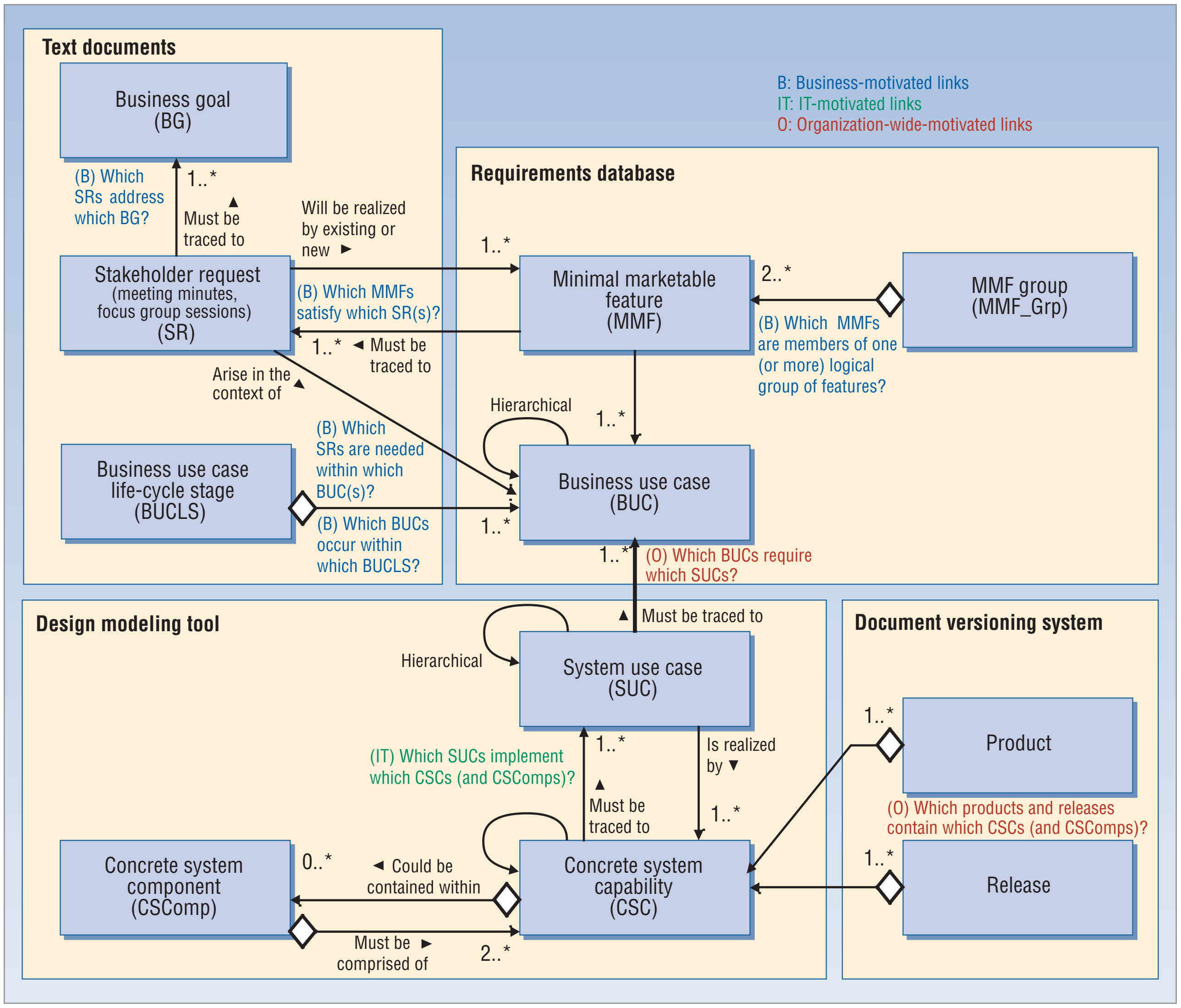

Text documents
Business goal (BG)
(B) Which SRs address which BG?
1..*
Must be traced to
Will be realized by existing or new
Stakeholder request (meeting minutes, focus group sessions) (SR)
(B) Which MMFs satisfy which SR(s)?
1..* Must be traced to
Arise in the context of
Business use case life-cycle stage (BUCLS)
(B) Which SRs are needed within which BUC(s)?
(B) Which BUCs occur within which BUCLS?
B: Business-motivated links
IT: IT-motivated links
O: Organization-wide-motivated links
Requirements database
1..*
Minimal marketable feature (MMF)
2..*
MMF group (MMF_Grp)
(B) Which MMFs are members of one (or more) logical group of features?
Hierarchical
1..*
Business use case (BUC)
1..*
1..*
(O) Which BUCs require which SUCs?
Must be traced to
Design modeling tool
Hierarchical
System use case (SUC)
(IT) Which SUCs implement which CSCs (and CSComps)?
1..*
Must be traced to
Is realized by
1..*
Concrete system component (CSComp)
0..*
Could be contained within
Must be comprised of
2..*
Concrete system capability (CSC)
Document versioning system
1..*
Product
(O) Which products and releases contain which CSCs (and CSComps)?
1..*
Release

Figure 4. Requirements framework developed for Siemens L&A. Making up-front decisions and explicitly modeling the traceability needs of project stakeholders provide the necessary physical infrastructure to support automated traceability for business-, IT-, and organization-wide-motivated links.

Define a suitable trace granularity. Project stakeholders also must decide on the appropriate level of trace granularity for each artifact type. For example, when tracing to UML class diagrams, they could generate a trace at the package, class, or method level.

Alexander Egyed and colleagues evaluated the economic value of tracing at lower levels of granularity measured by the effort needed to create the links versus value returned through tracing at various levels of precision.[11] Even ignoring the costs of maintaining links, they found that the benefits of improving the granularity of trace links beyond a certain level were very limited. In fact, their case study showed a tenfold increase in granularity returned only a twofold improvement in precision (correct links/all retrieved links).

Granularity must be carefully determined to effectively support stakeholders in their traceability tasks, while minimizing the effort involved to analyze and utilize the set of returned links. This can be especially problematic in large, weakly structured documents that might not contain clearly defined components at the desired granularity level. To mitigate this problem, automated traceability tools can cluster sentences into meaningful semantically related groups and then generate traces to those groups.

Support in-place traceability. In a software project, developers generally use a wide variety of third-party case tools to manage artifacts, and traceability should be provided to these artifacts as they reside within their native environments. We refer to this as *in-place* traceability.

Poirot delivers in-place traceability through an enterprise-level architecture, shown in Figure 5, that uses customized adapters to parse and reconstitute data from within third-party case tools. Each adapter uses the

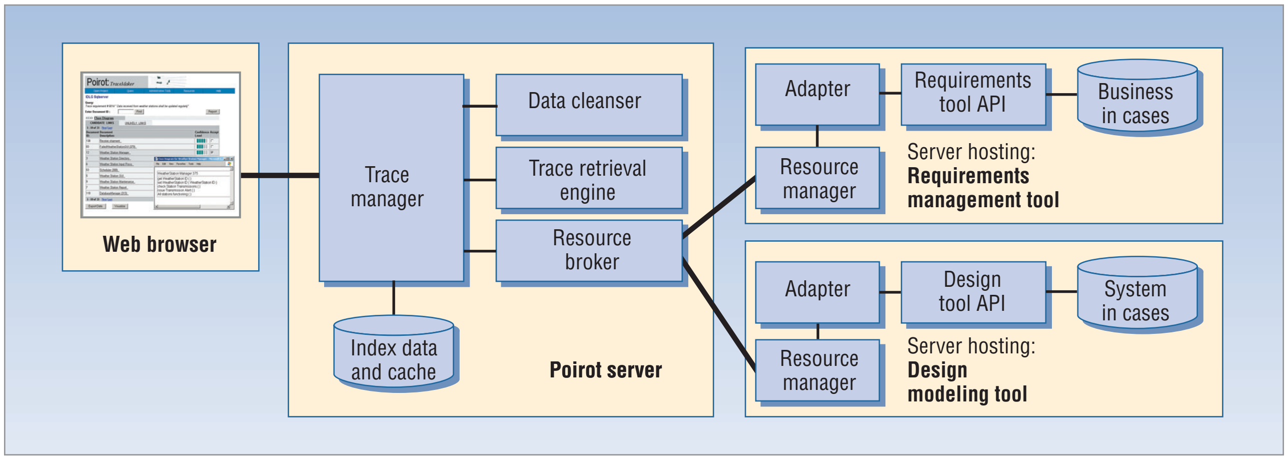


Figure 5. Poirot delivers in-place traceability through an enterprise-level architecture that uses customized adapters to parse and reconstitute data from within third-party case tools.

third-party case tool's API to retrieve the traceable data and transform it into a standard format. The data then passes to the Poirot server, which removes stop words, stems terms, counts term frequencies, and carries out other related activities. Customized adapters are currently available for a limited number of model types within various third-party case tools.

To facilitate reuse, an adapter must also allow the user to define filter conditions to retrieve targeted artifacts and sift out unwanted ones. For example, the SUCs in the L&A project represent a subset of activity diagrams embedded deep within an IBM Rational Rose hierarchy. A filter based on model types, pathname, partial filenames, and keywords differentiates SUCS from other activity diagrams.

Given the proprietary nature of many third-party case tools and the complexity that exists in the organization of artifacts within real-world data sets, it is essential to use an in-place traceability tool that is capable of establishing advanced filters and understands the nuances of specific third-party case tools. More general solutions such as enterprise-level search engines generally fail in this respect.

Creating traceable artifacts

In addition to the traceability environment, requirements that are well written and organized in meaningful ways also tend to return better traceability results than those that are haphazardly created. Several requirements best practices can significantly improve automated traceability results.

Use a well-defined project glossary. A well-constructed project glossary, defined during initial discovery meetings with stakeholders and used consistently throughout product development, will generally increase consistency in term usage and subsequently improve traceability.

In one of our experiments, we enhanced the basic probabilistic formula to weight terms found in the glossary more highly than other types of terms.[12] In the IBS project, for which developers used glossary terms throughout the design process, this enhancement factor led to precision improvements from 20 to 25 percent at recall values of approximately 95 percent, and from 55 to 74 percent in the top 5 percent of the links—that is, the change pushed good links to the top of the candidate list. In contrast, glossaries created for other projects at the end of the development phase led to no improvement in precision. In related work, Huffman Hayes and colleagues also showed that using a project glossary could improve the quality of retrieved traces.[7]

Write quality requirements. Requirements should exhibit generally accepted qualities of a good specification such as being correct, unambiguous, complete, consistent, prioritized, verifiable, understandable, identifiable, and so on. The qualities of completeness and conciseness are specifically important for automated traceability. In fact, this could be considered a best practice for requirements in general, but it is typically only consistently attainable in an organization that provides requirements specification training.

Construct a meaningful hierarchy. Including a strong hierarchy of information, such as headings within a requirements document or meaningful package names, can enable a trace retrieval tool to strengthen the semantics of individual requirements and help construct more accurate links.

Poirot's probabilistic formula incorporates hierarchical information as follows:

$$pr(a_j|q) = \left(\sum_{g \in pa_D(a_j)} \sum_i pr(a_j, g|t_i) pr(q, t_i) \right) \Big/ pr(q),$$

where $pa_D(a_j)$ represents the set of parents of a_j, that is, higher-level artifacts representing titles, subtitles, package names, and so on. The probability $pr(a_j,g|t_i)$ for any parent g of a_j is computed as

$$pr(a_j,g|t_i) \propto \frac{\text{freq}(a_j,t_i)}{\sum_k \text{freq}(a_j,t_k)} + \beta_D \frac{\text{freq}(g,t_i)}{\sum_k \text{freq}(g,t_k)}.$$

where the probability $pr(q,t_i)$ is defined as

$$pr(q,t_i) \propto \left[\text{freq}(q,t_i) + \beta_Q \sum_{h \in an_D(q)} pr(q|h)\ \text{freq}(h,t_i) \right] \Big/ n_i.$$

The probability term $pr(q|h)$ is computed as $pr(q|h) = 1/(M_{[q,h]} + 1)$, with $M_{[q,h]}$ being the distance of the ancestor from the query. The term $pr(q|h)$ can be regarded as a measure of the extent to which an ancestor document contributes information about a query q. A closer ancestor, such as a parent, is assumed to provide stronger information about q than more distant artifacts in the hierarchical topology.[7]

The optimal weights β_D and β_Q shown in these formulas were experimentally discovered as being 0.5 and 1, respectively. Experimental results showed that in certain data sets containing strong hierarchical information such as the IBS system, precision of trace results at a fixed recall value of 90 percent improved from approximately 26 to 31 percent when the retrieval process incorporated hierarchical data.

Bridge the intradomain semantic gap. As a single project typically includes various groups including customers, software developers, and systems engineers, stakeholders must often generate traces across domains containing artifacts developed using very different terminology to describe the same concepts. The semantic gap can be resolved by defining intradomain synonyms and utilizing a tool that can support domain-specific synonym matching. It is also essential to avoid reusing the same term to describe different concepts in different domains, as this will ultimately result in the generation of unnecessary links and decrease traceability result precision.

Create rich content. Elizabeth Hull, Ken Jackson, and Jeremy Dick[13] described the value of incorporating rationales and domain knowledge into the traceability infrastructure, and this approach can be useful for supporting automated traceability.

For example, the infrastructure shown in Figure 6a includes no rationale information; the only common terms shared between user requirement UR21 and the three system requirements (SR15, SR32, and SR53) are the stop words "the," "shall," and "to," and the word "vehicle," which is a relatively common term in this particular application and would likely return numerous links at fairly low levels of confidence. In contrast, the satisfaction argument included in Figure 6b introduces the additional shared terms "weight," "clearance," and "power," which are sufficient to support automated trace generation from the user requirement to each of the targeted system requirements.

The Siemens L&A data set for terse and rich SUCs illustrates the value of enhancing terse or hard-to-trace requirements with supporting rationale and domain information. This data set contained a total of 263 SUCs, including 221 terse ones containing an average of 5.3 words per requirement and 42 rich SUCs represented as activity diagrams. The highest achievable recall for the terse requirements was 58 percent with 4 percent precision, while the rich requirements achieved 90 percent recall with 31 percent precision. In the IBS, LC, and EBT data sets, which had average requirement word counts of 10.3, 17.0, and 13.3, respectively, requirements were traced successfully without the need for supporting data.

Introducing automated trace processes

Introducing new methods into an organization usually requires a planned process improvement initiative.

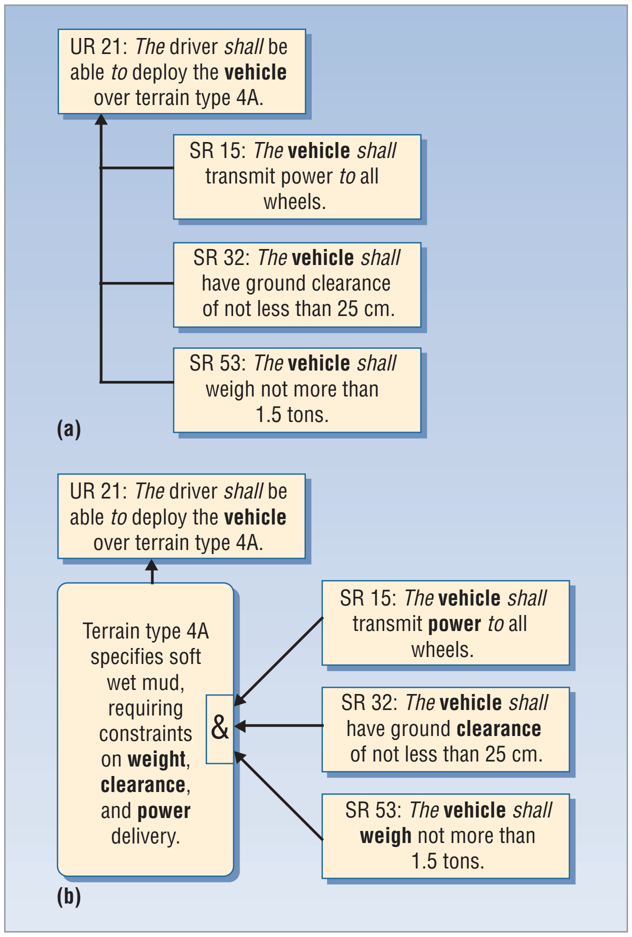

UR 21: *The* driver *shall* be able *to* deploy the **vehicle** over terrain type 4A.
SR 15: *The* **vehicle** *shall* transmit power *to* all wheels.
SR 32: *The* **vehicle** *shall* have ground clearance of not less than 25 cm.
SR 53: *The* **vehicle** *shall* weigh not more than 1.5 tons.
(a)
UR 21: *The* driver *shall* be able *to* deploy the **vehicle** over terrain type 4A.
Terrain type 4A specifies soft wet mud, requiring constraints on **weight**, **clearance**, and **power** delivery.
&
SR 15: *The* **vehicle** *shall* transmit **power** *to* all wheels.
SR 32: *The* **vehicle** *shall* have ground **clearance** of not less than 25 cm.
SR 53: *The* **vehicle** *shall* **weigh** not more than 1.5 tons.
(b)

Figure 6. Rich trace support for automated traceability, as illustrated by Hull, Jackson, and Dick.[13] (a) The only common terms shared between the user requirement and system requirements are meaningless stop words (italicized) and the word "vehicle"; automated traceability is not supported. (b) The introduction of additional shared terms (boldface) facilitates successful automated traceability between the system requirements and the user requirement. Figure adapted with permission.

This leads to the final best practice: *Use a process improvement plan.*

Three basic scenarios describe organizations that wish to adopt automated traceability methods.

The first scenario represents an organization with no traceability process currently in place. This organization desires to implement cost-effective traceability methods to more efficiently manage change and possibly meet more formal process-improvement initiatives. It might have large amounts of legacy data and no effective means for tracing it.

The second scenario represents an organization that has introduced traceability into a limited subset of artifacts such as high-level or critical requirements, code, and test cases but seeks to broaden its scope to include supplemental stakeholder documents without significantly increasing the effort involved.

The third scenario represents an organization that has traceability practices in place but is looking to replace them with more cost-effective and scalable methods.

Organizations can initiate a move to automated traceability by applying the best practices described here and implementing a pilot study designed to evaluate the traceability performance of a given data set. Experiments should be conducted to compare the results obtained from automatically created traces with manually created ones. Prior studies suggest that 20-35 percent precision should be achievable at recall levels of 90 percent. Where necessary, training and process improvement initiatives can improve results.

We derived these nine best practices from our experiences in applying automated traceability against several data sets. Researchers and practitioners interested in traceability continue to investigate automated methods to improve the performance of trace results and to provide increasingly sophisticated tool support.

The Center of Excellence for Software Traceability (www.traceabilitycenter.org), which is currently funded by the National Science Foundation, Siemens Corporate Research, and NASA, has been established to advance traceability research and to transfer traceability solutions to industry. Further information about automated traceability tools and other traceability support is available through the COEST Web site. ■

Acknowledgments

The work was partially funded by NSF grant CCR-0306303. Siemens Logistics and Automation in Grand Rapids, Mich., provided the Siemens data set. We also acknowledge the contributions of numerous DePaul University students who helped develop Poirot, especially Jun Lin, Rafal Habrat, Scott Bradley, Joseph Amaya, Grace Bedford, Oussama BenKhadra, Mary Brophy, Chuan Duan, Chan Chou Lin, Andre Robertson, and Xuchang Zou.

References

1. O.C.Z. Gotel and A.C.W. Finkelstein, “An Analysis of the Requirements Traceability Problem,” *Proc. 1st IEEE Int’l Conf. Requirements Eng.*, IEEE CS Press, 1994, pp. 94-101.
2. IEEE, *IEEE Std. 830-1998, IEEE Recommended Practice for Software Requirements Specifications*, 1998; http://standards.ieee.org/reading/ieee/std_public/description/se/830-1998_desc.html.
3. B. Ramesh and M. Jarke, “Toward Reference Models for Requirements Traceability,” *IEEE Trans. Software Eng.*, vol. 27, no. 1, 2001, pp. 58-92.
4. G. Antonio et al., “Recovering Traceability Links between Code and Documentation,” *IEEE Trans. Software Eng.*, vol. 28, no. 10, 2002, pp. 970-983.
5. J.H. Hayes, A. Dekhtyar, and S.K. Sundaram, “Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods,” *IEEE Trans. Software Eng.*, vol. 32, no. 1, 2006, pp. 4-19.
6. J. Cleland-Huang et al., “Goal-Centric Traceability for Managing Non-Functional Requirements,” *Proc. 27th Int’l Conf. Software Eng.*, ACM Press, 2005, pp. 362-371.
7. J. Cleland-Huang et al., “Utilizing Supporting Evidence to Improve Dynamic Requirements Traceability,” *Proc. 13th IEEE Int’l Conf. Requirements Eng.*, IEEE CS Press, 2005, pp. 135-144.
8. A. Marcus, J.I. Maletic, and A. Sergeyev, “Recovery of Traceability Links between Software Documentation and Source Code,” *Int’l J. Software Eng. and Knowledge Eng.*, vol. 15, no. 4, 2005, pp. 811-836.
9. X. Zou et al., “Supporting Trace Evaluation with Confidence Scores,” *Proc. 2005 Workshop Requirements Eng. Decision Support,* IEEE CS Press, 2005, pp. 1-7.
10. G. Salton and C. Buckley, “Term-Weighting Approaches in Automatic Text Retrieval,” *Information Processing and Management,* vol. 24, no. 5, 1988, pp. 513-523.
11. A. Egyed et al., “A Value-Based Approach for Understanding Cost-Benefit Trade-Offs During Automated Software Traceability,” *Proc. 3rd Int’l Workshop Traceability in Emerging Forms of Software Eng.,* ACM Press, 2005, pp. 2-7.
12. X. Zou, R. Settimi, and J. Cleland-Huang, “Phrasing in Dynamic Requirements Trace Retrieval,” *Proc. 30th Ann. Int’l Computer Software and Applications Conf.,* IEEE CS Press, 2006, pp. 265-272.
13. E. Hull, K. Jackson, and J. Dick, *Requirements Engineering*, 2nd ed., Springer, 2004.

Jane Cleland-Huang *is an assistant professor in the School of Computer Science, Telecommunications, and Information Systems at DePaul University’s Center for Requirements Engineering as well as president-elect of the Center of Excellence for Software Traceability. Her research inter-*

ests include requirements engineering, traceability, and software architectural design. Cleland-Huang received a PhD in computer science from the University of Illinois at Chicago. She is a member of the IEEE Computer Society and IEEE Women in Engineering. Contact her at jhuang@cs.depaul.edu.

***Brian Berenbach** is technical program manager for requirements engineering at Siemens Corporate Research, Princeton, New Jersey, with responsibilities for research, consulting, and corporate training. He has worked in the field of requirements engineering for more than 15 years and has been involved in requirements definition for a broad range of systems including medical, baggage-handling, mail-sorting, automated-warehouse, and embedded automotive systems. Berenbach received an MSc in physical chemistry from Emory University. He is a member of the IEEE Computer Society. Contact him at brian.berenbach@siemens.com.*

***Stephen Clark** is an enterprise solution manager at iRise, an application definition software and services company based in El Segundo, California. His research interests include requirements traceability, product requirements engineering methodologies, and organizational transformation through technology adoption. Clark is a member of the International Institute of Business Analysts. Contact him at StephenKClark@yahoo.com.*

***Raffaella Settimi** is an assistant professor in the School of Computer Science, Telecommunications, and Information Systems at DePaul University. Her research interests include information-retrieval methods, Bayesian learning, statistical methods for knowledge discovering under uncertainty, and statistical genetics. Settimi received a PhD in statistics from the University of Perugia, Italy. She is a member of the American Statistical Association and the Royal Statistical Society. Contact her at rsettimi@cs.depaul.edu.*

***Eli Romanova** is a master's student in the School of Computer Science, Telecommunications, and Information Systems at DePaul University's Center for Applied Requirements Engineering, and is currently interning in the Ultrasound Division at Siemens Medical Solutions, Mountain View, California. Her research interests include requirements engineering and human-computer interaction. Romanova received a BA in computer science from Hunter College. Contact her at eromanova@gmail.com.*

Who sets computer industry standards?
802.11
gigabit Ethernet
firewire
Together with the IEEE Computer Society, you do.
Join a standards working group at
www.computer.org/standards/