Enabling Traceability Reuse for Impact Analyses: A Feasibility Study in a Safety Context

Markus Borg
Dept. of Computer Science
Lund University
Lund, Sweden
markus.borg@cs.lth.se

Orlena C. Z. Gotel
Independent Researcher
New York, United States
olly@gotel.net

Krzysztof Wnuk
Dept. of Computer Science
Lund University
Lund, Sweden
krzysztof.wnuk@cs.lth.se

***Abstract*—Engineers working on safety critical software development must explicitly specify trace links as part of Impact Analyses (IA), both to code and non-code development artifacts. In large-scale projects, constituting information spaces of thousands of artifacts, conducting IA is tedious work relying on extensive system understanding. We propose to support this activity by enabling engineers to reuse knowledge from previously completed IAs. We do this by mining the trace links in documented IA reports, creating a semantic network of the resulting traceability, and rendering the resulting network amenable to visual analyses. We studied an Issue Management System (IMS), from within a company in the power and automation domain, containing 4,845 IA reports from 9 years of development relating to a single safety critical system. The domain has strict process requirements guiding the documented IAs. We used link mining to extract trace links, from these IA reports to development artifacts, and to determine their link semantics. We constructed a semantic network of the interrelated development artifacts, containing 6,104 non-code artifacts and 9,395 trace links, and we used two visualizations to examine the results. We provide initial suggestions as to how the knowledge embedded in such a network can be (re-)used to advance support for IA.**

***Index Terms*—impact analysis, issue management, traceability, data mining, semantic networks, visualization**

I. Introduction

To ensure system safety in areas such as medicine, nuclear engineering and aviation, rigorous standards for industry-specific software development are mandated [8], [9]. Developing software adhering to strict process requirements is costly [21], and this can inhibit making subsequent changes to the software [12]. However, as the lifecycles of many safety critical systems are often long, software evolution is inevitable, and this demands that change be handled with equal discipline.

One tedious development activity in the evolution of a safety critical software system is conducting Impact Analysis (IA) [1], [17], as specified by IEC 61511 [8]. IEC 61511 states that the impact of proposed software changes (e.g., for error corrections) should be formally analyzed before implementation. The IA typically consists of answering questions such as: "What code needs to be modified?" and "Which documents need to be updated to reflect the changes?" Moreover, to enable external agencies to assess process adherence and qualify for safety certification, the IA must be documented. As the IA requires the capture of trace links, to both code and non-code development artifacts, it is labor-intensive, dependent on system understanding and efficient navigation of the project information space [5].

IA is conducted by analyzing the relationships between development artifacts, and is classified as either *dependency analysis* or *traceability analysis* [3]. Dependency analysis can be automatically determined from source code, and this approach to IA is typically restricted to dependencies on the implementation level. Traceability analysis, on the other hand, includes relations among all types of artifact, and this broader approach to IA is unavoidable in the safety critical domain [27]. Unfortunately, automating IA among arbitrary types of artifact is a greater challenge; while semantic information about the relations is required, less formalism is generally available to determine this [18], so determination is often context dependent. The ensuing traceability analysis can be further helped or hindered by the representation used to describe the relations.

Several ways to represent trace links in software engineering have been proposed, such as traceability matrices, requirements dependency webs, and object models [18]. Another approach is to maintain a *semantic network* to represent semantic relations between concepts [25]. Semantic networks are a form of knowledge representation, and a typical output from ontological engineering. Their visual representation brings the potential for a quick analysis of the overall structure and relationships.

We suggest that a semantic network of development artifacts would provide a suitable knowledge base for a semi-automated approach to IA. Semantic networks have previously been shown feasible for software engineering search tools [19] and issue management [2], applications related to IA. Furthermore, semantic networks are navigable and flexible data structures, whose content can evolve over time [14], a crucial aspect in the dynamic context where IA is needed. However, creating and maintaining a high-quality semantic network that reflects the traceability among all the artifacts in a system demands significant manual effort at present, and one may question the return on investment.

Our proposal is to automatically build a semantic network, targeted to the IA task, based upon any IA that has already been undertaken for a system, effectively extracting and reusing the traceability associated with the most volatile

components. To explore the feasibility of our proposal, we study a large-scale and on-going safety critical development case, where the effort spent on IA is considerable, and where the engineers have requested IA support because the manual work involved is daunting [5]. The research questions of our feasibility study were:

RQ1 How can the traceability associated with past impact analyses be automatically extracted?

RQ2 What representational structure would facilitate the reuse of this traceability in future impact analyses?

RQ3 What additional analytical value would be gained by visualizing this traceability?

Section II describes the case company, and this sets out the motivation for the research highlighted in this paper. Section III then positions our research within the context of the wider traceability research challenges. Section IV outlines our approach and Section V presents the results of our preliminary investigations. Section VI summarizes the limitations and future work, while Section VII concludes.

II. Case Description

A large multinational company, active in the power and automation sector, applies the IA process that is the impetus for this research. The case company develops safety critical industrial control systems, governed by IEC 61511 [8]. The system under study within this company is certified to a Safety Integrity Level (SIL) of 2, as defined by IEC 61508 [9], and has evolved since the 1980s. Development was undertaken according to the first edition of IEC 61508, where only forward traceability is mandated (i.e., requirements→design→code→test). Also, while safety is a system characteristic, not all components in the system under study are safety classified. In the case company, the forward traceability is only maintained for development artifacts in the safety classified components, via references in documents.

As specified in IEC 61511 [8], the impact of proposed software changes should be analyzed before implementation. In the case company, this process is integrated in the Issue Management System (IMS). As part of the analysis, engineers are required to investigate impact, and report their results according to a project specific IA template (see Table I), validated by an external certifying agency. Several questions explicitly ask for trace links (6 out of 13 questions). The engineer is required to specify source code that will be modified, and also which related development artifacts need to be updated to reflect the changes (e.g., requirement specifications, design documentation, test case descriptions, test scripts and user manuals). Furthermore, the IA should specify which high-level system requirements are involved in the change, and which test cases should be executed to verify that the changes are correct once implemented in the system.

When creating an IA report for a new issue in the IMS, an engineer answers the questions of Table I. Trace links are specified using the full path to source code files, or by stating the formal artifact IDs of non-code development artifacts. The minimal level of granularity required in the IA report is file-level for source code, requirements are specified individually, and other artifacts are specified on a document-level. When the IA template is filled in, the resulting IA report is attached to the specific issue report in the IMS. The IA reports are essential artifacts used in the safety certification process. Due to their significance, they are reviewed internally before audits. Consequently, the trace links specified in the IA reports have the highest trustworthiness the company can provide.

TABLE I
Impact analysis template. Questions in bold font require explicit trace links. Adapted from Klevin [20].

Impact Analysis Questions	
1)	Is the reported problem safety critical?
2)	In which versions/revisions does this problem exist?
3)	How are general system functions and properties affected by the change?
4)	**List modified code files/modules and their SIL classifications, and/or affected safety related hardware modules.**
5)	**Which library items are affected by the change? (e.g., library types, firmware functions, HW types, HW libraries)**
6)	**Which documents need to be modified? (e.g., product requirements specifications, architecture, functional requirements specifications, design descriptions, schematics, functional test descriptions, design test descriptions)**
7)	**Which test cases need to be executed? (e.g., design tests, functional tests, sequence tests, environmental/electromagnetic compatibility tests, FPGA (Field-Programmable Gate Array) simulations)**
8)	**Which user documents, including online help, need to be modified?**
9)	How long will it take to correct the problem, and verify the correction?
10)	What is the root cause of this problem?
11)	How could this problem have been avoided?
12)	**Which requirements and functions need to be retested by product test/system test organization?**

Nevertheless, and due to the increasing system size, conducting an IA is a daunting task. While some tool support is available to identify the impact on source code (e.g., automated regression test suites, and breakpoints and crash dump files in a debugger), there is little support to identify the non-code impact. The available forward traceability only provides support if the engineer manages to identify impact on development artifacts in the safety classified components; in this case, the forward trace links can be investigated. As such, and despite the safety context, the engineers cannot routinely refer to any documented traceability to: (1) establish traces from new issue reports, (2) trace among non-safety development artifacts, or (3) trace backwards. In general, identification of impacted non-code artifacts is an information seeking activity [5]; if engineers do not know which artifacts will be impacted, they can either search for old IA reports related to the new issue, use key-word search in the document management system, browse documents in search for references to other artifacts, or ask more experienced colleagues.

III. Traceability Research in Context

The research that is described in this paper is tightly scoped with respect to the Grand Challenge of Traceability [16]. The premise is that the traceability that has been established to date

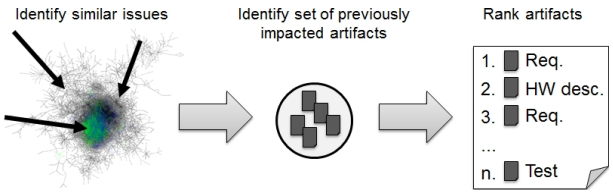

Fig. 1. Overview of a three-step recommendation process for IA. From left to right, identification of similar issues using text retrieval, identification of previous impact using neighborhood search, and artifact ranking.

for a particular software system is *purposed* and *trusted* (see [16]). We argue that this is a reasonable assumption to make for a company that develops safety critical software systems within a context that demands external certification of its IA process and reporting. We build upon this assumption to focus on addressing the challenge of making traceability *portable*, whereby: "traceability is exchanged, merged and reused across projects, organizations, domains, product lines and supporting tools" [16]. More specifically, we focus on research topic 6, to: "develop mechanisms to help extract, integrate and reuse traceability work products" [16].

Our research therefore seeks to extract the traceability that has been specified as a by-product of completing the IA reports that are mandated during the development of a system, and then to integrate the results as a semantic network of trace artifacts and links. While the extracted traceability is a partial view, this is the traceability associated with the most volatile parts of the system to date, so we argue a pragmatic starting point for future IA. Moreover, due to the restricted context of our research, the reliability of the traceability specified during IA is high. The objective is then to analyze and reuse this resulting traceability to predict the likely impact of future change requests for the same system as it evolves, and thereby reduce the manual effort currently demanded of the engineer. Accomplishing the reuse of traceability in the context of a single system and a single task (e.g., IA) is a necessary precursor to understanding and advancing the capability for wider and more habitual traceability reuse, a midterm destination on the roadmap for software and systems traceability research [15].

IV. Approach

Figure 1 depicts the three-step process to supporting IA that we envision. First, when a new IA is needed for an issue report, similar issue reports in the semantic network are identified using text retrieval techniques [10]. Second, neighborhood searches originating from the similar issue reports are used to identify a set of previously impacted artifacts. Third, the potentially impacted artifacts are ranked according to textual similarities and centrality measures in the network, and presented to the engineer. This would result in a recommendation system for IA [24].

In this paper, we only focus on the mechanism required to enable the last two steps of this process. Figure 2 provides an overview of a link mining approach that builds the semantic network, and is described in this section. It includes preprocessing steps, and visualization techniques to facilitate traceability analysis and reuse, as initially described by Marcus *et al.* [22].

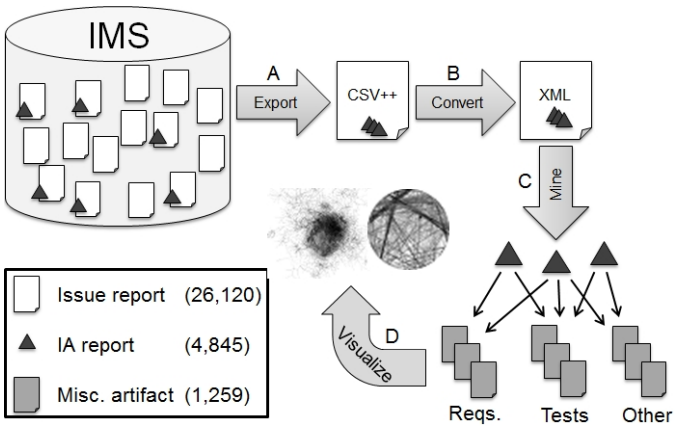

Fig. 2. The steps of the link mining approach. The resulting semantic network (the output from C) can be used as input to the process described in Figure 1.

A. Data Preprocessing

The IMS contained 26,120 issue reports submitted between 2000 and 2012. The issue reports correspond to different versions of a single software system (i.e., several different development projects), and range from low priority issues to project show-stoppers. All issue reports in the IMS were submitted by company-internal engineers, or as a consequence of defect reports collected by a customer support line. The issue reports in the IMS have unique identifiers, titles, and a natural language description. Moreover, there are 78 other fields that can be edited for each issue report, such as dates, identity of engineers, and relations to other issue reports. Our previous work discusses the issue reports of the IMS [6].

We exported all 4,845 issue reports from the IMS that had IA reports (out of 26,120) associated with them. Apparently, the majority of the reported issues are closed without formal IA reports. We found several reasons for this, such as non-repeatable issues, duplicate reports, and deferred changes. Also, issues not necessitating changes to any production code did not require IA, including pure document updates and changes to test code. We exported the issue reports to an extended CSV format, a feature provided by the IMS (A in Figure 2). The associated IA reports were contained within the CSV file as free text, with the IA template answers structured by question number (short descriptions in English or explicit trace links to development artifacts specified by unique IDs). We then converted the CSV file to an XML file (B in Figure 2), to be able to parse the semi-structured information using an off-the-shelf SAX parser.

B. Link Mining and Analysis

We base our approach on link mining [13] explicit references from IA reports. We used regular expressions to mine trace links from the IA reports (C in Figure 2). Due to the

fixed format of artifact IDs (described in Section II), this method could extract all correctly formatted trace links. Also, we used two heuristics to determine what type of trace links were extracted. First, due to the structure of the IA template (Table I), we knew to which question a specific trace link was an answer. As such, we could deduce that the meaning of the links was related to verification for Q7 and Q12. Second, as references to requirements and hardware modules have formats different to other development artifacts, we could also distinguish both *SpecifiedBy* links and *HWLinks*.

In the resulting semantic network, represented in GraphML [7], we used the graph editor *yEd*[1] to calculate centrality measures for each artifact, based on the number of incoming edges. Centrality measures show the relative importance of nodes in a network, and are used in social network analysis for key player identification [13] and on the Web for ranking search results [23]. For traceability, they can be used to identify the most often impacted artifacts. Potentially, the centrality measures could constitute input to an artifact-ranking engine in a tool providing support for IA. We also used *Gephi* (v. 0.8.1 Beta) [3] to compute structural statistics of the semantic network, displaying characteristics such as how many trace artifacts an IA report on average is linked to. These are shown in Table II and discussed in Section V.

C. Visualization

Visualization can improve the comprehension of multi-dimensional data sets [26], and can therefore potentially help to better understand the structure, behavior and evolution of software systems [11]. By enabling interaction with a visual representation of a traceability network (via zooming, panning, filtering, etc.), practitioners could begin to explore the traceability of a complex system at many levels. Due to the pattern recognition ability of human vision, we propose that this could support deviation detection in the network, such as variations in trace link densities, and pronounced clusters of trace artifacts, and so assist IA.

We visualized the semantic network using two different layouts in yEd (D in Figure 2). The *organic layout* is based on the force directed layout paradigm. The graph nodes are treated like physical objects with mutually repulsive forces, while the edges are considered to be metal springs attached to nodes, producing attractive forces if they are long and repulsive forces if they are short. By simulating these physical forces, the organic layout finds a minimum of the sum of the forces emitted by nodes and edges. Output from organic layouts typically show inherent symmetric and clustered graph structures, useful for finding highly connected backbone regions in a graph. This layout therefore brings a potential to discover artifact clusters and for a cluster to be distinguishable, its trace links must predominantly target other artifacts within the cluster. When conducting an IA for an artifact in a visually distinguished cluster, an engineer faces a significantly reduced search space. Moreover, a semantic network without any prominent clusters could be an indication of potential maintainability issues.

The *circular layout* displays a graph as an interconnected ring topology. The layout produces partitions of nodes depending on the connectivity structure of the graph. Thus, the circular layout emphasizes group and tree structures. We configured the layout to produce a single circle, with all trace artifacts placed on the circle perimeter. While the positioning of artifacts on the perimeter does not carry a direct meaning, artifacts with many connections in common are laid out closer together, as the layout attempts to minimize the total length of the links. Thus, when conducting an IA, a connection across the circle disk is an example of a trace link that might be difficult to anticipate in the system. An engineer conducting an IA could use this layout to identify several issue reports targeting a single artifact on another part of the circle perimeter. On the other hand, when conducting an IA for an artifact positioned in an area with nothing but connections along the circle perimeter, there is an indication that fewer cross-cutting concerns are involved, so the risk of unexpected side effects is lower. In both cases, the engineer gets visual support in identifying potentially impacted artifacts that are either positioned close to the investigated issue or across the circle.

V. Demonstration

This section discusses the application of the approach to the case data, and addresses our research questions.

A. Traceability Extraction (RQ1)

Using link mining, we discovered a semantic network of 6,104 development artifacts connected by 9,395 trace links, grouped in 1,885 weakly connected components (i.e., sets of nodes in which there exists a path from any node to any other). Table III shows that 1,259 non-code development artifacts have been pointed out as impacted from 9 years of formal IA, indicating that the system contains a set of repeatedly impacted artifacts. Table III shows the different types of trace links extracted, as well as their frequencies. About half of the trace links (3,996, 42.5%) target requirements (fine-granular links), while the rest target artifacts with a document-level granularity.

Computing the structural statistics for the semantic network (Table II) shows that the majority of the development artifacts are part of one large interconnected component, containing 65% of the artifacts (3,974). This dominating component also contains 97.0% (9,111) of the trace links in the network. However, as most of the trace artifacts are not directly linked, both the entire network and the largest component are by definition sparse. The semantic network has a diameter of 17 (i.e., the longest possible path between two artifacts), and the average path length in the largest component is 6.0. Both figures indicate that the connectedness of the largest component is high despite its sparsity. As such, from a given issue report, it is typically possible to identify a set of previously impacted

[1]yEd v. 3.9.2 http://www.yworks.com/en/products_yed_about.html

TABLE II
STRUCTURAL STATISTICS OF THE SEMANTIC NETWORK.

Measure	Total network (A)	Largest component (B)	A \ (B ∪ C)
#Nodes	6,104	3,974	320
#Edges	9,395	9,111	284
#Components	1,885	1	74
#Isolated nodes (C)	1,814	0	0
Diameter	17	17	7
Average path length	1.0	6.0	2.0
Average node degree	1.5	2.2	0.8

artifacts by following trace links, often several degrees away. If IA for a new issue is required, and a similar issue report is found in the largest component, a neighborhood search in the network can return potentially impacted artifacts.

There are also several components in the semantic network with fewer connected development artifacts. As many as 1,814 issue reports (37.4%), all having documented IA reports, have no specified trace links at all. Thus, they end up as isolated nodes in the network (see Table II), with no relations to other non-code artifacts. One explanation, expressed by an engineer at the company, is that IA reports often state that "the change deals with implementation details, no documentation covers this level of detail". This statement could be verified by adding trace links to source code to the semantic network, to investigate whether the occurrence of isolated nodes is reduced. Finally, 316 development artifacts (5.2% of the nodes) are part of interconnected components containing between 2 and 52 nodes. Generally, the potential to reuse trace links is proportional to the component size, as the potential builds on the number of links that can be followed.

Due to the formal use of artifact IDs for specifying impact relations at the company, *link mining allowed us to extract the traceability associated with past IA for the case software system automatically*.

B. Traceability Representation (RQ2)

We stored the extracted traceability information in a semantic network represented by GraphML. As such, we could represent the semantics of both trace artifacts and links. As presented in Table III, most nodes in the semantic network are issue reports (4,845, 79.4%), and the number of impacted non-code development artifacts is limited to 1,259.

To further explore the network structure, we ranked the artifacts in the semantic network based on their centrality measures. Our results showed that among the highest ranked artifacts, several types of artifact were represented, including system requirements, hardware descriptions, functional design descriptions and test specifications. We presented the top ten artifacts to three engineers at the company for an initial validation, who confirmed they were central to the system, i.e., known to often be impacted by changes. This suggests that the structure of the semantic network carries meaning, and that further inquiry into whether it could be used for the ranking of potentially impacted artifacts is motivated.

Regarding the semantics of the extracted trace links, the largest proportion were of type *SpecifiedBy* (3,996, 42.5%)

TABLE III
TYPES OF NODES AND LINKS EXTRACTED FROM THE IA REPORTS. NOTE THAT ALL LINKS HAVE ISSUE REPORTS AS SOURCES.

Node type	Description	#Nodes
Issue report	A single submitted issue report	4,803
Safety critical issue report	Issue report flagged as safety critical	42
Misc. development artifacts	Unique requirements, test case descriptions, user manuals, hardware descriptions, etc.	1,259
Trace link type	**Description**	**#Links**
SpecifiedBy	Link to a specific requirement describing the impacted behavior.	3,996
VerifiedBy	Link to a test specification that needs to be executed.	2,297
HWLink	Link to a hardware description that is impacted by the issue.	2,327
OtherLink	Trace link whose meaning could not be extracted.	775

(see Table III), partly due to their finer granularity. The number of *HWlinks* to hardware descriptions (2,327, 24.8%) and *VerifiedBy* links (2,297, 24.4%) were about equal. For 775 trace links (8.2%), we could not deduce the link semantics.

The semantic network is a dynamic data structure, which supported our work in this study. We were able to programmatically add new types of trace artifacts and links to the GraphML representation in an iterative manner, as new IA reports were added to the ongoing development Furthermore, using a graph editor, we could easily interact with the semantic network and follow specific traces. The possibilities to freely zoom, filter information, and search information (further discussed in Section V-C) were considered positive by the engineers in the company, and also deemed as necessary due to the complexity of the data. Thus we argue, compared to textual representations and traceability matrices, a semantic network better supports interaction and modification, especially with appropriate tool support. Moreover, while textual representations and matrices do not scale, the potential power of networked structures increases as more data is added, boosting its value and predictive power.

Our initial experiences suggest that *a semantic network could be used as a suitable knowledge representation for traceability reuse*. Such a structure meets Jönsson and Lindvall's requirements on consistency and semantics regarding input to automated IA tools [18]. On the other hand, while Jönsson and Lindvall also claim that completeness is an important aspect, we argue that a semantic network can deliver support also with partial traceability information. For example, even if our semantic network does not reflect the total traceability of a system, the fact that it does reflect the traceability of the most volatile components means it could still be used as a starting point and queried about potentially impacted artifacts for new IA. While the knowledge base would be smaller, the reliability of the information and thus the ability to signal some previous impact would remain, and constitute an alternative to completely manual work starting from scratch. Furthermore, as the semantic network can be seamlessly recreated and allows incremental additions, new trace links and artifacts can be

added after-the-fact, either as made available or in batch.

C. Traceability Visualization (RQ3)

Figures 3 and 4 show two overviews of the largest extracted component in the semantic network, containing 3,974 development artifacts (65%) and 9,111 trace links (95%). High-resolution color figures are also available online [4].

Figure 3 uses an organic layout. A highly connected region of issue reports and other development artifacts is clearly visible in the center of the figure, where the individual nodes are completely covered by edges. The figure does not display any clear clusters not dwarfed by the backbone region, suggesting that the long software evolution has resulted in a deeply intertwined system.

Figure 3 also displays three magnified views of the semantic network, all in the outer parts of the figure. Magnification A shows a network substructure where 15 issue reports (gray nodes) link to a single hardware description (black node). An engineer conducting IA for a similar issue could spot that this issue under investigation would most probably also impact this hardware description. Magnification B highlights an issue report with two outgoing trace links (encircled gray node), one *SpecifiedBy* link to an individual requirement (black ellipse) and one *VerifiedBy* link to a test case description (black triangle). An engineer performing IA for a related issue could use this view to quickly spot which requirement could be impacted by this change and which test case verifies this issue. Magnification C displays two issue reports flagged as safety critical (gray nodes) that both have outgoing *VerifiedBy* links to the same test case description (parallel dashed edges). An engineer performing IA for a similar issue could use this information both for test case selection, and as a warning that also the new issue under investigation might be safety critical. All three magnifications exemplify traceability captured in the semantic network, information that could be used in tool support to deduce and then explore potential impact.

Figure 4 uses a circular layout. The many trace links crossing the circle disk again shows that the development artifacts are highly interconnected. Magnification A displays a close-up of a highly connected part of the network, with trace links of all types along the circle perimeter and across the disk. Only two regions of the network, both on the right side of the figure (see C in the figure), are more separated. Magnification B presents another recurring pattern, where several nodes target a single node on another part of the perimeter, signaling a risk that there are important dependencies, something easily identified with this layout. In this example, several issue reports (just above C in the figure) are connected to the same hardware description (in magnification B). An engineer performing an IA on issues similar to those just above C in Figure 4 gets a visual indication that the hardware description (in magnification B) should be included in the investigation. The most pronounced observation from the circular layout is that most parts of the circle perimeter are connected by crosscutting trace links (across the disk), confirming complex

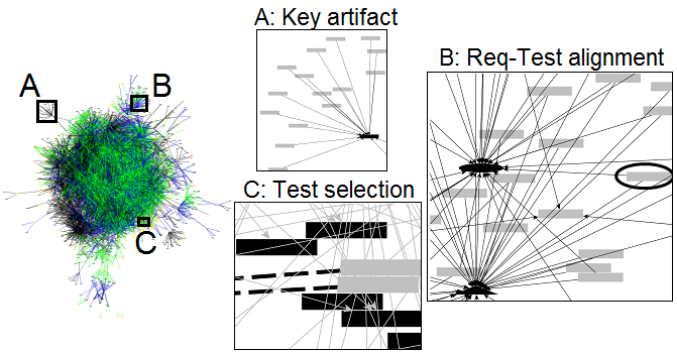

Fig. 3. The largest component of the semantic traceability network, visualized using an organic layout. The figure shows a highly connected backbone region.

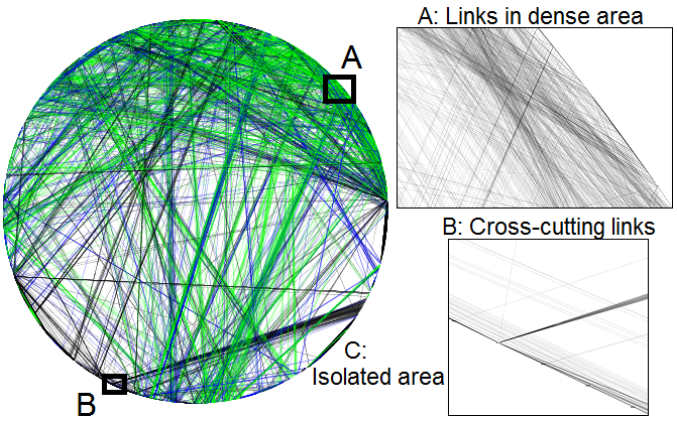

Fig. 4. The largest component of the semantic traceability network, visualized using a single circle layout. The figure shows the complex interconnectivity.

dependencies within the software system, and motivating the need to support IA.

Jönsson and Lindvall claim that the complex relations among artifacts in large software systems are often hard to visualize [18]. Our results confirm this statement, as the high link densities in Figures 3 and 4 obstruct most of the nodes in the semantic network. Consequently, effort should be made to find alternative ways to filter or slice the data to enable further discoveries from visual analytics. However, as our figures embed both link structure and semantics, it enables filtering based on these characteristics, and the potential to interactively explore the traceability is already promising.

VI. Limitations and Future Work

The research presented in this paper remains exploratory and focuses on the first two steps of the process described in Figure 1. Future research is required to further evaluate using a semantic network as a representation for mined trace links, and to explore alternatives. Then, work needs to be directed toward building the recommendation system and examining how it could facilitate traceability reuse in an IA tool. However, tool support for IA should itself be examined carefully, as increasing the level of automation for a task always presents new risks. An IA tool might lull engineers into a false sense of security, thus decreasing their tendency to identify impact

that is not suggested. Thus, to mitigate this risk, we need to qualitatively study how engineers in the case company currently conduct manual IA, to better understand how tool support for IA could be successfully deployed.

We focused on the relations among the non-code artifacts in our feasibility study (e.g., requirements, test specifications, and hardware descriptions), as they were considered more challenging to trace in the case company. One could question whether studying impacted development artifacts without consideration of impacted source code is a reasonable approach to take. While our results show that a considerable number of trace links can be extracted, and support can potentially be provided for a tool-neglected side of IA, future work should aim at also integrating code artifacts.

However, to focus on integrating source code in our case presented some challenges. First, the code is generally more volatile than the internal documentation in the case company; source code files are modified more frequently, added and removed at a higher rate, and turn obsolete at a faster pace. One option to explore this could be to include time information in the semantic network. As timestamps are available for all IA reports, the age of the created traces is possible to determine. This could be used to identify obsolete artifacts, and provide additional input to a ranking engine. Second, we suspected that it would be harder to automatically extract trace links to source code, as they are not consistently specified in the IA reports. We noticed significant variations in how engineers specify items in the code repository, indicating a need for more advanced parsing. Decreasing the granularity of trace links to source code, by primarily extracting links on a component level, could possibly mitigate this.

Another limitation of this initial work is that it does not capitalize upon the traceability that has already been established for the system. In the case company, the forward traceability was restricted to safety-critical components of the system and embedded in documentation. Our intention was to augment the tools that the engineer had available to support IA to assess feasibility. Future work should examine how to use the extracted traceability as a way to either validate or update the official traceability record, and then use the consolidated traceability to support IA more comprehensively.

VII. Conclusions

This paper focuses on IA in a safety critical software development context, where specifying explicit trace links to impacted development artifacts is currently a challenging and manual activity for engineers. We explored the feasibility of extracting and reusing the traceability from previously completed IA reports to support future IA, using link mining, and semantic network analyses, aided by visual techniques. This initial work is a first step toward more comprehensive support for IA in practice. The research also begins to tackle the challenge of achieving traceability that is portable, by: "develop[ing] mechanisms to help extract, integrate and reuse traceability work products" [16]. This is one of the many interim destinations on the road to traceability ubiquity [15].

References

[1] H. Ali, M. Rozan, and A. Sharif. Identifying challenges of change impact analysis for software projects. In *Proc. of the Int. Conf. on Innovation Management and Technology Research*, pages 407–411, 2012.

[2] A. Ankolekar, K. Sycara, J. Herbsleb, R. Kraut, and C. Welty. Supporting online problem-solving communities with the semantic web. In *Proc. of the 15th Int. Conf. on World Wide Web*, pages 575–584. ACM, 2006.

[3] S. Bohner. *Software Change Impact Analysis*. IEEE Computer Soc. Press, 1996.

[4] M. Borg. Accompanying website "Traceability network". http://serg.cs.lth.se/index.php?id=70335.

[5] M. Borg. Findability through traceability: A realistic application of candidate trace links? In *Proc. of the 7th Int. Conf. on Evaluating Novel Approaches to Software Engineering*, pages 173–181, 2012.

[6] M. Borg, D. Pfahl, and P. Runeson. Analyzing networks of issue reports. In *Proc. of the 17th European Conf. on Software Maintenance and Reengineering*, pages 79–88, 2013.

[7] U. Brandes, M. Eiglsperger, J. Lerner, and C. Pich. Graph markup language (GraphML). In R. Tamassia, editor, *Handbook of Graph Drawing and Visualization*. CRC Press, To appear.

[8] International Electrotechnical Commission. IEC 61511-1 ed 1.0, safety instrumented systems for the process industry sector, 2003.

[9] International Electrotechnical Commission. IEC 61508 ed 1.0, electrical/electronic/programmable electronic safety-related systems, 2010.

[10] A. De Lucia, A. Marcus, R. Oliveto, and D. Poshyvanyk. Information retrieval methods for automated traceability recovery. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*. Springer, 2012.

[11] S. Diehl. Software visualization. In *Software Visualization*, pages 1–13. Springer Berlin Heidelberg, 2007.

[12] S.W. Fraser. Can safety-critical software be flexible? In *Proc. of the Int. Conf. on Information Reuse and Integration*, pages 588–593, 2003.

[13] L. Getoor and C. Diehl. Link mining: a survey. *SIGKDD Explorations Newsletter*, 7(2):3–12, 2005.

[14] W. Gorka, A. Piasecki, and L. Bownik. Application of semantic networks in natural language issues. In S. Safeeullah, editor, *Engineering the Computer Science and IT*. InTech, 2008.

[15] O. Gotel, J. Cleland-Huang, J. Huffman Hayes, A. Zisman, A. Egyed, P. Grünbacher, and G. Antoniol. The quest for ubiquity: A roadmap for software and systems traceability research. In *Proc. of the 20th Int. Requirements Engineering Conf.*, 2012.

[16] O. Gotel, J. Cleland-Huang, J. Huffman Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, and J. Maletic. The grand challenge of traceability (v1.0). In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*. Springer, 2012.

[17] N. Hrgarek. Certification and regulatory challenges in medical device software development. In *Proc. of the 4th Int. Workshop on Software Engineering in Health Care*, pages 40–43, 2012.

[18] P. Jönsson and M. Lindvall. Impact analysis. In A. Aurum and C. Wohlin, editors, *Engineering and Managing Software Requirements*, pages 117–142. Springer, 2005.

[19] G. Karabatis, Z. Chen, V. Janeja, T. Lobo, M. Advani, M. Lindvall, and R. Feldmann. Using semantic networks and context in search for relevant software engineering artifacts. In *Journal on Data Semantics XIV*, pages 74–104. Springer, 2009.

[20] A. Klevin. *People, process and tools: A Study of Impact Analysis in a Change Process*. Master thesis, Lund University, http://sam.cs.lth.se/ExjobGetFile?id=434, 2012.

[21] J.C. Knight. Safety critical systems: challenges and directions. In *Proc. of the 24th Int. Conf. on Software Engineering*, pages 547–550, 2002.

[22] A. Marcus, X. Xie, and D. Poshyvanyk. When and how to visualize traceability links? In *Proc. of the 3rd Int. Workshop on Traceability in Emerging Forms of Software Engineering*, pages 56–61, 2005.

[23] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, 1998.

[24] M. Robillard, R. Walker, and T. Zimmermann. Recommendation systems for software engineering. *IEEE Software*, 27(4):80–86, 2010.

[25] J. Sowa. *Principles of Semantic Networks*. Morgan Kaufmann, 1991.

[26] E. Tufte. *Envisioning information*. Graphics Press, Cheshire, 1990.

[27] R. Turver and M. Munro. An early impact analysis technique for software maintenance. *Journal of Software Maintenance: Research and Practice*, 6(1):35–52, 1994.