Analyzing Networks of Issue Reports

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

Dietmar Pfahl
Dept. of Computer Science
University of Oslo
Oslo, Norway
dietmarp@ifi.uio.no

Per Runeson
Dept. of Computer Science
Lund University
Lund, Sweden
per.runeson@cs.lth.se

***Abstract*—Completely analyzed and closed issue reports in software development projects, particularly in the development of safety-critical systems, often carry important information about issue-related change locations. These locations may be in the source code, as well as traces to test cases affected by the issue, and related design and requirements documents. In order to help developers analyze new issues, knowledge about issue clones and duplicates, as well as other relations between the new issue and existing issue reports would be useful. This paper analyses, in an exploratory study, issue reports contained in two Issue Management Systems (IMS) containing approximately 20.000 issue reports. The purpose of the analysis is to gain a better understanding of relationships between issue reports in IMSs. We found that link-mining explicit references can reveal complex networks of issue reports. Furthermore, we found that textual similarity analysis might have the potential to complement the explicitly signaled links by recommending additional relations. In line with work in other fields, links between software artifacts have a potential to improve search and navigation in large software engineering projects.**

***Keywords*-issue reports; impact analysis; safety development; link mining; information retrieval;**

I. Introduction

In large projects, Issue Management Systems (IMS) contain thousands of issue reports. Issue reports must be analyzed and are managed through different states, until finally resolved and filed [12], [14]. Typically, issue reports contain a Natural Language (NL) description of the experienced issue, preferably instructions on how to repeat the problem, and additional information such as product version, priority, severity, and comments.

In development contexts where functional safety is important, software organizations have to follow certain standards (e.g., IEC 61511 [13]) requiring a thorough impact analysis for each individual issue report. The types of impacts result from questions such as: "What code needs to be modified?" and "Which documents need to be updated to reflect the changes?" The types of impacts can vary from context to context but no matter what is required in a specific case, developers in charge of analyzing issue reports must identify the traces from the issue to many other kinds of artifacts with as much accuracy as possible. This is a challenging and costly task. How could this task be supported?

The first possibility (A) to provide support to the analyzing developers is to help them decide whether the issue report at hand is a duplicate of another (open or closed) issue report. If an issue is the duplicate of an already closed issue, then the work is done. If an issue is the duplicate of another, still open, issue, then the engineer knows that only one of the reported issues needs to be resolved, the issue reports should preferably be merged [4], and the impact analysis effort must be invested only once. Previous work suggest using text retrieval techniques to support duplicate detection [17], [31], [25].

The second possibility (B) is to retrieve and exploit impact information (traces) contained in similar or related issue reports (cf. Figure 1). Closed issue reports, that have been fully analyzed and where all the impacts (traces to other artifacts) are known and documented, are particularly valuable in this regard.

The third possibility (C) is to provide developers with techniques and tools that help them identify traces to affected artifacts directly (cf. Figure 1). While the third option is the most appealing kind of support, it also is the most difficult to establish. One proposed way to support impact analysis is to recover trace links between software artifacts based on textual similarities [1], [21], [9].

In this paper, we present research related to support options A and B, i.e., helping to identify duplicate issue reports, and helping to identify similar or related issue reports and exploiting information contained in those reports. We conducted a study on issue reports from a proprietary safety-critical context (the Safety IMS) and, due to easy accessibility of IMSs in open source software (OSS) development projects, a deeper study using issue reports from the Android project (the Android IMS).

Apart from issue descriptions, the ∼20.000 issue reports in the Android IMS also contain comments that are added over the life-time of an issue report. Issue comments in the Android IMS consist primarily of natural language text, but sometimes a developer, for some purpose, makes the effort to explicitly point to another issue report using HTML hyperlinks. In this paper we present an exploratory study about the nature of such links, i.e., we are interested in understanding what meaning they have, and if they represent

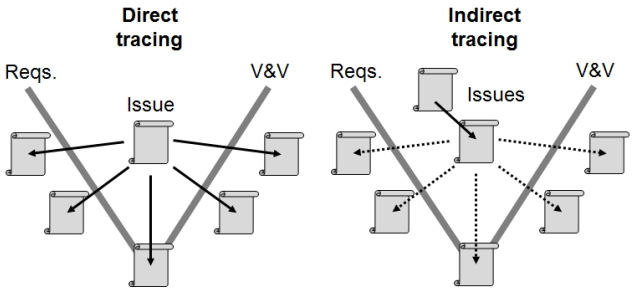

Figure 1. Impact analysis in the V-model. On the left, showing direct from an issue report to impacted artifacts (possibility C). On the right, showing traces to impacted artifacts via other issue reports (possibility B).

system knowledge that helps developers identify duplicates (cf. possibility A described above) or related issue reports (cf. possibility B described above).

Section II describes the impact analysis work task in the safety-critical context, and the typical content of issue reports in both the Safety IMS and the Android IMS. Section III presents how we conducted this study, and Section IV shows our results. In Section VII, we discuss our findings in light of the impact analysis work task and outline how issue networks can be used to improve navigating software engineering information spaces.

II. Motivation and data provenance

As specified in IEC 61511, the impact of proposed software changes must be analyzed before implementation [13]. Issues in an IMS, i.e., defect reports and change requests, are typically administered by a Change Control Board (CCB). The board distributes issues to responsible engineers for investigation. One way to formalize this activity is to use a fixed template containing questions that must be answered. Such templates have been reported to explicitly request trace links from an issue report to artifacts of other types [20], [5]. Example questions from such a template include:

- What code files/modules are (directly/indirectly) affected and have to be modified?
- What system functions are affected?
- Which documents are affected (requirements documents, design documents)?
- Which test cases are affected?
- Which user documents need to be modified?
- Which requirements and functions need to be re-tested?

We conducted our study on two IMSs. The Safety IMS contains 26,120 issue reports (63% defect reports), originating from embedded development in the domain of industrial control systems. The software is certified to a Safety Integrity Level (SIL) of 2 as defined by IEC 61508. The number of involved developers is in the magnitude of hundreds, distributed on sites in Europe, North America and Asia. The typical project duration is 12-18 months, and follows an iterative stage-gate project management model.

The issue reports, submitted between 2000 and 2012, correspond to several different system versions of a product, ranging from low priority issues to project stoppers. All issue reports in the Safety IMS were submitted by company-internal engineers, sometimes, however, as a consequence of defect reports collected by a customer support line. The content of the public Android IMS, 20,176 issue reports (80% defect reports), was made available as part of the MSR 2012 Mining Challenge [29]. In the Android project, 'medium' is by far the most commonly assigned issue priority, comprising 99.4% of all issues. Even though the Android open source project is led by Google, anyone with a registered account can access the IMS.

Issue reports in the Safety IMS have unique identifiers, titles, and a NL description. Moreover, there are 78 other fields that can be edited for each issue report. The fields are, for example, related to dates (for submission, disposition, validation, closing), identity of engineers (e.g., submitter, owner, validator, and closer), version information (e.g., build versions of the system and its components and targeted releases), and SIL level. For each issue report, there are also several attached 'notes' including decisions from the CCB, developer communication, clarifications from the report author etc. Unfortunately, we did not have access to the attached notes in this study.

An Android issue report contains a unique identifier, a title, a NL description, three nominal variables (status, type, and component), and an ordinal variable (priority). Also, an issue report holds information about opening and closing dates, the developer who reported it, the developer who currently owns it, and its number of stars (used in the IMS to let developers vote for issues and subscribe to email change notifications.) Furthermore, it is possible for developers to add comments to issues. Informal communication is favored over formal documents in OS projects [28], such as enabled by the comments. In general, communication in distributed development projects is a major challenge [12]. The over 100,000 issue comments in the Android IMS suggest that the comments constitute a significant channel of information exchange, and enabled us to study how issues are related.

III. Method

To explore the possibility to support the impact analysis using information in IMSs, we conducted a study combining data mining and qualitative artifact analysis. The Research Questions (RQ) below express the goals of this initial study. The RQs target possible support for (A) duplicate detection and (B) finding related issues, as introduced in Section I. As previously explained, RQ2 and RQ3 (addressing semantics of links) are limited to the Android IMS due to the availability of information.

RQ1 What sort of explicit issue networks can be discovered through link mining in IMSs?

RQ2 What do explicit links between issue reports (i.e., explicit issue networks) represent in the Android IMS?

RQ3 What kind of relations between issue reports does textual similarity analysis detect in the Android IMS?

Our first analysis was based on link mining. It is an approach commonly applied to support tasks such as object ranking, link prediction and sub-graph discovery [11]. Our approach was to extract *explicit links* among issue reports in the IMSs. The link structures lead to directed graphs of issue reports. We refer to a set of issue reports connected by explicit links as an *explicit issue network*. We represented issue networks in GraphML [6], expressing issue reports as nodes and links as directed edges. We visualized the result using the graph editor *yEd* [1]. Finally, we used *Gephi* (v. 0.8.1 Beta) [2] and *Social Networks Visualizer* [2] to compute structural statistics of the networks.

In the Safety IMS, extraction of explicit issue networks, i.e. the sub-graph discovery, was straightforward. We extracted explicit links from a field in the IMS called "Related cases". In the Android IMS on the other hand, no such field exists. Instead, we resorted to another link mining strategy. As presented in Section I, issue comments in the Android IMS contain references to other issue reports as part of the developer communication. We extracted this type of explicit links, expressed by HTML hyperlinks, using regular expressions. For both IMSs multiple links between the same pair of issues were treated as one link. Also, we stored self-links, i.e., issue reports with explicit links to themselves.

Our second analysis was a qualitative investigation of a sample of explicit issue networks extracted from the Android IMS to understand their meaning. This included manual study of issue reports and their related comments to evaluate relationships and the semantics of explicit links. As we were interested in linked structures rather than isolated issue reports with few or no links, we limited our study to these networks. The sample was selected to cover various network patterns. Then, we categorized explicit links as one of the four types:

1) **Related links**. Links created by developer to point out related, or possibly related, issue reports. Examples include expressions such as: "You might want to keep an eye on this issue as well: <link>" and "Please take a look also at this related discussion: <link>".
2) **Duplicate links**. Links created by developers to propose or claim that another issue report addresses the same issue.
3) **Clone links**. Links established by developers to highlight that a report with *identical* textual content in title and description exists. A clone link is a stronger type of duplicate link.
4) **Miscellaneous link**. Links created for other purposes, e.g., showing examples or discussing release planning, and links created by mistake. Examples include: "For your information, release X resolved: <link>", "a fix for <link> was not included in release X", and "On my phone, I cannot browse Google Code issues that contain a description longer than one page, for example <link>". Typical to the miscellaneous links is that they do not express a relation from the issue report the comment is attached to, but rather from the topic of the specific comment itself.

Finally, we studied the output from the link mining in the Android IMS compared to links established based on textual similarities, in line with what has been suggested for trace recovery based on Information Retrieval (IR) methods [9]. This analysis was conducted using standard techniques for text mining. We analyzed the similarity of the NL content in titles and descriptions using *RapidMiner* [22]. We calculated textual similarities based on the classic vector space model and cosine similarities [27]. Descriptions and titles of issue reports were represented as TF-IDF weighted terms, after removing stop words and applying Porter's stemmer.

IV. Results

A. Link mining (RQ1)

In both IMSs, we discovered large networks of issue reports. The 26,120 issue reports in Safety IMS contain 18,046 explicit links. Figure 3 shows an overview of the explicit networks extracted from the Safety IMS, limited to networks containing five or more issue reports, and Figure 4 shows a close-up of the largest network. About half of the issue reports are linked (48.0%), and 31.9% of the issue reports are included in issue networks of at least size 5. Further network measures are presented in Table I.

Thanks to the HTML structure of explicit links in comments in the Android IMS, we were able to mine all explicit links among issue reports using regular expressions. We extracted 3,449 unique explicit links among the 20,176 issue reports. A majority of the issue reports (16,655, 82.5%) neither link to other reports nor are the targets. Instead links tend to cluster, and 2,044 explicit links (59.7%) are present in explicit defect networks containing five or more reports. Figure 2 shows a visualization of the clusters containing five or more issue reports, in total 1,382 issue reports. More network measures are reported in Table I.

As further discussed in Sections VI and VII, network analysis has been successfully applied to object ranking [11]. Thus, we also evaluated whether it could be used to suggest issue priorities. However, we found no correlation between the number of links (neither in-links nor out-links) and the priority/severity assigned to an issue, neither in the Safety IMS nor in the Android IMS. On the contrary, we discovered

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

[2] SocNetV v. 0.81, http://socnetv.sourceforge.net/

Figure 2. Explicit issue networks in the Android IMS. The figure shows all networks containing five or more issue reports, in total 1,382 reports and 2,060 links. A to F are the sample link clusters we analyzed qualitatively.

Measure	Safety IMS	Android IMS
Nodes	26,120	20,176
Edges	18,046	3,449
Components	15,583	17,720
Density	2.6 x 10^{-5}	8.5 x 10^{-6}
Out-linked nodes	10,664	2,238
In-linked nodes	8,861	2,221
Reciprocal-linked nodes	4,646	701
Diameter	48	12
Avg. clustering coefficient	0.044	0.007
Average degree	0.691	0.171
Average path length	13.874	3.251

Table I
STRUCTURAL STATISTICS OF THE NETWORKS

both several highly prioritized issue reports with few explicit links, as well as non-prioritized issues with many explicit links. Figure 4 depicts a magnified region of the issue network, with one dominating issue report to the right, i.e., an issue report with a high number of inlinks.

B. Qualitative analysis (RQ2)

Since link mining only shows the presence of links, it must be combined with manual analysis to determine their semantics. Addressing RQ2, we present the results from a qualitative analysis of a sample of issue reports in the Android IMS. The sample, containing 111 reports and 225 links in 6 clusters (A-F), is shown in Figure 5. We refer to issue X as BX (short for BugX, with X representing the issue number). A close-up of an issue network is shown in Figure 3, and a summary of our findings is presented in Table I.

Network A (7 reports, 9 links) is dominated by duplicated reports. B3956 reports a presentation problem when browsing Google code issues using the Android web browser, and the author linked to B1725 just as an example. B3956 has then three cloned reports (B3957-B3959) and one duplicate (B4083). Eventually, in a B3956 comment, a Google engineer created links to all duplicates that were merged into this issue report.

Network B (6 reports, 8 links) contains issue reports related to volume issues with headphones. The network contains bidirectional connections between nodes, created when the same developer pointed out possible duplicates in the comments of B6708, B6709, and B8279. Apart from links to duplicates, we found one link to a “possibly related

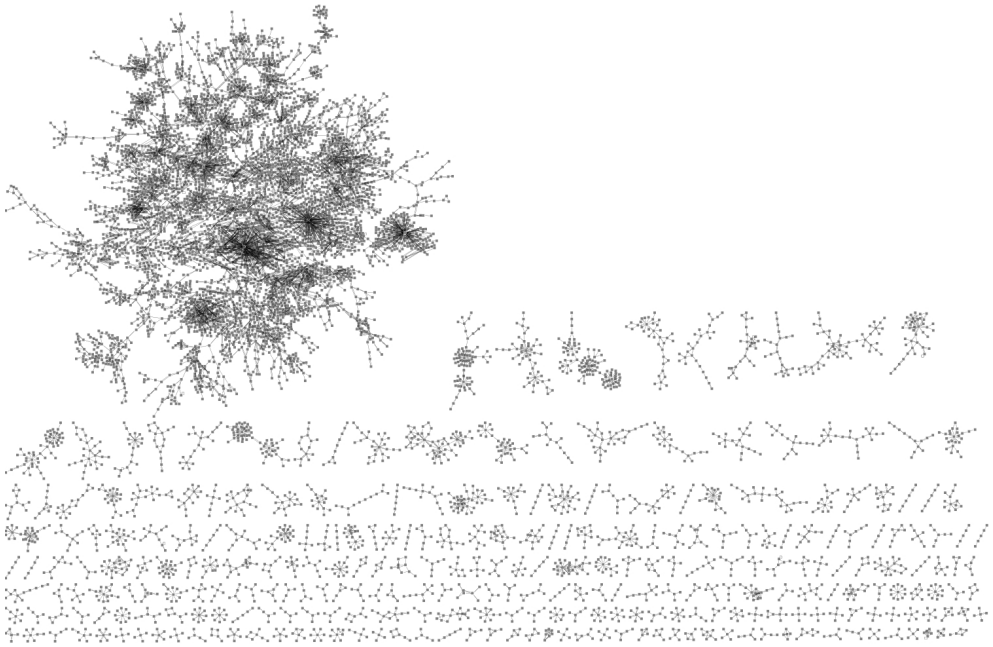

Figure 3. Explicit issue networks in the Safety IMS. The figure shows all networks containing five or more issue reports, in total 8,340 reports and 14,596 links.

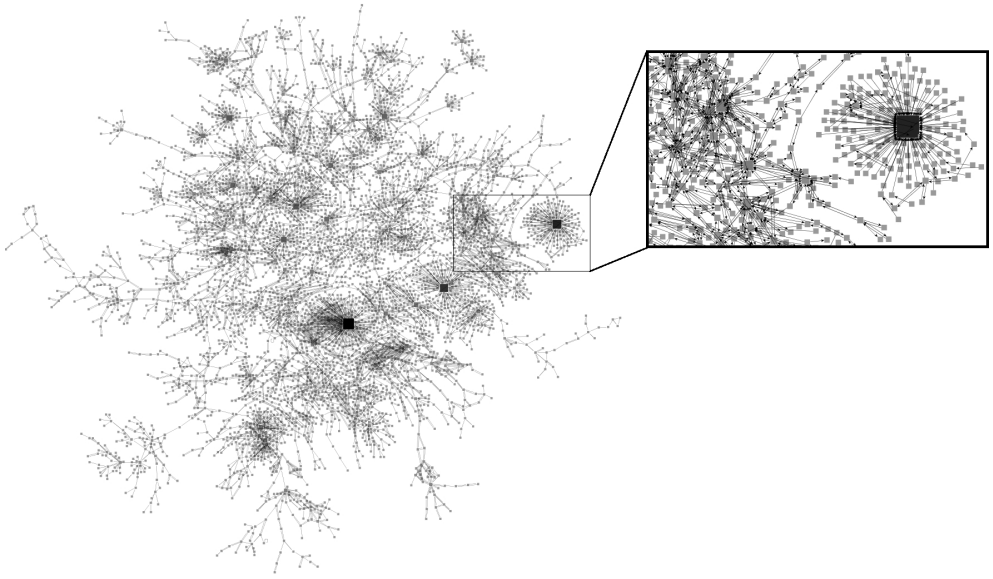

Figure 4. Close-up of the largest explicit issue network) in the Safety IMS, showing 5,414 reports and 10,403 links. The nodes with the highest degree centralities are represented by larger squares. A magnified region is shown to the right.

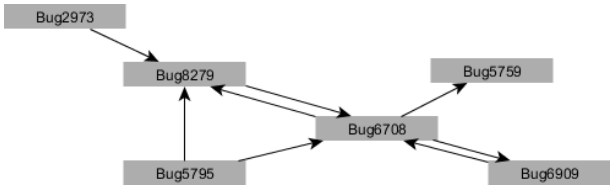

Figure 5. Explicit issue network B.

defect".

Most issue reports in Network C (24 reports, 28 links) address scrolling issues in the web-browser. Again, duplicates are the most common cause of links. However, we also identified referrals to other reports containing more information, possibly related issues, a link created to wrong issues (later corrected), and a link used in a replication step. Bidirectional connections were created in comments discussing possibly related issues.

Network D (43 reports, 108 links) has a central critical issue with 20 in-links and 14 out-links, reporting that SMSes intermittently are sent to the wrong contacts. This severe bug generated multiple defect reports, connected links expressing possible duplicates, related issue reports, and finally merged issues. Developers also created links in attempts to escalate issues. Moreover, two developers created dense subnetworks by copy-paste posting comments including links to possible duplicates in several issue reports.

Network E (23 reports, 22 links) only consists of 22 out-links from B12060, reporting that Google Talk does not display the correct name. This defect report has 22 clones, B12038-12059, merged into B12060 by a Google engineer.

Network F (8 reports, 51 links) has a dense link structure and includes 7 self-links. The issues all deal with synching of contacts. The dense link structure, and the self-links, were created when a developer posted the comment "Do we have one ground for many problems?" followed by links to 7 reports. This comment was copy-pasted to all seven reports.

C. Textual similarity analysis (RQ3)

We explored RQ3, also targeting the Android IMS, by first conducting a textual similarity analysis and then comparing the results from the textual similarity analysis with both the results from the link mining and from the qualitative analysis.

Again, we took the issue reports in networks A to F as our sample, containing 111 reports and 226 explicit links. 111 issue reports can have a maximum of $111 * 110 = 6105$ pairwise relations. Figure 6 shows the distribution of relative similarities for all 6,105 possible relations between issues. These relations we refer to as textually retrieved links, or just retrieved links (to distinguish them from the explicit links contained in the networks A to F).

Treating the explicit issue networks in the sample as undirected, we analyzed how many of the retrieved links, of

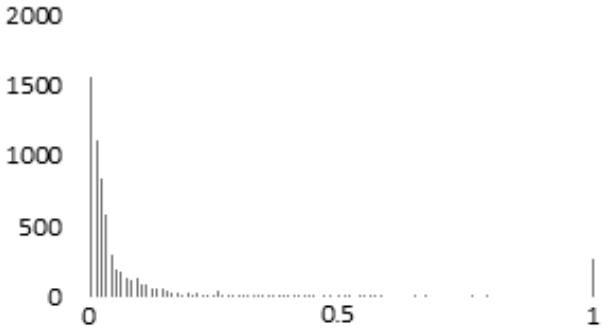

Figure 6. Distribution of cosine similarities between issue reports in the sample.

each type, were also explicit links. 267 of the retrieved links had similarity $= 1$, i.e., representing relations between issue reports of type 'clone'. By manual inspection, we found that all of the 226 explicit links of type 'clone' were included among the 267 retrieved links of the same type. The 41 retrieved links of type 'clone' not contained in the set of explicit links of the same type were redundant links, i.e., links that could be retrieved from the set of explicit links by applying the law of transitivity. For example, if there were explicit links of type 'clone' between issue reports A and B (indicating that B is a clone of A) and issue reports A and C (indicating that C is a clone of A), then it is clear that there must be also a relationship of type 'clone' between issue reports B and C, even if no explicit lank has been inserted by a developer. Thus, after adding all redundant links of type 'clone' in the networks of explicit links the total number of explicit and derived redundant links of type 'clone' equaled 267.

To further explore the nature of the retrieved links, we analyzed the first 74 non-clone links in the set of retrieved links, and compared them with the links in the set of explicit and redundant links. We chose 74 because it is a manageable number, and also it represents links in the set of retrieved links with reasonable cosine similarities, below 1 and above 0.35. A cosine similarity threshold of 0.35 corresponds to half the similarity proposed in previous work in linking software artifacts [21], [8], and is thus more inclusive. Table II presents our findings. A visualization of the findings is presented in Figure 7.

In Table III, we use the following abbreviations: 'Sim' means 'cosine similarity', 'T_Link' means 'textually retrieved link', 'E_Clone' means 'explicit clone link', 'E_Dupl' means 'explicit duplicate link', 'E_Rel' means 'explicit related link', 'E_Misc' means 'explicit miscellaneous link', 'T_Mean' means 'textually retrieved link that was not explicit but still carries a meaning', and 'T_False' means 'textually retrieved link that neither is explicit nor meaningful'. Retrieved links of type 'T_Mean' carry meanings corresponding to the kind of meanings that explicit links

			Distribution of link types					
	Nodes	Links	Clone	Dupl.	Rel.	Misc.	Network Character	Main cause of network
A	7	9	3	2	0	4	Midsize network, mixed link structure.	Cloned and duplicated reports.
B	6	8	0	6	1	1	Midsize network, some bidirectional connections.	Duplicated issue reports, reported in both ends.
C	24	28	0	15	10	3	Larger network, some bidirectional connections.	Duplicated and related reports.
D	43	108	5	55	36	12	Larger network, central critical issue with many in/outlinks.	Severe issue, duplicated reports. Copy/paste comments.
E	23	22	22	0	0	0	Larger network, central issue with many outlinks.	Cloned reports.
F	8	51	0	0	50	1	Midsize network, dense link structure, many self-links.	Copy/paste comments.
Sum	111	226	30	78	97	21		

Table II
SUMMARY OF THE EXPLICIT ISSUE NETWORK SAMPLE.

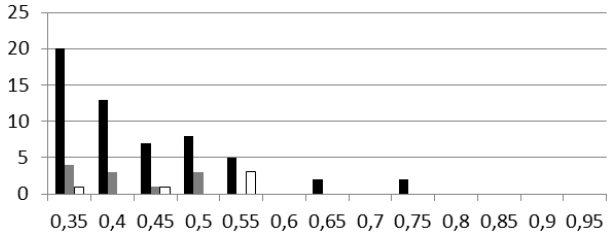

Figure 7. Meaning of the first 74 retrieved non-clone links. The X-axis shows cosine similarities between 0.35 and 0.95. The colors of the bars have the following meanings: black=explicit, gray=meaningful but not explicit, white=false positive.

Sim	T_Link	E_Clone	E_Dupl	E_Rel	E_Misc	T_Mean	T_False
0.35	26	0	13	7	0	4	2
0.4	16	0	7	6	0	3	0
0.45	9	0	3	4	0	1	1
0.5	11	0	5	3	0	3	0
0.55	8	0	4	1	0	0	3
0.6	0	0	0	0	0	0	0
0.65	2	0	2	0	0	0	0
0.7	0	0	0	0	0	0	0
0.75	2	0	2	0	0	0	0
0.8	0	0	0	0	0	0	0
0.85	0	0	0	0	0	0	0
0.9	0	0	0	0	0	0	0
0.95	0	0	0	0	0	0	0
1	267	267	0	0	0	0	0

Table III
NUMBER OF RETRIEVED LINKS WITH SIMILARITY 0.35 TO 1.

of types 'E_Dupl' and 'E_Rel' have. The meanings of the various types of explicit links were described in Section III.

Our results show that the 74 first retrieved non-clone links add more useful information than is available in only the explicit links. Of the analyzed retrieved links 57 links were also explicit (shown in black color in Figure 7), either of type 'duplicate' (36 links) or 'related' (21 links). The remaining 17 retrieved links did not match any of the explicit links (or redundant links). Of these 17 retrieved links we found that 11 (65%, shown in gray color in Figure 7) provide meaningful information and could just as well have been explicit, either as links of the duplicate or related type. On the other hand, 6 retrieved links (shown in white color in Figure 7) were false positives, i.e., links that do not add meaningful information and thus should be considered noise.

V. THREATS TO VALIDITY

This section discusses limitations of this initial study, as well as related threats to validity. Regarding the link mining, our sample of >25,000 safety issue reports and >20,000 Android issue reports can be considered large. However, since an important part of this study was conducted using qualitative analysis, we also selected a smaller sample of 111 issue reports from the Android IMS for thorough manual analysis. We discuss threats to internal validity, reliability and external validity [26] regarding the analyses conducted on the qualitative sample of 111 issue reports.

Internal validity concerns confounding factors that can affect the causal relationship between the treatment and the outcome. In this study, the selection bias is a threat. As we did no quantitative analysis, the issue networks were not randomly selected, but instead selected intentionally as suggested in the literature [26]. In qualitative research the units of analysis should be selected to be 'typical', 'critical', 'revelatory' or 'unique' in some respect [3]. In our study, we first identified all networks of explicitly linked issue reports in the total set of issue reports. Then we selected six networks (link clusters), labeled A to F, with specific properties (cf. Table I). Thus, our selection criterion was the uniqueness of the assumed root cause for the observed structure within a network (link cluster).

Reliability is concerned with whether other researchers would come to the same conclusions given the same data. The interpretation of data is the core of qualitative research [10]. To prepare for this threat we presented the classification scheme we used for explicit links, complemented by several examples from the Android comments in Sec-

tion III. However, we did not test whether other researchers agree with our classification. Interpretation was also required to classify non-explicit retrieved links as meaningful or not, and again we describe how we classified retrieved links but we did not test whether other researchers agree with our classification decisions.

Threats to *external validity* are of concern when generalizing findings. External validity is a major threat in this study, as it is unclear whether IMSs in other software engineering projects contain similar types of explicit and derived links and similar structures in networks of explicit links, as we detected in the the Safety IMS and the Android IMS. All we know at this point is that other IMSs, of both OSS and proprietary software development, typically contain explicit links between issue reports. For example, in IMSs of commercial systems such as Merant Tracker and Microsoft Team Foundation Server there are dedicated fields in issue reports for developers to signal relations to other issues. However, regarding the nature of both link-structures and the types of links (both explicit and retrieved) in networks of issue reports, more research is required. Also, in order to understand better to what extent findings derived from research on easier accessible OOS-related IMSs can be transferred to IMSs of company-proprietary development projects, a comparison between these two classes of IMSs needs to be conducted, similar to what has been done in research within source code repositories [24].

VI. Related Work

As many datasets today contain linked collections of interrelated data, link mining has become a popular data mining approach, combining research on link analysis, hypertext mining and graph mining [11]. A primary focus of the link analysis has been link-based object ranking, i.e., exploiting the link structure of a graph to order or prioritize elements within the graph. Link-based object ranking has been made particularly visible in web-retrieval, where the PageRank [23] and HITS [19] algorithms have proven successful. In software engineering, Kagdi *et al.* proposed using mining to recover traceability links between artifacts in a software repository [15]. However, their approach is different from ours as they mine the version history for co-changing artifacts, while we mine explicit hyperlinks.

Several researchers have proposed to recover trace links between software artifacts by applying IR techniques. The conjecture is that if two artifacts have a high textual similarity, they are likely to refer to the same concepts [9]. Antoniol *et al.* did pioneering work when they recovered traceability links between source code and related documentation, both using the Vector Space Model (VSM) and probabilistic retrieval techniques [1]. Later, Marcus and Maletic introduced Latent Semantic Indexing (LSI), another IR technique, to recover the same type of links [21]. They showed that they could get at least as good result using LSI as VSM, but without the need for stemming.

A specific type of relation between software artifacts is occurrence of duplicates, known to be a problem in large IMSs. Several researchers have addressed the challenge of detecting duplicated issue reports. Runeson *et al.* proposed using VSM to detect duplicates in an IMS at Sony Ericsson [25]. They demonstrated that about 2/3 of the duplicates could be found using textual similarity analysis based on VSM. Targeting the same challenge, Kaushik and Tahvildari compared different IR techniques [17]. They conducted their evaluations on IMSs from the OSS Eclipse and Firefox, and reported that 60% respectively 58% of the duplicated issue reports were found. Tian *et al.* instead expressed the issue duplicate detection as a classification problem [31] (i.e., duplicate/not duplicate). They used machine learning, and combined textual similarity with categorical features, e.g., component, priority and version. Based on an evaluation on issue reports from the Mozilla IMS, another OSS, they presented promising results, further confirming that textual similarity may indicate relationship between issues.

VII. Discussion and Future Work

In both the Safety IMS and the Android IMS, link mining discovered networked structures containing between 2 and 5,414 issue reports (RQ1). The results show that mining the explicit links created by engineers in IMSs is a feasible approach to extract relations among issue reports. We also found that there are larger explicit issue networks hidden in the Safety IMS, a database with a separate field for 'related issues', than in the Android IMS. Moreover, the Safety IMS contains a total issue network that is bigger, denser, and more connected with regard to the clustering coefficient.

The presence of networks in IMSs opens for further research on navigation of issues reports. In line with research on hyperlinked information in general [19], and web search in particular [23], networks of software artifacts have previously been used to significantly improve searching based on textual similarity in software engineering. Karabatis *et al.* [16] showed how semantic networks of software artifacts could be used to find additional related artifacts during search, i.e., improving recall. Also, they report on how it could be combined with contextual information about the searching engineer to filter the search results, i.e., improving precision. Furthermore, networks of software artifacts could be used as input to visualization techniques supporting engineers' information seeking, as proposed by Cleland-Huang and Habrat [7]. Visualization of large amounts of data can enable humans to conduct visual data mining [18], combining the computational power of computers with human knowledge and creativity.

While our link mining approach has the potential to discover initial networks of software artifacts, more is required to establish semantic networks [30]. RQ2 addressed the

semantics of explicit links between issue reports created in comments in the Android IMS. Our qualitative analysis indicates that explicit links, in networks of at least five issue reports, repeatedly are created to signal duplicated or related reports. Moreover, we found that developers signal duplicates with various degrees of certainty, from the weakly stated "possibly duplicated issues", to probable duplicates and definite clones. We also found that Android developers created several links of type 'duplicate' in batches when merging several issue reports into a single issue report.

On the other hand, we noticed that not all explicit links express meaningful relations between issue reports. For example, we found invalid links, i.e., links that clearly were not representing what the developer who introduced the link claimed, and links created during internal communication (e.g., moderating), e.g., links that are intended to escalate issues (similar to thread 'bumping' in WWW fora), and links that contain complaints from developers about some Android functionality. Still, a majority of the explicit links seems to represent valuable information about relations between issue reports provided by developers.

While explicit issue networks in the Android IMS normally appear to grow to sizes above five issue reports due to input from several developers working collaboratively, we also identified issue networks created by single developers. Single developers can generate large networks of explicit links by copy-pasting links in comments on multiple issues, or by submitting several identical reports, i.e., clones of issue reports. Consequently, developers can single-handedly cause relatively large networks.

Regarding RQ3, we found that the relations detected by textual similarity analysis seem to efficiently identify all available issue report clones. Moreover, our results confirm that a high textual similarity could be used as a duplicate warning, in line with previous findings [12], [15], [25]. Furthermore, the results from the textual similarity analysis also discovered meaningful relations between issue reports which were not contained in the set of explicit links.

Unfortunately, however, if compared to the occurrence frequency of invalid links in explicit link networks, textual similarity analysis seems to indicate more often relations between issue reports that turn out to be meaningless. This indicates that explicit links are generally of higher quality than retrieved links. Also, relations between issue reports expressed by explicit links are easier to extract than implicit relations discovered from mining sparse NL descriptions through textual similarity analysis.

We now discuss the answers to the three RQs in relation to the support provided for the impact analysis work task (i.e., possibilities A and B), as described in Section I. Based on our results of the comparison between retrieved links and explicit links in the selected set of 111 issue reports contained in network clusters A to F, it seems that the Android developers have a good knowledge about the existence of clones among issue reports, as textual similarity analysis did not detect relationships of the type 'clone' between issue reports that were not already contained in the set of explicit (and redundant) links. This seems to suggest that Android developers don't need automated textual similarity analysis in order to detect clones of issue reports.

Our analysis of both explicit links and retrieved links (not related to clones) in the Android IMS did not provide conclusive results with regards to helping developers finding traces to other development artifacts via the detection of relations between issue reports of the types 'duplicate' and 'related'. Since Android issue reports do not have any explicit links to other development artifacts, we focused in our analyses on the number of explicit and retrieved links that pointed to duplicated or related issue reports. In addition, we investigated whether automated textual similarity analysis has the potential to find duplicated and related issue reports not yet represented by explicit links.

Overall, we found that the total number of explicit links of type 'clone' is small compared to the total number of explicit links of types duplicate' and 'related'. We do not know whether this is typical for OSS development projects or for software development projects in general. In any case, our results seem to support that automated textual analysis has the potential to complement the knowledge of developers about duplicated and related issue reports. At least this was the case for the Android IMS. One problem associated with automated textual similarity analysis, well studied in the context of IR-based trace recovery [9], is that the degree of similarity between issue reports (below 1) does not seem to be a good predictor for meaningful relations between issue reports. 6 out of the 74 retrieved links with similarity degrees below 1 and above 0.35 (i.e., 8%), were false positives, i.e., were not meaningful.

Finally, in order to tackle our long-term goal of helping developers to find traces to artifacts other than issue reports, we plan to conduct further analyses within safety-critical development. While this paper is a first step toward extracting semantic networks of software artifacts, our next goal is to merge information from additional sources. Currently, we are working on adding links mined from a database of impact analysis reports, containing rich information about issue-related change locations in source code and impact on other artifacts such as test cases, design documents, and requirements documents. The resulting network, representing relations of different types among various artifacts, could be used both to improve IR in a software engineering context and constitute an initial knowledge representation in recommendation systems for software evolution.

REFERENCES

[1] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. In *Transactions on Software Engineering*, volume 28, pages 970–983, 2002.

[2] M. Bastian, S. Heymann, and M. Advani. *Gephi: An Open Source Software for Exploring and Manipulating Networks*. 2009.

[3] I. Benbasat, D. Goldstein, and M. Mead. The case research strategy in studies of information systems. *MIS Quarterly*, 11(3):369, 1987.

[4] N. Bettenburg, R. Premraj, T. Zimmermann, and K. Sunghun. Duplicate bug reports considered harmful... really? In *Proceedings of the International Conference on Software Maintenance*, pages 337–345, October 2008.

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

[6] 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.

[7] J. Cleland-Huang and R. Habrat. Visual support in automated tracing. In *Proceedings of the 2nd International Workshop on Requirements Engineering Visualization*, page 4, 2007.

[8] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. *Transactions on Software Engineering and Methodology*, 16(4), 2007.

[9] 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.

[10] U. Flick. *An Introduction to Qualitative Research*. SAGE, 2009.

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

[12] J. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *Transactions on Software Engineering*, 29(6):481–494, 2003.

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

[14] J. Johnson and P. Dubois. Issue tracking. *Computing in Science Engineering*, 5(6):71–77, 2003.

[15] H. Kagdi, J. Maletic, and B. Sharif. Mining software repositories for traceability links. In *Proceedings of the International Conference on Program Comprehension*, pages 145–154, 2007.

[16] 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 S. Spaccapietra and L. Delcambre, editors, *Journal on Data Semantics XIV*, pages 74–104. Springer, Berlin, 2009.

[17] N. Kaushik and L. Tahvildari. A comparative study of the performance of IR models on duplicate bug detection. In *Proceedings of the 16th European Conference on Software Maintenance and Reengineering*, pages 159–168, 2012.

[18] D. Keim. Information visualization and visual data mining. *Transactions on Visualization and Computer Graphics*, 8(1):1–8, 2002.

[19] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[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] A. Marcus and J. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proceedings of the International Conference on Software Engineering*, pages 125–135, 2003.

[22] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. YALE: rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge discovery and data mining*, pages 935–940, 2006.

[23] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report SIDL-WP-1999-0120, Stanford InfoLab, 1999.

[24] B. Robinson and P. Francis. Improving industrial adoption of software engineering research: a comparison of open and closed source software. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, pages 21:1–21:10, 2010.

[25] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of duplicate defect reports using natural language processing. In *Proceedings of the 29th International Conference on Software Engineering*, pages 499–510, 2007.

[26] P. Runeson, M. Höst, A. Rainer, and B. Regnell. *Case Study Research in Software Engineering. Guidelines and Examples*. Wiley, 2012.

[27] G. Salton, A. Wong, and C. Yang. A vector space model for automatic indexing. *Commununications of the ACM*, 18(11):613–620, 1975.

[28] W. Scacchi. Understanding the requirements for developing open source software systems. *IEEE Software*, 149(1):24–39, 2002.

[29] E. Shihab, Y. Kamei, and P. Bhattacharya. Mining challenge 2012: The android platform. In *Proceedings of the 9th Working Conference on Mining Software Repositories*, pages 112–115, 2012.

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

[31] Y. Tian, C. Sun, and D. Lo. Improved duplicate bug report identification. In *Proceedings of the 16th European Conference on Software Maintenance and Reengineering*, pages 385–390, 2012.