Open access • Proceedings Article • DOI:10.1145/1806799.1806817

Software traceability with topic modeling — Source link

Hazeline U. Asuncion, Arthur U. Asuncion, Richard N. Taylor

Institutions: University of California, Irvine

Published on: 01 May 2010 - International Conference on Software Engineering

Topics: Requirements traceability, Reverse semantic traceability, Software verification and validation, Goal-Driven Software Development Process and Traceability

Software Traceability with Topic Modeling

Hazeline U. Asuncion
Institute for Software Research
University of California, Irvine
hasuncio@ics.uci.edu

Arthur U. Asuncion
Center for Machine Learning and Intelligent Systems
University of California, Irvine
asuncion@ics.uci.edu

Richard N. Taylor
Institute for Software Research
University of California, Irvine
taylor@ics.uci.edu

ABSTRACT

Software traceability is a fundamentally important task in software engineering. The need for automated traceability increases as projects become more complex and as the number of artifacts increases. We propose an automated technique that combines traceability with a machine learning technique known as topic modeling. Our approach automatically records traceability links during the software development process and learns a probabilistic topic model over artifacts. The learned model allows for the semantic categorization of artifacts and the topical visualization of the software system. To test our approach, we have implemented several tools: an artifact search tool combining keyword-based search and topic modeling, a recording tool that performs prospective traceability, and a visualization tool that allows one to navigate the software architecture and view semantic topics associated with relevant artifacts and architectural components. We apply our approach to several data sets and discuss how topic modeling enhances software traceability, and vice versa.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Documentation*; D.2.9 [**Software Engineering**]: Management; G.3 [**Mathematics of Computing**]: Probability and Statistics—*Probabilistic algorithms*; H.3.3 [**Information Systems**]: Information Storage and Retrieval—*Information Search and Retrieval*

General Terms

Documentation, Management

Keywords

Software Traceability, Topic Model, Latent Dirichlet Allocation, Software Architecture

ICSE '10, May 2-8 2010, Cape Town, South Africa

1. INTRODUCTION

Large-scale industrial projects are often comprised of thousands of software development artifacts, such as requirements documents, design documents, code, bug reports, and test cases. The goal of software traceability is to discover relationships between these artifacts to facilitate the efficient retrieval of relevant information, which is necessary for many software engineering tasks [4]. While fully manual traceability approaches are typically only feasible for small projects, automated techniques also encounter difficulties in practice due to the lack of accuracy and the high level of false positive trace links generated [15, 24]. In this paper, our main contribution is a scalable approach to traceability based on a combination of automated link capture and topic modeling.

The traceability problem can be tackled both retrospectively and prospectively. In retrospective traceability, artifact relationships are inferred *ex post facto* from a static set of artifacts. Automated approaches such as information retrieval techniques generally perform traceability retrospectively. In contrast, prospective traceability generates trace links *in situ*, as artifacts are being created and modified during the development process. Prospective capture is the creation of putative links between artifacts based upon directly observed actions of the user on those artifacts. While we discuss the application of topic modeling in the retrospective setting, we mainly focus on the combination of prospective traceability and topic modeling in this paper.

There are several key benefits of the prospective approach. Prospective traceability allows for links to be captured in an online fashion, incrementally improving the system's traceability and allowing developers to immediately benefit from the trace links created [9]. Another benefit of prospective traceability is the presence of additional captured information, such as the temporal sequence of user actions in the design environment, which can be used to automatically generate trace links to heterogeneous artifacts.

Machine learning and information retrieval techniques have been used in the past to automatically generate traceability links [23, 25]. In particular, Latent Semantic Indexing (LSI) [20] has made significant inroads into the area of automated traceability [27, 41]. More recently, probabilistic topic models such as Latent Dirichlet allocation (LDA) [12] have been used for source code analysis [30, 31, 46]. LDA is an unsupervised machine learning technique that facilitates the automatic learning of semantic topics from the set of artifacts without requiring previous training data with training labels. Usually, these approaches have exclusively focused on retrospective traceability. In contrast, we propose

incorporating the LDA model into the task of prospective traceability. Another difference from previous LDA-based work is that we do not focus solely on source code analysis but rather investigate an architecture-centric approach [44] which creates trace links for all types of artifacts produced in the software development life cycle.

We argue that a positive symbiosis exists between prospective capture and topic modeling. Prospective link capture helps to identify a larger amount of potentially related artifacts (through observation of user actions), which gives rise to a more diverse and semantically-coherent set of "learned" topics. Meanwhile, these learned topics enhance the process of prospective capture through the visualization of the topical content of the software architecture and the efficient retrieval of semantically-similar artifacts. Furthermore, while topic modeling cannot be applied to certain types of artifacts (like sound or video files), prospective link capture can still trace to these artifacts by taking advantage of recorded temporal user actions. Conversely, prospective link capture is inadequate at determining the semantic nature of artifacts, a task at which topic modeling excels. Thus, these two techniques tightly complement each other. Note that existing retrospective machine learning techniques can also complement our prospective approach.

In the next section, we discuss the current state of software traceability research and discuss strategies for effective traceability. We then provide a brief overview of topic modeling techniques. We detail our approach of combining prospective traceability with topic modeling. Then we describe several tools that demonstrate the feasibility of our approach and show results on several data sets. We then conclude with future research directions.

2. SOFTWARE TRACEABILITY

The practical realization of software traceability is known to incur high overhead, and thus there has been a concerted effort to automate the generation of trace links. We highlight the current state of the art in both retrospective and prospective traceability, and we also discuss insights into effective traceability which guide our approach.

2.1 Retrospective traceability

Most automated traceability techniques fall under the category of retrospective traceability. Information retrieval techniques such as the Vector Space Model and Latent Semantic Indexing have been used to automatically generate candidate links based on textual similarity between artifacts [15, 19, 25, 27]. Other machine learning techniques have also been combined with program analysis and run-time monitoring to automatically link source code to use cases [23].

Probabilistic topic models like Latent Dirichlet Allocation have been used to mine semantic topics from source code. In particular, LDA has been used to automatically categorize code [46], and LDA topics have been interpreted to be analogous to cross-cutting concerns in aspect-oriented programming [10]. We discuss LDA in depth in Section 3.

Visualization techniques have been proposed to support the efficient identification of correct links. Duan and Cleland-Huang use cluster-based techniques to group candidate trace results that are presented to the user [22]. Other visualization techniques include tag clouds to graphically display term frequencies and a tree structure to represent the hierarchical structure in a requirements document [14]. Visualization is an important aspect of traceability as it allows users to verify the quality of trace links. As we will see later in the paper, topic modeling provides useful semantic information that can be used to visualize the semantic relationships between the traced artifacts and software architecture.

2.2 Prospective traceability

Prospective traceability approaches have largely been manual or minimally-automated until recently. Such techniques include embedding related artifact IDs into source code [34] and only automating peripheral traceability tasks, such as maintaining consistency between artifacts [8]. Tools like PRO-ART and TOORS support the online recording of links based on pre-specified user information. PRO-ART requires a process model to be manually specified prior to the recording of traces between artifacts [39]. TOORS requires training in formal specifications in order to mathematically express the relations between classes of artifacts [38]. Once these relations have been specified in the database, automated tracing between artifacts is possible.

Other research communities have also developed methods for prospectively capturing relations between artifacts. In the aspect-oriented research community, code is automatically related to concerns by requiring users to specify a concern prior to making code changes [37]. Model driven approaches use transformation techniques to automatically create links between source and target models [28, 32]. These approaches may require the specification of the meta-model and the transformation program or the specification of the transformation rule prior to the link capture. More recently, lightweight approaches to tracing between artifacts have been proposed in the program comprehension research community. Online recording of user interaction has been used to associate source code to tasks [29] as well as source code to team collaboration [1]. Prospectively capturing links between artifacts during an experiment lifecycle has also been used extensively in e-Science [3].

In previous work [9], we investigated an approach for automated prospective capture that utilizes open hypermedia techniques [5] and rules. Tool-specific adapters record user actions over heterogeneous platforms while users generate or access artifacts during software development. Rules are used to associate links with a relationship type. Developing these adapters and custom rules requires only a one-time overhead setup. The overhead of using this approach involves turning the record mode on or off, optionally applying rules for each recording session, and deleting extraneously recorded links which were not caught by the filters. The approach, however, suffers from the fact that prospective capture lacks insight into the semantic nature of the artifacts being linked. For instance, if a developer is multi-tasking and working on two different projects at once, prospective capture based on the recorded temporal sequence of actions may generate irrelevant trace links. Semantically sifting through artifacts is important for this task, and thus, we explore the use of topic modeling in prospective traceability.

2.3 Insights for effective traceability

Based on our experience with implementing a successful software traceability framework in an industrial setting [8] and our observations of traceability techniques within e-Science [9], our approach is guided by the following general insights.

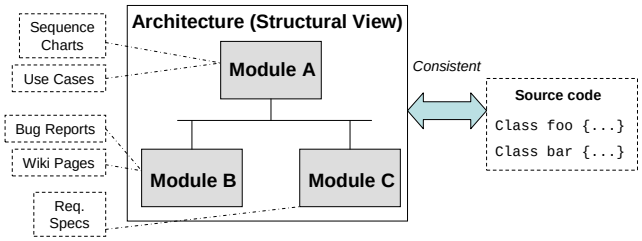

Figure 1: Architecture-centric traceability, where software artifacts are linked to specific modules.

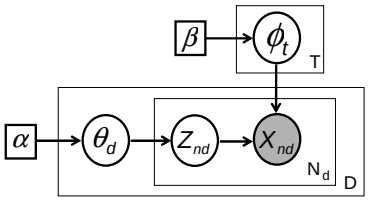

Figure 2: Graphical model for Latent Dirichlet Allocation. Unshaded/shaded circles denote hidden/observed variables, boxes denote parameters, and plates denote replication over indices.

Understand the limits of automation. As we have observed in previous work [8], it is important to distinguish between automatable traceability tasks and tasks that require human intervention. Thus, we aim to use automation techniques that minimize the manual tasks of artifact search and trace link post-analysis. As we detail later in the paper, our technique generates links prospectively as users perform development tasks. The generated links are then automatically categorized by semantic topics and visually represented to the user to facilitate link analysis. In addition, users can navigate to the artifacts in the context of their native editors to aid users in determining the correctness of the links.

Traceability must support user work practices. Both our previous work [8] and literature [21] suggest that performing traceability tasks should be a side effect to the users' development tasks in order to gain wide adoption. In addition, the traced information should provide direct benefits to the users. Our prospective capture approach seeks to capture links in the background while users perform their development tasks. The online improvements to the system's traceability and the visual representation of trace links can provide immediate benefits to users.

The traceability approach must be scalable. The literature reports that one of the difficulties with transferring traceability techniques to industry is that the examples are too small [43]. Thus, it is important that approaches for both capturing links and presenting the linked information to users be scalable. In the prospective setting, scalability is achievable due to the incremental capture and maintenance of trace links. Furthermore, advances in probabilistic inference algorithms have made topic modeling very scalable, as we will show later in the paper.

Traceability must be bounded. Literature suggests that it is infeasible to create links to every single artifact. With n artifacts, there are $\binom{n}{2}$ possible pairwise links between these artifacts, making it generally infeasible to evaluate and manage each of these potential links. The "trace for a purpose" strategy states that trace links should only be captured if there is a direct usage of the traced information [15]. In addition, creating direct links between line-level source code and high-level concepts can produce an unmanageable amount of links that make it difficult to analyze and use the links. We posit that centering links at the system's architecture level bounds the scope of traceability links to be captured. Figure 1 depicts architecture-centric [44] traceability. Artifacts are linked to particular architectural components, and there is the assumption that source code is generally consistent with the architectural description. Many design factors such as domain constraints, user requirements, and governmental regulations are addressed at the architectural level. Since architecture forms the nexus between artifacts in the problem space (e.g. requirements) and artifacts in the solution space (e.g. code, tests), the conceptual gap between the architecture and the other artifacts is smaller than, for example, the conceptual gap from source code to requirements. Finally, the fact that architecture is a high-level abstraction of the source code supports impact analysis, software evolution, and system comprehension, and these are tasks that software traceability aims to support as well.

3. TOPIC MODELING

Topic modeling is a widely-used machine learning technique for automatically inferring semantic topics from a text corpus. In order to provide a foundation for our topic-enhanced traceability approach, we briefly introduce Latent Dirichlet Allocation and discuss applications and limitations of topic modeling.

3.1 Latent Dirichlet Allocation

Automatically discovering the underlying structure of the data is an important problem in machine learning which has spurred the development of dimensionality reduction techniques. Latent Semantic Indexing (LSI), also known in the literature as Latent Semantic Analysis [20], is one such technique that attempts to address this problem. LSI has been utilized extensively in software engineering, since it can be easily applied to text corpora. In LSI, each document in the corpus is represented as a word count vector of length W, where W is the number of words in the corpus vocabulary. When the vectors of all D documents are placed side by side, one obtains a $W \times D$ matrix of counts, which LSI decomposes by singular value decomposition in order to map the documents to a lower dimensional latent space. Hofmann proposed Probabilistic Latent Semantic Indexing (PLSI), a probabilistic version of LSI which is able to achieve better results [26]. Improving upon PLSI, a fully generative Bayesian model known as Latent Dirichlet Allocation (LDA) was introduced by Blei et al. [12]; LDA deals with overfitting issues associated with PLSI and can achieve better results than PLSI. In this paper, the topic model that we use is LDA.

The graphical model of LDA is displayed in Figure 2. Each topic t is defined to be a probability distribution ϕ_t (over W words) drawn from a Dirichlet distribution with parameter β. Furthermore, each document d is associated with a probability distribution θ_d (over T topics) drawn from a Dirichlet with parameter α. Note that a sample obtained from a Dirichlet distribution is precisely a discrete distribution itself. A topic assignment variable z_{nd} is associated to each individual word token n in each document d and is sampled

from θ_d. The actual word token x_{nd} is sampled from its respective topic ϕ_t where $t = z_{nd}$. Since these word tokens are observed data (which can be represented as a $W \times D$ matrix of counts), we can use Bayesian probability calculus to invert the generative model given observed data and automatically learn the hidden variables ϕ_t for each topic t, and θ_d for each document d. It is important to note that LDA is an *unsupervised* machine learning framework, which means that no previous training data with training labels is required. The only required input to LDA is the set of documents (converted to a sparse $W \times D$ matrix after removing stopwords and performing stemming) and the desired number of topics T to be learned.

Recent advances in Bayesian inference for topic models have made it possible to learn an LDA model in near real-time on a moderately-sized set of documents. We use an efficient zeroth-order collapsed variational Bayesian inference algorithm (CVB0), in which an underlying variational distribution $q(z_{nd})$ over T topics is associated with each topic assignment z_{nd}. The algorithm consists of iteratively performing the following variational updates in a systematic scan over tokens,

$$q(z_{nd} = t) \propto \frac{N_{wt}^{\neg nd} + \beta}{N_{t}^{\neg nd} + W\beta} \left(N_{td}^{\neg nd} + \alpha \right)$$

where $N_{wk}^{\neg nd}$, $N_{k}^{\neg nd}$, and $N_{kd}^{\neg nd}$ are expected counts derived from $q(\mathbf{z})$. For more details, see Asuncion et al. [7].

While other algorithms such as fast collapsed Gibbs sampling can also be used [40], CVB0 is very fast – later in the paper we will show timing results when applying CVB0 to software artifacts. Furthermore, topic modeling can easily scale to hundreds of thousands of artifacts, especially when combined with distributed computing [6, 35]. We take advantage of the efficiency of CVB0 inference in our traceability tools.

3.2 Applications of LDA

Once an LDA model is learned on a corpus, one can use the learned probability distribution over words ϕ_t to display a list of W words, sorted by decreasing probability, for each topic t. For instance, if topic 1 has high-probability words "`school teach student book`", one can assume this topic to be about academics. Thus, the semantic content of the entire corpus can be summarized by displaying these topics[1].

One can also use document d's distribution θ_d over topics to determine the topical content of the document. For instance, if there are 4 topics and if $\theta_d = [0.5, 0.1, 0.05, 0.35]$, then one can infer that document d is mainly comprised of a combination of topics 1 and 4. Note that θ_d can also be compared against other θ'_d (using a similarity measure such as Kullback-Leibler divergence or cosine distance) in order to obtain a ranked list of topically-similar documents. Thus, LDA is useful for finding related documents as well as visualizing the topical content of each document. We incorporate these abilities into our traceability tools.

LDA has been applied to a variety of problems, including information retrieval [47] and entity resolution [36]. As mentioned earlier, LDA has also been used to find topics in source code [30, 31]. The main differences between our approach and previous LDA-based work are (1) we mainly focus on prospective traceability (2) we do not perform topic modeling on code, but rather on text-based artifacts (such as requirements/design documents) generated during the software lifecycle. Thus, our approach is complementary with other LDA-based work on source code analysis.

[1]For a brief demo of topic modeling on news articles, see: http://asuncion.ics.uci.edu/demo

3.3 Limitations of LDA

While LDA is very useful for analyzing text data, it is important to note the limitations of LDA. The first limitation is that the number of topics, T, needs to be pre-defined by the user. If T is small, then the topics are more general in nature and are more distinguishable from each other. If T is large, then nuanced topics may appear and topics may begin to overlap semantically. One way to address this problem is to learn multiple models using various T and visually inspect the topics. Alternatively, a non-parametric model known as Hierarchical Dirichlet Processes [45] extends LDA and seeks to learn the optimal T automatically.

Another limitation is that the visualization of LDA topics is limited to displaying the high-probability words and so the interpretation of the actual semantic nature of the topic is left to the user (e.g., for the topic "`school teach student book`", there is no automatic label denoting that this is an "academic" topic). Note that there has been some recent work that seeks to provide topic labels automatically [33].

4. OUR COMBINED APPROACH

We have reviewed the problem of traceability and the benefits that topic modeling brings. We believe that traceability can benefit from the application of topic modeling to software development artifacts. We outline how topic modeling can be effectively applied to prospective traceability.

Topic modeling enhances prospective capture by providing semantic information about the artifacts. Recall that our prospective traceability approach is centered on the architecture [44]. While a developer is working on a particular architectural component, our technique captures the developer's actions, such as opening a requirements specification, visiting a Wiki page, or modifying a bug report. The artifacts that the developer visits are then automatically linked to the architectural component on which the developer is working. Once these artifact links are recorded, the list of artifacts related to a component can be visualized by the developer by simply navigating to the particular component in the architectural graph. Topic modeling enhances the prospective capture by providing a learned set of topics that can help the developer to find artifacts or other relevant online documentation. For instance, if the developer needs to search through the project's entire set of artifacts, the developer can use the learned semantic topics to filter the artifact search. Furthermore, the developer can easily find similar artifacts by comparing θ_d and finding the closest matches.

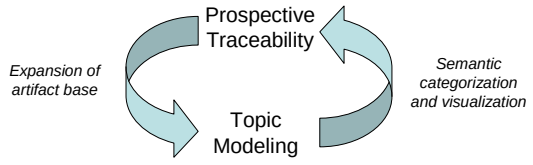

Figure 3: Symbiosis between prospective traceability and topic modeling.

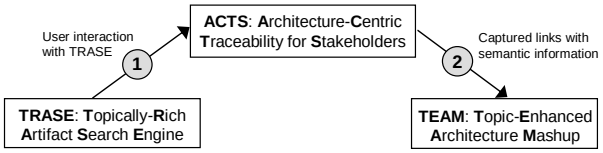

Figure 4: Tool support for our combined approach.

Topic modeling can also enhance the visualization of the system. Artifacts that have been gathered via prospective capture are now connected to the corresponding architectural component. When the developer views the component, the developer can view the artifacts along with a graphical depiction of the topic distributions (θ_d). Furthermore, the components themselves are now associated with the topics of their artifacts. We can define each component c's topic distribution Θ_c to be an average of the topic distributions of artifacts associated with the component: $\Theta_c = \frac{1}{N_{d \in c}} \sum_{d \in c} \theta_d$. Thus, we have a topic-enhanced bird's eye view of the entire system, and one can find which artifacts and components are related to particular semantic topics. This feature is important for system comprehension.

Not only is prospective capture enhanced by topic modeling, but the quality of the topics is enhanced by the prospective capture of new artifacts. As more artifacts and other online documentation are prospectively linked to the architecture, the quality of the topic model improves, as we will see later in the paper. The topics become more semantically coherent and interpretable, since the collection of underlying artifacts becomes larger and more diverse. As depicted in Figure 3, there appears to be a beneficial symbiosis between prospective capture and topic modeling.

There are other potential benefits to applying topic modeling to prospective traceability. False positives in automatic link generation can be minimized by comparing the θ_d distributions; if the difference is greater than a threshold, then that candidate traceability link can be treated as noise and be discarded. Furthermore, artifacts can be clustered together in groups based on semantic topic similarity and displayed to the developer, which aids in artifact retrieval and allows for automated recommendation of similar artifacts. Finally, extensions to the LDA topic model, such as the Author-Topic model [42] and the Dynamic Topic [11] model can give an indication of which topics developers are working on as well as the temporal evolution of topics, both of which are of interest to project management.

5. TOOL SUPPORT

We implemented several different tools that can be used to collectively perform prospective traceability with topic modeling. Figure 4 shows the high-level interplay between our topic-based artifact search engine (TRASE), our prospective capture tool (ACTS), and our topic-enhanced architecture visualization tool (TEAM). The user's interaction with TRASE can be prospectively recorded with ACTS, which produces trace links. These trace links are then augmented with topical information and are visualized within TEAM. We provide details for each tool and discuss the application of our approach on ArchStudio [16, 17], a mature software project at UC Irvine.

Figure 5: TRASE tool which performs LDA in real-time on the results of artifact search.

5.1 TRASE tool

To investigate our idea of improving prospective capture with topic modeling, we created the Topically-Rich Artifact Search Engine (TRASE). TRASE is a search engine over the artifacts of the project which dynamically learns an LDA topic model on the search results in real-time.

We implemented TRASE using a combination of Perl, AJAX, and Lucene technologies. Lucene provides keyword-based search results over the set of artifacts. Since our topic model inference algorithm (CVB0) is very fast (on the order of a few seconds), TRASE dynamically learns a new topic model over the artifacts returned by Lucene and displays the high-probability words of the learned topics (ϕ) as well as the distribution over topics (θ_d) for each artifact d returned by Lucene. As shown in Figure 5, each topic is represented by a different color, and the distribution θ_d is depicted as a color-bar underneath each search result. Thus, it is easy to visually determine the topical composition of each artifact.

TRASE also includes functionality for ordering the results by topic as well as by similarity to another document. Thus, the developer can filter by desired topic or find artifacts with the most similar topic proportions. For instance, if the developer wants artifacts similar to result #4, clicking on "Similar Pages" would order the results by similarity to #4, where the similarity measure used is KL-divergence.

As long as the prospective capture of ACTS is activated, each artifact that the developer visits via the TRASE tool is recorded as a trace link to the specific component on which the developer is working. Note that this LDA-based search technique can also be applied to general web pages on the Internet. TRASE is a topic-enhanced way to search for artifacts that enables prospective traceability to be more efficient and accurate.

5.2 ACTS tool

To capture links, we use our previously-built ACTS traceability tool on top of ArchStudio 4 [9]. ACTS prospectively captures links by recording the user interaction with the architecture and other artifacts. We center our links to the architecture and our first class n-ary links are stored in an XML Architecture Description Language (xADL) file [18]. ACTS combines prospective capture with the use of rules and open hypermedia adapters. The tool-specific recording adapters capture all the user actions on the artifacts and have been implemented for applications such as the Mozilla Firefox browser, MS Word, MS Excel, MS Powerpoint, and Adobe Acrobat. Rules are then applied to these recorded actions to determine traceability links, filter extraneous links, and assign link semantics. For instance, if a user visits URLs from the TRASE search results (or any other web site), the Firefox-recorder would capture links to these URLs.

Figure 6 shows the end of a recording session which lists the artifacts visited during that session. After performing some rule-based filtering, these artifacts are then associated with a particular component. Note that ACTS can capture links to artifacts of varying media types, such as images like .png files. This ability to capture links to non-text artifacts is an advantage to performing prospective traceability.

5.3 TEAM tool

To visualize our architecture with traceability links, we built the Topic-Enhanced Architecture Mashup (TEAM) tool. This tool aggregates the linked information from ACTS and overlays this information on top of the software architecture. TEAM also takes the results of the topic model and graphically renders the topical information of both the architectural components and the artifacts. As shown in Figure 7, TEAM allows one to view the entire system and identify which components are related to a particular topic. In order to semantically classify component c, we use the component's topic distribution Θ_c which was defined in Section 4. If the component's probability for topic t is greater than a threshold (i.e. $\Theta_{ct} > 0.25$, a threshold we set arbitrarily), we associate the component with that topic. Note that TEAM displays the high-probability words of each topic on the left pane. When a topic is clicked, the components associated with that topic are highlighted in the topic's color. Thus, it is easy to determine the topical makeup of various sections of the architecture. One can also zoom in on an individual component and view the topic distributions θ_d of the component's related artifacts, as shown in Figure 8. This visual information allows the user to quickly locate an artifact of interest, and once the user clicks on an artifact, the artifact is displayed in its native editor. Thus, at both the system-level view and the component-level view, users can identify at a glance both the trace and topic information and navigate to desired artifacts.

6. EVALUATION

To evaluate our traceability approach, we apply our techniques to the ArchStudio software project and we perform a feature comparison between our traceability tools and other tools in the literature. We also report timing and accuracy results when applying Latent Dirichlet Allocation to ArchStudio artifacts, and we compare the precision-recall scores of LDA and LSI on the EasyClinic data set [13].

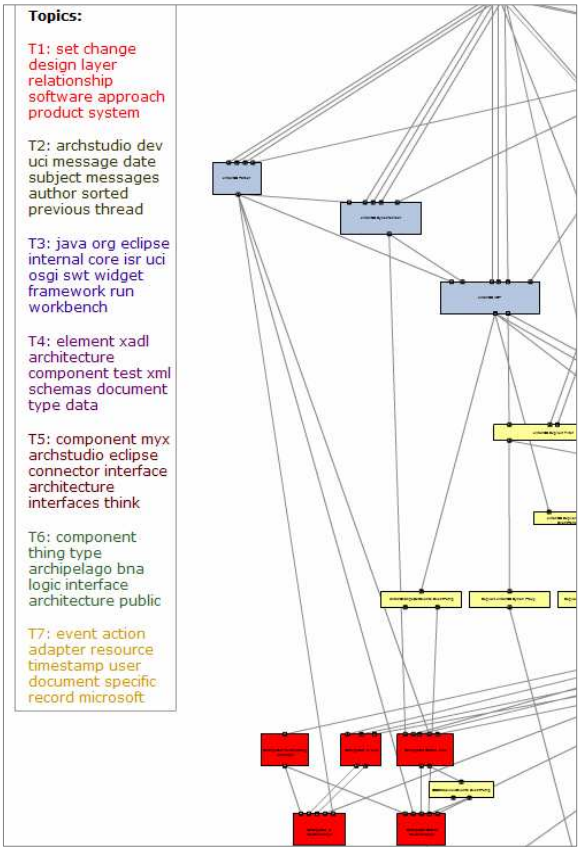

Figure 7: TEAM tool which provides a mashup of topical information over the software architecture.

6.1 Case study on ArchStudio

We conducted a case study on the ArchStudio 4 system [16, 17]. The latest version of ArchStudio consists of 56 components and 42 connectors, which corresponds to more than 85KLOC. The system comes with a heterogeneous set of artifacts: tutorials in PDF, presentations in PowerPoint, project specifications in Word, archived emails in an online mailing list, Wiki pages and bug reports in Trac, ArchStudio web pages, research publications in PDF, and developer notes in various file formats.

We followed the scenario of understanding the unfamiliar parts of the ArchStudio system. In the process of understanding each component, we navigated to various ArchStudio artifacts and web sites while prospective capture was enabled in ACTS; thus, links were being captured as we investigated each component. Since the architecture already has a consistent mapping with the underlying code (i.e. the system will not run if an inconsistency exists), it was not necessary to create links to the source code.

To make our set of artifacts amenable to topic modeling, we converted our artifacts to plain text files. A few documents were very long, and thus we broke these documents (by section) into separate files. All the other preprocessing steps (such as removing stop words and stemming) are automated within our topic modeling algorithm.

We tested the capability of our TRASE tool on our set of ArchStudio artifacts by querying various component names. We find that the tool performs very efficiently (returning re-

sults in a few seconds) and that the displayed semantic information is accurate. Figure 5 displays the results of querying "ArchEdit", which is a tool within the ArchStudio framework. We see that various ArchStudio-related topics are found: an XML schema topic (T1), a UCI email topic (T2), a BNA tree topic (T3), a Java/Eclipse topic (T4), a component/connector topic (T5), a similar Myx framework topic (T6), and an Archlight test topic (T7). In each of the returned results, the yellow topic appears since ArchEdit and Archlight terms are collocated in these documents. We also see the presence of the email topic (in olive-green) in results #2, #3, and #4 because these specific results are precisely the artifacts from email archives. Also note that result #2 is an email about the Myx framework with relationships to Archlight, which confirms the accuracy of the learned topic distribution that is displayed.

Once we captured our links through the ACTS prospective capture tool, we performed topic modeling on our set of artifacts, generating topic proportions for both the artifacts and related architectural components. We then rendered this information within the TEAM mashup tool. As depicted in Figure 7, we clicked on the first topic (about "change sets") and the resulting highlighted components (in red) were all related to this topic: ChangeSet Relationship Manager, ChangeSet ID View, ChangeSet Status View, ChangeSet Id Relationships, and ChangeSet Status Relationships.

Figure 8 displays the artifacts related to the Archipelago component, which is the visual editor within ArchStudio. Most of the linked artifacts are related to topic T6 which is about Archipelago or its visual elements. Topic T2 (the olive green topic), is found among the email artifacts. This result makes sense because the high-probability words within topic T2 are typically found in emails. As with the TRASE tool, the width of the color in the color-bar indicates the strength of its relation to the topic. The artifact in the fourth row is largely comprised of topic T5, which is the "Myx" topic. When we examined the artifact, we found that it is indeed largely about the Myx architectural style. The topic with the second-highest probability for this artifact is topic T4 (about "xADL"), and when we examined the artifact, we found that it also discusses xADL elements. Through these examples, we see how TEAM allows one to quickly identify the topics of a component's related artifacts.

When we presented our TEAM tool to the ArchStudio developers, the response was largely positive. One developer stated that topics aid in system comprehension. This developer recounted his previous experience of having to manually go through source code when he was initially learning ArchStudio, and he stated that this topic-based tool would have saved him time and effort in understanding ArchStudio:

"[TEAM] provides me a central access to all the available materials, so that I don't have to go to different places searching for the same topic, and possibly get overwhelmed by lots of unrelated materials. At this point, I believe the mashup tool not only saves users' time, but also improves the usability of existing technical materials."

In addition, the topic visualization also aided the developers in analyzing the correctness of the related artifacts. The visualization enabled the developers to quickly identify whether the captured links were correct and whether we were missing some links. The developers generally thought that the captured links were accurate.

The developers also gave several suggestions for improvement. One suggestion is to provide topics at finer levels of granularity to minimize the number of components to examine. Another suggestion is to make the representation of topics more clear, since there were similar high-probability words that appeared in more than one topic.

Table 1: Feature comparison between tools

Feature	**Our Combined Approach**	**Retro. Trace. [20, 23, 25, 27, 41]**	**Prosp. Trace. [3, 28, 29, 32, 38, 39]**
Generates candidate links based on textual similarity	X	X	
Generates links based on user interaction	X		X
Generates links across heterogeneous artifacts (i.e. non text-based)	X		
Uses fully probabilistic interpretation of semantic topics	X		
Adds links in an online, incremental fashion	X	X	X
Detects topics from artifacts automatically	X	X	
Visualizes semantic topics on architectural mashup	X		

6.2 Feature Comparison

Table 1 shows a feature comparison between our approach and the existing retrospective and prospective traceability techniques in the literature. The table shows that our approach provides capabilities that are not currently supported in other tools, such as generating traceability links across a wider range of artifacts including graphic and media files, using a fully probabilistic interpretation of semantic topics, and visualizing topics on top of the architecture graph through the use of mashups.

6.3 Accuracy and timing results for LDA

In this section, we briefly discuss accuracy and timing results when applying Latent Dirichlet Allocation to software artifacts. In Figure 9, we see that the test perplexity of the model decreases as the number of artifacts increases. Perplexity is a widely-used metric for topic models that indicates the quality of the model (and a lower perplexity indicates a better model) [7]. This unsurprising result suggests that it is beneficial to expand the artifact base for topic modeling, suggesting that the prospective capture of artifacts can benefit topic modeling just as topic modeling benefits prospective traceability.

In Figure 10, we show the amount of time it takes to learn a topic model for different settings of the number of topics T, as a function of the number of artifacts to model. We see that there is only a linear increase in computation time

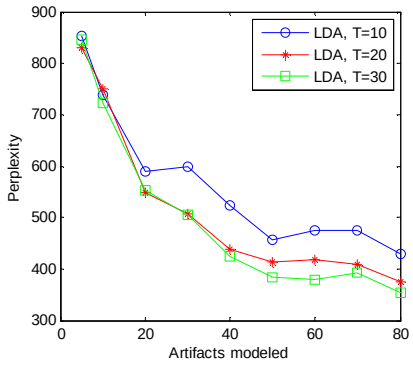

Figure 9: As the number of ArchStudio artifacts increases, a higher-quality topic model is learned.

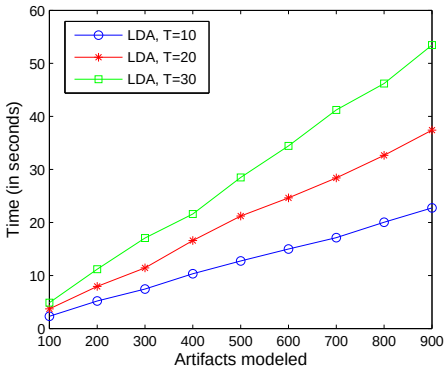

Figure 10: Timing results for LDA on ArchStudio artifacts.

as the number of artifacts increases. Note that the timing results are measured seconds and that we used a single-core 1.6Ghz machine to run these experiments. With a multi-core processor, one can perform topic modeling on a moderately-sized data set in near real-time, suggesting the scalability of our approach. We also note that the other facets of our approach are also scalable: ACTS is scalable since it performs link capture incrementally, and TEAM is scalable since mashup visualizations can be done on large-scale data with the appropriate tools (e.g. Google Maps API).

In Figure 11, we compare the precision-recall scores of LDA vs. LSI on the EasyClinic data which was used in the TEFSE 2009 traceability challenge [2]. Note that the precisions are generally low since we clumped the four different artifact types in the challenge into one global set of artifacts. For this data set, we see that LDA performs better than LSI, perhaps due to the fact that LDA is fully probabilistic and includes priors on θ and ϕ which can act as regularization. This result is somewhat expected, since Hofmann discovered that PLSI generally outperforms LSI [26] and since LDA is a fully Bayesian version of PLSI. Nonetheless, it is reassuring to know that in the realm of software artifacts, LDA performs as well or better than LSI on this data.

6.4 Discussion

Our ArchStudio case study suggests that combining topic modeling with prospective traceability provides useful results. Our task of learning the unfamiliar parts of the Arch-

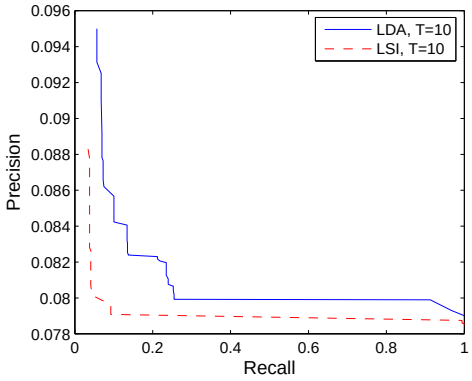

Figure 11: LDA perform better than LSI on the EasyClinic data set, in terms of precision-recall.

Studio system was supported by our traceability tools. We found that the learned semantic topics corresponded well with the actual content within the artifacts, and these topics allowed us to efficiently find and evaluate artifacts. The ArchStudio developers were also able to easily verify the traceability links by using the semantic information visualized by the TEAM tool. Our empirical results also suggest that LDA is competitive or superior to LSI and that our approach can scale to handle larger numbers of artifacts.

There are limitations to our approach. First, we assume the existence of an architecture, since we center our traceability links to the architecture. This is not an unrealistic assumption, since we believe that every system has an underlying architecture, whether or not it is explicitly documented. In the event that the architecture is not explicitly documented (or incomplete), we can create virtual components to correspond to the source code. Secondly, we perform our topic analysis on text-based artifacts. The non-text artifacts were ignored by the topic model algorithm. In the future, we plan to use text metadata associated to non-text artifacts in order to include them in the topic model.

7. CONCLUSIONS

Automated approaches for generating traceability links have largely focused on recovering trace links retrospectively. This paper builds upon our previous work of capturing links prospectively, while the artifacts are generated or accessed in the context of a development task. In this work, we employ topic modeling in order to aid users in analyzing the semantic nature of artifacts as well as the entire software architecture. Our approach is scalable since the trace links are incrementally captured, the problem space is bounded by tracing to the architecture, and the topic model algorithm is very efficient.

This paper marks the initial investigation into combining prospective link capture with machine learning techniques, and there are many open research questions: Can prospective link capture be successfully combined with retrospective IR techniques? Is it possible to relate the automatically generated topics to high-level concepts such as features, non-functional properties, or concerns? Is it possible to successfully perform topic modeling on long artifacts by section? Our results suggest that the combination of prospective traceability with topic modeling is a promising area of research that can be useful in practice.

8. ACKNOWLEDGMENTS

We thank the INF117 students, A. Marron, S. Cutler, and D. Kwok, for providing software for the visualization tool, and D. Purpura for the ACTS user-interface. We thank Y. Zheng and S. Hendrickson for evaluating the captured links rendered in the visualization tool. This effort is supported by the US National Science Foundation under grants IIS-08-08783, CCF-0917129, and by an NSF graduate fellowship.

9. REFERENCES

[1] The Jazz Project. http://jazz.net.

[2] TEFSE traceability challenge, 2009. http://web.soccerlab.polymtl.ca/tefse09/Challenge.htm.

[3] I. Altintas, O. Barney, and E. Jaeger-Frank. Provenance collection support in the Kepler Scientific Workflow System. In *Proc of the IPAW*, 2006.

[4] K. M. Anderson, S. A. Sherba, and W. V. Lepthien. Towards large-scale information integration. In *ICSE*, 2002.

[5] K. M. Anderson, R. N. Taylor, and J. Whitehead. Chimera: Hypermedia for heterogeneous software development environments. *ACM TOIS*, 18(3), July 2000.

[6] A. Asuncion, P. Smyth, and M. Welling. Asynchronous distributed learning of topic models. In *Advances in Neural Information Processing Systems 21*, 2009.

[7] A. Asuncion, M. Welling, P. Smyth, and Y. W. Teh. On smoothing and inference for topic models. In *Uncertainty in Artificial Intelligence (UAI)*, 2009.

[8] H. Asuncion, F. François, and R. N. Taylor. An end-to-end industrial software traceability tool. In *Proc of 6th Joint Meeting of the ESEC/FSE*, pages 115–124, 2007.

[9] H. Asuncion and R. N. Taylor. Capturing custom link semantics among heterogeneous artifacts and tools. In *Int'l Workshop on TEFSE*, Vancouver, British Columbia, 2009.

[10] P. Baldi, C. Lopes, E. Linstead, and S. Bajracharya. A theory of aspects as latent topics. In *OOPSLA*, 2008.

[11] D. Blei and J. Lafferty. Dynamic topic models. In *Int'l Conf on Machine Learning*, 2006.

[12] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[13] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella. On the role of the nouns in IR-based traceability recovery. In *Proc of 17th Int'l Conf on Program Comprehension (ICPC)*, 2009.

[14] J. Cleland-Huang and R. Habrat. Visual support in automated tracing. In *Int'l Workshop on Requirements Engineering Visualization*, 2007.

[15] J. Cleland-Huang, R. Settimi, E. Romanova, B. Berenbach, and S. Clark. Best practices for automated traceability. *Computer*, 40(6):27–35, June 2007.

[16] E. Dashofy. *Supporting Stakeholder-Driven, Multi-View Software Architecture Modeling*. Ph.D. (Info and Comp Science), Univ of Calif, Irvine, 2007.

[17] E. Dashofy, H. Asuncion, S. Hendrickson, G. Suryanarayana, J. Georgas, and R. N. Taylor. ArchStudio 4: An architecture-based meta-modeling environment. In *29th ICSE, Comp Vol*, 2007.

[18] E. Dashofy, A. van der Hoek, and R. N. Taylor. A comprehensive approach for the development of XML-based software architecture description languages. *ACM TOSEM*, 14(2):199–245, April 2005.

[19] A. De Lucia, R. Oliveto, and G. Tortora. Assessing IR-based traceability recovery tools through controlled experiments. *Empirical Software Engr*, 14:57–92, 2009.

[20] S. Deerwester, S. Dumais, T. Landauer, G. Furnas, and L. Beck. Improving information retrieval with latent semantic indexing. In *Annual Meeting of the American Society for Info. Science 25*, 1988.

[21] R. Dömges and K. Pohl. Adapting traceability environments to project specific needs. *CACM*, 41(12):54–62, 1998.

[22] C. Duan and J. Cleland-Huang. Clustering support for automated tracing. In *Proc of ASE*, 2007.

[23] M. Grechanik, K. S. McKinley, and D. E. Perry. Recovering and using use-case-diagram-to-source-code traceability links. In *Proc of 6th Joint Meeting of the ESEC/FSE*, 2007.

[24] J. H. Hayes and A. Dekhtyar. Humans in the traceability loop: Can't live with 'em, can't live without 'em. In *Int'l Workshop on TEFSE*, 2005.

[25] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE TSE*, 32(1):4–19, 2006.

[26] T. Hofmann. Unsupervised learning by probabilistic latent semantic analysis. *Machine Learning*, 42(1):177–196, 2001.

[27] H. Jiang, T. Nguyen, I. Chen, H. Jaygarl, and C. Chang. Incremental latent semantic indexing for effective, automatic traceability link evolution management. In *Proc of Automated Software Engineering (ASE)*, 2008.

[28] F. Jouault. Loosely coupled traceability for ATL. In *Proc ECMDA Workshop on Traceability*, 2005.

[29] M. Kersten and G. C. Murphy. Mylar: A degree-of-interest model for IDEs. In *Aspect-oriented Soft Dev*, 2005.

[30] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi. Mining concepts from code with probabilistic topic models. In *Proc of ASE*, 2007.

[31] G. Maskeri, S. Sarkar, and K. Heafield. Mining business topics in source code using latent dirichlet allocation. In *Proc of India Software Engineering Conf*, 2008.

[32] N. Medvidovic, P. Gruenbacher, A. Egyed, and B. W. Boehm. Bridging models across the software lifecycle. *Journal of Systems and Software*, 68(3), 2003.

[33] Q. Mei, X. Shen, and C. Zhai. Automatic labeling of multinomial topic models. In *KDD*, 2007.

[34] C. Neumüller and P. Grünbacher. Automating software traceability in very small companies - a case study and lessons learned. In *Proc of ASE*, 2006.

[35] D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed inference for Latent Dirichlet Allocation. *Advances in Neural Info Processing Systems*, 20, 2007.

[36] D. Newman, C. Chemudugunta, and P. Smyth. Statistical entity-topic models. In *KDD*, 2006.

[37] E. Nistor. *Concern-Driven Software Evolution*. Ph. D. Thesis (Info and Comp Science), Univ of Calif, Irvine, 2009.

[38] F. Pinheiro and J. Goguen. An object-oriented tool for tracing requirements. *IEEE Software*, 13(2):52–64, 1996.

[39] K. Pohl. PRO-ART: Enabling requirements pre-traceability. In *Proc Requirements Engineering*, 1996.

[40] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling. Fast collapsed Gibbs sampling for Latent Dirichlet Allocation. In *14th ACM SIGKDD*, 2008.

[41] D. Poshyvanyk, A. Marcus, V. Rajlich, Y.-G. Gueheneuc, and G. Antoniol. Combining probabilistic ranking & latent semantic indexing for feature identification. In *ICPC*, 2006.

[42] M. Rosen-Zvi, T. Griffiths, M. Steyvers, and P. Smyth. The author-topic model for authors and documents. In *UAI*, 2004.

[43] D. I. K. Sjoberg, B. Anda, et al. Conducting realistic experiments in software engineering. In *Int'l Symp on Empirical Software Eng*, 2002.

[44] R. N. Taylor, N. Medvidovic, and E. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, 2010.

[45] Y. W. Teh, M. Jordan, M. Beal, and D. Blei. Hierarchical dirichlet processes. *Journal of the Am Stat Assoc*, 101(476):1566–1581, 2006.

[46] K. Tian, M. Revelle, and D. Poshyvanyk. Using latent dirichlet allocation for automatic categorization of software. In *Mining Software Repositories*, 2009.

[47] X. Wei and W. B. Croft. LDA-based document models for ad-hoc retrieval. In *SIGIR Conf.*, 2006.

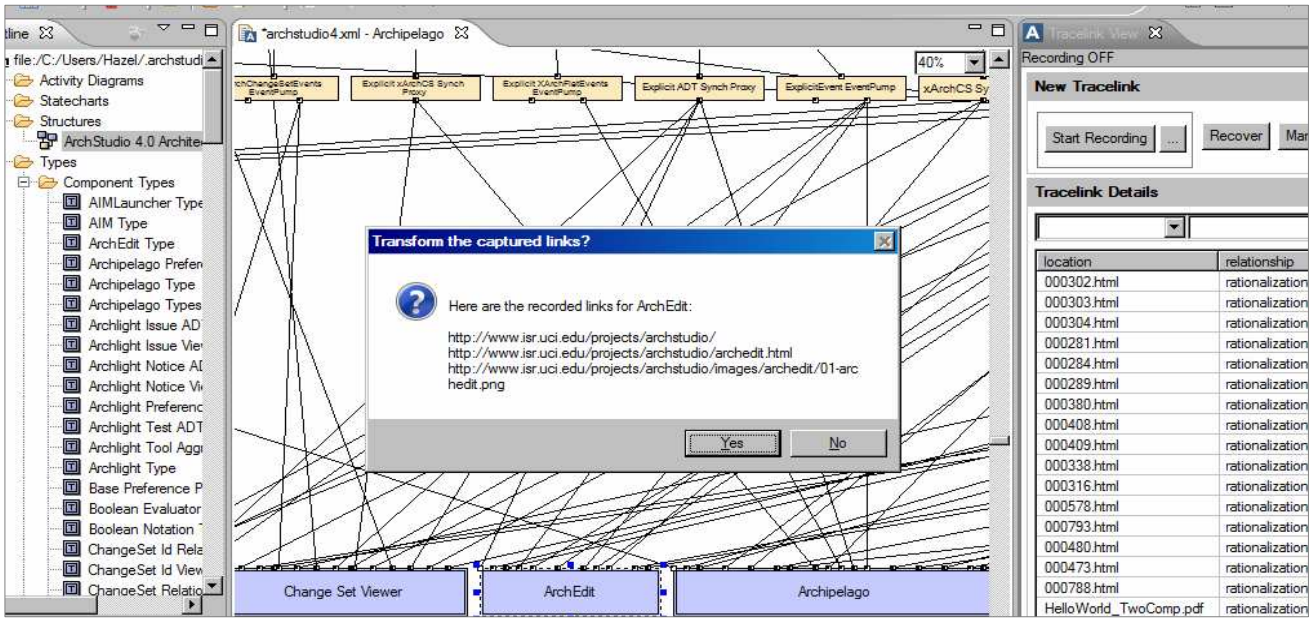

Figure 6: The ACTS prospective capture tool automatically traces between the architectural components on which the developers are working and the artifacts that the developers access.

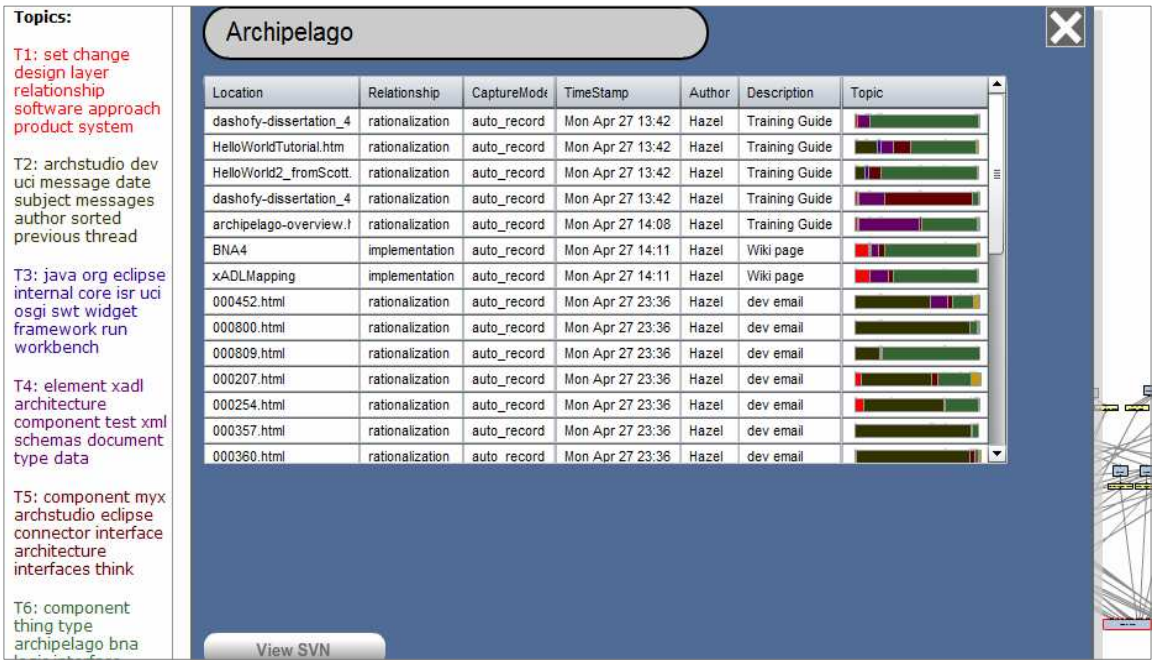

Figure 8: Clicking on an individual architectural component within the TEAM tool opens a list of relevant artifacts, each displaying a set of topic proportions.