2nd International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering, Montreal, Canada, May 2019

Leveraging Usage Data of Software Architecture Artefacts

Moon Ting Su
Faculty of Computer Science and Information Technology
University of Malaya
Kuala Lumpur, Malaysia
smting@um.edu.my

John Grundy
Faculty of Information Technology
Monash University
Clayton, Australia
john.grundy@monash.edu

John Hosking
Faculty of Science
University of Auckland
Auckland, New Zealand
j.hosking@auckland.ac.nz

Ewan Tempero
School of Computer Science
University of Auckland
Auckland, New Zealand
e.tempero@auckland.ac.nz

***Abstract*—** A number of existing fields (such as computational wear, social navigation, collaborative filtering, social filtering, and wear-based filtering) have shown that usage data (comprising interaction and annotation data) can be leveraged to provide assistance for sense-making of materials, for finding and filtering information, and for exploring an information space. We believe providing such assistance in Software Architecture Knowledge Management (SAKM) can enhance two of the five knowledge management activities of SAKM, namely, Architecture Knowledge (AK) sharing and AK application/using. To this end, we propose to include usage data capture and visualization as essential features of SAKM tools. We discuss key requirements and challenges for SAKM tools leveraging such usage data. We describe KaitaroCap, an exemplar tool that captures and provides visualization of usage data, in particular the usage data of software architecture documents.

Keywords-software architecture; knowledge management; usage data; visualization

I. INTRODUCTION

In this paper, we put forward a proposal to include usage data capture and visualization as essential features of Software Architecture Knowledge Management (SAKM) tools. We posit that automated capture of how software personnel actually make use of Software Architecture (SA) artefacts could produce a valuable dataset for empirical research in the SA domain. Usage data can be categorized into interaction and annotation data. Interaction data refers to data generated from users' interaction with software artefacts such as selecting, clicking, traversing, opening, editing, and so on. Annotation data refers to additional information users attach to the software artefacts to jot down personal opinion (such as in the forms of ratings and comments) on certain aspects of the artefacts (such as quality and relevance) or to create additional categorization of software artefacts (such as in the forms of tags) not supported by the underlying tool. Leveraging a community's or own's past usage data of SA artefacts could bring a number of benefits. For example, during maintenance it would be possible to see which SA artefacts are most frequently accessed or edited. The former could signify the most important artefact and the latter, the most problematic artefact or centre of change.

Section II details existing work on leveraging usage data. Section III describes the related work in SAKM. Section IV presents our proposal for leveraging usage data in SAKM. Section V presents KaitoroCap, an exemplar tool that provides usage data capturing and visualization features. Section VI concludes the paper.

II. EXISTING WORK ON LEVERAGING USAGE DATA

A. Computational Wear

In the domain of document processing, computational wear [1] refers to recording users' interactions with computational objects (such as documents, menus and emails), and making the interaction history available as parts of the objects, thereby creating history-enriched objects [2]. Some examples of computational wear are edit wear and read wear. Edit wear involves capturing of authors', and read wear involves capturing of readers' interaction histories with computational objects and making that available on the objects. Wears (or 'usage marks') provide additional information that helps in sense-making of the materials, for example, highlighting the most unstable portions where editing is frequent and most read portions.

An implementation example of computational wear is the Zmacs editor, where wears are mapped onto document scroll bars to mark where editing and reading occur [1]. The 'wears' display the usage history of a document. For example, read wear displays a document's readership history based on read time accumulated for each visible line.

B. Social Navigation

Social navigation is a type of information navigation mechanism [3], where decisions of moving from one item to another in the information space are informed by the behaviour of other people [4]. The behaviour can be visible, such as the movement of users from one online chat room to another, or aggregated and hidden in the interaction history of an object. Ideas in social navigation often originate from the physical world [4]. Examples of social navigation in an online world are the inclination to join a chat room based on the current active users and discussion topic, and the tendency to follow information traces [4] left by previous users. The traces such as comments, reviews, ratings, and so on, can be used as shortcuts to information needed [5]. Users leave 'footprints' [5], known as computational wear [1] that others can use to find their way through the same information space.

Key properties of social navigation are dynamism and personalisation [4]. Social navigation traces are not pre-planned but grown dynamically. Therefore, social navigation reflects what people actually do rather than what designers expect people should be doing. In addition, the navigational advice is personalised to the user. Some researchers also regard the visibility of the aggregated behaviour of a community as an important aspect of social navigation systems [6]. Examples include [4]: Amazon.com; EPOL for online food shopping [7]; CoWEB [8] that annotates links on web pages; IBM's WBI (Web Intermediaries) toolkit [9]; Footprints [10]; Edit wear and read wear in Zmacs editor [1].

C. Collaborative Filtering

Collaborative Filtering [11] is "the process of filtering or evaluating items using the opinions of other people" [6]. In collaborative filtering, people collaborate to help one another to perform filtering of information by recording their reactions (in the form of annotations) to the documents they read [11]. The annotations can be used by others' 'filters' to sieve the information to receive. For example, in the case of the Tapestry system supporting moderated newsgroups [11], a user can filter for articles annotated by a specific user. Collaborative filtering systems typically use users' ratings to filter information. Ratings can be scalar, ordinal, binary or unary and collected through explicit and/or implicit means [6].

D. Social Filtering

Previous works on collaborative filtering and social navigation focus on 'implicit' collaboration [6]. In 'implicit' collaboration, the users from whom recommendations are based remain anonymous. In 'explicit' collaboration, the identities of the members of a community are more exposed. This enables users to choose people with similar tastes or interests, and then refine the trust he or she has on these users based on other users' opinions of them. The users-rating-users [6] which is related to the notion of reputation [4], creates explicit social networks which users could navigate to find items-of-interest. This leads to social filtering (also known as social recommendation or social information processing) [12]. This involves social media sites which include blogs, wikis, media sharing sites such as Flickr, del.icio.us and Digg [12]. The characteristics of social media sites are: users actively create or contribute to the content, annotate content with tags, evaluate content by voting or using it, create social networks by making those with similar interests 'contacts' or 'friends'. Examples include social news aggregator such as Digg [12], and photos sharing site such as Flickr [13].

E. Wear-based Filtering

Wear-based filtering [14] refers to a combination of computational wear [1] and collaborative filtering [11, 15]. It makes use of data collected on previous users' interactions with certain computational objects, to filter information presented to future users. Wear-based filtering has been proposed to improve the interface of programming development environment to aid programmers in navigating and understanding unfamiliar source code in order to make changes to it [14]. This involves tracking programmers' code traversal information by using a logger. The interaction data is used to guide other programmers to portions of code that are most related to the current code under inspection. Examples include Team Tracks [16]: 1) Class View Favourites or Code Favourites [14], and 2) Related Items which is similar to Frequently Accessed Next List [14] but also includes most frequently-visited elements.

F. Other Forms of Filtering

To reduce information overload during development, the information presented by a development environment can be filtered and ranked using a Degree-Of-Interest (DOI) model constructed based on a programmer's interaction history with the software program artefacts [17, 18]. This approach involves constructing a task context using a programmer's interaction history with software program artefacts when working on a task [18] or tasks [17], and using the structural relationships of the program artefacts. A weighting for each element in the task context is calculated based on the frequency of interaction with the element and how recent the interactions are. DOI is implemented in Mylar [18] and Mylyn [19] for Eclipse. On the activation of a task, Mylar monitors a user's interactions with code, and uses that to capture in a DOI model, the relevance of code elements to the task [18]. The DOI model is used to filter Eclipse's interface.

DOI has been combined with degree-of-authorship (DOA) to form degree-of-knowledge (DOK) [20] to find who knows what about the code. A developer's DOK value for each source code element is calculated by combining the developer's authorship and interaction information for that element. The DOK value serves as an indicator of a developer's knowledge of that code element. Tasktop [21] and Tasktop Dev [22] extends Mylyn's tracking of interactions with code to include interactions with web pages and desktop documents. The DOI model built from these is used to filter the Eclipse's interface to include only the code, URLs of web pages and documents that are needed for a task.

G. Summary of Work on Leveraging Usage Data

In summary, existing work on leveraging usage data spans a number of areas that are inter-related but different. Computational wear focuses on creating 'usage wear' on computational objects by making users' interaction history

with the objects part of the objects. Social navigation employs 'usage wear' as navigation trace or 'footprint' to help other users in navigating the same information space more quickly. Collaborative filtering focuses on the collaboration between users in providing annotations (which is one type of 'usage wear') on information read. The annotations serve as collaborative filters to sieve the information to receive. Social filtering uses explicit collaboration enabled by exposed identities of a community's members, to extend collaborative filtering to use social networks for filtering, and to refine social navigation to center on explicitly created social networks. Wear-based filtering combines computational wear and collaborative filtering, and uses users' interaction history with source code elements to filter the interface of programming development environment. The set of studies on DOI model has a similar purpose, but differs in its emphasis on the notion of task and its uses of a user's own usage history in the filtering of the interface.

III. Related work

A. SAKM

Effective management of Architecture Knowledge (AK) is critical for improving an organization's architectural capabilities and the quality of the architecting process [23]. A recent systematic literature review (SLR) has identified 56 SAKM approaches and their general and advanced support for the five knowledge management (KM) activities (capturing, using, maintaining, sharing, and reuse of AK) [24]. In the SLR, an SAKM approach is "any concept, method, tool, and technique (or any combination thereof) that has been specifically developed to support SAKM within software development".

AK capturing refers to making AK explicit by formalizing and storing it [24]. AK sharing refers to distributing the documented knowledge among different stakeholders and in different contexts, and making AK accessible to involved stakeholders easily. AK application/using refers to using the AK in different architecture-related activities. AK maintenance refers to keeping documented AK up-to-date. AK reuse refers to reuse AK across projects.

General support for KM activities provides basic data management (Create/Read/Update/Delete) functionality for AK [24]. Advanced support extends the general book-keeping functionality to include additional support to address efficiency and usability for a particular SAKM activity. For example, advanced support for AK capturing might include ways to reduce effort required in capturing.

The SLR found that, except for the activity of AK using, the support for the other four KM activities are insufficiently addressed causing none of the numerous SAKM approaches been adopted by the industry.

Li et al. conducted a systematic mapping study on the application of KM activities in assisting software architecting activities [25]. In this study, KM activities (as called in [24]) are termed knowledge-based approaches, and a knowledge-based approach refers to any approach from KM that can directly contribute to the architecting process. Five knowledge-based approaches (or activities) were identified: knowledge capture and representation, reuse, sharing, recovery (such as documenting past architectural design decisions), and reasoning (such as reasoning based on the rationale of existing design decisions to make new decisions). They found that among the KM activities, knowledge capture and representation is the most popular approach used in architecting activities, and knowledge recovery approach is seldom used [25].

B. SAKM tools

SAKM tools have undergone three periods of evolution [26]. Most of the first generation (2004-2006) tools focused on knowledge representation and capturing using templates list of attributes. The second generation (2007-2010) tools focused more on AK sharing (through Web or Wiki technologies, or tool repository accessible by collaborative group of users) and basic personalization mechanisms. The third generation (2011–2014) tools focused more on the collaborative aspects, reuse of AK, fuzzy decision-making and providing advice on the creation and use of AK.

Better SAKM requires better support for evolution and more advanced collaborative mechanisms (such as for distributed teams to reach group consensus) [26]. In line with that, we propose collaborative mechanisms that leverage usage data of team members.

C. SA Visualization

A recent SLR found four types of visualization techniques (VTs) being used in SA [27]: graph-based, notation-based, matrix-based, and metaphor-based. The graph-based VT uses nodes and links to represent the structural relationships between architecture elements. Notation-based VT uses modeling languages such as SysML, UML, and specific notation languages for SA visualization. Matrix-based VT typically uses tabular form to provide complementary information to graph-based VT. Metaphor-based VT uses familiar physical world contexts to visualize SA entities and their relationships. For example, using a city metaphor, a source file is represented as a "office building" and the number of floors of the building indicates the number of global variables declared in the file.

The main uses of VTs in SA are for architecture recovery and evolution activities [27]. One of the top three purposes of using VTs in SA is to "improve search, navigation and exploration of architecture design". This is in line with the advantages we expect to gain from visualizing usage data especially during the architectural maintenance and evolution activity. Types of VTs that can be used for visualization of usage data are graph, matrix, and metaphor.

D. Software Architects and AK

Practicing software architects tend to be lonesome decision makers and not community builders, spending most of their time on making architectural decisions, who mainly consume but neglect documenting and sharing of AK [28].

More experienced architects were more engaged in auditing activities and documenting AK. In addition, practicing software architects do not perceive themselves as oriented to communication with other stakeholders [29]. In the aspects of mentoring and leadership, software architects want to do more than what they have been doing [30].

Capturing and sharing their usage data and profiles can help software architects to be involved indirectly in community building, communication with other stakeholders, and mentoring novice members in the team.

IV. LEVERAGING USAGE DATA IN SAKM

The review in Section II shows that the additional information derived from usage data are useful in a number of ways. These include providing assistance for sense-making of materials, finding information, filtering information, and exploring an information space by following other users of the same space. We believe providing such assistance in SAKM can enhance two of the five knowledge management activities of SAKM, namely, *AK sharing* and *AK application/using*.

The usage data together with users' profiles and projects' profiles could provide a valuable dataset for empirical research related to usage data of AK. In particular, it can be used to study how computational wear, social navigation, collaborative filtering, social filtering and wear-based filtering can enhance the two activities of SAKM.

A. AK sharing

The current support for *AK sharing* ranges from some form of central knowledge repository remotely accessible by different stakeholders to the provision of specific concepts, processes, or tools for distributing knowledge among stakeholders (e.g., notifications or personalization) [24].

Information derived from usage data can be regarded as an auxiliary knowledge, in addition to the four other categories of AK, namely, Reasoning Knowledge, Context Knowledge, General Knowledge and Design Knowledge [31]. This auxiliary knowledge leverages a project's team members to provide additional assistance for AK sharing among the team members. AK can be distributed to relevant team members, can be found, filtered, and explored based on the interaction behaviour, opinions and profiles of other members.

Authorship and readership (viewing) history of AK can be derived from members' interaction with AK and can be superimposed as computational wear (edit wear and read wear) on AK. This can be used to identify the most frequently accessed or edited AK. The former could signify the most important AK and the latter, the most problematic AK or centre of change. These "usage wears" support social navigation by providing 'footprints' that other members can follow to navigate the AK space more quickly. Members can share their opinions of AK by annotating the AK with ratings, tags, comments, ranking and vote. This annotation data can be used as collaborative filters to sieve the AK to receive. The profiles of team members can be exposed to support social filtering, where members explore the AK space based on the identities of the AK producers and consumers. Wear-based filtering can be achieved by using "usage wears" to filter the AK space a member sees.

Exploration paths derived from interaction data can be used to support the transfer of a special type of tacit knowledge, namely, the procedural knowledge on how people explore existing AK.

Cognitive psychologists classify knowledge mainly into declarative and procedural [32]. Declarative knowledge comprises descriptions of facts and things, or of methods and procedures. Declarative knowledge can be articulated (explicit knowledge). In the knowing-is-in-the-doing view [33], procedural knowledge manifests itself in the doing of something, and is reflected in motor or manual skills and in cognitive or mental skills, and is tacit [32]. For example, playing piano, riding a bicycle, reading customers' faces and moods, thinking, reasoning and making decisions. Being tacit, procedural knowledge cannot be articulated but can be communicated or transferred. As an example, the ability to recognise a face cannot be verbalised but can be developed through exposure to pictures of the face.

B. AK application/using

The current support for *AK application/using* ranges from functionalities to visualize and browse captured AK to the provision of process or tool for specific architecture-related activities such as architecture evaluation or review, and architectural analysis [24]. In addition to the captured AK, usage data can be used to deduce the importance of AK, time spent on AK, and to show recently-visited AK, time line of AK visit, exploration paths through AK space, and possible hidden relationships between AK. This additional information could be useful in architectural maintenance and evolution, which is one of the specific software architecting activity, where architectural changes are accommodated [25]. Other specific architecting activities that cover the entire architecture lifecycle are architectural analysis, synthesis, evaluation, and implementation [25].

In large and long-lived systems where original team members might be gone, the additional information gleaned from usage data could be useful in maintaining and evolving the SA of a system. For example, the architectural model (one type of AK that falls under Design Knowledge [31]) that underwent the most number of changes with timeline of changes that spans across the project life-cycle, could signify a problematic model that should be replaced.

C. Usage data capture and use

To leverage usage data to enhance AK sharing and AK using, four categories of data have to be captured, namely, interaction data, annotation data, user profile data and project profile data. Interaction data is needed to support computational wear, social navigation, and wear-based filtering. Annotation data is needed to support social navigation, collaborative filtering and social filtering. User profile data is needed to support social navigation, collaborative filtering and social filtering.

Table I shows the suggested types of usage data to capture in a SA tool. We suggest that a usage data capture feature should collect all five types of interaction data but can be customised to collect only the required annotation data.

TABLE I. TYPES OF USAGE DATA TO COLLECT

Interaction Data	
Type of data	**Could be used to**
Number of visits	Deduce importance of artefact
Low-level interaction events with artefacts (such as mouse, keyboard and window events)	Deduce time spent on artefact Deduce importance of artefact
Visitation sequence	Create visualization of exploration path Show possible hidden relationships between artefacts
Timestamp of visit	Same as previous row, and Show recently-visited artefacts Show timeline of artefacts visit
Number of changes (edit, delete)	Identify problematic/unstable/center of change artefact
Annotation Data	
Type of data	**Could be used to**
Ratings	Capture scaled personal opinion on certain aspects of the artefacts (such as quality, relevance, importance, etc.)
Tags	Search artefacts Group artefacts
Comments	Capture mental note explicitly or more detailed personal opinion
Ranking	Rank artefacts
Vote	Gather collective preference

The basic information that constitutes a user profile should include user ID, name, job title, experience, skills set and location. Project profile should include project ID and name, start and end dates, members' IDs and roles in the project (such as senior software architect, developer, maintainer, and so on), application domain, and current status.

The usage data and the corresponding users' and projects' profiles can be used to gain insights into research on usage data of AK. For examples, whether there are different types of contributors to the different types of usage data; how do they differ based on their experience, skill set and role; how does the application domain affect the usage data; impact of geographical distance between contributors on usage data, and so on.

Key Requirements - We need to collect usage data at the level of granularity needed. The usage data capturing capabilities should be tailorable to collect usage data for different levels of granularity of SA artefacts. For example, collecting usage data on a per element, per model, per page or per section basis. We need to separate the content of the artefacts from the means of capturing usage data. For example, annotation fields that are used to collect annotation data can be inserted dynamically alongside the artefacts. We need to capture usage data in machine-parsable format where scripts or programs can transform it into the format for analysis.

Usage data visualization – We need to display usage data alongside artefacts (such as showing number of visits or ratings directly on the artefacts). A visualization is needed that provides an overview of usage history of a group of artefacts. For example, showing number of visits for all artefacts of a project in a single graph, showing timeline of artefacts visit, and exploration path. We need to display usage data both in raw form, for example, number of visits and ratings, and in aggregated form, for example, average ratings and percentage of users who visited the artefacts. We need automated analysis of usage data to produce aggregated usage data. Finally, we need to create alternative linkage structures between SA artefacts based on usage data. For example, linking SA artefacts based on exploration paths. This can provide some insight on whether a set of SA artefacts are actually used in the way they are pre-chained by formal models.

Key qualities – **Simplicity**: Interaction data should be collected automatically by the tool with minimum intervention by the users. Easy-to-use annotation fields should be provided to enable users to provide annotation data with little effort. **Reliability**: A tool should be reliable in the capturing and saving of the usage data to ensure that correct usage data is collected. **Performance**: Collecting interaction data involves capturing the low-level events (such as mouse events, keyboard events and so on) generated from users' interactions with the SA artefacts. The tool should be able to capture and save a high volume of these events in real time without much delay noticeable by the users.

Key Challenges – **Noise**: Some portions of interaction data could be generated by non-intentional actions and it is difficult to identify them. For example, a user may access SA artefacts absent-mindedly, but interactions pertaining to this are captured together with intentional interactions. These false steps can taint the interaction data [14], and any derivatives of it. To mitigate this problem, conscientious judgements of users solicited in the forms of explicit annotation data could be used to verify findings produced from interaction data. **Users**: The success in leveraging usage data is very much dependent on the users of the SAKM tools, which include both the producers and consumers of AK. They need to be convinced of the benefits leveraging usage data could bring and motivated to use the usage data capturing features. With proper tool support, interaction data can be captured in the background without much user interventions. **Robustness:** To reduce implementation overlap in different SAKM tools, a framework containing generic usage data capturing and visualization functionalities that can be customized and extended by different SAKM tools should be developed. Such framework can be a part of the community-wide infrastructure for architecture-based software engineering.

V. KAITOROCAP

This section presents an exemplar tool that captures and provides visualization of the usage data of SA artefacts, in particular the usage data of software architecture documents (ADs). KaitoroCap [34-37] is a plug-in for the Atlassian Confluence Enterprise Wiki [38], that supports document creation and dynamic restructuring; annotation; exploration path capturing, visualisation and searching. KaitoroCap can be used for all types of documents and is not limited to ADs.

A. Design and Implementation

KaitoroCap leverages the existing functionalities provided by a wiki environment in the creation and management of documents in the form of wiki pages. In addition, it provides the following:

i. Support for dynamic restructuring of a document, through automatic creation of a page model for each wiki page created or edited.

ii. Capture of users' exploration paths through document (in terms of sequence of visiting pages, interactions with sections of a page, annotation fields and hyperlinks).

iii. Capture of users' annotation data in terms of rating (R), tagging (T), commenting (C) (abbreviated to RTC) on per section basis, through annotation fields inserted dynamically on every section of a page.

iv. Visualisation of exploration paths in the form of tree-view. Contents of visited sections are embedded inside the tree-view visualisation of an exploration path, making a path a dynamic restructuring of the document based on a consumer's usage of the document.

v. Additional search facilities, based on exploration paths.

KaitoroCap was developed using the Atlassian plug-in SDK [38], XWork Action [38] for the controller, Velocity [38] for the view, Java beans for the model, and JQuery [39] for the client-side scripting. Fig. 1 shows the high-level design of KaitoroCap [35] [37].

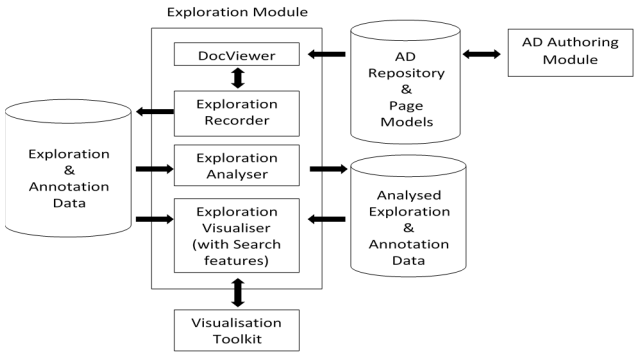

Figure 1. High-level Design of KaitoroCap [37]

Fig. 2 shows an example of creating a new page using KaitoroCap. A section of the page is denoted by a 'level 2 html heading (<H2>)'. When the page is opened for viewing, KaitoroCap constructs the corresponding annotatable section comprising this heading and all succeeding elements until the next <H2>. Fig. 3 shows two of the respective annotatable sections created. The section which the user is interacting with is differentiated from other sections by changing its background colour to green.

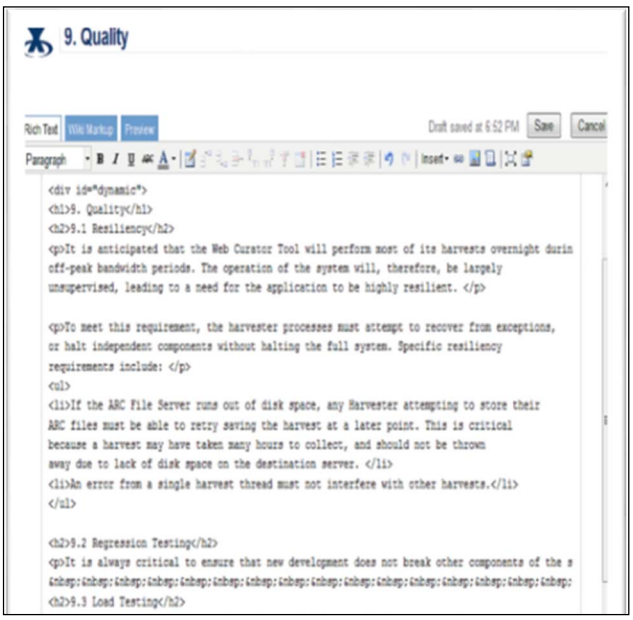

Figure 2. Creating a new page for an AD

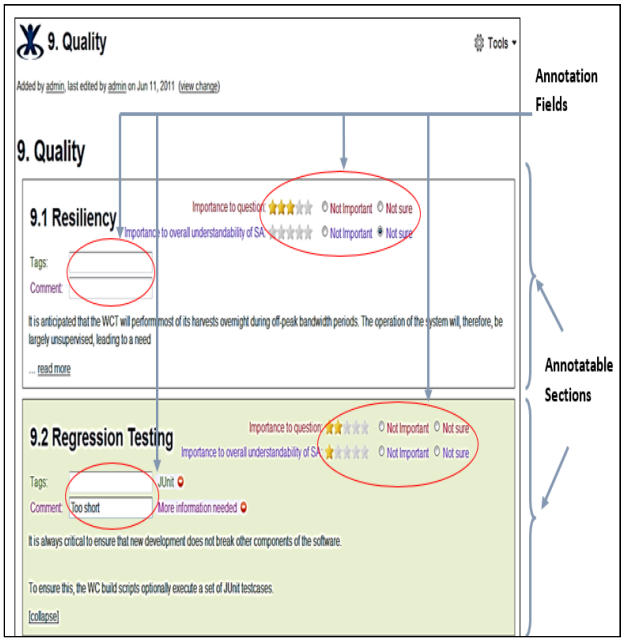

Figure 3. The resulting page with annotatable sections created dynamically

Figures 4-7 show some example tree views of exploration paths created by KaitoroCap. A tree view shows all the events (such as Page Visit, Page Load, Page Unload, Read More, Collapse, Rate, Tag, Comment, Clicking on hyperlink) captured during an exploration path capture.

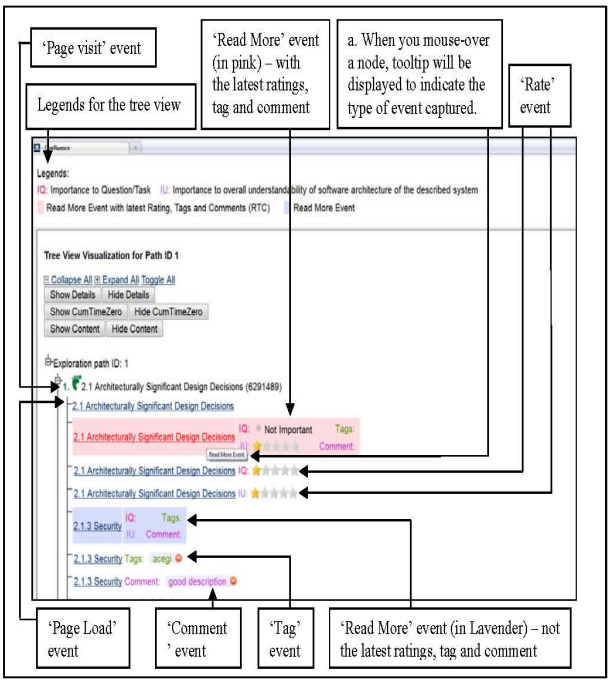

Figure 4. Tree View of Exploration Path (Part a)

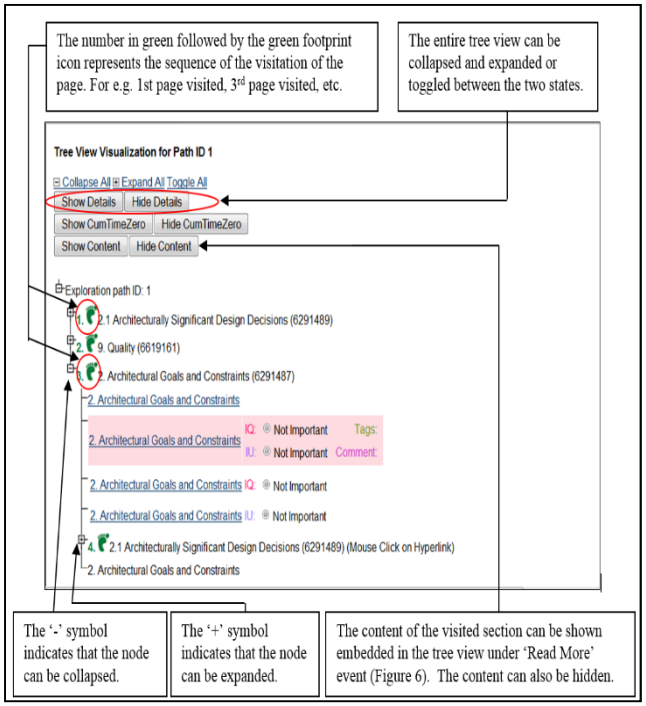

Figure 5. Tree View of Exploration Path (Part b)

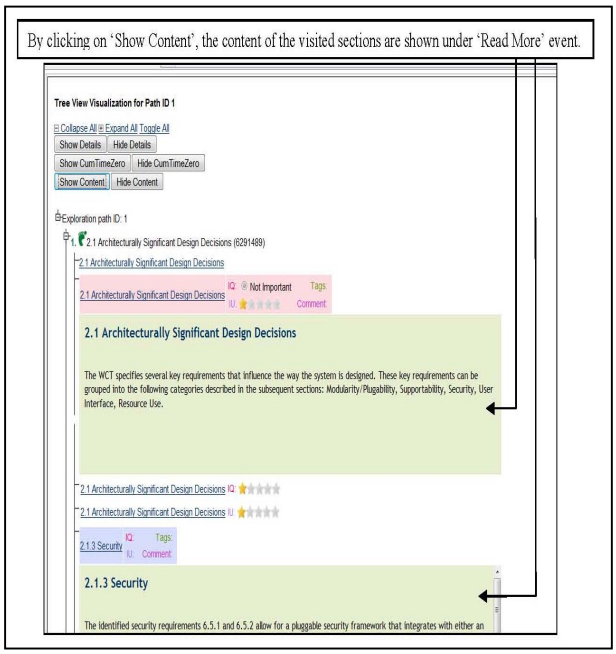

Figure 6. Content of the visited sections embedded in the tree view

Fig. 7 shows two paths found using 'Search Exploration Path'. The tree views here are more compact, showing only sections of the AD expanded for further reading (i.e. the 'Page Visit' and 'Read More' events).

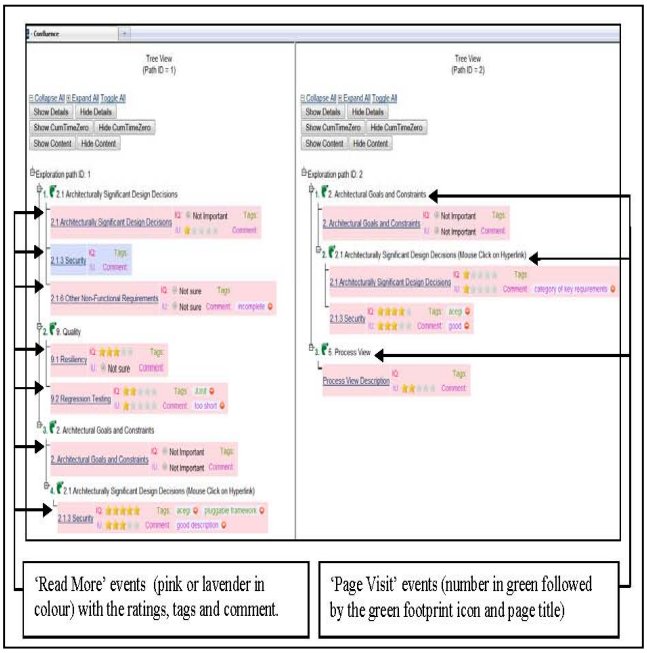

Figure 7. Compact Tree-view of Expanded Exploration Paths.

B. *Comparison with other SAKM tools*

A number of technologies deemed valuable for building SAKM tools [40], were used in the development of

KaitoroCap: wiki, voting and ranking (rating sections of AD), plug-in and Web 2.0 (tagging sections of AD).

KaitoroCap supports the following: management of ADs, the main medium AK resides in [41]; management of explicit AK in the form of documented AK expressed in natural language or images in ADs; codification of AK and transfer or sharing of a special type of tacit knowledge, namely the knowledge on how people explore AK in ADs. By making available consumers' exploration paths and annotations, KaitoroCap provides alternative support to other consumers, especially novices, in searching related AK.

With reference to a recent SLR on SAKM approaches (which include tools) [24], none of the reviewed SAKM tools capture users' exploration of architectural information in ADs as KaitoroCap does. SAKM tools having features that show similar ideas are: ADDSS - replay design decision tree [31, 42]; visualization of decision chronology [43]; Decision Buddy - solutions ranking and selection by users [44]; ISARCS (Intelligent Software Architecture Rationale Capture System) - collective opinion analysis [45]; DVIA (Design Verbal Interventions Analysis) - analyse transcribed meeting logs to find team members opinions about the design [46]. In comparison, KaitoroCap allows the retracing of users' exploration paths through different sections of a AD. Contents of visited sections are dynamically embedded into a path making the path a restructuring of the AD. It also supports the searching and visualisation of exploration paths. A path supports future navigation to the same collection of sections. KaitoroCap supports collection of ratings, tags, and comments, through dynamic creation of annotation fields.

C. *Comparison with Existing Works on Leveraging Usage Data*

KaitoroCap captures a consumer's (or reader's) interaction history with an AD when he or she is performing an information-seeking task. It shows the interaction history as an exploration path in the form of a tree-view. Visited section contents are embedded automatically into the path, making the path a restructuring of the AD explored.

An exploration path, showing pages and sections visited, sequence of visitation, annotations given (in terms of ratings, tags and comments), can be regarded as a manifestation of the 'wear' (or usage) of a document by AD consumers. By showing other consumers' information traces (i.e. exploration paths, ratings, tags and comments), KaitoroCap supports forms of social navigation. The information traces are dynamically grown, and personalised for specific task.

KaitoroCap bears similarities to two social navigation systems that record exploration paths. Compared to IBM's WBI [9], KaitoroCap records a user's sequence of visiting sections within a wiki page in addition to recording his or her sequence of visiting pages. Footprints [10] aggregates the interaction history of all users in visualizations. This removes the opportunity to see individual paths. KaitoroCap does not combine users' paths. Footprints persists comments immediately but persists other data overnight. KaitoroCap persists all usage data to its database immediately.

Studies on DOI (Mylar/Mylyn, and Tasktop) make use of a user's own usage history. This differs from the other studies reviewed and KaitoroCap. Earlier Mylar versions aggregated interaction data across a developer's workday. Subsequent implementation [18] is similar to KaitoroCap where data collected is task-specific (i.e. scoped by task). In subsequent work on Degree-of-Knowledge model to identify expertise in or familiarity with source code, the aggregation of interaction data again involves all of a developer's work [20]. Different from KaitoroCap, Tasktop [21, 22] does not track interactions with sections in documents or web pages.

DOI work (Mylyn/Mylar and to a lesser extent Tasktop) and wear-based filtering by Deline et al. [14, 16], focus on source code elements, i.e. structured data with explicit relationships. KaitoroCap focuses on semi-structured ADs.

VI. CONCLUSION

Research in several areas has demonstrated the benefits of leveraging usage data of software artefacts. This led us to propose including usage data capture and visualization as essential features of SAKM tools. Automated capture of usage data can serve as a valuable unbiased dataset for empirical research in AK usage. Visualization enables SAKM tools to support a new type of AK sharing, sharing usage knowledge among team members. We outlined the desirable functionalities and qualities of these features and the challenges in leveraging usage data of SA artefacts. We discussed KaitoroCap, as an exemplar. KaitoroCap captures interaction data (manifested as exploration paths) and annotation data (ratings, tags, comments, etc). Exploration paths serve as usage 'wear' left by previous consumers. Visible usage data (e.g. exploration paths, ratings, tags and comments) serve as information traces supporting forms of social navigation. We are exploring how leveraging further SA artefact usage data could provide insights to improve SAKM tools and the development of a usage data capture and visualization framework as a part of a community-wide infrastructure for architecture-based software engineering.

ACKNOWLEDGMENT

We acknowledge support by: University of Malaya and Ministry of Higher Education, Malaysia, via sponsorship of first author's study; PReSS (No. 3624232), University of Auckland; and Software Process and Product Improvement for New Zealand Software Industry (UOAX0710).

REFERENCES

[1] [W.C. Hill, J.D. Hollan, D. Wroblewski, and T. McCandless, "Edit wear and read wear", Proc. SIGCHI Conference on Human Factors in Computing Systems, ACM, 1992, pp. 3-9, doi:10.1145/142750.142751.

[2] W.C. Hill, and J.D. Hollan, " History-enriched digital objects: Prototypes and policy issues", The Inf. Society, vol. 10, no. 2, 1994, pp. 139-145.

[3] P. Dourish, and M. Chalmers, "Running out of space: Models of information navigation", Proc. Human Computer Interaction '94, ACM Press, 1994.

[4] A. Dieberger, P. Dourish, K. Höök, P. Resnick, and A. Wexelblat, "Social navigation: techniques for building more usable systems", interactions, vol. 7, no. 6, 2000, pp. 36-45, doi:10.1145/352580.352587.

[5] A.J. Munro, K. Hook, and D. Benyon, “Footprints in the snow”, in Social navigation of information space, A.J. Munro, K. Hook, and D. Benyon, Eds. Springer, 1999, pp. 1-14.

[6] J.B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, “Collaborative filtering recommender systems”, in The adaptive web, P. Brusilovsky, A. Kobsa, and W. Nejdl, Eds. Springer-Verlag, 2007, pp. 291-324.

[7] M. Svensson, K. Höök, J. Laaksolahti, and A. Waern, “Social navigation of food recipes”, Proc. SIGCHI Conference on Human Factors in Computing Systems, ACM, 2001, pp. 341-348.

[8] A. Dieberger, and M. Guzdial, “CoWeb - experiences with collaborative web spaces”, in From Usenet to CoWebs, C. Lueg, and D. Fisher, Eds. Springer, 2003, pp. 155-166.

[9] P. Maglio, and R. Barrett, “Intermediaries personalize information streams”, Communications of the ACM, vol. 43, no. 8, 2000, pp. 96-101, doi:10.1145/345124.345158.

[10] A. Wexelblat, and P. Maes, “Footprints: history-rich tools for information foraging”, Proc. SIGCHI conference on Human Factors in Computing Systems, ACM, 1999, pp. 270-277.

[11] D. Goldberg, D. Nichols, B.M. Oki, and D. Terry, “Using collaborative filtering to weave an information tapestry”, Communications of the ACM, vol. 35, no. 12, 1992, pp. 61-70.

[12] K. Lerman, “Social information processing in news aggregation”, IEEE Internet Computing, vol. 11, no. 6, 2007, pp. 16-28.

[13] SmugMug Inc., “Flickr”, 2019. www.flickr.com.

[14] R. DeLine, A. Khella, M. Czerwinski, and G. Robertson, “Towards understanding programs through wear-based filtering”, Proc. 2005 ACM Symposium on Software Visualization, ACM, 2005, pp. 183-192, doi:10.1145/1056018.1056044.

[15] U. Shardanand, and P. Maes, “Social information filtering: algorithms for automating "word of mouth"”, Proc. SIGCHI Conference on Human Factors in Computing Systems, ACM Press/Addison-Wesley Publishing, 1995, pp. 210-217, doi:10.1145/223904.223931.

[16] R. DeLine, M. Czerwinski, and G. Robertson, “Easing program comprehension by sharing navigation data”, Proc. 2005 IEEE Symposium on Visual Languages and Human-Centric Computing, IEEE, 2005, pp. 241-248, doi:10.1109/VLHCC.2005.32.

[17] M. Kersten, and C.M. Gail, “Mylar: a degree-of-interest model for IDEs”, Proc. 4th International Conference on Aspect-oriented Software Development, ACM, 2005, pp. 159-168.

[18] M. Kersten, and C.M. Gail, “Using task context to improve programmer productivity”, Proc. 14th ACM SIGSOFT International Symposium on Foundations of SE, ACM, 2006, pp. 1-11.

[19] The Eclipse Foundation, “Mylyn”, 2019. http://www.eclipse.org/mylyn/.

[20] F. Thomas, O. Jingwen, C.M. Gail, and M.-H. Emerson, “A degree-of-knowledge model to capture source code familiarity”, Proc. 32nd ACM/IEEE International Conference on Software Engineering, ACM, 2010, pp. 385-394, doi:10.1145/1806799.1806856.

[21] R. Elves, “Tasktop for Eclipse - Get more out of Mylyn”, 2008. https://dzone.com/articles/more-productive-programming-wi.

[22] Tasktop Technologies Inc., “Tasktop Dev for Eclipse”, 2013. http://tasktop.com/eclipse#benefits.

[23] M.A. Babar, I. Gorton, and R. Jeffery, “Capturing and using software architecture knowledge for architecture-based software development”, Proc. Fifth International Conference on Quality Software, IEEE, 2005, pp. 169-176, doi:10.1109/QSIC.2005.17.

[24] R. Weinreich, and I. Groher, “Software architecture knowledge management approaches and their support for knowledge management activities: A systematic literature review”, Inf. Softw. Technol., vol. 80, 2016, pp. 265-286.

[25] Z. Li, P. Liang, and P. Avgeriou, “Application of knowledge-based approaches in software architecture: A systematic mapping study”, Inf. Softw. Technol., vol. 55, no. 5, 2013, pp. 777-794.

[26] R. Capilla, A. Jansen, A. Tang, P. Avgeriou, and M.A. Babar, “10 years of software architecture knowledge management: Practice and future”, J Syst. Softw., vol. 116, 2016, pp. 191-205, doi: 10.1016/j.jss.2015.08.054.

[27] M. Shahin, P. Liang, and M.A. Babar, “A systematic review of software architecture visualization techniques”, J Syst. Softw., vol. 94, 2014, pp. 161-185, doi: 10.1016/j.jss.2014.03.071.

[28] J.F. Hoorn, R. Farenhorst, P. Lago, and H. van Vliet, “The lonesome architect”, J Syst. Softw., vol. 84, no. 9, 2011, pp. 1424-1435.

[29] Sherman S., and U.-S. N., “What Do Software Architects Think They (Should) Do?”, in Advanced Information Systems Engineering Workshops. CAiSE 2014. Lecture Notes in Business Information Processing, Iliadis L., Papazoglou M., and P. K., Eds. Springer, Cham, 2014, pp. 219-225.

[30] S. Sherman, and I. Hadar, “Toward Defining the Role of the Software Architect”, Proc. 2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering, 2015, pp. 71-76, doi:10.1109/CHASE.2015.17.

[31] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, and M.A. Babar, “A comparative study of architecture knowledge management tools”, J Syst. Softw., vol. 83, no. 3, 2010, pp. 352-370.

[32] F. Nickols, “The knowledge in knowledge management”, in The knowledge management yearbook 2000-2001, J.A. Woods, and J. Cortada, Eds. Butterworth-Heinemann, 2000, pp. 12-21.

[33] J.R. Anderson, Cognitive Psychology and its implications, W.H. Freeman and Company, 1995.

[34] M.T. Su, J. Hosking, and J. Grundy, “Capturing architecture documentation navigation trails for content chunking and sharing”, Proc. 2011 9th Working IEEE/IFIP Conference on Software Architecture (WICSA), IEEE, 2011, pp. 256-259.

[35] M.T. Su, J. Hosking, and J. Grundy, “KaitoroCap: A document navigation capture and visualisation tool”, Proc. 2011 9th Working IEEE/IFIP Conference on Software Architecture (WICSA), IEEE, 2011, pp. 359-362, doi:10.1109/wicsa.2011.58.

[36] M.T. Su, J. Hosking, J. Grundy, and E. Tempero, “Usage-based chunking of Software Architecture information to assist information finding”, J Syst. Softw., vol. 122, 2016, pp. 215-238.

[37] M.T. Su, “Supporting Information Searching in Software Architecture Documents (PhD Thesis)”, Department of Computer Science, University of Auckland, Auckland, New Zealand, 2014.

[38] Atlassian, “Atlassian Confluence”, 2018. https://www.atlassian.com/software/confluence.

[39] The jQuery Foundation, “JQuery”, 2019. http://jquery.com/.

[40] P. Liang, and P. Avgeriou, “Tools and technologies for architecture knowledge management”, in Software Architecture knowledge management, M.A. Babar, T. Dingsøyr, P. Lago, and H. van Vliet, Eds. Springer, 2009, pp. 91-111.

[41] A. Jansen, P. Avgeriou, and J.S. van der Ven, “Enriching software architecture documentation”, J Syst. Softw., vol. 82, no. 8, 2009, pp. 1232-1248, doi:10.1016/j.jss.2009.04.052.

[42] R. Capilla, F. Nava, S. Pérez, and J.C. Dueñas, “A web-based tool for managing architectural design decisions”, ACM SIGSOFT Software Engineering Notes, vol. 31, no. 5, 2006, pp. 4.

[43] L. Lee, and P. Kruchten, “Visualizing Software Architectural Design Decisions”, Proc. 2nd European conference on Software Architecture, Springer-Verlag, 2008, pp. 359-362.

[44] S. Gerdes, M. Soliman, and M. Riebisch, “Decision buddy: tool support for constraint-based design decisions during system evolution”, Proc. 2015 1st International Workshop on Future of Software Architecture Design Assistants (FoSADA), 2015, pp. 1-6.

[45] N. Chanda, and X.F. Liu, “Intelligent analysis of software architecture rationale for collaborative software design”, Proc. 2015 International Conference on Collaboration Technologies and Systems (CTS), 2015, pp. 287-294, doi:10.1109/CTS.2015.7210436.

[46] G. Pedraza-Garcia, H. Astudillo, and D. Correal, “Analysis of design meetings for understanding software architecture decisions”, Proc. 2014 XL Latin American Computing Conference, 2014, pp. 1-10.