Semi-Automatic, Inline and Collaborative Web Page Code Curations

Rutishauser, Roy Adrian ; Meyer, André N ; Holmes, Reid ; Fritz, Thomas

Abstract: Software developers spend about a quarter of their workday using the web to fulfill various information needs. Searching for relevant information online can be time-consuming, yet acquired information is rarely systematically persisted for later reference. In this work, we introduce SALI, an approach for semi-automated linking web pages to source code locations inline with the source code. SALI helps developers naturally capture high-quality, explicit links between web pages and specific source code locations by suggesting links for curation within the IDE. Through two laboratory studies, we examined the developer's ability to both curate and consume links between web pages and specific source code locations while performing software development tasks. The studies were performed with 20 subjects working on realistic software change tasks from widely-used open-source projects. Results showed that developers continuously and concisely curate web pages at meaningful locations in the code with little effort. Additionally, we showed that other developers could use these curations while performing new and different change tasks to speed up relevant information gathering within unfamiliar codebases by a factor of 2.4.

Semi-Automatic, Inline and Collaborative Web Page Code Curations

Roy Rutishauser
Department of Informatics
University of Zurich
Zurich, Switzerland
rutis@ifi.uzh.ch

André N. Meyer
Department of Informatics
University of Zurich
Zurich, Switzerland
ameyer@ifi.uzh.ch

Reid Holmes
Department of Computer Science
University of British Columbia
Vancouver, Canada
rtholmes@cs.ubc.ca

Thomas Fritz
Department of Informatics
University of Zurich
Zurich, Switzerland
fritz@ifi.uzh.ch

***Abstract*—Software developers spend about a quarter of their workday using the web to fulfill various information needs. Searching for relevant information online can be time-consuming, yet acquired information is rarely systematically persisted for later reference. In this work, we introduce SALI, an approach for semi-automated inline linking of web pages to source code locations. SALI helps developers naturally capture high-quality, explicit links between web pages and specific source code locations by recommending links for curation within the IDE. Through two laboratory studies, we examined the developer's ability to both curate and consume links between web pages and specific source code locations while performing software development tasks. The studies were performed with 20 subjects working on realistic software change tasks from widely-used open-source projects. Results show that developers continuously and concisely curate web pages at meaningful locations in the code with little effort. Additionally, we found that other developers could use these curations while performing new and different change tasks to speed up relevant information gathering within unfamiliar codebases by a factor of 2.4.**

***Index Terms*—Semi-automated link curation, knowledge management, web browsing, collaboration**

I. Introduction

Software engineering is a knowledge-intensive profession. While *"source code is king"* [1], developers often need additional information to understand and maintain software [2], [3]. The internet has become an essential source for fulfilling information needs [4], [5], and browsing the web takes a large part of developers' time [6], [7], [8]. Unfortunately, information acquired from web pages is rarely systematically persisted, maintained, and shared with collaborators [8], [9], [10], [11], [12]. This means that the *implicit* links between source code and any relevant web pages, such as API documentation, tutorials, or code snippets, are usually lost, which can negatively impact productivity [2], [13]. Recovering this important implicit knowledge from collaborators has been shown to require *"great effort"* [2].

While links can be persisted as comments within source code, they can quickly clutter the code. Developers must also both remember save the links they used, and must do so manually, inhibiting this form of persistence from being consistently applied [10], [11]. Several approaches have been devised to increase the retention of these links, either through fully-automated or manual techniques. *Fully-automated* approaches (e.g., [14], [15]) try to detect implicit links, usually by leveraging developers' "expertise" based on their interaction history, but unfortunately suffer from many false positives. In contrast, *manual* approaches (e.g., [6]) do not suffer from false positives but offer little support for easing the collection and curation of web pages. Crucially, prior research has not examined the costs associated with curating the links between web pages and source code, nor has it investigated the value of having access to these links.

To address these shortcomings, we have created SALI, a **S**emi-**A**utomated approach for **L**inking web pages to source code locations **I**nline. SALI is semi-automated in that it monitors developers' web page and source code interactions, estimates the relevance of the web pages they have visited to the source code they are working on, and provides link recommendations between these web pages and specific source code locations inside the IDE. These recommended links are not automatically preserved: the developer has to explicitly choose to include them if they consider them "relevant". We refer to this inclusion as *curation*. Once curated, these links are automatically available to all developers on the same project.

Our work examines the two most salient aspects of preserving such implicit links in a collaborative setting:

RQ1: Can our approach help developers identify and curate relevant implicit links between web pages and specific source code locations?

RQ2: Can developers successfully leverage previously-curated links on their own change tasks?

We performed two consecutive laboratory studies to examine these research questions. In the first study, ten developers worked on two realistic change tasks and were asked to curate links using our approach. In the second study, ten different developers worked on two follow-up tasks, one task with developer-curated links from the first study and one without. The first study showed that developers continuously curated relevant web pages with little effort with our approach. The second study demonstrated that these curations could reduce the time required to locate relevant starting points for other tasks.

This paper makes three contributions:

- First, we introduce SALI, a semi-automated approach for

curating links between web pages and source code to foster knowledge exchange within a development team.
- Second, a study demonstrating when and how developers curate web pages to specific locations in the source code, the links they choose to curate, and the effort associated with link curation using SALI.
- Third, a follow-up study demonstrating the utility of developer-curated web page links for collaborators working on future change tasks.

Together, these two studies provide the first investigation into both the curation and consumption of web page links, showing that developers are able to curate links and that these links can be used to improve software development. The supplementary material, including the source code for SALI, is available online [16].

II. Related Work

While completing their daily tasks, developers need to keep track of a broad set of information from several different sources [3], [17], [18]. These information needs comprise questions ranging from understanding what coworkers have been working on, to what code caused a specific program state, to how resources developers depend on might have changed. Fritz and Murphy [17] identified a total of 78 questions that developers frequently ask. Answers to these questions are often dispersed across different artifact types and independent data repositories, including source code, changesets, work items, and web pages, making rediscovery tedious [17]. Another challenge in fulfilling developers' information needs is caused by information that was never explicitly recorded, making it much more time-consuming to recover [2]. While completing tasks, developers frequently encounter problems that require understanding the context of prior code changes; when this context cannot be directly acquired, developers often have to defer completing their task until more data can be gathered [3].

A. Information Artifact Linking

A variety of prior approaches aimed to support developers by linking different types of artifacts to ease information access. These approaches identified links between emails [19], conversations [20], or a combination of multiple information types [21], [17], including source code, bugs, and work items, based on structural relationships or textual similarity [22], [23]. Hipikat [21] is a recommendation system that can infer links between many different types of artifacts and present the results within the IDE. Codebook [24] builds up a graph inspired by social networks, linking many artifacts together and exposing links underlying the social aspects of software development. This graph helps developers to keep track of task dependencies, connections to other teams, artifacts, project histories, and rationale. Similar to these approaches, SALI captures cross-cutting user activity (in the browser and the IDE) and ultimately supports rediscovery. In contrast, our approach leverages the expertise of the developer to curate relevant web pages and link these to meaningful and precise locations in the source code. Finally, TagSEA [25] is a collaborative and lightweight source code annotation tool that enhances navigation, coordination, and capture of knowledge. To improve source code orientation, TagSEA uses inline highlighting of the manually placed annotations. SALI repurposes this concept of user-defined, inline, and visual landmarks but aims to reduce manual effort by semi-automatically capturing external knowledge at specific locations in codebases.

B. Integrating Web Pages

The web plays an increasingly important role in supporting software developers' information needs [4], [26], [5]. Q&A web pages, such as StackOverflow, are prominent in this space, allowing developers to exchange knowledge and discuss coding problems. Overall, developers were shown to spend almost one-quarter of their work time on the web [27], [7], often using the web to answer development-related questions [6], [8].

Due to the prevalence of the web in developers' workflows, various studies have explored how developers use the web to fulfill their information needs [28], [29], [27], [30], [28], [31]. Unfortunately, the vast amount of information available on the web comes at a cost: researchers found that code-related online searches are more time-consuming than regular searches [29], that they are often less successful for novice developers [32], and that it can be challenging to ask the right questions [33], [31]. Researchers addressed this problem with approaches that ease web search and information retrieval by integrating web context into the IDE through search [34], source code examples [35], or StackOverflow posts [36], [37]. While these approaches focus on surfacing novel information from the web to developers in the browser or the IDE, SALI focuses on the rediscovery of previously visited web pages to curate implicit links between relevant web pages and source code for sharing with collaborators.

Most closely related to SALI are Codetrail [6], Reverb [15], and HyperSource [14]. These approaches support users in capturing web context and making it accessible within the IDE to ease information rediscovery without cluttering the source code. Reverb automatically recommends relevant previously visited web pages to developers by extracting and matching code elements from the active editor viewport to compare against previous browsing history. SALI differs from Reverb in that it offers an inline display of curated web pages without modifying the source code, focuses on collaborative use cases, and provides the user agency over the displayed selection of web pages through curation. Codetrail connects the IDE with the web browser to detect web pages containing Java API documentation and automatically links the documentation in the source code for personal or collaborative use. In contrast, SALI supports developers in retaining relevant information with a language-agnostic and semi-automated approach to linking web pages at low granularity levels (lines or keywords). Finally, Hypersource is an extension for IDEs that associates a developer's browsing history with source code edits. Hypersource is most closely related to our approach; however, the semi-automated curation feature of SALI leaves the agency of deciding about the most relevant web pages and

specific location in the source code with the developer, reducing irrelevant web pages at the wrong source code locations. Moreover, while [14] shows that artificially selected links presented through Hypersource can be useful to developers for code understanding, we aim to go one step further by closing the loop between the creation and consumption of link curations by using real developer-curated links of our first study in our second study.

In summary, the approach presented in this work balances the lower effort associated with automated linking with the higher quality associated with manual curation approaches. SALI provides developers with agency over which resources are saved and shared, offering a low-friction and flexible option to exchange crucial implicit knowledge collaboratively.

III. Approach

To help developers capture relevant links between source code locations and web pages, we developed a *semi-automated* approach called SALI.

A. Concepts

SALI is based on three core concepts: semi-automated recommendations, inline curations, and collaborative sharing.

Semi-automated. SALI monitors developers as they navigate web pages and interact with source code. Relevant links between these two types of artifacts are automatically determined based on temporal and lexical information, and these are presented as recommendations to the developer within the source code and the IDE. The developer can then "curate" these recommended links to web pages with a single click to associate them with a specific source code location. While this curation requires the developer to manually select links, the automated recommendations ease the selection, and the active curation by the developer ensures that only relevant, high-quality links are being curated.

Inline. SALI's goal is to capture curations—curated links between web pages and relevant locations in the source code—and to render them subtly without cluttering the source code. By visualizing curations inline within the IDE (see Figure 1), SALI makes previously implicit links to web pages explicit and accessible within the source code with a minimally invasive mechanism [38]. While SALI overlays the source code with curated links in the IDE, the link itself is stored separately from the code. This separation avoids cluttering the code or affecting users that do not use SALI and allows the addition of further metadata, such as the creation date.

Collaborative. As curations capture information relevant to the source code, the linked resources may be valuable to other developers working on the same code. SALI supports sharing relevant links by automatically synchronizing curations using the project's version control system. Other developers working on the same project can see and take advantage of these curations. Only links a developer has actively curated are shared to avoid leaking private information, to give the developer agency over the information shared, and to ensure that only high-quality links are persisted.

B. Prototype

SALI is implemented as a Visual Studio Code (VS Code) extension with four main components that are described below:

Interaction monitor. SALI monitors changes in source code files and interactions with web pages in the browser and extracts lexical and temporal features from these interactions. For *source code files*, SALI captures the content and line number of any line modified by a developer. The line content consists of the tokens delimited by whitespace or punctuation, stored as a bag-of-words (BoW). For *web pages*, SALI captures each visit to a web page within 5 minutes before and after a developer accesses a modified source code file using standard browser extension APIs[1], without violating any security restrictions. We chose this window as prior work has shown that a 5-minute window is a good proxy for capturing development-related activity [6]. For each captured web page, SALI fetches the page title and content by retrieving the text contents within the body of each HTML document (except for 'script' tags) and then extracts tokens delimited by whitespace or punctuation using a BoW representation. Finally, SALI keeps track of the web page access frequency, recency, and duration.

In an additional processing step, both source code and web page BoWs are filtered for common stop words (for details see [16]). Since programming languages often use underscores or camel case in identifier naming conventions, we refrained from applying further processing of the tokens. This design decision retains the lexical integrity of these identifiers and promotes exact matches between the web page and source code content. The prototype also does not filter programming language-specific keywords to offer language-agnostic curations and to support curations in any file. All monitored data is only logged locally on the user's machine to protect the user's privacy.

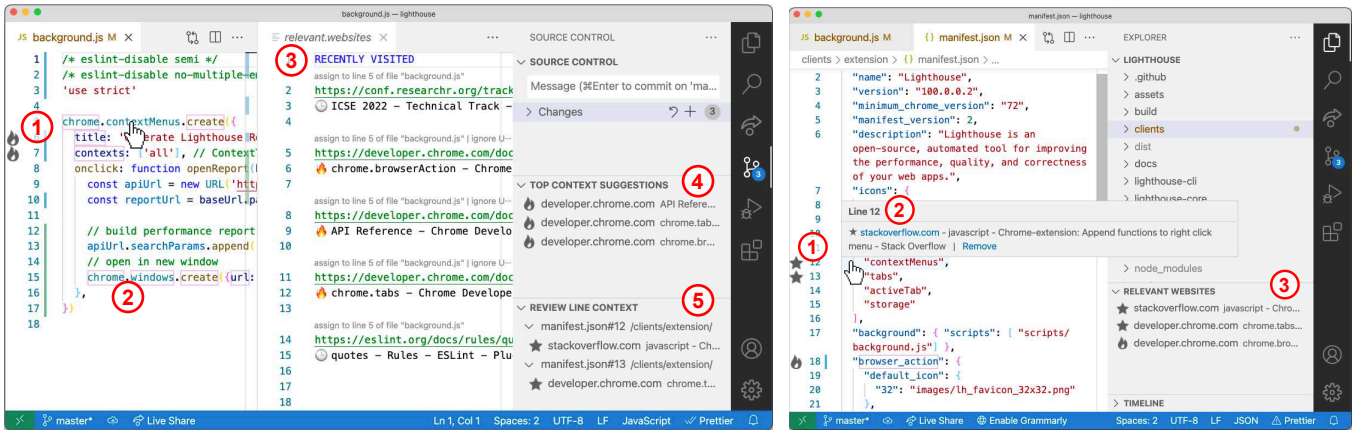
Link recommender. SALI generates link recommendations every 60 seconds on two granularity levels: *keywords* and code *lines*. These levels enable capturing generic line-level concepts as well as more specific keyword-level concepts while also preventing developers from going through long lists of curations that might accumulate on the method or class level.

To identify link recommendations, SALI extracts the token(s) from lines changed by a developer in the code and calculates the *similarity* to the tokens extracted from each web page the developer interacted with within the 5-minute window around a source code modification (5 mins before and 5 mins after). For source code keywords, SALI measures similarity based on the widely-used tf-idf [39] equation:

$$tfidf_{t,d} = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} * (log\frac{N}{n_t} + 1)$$

where $f_{t,d}$ denotes the frequency of the keyword t in the web page d, $\sum_{t' \in d} f_{t',d}$ denotes the total number of terms in the web page d, and $\frac{N}{n_t}$ denotes the fraction of the total number of

[1]https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/tabs



(a) *Curating* web pages

(b) *Consuming* curated web pages

Fig. 1: (a1) The flame icon in the gutter indicates a link recommendation for a line; (a2) an inline link recommendation for a keyword recognizable by its purple outline; (a3) temporary visible list of the recently visited web pages opened through the *recent mode*; (a4) highest-scoring link recommendations in the *SCM Panel*; (a5) *review context views* in the *SCM Panel*. (b1) The star icon in the gutter denotes line curations; (b2) by hovering over curated line 12, an inline popup with the linked web page appears; (b3) the *relevant web pages view* lists curated and recommended web pages for the currently opened code file.

unique web pages N a developer visited within the 5 minutes before and after accessing modified source code. divided by n_t, the number of unique web pages containing keyword t.

Since repeated access to a web page indicates high relevance, we further multiply the tf-idf score with the logarithmically scaled number of page visits d within the $2*5$-minute time window.

For source code lines, SALI measures similarity based on the Jaccard index [40], which indicates how well the tokens of a modified code line overlap with a web page's tokens. Similar to the scoring for keywords, SALI multiplies the Jaccard score with the logarithmically scaled number of visits to a web page within the time window. A final post-processing step examines blocks of consecutively modified code lines and discards duplicate link recommendations for all but the most similar code line within the block.

IDE frontend: collecting and presenting links. SALI displays recommended and curated links in two ways: inline with the source code and aggregated in side panel views (see Figure 1). The *inline* visualization integrates the recommended and curated links within the source code editor by highlighting keywords or marking code lines in the gutter (using the flame icon for recommendations and the star for already curated links). For recommendations, hovering over the highlighted keyword or the flame icon opens a popup with up to three recommendations. Clicking on a recommendation confirms it and turns it into a curation. Hovering over a previously-curated link opens a popup with the title and the domain name of the curated web page. Complimentary to highlighting link recommendations, SALI supports developers in manually linking the most recently visited web pages to any keyword or line in the source code via the context menu. Developers can use the *recent mode* to curate a link they want to keep at any location; this eliminates the need to copy and paste the link (see Figure 1a (3)).

Additionally, SALI offers four views within VS Code for managing curations. Three views in the Source Control Management (SCM) Panel support reviewing and revising curated web pages. *Two review views* provide a summary of curated web pages, and *one view* lists the top recommendations for the current code changes that a developer may have missed. Finally, the *relevant web pages view* in the Explorer Panel lists all recommended and curated links relevant to the code in the viewport for (re-)discovery, as inspired by Reverb [15].

Backend: collaborative sharing and persistence. SALI automatically persists, shares, and maintains curated links in the code repository, making them accessible to every developer on the project after they commit their changes and curations. At the same time, our approach protects developers' privacy by discarding recommendations that are not explicitly curated after each commit. To not clutter code files, SALI persists curated links in a metadata file in the root directory of the code repository.

IV. Study Method

We conducted two consecutive laboratory studies to answer our research questions (see Figure 2 for an overview). To investigate the costs associated with curating previously implicit links between web pages and source code locations (RQ1), we performed the *curation study* (Section IV-B) with ten participants that worked on two code change tasks and were instructed to curate links using SALI. To examine the value of link curations (RQ2), we conducted a second study (*consumption study*, Section IV-D) with ten different developers that worked on two follow-up change tasks extending the source code modified in the first study. Each participant worked on one task with curated links (treatment) and one without (control). We randomly selected sets of curated links from the curations collected in the *curation study* and mapped them to the best code solution (reference solution) from the

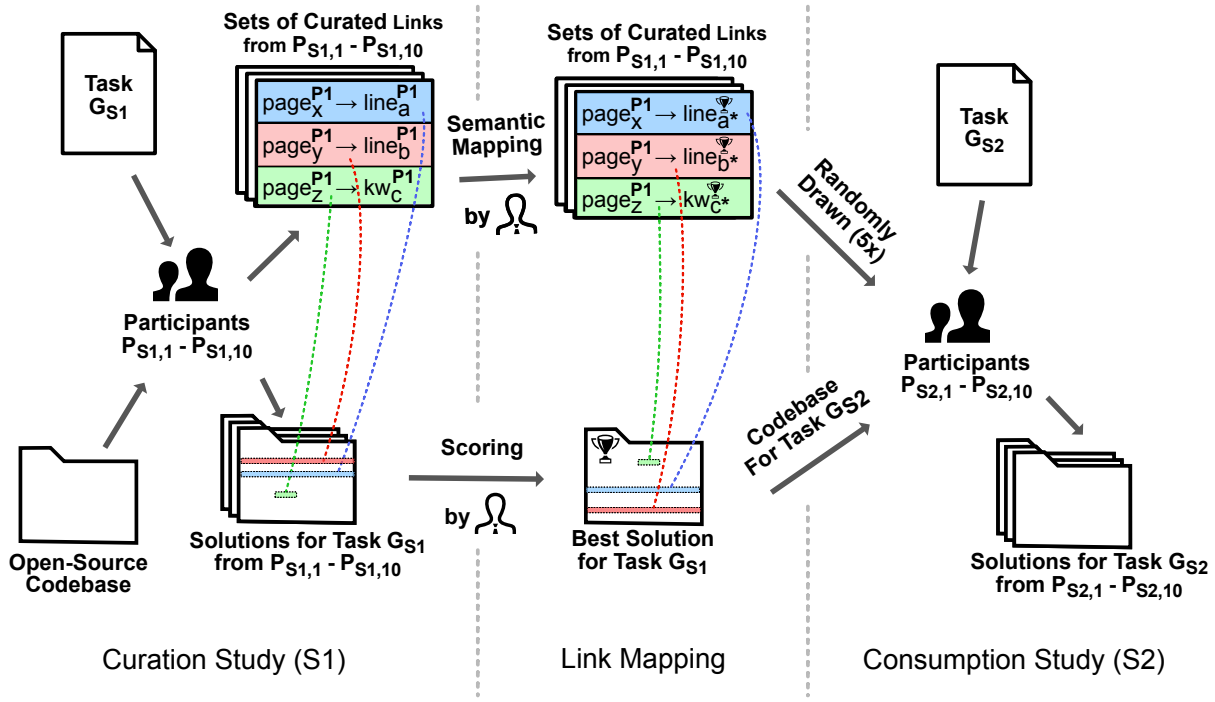


Fig. 2: Design of the overall study method illustrated for the Gifshot tasks G_{S1} and G_{S2} (same applies to the Lighthouse tasks). Participant-curated links from the *curation study* were used as input for the subsequent *consumption study*.

first study to improve comparability between participants (see Section IV-C). Both studies were approved by the institutional ethics board.

A. Change Tasks

We designed four realistic change tasks for two widely-used open-source JavaScript projects. The tasks, listed in Table I, represent useful and realistic additions to the existing systems that involve working with external, well-documented APIs.

The first project, called Gifshot[2], is a library for creating animated GIFs from image streams. The tasks G_{S1} and G_{S2} ask participants to extend Gifshot and extract specific parts of an image (a person or a face) using the *BodyPix* model from the TensorFlow JavaScript library[3]. The tasks are completed when the expected visual effect can be observed.

The second project, called Lighthouse[4], is a Google Chrome extension for improving web page quality and performance. The tasks L_{S1} and L_{S2} ask participants to improve the Lighthouse extension by adding context menu items and actions. The tasks are completed when the expected menu items and actions can be observed.

Task evaluation scoring. To assess the quality of participants' task solutions, we scored each by applying a rubric (similar to [41]). More specifically, we awarded one point for each correctly completed *solution step* and half a point for each step that had syntax errors but was otherwise correct.

[2]https://github.com/yahoo/gifshot
[3]https://github.com/tensorflow/tfjs-models
[4]https://github.com/GoogleChrome/lighthouse

Short Task Description	# Steps
G_{S1}- Add background removal feature to Gifshot's GIF creation to color everything but the person in an image frame black.	4
G_{S2}- Extend the previously added Gifshot background removal feature to only leave the face showing.	4
L_{S1}- Add a "Generate Report" context menu item to generate a Lighthouse report for the current tab in Chrome when clicked.	4
L_{S2}- Modify the context menu to allow reports to be created for either the current tab or all tabs in the current browser window.	3

TABLE I: Simplified task descriptions with number of steps for a solution. Full descriptions and steps can be found at [16].

The maximum score for an executable and complete solution for task L_{S2} was 3 points and 4 points for all other tasks.

B. Study 1: Curation Study

To investigate the costs and experience associated with link curation using our approach and to collect authentic links for the subsequent *consumption study*, we conducted a laboratory study in which developers performed G_{S1} and L_{S1} and were asked to curate links as they worked.

Procedure. The study was conducted in person on participants' personal computers. We provided each participant with a detailed explanation of SALI, which included a short video introduction and a 10-minute hands-on tutorial. We then asked each participant to work on the two tasks G_{S1} and L_{S1}. For each task, participants received the change task description and instructions on how to build and run the open-source project, along with a starting point (specific method) from

which they could begin their task. We provided the starting point to reduce the initial navigation time in the unfamiliar codebase, similar to prior work [31]. Our guideline was that 40 minutes should be sufficient for each task, although this was not strictly enforced, and participants could stop when they felt they were done or were stuck. After participants finished their task work, we informed them how to access the logged data for inspection and asked them to upload it with their solutions to a secured university server. After participants submitted their data, we conducted a 20-minute semi-structured interview. In the interviews, we asked participants about their experience working on the tasks and with SALI, their current practices for documenting relevant web pages, and demographic and background questions. In the end, every participant received a USD 30 gift card. The complete materials can be found in the supplementary material [16].

Pilot. We conducted a pilot study with four participants to learn about any experimental issues. Two participants felt time pressured and neglected to curate any links. One participant found our instructions too terse. To address these issues, we turned the 30-minute time limit into a 40-minute recommendation and improved the pre-study training.

Participants. We recruited 10 participants aged between 24 and 37, with an average age of 29.5 years ($\pm$4.4). We used snowball sampling from our personal and professional networks for participant recruitment. Nine participants identified as male, and one chose not to reveal their gender. Their job titles included software engineer (4), Ph.D. student (3), information security specialist (1), solution architect (1), and software engineer/student (1). Participants had up to 10 years of professional software engineering experience, with an average of 4.1 years ($\pm$3.1). Only one participant did not have any professional software engineering experience. Their self-assessed JavaScript experiences on a 5-point ordinal scale (1: none, 5: expert) were 2 (4 participants), 4 (4), and 5 (2). Half of the participants used VS Code weekly (4 participants) or daily (1). The other half had little experience with VS Code since they use it only on a monthly/yearly basis (3) or had never used it before (2). From the 10 study participants, we discarded the log data of one participant ($P_{S1,9}$) as there was an error during the transmission that could not be recovered.

Data collection and analysis. For the interviews, we recorded and transcribed the audio recordings. We then coded the transcripts using a semantic and inductive bottom-up approach. We performed a thematic analysis to qualitatively analyze the transcripts by following the six steps defined by the framework: familiarization, coding, generating themes, reviewing themes, defining and naming themes, and write-up [42]. To reduce observer bias, the first two interviews were open-coded by two authors of the paper, and a common set of codes was agreed upon. Subsequently, one author iteratively coded the remaining interviews. For themes, two external researchers, a post-graduate student with no prior involvement in the project and a graduate student, both familiar with thematic analysis, generated themes based on the codes obtained in the second step. We concluded the open coding by reviewing and defining these themes. We present each theme in its own paragraph in Section V-A.

To examine participants' behavior, SALI automatically logged the URLs and access times of all visited web pages, the source code files participants opened (focus events) or modified, and interactions with SALI (link curations, link removals, used curation modes, and interaction sources).

Finally, we collected the source code repositories comprising participants' solutions and curations for the change tasks, and we scored the solutions as described in Section IV-A.

C. Curated Link Mapping

As the goal of the *consumption study* was to build on the participant-generated curations from the *curation study*, we first selected a reference solution for the *consumption study* participants and then mapped the curations from the *curation study* participants onto that solution. We chose one solution per task so that all participants start at the same point for a fair comparison of the treatment groups. The mapping was required because the participant-generated curations were on their own code solutions, each of which was unique.

Using the scoring rubric (Section IV-A), the average score in the *curation study* was 3.2 out of 4 ($\pm$0.7) for G_{S1} and 2.9 out of 4 ($\pm$0.8) for L_{S1}. $P_{S1,3}$ created the best solution for G_{S1}. $P_{S1,2}$ created the best solution for L_{S1}. We used these reference solutions for all follow-up tasks.

We mapped the curated links from all participants who scored over 50% on G_{S1} and L_{S1} (16 of the 20 solutions met this threshold). For each participant-curated link on those 16 solutions, we mapped the curation to the semantically identical location in the reference solution. 57 of the 61 curations could be directly mapped; we discarded the remaining 4 since they corresponded to code locations that were not part of the reference solution. Each of the 10 participants in the *consumption study* was given one *curation study* participant's set of mapped curations at random; no curation set was reused. By assigning each participant in the second study a set of curations from *one* random participant in the first study, we try to account for the variation in the curations (and their quality) that different developers make.

D. Study 2: Consumption Study

To investigate the benefits of having curated links while performing a change task, we conducted a laboratory study where developers performed G_{S2} and L_{S2} using curated links from the *curation study*.

Procedure. The overall procedure of the *consumption study* was similar to the one of the *curation study*, including the detailed explanation of SALI and participants being asked to work on two tasks (G_{S2} and L_{S2} in this case). Unlike the *curation study*, participants were asked to finish each task in a maximum of 40 minutes to compare their performances. After the participants finished their two tasks, we concluded the session with a 20-minute semi-structured interview. The interviews featured the same demographic questions as the *curation study* and asked additional details about how they

used the web and SALI as they completed their tasks. In the end, every participant received a USD 30 gift card.

We used a within-subjects design for the *consumption study*, with each developer performing one of the two tasks with access to a set of curated links created by a prior random participant from the *curation study* (treatment condition) and one without (control condition). The control condition without additional web pages is a typical and realistic scenario developers face in practice every day [2], [3]. We assigned the tasks and the conditions to participants in a counterbalancing order to mitigate learning effects: half of the participants solved their first task with curations (treatment condition) from the prior study before attempting to solve the second task without curations (control condition); the other half performed their tasks with the condition assigned in the opposite order.

Pilot. We conducted a pilot with 10 participants, revealing that the follow-up tasks were too complex to complete in the initially planned 30 minutes. As a result, we increased the amount of time allocated for both tasks to 40 minutes each, simplified L_{S2}, and gave clearer and more thorough instructions for G_{S2}. We further disabled SALI's curation features for the *consumption study* since pilot participants started curating links without being asked, interfering with the objective of RQ2.

Participants. We recruited 10 new participants with the same means as in the *curation study*. The participants were between 20 and 42 years of age, with an average of 27.6 years ($\pm$6.4). Three participants identified as female and seven as male. Their job titles included software developer (3 participants), computer science student (3), part-time software developer/student (2), Ph.D. student (1), and information security specialist (1). Participants had between 0 and 21 years of professional software engineering experience, with an average of 3.7 years ($\pm$6.3). Their self-assessed JavaScript experiences on a 5-point ordinal scale (1: none, 5: expert) were 1 (1 participant), 2 (1), 3 (3), 4 (4), and 5 (1). Most participants use VS Code on a daily (6) or weekly (1 participant) basis, while the remaining participants use VS Code on a monthly (2) basis or had no experience (1) with the IDE. From the 10 study participants, two participants ($P_{S2,4}$ and $P_{S2,8}$) did not notice the curated links in the treatment condition with access to developer-curated links. Without noticing curations, their participation could have been counted as part of the treatment and/or control group. To avoid ambiguity, we, therefore, excluded their quantitative data and only report on their semi-structured interview responses.

Data collection and analysis. To evaluate participants' task performance, we scored the submitted solutions and recorded the time of task completion (max. 40 minutes). To examine participants' behavior, SALI logged their interactions with it, focusing on which curated links they opened, from which location in the source code or which aggregated view they accessed, and at what time during the task. SALI further logged general browser usage, search engine use, and web page access. For the interview data, we used the same methodology as for the one in the *curation study*.

V. Results

This section presents the analysis from the two studies with respect to the link curation process (RQ1) and the value of previously curated links to future collaborators (RQ2).

A. RQ1: Web Page Curation

In the *curation study*, participants were able to use our approach to continuously and concisely curate links between web pages and code. Participants acknowledged the perceived value of these curations and experienced little overhead from the curation process.

Developers curated links actively and continuously. Participants curated between 1 and 9 links per change task, with an average of 4.0 links ($\pm$2.2) per task. Most participants curated continuously throughout each task, as illustrated in Figure 3. On average, participants curated a link every 5.2 minutes ($\pm$6.0). While there is a slight tendency to curate more towards the end of a task, most participants (9 out of 10) explicitly stated that they curated links throughout their programming work, specifically so they would not forget the relevant web pages. For example, one participant stated:

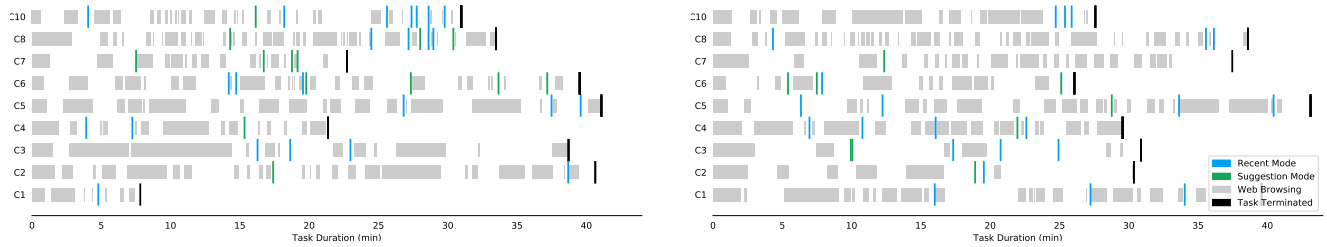
> "*Developers might lose context super fast. [...] So, for me, it's always better to do it [i.e., curating links] on the fly, and perhaps I could review it later.*" - $P_{S1,5}$

Developers curated consciously and concisely. During the tasks, participants spent an average of 35.2% ($\pm$6.1%) of the total task time using the web browser, visiting an average of 26.3 ($\pm$14.9) unique web pages. Of these web pages, participants only linked 2.5 ($\pm$1.2) unique web pages at various locations in the code, with an average of one curated link per 4.5 lines of written code. Some of these web pages were curated multiple times at multiple locations throughout the codebase. Several participants wanted to be even more precise, stating that they would prefer to link to specific sections within a web page, rather than the full page. Reflecting this, 25% of curated web pages included an anchor in the URL referring to a specific location on the web page.

In addition to being highly selective in filtering the web pages they visited for curation, participants further expressed the conscious and deliberate nature of the link curation process during the post-task interviews. They commonly stated that when they curated links, they considered how other developers might use a given curation, or even that they felt a responsibility towards their future peers:

> "*I guess I was trying to keep in mind if it was somebody else who was going to be using it–would this be a useful link to have or not in the future?*" - $P_{S1,4}$

Link curations required minimal to no effort and also provided value to curators themselves. A majority of participants (7 out of 10) explicitly stated that SALI was easy to use and intuitive for curating web pages, and three participants liked that the approach did not "bloat" the source code with code comments for persisting the curations. Only



(a) Task L_{S1} (b) Task G_{S1}

Fig. 3: Web browsing (gray) and curating links (green and blue) over time, per task and participant in the *curation study*.

four participants explicitly mentioned that the curation process required additional effort, but that it did not necessarily slow them down and was still perceived as valuable.

> "*There's no way I would have been faster [without curating]. I wouldn't say that. Is it extra work in the mental sense because you still have to think about it? It certainly is. It's like a bit of reporting. It's very low effort, but something you have to have in the back of your mind.*" - $P_{S1,1}$

Other participants stated that curating links did not require additional effort as it decreased their overall work:

> "*I am someone who documents as well as possible for others. Therefore, this actually corresponds to my normal workflow.*" - $P_{S1,2}$

In general, all participants considered curating links with SALI a valuable activity, providing benefits for themselves and others. These benefits ranged from keeping track of important web pages, to fostering better and more frequent documentation of the source code, through to reflecting on their programming work. Curating links also eliminated fear of forgetting relevant web pages.

> "*Partly, curating is helpful for me, too. It's basically a process to reflect on how my code came to exist.*" - $P_{S1,10}$

> "*I guess as soon as I saw it, I'm like, aha! I found the thing I need for the task. So I guess it was kind of a bookkeeping thing that I don't want to lose this cause I found it.*" - $P_{S1,4}$

Developers generally curated similar or identical web pages for a task. In total, the 10 *curation study* participants made 72 link curations in various code locations, linking a set of 24 unique web pages (11 for G_{S1}, 13 for L_{S1}). Of the 72 curations, 93.1% referred to external APIs relevant for the study task completion. The remaining 5 curations comprised information on the web platform itself, predominantly (4 of 5) on the Canvas API[5], which was used to render webcam outputs for G_{S1} and which many participants found challenging to use. The most commonly curated kind of web page was official documentation (58.3%), followed by Q&A posts (27.8%), and tutorials/blog posts (9.7%).

[5]https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

Considering the links participants curated for both tasks, the results further showed that different participants often agreed on which web pages were most relevant for documenting a particular task. While there were 16 unique web pages that were each curated by only one participant, the remaining 8 unique web pages were curated by an average of 3.6 ($\pm$1.9) participants, with some web pages being curated by 6 (L_{S1}) and 7 (G_{S1}) of 10 participants.

Developers linked web pages to semantically meaningful and precise locations in the code. Participants predominantly assigned web pages to semantically related words and lines in the code. For 70.0% of the 20 *keyword-based link curations*, the chosen keywords in the code were specific object or method names from the curated web pages. 25.0% of the curations were assigned to specific variable names.

Of the 52 *line-based link curations*, participants either chose lines that contain a specific API method invocation (40.4%), an import statement (25.0%), an attribute (15.4%), or a method signature (7.7%) that were all semantically related to the curated web page. Only the line contents of 6 (11.5%) curations could not be clearly categorized.

SALI's inline link recommendations in the coding editor were valuable but could be invasive. All but one participant of the *curation study* curated some of the automatically presented recommendations, with a total of 30.0% curations stemming from inline recommended links (55.6% of the keyword curations, 19.0% of the line curations). Participants stated that the visual cues and the immediate nature of link recommendations facilitated the curation process.

> "*I found it good that I immediately received recommendations about which web pages I had visited recently. That definitely makes it easier.*" - $P_{S1,1}$

> "*I think it's quite cool - especially that you get a suggestion when you hover over it: "hey, didn't you want to link this page?"*" - $P_{S1,7}$

At the same time, seven participants mentioned that a high number of recommendations on the keyword level can be quite invasive, especially since the keyword recommendations are highlighted within the code.

> "*So having too many square boxes with keywords might be disruptive, especially for people that lose attention easily, which I'm one of them.*" - $P_{S1,5}$

The main participant-suggested improvements for the inline link recommendations were: (a) improving the accuracy and precision of recommendations, and (b) refining how the recommendations were presented using the inline popups (see Figure 1a). One problem raised was due to the fact that when SALI displayed link recommendations, only the web page title and domain are shown, making it difficult for participants to quickly recognize the web page.

Most curations were based on recently visited web pages. 70.0% of all curations were made in the *recent mode*, assigning a recently visited web page from the list presented by SALI to a specific location in the code. This high number of curations demonstrates that even this simple mode—which is solely based on tracking a user's recent interaction with web pages—has value and supports successful curation. An analysis of the participants' browsing histories leading up to link curations further showed that *recency* (with an Area Under the Curve (AUC) score of 0.91) is a better temporal feature than *visit frequency* (0.66) and *usage time* (0.74) for predicting whether a web page is curated. This insight suggests that updating SALI to more strongly favor *visit recency* could improve recommendation performance.

Proximity of curated features to the actual source code is important. The majority of all user interactions (89.8%) originated from SALI's source code editor inline curation features; side panel views were rarely used. Only one participant opened a single web page through the *relevant web pages view* (depicted in Figure 1b) and no participants interacted with the highest-scoring link recommendations in the *SCM Panel*; the remaining 9.1% of interactions stemmed from the *review context views*. This focus on inline representations close to the developer's field of attention confirms earlier observations [38].

Curation fit into existing workflows. The *curation study* provides relevant first insights about the ability of link curation to fit into developers' workflows. Overall, 8 of 10 participants said they could see themselves using this or a similar approach in a real-world application. Two other participants did not explicitly answer the question, neither denying nor affirming future use. Some participants explicitly stated that SALI would work well within their existing workflows since it integrated into the editor, did not bloat the code, and retained important implicit knowledge.

> "*[SALI] kind of mirrors my normal workflow because whenever I have a site or a website that I copied code from, I usually leave a comment with the link from that website, myself. So the problem with that is just that if you have a bunch of links that it'd get super bloated and having that hidden in an extension...*" - $P_{S1,5}$

Several participants (6) also stated that SALI would be helpful for teams, and one emphasized that it could be valuable for developers new to a project ($P_{S1,4}$).

RQ1 Summary

SALI enables developers to quickly and concisely curate links between web pages and specific source code locations. Curation support should integrate with developer workflows, be minimally invasive, and provide visual cues with precise recommendations.

B. RQ2: Collaborative Curation Value

In the *consumption study*, a different set of participants worked on two follow-up tasks, using developer-curated links from the *curation study*. Overall, participants found the curations helpful for getting to relevant information faster and for completing the tasks.

Curations sped up locating relevant information, even on different tasks. Even though the curated links in the codebase were created for tasks different to the ones in the *consumption study*, most participants (8 of 10) made use of these curated web pages to solve their tasks. Of the 30 previously-curated links mapped into the reference solutions, 21 of them referenced specific locations in files where participants had to perform code changes to complete the task. Of these 21 curations, participants followed a total of 13 curated links, often relatively soon after they started working on the task (on average 3.3 ($\pm$1.7) minutes into the task). 10 of these curations were explicitly considered helpful by participants. In one case, a participant initially decided a curation was not relevant, but they added that it was still a meaningful contextual reference point for them. Other participants explicitly mentioned that the curations were helpful for navigating to task-relevant web pages, even if the curated web pages did not contain the exact information they were looking for. Overall, participants valued the curated links as an initial starting point for better understanding the tasks they were facing:

> "*I think curations give you a starting point, and then you got to explore. So you always learn something from the curations, as long as you get some information that's useful.*" - $P_{S2,7}$

Participants without curations took more time to identify the web page that was most relevant to solving the task according to them: 12.1 ($\pm$4.0) and 8.9 ($\pm$10.8) minutes without curations for G_{S2} and L_{S2}, respectively (overall average of 10.5 $\pm$7.8 minutes), compared to 5.1 ($\pm$3.1) and 3.8 ($\pm$2.4) minutes without curations (overall average of 4.4 $\pm$2.6 minutes). This accounts for a speed-up factor of 2.4 to locate the most relevant web page. An independent, one-tailed Welch's t-test $t = 2.092, p = 0.034$ showed that this factor is significant at the 0.05 level.

Without curations, some developers initially felt "lost". Without curated links, 6 participants reported having trouble at the beginning of their task, often stating that they felt "lost"

until they found relevant information online. One participant even studied the wrong API initially and only stumbled upon the relevant documentation after 25 minutes:

> "*I thought I was supposed to look at the Lighthouse GitHub page [...] and I was going through the documentation, but it was not correct.*" - $P_{S2,1}$

The same participant performed much better when they had access to curations on their next task and stated that the curated link helped them to have a contextual starting point from which it was "pretty easy" to navigate to the relevant web pages.

Curations may have improved task performance. Participants performed better with the links curated by other developers using SALI for task L_{S2} and performed equally well for task G_{S2}. Overall, participants scored an average of 3.5 ($\pm$1.0) out of 4 points for G_{S2} and 1.4 ($\pm$1.0) out of 3 points for L_{S2} without curations. In contrast, participants with access to curated links scored an average of 3.5 ($\pm$0.4) points for G_{S2} and 2.5 ($\pm$0.4) points for L_{S2}. These scores present a 78.6% improvement in completion score for L_{S2}. An independent Welch's t-test $t = 0.986, p = 0.348$ showed no significant differences in the average scores for L_{S2} with and without curations. A comparison in terms of time is not meaningful because the *consumption study* capped the time per task to 40 minutes, and several participants did not complete all solution steps in time.

Specificity and proximity of curated links to source code locations were important. Participants appreciated it when curated links were presented inline in the source code on the granularity level of keywords and lines. A majority (6 of 10) of participants explicitly mentioned that they liked the keyword granularity:

> "*It's nice to actually have the highlighting around the keyword because then you understand what the context is that the URL is provided for.*" - $P_{S2,2}$

Several (4 of 10) participants also stated that they did not like the idea of higher-level granularities (e.g., a curated link for a whole file, module, or class) and that they would consider link curations on that level almost as a "red flag" ($P_{S2,9}$):

> "*The class level gives you a lot of irrelevant information. And it will not pinpoint you accurately to the right documentation [...].*" - $P_{S2,1}$

While participants of the *consumption study* did use the *relevant web pages view* side panel view somewhat more than participants of the *curation study*, it was still sparsely used. Only 2 of 10 participants considered the *relevant web pages view* side panel to be helpful.

Positive feedback on the overall experience. 8 of 10 participants commented positively on the approach, especially within a team setting for onboarding new team members, leaving hints for others, and generally preserving information that would get lost otherwise. Several participants also explicitly mentioned that they did not find the curations distracting, and two participants further mentioned that SALI could help them personally to resume tasks faster and reduce time for finding the relevant website:

> "*If it's something more specific and there's a curation, then I think it's helpful, because otherwise you'd have to google 1000 times and transform the query until you somehow come up with something reasonable.*" - $P_{S2,10}$

RQ2 Summary

Curations can reduce the time required to locate relevant web pages by providing contextual starting points for navigation. Proximity and specificity of curations within the source code increase their utility, even for tasks different from the one they were created for.

VI. Discussion

The presented work provides initial evidence on the benefits of a semi-automated approach for curating links in a collaborative setting and is one of the first to examine both the curation and the consumption of implicit links.

Capturing implicit knowledge. Prior research has investigated manual and fully-automatic methods for capturing implicit links between web resources and source code. Our two studies suggest that the balance of these two approaches taken by SALI allows developers to reduce the overhead of manual approaches without the precision challenges of fully-automated techniques. While prior work has shown that links provided by researchers for a specific task may improve performance (e.g., [43]), the combination of our two studies shows that user-generated links can have similar positive effects on developers even when they work on different tasks.

Link presentation. Our study results indicate the importance of finding the right balance between convenience and invasiveness for presenting link recommendations and curations. For example, two participants of the *consumption study* did not notice the curations in their IDE, suggesting that they should be more prominently displayed. Additionally, good recognizability of previously visited web pages in the IDE was particularly challenging for multiple developers. Providing web page thumbnails could ease resource recognition in these cases and aid web page recall [44], [45].

Value and relevance dependent on seniority and experience. Participants' interview responses and previous research [3] suggest that curated links may be especially valuable for developers new to a codebase. At the same time, the links a senior developer on a project considers relevant and curates might differ from the ones relevant to a newcomer. While our approach encourages developers to curate links relevant to the current task context, we observed reservations (4 of 7 participants) about curating "basic" knowledge:

> "*When you're more familiar with your codebase, you implicitly assess things as common knowledge. So you may*

> *not remember to put the link there because for you it's like "everyone knows that already".*" - $P_{S1,6}$

Future work should investigate how to best support the different experience levels without requiring additional effort.

Application in practice. While participants' feedback on SALI and its use was positive, there are several open challenges regarding long-term usage and curation link maintainability in the field. The current approach already automatically updates line numbers after file modifications. By linking curations to specific lines or keywords, developers that make code changes will only need to re-evaluate the specific curations linked to the changed code. Yet, other changes, such as evolving keywords, code refactorings, or modifications to web page URLs, also have to be addressed but are not unique to our approach. For example, link decay in source code is a general challenge [12]. For link decay, SALI could be extended to automatically check for web pages that are no longer accessible or outdated, indicate this to developers, or even remove these links automatically without having to alter the source code.

Due to the nature of laboratory studies, we can also not provide conclusive evidence of whether developers will continue curating links beyond a controlled setting. However, we believe that our observations, the positive feedback from our participants, and the voluntary curation of several participants in our *pilot consumption study* provide initial evidence for the approach's potential and warrant a longitudinal field study as a next step.

VII. Threats to Validity

Our work has several limitations commonly associated with laboratory studies.

External validity. The studies included 14 professional and 6 students/part-time developers but may not represent all professionals. All participants were unfamiliar with the codebase used for our change tasks, unlike all non-newcomers. Although our four selected tasks were from real codebases, they may not be representative of all development tasks. We designed the tasks to (a) simulate real-world OSS change tasks as they could be found on the projects' roadmaps, (b) allow participants to solve the tasks within the allotted time without deep prior knowledge of the project, and (c) required to gather information from the web. The results might differ for other tasks, especially tasks that are much easier to solve without having to look up any information on the web or tasks that are much harder.

Our work also did not study SALI in a field setting. Adaption, utility, and quality of curations may vary depending on other factors, such as the number of contributors or project maturity. Further research is needed to show how our results generalize in the field with other developers, tasks, and projects.

Internal validity. Other forms of information could have been helpful to participants in the treatment of the *consumption study*, where participants were not shown any curated links. We made this decision since it better reflects the current reality where information from the web is rarely preserved [8], [9], [10], [11], [12]. We chose not to create artificial curation sets (e.g., for explicitly testing the impact of false positives) as we wanted to study the scenario where curators have no clear understanding of future change tasks. Using realistic developer-curated links allowed us to gather insights into whether these links between source code locations and web pages for one task could be useful for future tasks. To study the effect of developer-created curations on change task performance under controlled conditions, the authors mapped the curation locations to a shared codebase for all follow-up tasks; we reduced the impact of this by mapping to semantically identical locations in the target codebase and refraining from filtering or modifying the curated links.

Construct validity. We have yet to establish conclusive evidence that the additional time required for curating links does not outweigh their future value, although, during the *curation study*, participants mentioned that curating did not impact their *perceived* productivity.

VIII. Conclusion

Web-based resources are commonly used during software development, but the implicit links between web pages and source code are usually lost. This paper introduced and evaluated SALI, a semi-automated, inline approach for curating links between specific source code locations and web pages. SALI was evaluated with real developers to determine both the costs of curating links and the benefits to future developers using those curated links for different tasks. The results show that developers curated web pages continuously and concisely at meaningful source code locations with little effort. The results also show that for developers unfamiliar with a codebase, curated links can speed up locating relevant information for specific parts of the source code by a factor of 2.4. Our study is the first to date that shows both the costs and benefits associated with recording and using web page curations; our findings suggest that the implicit information captured by these valuable links need not be lost and that support for collecting and rendering these links should be a part of standard development practice.

IX. Data Availability

All study instruments, data, and analysis scripts as well as the source code for SALI are available in our replication package https://doi.org/10.5281/zenodo.7622496.

X. Acknowledgements

We thank our study participants for their participation. We also thank the anonymous reviewers for their valuable feedback. This work was funded by SNF under grant 200021_207916.

REFERENCES

[1] J. Singer, "Practices of software maintenance," in *Proceedings of the International Conference on Software Maintenance (ICSM)*, 1998, pp. 139–145.

[2] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: A study of developer work habits," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2006, pp. 492–501.

[3] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2007, pp. 344–353.

[4] C. Treude, O. Barzilay, and M. A. Storey, "How do programmers ask and answer questions on the web? (NIER track)," *Proceedings of the International Conference on Software Engineering (ICSE)*, pp. 804–807, 2011.

[5] H. Li, Z. Xing, X. Peng, and W. Zhao, "What help do developers seek, when and how?" *Proceedings of the Working Conference on Reverse Engineering (WCRE)*, pp. 142–151, 2013.

[6] M. Goldman and R. C. Miller, "Codetrail: Connecting source code and web resources," in *Proceedings of the Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2008, pp. 65–72.

[7] A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz, "The work life of developers: Activities, switches and perceived productivity," *IEEE Transactions on Software Engineering*, vol. 43, no. 12, pp. 1178–1193, 2017.

[8] A. Grzywaczewski, R. Iqbal, A. James, and J. Halloran, "Software developers' information needs: Towards the development of intelligent recommender systems," in *Proceedings of the International Conference on Ubiquitous and Collaborative Computing (UBICOM)*. Swindon, GBR: BCS Learning & Development Ltd., 2011, pp. 66–74.

[9] F. Zampetti, L. Ponzanelli, G. Bavota, A. Mocci, M. Di Penta, and M. Lanza, "How developers document pull requests with external references," in *Proceedings of the International Conference on Program Comprehension (ICPC)*, 2017, pp. 23–33.

[10] J. Jiang, J. Cao, and L. Zhang, "An empirical study of link sharing in review comments," in *Software Engineering and Methodology for Emerging Domains*, 2017, pp. 101–114.

[11] S. Baltes and S. Diehl, "Usage and attribution of Stack Overflow code snippets in GitHub projects," *Empirical Softw. Engg.*, vol. 24, no. 3, jun 2019.

[12] H. Hata, C. Treude, R. G. Kula, and T. Ishio, "9.6 million links in source code comments: Purpose, evolution, and decay," *Proceedings of the International Conference on Software Engineering (ICSE)*, pp. 1211–1221, 2019.

[13] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2013, pp. 712–721.

[14] B. Hartmann, M. Dhillon, and M. K. Chan, "Hypersource: Bridging the gap between source and code-related web sites," in *Proceedings of SIGCHI (CHI)*, 2011, pp. 2207—2210.

[15] N. Sawadsky, G. C. Murphy, and R. Jiresal, "Reverb: Recommending code-related web pages," in *Proceedings of the International Conference on Software Engineering*, 2013, p. 812–821.

[16] Supplementary material. [Online]. Available: https://doi.org/10.5281/zenodo.7622496

[17] T. Fritz and G. C. Murphy, "Using information fragments to answer the questions developers ask," in *Proceedings of the International Conference on Software Engineering (ICSE)*, vol. 1, 2010, pp. 175–184.

[18] N. Haenni, M. Lungu, N. Schwarz, and O. Nierstrasz, "A quantitative analysis of developer information needs in software ecosystems," in *Proceedings of the European Conference on Software Architecture Workshops*, 2014.

[19] A. Bacchelli, M. Lanza, and R. Robbes, "Linking e-mails and source code artifacts," vol. 1, 2010, pp. 375–384.

[20] A. Y. Wang, Z. Wu, C. Brooks, and S. Oney, *Callisto: Capturing the "Why" by Connecting Conversations with Computational Narratives*, 2020, pp. 1–13.

[21] D. Čubranić and G. C. Murphy, "Hipikat: Recommending pertinent software development artifacts," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2003, p. 408–418.

[22] G. Venolia, "Textual allusions to artifacts in software-related repositories," in *Proceedings of the International Workshop on Mining Software Repositories (MSR)*, 2006, pp. 151–154.

[23] R. Holmes and A. Begel, "Deep intellisense: A tool for rehydrating evaporated information," in *Proceedings of the International Working Conference on Mining Software Repositories (MSR)*, 2008, pp. 23–26.

[24] A. Begel, Y. Khoo, and T. Zimmermann, "Codebook: Discovering and exploiting relationships in software repositories," vol. 1, 2010, pp. 125–134.

[25] M.-A. Storey, L.-T. Cheng, I. Bull, and P. Rigby, "Shared waypoints and social tagging to support collaboration in software development," in *Proceedings of International Conference on Computer Supported Cooperative Work (CSCW)*, 2006, pp. 195—198.

[26] C. Parnin and C. Treude, "Measuring api documentation on the web," in *Proceedings of the International Workshop on Web 2.0 for Software Engineering (Web2SE)*, 2011, pp. 25—-30.

[27] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, "Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code," in *Proceedings of SIGCHI (CHI)*, 2009, pp. 1589—-1598.

[28] G. Gao, F. Voichick, M. Ichinco, and C. Kelleher, "Exploring programmers' api learning processes: Collecting web resources as external memory," in *Proceedings of the Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2020, pp. 1–10.

[29] M. M. Rahman, J. Barson, S. Paul, J. Kayani, F. A. Lois, S. F. Quezada, C. Parnin, K. T. Stolee, and B. Ray, "Evaluating how developers use general-purpose web-search for code retrieval," in *Proceedings of the International Conference on Mining Software Repositories (MSR)*, 2018, pp. 465—475.

[30] F. Shull, F. Lanubile, and V. Basili, "Investigating reading techniques for object-oriented framework learning," *IEEE Transactions on Software Engineering*, vol. 26, no. 11, pp. 1101–1118, 2000.

[31] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," *IEEE Transactions on Software Engineering*, vol. 32, no. 12, pp. 971–987, 2006.

[32] A. Li, M. Endres, and W. Weimer, "Debugging with stack overflow: Web search behavior in novice and expert programmers," 2022.

[33] E. Duala-Ekoko and M. P. Robillard, "Asking and answering questions about unfamiliar apis: An exploratory study," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2012, pp. 266–276.

[34] J. Brandt, M. Dontcheva, M. Weskamp, and S. Klemmer, "Example-centric programming: Integrating web search into the development environment," vol. 1, 2010, pp. 513–522.

[35] N. Sawadsky and G. C. Murphy, "Fishtail: From task context to source code examples," in *Proceedings of the Workshop on Developing Tools as Plug-Ins*, 2011, pp. 48—51.

[36] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Seahawk: Stack overflow in the IDE," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2013, pp. 1295–1298.

[37] L. Ponzanelli, G. Bavota, M. D. Penta, R. Oliveto, and M. Lanza, "Prompter: A self-confident recommender system," in *In Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, 2014, pp. 577–580.

[38] X. Liu and R. Holmes, "Exploring developer preferences for visualizing external information within source code editors," in *Proceedings of the Working Conference on Software Visualization (VISSOFT)*, 2020, pp. 27–37.

[39] G. Salton and M. McGill, "Introduction to modern information retrieval." 1986.

[40] P. Jaccard, "Etude comparative de la distribution florale dans une portion des Alpes et du Jura," 1901.

[41] R. DeLine, M. Czerwinski, and G. Robertson, "Easing program comprehension by sharing navigation data," in *Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2005, pp. 241–248.

[42] V. Braun and V. Clarke, *Thematic analysis.* American Psychological Association, 2012.

[43] M. Adeli, N. Nelson, S. Chattopadhyay, H. Coffey, A. Henley, and A. Sarma, "Supporting code comprehension via annotations: Right information at the right time and place," in *Proceedings of the Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2020, pp. 1–10.

[44] A. Woodruff, A. Faulring, R. Rosenholtz, J. Morrsion, and P. Pirolli, "Using thumbnails to search the web," in *Proceedings of SIGCHI (CHI)*, 2001, pp. 198—-205.

[45] J. Teevan, E. Cutrell, D. Fisher, S. Drucker, G. Ramos, P. Andre, and C. Hu, "Visual snippets: Summarizing web pages for search and revisitation," in *Proceedings of SIGCHI (CHI)*, 2009.