







Integrating Issue Management Systems of Independently Developed Software Components

Sandro Speth¹^(✉), Uwe Breitenbücher², Niklas Krieger¹,
Pia Wippermann¹, and Steffen Becker¹

¹ Institute of Software Engineering, University of Stuttgart, Stuttgart, Germany
{Sandro.Speth,Niklas.Krieger,Pia.Wippermann,
Steffen.Becker}@iste.uni-stuttgart.de

² Herman Hollerith Zentrum, Reutlingen University, Reutlingen, Germany
uwe.breitenbuecher@reutlingen-university.de

Abstract. Modern component-based architectural styles, e.g., microservices, enable developing the components independently from each other. However, this independence can result in problems when it comes to managing issues, such as bugs, as developer teams can freely choose their technology stacks, such as issue management systems (IMSs), e.g., Jira, GitHub, or Redmine. In the case of a microservice architecture, if an issue of a downstream microservice depends on an issue of an upstream microservice, this must be both identified and communicated, and the downstream service's issues should link to its causing issue. However, agile project management today requires efficient communication, which is why more and more teams are communicating through comments in the issues themselves. Unfortunately, IMSs are not integrated with each other, thus, semantically linking these issues is not supported, and identifying such issue dependencies from different IMSs is time-consuming and requires manual searching in multiple IMS technologies. This results in many context switches and prevents developers from being focused and getting things done. Therefore, in this paper, we present a concept for seamlessly integrating different IMS technologies into each other and providing a better architectural context. The concept is based on augmenting the websites of issue management systems through a browser extension. We validate the approach with a prototypical implementation for the Chrome browser. For evaluation, we conducted expert interviews, which approved that the presented approach provides significant advantages for managing issues of agile microservice architectures.

Keywords: Microservices · Issue management · Service engineering · Component-based architectures · Gropius · Browser extension

1 Introduction

The component-based architectural style, e.g., microservices, gained much attention since it enables the development of individual software components independently from each other and composing them systematically to new systems [1]. Developing components independently has significant advantages, e.g.,

© The Author(s) 2023

C. J. Stettina et al. (Eds.): XP 2023, LNBIP 475, pp. 3–19, 2023.

https://doi.org/10.1007/978-3-031-33976-9_1

microservices can be implemented in different programming languages by different teams [2]. However, this independence can also result in drawbacks when it comes to managing issues [3], such as bug reports or feature requests. For example, in microservice architectures, we can observe that often different issue management systems are used for different microservices. Moreover, as soon as an architecture includes 3rd-party services, such as SaaS offerings, or open-source components, these, of course, have their own issue management systems, and it is clear that not all services are developed by the same team. Therefore, it is very unlikely that all components of a larger system use the same issue management system. Hence, different teams might use different *issue management systems* (IMSs), e.g., Jira, GitHub, or Redmine [4]. In the following, we use microservices as an example of independently developed components to make the problem clear. For example, suppose a malfunction, anomaly, or failure of a *Checkout* microservice actually results from a bug in an invoked *Payment* microservice, e.g., missing parameters through a change violating the services contract [5]. Figure 1 depicts such a scenario. In this case, the malfunction of the *Checkout* microservice should be reported as an issue relating to its causing bug of the *Payment* microservice, hence, allowing the developers of the *Checkout* service to track the bug report’s status of the *Payment* service. Furthermore, the *Checkout* service’s team should communicate the issue to the *Payment* service’s developer team to resolve the issue. However, documenting and tracking such *cross-component issues* is challenging [3], primarily since no issue management system supports explicit relations to other issues managed by another issue management system as different IMSs are not integrated with each other. Thus, if issues of microservices are managed by different tools, e.g., one microservice is managed in Jira, the other in Redmine, it is not supported by any IMS to semantically relate the issues with each other, which is often required to understand the dependencies of issue propagations, e.g., because of cascading failures, and how issues need to be solved. Furthermore, developers often communicate via the issue’s comments and keeping track of the related issues results in switching between the different issue management systems. This leads to many context switches and, thus, ineffective communication and preventing the developers from focused work. However, especially in modern agile processes like Scrum, “getting things done” is a central value that is thereby hindered. Furthermore, it is required to describe the dependency relation between the two issues while putting them in their correct architectural context to prevent the developers of *Checkout* microservice from trying to fix the malfunction separately, which is problematic since another microservice causes it. Therefore, we require to bypass the limitation of IMS boundaries and integrate their issues into each other, which leads to our first research question:

RQ1: “How could different issue management systems such as Jira and GitHub be seamlessly integrated to avoid context switches through additional tooling?”

In addition, developers need an efficient solution to discover dependencies of their own issue to issues in other issue management systems, as these are often

hidden in current systems only via natural language in the comments. This leads us to our second research question:

RQ2: “How could developers in their own issue management system quickly identify dependencies of their own issues on issues of other issue management systems?”

For this reason, in this paper, we propose a concept that makes issues from different issue management systems available in each other in an IMS-independent way. The concept is based on seamlessly augmenting the websites of issue management systems such as Jira, Redmine, or GitHub via Browser extensions. In doing so, we use Gropius, which we developed in previous work [6, 7], in the extensions to synchronize issues and enable Gropius features, such as semantic links in the IMS, that do not otherwise exist. Gropius is a standalone IMS platform that enables issue management across the boundaries of every single software component and their IMSs while putting the issues in their correct architectural context. However, while Gropius offers a top-down view of the entire architecture and their issues as a separate tool, developers still require a seamless developer-specific view from inside a component in their usual issue management systems without switching between IMSs. Therefore, our concept maps the top-down view of the architecture to a developer-specific view of the architecture from within a component. This hides unnecessary or disruptive information from the developer and puts the focus on the developer’s own component and the upstream or downstream components from it. Thus, if certain issues of a component are shown, for example, in GitHub, associated issues that might be managed by other issue management systems such as Jira are directly embedded in the website of GitHub, which enables quickly recognizing dependencies and navigating to other issues. Developers should therefore get their work done faster and communicate more efficiently via the dependent issues’ comments with other teams. We validated the practical feasibility of this *Gropius Browser Extension* concept by a prototypical implementation for the Chrome browser and GitHub as an exemplary issue management system. Moreover, we evaluated the concept and the prototype through expert interviews. The experts approved that the approach provides significant advantages for managing issues of individual services in a component-based architecture, e.g., independently developed microservices.

2 Fundamentals of Issue Management and Gropius

Today’s issue management systems (IMS), such as Jira or Redmine, allow developers to document, manage and track issues, e.g., bug reports or feature requests, for their projects, discuss the issues’ context with other developers, and collaborate on them. Often, architectural design decisions are also discussed in issues as they both influence each other. Therefore, issue management systems often act as the main tool to coordinate their work and become increasingly central to every software development process as a hub for communication

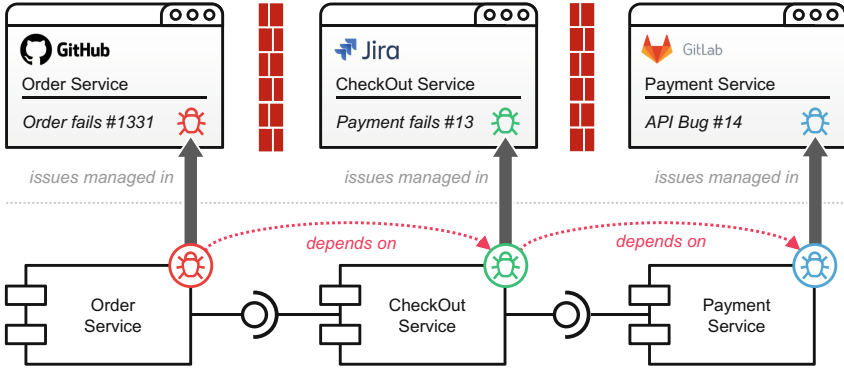


Fig. 1. Motivating scenario showing how bugs propagate through the architecture.

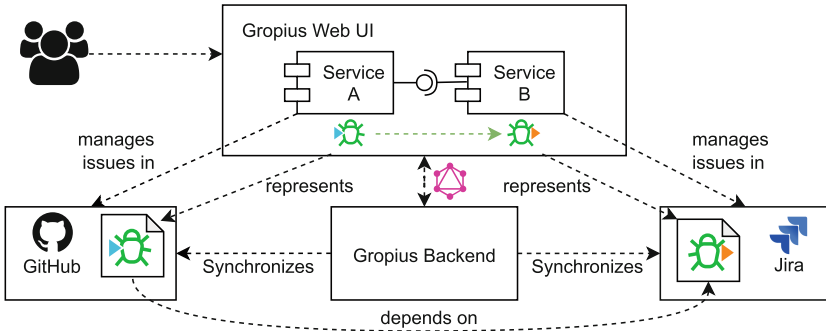


Fig. 2. Overview of the Gropius concept and architecture.

and collaboration [8]. Furthermore, issue management systems offer the possibility for distributed negotiations required to resolve the issue [9]. However, in modern software development, individually developed components typically use their independent issue management systems. Therefore, linking an issue from one IMS to another systems’ issue, e.g., for a dependency relation, is not directly possible as traditional IMSs are restricted to their component’s boundaries. Nevertheless, related issues should be linked amongst each other [8] in order to provide all helpful information required to resolve the issue and increase its quality [10]. Gropius attempts to remove this limitation and facilitate (1) synchronization of issues across different issue management systems, and (2) relations between those cross-component issues. In Gropius, each issue can be associated with one or more components that together form the architecture of a software system. Such a software system is represented in a project in Gropius, i.e., a *Gropius Project*. In general, Gropius supports various kinds of components, e.g., microservices, libraries, and infrastructure. One goal of Gropius is to effectively represent the impact of issues on the architecture and issue propagation, as well as architectural dependencies between the components. For this reason, in the *Gropius Web UI*, components of such a Gropius project are modeled in a UML

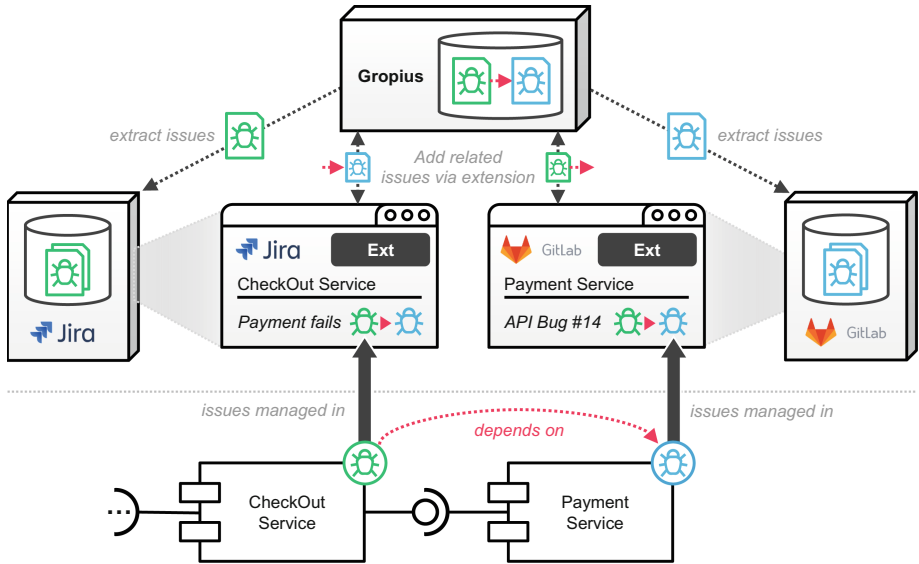


Fig. 3. Overview and architecture of the Gropius Browser Extension concept.

component diagram-like graph. Stakeholders with an overview of the entire software system, e.g., a software architect, can create representations of components and their provided interfaces in Gropius and connect them to model the architecture. Furthermore, each component in Gropius is either Gropius internal or requires a URL to its actual IMS. Each component is attached with the issues that affect that component. Through Gropius, developers can create relations between issues of different components which are added to the issues. These relations are graphically represented by arrows and collectively depicted in the issue view metadata. To enable these cross-component issue features while leveraging the issues of the actual IMSs, Gropius acts as a wrapper over traditional IMSs, such as GitHub or Jira, as depicted in Fig. 2 and offers a GraphQL API to clients. Through specific adapters for the different IMSs, Gropius synchronizes issues between the systems. Furthermore, Gropius keeps a copy of the issues, especially for data that is not supported by the actual IMSs, such as cross-IMS issue relations.

3 The Gropius Browser Extension Concept

This section presents the conceptual idea of the *Gropius Browser Extension (GBE)* for integrating different issue management systems used by independently developed software components. First, we give a brief overview of the concept in Sect. 3.1. Afterwards, we explain the concept with regard to elicited requirements in detail. Please note that while we use microservices as a common example for independently developed software components, the GBE concept is not restricted

to microservices but also enables other independently developed and includable software projects, e.g., libraries or infrastructure components such as Kubernetes and Docker. Therefore, we use the general term “*component*” for the remainder.

3.1 General Idea and Objectives

Figure 3 shows the general idea based on the motivating scenario. The main objective is to enable *cross-component issue management across different issue management systems in a seamless manner* using their standard websites in a traditional way. The websites get seamlessly enriched by the extension with information about related issues of dependent components in the architecture, which are possibly managed by other issue management systems. This integration concept avoids two drawbacks of existing issue management approaches: (i) First, relations between issues managed by different IMSs are typically documented using textual comments, which provides no systematic means and hinders “browsing” them. (ii) Second, additional tooling, such as the Gropius Web App, that enables cross-component issue management is not required for creating, relating, and managing issues, thus, reducing context switches of developers.

3.2 Overview and Architecture of the GBE

The Gropius Browser Extension is a plugin that runs in a web browser and automatically manipulates the HTML DOM of shown websites, in our case, the websites of issue management systems such as Jira or Redmine. The extension connects to the Gropius backend, which contains information about issues in different IMSs and also contains dependencies between them as well as other metadata [6, 7]. Thus, the Gropius backend acts here as a *normalized store* for issues from different IMSs, which are automatically extracted and synchronized by Gropius in a certain time interval.

The GBE has access to the DOM of the currently shown page of the issue management system, from which it retrieves relevant context information, e.g., which issue is currently shown. Using this information, the GBE calls the Gropius backend to fetch cross-component issue features for the current issue, e.g., related issues and their respective components. This information is additionally injected into the issue management system’s website by manipulating its HTML DOM, hence, enabling Gropius functionality directly in the component’s issue management system’s website following its look and feel. Thus, this provides a seamless integration of the Gropius features into the regular work of developers as no additional website or tool needs to be used to see issue relations, issues of other components, etc. Especially, content based on different issue management systems can be integrated into the developer’s used issue management system.

In the following, we concretely describe the Gropius features that are integrated using the GBE into the standard website of the issue management systems. Please note that we describe in this section only the concepts and provide details on the technical implementation in Sect. 4, which describes our open-source prototype of the Gropius Browser Extension.

3.3 Gropius Features Integrated in the Browser Extension

In this section, we describe the Gropius features and functionalities that are integrated by the Gropius Browser Extension into the standard websites of issue management systems and how they are realized in the presented architecture.

Browser Extension Feature 1: Creating Gropius Project. Before developers can use any Gropius features in the used IMS, they must create a project in Gropius, add relevant components to this project, and specify the respective issue management systems. This functionality is supported directly by the GBE as otherwise the external Gropius Web App needs to be used, which would lead to context-switches. Therefore, we integrated a feature to create a Gropius project or to add an IMS's project, e.g., a GitHub repository, to an existing project. This way, developers can add microservices, libraries, and other components to their Gropius project when browsing through the respective project of the IMS used. Please note that the way this feature is shown and how it is triggered, e.g., through a button, is IMS-specific. Therefore, the browser extension will look and feel slightly different for GitHub than for Jira and other systems.

Browser Extension Feature 2: Issue Dependencies and Architectural Context. One main objective of the Gropius Browser Extension is to enrich an issue currently shown in an issue management system's website with additional information regarding its dependencies to other issues. Especially if the related issues belong to another component, the extension must provide the architectural context in an understandable fashion in order to enable the developer to quickly identify the impact and dependencies of the issue to other components and their issues. Therefore, the GBE injects two kinds of additional information to the issue view of an IMS: (1) the related issues of the same component and of other components, and (2) an excerpt of the architecture graph that shows the component of this issue and also all other components having issues to which the currently shown issue relates to.

For the first kind of injected information, i.e., the list of related issues of the same or other components, the Gropius Browser Extension shows for the currently shown issue which components and other issues the shown issue directly affects or is otherwise related to. To easily create relations between the current issue viewed to other issues of the same or a different component, the Gropius Browser Extension enables to add such relations directly in the issue view page of the issue management system, thus, no further tooling is required. For example, if developers create an issue for the own component and know that this issue actually results from another component's issue, i.e., has a dependency to it, they are interested in directly adding the relation to the other issue in order to specify the complete dependencies. Without the Gropius Browser Extension, developers would have to add a text comment including the URL to the related issue of a possibly different issue management system and write an explanation which describes the semantics of the relation. Of course, text-based issue relations are

only a workaround that tries to compensate this obviously missing feature to link issues in different systems. On the other side, using the Gropius Browser Extension, this problem is solved since relations to issues of the same component or other components can be directly added in the issue’s view page including a precise specification of the semantics of the relation, e.g., that the issue *depends on* an issue of another component in another IMS.

In general, adding such relations are especially complex for issue management systems that already allow semantic issue relations, for example GitLab’s *Linked issues* or Redmine’s *Related issues*. While such issue management systems allow relating issues within the same issue management system provider, relating issues to other providers, e.g., issues of a GitHub repository, is impossible. Therefore, the GBE enables integrating relations to issues of different issue management system providers in the currently shown issue view.

The second kind of information the Gropius Browser Extension adds to an issue’s view is an excerpt of the overall architecture of the system that shows (a) the component of the currently shown issue as well as (b) all other components having issues to which the shown issue relates. Thus, this provides a visual overview of the architectural context and enables developers to quickly understand the impacts and causes that the shown issue has, i.e., all dependencies.

Browser Extension Feature 3: Entire Architecture of the System. The Gropius Browser Extension aims at enabling developers a quick overview of issues and their relations to other issues of possibly other components in the architecture. Therefore, the previously introduced Browser Extension Feature 2 enables to show an architectural excerpt that provides an overview of the directly related issues and their respective components. However, this excerpt only shows a part of the system’s architecture and often developers need to see the entire architecture of the system in which the current component is used. Therefore, the GBE enables to load the entire architecture of a system that includes all components, their issues, and their dependencies into the currently used issue management system’s website. Thus, developers can browse the entire architecture and directly jump to the IMSs of other components by clicking on the component in the graph or on one of its issues. In the latter case, a new tab opens the other IMS and shows the clicked issue. This enables browsing through the entire architecture of the system directly in the IMS and avoids that another tool needs to be used by developers to inspect the overall system architecture.

Please note that a certain component and its issues can be part of multiple different systems. For example, a component such as the Payment Service might be used in our webshop motivation example but also in a car rental application or other applications that require payment functionality. In this case, a certain component would be part of multiple different system architectures which are managed in Gropius as individual Gropius projects. Thus, one Gropius project represents one system, describing its architecture, and listing all components with the corresponding issues. Therefore, the Gropius Browser Extension enables

switching between Gropius projects in the issue management systems website and then shows the architecture and all components of current selected project.

Browser Extension Feature 4: Create Issues for Other Components Managed in Other Issue Management Systems. In many situations, a bug in one component results from a bug in an invoked component. For example, in the motivating scenario, the bug in the Payment Service’s API leads to an issue in the CheckOut Service that depends on this API. Thus, if a developer of the CheckOut Service detects that the checkout functionality does not work correctly, the developer needs to report this as an issue for the CheckOut Service. However, if the developer directly understands that the root cause of this bug is the broken API of the Payment Service, maybe as a result of a failure root cause analysis, also a bug should be created for the Payment Service and the two issues should be linked, i.e., that the checkout bug results from the payment’s API bug. Therefore, the Gropius Browser Extension supports creating issues also for other components of the architecture directly in the issue’s view page of the issue management system, even if the current IMS is different from the IMS of the other component. Based on this feature, developers do not require to open the other issue management system to report the causing issue. This is a significant advantage since such a context switch requires the developers to first orient themselves in the other issue management system.

4 Prototypical Validation

To validate the practical feasibility of the presented concept, we implemented a prototype of the Gropius Browser Extension for the Google Chrome Browser. The implementation is open source available¹. Section 4.1 describes the technical design of the prototype. Moreover, Sect. 4.2 shows how the implemented browser extension for Chrome supports GitHub, i.e., how the extension interacts with the Web UI of GitHub and how issues from other issue management systems are embedded.

4.1 Technical Design of the Gropius Browser Extension for Chrome

The architecture of our Gropius Browser Extension is based on the *web extensions framework*, which is an official W3C standard cross-browser architecture² and supported by all popular desktop web browsers except for Safari.

Figure 4 depicts our prototype’s architecture for the Google Chrome browser. Following the web extension framework, our prototype consists of three components (1) *background pages*, (2) *UI pages*, and (3) *content scripts*. The extension executes background pages as soon as loaded. Especially code that maintains long-term state and operations, which should be independent of any website’s

¹ <https://doi.org/10.5281/zenodo.6810943>.

² <https://browserext.github.io/browserext/#availability-csp-content>.

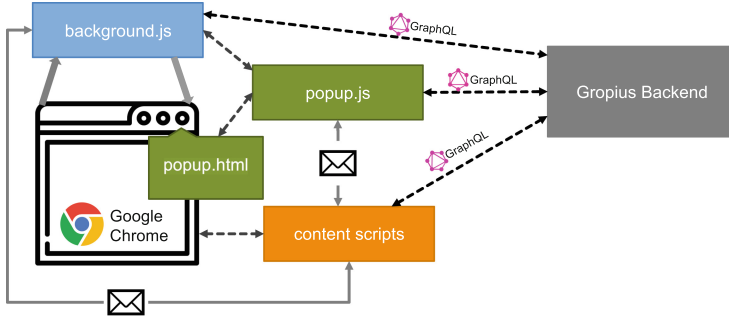


Fig. 4. Architecture of the Gropius Browser Extension prototype for Google Chrome.

or browser window’s lifetime, must be added as background pages. In the context of our extension, the background scripts decide whether one of the content scripts should be active, i.e., the content script for a GitHub issue view, if the developer has opened the GitHub page. When clicking on the extension popup button in Chrome’s upper right part, it offers UI elements and options for login to Gropius to the user via the UI pages component, i.e., *popup.js* and *popup.html*. While the background and UI pages can access the browser’s API, they cannot interact with the issue management system’s website. In contrast, the content scripts interact with the issue management system’s website, however, having only limited access to the browser’s API. Therefore, content scripts run in the context of the issue management system’s website, and the background and UI pages run in the context of our GBE. Content scripts enable the way websites are displayed to the user and their functionalities to be manipulated. In our context, this means adding UI elements related to Gropius to the issue management system’s website, thus, enhancing the issue management system’s website with additional Gropius functionalities. For example, a button to add a new related issue in the issue’s view, which opens a dialogue to add the relation, is injected here. The communication between content scripts, background, and UI pages works via messages. Therefore, components listen to other components’ messages and respond via the same channel. The extension interacts with the Gropius backend via its provided GraphQL API to add or retrieve architectural and Gropius project information, issues, and issue relations.

4.2 Implementation for GitHub and Case Study

We implemented our prototype in Vue.js, a progressive JavaScript framework for web applications. For our prototype, we especially use the plugin “vue-cli-plugin-browser-extension”, a third-party library for Vue that supports creating extensions for popular browsers. For our prototype, we focus on Google Chrome as web browser and the issue management system of GitHub. In the following, we describe the prototype based on an extended version of the example scenario

Failed to checkout a new order #1

Edit

New issue

Open

opened this issue 13 hours ago · 0 comments

Gropius Component Graph

```

graph LR
    Order-Service -- REST-API --> CheckOut-Service
    CheckOut-Service -- REST-API --> Payment-Service
    
```

Active Gropius Projects: Webshop

Related Issues:

- Payment fails if credit card holder's na...
Component(s): Payment-Service
- Cannot process order with German uml...
Component(s): Order-Service

Assignees: No one—assign yourself

Labels: bug

Projects: None yet

commented 13 hours ago (Owner)

Brief: When calling the API endpoint to check out a new order and provide the order ID, and selected payment information, the API returns an HTTP status code 400 Bad Request. Additionally, the log contains error messages regarding a parsing error of the credit card holder's name, if the name contains special characters.

Steps to reproduce:

Fig. 5. Browser screenshot of the Gropius Browser Extension prototype.

described in Sect. 1, i.e., the webshop application consisting of the (1) Order Service, the (2) CheckOut Service, which is the service our development team focuses on, and the (3) Payment Service.

The Checkout Service consumes the Payment Service’s API to handle the payment of orders. The Checkout Service’s issues are managed in GitHub and the other services in different IMSs. The prototype allows adding a GitHub repository as a component to an existing Gropius project (cf. Feature 1). To add a GitHub repository which is opened in the web browser to a Gropius project, the extension injects a button in the sidebar of the repositories “Code” tab. After modelling the provided interfaces and architectural dependencies in the Gropius Web App, developers can add relations between issues of the CheckOut Service and issues of the other components (cf. Feature 2). Figure 5 depicts an example issue for the CheckOut Service, which describes a bug regarding checking out an order. On top of the issue’s description, the GBE injects an architecture graph to show the issue’s architectural context (cf. Feature 2 & 3). Our prototype injects two additional metadata in the issue view’s sidebar above the assignees. The “Active Gropius Projects” part allows switching the active Gropius project in which the CheckOut service is included. In the example, the Gropius project “Webshop” is selected as active. Therefore, the extension shows in the “Related Issues” part related issues and their components in the context of this project (cf. Feature 2). The icons next to the related issue’s titles indicate whether the currently viewed issue depends on the related issue or causes it. Additionally, the “Related Issues” part allows developers to add new relations to issues of

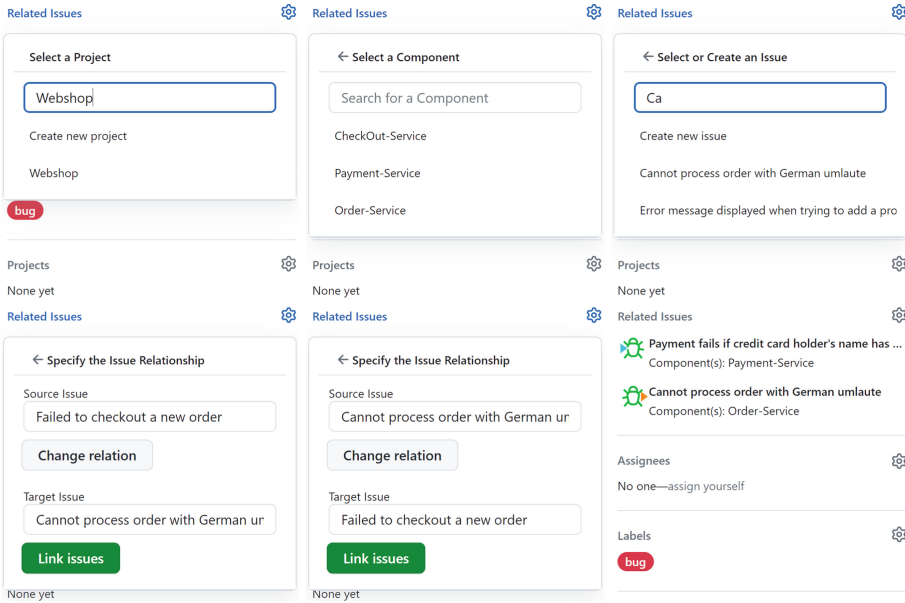


Fig. 6. Steps for adding a related issue.

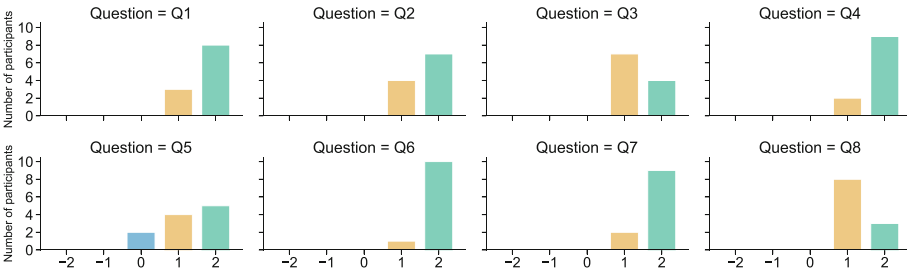


Fig. 7. Results of the evaluation for each question.

the same or different component or remove existing relations. As depicted in Fig. 6, when adding new relations, the extension opens a dialogue frame where a developer has first to select the component of the related issue and then select the issue which should be related. Furthermore, instead of selecting an existing issue, a developer can create new issues for the chosen component directly in the issue’s view (cf. Feature 4).

5 Evaluation

We evaluated our concept based on a *Goal Question Metric* approach and a prototype version of our software. We formulated the goal “enabling integrated context-aware issue relationship management for independently managed

software systems". Based on this, we derived questions evaluating the general concept and how our prototype fulfills the goal. To answer the questions, we conducted an expert survey. As the rating scale for the questions, we used five interval points where the endpoints are named to match associated questions. Therefore, the scale ranges from -2 over 0 to 2 , where -2 means an entirely negative evaluation, 0 is the neutral element, and 2 means an entirely positive evaluation. Since our concept aims especially to improve the developers' issue management across multiple independently managed components, we consulted eleven experienced industrial software developers from different small-sized to big-sized companies in Germany and Canada. The participants worked between 3 and 15 years in developing component-based systems, during the last years also focusing on microservices. Most of their roles are developer and software architect. Furthermore, while developing, their components interacted with components developed by other teams. We conducted the expert survey by hosting interviews in which we showed the prototype version of the GBE and an example architecture with issues to the participants. Afterwards, we provided them with our questionnaire.

5.1 Results of the Expert Survey

This section describes the questions and the results of the survey.

- (Q1) *"How helpful is the main feature of the Gropius Browser Extension to see related issues of other components?"* The scale ranges here from *not helpful at all* (-2) to *very helpful* ($+2$). In total, eight experts (73%) rated (Q1) with $+2$ and three (27%) rated it with $+1$, which means that they evaluated to see related issues of other components in a concise way as very helpful.
- (Q2) *"How big do you estimate the time savings enabled by the Gropius Browser Extension to find related issues of other components (compared to the common approach without the extension)?"* The interviewees were asked to rate this with a value ranging from *very low* (-2) to *very high* ($+2$). Seven of the experts (64%) rated $+2$, four experts (36%) rated $+1$. Thus, developers estimate that the extension may help saving time for finding related issues.
- (Q3) *"How important do you consider the presented concept of the Gropius Browser Extension for the development process of software and its operation?"* The interviewees were asked to rate this with a value ranging from *not important at all* (-2) to *very important* ($+2$). For (Q3), four experts (36%) rated a $+2$ while seven experts (64%) stated the importance of our concept for the development process with a $+1$. Signals that our concept is generally important but not significantly required.
- (Q4) *"How well do you find the Gropius Browser Extension integrated into the look and feel of GitHub?"* The interviewees were asked to rate this with a value ranging from *not good at all* (-2) to *very good* ($+2$). Overall, nine experts (82%) evaluated the integrated look and feel of our prototype with

+2, and two experts (18%) stated a +1, which strongly indicates that our prototype seamlessly integrates into GitHub.

- (Q5) “*How high do you rate the chance that the Gropius Browser Extension will be accepted in the industry?*” The interviewees were asked to rate this with a value ranging from *very low* (−2) to *very high* (+2). The acceptance of our concept within the industry was evaluated as relatively high. However, the results are well distributed between +2 and 0, i.e. five times (45%) +2, four times (36%) +1, and two times (18%) 0. Therefore, not every expert was convinced of acceptance in the industry.
- (Q6) “*How important do you consider the concept of cross-component issues for the software engineering process of complex systems that consist of several independent components?*” The interviewees were asked to rate this with a value ranging from *not important at all* (−2) to *very important* (+2). For (Q6), ten experts (91%) rated the concept of cross-component issues as very important, i.e., appointing a value of +2, and one expert (9%) esteemed it as important with a value of +1.
- (Q7) “*How quickly can you recognize that the issue currently viewed has dependencies on issues of other components?*” The interviewees were asked to rate this with a value ranging from *very slow* (−2) to *very fast* (+2). With nine experts (82%) rated (Q2) with +2 and two experts (18%) rated a +1, the average of +1.82 means that the experts could recognize dependencies on issues other component’s issues very fast, thus, supporting the usability.
- (Q8) “*How user friendly is editing and adding related issues?*” The interviewees were asked to rate this with a value ranging from *not user friendly at all* (−2) to *very user friendly* (+2). Three experts (27%) judged the user friendliness for editing and adding related issues with a +2 and eight (73%) with a +1. Therefore, the average of +1.27 supports the user friendliness.

5.2 Summary of the Evaluation and Threats to Validity

To summarize, the feedback regarding our concept was very positive. Therefore, our expert survey shows that a problem in managing cross-component issues in complex software systems, such as microservices, is seen, supporting some of the challenges outlined by Mahmood et al. [3]. Often, ratings of +1 and below were justified by the fact that there are more important tasks, especially regarding the entire development process and operation. Additionally, our prototypical implementation was rated very good. In particular, the immersive integration and design to GitHub’s UI was frequently emphasized in debriefings. While our concept provides one possible answer for our first research question, especially the answers to question 7 support that integrating the dependency graph into the traditional IMS’s UI and adding relations to other issues in a semantic clear way enables efficient identification of the own issue’s dependencies, thus, answering our second research question.

Please note that since we conducted an expert survey, the given answers depend on personal opinions. Thus, the results are subjective, resulting in a

threat to internal validity and external validity regarding how representative our surveyed group is [11]. Furthermore, there is a threat to construct validity if the experts misunderstood some questions. However, these threats were solved by allowing the interviewees to ask questions that were answered.

6 Related Work

There are several attempts for cross-component issue management. Various Redmine and Jira forums³ discuss how issues can be related to multiple IMS projects. Proposed solutions are always limited to one IMS provider and are usually plugins for the respective IMS providers instead of a provider-independent browser extension. Issue tracking across the boundaries of an IMS provider is not enabled, and is, therefore, not practical for our use case where components are developed independently and thus manage their issues in different IMS providers, e.g., GitHub and Jira. Especially for Jira and GitHub, plugins exist that enable approaches that support multiple IMS projects from the same provider. Synchronization of issues across multiple Jira projects is enabled by the Jira plugin *Backbone Issue Sync*⁴ and the *Multi Project Picker*⁵ Jira plugin removes the restriction that an issue can only belong to one Jira project. For GitHub, there is the plugin ZenHub⁶ which offers additional project management functionalities. With ZenHub, issues are not restricted to one GitHub repository anymore. However, ZenHub does not support linking GitHub issues to issues of other IMS providers. Additionally, there are browser extensions for issue management in Jira which aim at saving time, e.g., JIRA Assistant, Zephyr, JIRA Template Injector, and Google to Jira. However, similar to the plugins above, none of the extensions enables issue relations across different IMS projects or providers. We followed with GropiusVSC, an IDE extension for Gropius, a similar approach as we to reduce context-switches and improve a developer's productivity [12]. GropiusVSC offers a developer for a selected component of a Gropius project the list of issues and an issue view. The issue view shows the issues title, description, and related issues with their components. By clicking on a related issue of another component, this component is automatically selected as active component, and GropiusVSC shows the issue's view. While GropiusVSC is helpful during development time, a relevant part of a developer's week consists of meetings, e.g., Sprint Planning, or other occasions where developers work within the actual issue tackers' web pages, and, therefore, not sufficient alone. Hence, our browser extension helps developers by integrating context-aware issue management across a component's boundary in the component-specific IMS.

³ <http://bit.ly/3EdNU0g>, <https://bit.ly/Iuk3NrP>, and <https://bit.ly/3IKKv2n>.

⁴ <https://www.k15t.de/software/backbone-issue-sync-for-jira>.

⁵ <http://bit.ly/3xqofh5>.

⁶ <https://www.zenhub.com/>.

7 Conclusion

The central objective of this work was that developers of a component could manage cross-component issues in the context of the issue management system they use. We achieved this by the Gropius Browser Extension concept that seamlessly integrates various issue management systems via the Gropius backend. Our prototype has shown that the concept can be practically realized while the conducted expert interviews in the evaluation confirm the relevance of the approach. The main advantages of the approach are the reduction of developers' context switches and the avoidance of a separate tool in the development process to manage issues in different management systems. In future work, we will extend our prototype for all common issue management systems, such as Jira.

References

1. Szyperski, C., Gruntz, D., Murer, S.: *Component Software: Beyond Object-oriented Programming*. Pearson Education (2002)
2. Nygard, M.: *Release It!: Design and Deploy Production-Ready Software*. Pragmatic Bookshelf (2007)
3. Mahmood, S., Niazi, M., Hussain, A.: Identifying the challenges for managing component-based development in global software development: preliminary results. In: *Science and Information Conference (SAI)*. IEEE 2015, pp. 933–938 (2015)
4. Speth, S.: Semi-automated cross-component issue management and impact analysis. In: *Proceedings of 2021 36th IEEE/ACM International Conference on Automated Software Engineering*, IEEE, pp. 1090–1094 (November 2021)
5. Ramírez, F., Mera-Gómez, C., Bahsoon, R., Zhang, Y.: An empirical study on microservice software development. *SESoS/WDES* **2021**, 16–23 (2021)
6. Speth, Sandro, Breitenbücher, Uwe, Becker, Steffen: Gropius — a tool for managing cross-component issues. In: Muccini, Henry, Avgeriou, Paris, Buhnova, Barbora, Camara, Javier, Caporuscio, Mauro, Franzago, Mirco, Koziolok, Anne, Scandurra, Patrizia, Trubiani, Catia, Weyns, Danny, Zdun, Uwe (eds.) *ECSA 2020. CCIS*, vol. 1269, pp. 82–94. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59155-7_7
7. Speth, S., Becker, S., Breitenbücher, U.: Cross-component issue metamodel and modelling language. In: *Proceedings of the 11th International Conference on Cloud Computing and Services Science*, SciTePress, pp. 304–311 (May 2021)
8. Bertram, D., Volda, A., Greenberg, S., Walker, R.: Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams. In: *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work*, pp. 291–300 (2010)
9. Sandusky, R.J., Gasser, L.: Negotiation and the Coordination of Information and Activity in Distributed Software Problem Management. In: *Proceedings of the 2005 International ACM SIGGROUP Conference on Supporting Group Work*, ACM, pp. 187–196 (2005)
10. Bettenburg, N., et al.: What makes a good bug report? In: *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. *SIGSOFT 2008/FSE-16*, pp. 308–318. ACM (2008)

11. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. *Emp. Softw. Eng.* **14**(2), 131–164 (2008)
12. Speth, S., Krieger, N., Breitenbücher, U., Becker, S.: Gropius-VSC: IDE support for cross-component issue management. In: Companion Proceedings of the 15th European Conference on Software Architecture, CEUR (October 2021)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

