GemChecker: Reporting on the Status of Gems in Ruby on Rails Projects

Jamie Cleare
School of Computing
University of Portsmouth
Portsmouth, United Kingdom
jamie.cleare@myport.ac.uk

Claudia Iacob
School of Computing
University of Portsmouth
Portsmouth, United Kingdom
iacob@port.ac.uk

***Abstract*— Ruby projects rely on gems, i.e. package libraries which provide a variety of features and functions. Once a package library has been installed onto an application, checking if it has become out of date or if it is poorly maintained can only be done manually for Ruby on Rails projects. This is both error prone and time consuming. Out of date gems can potentially introduce vulnerabilities that may only become obvious at a later stage. In this paper, we introduce GemChecker, a software tool designed to support Ruby on Rails developers in gaining knowledge about the version status of gems installed upon their application. GemChecker is designed to: a) allow queries of the latest version available for a gem, b) summarize the results of checking the versions of all the gems associated with a particular project, and c) support software maintenance tasks by alerting developers of code deprecation in gems used by a particular project, of new versions being released for particular gems, and when a gem used by a particular project is out of date.**

Keywords—Gems, version history, Ruby on Rails.

Tool link: https://bit.ly/2INaaPO

I. Introduction

Ruby on Rails projects rely on gems, i.e. package libraries which provide a variety of features and functions. While many of these are developed by the Ruby community, each gem complies with the same repository format. Information about each gem is available at the companion website [3] and it includes details about the different versions available for a particular gem. Once a package library has been installed onto an application, checking if it has become out of date or if it is poorly maintained can only be done manually. This is a challenge for a couple of reasons. First, out of date packages can introduce vulnerabilities and deprecated code into the application using it. These can be difficult to spot or they only become apparent when their consequences start showing up. Second, relying on a process where individual gems version status is checked manually is time consuming and error prone. Complex Ruby on Rails projects may rely on hundreds of gems which can change regularly or without following any pattern. It is not feasible to check the status of these gems manually at regular periods of time. In this paper, we introduce GemChecker, a software tool supporting Ruby on Rails developers in gaining knowledge about the version status of gems installed upon their application. GemChecker alerts developers to different version related issues that may affect their applications, with the intention of supporting them in maintaining Gem versions, prevent vulnerabilities and avoid introducing deprecated code into their applications.

II. Usage Scenarios

GemChecker is designed to support several maintenance scenarios for Ruby on Rails projects, including:

Scenario 1: Jane is a developer maintaining 3 large Ruby on Rails projects. Each project uses over 100 gems and the version status of 20% of these gems is critical for the security of the overall project. The only way of understanding how many of the gems used by a project are out of date at any given time is for Jane to manually inspect the history of each of the gems used. With 300 gems used across projects, this is time consuming and error-prone. GemChecker supports Jane by automatically generating a report including the currently used version and the latest version available for each gem within each project. All the gems currently out of date are highlighted.

Scenario 2: Ann is a tester responsible for a suite of Ruby on Rails systems. These systems are regularly updated and Ann is responsible for ensuring that regression testing is performed for any new release. As all the projects use gems, part of the regression testing process is to ensure that any code deprecation within the gems does not affect the existing functionality of the systems. The only way for Ann to ensure this is by first manually identifying all gems used by the systems which include code deprecation. With new versions of these systems being released regularly, this task is tedious and error-prone. GemChecker supports Ann by automatically detecting code deprecation in all the gems used by any of the projects.

Scenario 3: Mary is a project manager of a complex Ruby on Rails project comprising of 1 million lines of code and using 785 gems. Mary's responsibilities include keeping track of the versions available for all the gems used by the project and notifying the development team when a new version is made available for a particular gem. The development team will then decide whether the project will keep the current version used or upgrade the gem to its most recent version. GemChecker supports Mary by automatically alerting her whenever a new version of a gem used as part of the project is released.

Table 1 – GemChecker information classified based on the 4 types if notifications available in Atom

Information • New version available for a gem • New dependencies identified for a gem • All gems are up to date • X number of gems require attention	**Success** • Update successful • It is safe to upgrade <gem name>
Warning • Gem out of date • New version available for gem and amount of time since release • Deprecation in new versions • Unsupported gems • X number of gems out of date	**Error** • Gem incompatibility • Could not find gem • Could not find gemserver • Deprecation message

III. Analysis and Design

In developing GemChecker, we worked closely with a small-size team of Ruby on Rails developers who helped us in both eliciting the requirements for GemChecker and in evaluating the system after every iteration of its development. The team consisted of six developers, who possessed a range of development experience ranging between less than a year to over 20 years.

A. Requirements

A focus group was organized with the 6 Ruby on Rails developers in order to get a better understanding of their requirements for GemChecker both in terms of visualization and configuration and the type of information they would like to be made available about the gems used as part of the projects they maintain. For eliciting their requirements in terms of configuration and visualization, we asked them to brainstorm for features they would like to see as part of the tool. The first author ran the focus group, facilitating the discussion and stirring it to concrete examples of situations where the participants would make use of GemChecker in their day to day work.

Amongst the ideas all developers were enthusiastic about was developing GemChecker as an add-on to Atom [7], an open source text editor. Atom makes available four types of notifications: Information, Success, Warning, and Error. While Atom defines the overall meaning of each notification, it does not prescribe the types of information associated with each notification. We asked our participants to brainstorm for types of information they would associate with each notification in the context of using GemChecker as part of maintaining their projects. The participants associated successful updates or confirmations that certain upgrades are safe in the context of the project with 'Success' notifications. 'Error' notifications were associated with specific gem incompatibilities, code deprecation being identified, and failing to locate either the gems or information about the gems on the Ruby gems server. The participants expressed interest in being notified when a new version is available for a gem used by projects they maintain, new dependencies are identified for a gem used, or if at any given time all gems used by them are up to date. If specific gems required further attention, participants wanted to know the exact number of those gems and they wanted the option of getting further information about those particular gems. Additionally, participants wanted to be warned when any particular gem used as part of the project was out of date and whether new versions of the gem were available. They also wanted to be warned when certain gems became unsupported (Table 1).

The participants agreed that GemChecker should be designed as an add-on for Atom. All the requirements elicited for GemChecker apply in the context of the project open at any given time in the Atom editor and all the gems used by this project:

R1: *Check Gem Details*

R1.1: GemChecker should allow queries on the latest version available for a gem

R2: *Check Gem Maintenance*

R2.1: GemChecker should alert the developer of any code deprecation within the gem.

R2.2: GemChecker should alert the developer when a new gem version is released.

R2.3: GemChecker should alert the developer when a gem is out of date.

R2.4: GemChecker should alert the developer when a gem is out of date by a certain amount of time.

R2.5: GemChecker should alert the developer when information about a gem could not be collected of the server could not be reached.

R3: *Check all Gems Versions*

R3.1: GemChecker should be able to display the version status of all the gems associated with a particular project

R3.2: GemChecker should alert the user to the number of gems that require attention, eg. they are out of date, include deprecated code.

R4: *Generate Summary Report*

R4.1: GemCheker should summarize the results of checking the versions of all the gems associated with a particular project.

R5: *Configuration*

R5.1: The developer should be able to configure the warning levels for each of the requirements above.

B. Architecture

The architectural model for GemChecker is depicted in Figure 1. Atom provides a Workspace visual environment on which the files of a Ruby on Rails project are displayed and edited via the TextEditor component. A Ruby on Rails project - Project A - is associated with two files which include information relevant to the gems used by the project and the versions used as part of the project, namely Gemfile and Gemfile.lock. GemChecker is developed as an Atom add-on comprising two main components: GemDetective and SummaryWriter. GemDetective retrieves the names, descriptions, and versions of all the gems used by the project currently active in the Workspace from the project's respective Gemfile and Gemfile.lock files. Using this information, the GemDetective component dispatches an AJAX call to the gem's documentation, held on RubyGems.org using the RubyGems.org API.

Once GemDetective has received a successful response from the RubyGems.org API, the information is converted into a notification using Atom's Notification and NotificationManager components and displayed in the Atom environment. The SummaryWriter component generates a single summary notification for all gems used by the project currently active in the Workspace and write the details of each gem to a text file. All gems included in this summary report are grouped based on the out of date severity – all gems one version out of date, all gems 2-4 versions out of date, and all gems 5+ versions out of date.

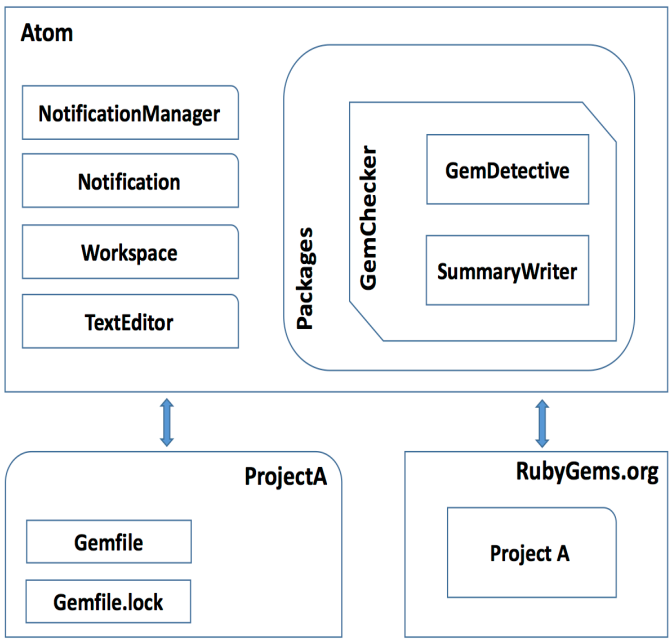

Figure 1 – GemChecker Architectural Model

IV. EVALUATION

We evaluated GemChecker in terms of both usability and performance. This section describes both evaluation processes in detail.

A. Usability Evaluation

As part of the usability evaluation, the 6 developers involved in the requirements elicitation process were invited to take part in a usability laboratory study designed to measure the effectiveness of GemChecker and the user satisfaction with the tool. The six participants were asked to follow a pre-defined scenario, written to ensure that all the features of GemChecker are included in it. The scenario is described below, and it uses a Ruby on Rails medium size project all 6 developers are familiar with.

Step 1 – Check Gem Details Feature: *You decide to investigate what the function of the "Devise" gem is. Please locate the gem in the Gemfile. Use GemChecker to obtain the details for this gem*.

Step 2 – Check Gem Maintenance Feature: *Locate the gem Devise within the Gemfile. Check the gem's maintenance status.*

Step 3 – Check all Gems Versions Feature: *Check the gem versions of all the files in the Gemfile.*

Step 4 – Generate Summary Report Feature: *Generate the Gemfile summary report for the project.*

After following the scenario, all participants were asked to answer a set of questions. These questions were of two types:

a) Effectiveness related. These 11 queries ask for partial results participants would have obtained as a result of following the scenario. They are all similar in natural and some examples include: "What is the latest version of the Devise gem?", "Is the "Starburst" gem well maintained?", "How many gems in total are out of date?", and "How many gemas are 5 or more versions out of date?". Incorrect answers to these queries would signal issues with the effectivenss of the tool when in use. The results show that all the 6 participants were able to answer all the 11 queries correctly.

b) User satisfaction related: These 15 questions ask participants to rate on a scale from 1 to 4 (maximum) the level of helpfullness and ease of use for each of the features of GemChecker in the context of the scenario followed. Aditionally, participants were asked to rate and comment on the tool as a whole.

Based on these answers, we identified the following results:
R1: 66.7% of the participants found the Check Gem Details feature very easy to use and very helpful.
R2: 100% of the participants found the Check Gem Maintenance feature easy to use, with 83.3% of them finding the feature very helpful.
R3: 66.7% of the participants found the Check all Gems Versions feature very easy to use, with 83.3% of them finding the feature very helpful.
R4: 66.7% of the participants found the Generate Summary Report feature very helpful, with only half of the participants finding the feature very each to use.

Overall, 66.70% of the participants found GemChecker very easy to use, and 83.30% admitted they would be likely to use this tool as part of their day to day work moving forward. The most popular feature was the Generate Summary Report, followed by the Check all Gems Versions. The comments participants gave are all positive, with some considering GemChecker as being a unique solution for a significant challenge Ruby on Rails developers face in maintaining their applications: "*I'm not sure if other ways to check gem versions in bulk exist – using the Atom package was a lot easier than it would be doing it in the command line, too*".

B. Performance Evaluation

We selected the 50 most popular Ruby on Rails repositories on Github and used them as case studies for evaluating GemChecker. The total number of gems used by each project varied from 1 (minimum) to 232 (maximum), with an average number of gems of 35.02 per project. For each project, we generated a Summary Report (see Requirement R4). As part of this evaluation we measured:

a) Efficiency: defined as the time required for generating the Sumary Report for each project. A timer was set to automatically measure the time required to generated each report. The average time required to generate the reports was 6 seconds per report, with 21.5 seconds maximum (for project associated with 232 gems), 8 milliseconds minimum (for project associated with 2 gems), and a median value of 1.76 seconds per report.

b) Effectiveness: defined as the accuracy of the generated summary. The first author manually validated the results generated by GemChecker for all the 50 projects considered. The accuracy of the results generated by GemChecker is 100%.

The number of out of date gems varied from project to project, with an average of 65.80% of the gems being out of date per project. The number of versions between the latest version released for a gem and the version currently used as part of the project varied as well, with an average of 35.72% of the gems per project being 5 or more versions out of date. Overall, 2 out of the 50 projects (4%) we used as a case study were not using any out of date gems. These projects were relatively small, however, one using 1 gem and the other one using 2. 10% of the projects we analyzed were using out of date gems exclusively. While more can be done to further explore and explain these results, we are satisfied with the performance of GemChecker.

V. Related Work

Tool support for managing and capturing API changes is available [4], [5]. These tools are designed to support API maintainers and rarely address the concerns of third party developers who maintain projects which use these APIs. Additionally, the majority of these tools and their approaches focus on a limited number of platforms and languages, mainly object-oriented languages such as Java or web-based libraries. Guidelines on maintaining and managing API changes are also targeting APIs maintainers [6], with little support being provided to API clients and their work practices. Previous work has shown that developers find out about new APIs through web search [1]. The process is entirely manual and it relies mostly on software development blogs such as Stack Overflow. Similarly, code deprecation in APIs used is only discovered based on developers exploring external resources such as development forums. Zhou et al. introduce the Deprecation Watcher, a software tool that alerts developers of deprecated APIs in web based code examples [1]. The design of the tool if informed by the analysis of 26 open source Java systems and their 690 versions which looked at how deprecation is used in practice and how it evolves over time.

The dilemma between migrating to a newer version of an API and exploring the cost required by such migration or not migrating and exposing the system to the bugs and the security risks that come with this is addressed in [2]. Research suggests that the majority of the changes identified in the APIs analysed could not be automatically translated into the client code, and required significant additional effort from the side of either the API maintainer or the client. Additionally, identifying incompatibilities between the APIs used as part of the project when migrating these third-party software libraries to a newer version is a challenge. Tool support in this area includes VerXCombo [8], an interactive data visualization tool which identifies best-fit library combinations using the 'wisdom of the crowd' popularity metrics. Similarly, Dependabot [9] is a GitHub add-on that opens individual pull requests for each outdated dependency file identified as part of a GitHub repository. GemChecker complements such tools by integrating directly with a code editor, and allowing developers to identify and manage out-of-date third-party libraries used by their projects.

VI. Conclusions and Limitations

This paper introduces GemChecker, a tool developed as an answer to the day to day challenges Ruby on Rails developers face when maintaining complex software systems that rely on large numbers of gems. The lack of any support in identifying changes or new versions of these gems led us to collaborate with a small size team of Ruby on Rails developers in eliciting the requirements of GemChecker. The tool alerts and informs developers of changes in the gem versions of the gems used by the projects they maintain, supporting a number of maintenance scenarios.

GemChecker currently only works for Ruby on Rails projects with Atom integration. Future work will look into expanding GemChecker to work with other languages and possibly other code editors. The option of using GemChecker as a stand-alone appliation is not yet implemented as this was not one of the requirements of the developers we worked with. However, the scenarios for using GemChecker as a stand-alone application are being explored, together with the option of integrating GemChecker with other IDEs, such as Eclipse.

References

[1] Jing Zhou and Robert J. Walker. 2016. API deprecation: a retrospective analysis and detection method for code examples on the web. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016). ACM, New York, NY, USA, 266-277.

[2] Bradley E. Cossette and Robert J. Walker. 2012. Seeking the ground truth: a retroactive study on the evolution and migration of software libraries. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE '12). ACM, New York, NY, USA, , Article 55 , 11 pages.

[3] http://rubygems.org

[4] Johannes Henkel and Amer Diwan. 2005. CatchUp!: capturing and replaying refactorings to support API evolution. In Proceedings of the 27th international conference on Software engineering (ICSE '05). ACM, New York, NY, USA, 274-283.

[5] Ittai Balaban, Frank Tip, and Robert Fuhrer. 2005. Refactoring support for class library migration. In Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '05). ACM, New York, NY, USA, 265-279.

[6] Joshua Bloch. 2006. How to design a good API and why it matters. In Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications (OOPSLA '06). ACM, New York, NY, USA, 506-507.

[7] ATOM: https://atom.io

[8] Yuki Yano, Raula Gaikovina Kula, Takashi Ishio, and Katsuro Inoue. 2015. VerXCombo: an interactive data visualization of popular library version combinations. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension* (ICPC '15). IEEE Press, Piscataway, NJ, USA, 291-294.

[9] Dependabot: https://dependabot.com/