



IssuePlayer: An extensible framework for visual assessment of issue management in software development projects

Vahid Garousi *, James Leitch

Software Quality Engineering Research Group (SoftQual), Department of Electrical and Computer Engineering, University of Calgary, 2500 University Drive NW, Calgary, AB Canada T2N 1N4

ARTICLE INFO

Article history:

Received 25 September 2009

Received in revised form

24 February 2010

Accepted 7 March 2010

Keywords:

Software development projects

Software visualization tools

Issue management

ABSTRACT

This article presents a software visualization framework which can help project managers and team leaders in overseeing issues and their management in software development. To automate the framework, a dashboard tool called IssuePlayer is developed. The tool is used to study the trends in which different types of issues (e.g., bugs, support requests) are submitted, handled and piled up in software projects and use that information to identify process symptoms, e.g., the times when the code maintenance team is not responsive enough. The interactive nature of the tool enables identification of the team members who have not been as active as they were expected to be in such cases. The user can play, pause, rewind and forward the issue management histories using the tool. The tool is empirically evaluated by two industrial partners in North America and Europe. The survey and qualitative feedback support the usefulness and effectiveness of the tool in assessing the issue management processes and the performance of team members. The tool can be used complementarily in parallel with textual information provided by issue management tools (e.g., BugZilla) to enable team leaders to conduct effective and successful monitoring of issue management in software development projects.

Crown Copyright © 2010 Published by Elsevier Ltd. All rights reserved.

1. Introduction

Most software development projects are developed iteratively with constant feedback from different stakeholders, e.g., developers, clients, users, managers, etc. Recorded feedback from stakeholders is usually referred to as *issues* in software development projects.

Most software development projects have issue repositories. For example, SourceForge (www.sourceforge.net) provides a tool called “Tracker” which allows the project members to manage four types of issues: bugs, support requests, feature requests and source code patches [1]. SourceForge is often referred to as the world’s largest

online repository for Open-Source Software (OSS) development [1] and provides various tools that allow open-source projects to develop, distribute and support their software projects.

It is important for software project managers and team leaders to track, monitor and assess different aspects of the issue management processes, such as their performance, timeliness, responsiveness, efficiency and quality [2].

There are numerous good issue management tools in the market which are either commercial (e.g., Help Desk [3] and EnterpriseWizard [4]) or free/open-source (e.g., BugZilla [5] and Tracker [1]). Although these tools help the development and support teams to receive, analyze and address issues, they however provide little or no functionality to help project managers and team leaders to track, monitor and assess issue management processes from a higher level perspective (as discussed above).

* Corresponding author.

E-mail addresses: vgarousi@ucalgary.ca, rickvanprim@gmail.com (V. Garousi).

URL: <http://www.softqual.ucalgary.ca> (V. Garousi).

Table 1

Mapping of the proposed tool along the classification scheme proposed by Maletic et al. [9].

| Tool | Task | Audience | Target | Representation | Medium |
|-------------|--|------------------------------|--|---|---------------|
| IssuePlayer | Process evaluation and improvement, team productivity evaluation | Project manager, team leader | Issue repository (database) | Issue and developer bins, timelines, animations of issues coming in and going out, different color intensities for different issue priorities | Color monitor |
| SHrimP | Reverse engineering, maintenance | Expert developer | Source code, documentation, static design-level information, medium Java systems | 2D graphs, interactive, drill-down | Color monitor |

Similar to other types of textual data, it is usually very challenging to analyze issue management processes based on only textual information in an effective and efficient manner, for example, when project managers want to detect process symptoms such as slow reaction of their team to incoming issues. This is especially more challenging when there exist huge amounts of issue data logs (tens of thousands of issues per year for large-scale projects). This challenge is an opportunity where visualization of issue management processes can be helpful, in which a visual representation of data in issue repositories is abstracted out in an effective and efficient schematic form [6].

To achieve the above goals, we are proposing a software visualization framework. To automate this framework, we present *IssuePlayer*, which is a dashboard tool intended for project managers and team leaders overseeing software development. In its current version, the *IssuePlayer* tool provides various features to help managers and team leaders in different tasks, e.g., automatic or manual detection of process symptoms. Ideas for adding several additional useful features are also discussed in this article. It visualizes historical information of incoming and handled issues during the project's lifetime in a GUI environment. The user can play, pause, rewind and fast-forward the historical issue management processes. It also efficiently displays information about the developers and issues.

The proposed visualization tool can be used to study the trends in which different types of issues are submitted, handled and piled up in software projects and use that information to identify process *smells* (symptoms) and suggest guidelines to potentially improve issue management processes. Just as source code can have “bad smells” [7] that suggest possible source-code problems that may need to be addressed by developers, there can be different types of process smells indicating that a team needs to rethink its process, whether a development, issue management, or any other type of software process. For example, two process smells identified in the software engineering community are [8]: (1) poor communication/collaboration within the team, and (2) not having the right people involved with the project.

To empirically evaluate the proposed tool, we have worked with two of our industrial partners to use the tool to analyze their issue repositories, evaluate and improve their processes, and also to evaluate their teams' productivity. Both the quantitative and qualitative results from

the empirical study support the usefulness of the proposed tool in assessing the performance and efficiency of development processes and identifying potential process symptoms.

To classify our proposed framework inside the family of software visualization tools, we classify it using an existing classification scheme for software visualization tools [9], as shown in Table 1. The scheme includes five dimensions: task, audience, target, representation and medium. Another software visualization tool called SHrimP as classified by [9] is also given for reference.

The remainder of this article is structured as follows. A survey of the related work is presented in Section 2. The *IssuePlayer* visualization framework and tool are proposed in Section 3. The usage scenarios of the framework and tool are discussed in Section 4. The software architecture and usage of the proposed toolset are presented in Section 5. Two empirical case studies using the tool are discussed in Section 6. Finally, Section 7 concludes this article and discusses some future work directions.

2. Related work

The subject area of the current work is in the intersection of visualization of software processes, and issue management in the context software development projects.

Issue repositories usually contain a rich amount of information which can be used to perform different types of analyses to mine useful findings about team (group) dynamics, development processes, issue management processes and so on [10]. Mining Software Repositories (MSR) [10] is a very active research area which is gaining increasing popularity in both academia and industry. However, there are only a handful number of works in the MSR literature relating specifically to software processes, e.g., [11–13], while most works focus rather on other subjects such as source code repositories, and social networks among developers [10].

To the best of the author's knowledge, there is no existing work to visually mine the issue repositories with the intent to highlight the issue pile and developer/testers performance. Two somewhat related work are *SimSE* [14] and *CodeSwarm* [15].

SimSE [14] is an educational software engineering game-like simulation environment that allows students to practice a “virtual” software engineering process or sub-

process in a graphical, interactive, and fun setting in which direct, graphical feedback enables them to learn the complex cause and effect relationships underlying the processes of software engineering.

CodeSwarm [15] is another tool which shows the history of commits in a software project. A commit happens when a developer makes changes to the code or documents and transfers them into the central project repository. Both developers and files are represented as moving elements. When a developer commits a file, it lights up and flies towards that developer. Files are colored according to their purpose, such as whether they are source code or a document. If files or developers have not been active for a while, they will fade away. Project issues, as defined in Section 1, are not taken into account in this tool.

Another similar area of research is software process simulation [16,17] and systems dynamics [18]. While interesting and useful, works in these bodies of knowledge study the dynamics of software development by only simulating the entire development process. There is no existing work in those areas to study the software process by replaying a historical data set of issue repositories.

3. IssuePlayer visualization framework

The IssuePlayer visualization paradigm is discussed in Section 3.1. We then present in Section 3.2 the IssuePlayer's toolset which implements the framework.

3.1. Visualization paradigm

The IssuePlayer framework is a visual animation of flows of incoming and handled artifacts, as well as the involved people (team members) during the lifetime of a software development project. An artifact here denotes the general terminology of software artifacts, e.g., requirements or design documents, bugs, use-cases, source code modules and support requests. A conceptual visualization of the framework is shown in Fig. 1. The idea is to visualize the project managers and team leaders, which project artifacts were submitted to be processed in each time instance during the project, which team members, e.g., developers, testers, were more active in working on those artifacts, and also how long the artifacts were waiting in the pool before they were handled, i.e., implemented.

The controllable animation makes it more effective, interactive, and also attractive to observe and study the history of what has actually happened during past weeks and months of a project. Among many features, the user, i.e., project manager or team leader, can pause the animation at any time, rewind or fast-forward it, can control its speed, and can also hover the mouse over each of the team members' photo or each artifact to see the details.

The photos of team members are automatically resized (shrink or grow) depending on their contribution to each activity, e.g., bug submission and bug fix. For example, consider the first developer under activity 1, i.e., bug submission, in Fig. 1. If this team member has actively submitted more bugs in the past, his/her photo will

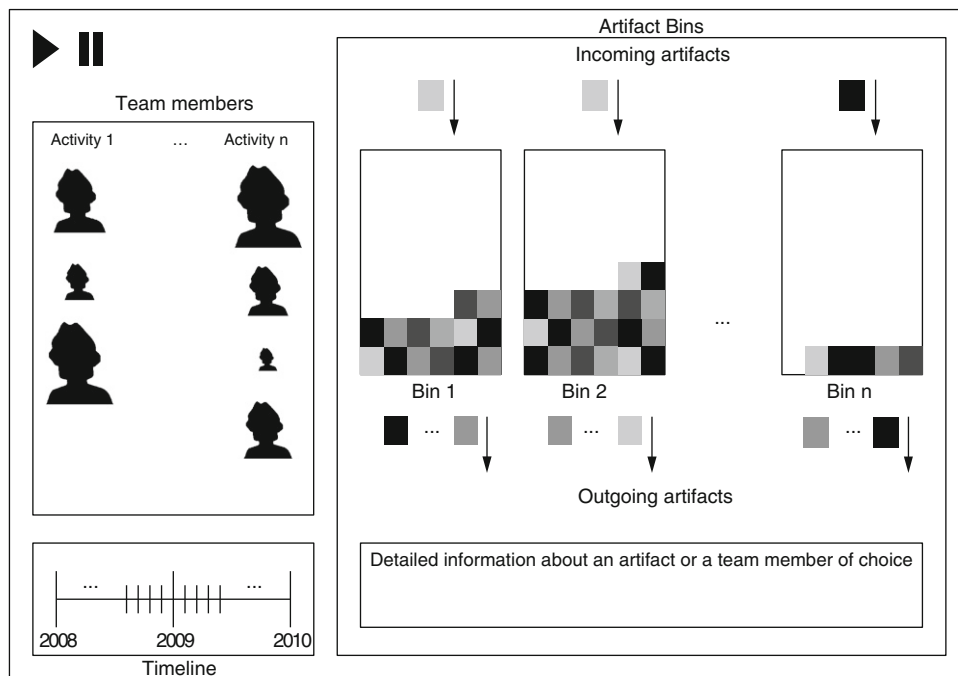


Fig. 1. A conceptual visualization of the IssuePlayer framework.

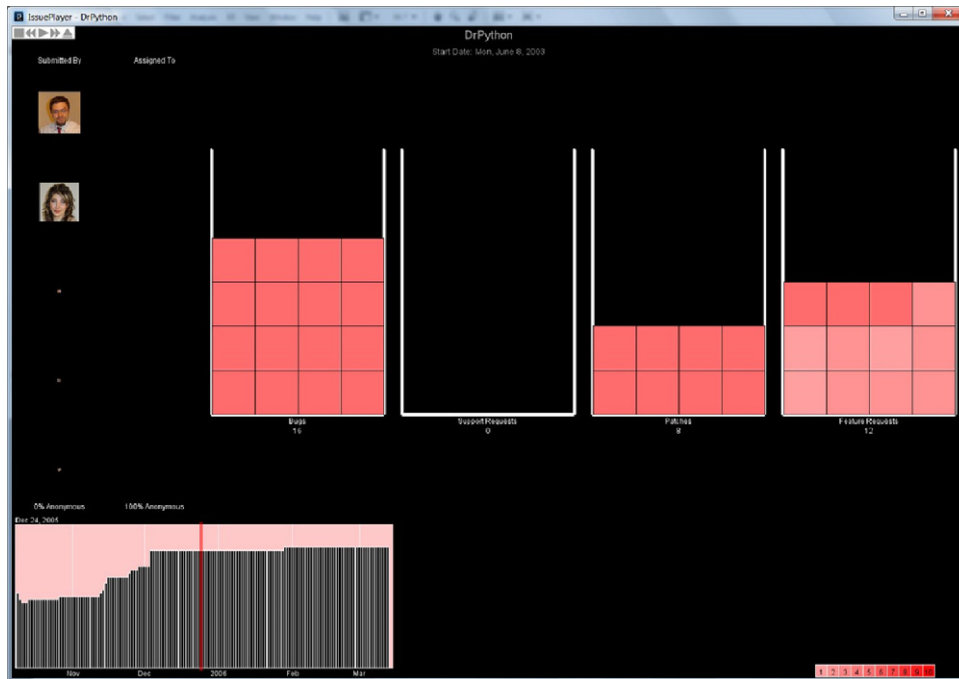


Fig. 2. . User interface of the IssuePlayer tool.

slightly increase in size, denoting the presence of that particular team member in bug submissions tasks. This animated notion can help the team leads to effectively extract such qualitative information without having to perform various searches in the text-based log information of the bug database and trying to determine the extent to which a given team member has been active in doing the task(s) assigned to him/her.

The artifact bins render artifacts' opening and closing by flowing them into their respective bins. The color intensity of each artifact denotes its priority or severity, e.g., priority of a requirements document to be developed, or the severity of a bug. This allows the user to visually track if more intense artifacts were actually addressed first. User can click on any artifact or team member of choice, and the system will "lock in" on that entity showing the detailed relevant information in the detail information panel.

The timeline shows the number of open issues on a given day as well as on the surrounding days. When the time increment is changed from days to weeks to years, the timeline re-adjusts itself accordingly. Hovering on a bar on the timeline activates a tool-tip to display indicating the number of artifacts represented by that bar.

3.2. IssuePlayer tool

To implement the framework, as the first proof-of-concept tool, we have developed a Java tool called IssuePlayer [19]. As a concrete example of the framework, a snapshot of the IssuePlayer is shown in Fig. 2. Development details of IssuePlayer and its associated toolset are presented in Section 5. The entire IssuePlayer

toolset is available as open source under the GNU General Public License (GPL) online [19].

In this implementation, the tool has been developed to support the four issue types of SourceForge: bugs, support requests, feature requests and source code patches [1]. Note that the tool is easily generalizable to other software artifacts. For example, one can adopt and customize the IssuePlayer's source code to apply it to higher level software development life cycles, and to support artifacts such as requirements and design documents. Such artifacts can have their own "Bins" and their incoming and outgoing flows can be animated in similar fashion as it is currently supported for four issue types.

Fig. 2 shows a snapshot of the tool animating the issue flows across the lifetime of an open-source project called *DrPython* (<http://drpython.sourceforge.net>). The Developer Bins display the pictures of the top 5 developers who have submitted issues and also the top 5 developers who have been assigned to fix issues. Each developer's picture¹ will grow/shrink depending on their current amount of issues relative to the amount of issues of the other top developers. IssuePlayer has the capability to read the graphic files of developers' pictures in JPEG format from a specific directory and use them in the visualization. The tool's online user manual specifies more details about such features [20].

Recall from Section 1 that SourceForge's Tracker system supports four predefined issue types: bugs,

¹ Note that the developers' pictures shown in Fig. 2 do not correspond to the real developer user IDs/names as queried from the SourceForge. They are a collection of photos from the personal gallery of the author (used by permission).

support requests, feature requests and source code patches [1]. However, developers of a project can define additional issue types if they need to. The jEdit's development team has defined five more issue types, e.g., *Plugin Bugs*, *jEditLauncher Bugs*. IssuePlayer is capable of automatically loading extra bins for extra issue types if they have been defined for a project (as shown in Fig. 4). It seems the jEdit's development team has done this to categorize bugs in further detail to perhaps better manage them. In such a case, IssuePlayer automatically re-adjusts the display and shows all the issue bins accordingly.

4. Usage scenarios

To discuss the usage scenarios of IssuePlayer, we choose in Section 4.1 two real-world example issue repositories and analyze them using IssuePlayer. Using the example issue repositories, we then discuss five usage scenarios:

- Automatic or manual detection of process symptoms (Section 4.2)
- Evaluation of personnel and team performance (Section 4.3)
- Identification of problematic components and releases (Section 4.4)
- Simulation and training of Improvements in issue management processes (Section 4.5)
- Application to other software artifacts (Section 4.6)

For each usage scenario, we also discuss why the visualization/animation provided by IssuePlayer can be better, i.e., more effective, than static textual information provided by traditional issue management tools.

4.1. Analysis of two example issue repositories

To discuss the usage scenarios of the framework and tool, consider the three snapshots from the GUI tool as shown in Figs. 3 and 4. For illustrative examples, two active SourceForge development projects, named *jEdit* and *DrPython*, are chosen. For the interested reader, the key information and statistics about the two projects are shown in Table 2. According to two earlier studies [21,22], these two projects are two of top active projects hosted in SourceForge in terms of issue activities, and also user downloads.

The snapshot in Fig. 3 shows a session in which a user is analyzing the issue management process of *DrPython* project. The snapshot is taken at one particular day during the project's lifetime, i.e., December 24, 2005. The user has clicked on an issue and can see the details of the issue as extracted from the issue repository database. The two snapshots in Fig. 4 show two different time instances during the development lifecycle of the jEdit project [23].

4.2. Automatic or manual detection of process symptoms

By using IssuePlayer, project managers and team leaders can notice the problem spots of their team's issue handling process. For example, they can find the past periods in which many more bugs were submitted than were fixed. This might reveal that the debugging team, perhaps the same as the development team, had a long back-log of other tasks/issues and did not manage to address those bugs. Or, the team did not function very effectively or efficiently. Such feedback can be used to make changes, e.g., personnel, in the team, or revise and (possibly) strengthen the issue management processes. Also, workforce re-alignment and/or other managerial changes might be considered.

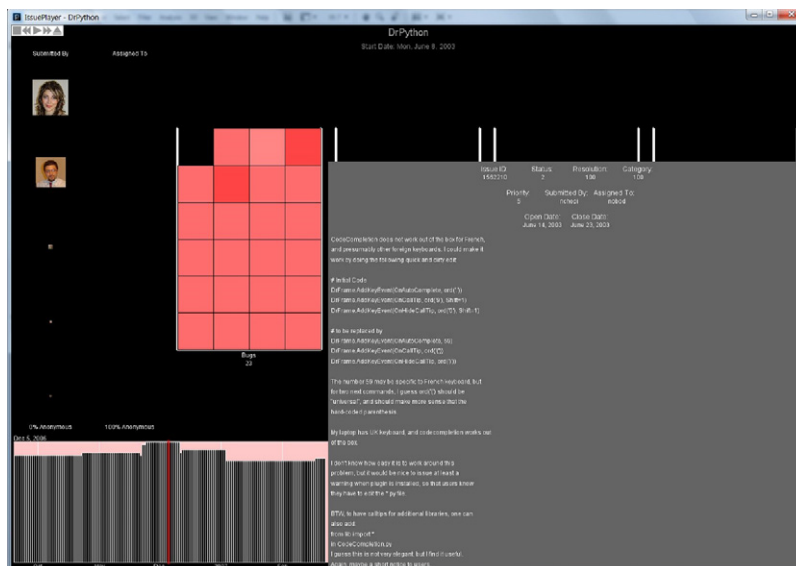


Fig. 3. Analyzing the issue management process of DrPython using IssuePlayer.

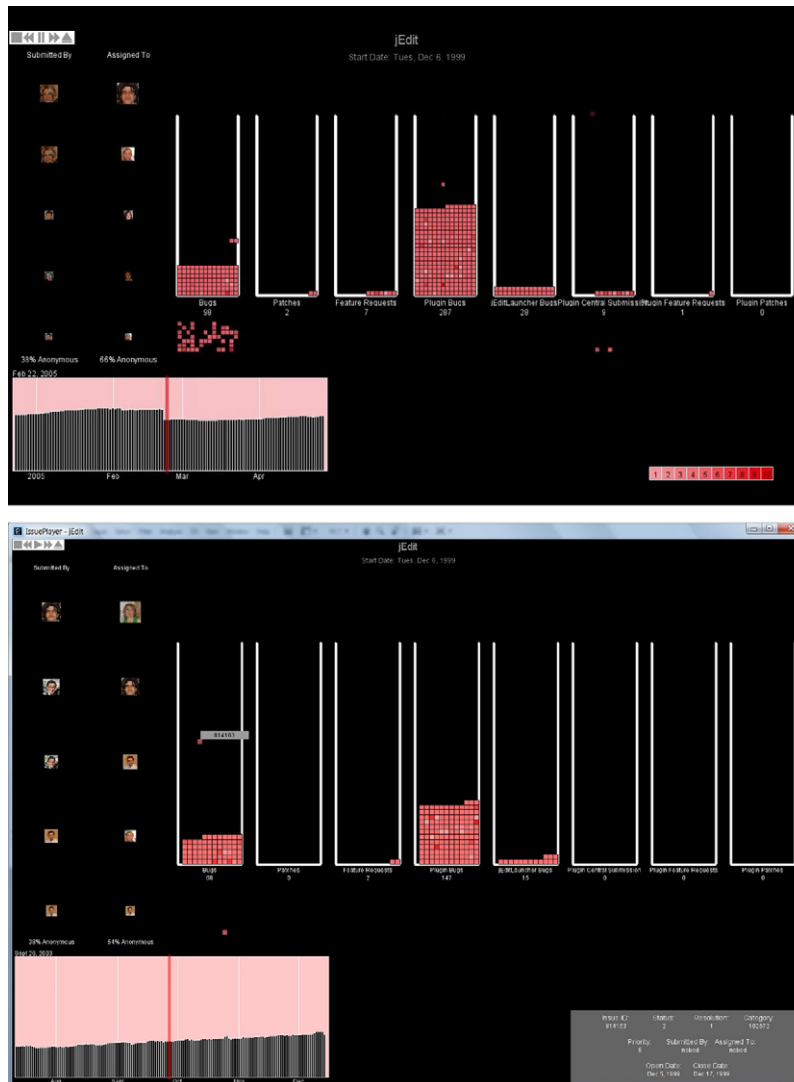


Fig. 4. Analyzing the issue management process of jEdit using IssuePlayer.

Table 2

Key information and statistics about the two projects jEdit and DrPython.

| Attribute | Projects | |
|---|--|--|
| | jEdit | DrPython |
| Programming language | Java | Python |
| KLOC (Kilo line of code) | 71 | 15.4 |
| Latest version (as of September 2009) | 4.2 | |
| Registered in SourceForge (inception date) | December 6, 1999 | June 8, 2003 |
| Number of project administrator ^a | 9 | 2 |
| Number of developers ^a | 158 | 4 |
| SourceForge activity percentile ^a | 99.98% | 97.84% |
| Average number of issues submitted (per month) ^a | 52.15 | 3.21 |
| Average number of issues closed (per month) ^a | 46.28 | 3.14 |
| Average number of downloads (per month) ^a | 45,112 | 2,171 |
| Website | sourceforge.net/projects/jedit www.jedit.org | drpython.sourceforge.net |

^a The statistical data are as of September 2008.

We discuss next two related major process symptoms in detail and how the tool can be useful in detecting them.

4.2.1. Slow reaction to issues

Recall that the snapshots in Fig. 3 belong to an open-source software (*DrPython*). There have been numerous studies arguing that the process nature of open-source and proprietary software development projects can be similar and different depending on the project [24]. Let us suppose that the snapshots correspond to a commercial software development project. Once the IssuePlayer animation is played around the time instant shown in Fig. 3, user can observe that the project had a large wave of incoming issues in late 2005, and the team was not very proactive in handling them in a timely manner (especially, the issues of type bugs and patches). This is since the bugs and patches bin have 16 and 8 items, respectively, in December 24, 2005. To observe how this process symptom arises in an animated format, we invite the reader to see the online videos [19].

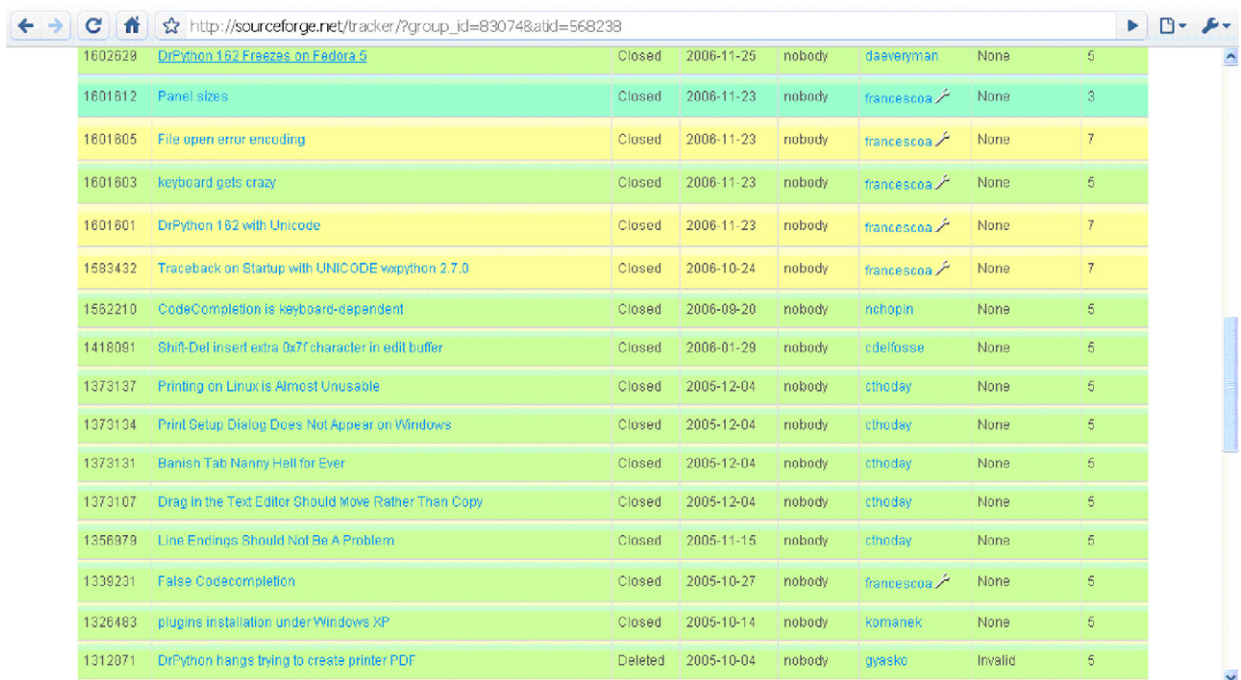
In other words, the user can visually see that days pass and no or few issues are being addressed. This would make the user wonder why the debugging team is so slow on those days. It is also possible to review the issues which are addressed and those which are being ignored. More importantly, rewinding and fast-forwarding the animation provides an effective process inspection tool to pinpoint exactly which issues were the most challenging ones to handle. To motivate our discussion of the animation provided by IssuePlayer, let us consider a snapshot of latest *DrPython* bugs as shown in Fig. 5.

Although textual tables provide a detail view of each issue and its status, usually managers are interested in a

high-level view of the process rather than detailed information. Consider, for example, a large project with hundreds of issues coming in and being resolved every day. Browsing through detailed textual tables in such a case can be very tedious and would not easily provide a manager with information on the times when the team was responsive and when it was not. In consultation with our industry partners and asking them to empirically evaluate IssuePlayer, they confirmed that visualization helps a lot in this aspect. The tool also enables the user to view the detailed description of an issue whenever needed. At any time during the animation of the issues flow, the user can pause the animation, and view the details of each issue. One such example is shown in Fig. 3.

Furthermore, the interactive nature of the IssuePlayer tool enables identification of the team members who have not been as active as they were expected to be. This is a feature that is not easily provided by simple 2D graphs. For example, let us consider Fig. 6 which shows the trends of monthly opened and closed issues versus release numbers of the *jEdit* project [23]. This graph is taken from one of our recent works [26] in which we investigated the success factors of open-source software projects across their lifetime.

Although these types of 2D graphs provide high-level “aggregate” information of how the entire team has reacted to incoming issues, however, it does not provide details on how each developer has handled each incoming issue assigned to him/her, and also which types of issues were handled sooner compared to other types. The order in which issues are chosen to be solved is a complex decision by itself. Meaningful issue priorities are usually chosen by users, but in reality decisions to choose issues



| ID | Title | Status | Date | Reporter | Assignee | Priority | Count |
|---------|--|---------|------------|----------|------------|----------|-------|
| 1602629 | DrPython 162 Freezes on Fedora 5 | Closed | 2006-11-25 | nobody | daaveryman | None | 5 |
| 1601812 | Panel sizes | Closed | 2006-11-23 | nobody | francescoa | None | 3 |
| 1601805 | File open error encoding | Closed | 2006-11-23 | nobody | francescoa | None | 7 |
| 1601803 | keyboard gets crazy | Closed | 2006-11-23 | nobody | francescoa | None | 5 |
| 1601801 | DrPython 162 with Unicode | Closed | 2006-11-23 | nobody | francescoa | None | 7 |
| 1583432 | Traceback on Startup with UNICODE wpython 2.7.0 | Closed | 2006-10-24 | nobody | francescoa | None | 7 |
| 1562210 | CodeCompletion is keyboard-dependent | Closed | 2006-09-20 | nobody | nchopin | None | 5 |
| 1418091 | Shift-Del insert extra 0x7f character in edit buffer | Closed | 2006-01-29 | nobody | cdelfosse | None | 5 |
| 1373137 | Printing on Linux is Almost Unusable | Closed | 2005-12-04 | nobody | cthoday | None | 5 |
| 1373134 | Print Setup Dialog Does Not Appear on Windows | Closed | 2005-12-04 | nobody | cthoday | None | 5 |
| 1373131 | Banish Tab Nanny Hell for Ever | Closed | 2005-12-04 | nobody | cthoday | None | 5 |
| 1373107 | Drag in the Text Editor Should Move Rather Than Copy | Closed | 2005-12-04 | nobody | cthoday | None | 5 |
| 1356979 | Line Endings Should Not Be A Problem | Closed | 2005-11-15 | nobody | cthoday | None | 5 |
| 1338231 | False Codecompletion | Closed | 2005-10-27 | nobody | francescoa | None | 5 |
| 1326483 | plugins installation under Windows XP | Closed | 2005-10-14 | nobody | komanek | None | 5 |
| 1312071 | DrPython hangs trying to create printer PDF | Deleted | 2005-10-04 | nobody | gyasko | Invalid | 5 |

Fig. 5. A list of latest bugs in the online issue tracker of *DrPython* [25].

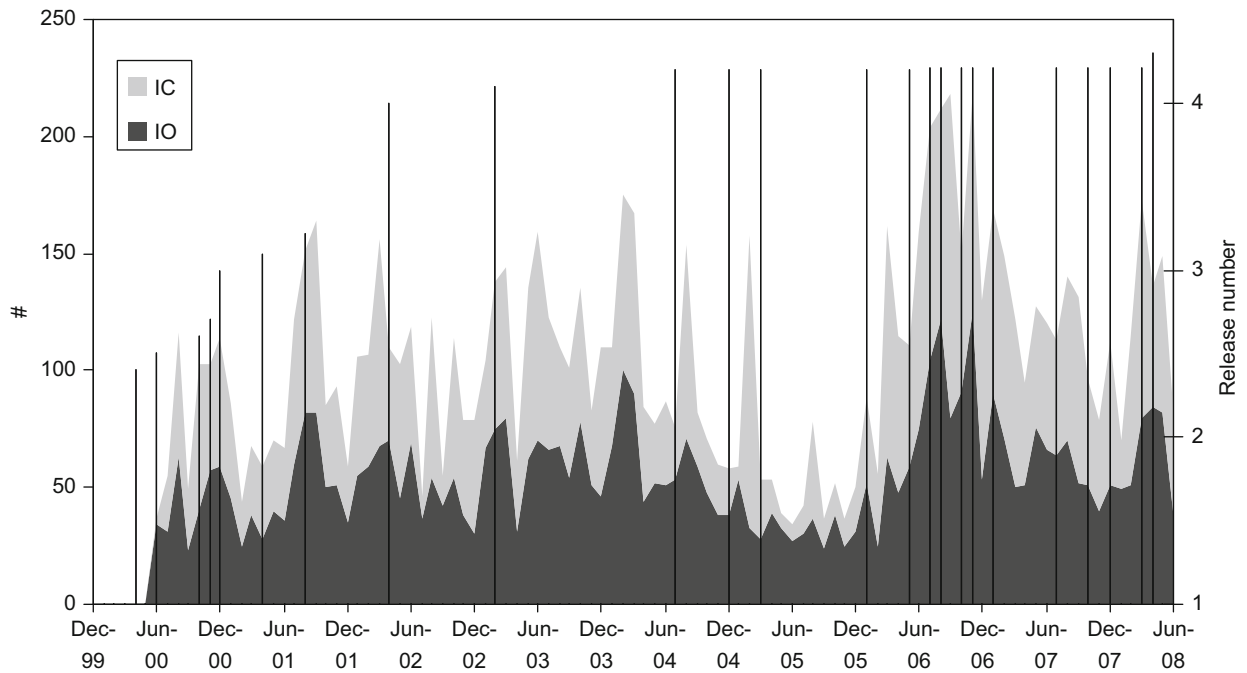


Fig. 6. Trends of monthly opened (IO) and closed issues (IC) versus release numbers of the jEdit project [23].

to solve is based on complex ad-hoc rationale, e.g., when a client sends an email saying that he has a serious business problem that a bug has caused for him, the project manager might decide to urgently move forward with fixing that issue even if it has a lower priority value compared to other issues.

Analysis of such team behavior and issue choices becomes more challenging for large teams such as the jEdit's team which has nine project administrators, and 158 developers (as of September 2008) [23].

The above discussion highlights the usefulness of IssuePlayer compared to textual or simple 2D graphs when different team members (e.g., testers) have been involved with the project in different time periods, and the incoming issues have had different priorities. In such a case, process improvement and/or team realignment decisions have to be made by having a holistic view on the process in which both issues, their priority values, and team members should be considered at the same time.

In summary, we do not recommend to replace the detailed textual information of issues with the IssuePlayer tool, but we note that IssuePlayer is a complementary tool to provide a higher-level of abstraction for stakeholders. This is similar in analogy to the case when software engineers review software design models, e.g., those in UML, rather than source code to grasp an idea of how a given software system has been implemented.

4.2.2. Pile up and aging of issues

Perhaps, the most critical process symptom in this context is to have too many pending (piled up) issues. Such a symptom might lead to many negative symptoms in the long run for a project, e.g., users (clients) will find

their issues not addressed on-time and might lose interest in the project, or in the case of industrial projects, they may decide to cancel their contract with the software company. Also, the symptom can lead the issues to pile up in larger and larger numbers, requiring more and more efforts to handle them all. Note that although the projects presented so far are OSS projects, but similar trends are also observed in commercial software projects, as per discussions with our industrial partners.

To investigate this smell, the average age of each issue type can be studied, i.e., how long (in days) issues have been waiting in the pool to be processed (closed). This metric is provided by the Tracker statistics feature in SourceForge. For example, assume there are two pending issues in a project. If the first issue is submitted on July 1st, and the second one on August 1st of the same year, the Average Issue Age (AIA) on September 1st will be: $(31+62)/2=46.5$ days.

We can also differentiate different issue types and calculate the following metrics:

- AIA: Average Issue Age
- ABA: Average Bug Age
- APA: Average Patch Age
- AFA: Average Feature request Age

To automate the detection of issue-pile-up smell, one can easily develop software agent inside IssuePlayer. While the issues are being played along the time axis, if at any time instant, the average issue age of choice exceed a predefined threshold (e.g., 50 days), the tool may stop the animation and raise a "red flag" with the appropriate

message to the user. This can be useful in both online and offline usage of IssuePlayer. In the case of online (live) usage, the user can then take the appropriate action, e.g., send a message to the debugging/maintenance team. This will enable the project managers in monitoring their projects and ensuring that the issue submission and handling tasks proceed as planned.

If the tool is used in offline mode (i.e., reviewing the historical issue data), the manager can extract lessons learned with respect to the team's process, and think of applying process improvement practices.

4.3. Evaluation of personnel and team performance

One of the major usages of the tool is to review and evaluate personnel and team performance during a past period. Analysis and improvement of software development team performance have been identified as two of the most critical and challenging undertakings in the area of software engineering [27,28].

For example, the developers' bins in Figs. 3 and 4 show the top 5 team members in each of the issue submission and issue handling categories. Such information can be used for performance assessment, awards, promotion, swapping team members or other types of purposes.

As an example, the top snapshot in Fig. 4 shows a time instant in which suddenly a large number of bugs were addressed. Close inspection of this case and discussions with team members might reveal a positive team dynamics in that particular point of time, i.e., a few team members might have played the roles of champions in aggressively solving a large number of pending issues.

In the current implementation of the tool, two categories of users are shown: issue submission and issue handling. For the case of bugs, the issue submission category can help to identify the most active testers who are submitting bug reports. The visualization of the issue handling category would reveal the most active bug fixers in this case.

4.4. Identification of problematic components and releases

Visualization and animation using IssuePlayer can also be used to locate which particular releases or components were problematic, i.e., they brought a large wave of new bugs, support requests, etc.

The user can have the release timeline of the project on his/her desk and compare the historical issue flow to that timeline. This can reveal which releases or components were the most *buggy* (non-stable), as one can visually see which release brought a larger wave of new bugs and each of those bugs and their submitters, i.e., from the project client's team, can be studied one by one. At the same time, the reaction of the team to those waves of issues can be evaluated.

Furthermore, as per a feature request raised by one of our industrial partners, it is important for team leaders to quantitatively evaluate how problematic (e.g., defective) each component has been at each point in time in the past. To automate the above task, we are now in the

process of adding a new visual feature to provide visual schematics for components with respect to their level of being defective. We have defined the following metric which can be calculated automatically by IssuePlayer:

$$\text{BugIndex}(\text{component } c, \text{time } t) = \frac{\text{Number of open bugs for component } c \text{ at time } t}{\text{Total number of open bugs at time } t}$$

The above metric is being used to visualize an intensity-based (also called thermal) schematic for each component. An example GUI currently under development is shown in Fig. 7, where the component names are those of the jEdit software [23]. In this updated GUI, compared to Fig. 1, the right-most components bin has been added. As the animation moves forward, the bug intensity level of each component changes (either towards full red, or towards white) denoting the normalized ratio of current open bugs submitted for each component. In one extreme, the above metric value is 1, meaning that all open bugs are for one single component (denoting a very problematic component). In the other extreme (value=0), the measure denotes that the component did not have any open bugs in the particular time, i.e., it is quite reliable.

All the above time-sensitive information and observation would provide the analyst with a sense of how each component gets more or less reliable (stable) with the passage of time and which team members have been involved in identifying and solving the associated bugs. Such information is especially vital for the testing and quality assurance (QA) team leaders [29].

4.5. Simulation and training of improvements in issue management processes

Note that, as we have discussed so far, the main intent of IssuePlayer has been to play the historical issue flows and to study team members' degree of involvement in the process. Another major usefulness of the tool is to allow the team leader to evaluate the outcome of an artificial improvement to a given historical issue management data set. By an artificial (versus real) improvement, it is meant that the change is not real, but rather in a laboratory setting.

For example, after a team leader has played a historical issue data set, he/she might wonder that it would have been better if a less productive debugger developer was replaced by a more productive one, whose productivity rate (average daily rate of bug resolution) is known. In this case, the above artificial change can be made in the back-end issue database and also the appropriate source code module of the IssuePlayer needs to be modified. This will enable the user to observe the effect of such change and this can be used as an experience for future projects.

Furthermore, the tool can be useful in effective training of newly-hired project managers and team leads in a company by helping them identify the problem spots and the right actions to take in given circumstances. This can be compared to the comprehensive flight simulator tools which trainee pilots get trained with before actually flying real aircrafts. In our context, trainee project managers can

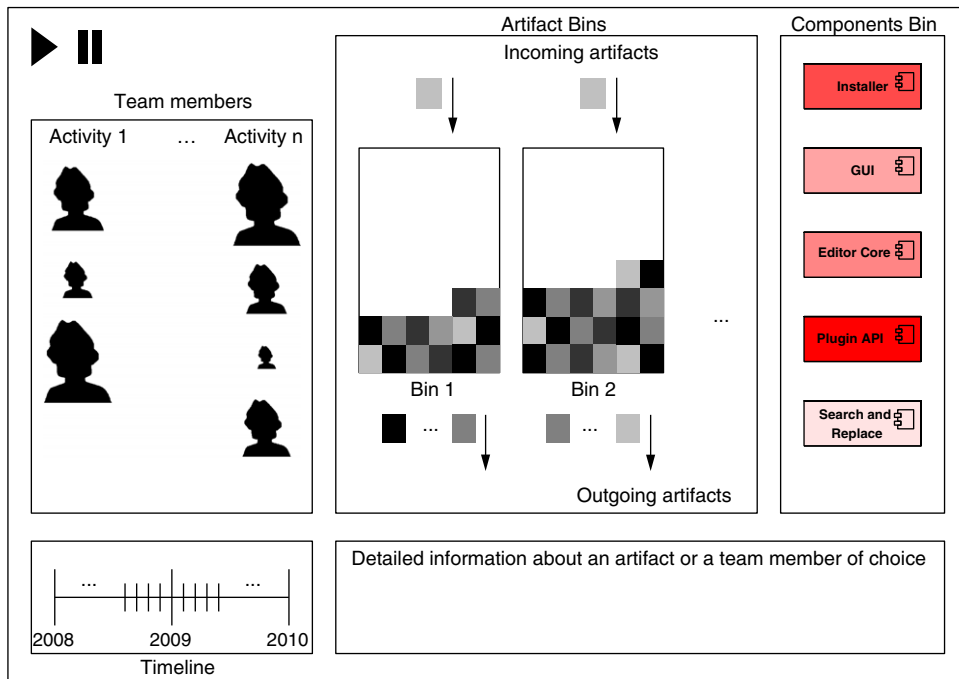


Fig. 7. A new feature of the tool currently under development.

get trained with this issue management player (as a simulator) before managing a large team in reality.

4.6. Application to other software artifacts

In all the above discussions, we discussed how the IssuePlayer can be used to play the issues submitted by project stakeholders. Other than issues, the proposed framework can also be useful in evaluating other historical software artifacts, e.g., requirements and design artifacts. For example, imagine each UML use-case as a box in IssuePlayer dropping in a specific bin for use cases. This will animate the process in which system analysts develop and submit use cases to the system designers in a software development team.

Afterwards, the design team would produce design documents as artifacts and would pass them on to the developers (programmers) via the development pool. If appropriate timing information for the above process is recorded in a project, IssuePlayer can be used to evaluate the historical process in a fashion similar to issues.

Going one step further, the recent Test-Driven Development (TDD) framework [30] recommends developing test cases first and then the piece of code to make them pass. IssuePlayer can also be extended to support the visual evaluation of test cases incoming/outgoing flows and their wait time in pool before being developed.

5. Architecture and usage of the toolset

This section presents to the interested reader the software architecture, the usage and some development details behind the IssuePlayer toolset.

An architectural overview of the toolset is presented in Section 5.1. Section 5.2 presents the data importer module of the toolset. IssuePlayer's GUI tool is introduced in Section 5.3.

5.1. An overview

IssuePlayer has been developed in Java. As the visualization backbone, we have used the popular *Processing* library [31], which is an open source project initiated by researchers at the MIT.

Our development team designed and developed the IssuePlayer's toolset in several build iterations using feedback from our industrial partners and students. A developers manual for those who might extend IssuePlayer is provided online [19].

Fig. 8 depicts an overview of the IssuePlayer architecture and its usage context. As discussed in Section 1, in most software development projects, project stakeholders use issue management software to track issues, e.g., BugZilla [5] and Tracker [1]. Such a tool stores the issue data in an issue database using a database management system (DBMS) such as Oracle, DB2, or MySQL.

The IssuePlayer toolset consists of four components: (1) data importer tool, (2) database interface, (3) a dedicated database and (4) the main GUI tool. The data importer tool is used to import the issue data from the generic issue database into IssuePlayer's dedicated database. IssuePlayer has been designed to flexibly work with various issue management software and databases. For this purpose, it requires the data importer tool to include the suitable database interface component to be able to

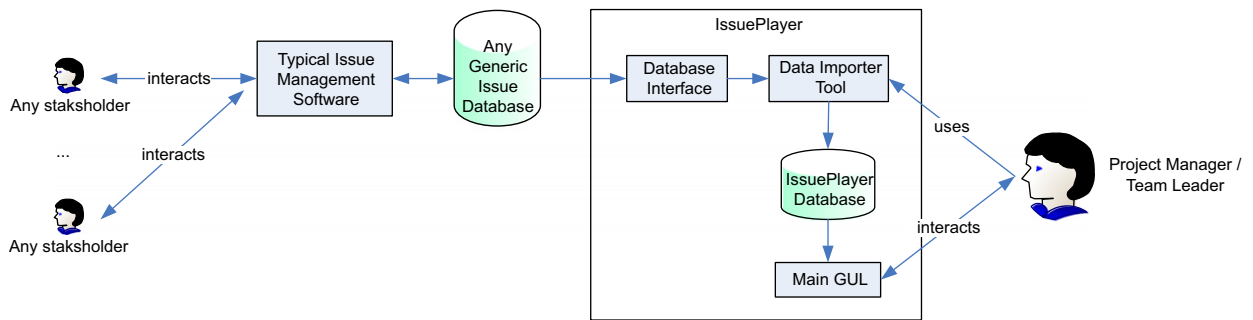


Fig. 8. An overview of the IssuePlayer architecture and usage context.

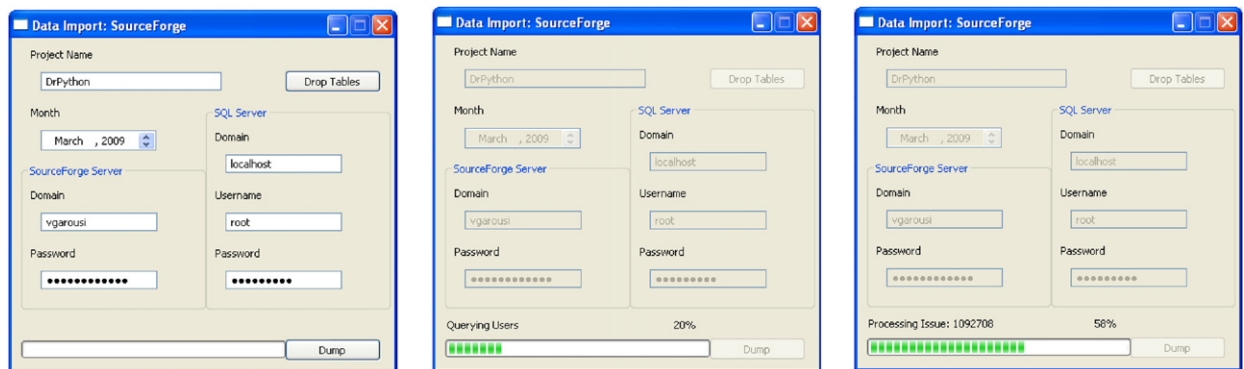


Fig. 9. Three snapshots from the data importer tool.

read (query) the necessary data from different database schemas of different issue management tools, e.g., EnterpriseWizard [4], BugZilla [5] and Tracker [1].

Having a dedicated database for IssuePlayer might be mistakenly interpreted as data duplication, however due to the following two major reasons, we learned by experience that it is important to have such a dedicated database:

- **Performance:** The first benefit of having a simpler, smaller (with less tables), dedicated and more focused database is that of data manipulation. For example, the “join” operations are much faster compared to data manipulation directly on generic issue databases. This is since the data are filtered to only those which are needed by IssuePlayer to visualize issue management processes. Unnecessary data of the issue management tools are not stored in the dedicated database.
- **Offline playback:** If the data from generic issue databases have been selectively copied into the IssuePlayer's dedicated database, even when a particular generic issue database is not available, IssuePlayer can still function and playback the issues offline.

At the time of this writing, two database interfaces have been developed to link and integrate IssuePlayer with two popular issue management tools: SourceForge's Tracker [32], and a customized version of BugZilla [5], used by one of our industrial partners. Based on the information

provided in this article and also the IssuePlayer's developer manual [19], it should be straightforward to develop other database interfaces to integrate IssuePlayer with other DBMS's.

The following two sub-sections discuss the features and the usability aspects of the data importer tool and the IssuePlayer's main GUI tool.

5.2. Data importer tool

Three snapshots from one implementation of the data importer tool are shown in Fig. 9. This version is integrated to work with the SourceForge's Tracker [1]. We also have a version which works with a customized version of BugZilla [5].

The version of the data importer shown in Fig. 9 has been set to query the data from the SourceForge Research Data Archive (SRDA) [33] and to store the data in the IssuePlayer's internal database. The name of the development project to be queried should be given in the window first. Then, the monthly dump month name should be provided. Furthermore, SRDA user information should be provided. The account information for the IssuePlayer's database should finally be given.

The DBMS chosen for the IssuePlayer's database is MySQL [34]. In addition to being flexible and easy to administer, MySQL is free for academic and personal purposes and is used by many open-source projects.

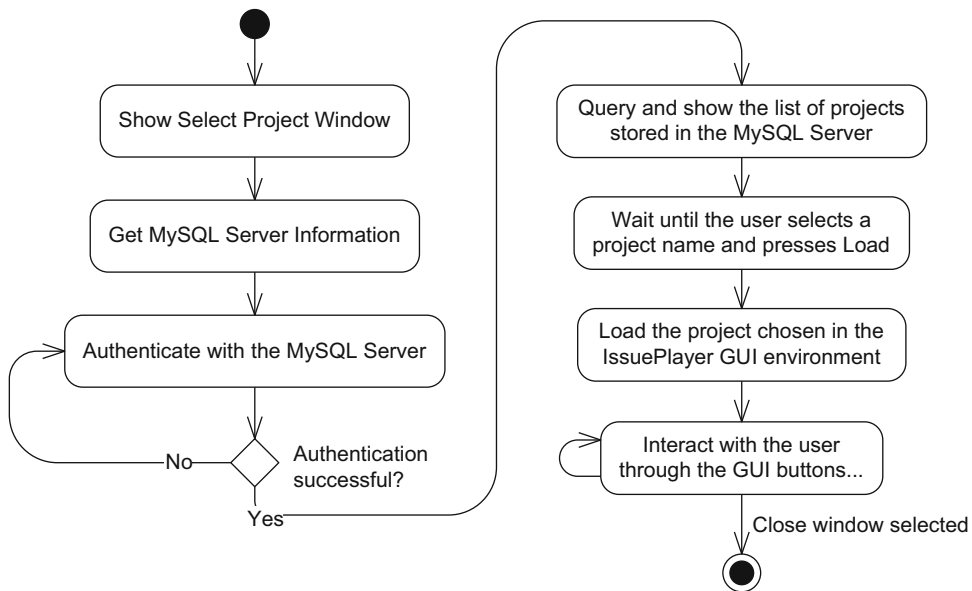


Fig. 10. . A high-level activity diagram for the IssuePlayer GUI tool

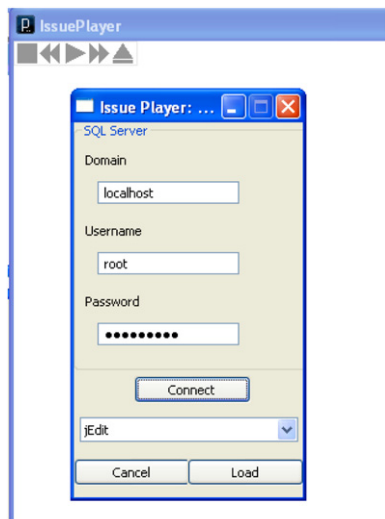


Fig. 11. IssuePlayer's Select Project window.

Once the importer tool finishes importing the issue data for a given development project, the data are in the MySQL server (as specified by the user). The user may now either decide to import the issue data of other project(s) or to go to the IssuePlayer GUI to start *playing* the issues.

5.3. IssuePlayer's GUI tool

A high-level activity diagram of how the GUI tool works is shown in Fig. 10. The tool first shows the *Select Project* window, a snapshot of which is shown in Fig. 11. The user is expected to enter the IssuePlayer's database server login information and click on *Connect* to get the list of projects. Once the list of projects is populated, the

user selects the project he/she would like to visually analyze. Once a project is chosen, the tool loads the project's issue data in the GUI environment, and starts to interact with the user.

6. Two empirical case studies

To further study the usefulness of the proposed visualization framework, we conducted two independent empirical studies with two of our industrial partners. For confidentiality reasons, the names of our partners cannot be revealed. Partner company one (P1) is a development firm in Canada which specializes in developing oil and gas software. Partner company two (P2) is an outsource development firm based in the Republic of Azerbaijan which develops a variety of applications, but most recently focuses on accounting software. Both firms happen to use BugZilla, while each has made their own minor customizations to IssuePlayer, e.g. storing more details about each bug report such as the exact procedure to duplicate it.

From firm P1, two project managers and one team lead agreed to use IssuePlayer for visual assessment of issue management in three of their development projects. For the purpose of this study, they used IssuePlayer for the duration of 3 weeks in the summer of 2009. From firm P2, one project manager and one team lead agreed to use IssuePlayer. They used IssuePlayer for the duration of 2 months.

To empirically evaluate the tool, we report both the quantitative and qualitative findings of the two case studies. A questionnaire was sent to the participating project managers and team leads, and the results are analyzed in Section 6.1. Qualitative feedback from the users is discussed in Section 6.2.

6.1. Questionnaire results

We prepared a questionnaire to quantitatively measure the users' perception and satisfaction with the IssuePlayer tool. The questionnaire and its statistical results are shown in Table 3. The following 7-point Likert scale [35] was used for answering the survey questions.

| | |
|-----------------------|--------------------|
| 1: strongly disagree | 5: some-what agree |
| 2: disagree | 6: agree |
| 3: some-what disagree | 7: strongly agree |
| 4: neither | |

Table 3
Perception and satisfaction questionnaire and its statistical data.

| Question number | Questionnaire item | Average response |
|--|--|------------------|
| Before vs. after using the IssuePlayer tool | | |
| 1 | Before I used the IssuePlayer tool, I was able to effectively detect process symptoms in my project(s)' issue management | 3.80 |
| 2 | After I used the IssuePlayer tool, I was able to effectively detect process symptoms in my project(s)' issue management | 5.40 |
| 3 | Before I used the IssuePlayer tool, I was able to effectively monitor the performance of developers (debuggers) in my project(s)' issue management | 4.20 |
| 4 | After I used the IssuePlayer tool, I was able to effectively monitor the performance of developers (debuggers) in my project(s)' issue management | 6.00 |
| 5 | Before I used the IssuePlayer tool, I was able to effectively monitor the performance of testers in my project(s)' issue management | 4.00 |
| 6 | After I used the IssuePlayer tool, I was able to effectively monitor the performance of testers in my project(s)' issue management | 5.80 |
| The tool itself | | |
| 7 | I believe the tool has the set of necessary functional features | 5.40 |
| 8 | I believe the tool has a friendly GUI | 5.60 |

Note that question pairs 3,4 and 5,6 are different, as developers (debuggers) have different roles than testers. The latter users detect defects (bugs) and submit them into issue management tools, while the former users fetch issues from the repository and fix bugs. Separate visualizations of these two types of users are supported in IssuePlayer (Section 4).

The questionnaire was distributed to the five users (3 in P1 and 2 in P2) after they used the tool. Their participation in the survey was voluntary, anonymous and confidential. All five users filled out and returned the questionnaire. The average values of each questionnaire question in 7-point Likert scale are reported in Table 3. The histogram of questionnaire results is also visualized as a stacked-column chart in Fig. 12. The main observations based on the histogram values are discussed next.

- Out of 40 question answers (5 students*8 questions), there was no “strongly disagree” answer for any question.
- According to the answers of the three pairs of the before–after questions (#1–#6), the users' perception in doing management and monitoring tasks after using the tool has improved.
 - Question 2: 60% of users “agreed” that the tool enabled them to effectively detect process symptoms in their project(s)' issue management.
 - Question 4: 1 user “strongly agreed” and 3 users “agreed” that the tool enabled them to monitor the performance of developers (debuggers) in their project(s)' issue management.
 - Question 6: 1 user “strongly agreed” and 2 users “agreed” that the tool enabled them to monitor the performance of testers in their project(s)' issue management.
- 4 of 5 users agreed that the tool has the set of necessary functional features, and that the tool has a friendly GUI.

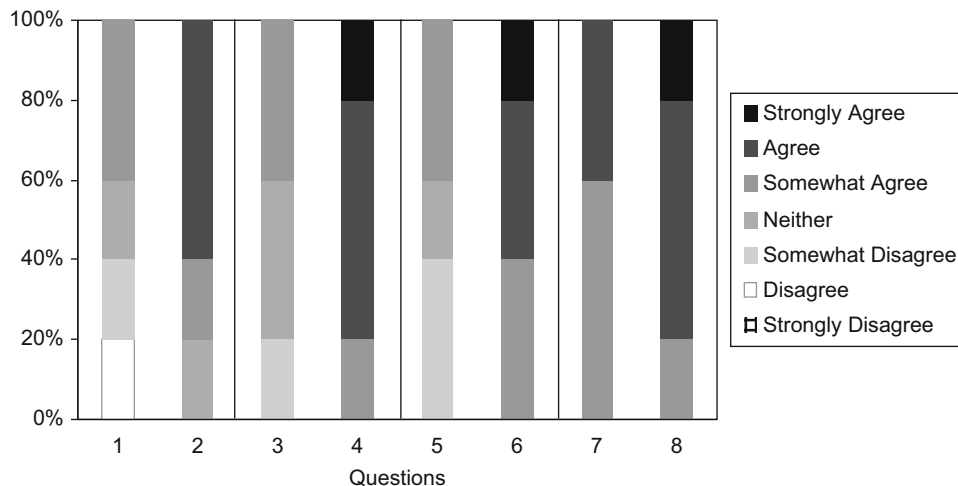


Fig. 12. Histogram of questionnaire results.

6.2. Qualitative feedback

The five users provided anonymous written comments in the questionnaire forms. A selected list of those comments is discussed next (similar comments have been merged). The comments which need analysis are analyzed.

- *“In general, I found the tool very useful and beneficial. For example, it helped me to pinpoint several time spots in which my code maintenance team were not as productive as I used to think. This actually hinted me to go and talk to them about those days. We actually found out interesting issues happening on those days!”*
- *“The tool enabled our team to evaluate its responsiveness to issues submitted by our clients, and be more prompt. We have already installed this tool for the use of our other teams as well, and plan to use it regularly!”*
- *“The tool was useful, but I had to manually detect process symptoms. I hope the tool can be extended to incorporate automatic detection of process symptoms.”*
 - Analysis: The user mentions the exact point we raise in Section 3.2.5. We hope to enhance the tool in its next versions by adding the above feature.
- *“Perhaps, it is also a good idea to have a live version of this program so that it is always running on a screen in the hallway and everyone is always informed of the issue pile-up status around the clock (i.e., 24/7)”.*
 - Analysis: This is a constructive feedback. We are also adding this to the to-do list of the tool's next version.
- *“The tool was very beneficial and interesting in the context of one of our projects for which we had an outsourcing team overseas. Our local team was finding the bugs and the remote team was fixing them. I, as the project manager, could easily track what is happening and who is fixing what and when”.*
 - Analysis: This comment is interesting and also reveals the potential usefulness of the tool in the context of global (distributed) software development [36].

7. Conclusions and future works

This article presented a software visualization framework which can help project managers and team leaders in overseeing issues and their management in software development. To automate the framework, a dashboard tool called IssuePlayer was developed. The tool can be used to study the trends in which different types of issues (e.g., bugs, support requests) are submitted, handled and piled up in software projects and use that information to identify process symptoms, e.g., the times when the code maintenance team is not responsive enough. The interactive nature of the tool enables identification of the team members who have not been as active as they were expected to be in such cases. The user can play, pause, rewind and forward the issue management histories using the tool. The tool is empirically evaluated by two industrial partners in North America and Europe. The survey and qualitative feedback support the usefulness and effectiveness of the tool in assessing the issue

management processes and the performance of team members. The tool can be used complementarily in parallel with textual information provided by issue management tools (e.g., BugZilla) to enable team leaders to conduct effective and successful monitoring of issue management in software development projects.

As a concluding remark, it should also be noted that the current version of the toolset has been useful in several real projects. The open-source toolset provides a starting code base upon which extra features may be developed as needed by other researchers and practitioners.

We plan to add additional realistically useful features to IssuePlayer, such as: identification of problematic components and releases (as discussed in Section 4.4), application to other software artifacts, e.g., requirements and design artifacts (as discussed in Section 4.6).

We plan to conduct further empirical evaluations of the tool in industrial settings. It should also be worthwhile to investigate the usage and adaptation of artificial intelligence (AI) techniques in our framework to extract high-level knowledge about the development processes.

Acknowledgements

This project was supported by the Discovery Grant no. 341511-07 and an Undergraduate Student Research Award (USRA) from the Natural Sciences and Engineering Research Council of Canada (NSERC). The author would like to thank James Leitch for his help in the development of the tool.

Appendix A. Supplementary Information

Supplementary data associated with this article can be found in the online version at [doi:10.1016/j.jvlc.2010.03.001](https://doi.org/10.1016/j.jvlc.2010.03.001).

References

- [1] SourceForge Documentation. <<http://alexandria.wiki.sourceforge.net>>, last accessed: September 2009.
- [2] K.R. Bainey, Integrated IT project management: a model-centric approach: Artech House, 2004.
- [3] Autotask Help Desk. <www.autotask.com>, last accessed: September 2009.
- [4] Enterprise Wizard. <<https://www.enterprisewizard.com>>, last accessed: September 2009.
- [5] BugZilla. <www.bugzilla.org>, last accessed: September 2009.
- [6] F.H. Post, G.M. Nielson, G.-P. Bonneau, in: Data Visualization: The State of the Art, Kluwer, 2003.
- [7] M. Fowler, K. Beck, J. Brant, in: Refactoring: Improving the Design of Existing Code, Addison-Wesley, 1999.
- [8] S. Ambler, Model reviews: best practice or process smell? <<http://www.agilemodeling.com/essays/modelReviews.htm>>, last accessed: August 2008.
- [9] J.I. Maletic, A. Marcus, M.L. Collard, A task oriented view of software visualization, in: Proceedings of the International Workshop on Visualizing Software for Understanding and Analysis, 2002, pp. 32–40.
- [10] T. Xie, J. Pei, and A.E. Hassan, Mining software engineering data, in: Companion of the International Conference on Software Engineering, 2007, pp. 172–173.

- [11] K. Crowston, H. Annabi, J. Howison, and C. Masango, Towards a portfolio of FLOSS project success measures, in: Workshop on Open Source Software Engineering, International Conference on Software Engineering, 2004, pp. 29–33.
- [12] K. Crowston and B. Scozzi, Coordination practices for bug fixing within FLOSS development teams, in: Proceedings of the International Workshop on Computer-Supported Activity Coordination, 2004.
- [13] C. Jensen and W. Scacchi, Data mining for software process discovery in open source software development communities, in: Proceedings of the Workshop on Mining Software Repositories, 2004, pp. 96–100.
- [14] E. Navarro, SimSE: a software engineering simulation environment for software process education PhD thesis, School of Information and Computer Sciences, University of California Irvine, 2006.
- [15] M. Ogawa K.-L. Ma, code_swarm: a design study in organic software visualization, in: Proceedings of the IEEE Information Visualization Conference, 2009.
- [16] ProSim tool, <http://www.prosim.net>, last accessed: January 2010.
- [17] W. Scacchi, Experience with software process simulation and modeling, *Journal of Systems and Software* 46 (2–3) (1999) 183–192.
- [18] T.K. Abdel-Hamid, S.E. Madnick, Lessons learned from modeling the dynamics of software development, *Communications of the ACM* 32 (12) (1989) 1426–1438.
- [19] J. Leitch and V. Garousi, IssuePlayer website <http://code.google.com/p/issueplayer>, last accessed: September 2009.
- [20] V. Garousi and J. Leitch, IssuePlayer user manual. <http://issueplayer.googlecode.com/files/IssuePlayer%20User%20Manual.pdf>, last accessed: September 2009.
- [21] V. Garousi, Investigating the success factors of open-source software projects across their lifetime. *Journal of Software Engineering Studies* 4 (1) (2009) 2–15.
- [22] V. Garousi, Evidence-based Insights about issue management processes: an exploratory study, in: Proceedings of the International Conference on Software Process (ICSP), 2009, pp. 112–123.
- [23] jEdit on SourceForge. <http://sourceforge.net/projects/jedit>, last accessed: September 2009.
- [24] E.S. Raymond, The cathedral and the bazaar, *First Monday Journal* 3 (3) (1998).
- [25] DrPython on SourceForge. <http://sourceforge.net/projects/drpython>, last accessed: January 2010.
- [26] V. Garousi. Investigating the success factors of open-source software projects across their lifetime. *Journal of Software Engineering Studies*, special issue on Open-Source Software development 2010; in press.
- [27] B.W. Boehm, Improving software productivity, in: W. Barry (Ed.), *Boehm's lifetime contributions to software engineering*, IEEE Computer Society, 2007.
- [28] N. Gorla, Y.W. Lam, Who should work with whom? Building effective software project teams, *Communications of the ACM* 47 (6) (2004) 79–82.
- [29] S.H. Kan, in: *Metrics and models in software quality engineering*, 2nd ed., Addison-Wesley, 2002.
- [30] K. Beck, in: *Test-driven development: by example*, Addison-Wesley, 2002.
- [31] Processing project. www.processing.org, last accessed: September 2009.
- [32] <http://alexandria.wiki.sourceforge.net>, last accessed: August 2008.
- [33] Greg Madey, editor. The SourceForge Research Data Archive (SRDA). Dame: University of Notre; <http://zerlot.cse.nd.edu>, last accessed: September 2009.
- [34] MySQL website. <http://www.mysql.com>, last accessed: September 2009.
- [35] R. Likert, A technique for the measurement of attitudes, *Archives of Psychology* 140 (1932) 1–55.
- [36] S. Sahay, B. Nicholson, S. Krishna, in: *Global IT outsourcing: software development across borders*, Cambridge University Press, 2003.