Bug Reproduction: A Collaborative Practice Within Software Maintenance Activities

Dhaval Vyas, Thomas Fritz and David Shepherd

Abstract Software development settings provide a great opportunity for CSCW researchers to study collaborative work. In this paper, we explore a specific work practice called *bug reproduction* that is a part of the software bug-fixing process. Bug reproduction is a highly collaborative process by which software developers attempt to locally replicate the 'environment' within which a bug was originally encountered. Customers, who encounter bugs in their everyday use of systems, play an important role in bug reproduction as they provide useful information to developers, in the form of steps for reproduction, software screenshots, trace logs, and other ways to describe a problem. Bug reproduction, however, poses major hurdles in software maintenance as it is often challenging to replicate the contextual aspects that are at play at the customers' end. To study the bug reproduction process from a human-centered perspective, we carried out an ethnographic study at a multinational engineering company. Using semi-structured interviews, a questionnaire and half-a-day observation of sixteen software developers working on different software maintenance projects, we studied bug reproduction. In this paper, we present a holistic view of bug reproduction practices from a real-world setting and discuss implications for designing tools to address the challenges developers face during bug reproduction.

D. Vyas (✉)
Queensland University of Technology, Brisbane, QLD, Australia
e-mail: d.vyas@qut.edu.au

T. Fritz
University of Zurich, Zurich, Switzerland
e-mail: fritz@ifi.uzh.ch

D. Shepherd
ABB Corporate Research, Raleigh, NC, USA
e-mail: david.shepherd@us.abb.com

C. Rossitto et al. (eds.), *COOP 2014 - Proceedings of the 11th International Conference on the Design of Cooperative Systems, 27–30 May 2014, Nice (France)*,
DOI: 10.1007/978-3-319-06498-7_12, © Springer International Publishing Switzerland 2014

1 Introduction

Companies with a large software portfolio have in-house maintenance support. Their software maintenance divisions frequently get 'bugs' from customers as well as from testing and development teams who are continuously working towards adding new requirements and improving existing products. Maintenance divisions use bug tracking systems where all information related to bugs is stored: starting from when it was encountered to when it was implemented in a product.

Bug reproduction is a highly collaborative activity that starts at an early stage of bug-fixing process. When a bug is encountered, the bug reporter provides relevant information about the bug in a bug tracking system and describes how it can be recreated. Often detailed description of problems, software versions, screenshots, step-wise guidance or navigation is provided by attaching images, videos and textual information [4, 5, 11]. Using this information, developers locally re-create the scenario in which the bug was detected. When developers are able to successfully reproduce the bug on their own machines, they can answer several questions related to where a problem is located in the code and how it can be fixed. Previous studies have shown that bug reports often lack useful information that may be needed for bug reproduction [5]. Additionally, it is sometimes difficult for customers to know what type of information a software maintenance team will require in order to fix a bug. In cases where it may not be possible to reproduce a bug easily, developers ask for more details from customers. This causes delays and overheads in product development and maintenance.

In order to study the bug reproduction process in detail, to explore major hurdles and challenges developers face and to elicit ideas for developing tools to support bug reproduction, we carried out an ethnographic field study with sixteen developers in a software maintenance division of an engineering conglomerate. We studied their bug reproduction practices using semi-structured interviews, in situ observation sessions and a questionnaire. Our results provide a holistic view on the bug reproduction process, where we provide insights into what type of information is provided by customers, how bug reproduction is carried out and how that helps in fixing bugs. Our results show that insufficient information from customers, tedious logistical efforts for bug reproduction setups and the contextual issues of bugs are the three major challenges to bug reproduction. We also found that developers find 'steps for reproduction' and 'trace logs' to be the most important information for reproducing bugs. Surprisingly, our findings show that bug reproduction contributed towards improving developers' confidence for going about fixing bugs. Based on our findings, we provide several design implications such as the use of tracing and monitoring mechanisms at customers' site to allow quick access of useful information for developers, adding templates and annotations to bug tracking systems and connecting bug tracking systems with the work environments of relevant stakeholders involved in the bug reproduction cycle.

In the rest of this paper, we start by describing some related work in this field. We then describe our approach and methods used for this field study, followed by

our findings. Finally, we provide implications for providing adequate support for bug reproduction activities.

1.1 Bug Reproduction and CSCW

The topic of bug reproduction has traditionally been studied in the software engineering and software maintenance communities. A study such as this can be relevant for the CSCW community for the following three reasons:

1. ***Study Collaboration***: Bug reproduction presents an interesting case of collaborative practices between developers and bug reporters (customers or testers), where communication is rarely direct and often mediated through a bug tracking system or through other stakeholders such as product team managers and customer support professionals. The information communicated is often of a multi-modal nature (screen-shots, videos, texts) and is highly dependent on the context within which a bug is encountered.
2. ***Empirical Value***: Unlike the methods used in the traditional software engineering research [4, 15, 16, 20, 26], we apply an ethnographic approach to gain access to the real-world practices of developers in their ongoing bug-fixing projects. This research will allow us to gain a holistic view of the bug reproduction process, where we can contribute towards an improved empirical foundation for understanding software bug-fixing practices.
3. ***Design Ideas***: A user-centered approach such as ours would allow us to connect the empirical findings gained from ethnographic work to novel designs and tool ideas that can improve current practices of bug reproduction. This in fact is our main goal—to develop better tools to improve productivity and efficiency.

2 Related Work

In the following, we provide a short literature review on bug-fixing. We highlight the contributions from the CSCW community for studying software development activities and then move onto the traditional software engineering literature.

CSCW researchers have long been highlighting different collaborative aspects of software development practices. Empirical studies on groupware technologies have emphasized the role of group awareness [9], work dependencies [12], and aspects relating to collaborating work cultures [21]. Ethnographic methods are used to study specific practices related to, for example, the use of configuration management tools [8], workflow management activities [10] and software testing practices [18]. Within software development and maintenance teams, bug tracking systems serve as the medium through which not only developers and customer can coordinate their activities but other stakeholders such as product managers, testers,

and customer support professionals can also interact and communicate. There are several studies on bug tracking systems done within the HCI/CSCW community [2, 8, 19, 22, 25]. Based on a qualitative study involving 15 developers, Bertram [3] highlighted that bug tracking systems were used as (1) a knowledge repository where activities from different stakeholders were getting stored, (2) a boundary object [24], to fulfill the needs of different stakeholders, and (3) a communication and coordination hub. Studies have shown that bug tracking system serves as a tool to negotiate specific details of bug-fixing activities [10].

Within the software engineering community, Zhang et al. [27] explored the most important factors that affect the bug-fixing process: type of bug, severity of bug, operating system, and description of bugs. The role of software users (or customers) in bug-fixing is also emphasized in several studies. Developers require different types of information from users in order to fix bugs. Bettenburg et al. [4] explored a set of information required in a bug report by collecting responses from 466 developers. Their study highlighted that bug reports often have a strong mismatch between what developers needed and what information was provided by users. Based on the analysis of 600 bug reports, Breu et al. [5] developed categories of questions that are asked by developers to the users who reported bugs. Frequently asked questions were related to missing information, clarifications, triaging, debugging, correction, status enquiry, resolution and process. Other similar studies included the use of card sorting methods [15] for exploring how bugs can be reported and resolved in the form of design recommendation for new bug tracking systems.

Several studies have explored the importance of different information required for bug reproduction, such as trace logs and steps to reproduce. For example, Schroter et al. [23] carried out an empirical study on the usage of stack traces by developers from the ECLIPSE project and found that bug reports with stack traces are fixed faster than bug reports without them. They also found that bugs are likely to be found in one of the top ten stack frames. Herbold et al. [11] developed a non-intrusive, easily to integrate GUI-based monitoring mechanism which would automatically collect usage logs of different user activities and allow replaying them for the purpose of reproducing bugs whenever they occur. It is also important to note that in some cases it might be embarrassing for companies when such privacy-centric data is reviled. To support this need, Casto et al. [6] developed a mechanism by which software developers are provided with new input values that can be as useful as the original input values that can be used in bug reproduction. This way less information is revealed to developers and companies' privacy is also protected.

3 Field Study

In our research, we aimed at gaining access to developers' natural practices to be able to learn about their software bug-fixing and in particular bug reproduction practices. We believed that an *in situ* account on developers may shed light on the social and situated nature of software bug-fixing activities.

3.1 Methods

Over a period of three months, we conducted an ethnographic field study of 16 software developers working in a software maintenance division of a multinational engineering conglomerate. These developers belonged to 8 different software product teams. We used the following three methods in our research.

1. **In situ Observations**: We video recorded developers' real-time software bug-fixing activities at their workspace. We started our observations from the beginning when a bug was reported and assigned to developers. These in situ observations in most cases lasted half-a-day; however, in some cases we prolonged our interactions with the developers to follow the complete bug-fixing process. At the end of our sessions, we collected all artefacts that were being used in their bug-fixing activities such as bug reports and related documents.
2. **Semi-structured Interviews**: Following the observations, we carried out semi-structured interview at developers' workspace, where we asked our participants questions related to their bug-fixing processes and practices. We aimed at getting insights into their use of different tools, their collaboration and communication practices, their use of bug tracking systems, and so on. Additionally, we asked participants to give an account of at least two bugs that they recently fixed. These interviews lasted for 45 min to 1 h per participant.
3. **Questionnaire**: At the completion of the observation and interview sessions, we sent out a questionnaire to all 16 participants. We developed this questionnaire using the inputs from observation and interview sessions. The questionnaire used five-point Likert scale and focused on understanding participants' preferences related to bug reproduction practices, e.g. what type of information is most useful, how bug reproduction helps in following bug-fixing activities.

3.2 Participants

In the facility where we were carrying out our research, there were more than 800 software developers working on a large variety of products from domains such as automation, power and robotics. Our selection of developers aimed at adding heterogeneity in our data sample and hence allowing generalizability in our findings. To recruit software developers for our study, we contacted division managers of these different product domains and using their help recruited developers from different software development teams. We also ensured that we selected no more than two developers from one team.

Table 1 provides participant details of our field study. In general, we involved developers who were working on SCADA (supervisory control and data

Table 1 Participant details

Product type	No. of developers	No. of bugs studied
SCADA products	5	6—Observation; 6—Interviews
Automation SW 1	3	2—Observation; 6—Interviews
Automation SW 2	4	3—Observation; 3—Interviews
Robotics SW	1	2—Interviews
Power product	3	3—Observation; 6—Interviews
Total	16	39

acquisition), automation, power and robotics products. From these 16 developers, we studied 39 bug-fixing cases taking into account the natural practices of these developers. From our field study, we collected a large amount of videos, field and interview notes, bug reports and bug-related artefacts. The results presented below were obtained through a qualitative analysis [7] of our collected data. We started by creating a large affinity wall [13] using post-its and used open-coding to derive larger concepts and categories.

4 Results

From our field study, we derived several interesting perspectives on bug reproduction and were able to identify important social dimensions to the bug reproduction process.

4.1 Social Process of Bug Reproduction

Software bug reproduction is an activity developers perform to locally recreate the situation in which a bug (or a defect) was originally observed at the site of a customer or a tester. It is a widely-used practice in software maintenance and typically starts at an early stage of big fixing activities. It is a highly social activity because it involves communication and collaboration between several actors, including developers, testers, customers, product managers, customer support professionals among others.

In our study, we attempted to capture a holistic view of bug reproduction. Figure 1 shows a high level view of the social side of the bug reproduction process, as reported by the developers who participated in our field study. This figure particularly focuses on the customer reported bugs. When a customer encounters a bug, he/she reports it to the customer support offered by the software company. Here, the customer provides all the basic information about the

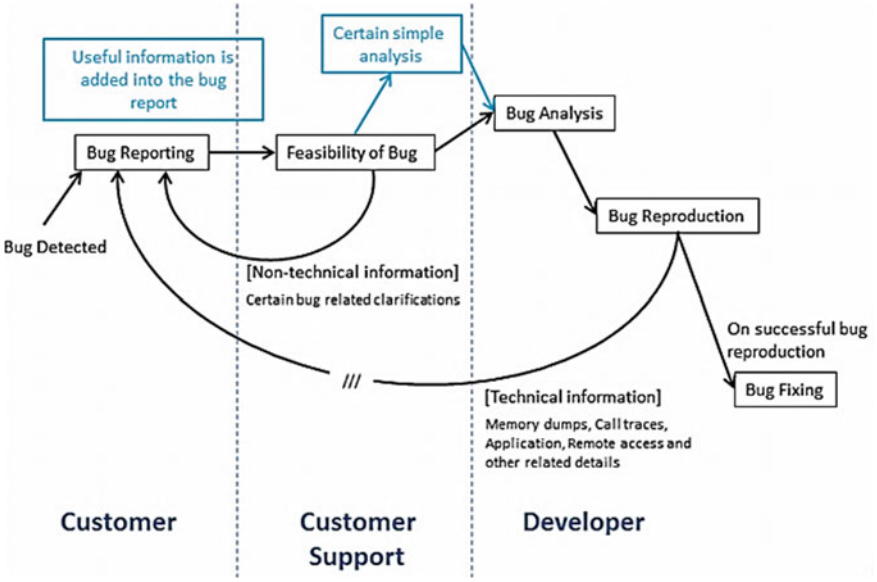

Fig. 1 The social process of bug reproduction

scenario within which the bug occurred. Using these details, the customer support professionals check the feasibility of the bug and check if all the software configurations are in place. At this stage, they would provide a fix, if the problem is simple such as a wrong configuration was used or date format mismatched. If they cannot fix the problem, they report it into the bug tracking system. Customer support professionals are not developers themselves; hence they cannot solve any technical problems. They work as a mediator between customers and developers. While the bug is reported into the bug tracker, the customer support team adds details such as software version and module, OS version, description of the problem and steps for reproduction into the bug tracker. If such information is not added the customer support professionals collaborates with the customer and adds this information. Once a bug is reported into a bug tracking system, it is assigned to a responsible product team manager based on the matching of appropriate software modules. The product team manager also does a feasibility check on the bug and assigns it to an adequate developer to fix this bug. As a part of an initial analysis of the bug, the developer starts the bug reproduction process utilizing the information that is provided in the bug tracker. If the developer needs more information regarding the bug, he contacts the customers using the help from customer support team. Developers rarely have direct contacts with the customer, but if needed they can also have direct phone calls or video chats with the customer.

4.2 Perceived Advantages of Bug Reproduction

The software developers who participated in our field study provided useful insights on how they perceived the use of bug reproduction in their everyday bug-fixing activities. Overall, we elicited three perceived advantages of bug reproduction: (1) understanding the problem; (2) fixing bugs in a quick manner; and (3) increasing confidence level of developers.

One of the reasons for carrying out bug reproduction is for developers to see how and why a bug occurs and to have a better understanding of the bug. While reproducing bugs developers gain the firsthand experience of steps that lead to a bug and how the bug behaves. The power and automation products of the company that we studied were being run in multiple industries such as minerals, pulp and paper, cement, and oil and gas domains. Hence, when a bug is reported by a customer, it is only through bug reproduction developers can know how the software was being used and what configuration and settings were in place at the customer's site. In many cases, the description of a bug or its screenshot provided in a bug tracking system may not be enough for a developer to sufficiently understand the bug. It is through reproducing bugs that developers can develop a better understanding of the problem at hand. The following is a comment from a developer, which indicates our finding:

> If there is a UI related or a printing related issue then bug reproduction may not be necessary. But if I get issue related to system crash or similar then I need to investigate how the software is configured on the site of the customer. We need to interact with them and get required information.

More importantly, a successful bug reproduction allows developers to fix bugs in a quicker manner. While debugging, a successful bug reproduction allows developers to better locate the precise area in the code where the problem is located. Bug reproduction saves a considerable amount of time as developers can focus on specific parts and flows of the code that need attention. A developer commented:

> There are thousands of lines of code in this software and it is impossible to know everything in it because some of the code is legacy. With bug reproduction we can limit our efforts. We need not find each and every flow inside the code. If we know that these are the steps to reproduce the bug then we can pinch on that particular flow in the code and target only that flow to solve the problem.

The third benefit of bug reproduction was that it increased the perceived confidence level of developers before actually fixing a bug. Successful bug reproduction meant that details provided by the bug reporter are enough and the developer can directly focus on the fix. This part will be elaborated in the later part of this paper.

5 A Holistic View of Bug Reproduction

5.1 When a Bug is Reported…

When a fault occurs during use, it is reported in a bug tracking system. A bug can be anything from a system crash or hang to any inconsistent behavior of a system. The bug initiator uses the bug tracking system to provide a description of the problem, details of the system configuration (e.g. product version, OS version), steps for recreating the bug, screenshots of the system interface and pointers to the location of the problem and other relevant information. Often this information is added to the bug report in the bug tracker; however, in some cases information is transferred via emails and file transfers. The bug tracking system serves as a common tool for multiple objectives for different stakeholders [1, 24].

Figure 2 is an excerpt from a bug report where the bug initiator has provided screenshots of a software that had a bug in it and pointed out certain fields in red color to indicate the problem. When a bug is reported by a tester or a developer, they tend to provide quite detailed, technical information in the bug report, where code patches, screenshots of a debugger and trace logs are attached. This way an effort is made by the bug initiator to provide detailed information about the bug.

Often, in the case of customer reported bugs, it was not easy for the customers to provide sufficient information in bug reports, as they themselves were not expert enough to provide such details. In such cases, developers would need to request details such as trace logs and memory dumps. There are two major challenges to this activity: (1) it may not be clear what information would be relevant to reproduce a bug, and (2) even when sufficient details are provided in the bug report a developer may not be able to reproduce the bug on his own machine. The following is a comment by a developer:

> Sometimes, customers miss to provide very basic information in their bug report and it takes us long time to reproduce the problem. One time, a customer missed to provide the correct time zone and we were not able to reproduce the problem for two weeks.

We observed the use of videos to provide information related to bugs. In particular, when it was important to convey some dynamic behaviors of bugs or some difficult to explain phenomenon videos were frequently used. The developers working on an embedded software for a power product portfolio dealt with hardware such as relays, breakers and transformers. In these cases, the use of videos to provide bug reproduction details was preferred by development teams. Here is a comment from of the developers from power products:

> Our partner team in Finland received this bug from a customer. When they could not fix it, they sent this bug to us. They also sent a video and trace-logs. This bug occurs once in may be 20 times. So, it is really hard for us find out the exact reasons. The video gives a dynamic view of the bug.

In the above case, the bug was reported by a customer. However the video was captured by a local development support team who had interacted with the

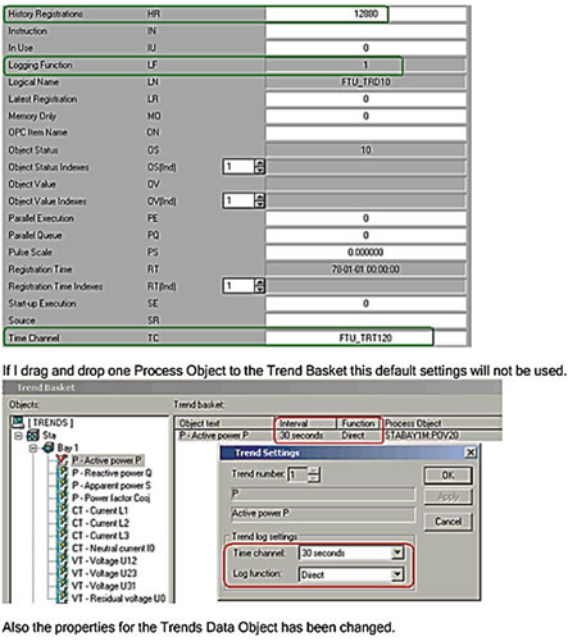

Fig. 2 An excerpt from a bug report

customer during the initial stages and attempted to fix the bug. The video had captured the complete hardware setup so that the development team can have a comprehensive knowledge of the system configuration.

There were several other cases, where bugs were reassigned to a different team as it was not possible to reproduce it on the first try. This usually happened between local and global development teams. In such cases, the development teams interact with each other and usually the former development team provides details related to their bug reproduction efforts. The following is a comment left on a bug report by a development team to give details of their bug reproduction.

> There was no straight forward procedure to reproduce this error. I had done some random "monkey testing" for the WHMI. The WHMI was in timeout and after longer period of time I re-authenticated with Firefox browser (Event list was the page which was open before the timeout and SSL was enabled). After re-authentication the debugger stopped at the breakpoint (See attached image).

5.2 During Bug Reproduction…

After a bug goes through initial feasibility check, the responsible product team manager assigns the bug to an appropriate developer of his team. Utilizing the information provided by the bug initiator, the developer attempts to reproduce the bug on his own machine.

In the case of a customer reported bug, timing becomes an important factor. Depending on the company's contract with customers, developers need to fix the bug and dispatch results in a week to 10 days. Hence, the reproduction needs to happen as soon as possible with minimal delays. For developers reproducing a bug require some effort in changing their current system configurations. Some development teams had access to a testing lab, where they can reproduce or test on specialized machines. However, the majority of developers were initially using their own machines for reproducing bugs and only when it was not possible to reproduce bugs, they went to the testing lab. The following is a comment made by a developer on how time consuming reproduction can be.

> Our team has a very dynamic bug-fixing process, although I do feel that it sometimes hampers our development process. When we get a bug, we have to remove our existing system settings and stop our ongoing development work, apply the customer's configurations, load all the software customer has been using. So, in all we end-up spending 1–2 days in only creating the right setup.

Developers rely heavily on the information supplied by customers; hence they tend to work with the data that is in bug trackers. Apart from the system configuration data, customers provide steps for reproducing, screenshots of the software's UI (e.g. Fig. 2), and a description of the problem referring to observed and expected behaviors. For customers, supplying this type of information is relatively easy as it is visible and observable. However, in case of a system crash or hang, the above mentioned information may not allow developers to successfully reproduce a bug and they often ask for more information from the customers. Often, customers themselves would not know or remember what exactly they did which caused the problem such as the system crash or hang. In such cases, developers try to extract trace logs and memory dumps from the customers' systems.

In cases, where it was not possible to reproduce a bug based on the given instructions, especially when bugs occur on the server, developers were able to remotely log into the customers' system and observe its behavior. Generally, customers allow remote log-in only when the system is not live. Additionally, developers use 'debug DLLs' to generate memory dumps from remotely debug customer's system. Here is a comment made by a developer:

> In a scenario where we cannot reproduce a bug on our own machine, we have the option of placing our debug DLLs in the customer's system and we can try to collect memory dumps this way. If we are able to reproduce then we use WinDBG debugger to debug the system.

There were also some challenges with the input that customers' were providing in bug trackers. In a small number of cases, developers reported that they received insufficient information from customers. There were examples where customers provided only a screenshot of their system or provided minimal description of the problem they were facing. In such cases, developers would randomly create use cases with certain interactions and try to reproduce bugs. The following is a comment made by one developer:

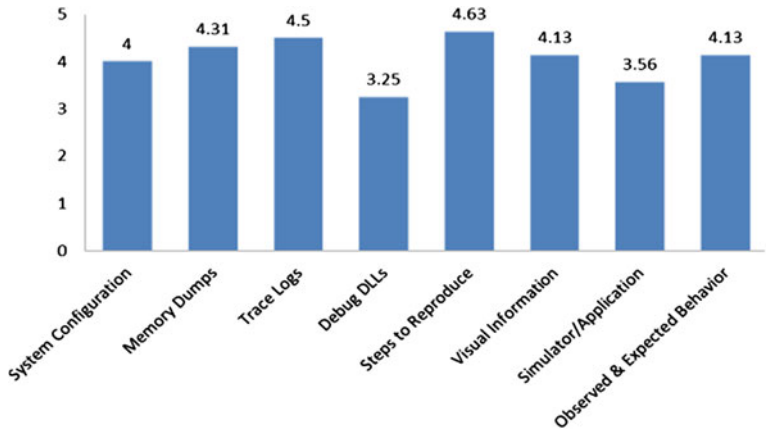

Fig. 3 Average rating of our participants' preference for different bug reproduction information (n = 16)

> Recently, we had a bug which we have not been able to reproduce, till now. We have a Chinese version of our software and a customer reported a bug where Chinese fonts looked too small. All we had in the bug tracker was a screenshot of the system. No further information was provided. So we tried to reproduce this bug in different OSes. We tried Windows 7, 2008, Advanced Server R2, XP. We also changed the themes of the Windows, because themes also affect the font size. Our initial guess was Windows 7 as the title bars in the screenshot was matching it. When all these attempts failed, we asked the customer and he clarified that he was using Windows 2008 Advanced Server with the theme of Windows 7. Even then, we haven't been able to reproduce this bug in the customer's suggested environment.

Following the completion of our observation and interview sessions with all our participants, we sent out a questionnaire based on their qualitative feedback. A section of this questionnaire was about exploring developers' preference on 'what type of bug reproduction information they prefer from customers or testers'. We selected eight of the most frequently mentioned items from their feedback and asked them to rate these on a five-point Likert scale. Our main intention here was to validate the feedback of our participants and compare the importance of these individual items with one another. These eight items were: system configuration details (e.g. OS and product details, versions, hardware and software model); memory dumps, trace logs, debug DLLs, steps to reproduce, visual information (e.g. screenshots of software interface, videos); simulator or application used for testing; and observed and expected behaviors of the system. The result of this exercise is provided as a chart in Fig. 3. We found that 'steps to reproduce' ($\bar{x} = 4.63$) scored highest as the most preferred item for reproducing bugs, whereas the use of 'trace logs' ($\bar{x} = 4.5$) was also highly rated. 'Debug DLLs' ($\bar{x} = 3.25$) and 'simulator/application' ($\bar{x} = 3.56$) scored lower in our questionnaire, which suggests that these items served a niche requirement of our participants. These two items were by no means less important to the developers; in fact, these were used when other items turned out not to be helpful in reproducing a bug.

5.3 After a Bug is Reproduced…

Once developers are able to reproduce bugs, the fixing of bugs and further actions become clearer. Developers mentioned two main advantages of successful bug reproduction: (1) reducing the amount of fixing efforts and (2) giving confidence to developers in future actions on the bug.

When developers successfully reproduce bugs, using for example trace logs or memory dumps, they are actually able to locate specific lines of code where there is a problem. So, rather than looking and checking through a large amount of code—spread across different files, they reduce their efforts by going directly to the point where they need some fixing. Additionally, this also reduces their efforts for debugging, as the reproduction process indicates where a problem situated. The following are comments from two different developers:

> If I get steps to reproduce a bug, I will not have to look at other flows in the code. I just have to follow the flow that is described in the bug reproduction steps. For example, if there is a crash in a system, there could be more than one reason why it crashed. But it is important to know that during which activity it crashed, if we know these last two or three steps of the user then we are able to point out what exactly caused the crash.
>
> Bug reproduction helps in reaching functional level problems. It in fact gives a shortcut to reach to the problem, without having to go through the whole code.

In addition to reducing the overall efforts, bug reproduction also helps in supporting and informing further activities on bugs. Once a bug is reproduced, developers have to provide an analysis of the impact of the bug fix and provide details about what needs to be changed in the current system, what other features will be affected by the change, where else changes will be needed and provide test scenarios, among other things. As a testing team and a QA team will be part of the bug-fixing activity, they will provide their input on the proposed fix and analysis provided by the developer. The testing team will create their own test beds based on developers work and the QA team will verify the quality of the fix and provide their feedback on the bug tracker itself. The product manager and other senior level stake-holders would then take a call on how to proceed: whether to send this fix to the customer or add it to the next product release.

The questionnaire that we developed also had a question related to this part of the bug reproduction. Based on the observation sessions and interview feedback from our developers, we wanted to validate certain categories based on the question: 'how does bug reproduction help'. From their qualitative feedback, we selected six categories: understanding of bug patterns, limiting debugging efforts, locating fault in code areas, creating test cases, impact analysis and ensuring bug fix. On a five-point Likert scale, we asked all the 16 participants to rate these categories. As we described in the previous section, our main intention here was to validate the feedback of our participants and compare the importance of these individual items with one another. The result of this exercise is provided as a chart in Fig. 4. This figure shows that there is no strong difference among these categories. We found that 'understanding the bug pattern' ($\bar{x} = 4.29$) and 'ensuring

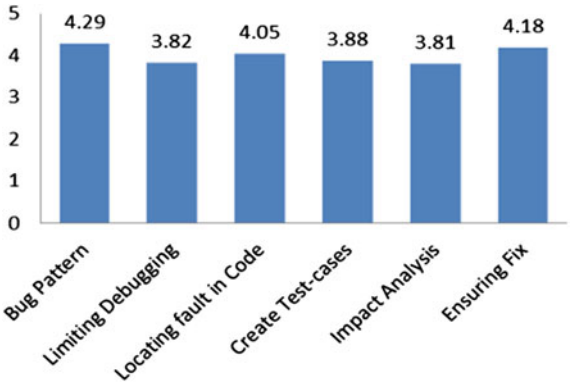

Fig. 4 Average rating of our participants' perceived advantages of bug reproduction (n = 16)

bug fix' ($\bar{x} = 4.18$) scored relatively higher. As such these two categories do not lead to any technical improvements for developers and other stake-holders. Rather, these two results point to the improved confidence level of developers. Understanding the bug pattern and getting an assurance of fixing a bug are the two subjective advantages a successful bug reproduction brings. Other categories such as impact analysis ($\bar{x} = 3.81$), creating test-cases ($\bar{x} = 3.88$) and locating fault in code areas ($\bar{x} = 4.05$) are the examples of technical improvements that bug reproduction supports.

6 Discussion

Studies [14, 17, 19] have shown that software developers spend a large amount of time on code evolution, bug-fixing and other maintenance related activities. From an HCI perspective, we have brought out the social side of software bug-fixing—in particular the practices related to bug reproduction. Unlike the studies done in the software engineering community [4, 15, 16, 20, 26], our study has focused on gaining access to the *in situ*, natural practices of developers working on real-world problems.

6.1 Challenges to Bug Reproduction

Our findings show that bug reproduction is a highly communication intensive activity. In our study, we found three major challenges to current bug reproduction practices: (1) lack of details from customers, (2) tedious logistical efforts, and (3) contextual issues of bugs.

Bug reproduction relies heavily on the inputs from customers. We found that developers often find insufficient information for reproducing bugs provided by

customers. As a result they based their bug reproduction on previous experiences and hunches. This was shown in one of the examples where a developer who was provided with a screenshot of a bug, had to use his hunch to determine the OS of the customer's machine and had to try several different OS versions. Similarly, logistical efforts needed to reproduce a bug also posed a challenge. When a bug was reported, in order to reproduce it developers needed to configure their machine and apply the same setting in which the bug was reported at the customer's site. This would involve changing the OS of their local machine for the version on which the buggy software product runs. Importantly, when such an effort is required, developers had to interrupt their on-going work and carry out changes in their machines and get back to the original settings when a bug is reproduced and fixed. Thirdly, the contextual issues at a customer's site may not be easily predictable. In such cases, a customer may not be aware of the information that is required by the development team for solving a problem. For example, in the cases of a system crash or a system hang, it was not always possible for a customer to know the previous steps that led to the problem—which are typically required to reproduce such a bug.

6.2 Interaction with Customer

Developers rarely had direct interactions with customers for discussing bug reproduction related issues. The local and global support centers facilitate communication between both sides. In some cases, product managers of development teams get involved as mediators in this chain of communication. The local and global support professionals are not technically skilled to understand bugs in detail or to know if the information provided by customers is sufficient. Additionally, the language used by bug reporters may be very different from customer to customer. In many cases, customers would only be able to talk about the UI related interactions. Another issue when interacting with customers is that developers often face difficulty in accessing required information. At times, when developers would like to get access to trace-logs from a customer's machine they need to properly instruct their customers on how to install such patches that will yield trace-logs.

6.3 Practical and Subjective Sides of Bug Reproduction

Bug reproduction offered both practical and subjective advantages. On the practical side, bug reproduction helped developers locate the area where bugs were present in a quicker way, allowed them to carry out an impact analysis and helped them in creating use cases and scenarios for developing their fixes. Importantly, we also observed that there was a strong experiential side to bug reproduction activities. The main purpose of carrying out bug reproduction is for developers to

be able to observe and experience how a bug occurs and how it behaves. This experience of being able to observe a bug is what adds to the confidence level of developers. Our results have shown the perceived advantages that a successful bug reproduction brings, such as increasing developers' confidence level by ensuring the bug fix and by providing indications about bug patterns. These aspects do not bring any technical advantages to developers, but they are perceptual and experiential in nature [19].

7 Implications

One of the most important aspects of bug reproduction is that it facilitates development teams not only to visualize a problem on their own machines, but in the process of reproducing a bug, it provides useful information about future activities related to bug-fixing. However, bug reproduction brings several challenges. A major problem that we observed was about communicating the *right* information, as in some cases customers provided insufficient information and in other cases customers did not know what information needs to be provided.

There are two important implications on the topic of supporting communication between customers (or bug reporters) and development teams. One implication is on developing tools that support customers in sufficiently understanding the encountered problems so that they can better provide appropriate and relevant information to the developer for fixing the bug. For example, tools may be developed to monitor activities of customers and on request or during a problem display records of these activities on an abstraction level that is adequate for the customer. This will empower customers to keep an account of their activities and provide relevant details at the time of reporting bugs. The second design implication is about supporting the automatic retrieval of 'relevant' information from a customer's local setup and making them available to the developers'. One of the ways this can be done is through the use of tracing mechanisms. Developers can build tracing mechanisms as a part of the software product that can be used to trace data related to the software usage in the field. This type of tracing could, for example, have different levels (mild, normal, extreme) which can be changed during runtime. Since tracing may increase the load on the software, the feature can be adapted based on the required level of detail. This way, whenever there is a bug reported in the system a responsible developer can easily extract the trace-log and can extract details about what led to such a bug. Privacy can be a major issue here as bugs reported by customers may have sensitive information. An approach similar to Castro et al. [6] could be explored to ensure that real values and data does not get misused. In addition to collecting the activity logs and trace logs, systems could also collect some contextual information related to the software setup (e.g. OS details) and configuration at place at the customer's site.

A bug tracking system is central in supporting communication and coordination between different parties involved in bug-fixing activities. Apart from some informal discussions on the phone, all important information is provided in the bug tracker. To deal with the issue of insufficient information provided by a customer, a template-based approach in bug trackers can be used. In this case, the bug tracker can have a dedicated section for bug reproduction where customers and support centers need to provide all the relevant information that may be required by a developer. Certain details can be made mandatory, for example, providing software product details, customers' system configurations, bug descriptions and other relevant details. Although, we did not generate any strong evidences in our research, developers tend to agree that certain information is required to deal with certain type of bugs. For example, in the cases of a software crash and hang, the use of trace-logs becomes very important. Similarly, a memory leak issue could also be a potential reason for software crash or hang, and in such cases memory dumps are also required by developers to study the bug. By making such details mandatory in a bug tracking system, a lot of time can be saved from a developers' point of view. Even better would be to dynamically adapt the template to the kind of bug reported and the system used, given that different systems might require different information for bug reproduction. We suggest that a detailed study can answer what kind of information is required by developers for specific bug types.

Our findings show that there are multiple people involved in the bug reproduction cycle, e.g. customer support professional, product managers, developers, testers and customers. Bug tracking systems should be appropriately integrated into the work environments of these different stakeholders, for example, a simple and natural language interface support for customers. Bug tracking systems can also add features for bug annotations or adding metadata so that better searching, filtering and sorting can be supported [3].

8 Conclusion

Bug reproduction is a social activity that involves participation from several stake holders besides developers and customers. Our findings show that the role of customers goes beyond merely reporting bugs. In fact, their interactions and inputs are needed at various stages of bug reproduction. From an ethnographic field study in an industrial setting, we examined current practices of bug reproduction and elicited challenges that developers face. Our results showed that developers find 'steps for reproduction' and 'trace logs' to be the most important information for reproducing bugs. At the same time, it showed that bug reproduction is as much a confidence building measure as a technical procedure that developers follow at the beginning of a bug-fixing activity. Based on our findings, we also provide several design recommendations such as the use of tracing and monitoring mechanisms,

adding new features (templates and annotations) to bug tracking systems and appropriately integrating them into the work environments of different stakeholders.

References

1. Ackerman MS, Halverson C (1999) Considering an organization's memory. In: Proceedings of computer supported cooperative work, ACM, pp 39–48
2. Avram G, Bannon L, Bowers J, Sheehan A, Sullivan DK (2009) Bridging, patching and keeping the work flowing: defect resolution in distributed software development. Comput Support Coop Work 18(5–6):477–507
3. Bertram D, Voida A, Greenberg S, Walker R (2010) Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams. In: Proceedings of the ACM conference on computer supported cooperative work, ACM, pp 291–300
4. Bettenburg N, Just S, Schröter A, Weiss C, Premraj R, Zimmermann T (2008) What makes a good bug report? In: Proceedings of the 16th ACM SIGSOFT international symposium on foundations of software engineering, pp 308–318
5. Breu S, Premraj R, Sillito J, Zimmermann T (2010) Information needs in bug reports: improving cooperation between developers and users. In: Proceedings of the 2010 ACM conference on computer supported cooperative work, ACM, pp 301–310
6. Castro M, Costa M, Martin JP (2008) Better bug reporting with better privacy. ACM SIGARCH Comput Archit News 36(1):319–328
7. Corbin J, Strauss A (eds) (2008) Basics of qualitative research: techniques and procedures
8. Grinter RE (1996) Supporting articulation work using software configuration management systems. Compute Support Coop Work (CSCW) 5(4):447–465
9. Gutwin C, Penner R, Schneider K (2004) Group awareness in distributed software development. In: Proceedings of the 2004 ACM conference on computer supported cooperative work, ACM, pp 72–81
10. Halverson CA, Ellis JB, Danis C, Kellogg WA (2006) Designing task visualizations to support the coordination of work in software development. In: Proceedings of the 2006 20th anniversary conference on computer supported cooperative work, ACM, pp 39–48
11. Herbold S, Grabowski J, Waack S, Bünting U (2011) Improved bug reporting and reproduction through non-intrusive GUI usage monitoring and automated replaying. In: IEEE 4th international conference on software testing, verification and validation workshops, pp 232–241
12. Herbsleb JD, Mockus A, Finholt TA, Grinter RE (2000) Distance, dependencies, and delay in a global collaboration. In: Proceedings of the 2000 ACM conference on computer supported cooperative work, ACM, pp 319–328
13. Holtzblatt K, Wendell JB, Wood S (2005) Rapid contextual design: a how-to guide to key techniques for user-centered design. Elsevier
14. Jones C (2013) The economics of software maintenance in the twenty first centry. Unpublished manuscript. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.88.7697. Accessed 22/11/2013)
15. Just S, Premraj R, Zimmermann T (2008) Towards the next generation of bug tracking systems. In: IEEE symposium on visual languages and human-centric computing, VL/HCC, IEEE, pp 82–85
16. Ko AJ, Myers BA, Coblenz MJ, Aung HH (2006) An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. IEEE Trans Softw Eng 32(12):971–987

17. LaToza TD, Venolia G, DeLine R (2006) Maintaining mental models: a study of developer work habits. In: Proceedings of the 28th international conference on software engineering, ACM, pp 492–501
18. Lientz BP, Swanson EB, Tompkins GE (1978) Characteristics of application software maintenance. Commun ACM 21(6):466–471
19. Martin D, Rooksby J, Rouncefield M, Sommerville I (2007) 'Good' organisational reasons for 'Bad' software testing: an ethnographic study of testing in a small software company. In: 29th international conference on software engineering, ICSE, IEEE
20. Müller S, Fritz T (2013) Stakeholders' information needs for artifacts and their dependencies in a real world context. International conference on software maintenance (ICSM), IEEE, pp 290–299
21. Ohira M, Hassan AE, Osawa N, Matsumoto KI (2012) The impact of bug management patterns on bug-fixing: a case study of eclipse projects. In: 28th international conference on software maintenance, IEEE, pp 264–273
22. Olson JS, Olson GM (2003) Culture surprises in remote software development teams. Queue 1(9):52
23. Schmidt K, Simone C (1996) Coordination mechanisms: towards a conceptual foundation of CSCW systems design. Compute Support Coop Work 5(2–3):155–200 (Kluwer Academic Press)
24. Schroter A, Bettenburg N, Premraj R (2010) Do stack traces help developers fix bugs? In: 7th IEEE working conference on mining software repositories, pp 118–121
25. Star SL, Griesemer JR (1989) Institutional ecology, 'translations' and boundary objects: amateurs and professionals in Berkeley's museum of vertebrate zoology, 1907–39. Soc Stud Sci 19(3):387–420
26. Tellioğlu H, Wagner I (1999) Software cultures. Commun ACM 42(12):71–77
27. Zhang F, Khomh F, Zou Y, Hassan AE (2012) An empirical study on factors impacting bug-fixing time. In: 19th working conference on reverse engineering (WCRE), IEEE, pp 225–234