



An Empirical Study of Security Requirements in Planning Bug Fixes for an Open Source Software Project

Saad bin Saleem and Yijun Yu and Bashar Nuseibeh

10 January, 2012

Department of Computing
Faculty of Mathematics, Computing and Technology
The Open University

Walton Hall, Milton Keynes, MK7 6AA
United Kingdom

<http://computing.open.ac.uk>

An Empirical Study of Security Requirements in Planning Bug Fixes for an Open Source Software Project

Saad bin Saleem¹ Yijun Yu¹ Bashar Nuseibeh^{1,2}

¹Centre for Research in Computing, The Open University, United Kingdom

²Lero – The Irish Software Engineering Research Centre, Ireland

{s.b.saleem, y.yu, b.nuseibeh}@open.ac.uk

Abstract— it is often difficult to estimate the resources needed to plan for bug fixing activities in software development projects. Security bug fixes are commonly implemented as patches in response to emergent common vulnerability and exposure (CVE) reports. In this paper we investigate how to plan for bug fixing, and whether security related bug fixes are different from other bugs. In a preprocessing step, we classify security and non-security bugs by using a definition of security requirements to elicit the keywords such as 'protection', 'assets' and 'malicious attackers', and by ranking their frequency of occurrences in the bug descriptions. We then create two release-planning inputs: one about the entire bug fixing activities, and another about bug fixes related to security requirements only. The results of the release plans are compared, with the bug fixing events recorded in the software repositories. Through a Samba case study, we show that it is possible to fix more high-priority bugs within limited given resource, and that bugs related to security requirements are materially different from other kinds of bugs.

Keywords- *Security Requirements; Release Planning; Bug fixes; Empirical study; Open-Source Software;*

I. INTRODUCTION

Evolving open source software projects are increasingly common. Many have become an integral part of security-critical software products [1]; which is evident from the growing number of Common Vulnerability Exposure (CVE) records¹ related to open source projects. Although release planning optimises the number of important features deliverable within the given resources [8], [10], to achieve the same for security-critical open-source projects is hard, primarily for two reasons: (1) it is hard to estimate the amount of resources available for the open-source projects due to the global distributed development with developers participating on the volunteering basis, and their effort is hard to measure or quantify; (2) it is difficult to anticipate the occurrences of security issues because they are often regarded as “emerging” known unknowns. It is known that release planning should be performed as earlier as requirements are collected to find optimal opportunity to manage the delivery. However, in practice of release planning for open-source projects, requirements are rarely available in the documentation; and release planning is often regarded as an after-thought: normally recording problems that have been fixed, rather than problems to be solved. However, change management (e.g., CVS) and issue tracking (e.g., Bugzilla) systems are used widely in such projects, making it possible to obtain event-based data. This paper reports our experience of the empirical case study of applying the ReleasePlanner, an industrial release planner, to the data extracted from an open-source project. In this study, our main related research question is: (1) to what extent the tool supported release planning using ReleasePlanner is useful to plan the security requirements bug fixes for an open source project. The main question is further subdivided into the following questions,

RQ1: How useful is the tool supported release planning using ReleasePlanner for planning bug fixes of an open source project?

RQ2: What is the difference between planning all the bug fixes in comparison to only security requirements bug fixes?

The approach we took is an in-depth case study of a representative open-source project. The experiments are conducted systematically and the method is applicable to other security-critical open-source projects as long as they also use issue-tracking systems. Unlike existing approach on analysing evolving quality requirements of open-source projects [7], in this work, we focus the analysis on security requirements and select the bug reports that explicitly refer to the “protection”, “assets” and “malicious intentions”. Furthermore, the keywords to be used

¹ <http://cve.org>

to query the bug reports were derived from the descriptions of CVE records that are relevant to the case study, rather than be subjectively chosen by the analysts. In order to assess the benefits of the gain of release planning, we also compare the results of the planned bug fixes with those happened in the past during the project history. The average time to fix any type of bug e.g. functional, or other non-functional except security is compared with security requirement bug fixes.

The remainder of the paper is organised as follows. Section II overviews the objective and nature of the research questions, explains the rationale for why we chose the open-source Samba project for the case study. Section III details the steps we took to mining the data from the publicly available software repository, transforming it into inputs for the ReleasePlanner, and then comparing our results to the outputs of the ReleasePlanner. Section IV attempts to interpret the results of our comparison, and reports on our general experience and lessons learnt. Finally, Section V discusses some threats to the validity of our study, Section VI compares related work, and Section VII concludes.

II. MOTIVATION AND CHOICE OF CASE STUDY

There is a common perception that one cannot always plan before the security incidents emerge during software development. However, there is also a little evidence that one cannot plan a project for security requirements that were known at the time of planning. To check this perception, however, it is hard to collect any evidence from commercial projects where security-related assets are usually restricted from a direct access by researchers. On the other hand, security issues are increasingly more important in essential evolving open-source products [1], such that they are openly discussed and recorded into a database of an associated issue-tracking system. This availability of data presents one with an opportunity to look at the feasibility of planning, security solutions ahead of the potential incidents. Although advanced release-planning tools have been applied to several commercial projects [8], [11], there is no evidence that such tools have ever been applied to an open-source one. Therefore, the purpose of first research question is to check applicability of the tool supported release planning for any types of functional and non-functional bug fixes. On the hand, the second question is more focused on any difference between the security bug fixes with any other type of functional and non-functional bug fixes. To assess these questions, one has to choose an open-source project that is security-critical, whereby release planning has been carried out manually so that it makes sense to compare the effectiveness with the tool-supported alternative. The Samba project matches our search criteria, because its developers openly discuss release planning in the wiki² and provide both release notes and bug reports online³. It is one of the few projects classified by the “authentication” label in almost all archived open-source projects⁴. Moreover, at the point of study there have been already 39 CVE database records directed to the Samba project. The Samba product provides file and print services to SMB/CIFS clients. Here Server Message Block (SMB) is a dialect of the Common Interface File System (CIFS) protocol, a transport independent protocol that provides a mechanism for client systems to use file and print services on servers. Samba is a free distribution of this protocol to support the communications between a Linux file server and any Microsoft Windows client.

III. DATA COLLECTION AND ANALYSIS

To assess our research questions, we have selected data from release notes and bug reports according to the following rationales from the project website.

A. *Treating open bugs as requirements for planning*

In most of the open-source projects requirements are not directly documented, instead in the Samba project, they are embedded in bug reports: those bugs classified as “enhancement” reflect new requirements, whilst those “error fixes” (including most security fixes) indicate that certain existing requirements (including security requirements) are not satisfied. The developer may consider implementing these tasks later, when the status of these bugs is either “NEW” or “OPEN/REOPEN”. Although these bug reports do not necessarily reflect the early

² <http://wiki.samba.org>

³ <http://bugs.samba.org>

⁴ <http://ohloh.org>

requirements, one could consider them as “late requirements” that are appropriate for the planning. In remainder of the paper, we call the bugs that have been proposed but not yet fixed at the time of release planning as “open bugs” or interchangeably “requirements”. Of course, not all the requirements are necessarily reported as bugs. Ideally one would find out all the requirements from the source code that are generally available in the open source projects. However, in relation to the abstraction level of requirements for strategic release planning, we chose not to include source code in our data collection step because the traceability links between bugs and code are as low as 30% for most open-source projects, according to the studies reported by Devanbu et al. [2].

B. Open bugs selection criteria

All the bugs that had not been fixed at the time of planning a major release were selected as open bugs. Therefore, any bug open at the time point of planning a major release is selected in the list of open bugs to plan. In the Samba project, a major release is delivered after approximately the nine months’ time span. There are a lot minor releases to fix bugs are not included in the planning because bug fixes is the only purpose of minor releases and there are no additional features request are included in such releases. For security requirements bug fixes, we have adopted the Haley’s et al. [6] definition that security is about protecting important assets of a system from the harm caused by the malicious attacker. Therefore, all the bugs related to protection of the system are considered as security related while assets related bugs can be based on the business requirements and malicious intension have to do with the stakeholders.

C. Selecting open bugs from the project documentation

The bug reports and Release Notes are the two sources of collecting open bugs. In the bug reports, planning related data tells us the what (description of the bug id), who (the reporter of bug), the when (the time when the bug was initially reported and the time when the bug was fixed), and the How Much (severity assessed by the reporters) each requirement is handled in the evolution history of the project. Moreover, the bug reports include the dependencies among different bugs, and the status of the bug in its activity (NEW, REOPEN, ASSIGNED, FIXED, CLOSED) etc. Therefore, we selected the each bug-id, the bug description, the open and close date of the bug from all the bug reports available at samba repository of bugs³. This data collection from bug reports is done using the scripts mentioned at our online data collection repository⁵. We used scripts for data collection instead of manually looking at the each bug description because the scripts helped us to collect large bench of bugs efficiently and accurately. The risk of miss match between the given information on samba website and bug reports was higher in the manual data collection.

The data about all the major releases of samba was collected from the Release Notes available at the samba project website. In the Release Notes all the “notable” changes happened since the previous release are recorded. Noteworthy changes in such release notes (as highlighted in separate sections of the document) are those bugs fixed including the security ones to the public awareness of CVE records. For the Sample project, these fixes are called “Security Patches” and the corresponding patched releases are called “Security Releases”.

To select the security requirements bug fixes, we compared the security bugs whose descriptions have the higher $tf*idf$ (term frequency times inverse document frequency) weighted sum against a list of query keywords. The list of query keywords are derived from the calculation of all the security bugs in relation to the CVE entries and a small set of non-security bugs checked manually. In addition, we filtered out all the stop words after applying the Porter stemming algorithm [21] to the keywords and we removed any words that have nothing to do with the concept of “protection”, “asset” and “malicious intentions”. In this way, only a smaller set of keywords are used to query all the bug descriptions using the standard vector-space based similarity calculation of information retrieval techniques. As a result, the bugs whose descriptions match with this selected security requirements-related keywords are used to perform release planning, as opposed to using all the bugs⁵ that matches with handful domain-independent keywords in the ISO 9126 standard [7].

⁵ http://sead1.open.ac.uk/samba_analysis/

D. Preparing Input for the ReleasePlanner

In our assessments, all the above-mentioned data in release notes and bug reports are relevant to the release planning as if it was conducted at the beginning of the major releases. Current Samba release notes, however, are only documenting the bug fixes that have happened by the end of the releases, rather than recording the bugs to be fixed for the future releases. In order to generate the optimal release plan, therefore, some processing is needed to turn the data we gathered into an input for the ReleasePlanner tool [3]. Table I depicts the mappings from the Samba documentation of various sources to the input for the ReleasePlanner. Open bugs are selected as the planning objects because they are required to be solved in the future of the planning time.

TABLE I. MAPPINGS DATA COLLECTED TO INPUT OF THE RELEASEPLANNER

From: documentations	On: sources	To: ReleasePlanner
Release dates duration	Release notes	Resources to plan in days
Major release numbers	Release notes	Release names
Bug id	Bug reports	Planning object (requirement) id
Bug description	Bug reports	Requirement name
Bug reporters email address	Bug reports	Stakeholder names and associated requirement
Bug report fixed date - Bug report creation date	Bug reports	Maximum resource to implement the planning object
Bug severity	Bug reports	Urgency
Bug classifications by "security" terms	Bug reports	Security requirements group
Bug classification by "CVE" numbers	Release notes	public security requirements group
Other open bugs	Bug reports	Other requirements

1) Preparing data to answer RQ1

To assess the RQ1, we prepared the corresponding inputs for the ReleasePlanner to compare with the manual release planning for the two major releases according to the wiki documentation (i.e., 3.0.25, 3.2.0). We collected the total number of features delivered in each release to compare the "stakeholder features points" -- the objective function of the ReleasePlanner as the sum of all the features (i.e., bug fixes) delivered in the releases weighted by the priority of bug fixes by generating the optimal solution.

2) Preparing data to answer RQ2

To assess the RQ2, first we collected only open security bug fixes at time point of the each major release from the set of all the open bugs. Then, we have calculated overall resource to be utilized in reality for security bug fixes. In next step, the estimated total time (resource) to spend on fixing the each open security bug is calculated. The stakeholder priorities and preferences for only open security bugs are selected from the data of all the open bugs. At-last, two input Excel sheets are prepared for the two major releases 3.0.25 and 3.2.0.

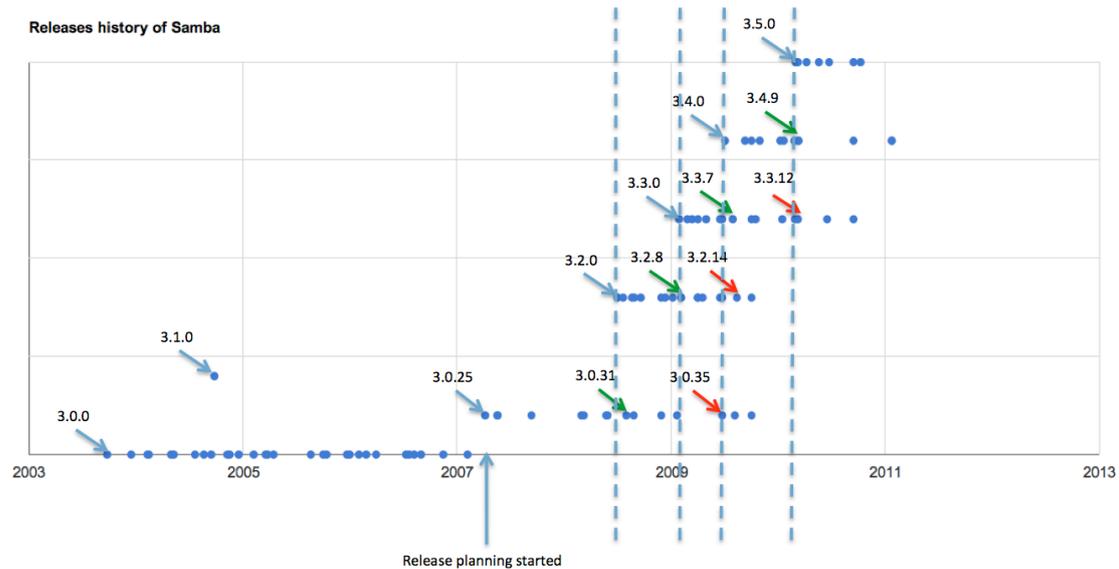


Figure 1: The evolution history of Samba releases: release planning starts at 3.0.25, and maintenance releases start at the beginning of next release series, security releases start at the beginning of the next maintenance series, and release series discontinue at the beginning of the next security release series.

IV. RESULTS AND LESSON LEARNED

During the evolution history of Samba (see Figure 1), 97 releases were documented in the 67 release notes on the wiki, among which only 67 releases since 3.0.25 are really used in the manual release planning process. There is no further release on the 3.1.x series besides the one (i.e., 3.1.0) created temporarily for beta testing. Therefore the Samba project at releases 3.0.25, 3.2.0, 3.3.0, 3.4.0 and 3.5.0 are suitable time points for release planning when the tool-supported release planning could be simulated and compared. In reality, these cover duration of 452, 210, 157, 241 and 221 days respectively. Informed by these durations, we could configure the ReleasePlanner to plan respectively on the exact dates of these major releases 6-April-2007, 1-July-2008, 27-Jan-2009, 3-July-2009 and 1-March-2010. The documentation says that the team wishes to have regular release cycle of nine months to deliver a major release, which is only roughly followed. Each of these major releases has a number of subsequent minor releases. According to the documentation, there are four modes in these subsequent minor releases for each branch starting at 3.x.0: upon creation of a major release series, the previous release series turns into the maintenance mode for bug fixes, and the previous maintenance series turns into security fix model to resolve security issues; and the previous security fix mode series turns into discontinued mode with no further changes. Therefore the planned time span of maintenance mode shall be 18 months with 9 months each for bug fixes and security fixes modes. The followings are results and observations we experienced that are specific to the research questions raised earlier.

A. Release Planning for the Open bugs (RQ1)

Concerning RQ1, how useful is the tool supported release planning using ReleasePlanner for planning bug fixes of an open source project, we conducted research by preparing the input for the ReleasePlanner tool from the data collected from the release notes and bug reports analysis. The following are my For preparing input, w

It is very important to decide about the number of releases to plan before release planning of any project. Therefore, the first input is a list of releases to plan and the each release priority. As mentioned earlier, the data of five major releases is included in this study for the planning purpose which is collected from the release notes. The sixth release is not included in our study as it was in the process of delivery at the time when this study is conducted. An equal priority level is assigned to each release as no such priority is available in the Samba project documentation.

The second input is a list of open bugs as “Planning objects”. A bug report is known to be “OPEN” at time T only if $t1 \leq T$ and $t2 > T$, where t1 and t2 are respectively the time when the bug was reported and when the bug was fixed (if it has been fixed ever). This avoids including the problems that were already solved ($t2 \leq T$) or unknown at the time of planning ($t1 > T$).

The third input is a list of total time spent on fixing the each bug. The bug fix time is calculated based on a bug’s opening and closing date. Given that there are 40 developers in the entire project history and not all of them are full-time developers, the amount of resources to be allocated must be smaller than 40 times the project release duration.

The fourth input is stakeholders’ priorities. These are filled by the email addresses from the reporter of the bugs, and the severity/priority of the bugs respectively.

After preparing these data, we could ask the ReleasePlanner for the optimal plans. According to our experience, it took less than 30 seconds for the tool to import one configuration file we prepared as Excel spread sheets, while it took only 5 seconds to optimise. Without our pre-processing scripts (which by itself took about an hour to run including the time of crawling the websites of Samba), it would have taken weeks to import these data for one release plan of this scale. Although it is certainly an area for improving the ReleasePlanner tool, it is not a blocker for us to assess the feasibility of release planning for the Samba open-source project. Figure 2 and 3 shows the results of the release planning simulated at the time 7 April 2007. The total number of open bugs of Samba planned here are 242, among which 110 have been definitely fixed over the period of the remaining life of the project until the time of our assessment (i.e., 5-Oct-2011). First we estimated for each fixed bug that the effort could not be more than the response time between the day the report was created and the day when the bug was fixed. For each unfixed bug, the average effort of those fixed ones (i.e., 199.82 days on average for all the 4539 fixed bugs from total of 7770 bugs) is used to estimate their effort. Using the feature of ReleasePlanner to define pre-assigned releases for every fixed open bug, we were also able to calculate the objective function of the ReleasePlanner: the stakeholder feature points on those manually planned releases. For example, the 110 open bugs that had been fixed after 7 April 2007 were assigned to the corresponding releases in which the fixes had happened. By estimating the effort for each of these bugs, we could tell the ReleasePlanner the maximal resources for each release that was only sufficient for those pre-assigned open bugs to be fixed according to what had happened in the reality. In this case, the total number of days of estimation for the 3.0.25, 3.2.0, 3.3.0, 3.4.0 and 3.5.0 releases are respectively allocated by 25277, 39105.5, 58387, 42392 and 30917.5 person-days, which is just enough for the 110 bugs to be fixed according to what had happened in reality. In other words, there are still 132 bugs remaining to be fixed according to this manual plan. As an indicator of the manual planning performance, the total stakeholder feature points calculated by the ReleasePlanner is 530200 for this one and only one “optimal” plan (see 2).

Explanation	Alternative 1
Degree of optimality	 100.0%
(Stakeholder feature points)	(530200)
Release	Alternative 1
3.0.25	50
3.2.0	12
3.3.0	29
3.4.0	12
3.5.0	7
Total	110 of 242

Figure 2: The “plan” reflecting the reality of Samba for the open bugs at release 3.0.25, showing the stakeholder feature points calculated by the ReleasePlanner tool

Explanation	Alternative 1	Alternative 2	Alternative 3	Alternative 4	Alternative 5
Degree of optimality					
(Stakeholder feature points)	100.0% (1081285)	99.7% (1078276)	99.6% (1077070)	99.5% (1076668)	99.5% (1076314)
Release	Alternative 1	Alternative 2	Alternative 3	Alternative 4	Alternative 5
3.0.25	24	26	49	42	25
3.2.0	21	17	27	28	18
3.3.0	132	135	108	112	135
3.4.0	23	23	19	20	21
3.5.0	9	11	7	8	12
Total	209 of 242	212 of 242	210 of 242	210 of 242	211 of 242

Figure 3: Best five plans produced by the ReleasePlanner, showing much higher “stakeholder feature points” comparing to Figure 2.

Explanation	Alternative 1
Degree of optimality	
(Stakeholder feature points)	100.0% (1330314)
Release	Alternative 1
3.2.0	129
3.3.0	57
3.4.0	35
3.5.0	17
Total	238 of 401

Figure 4: Reality of Samba: 238 out of 401 open bugs at 3.2.0 are fixed later

Explanation	Alternative 1	Alternative 2	Alternative 3	Alternative 4	Alternative 5
Degree of optimality					
(Stakeholder feature points)	100.0% (2262800)	99.8% (2257555)	99.8% (2257555)	99.7% (2256605)	99.7% (2256605)
Release	Alternative 1	Alternative 2	Alternative 3	Alternative 4	Alternative 5
3.2.0	48	50	53	51	51
3.3.0	219	219	216	220	220
3.4.0	51	51	51	51	51
3.5.0	18	18	18	18	18
Total	336 of 401	338 of 401	338 of 401	340 of 401	340 of 401

Figure 5: The five best plans regenerated by the ReleasePlanner starting at release 3.2.0 comparing to the reality in Figure 4.

In contrast, when the same amount of resources was given to the release planning without the constraint of pre-assignments, 209 out of the 242 open bugs can be fixed according to the same effort estimation. It amounts to 1081285 stakeholder feature points (see3), a substantial increase of 103 % had the open-source project adopted the ReleasePlanner tool.

Since most open-source projects including Samba are evolving, release planning need to be conducted iteratively to address the new “known unknowns”. This is called “re-planning” [8] To illustrate this, we also

simulated the re-planning on 1-July-2008, or Samba 3.2.0, which already accumulated 401 open bugs (after fixing 52 of the 242 open bugs at 3.0.25) to plan for the remaining 4 major releases. As shown in Fig. 4 and 5, the number of “stakeholder feature points” increases from 1330314 to 2262800 by 70%.

B. Open Security requirements bug fixes(RQ2)

We have confirmed from the various sources that the Samba project is a security-critical: its security bugs have been announced to the public in the CVE records. Every major development branch of Samba has a maintenance release mode dedicated to address those CVE recorded security bugs; and it is classified in the open source archive ohloh.net as one of the 6 most relevant projects to the “authentication”.

However, we only found 30 security-fixed releases throughout the evolution history that are addressing as few as 39 public CVE records. The number of documented bugs that are associated with these CVE records is 23 (3 of them are with empty descriptions). Then we performed the tf-idf computations on the 20 descriptions of the bug entries against the descriptions of the total number of 4539 bugs. The method we used here is a standard information retrieval, by computing the term frequency using the occurrences of the words within individual bug descriptions, and by computing the inversed document frequency by the logarithm of division between the total number of bugs and the occurrence of the term in all bug descriptions). In computing the terms, we also used the Porter stemmer [21] and the stop-words removal to accumulate the frequencies of non-stop words that share the same stem as the frequency of the shared stem word. This produces a matrix of 59,152 terms (non-stop step words) as rows and 4539 bug descriptions as column vectors.

Out of the 1,068 terms of the CVE-related bug descriptions, we manually classified them into four categories: “protection”, “asset”, “malicious intentions” and “none of the above”. For examples, “audit”, “encrypt” are mechanisms for protection; “user-session key”, “root password” are assets; and “exploit”, “mask”, “attacker” reflect malicious intentions. Then we only used these 512 terms to formulate the query on security requirements.

Then we computed the cosine similarity between this query vector of all the 512 security-requirements related terms occurring in the 20 security bug descriptions and the 4539 document vectors altogether, and ranked them by relevance. Finally, we obtained 107 highest ranked bugs as the matching security bugs.

As a part of addressing the RQ2, we have checked that how many of security bug fixes are open at the time point of the each major release in comparison to all the other types of bugs. Figure 6 is summarizing the comparison of the data. First, we checked at the time of every major release, how many bugs were considered “OPEN” (they were confirmed by the developers to be addressed, as opposed to the “NEW” ones that were just reported)? Our Null hypothesis here was that “less than 10% of security bugs are OPEN at the time of release planning”. Out of the 512 bug reports in the evolution history, however, there are 512 OPEN in total, a much larger portion of which (12~27%) were still considered OPEN at the starting time of the five major releases. These facts completely rejected the null hypothesis. In addition, as shown in figure 6, substantial amount of the OPEN bugs (16.9~18.7%) are related to security in the five planning time points. For example, out of the 242 open bugs at the release 3.0.25, already 24 are related to security issues. In total, the number of security requirements (512) is much bigger than the number of public CVE numbered issues (39) reported in its history.

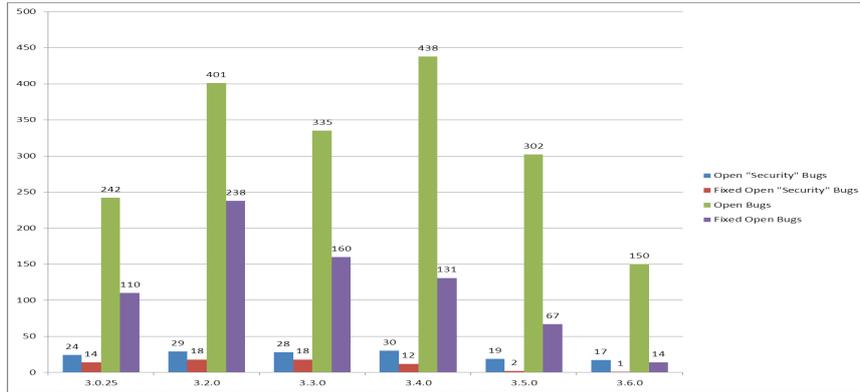


Figure 6:The number of OPEN “Security” bugs at the time of the release planning for the five major Samba releases, comparing with the number of OPEN bugs. Also shown are the number of such bugs that have been fixed since the previous major releases.

Now, in the next stage, we only plan the open security bug fixes to see any difference in planning of security releases in comparison to the planning of all the open bugs. The results in figure 7 shows a real plan indicating 14 out of 24 bugs fixed, while the optimized plan is delivery more stakeholder feature points as show in the figure 8. In contrast, the figure 9 and 10 are showing the same results at the time point of release 3.2.0.

Our findings demonstrate that the security fixes of an open source project are optimized in a similar fashion like the any other bug fixes. However, there is more time resource required in fixing the security fixes in comparison to all the other types of bugs.It is observed that the average effort of fixed security bugs is 534.64 days on average for all the 4539 fixed bugs; which is almost 3 times higher than the normal bugs average fixed days reported earlier. Therefore, it is evident that security bug fixes estimation needs to be done differently from the other bugs.

Explanation	Alternative 1
Degree of optimality	<div style="width: 100%; height: 10px; background-color: green;"></div> 100.0%
(Stakeholder feature points)	(21669)

Release	Alternative 1
3.0.25	6
3.3.0	7
3.4.0	1
Total	14 of 24

Figure 7:Reality of Samba: 13 out of 24 security requirements related bugs (RQ2).

Explanation	Alternative 1	Alternative 2	Alternative 3	Alternative 4	Alternative 5
Degree of optimality (Stakeholder feature points)	 99.2% (35073)	 99.2% (35039)	 96.7% (34162)	 94.3% (33340)	 94.3% (33340)
Release	Alternative 1	Alternative 2	Alternative 3	Alternative 4	Alternative 5
3.0.25	5	4	5	5	5
3.3.0	13	13	13	12	12
3.4.0	2	2	2	2	2
Total	20 of 24	19 of 24	20 of 24	19 of 24	19 of 24

Figure 8: The five best plans generated by the ReleasePlanner for security requirements related bugs (RQ2).

Explanation	Alternative 1
Degree of optimality (Stakeholder feature points)	 100.0% (31720)
Release	Alternative 1
3.2.0	4
3.3.0	11
3.4.0	3
Total	18 of 29

Figure 9: Reality of Samba: 18 out of 29 security requirements related bugs (RQ2).

Explanation	Alternative 1	Alternative 2	Alternative 3
Degree of optimality (Stakeholder feature points)	 100.0% (39345)	 96.4% (37910)	 95.8% (37684)
Release	Alternative 1	Alternative 2	Alternative 3
3.2.0	3	3	2
3.3.0	16	16	18
3.4.0	4	4	4
Total	23 of 29	23 of 29	24 of 29

Figure 10: The five best plans generated by the ReleasePlanner for security requirements related bugs (RQ2).

V. THREATS TO VALIDITY

C. Internal Validity

The data collected for the Samba project is based on the same set of documents, a normal developer and stakeholder would have seen. We interpreted the data based on audited facts such as database entries and timings, rather than based on interpretations of the natural language descriptions. The main threat to internal validity is the use of term “security” in search for security-related bug reports. Our purpose here is to find out whether there are

more security bugs than those emergent ones from CVE records. Therefore more security-related bugs found without matching keyword “security” would only increase the number of open security bugs at every planning stage, thus lower the ratio of security requirements that cannot be planned.

D. Construct Validity

Our assessment approach is replicable for other open-source projects as long as they use Bugzilla to report bugs and document the CVE numbers in a traceable manner down to the bug fixes. To compare a tool-supported release planning with that of manual ones, however, it requires the open-source projects to have certain release planning in the first place. The other threat to the construct validity is whether open bugs could capture most of the tasks for open-source developers (for example, not all the tasks developer perform are related to bug fixes). The estimation of resources for each bug fix task using the realistic data may also exaggerate the amount of time it took to fulfil these tasks. However, for this particular threat, even commercial projects cannot estimate accurately. An estimation based on the SLOC metric is perhaps a complementary way to address this issue. However, a study of the traceability between code and the bug fixes is beyond the current scope of this study.

E. External Validity

By choosing a well-documented open-source project in the public domain, we have avoided a number of concerns on the external validity for the case study. In order to make our study replicable by other researchers, we have made the scripts of the bug crawler, release notes parser and the results analysis available on the following website: http://sead1.open.ac.uk/samba_analysis.

VI. RELATED WORK

The selection of requirements in different releases is important to deliver software product on time and within the available resources; the uncertainty of the resources in open source projects further complicates the problem. Bangall et al., have modelled the next release problem to select the optimal plans [19]. In this paper, we are not modelling the problem instead we are using real data of open source project to compare the planning of security and non-security bug fix activities.

Gueorguiev et al., have proposed a search based technique to the problem of selecting a robust plan which balances multiple objectives [17]. In this work, we are not proposing any approach; rather this work is focus on the looking at the benefits of optimal planning in terms of delivering more weighted prioritized bug fixes of the stakeholders.

Antoniol et al. have evaluated three search based techniques (genetic algorithm, hill climbing and simulated annealing) for optimal project planning [18]. Precisely they have considered the developers as primary resource and ordered the Work Packages (WPs) according to the number of developers to complete the project within required time. They highlighted interesting result that increasing the size of developers does not improve performance for the large staffing levels. The staffing problem was constructed by using the two genome representations for the Y2K maintenance project. However in this paper, our scope of planning is releases instead of the whole project and we are using the automated tool ReleasePlanner for optimization instead of the manual search based techniques.

In another study Xiao et al. [20], have evaluated the genetic algorithm technique for resource scheduling of development and testing activities of the bug fixes. They concluded that in this way a better bug fixing policy can be followed to balance the different constraints in the each activity. Our approach is different as we have used the automated tool ReleasePlanner for the scheduling of bug fixing activity. Using such tool makes it more convenient to select the best alternative solution. Second, we have separated security requirement bugs from all the other types of bugs instead of considering all of them as one type.

F. Evolving Security Requirements

It is complicated to introduce changes for the secure systems, where vulnerabilities could be exposed at any time and implementation of new requirements could affect the security requirements of the system. According to Tun et al. [12] security experts face many difficulties while introducing a change in the system. Similarly, Nhlabatsi et al [9] have conducted a survey of existing techniques to deal with changing security requirements and found a gap

to manage evolving security requirements. For study of evolution of general quality requirements, Ernst et al [4] applied a natural language processing technique to the bug reports of several open-source projects in order to understand the evolution of quality requirements with respect to the ISO 9126 standard. Through the analysis of evolving bug reports of another case study, Wermelinger et al. [11] found that the social relationships between assignees and reporters can be inferred from the architectural relationships between software components. Therefore it is also possible that one can infer bug dependencies by combining the domain knowledge of software architecture.

G. Release Planning

Software systems are delivered in series of releases with additional functionalities during incremental software development [5], in which the decisions about what and when to release are very important. Release Planning (RP) is a guided way to help organisations decide which requirements to be delivered in which release (i.e., strategic) and which time to deliver the releases (i.e., operational). According to Ruhe et al. [5] a strategic RP is performed at the product level to select the optimal sets of requirements in a sequence of candidate releases, while the operational RP is done at the project level in order to plan the delivery of each individual release. Saad et al., [10], have performed a systematic review of strategic release planning approaches that include the Evolve* method, which is implemented by the ReleasePlanner tool. It is evident from an industrial case study that using ReleasePlanner is quite useful to plan for the evolving requirements of software maintenance projects [14]. However, the scope of current study is to see the applicability of such tool for security critical evolving open source project.

VII. CONCLUSIONS

The paper demonstrated that it is possible and beneficial to perform tool-supported release planning on an open-source project, Samba, especially for the known open security issues at the time point of release planning. Through this case study, we also reported the experience of how to turn OSS project documentation into ready-to-use release plans, and how security requirement can be collected from the security bugs using tf*idf. It is reported that security requirements bug planning is materially different from all the other bugs.

ACKNOWLEDGMENT

We are thankful to Dr. Guenther Ruhe and the team of Expert Decisions Inc for providing us access of their commercial tool the ReleasePlanner. We also want to greatly acknowledge Drs. Charles Haley, Thein Than Tun, and Hui Yang for their useful comments on the earlier draft of the paper.

REFERENCES

- [1] Cowan, C.; "Software security for open-source systems," *Security & Privacy, IEEE*, vol.1, no.1, pp. 38- 45, Jan.-Feb. 2003.
- [2] Premkumar T. Devanbu: Reverse Engineering the Bazaar: Collaboration and Communication in Open Source Development. WCRE 2008: 4.
- [3] Al-Emran, A.; Pfahl, D.; Ruhe, G.. "Decision Support for Product Release Planning Based on Robustness Analysis," *Requirements Engineering Conference (RE), 2010 18th IEEE International*, vol., no., pp.157-166, Sept. 27 2010-Oct.
- [4] Neil A. Ernst, John Mylopoulos, "On the perception of software quality requirements during the project lifecycle". Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ). Essen, June 30-Jul 2 2010.
- [5] D. Greer and G. Ruhe, "Software release planning: an evolutionary and iterative approach," *Information and Software Technology*, vol. 46, no. 4, pp. 243–253, 2004.
- [6] C.B. Haley, R. Laney, J.D. Moffett, and B. Nuseibeh. Security requirements engineering: A framework for representation and analysis. *Software Engineering, IEEE Transactions on*, 34(1):133_153, 2008.
- [7] Abram Hindle, Neil A. Ernst, Michael W. Godfrey, and John Mylopoulos. 2011. Automated topic naming to support cross-project analysis of software maintenance activities. In *Proceeding of the 8th working conference on mining software repositories (MSR '11)*. ACM, New York, NY, USA, 163-172.

- [8] Joseph Momoh and Guenther Ruhe. Release planning process improvement- an industrial case study. *Software Process: Improvement and Practice*, 11(3):295_307, 2006.
- [9] *Armstrong Nhlabatsi.et al*, Security requirements engineering for evolving software systems: A survey. *Journal of Secure Software Engineering*,1(1):54_73, 2009.
- [10] Saad Bin Saleem and Muhammad Usman Shafique. A study on strategic release planning models of academia and industry through systematic review and industrial interviews. Master's thesis, School of Engineering, Blekinge Institute of Technology Sweden, 2008.
- [11] Michel Wermelinger, Yijun Yu, and Markus Strohmaier: Using formal concept analysis to construct and visualise hierarchies of socio-technical relations. *ICSE Companion 2009*: 327-330.
- [12] Thein Than Tun, Yijun Yu, Charles Haley, and Bashar Nuseibeh. Model-Based argument analysis for evolving security requirements. In 2010 Fourth International Conference on Secure Software Integration and Reliability Improvement, pages 88_97, Singapore, Singapore, 2010.
- [13] Omolade Saliu and Guenther Ruhe. Supporting software release planning decisions for evolving systems. In *Proceedings of the 29th Annual IEEE/NASA on Software Engineering Workshop*, pages 14_26. IEEE Computer Society, 2005.
- [14] P. Bhawnani, G. Ruhe: ReleasePlanner® - Planning new Releases for Software Maintenance and Evolution, *Industrial Proceedings of the 21st IEEE International Conference on Software Maintenance*, Budapest, Hungary, September 25-30, 2005, pp. 73-76.
- [15] P. Bhawnani, G. Ruhe, F. Kudorfer, and L. Meyer, "Intelligent Decision Support for Road Mapping a Technology Transfer Case Study with Siemens Corporate Technology," *Technology Transfer*, pp. 35-40, 2006.
- [16] G. Ruhe: *Product Release Planning: Methods, Tools and Applications*, CRC Press, ISBN 10: 0849326206, publication date June 17, 2010, 336 pages.
- [17] Stefan Gueorguiev, Mark Harman, Giuliano Antoniol: Software project planning for robustness and completion time in the presence of uncertainty using multi objective search based software engineering. *GECCO 2009*: 1673-1680.
- [18] Giuliano Antoniol, Massimiliano Di Penta, Mark Harman: Search-Based Techniques Applied to Optimization of Project Planning for a Massive Maintenance Project. *ICSM 2005*: 240-249.
- [19] A. J. Bagnall, V. J. Rayward-Smith and I. M. Whittlely, (2001), *The Next Release Problem*, *Information and Software Technology*, Vol. 43, 14, p. 883-890
- [20] Junchao Xiao and W. Afzal, Search-based Resource Scheduling for Bug Fixing Tasks, in *2010 Second International Symposium on Search Based Software Engineering (SSBSE)*, 2010, pp. 133-142.
- [21] P. Willett, "The Porter stemming algorithm: then and now," *Program: electronic library and information systems*, vol. 40, no. 3, pp. 219-223, Jul. 2006.