Defect Bash - Literature Review

Xuejiao Zhou and Mika Mäntylä
Department of Computer Science and Engineering, School of Science, Aalto University, Espoo,
P.O. Box 19210, 00076 Aalto, Finland
{jiao.zhou, mika.mantyla}@aalto.fi

Keywords: Defect Bash, Literature Review, Software Quality, Software Testing, Verification and Validation.

Abstract: Defect bash is a co-located testing session performed by a group of people. We performed a systematic review of the academic and grey literature, i.e. informally published writings, of the defect bash. Altogether, we found 44 items (17 academic and 27 grey literature sources) that were identified useful for the review. Based on the review the definition of defect bash is presented, benefits and limitations of using defect bash are given. Finally, the process of doing defect bash is outlined. This review provides initial understanding on how defect bash could be useful in achieving the software quality and lays foundation for further academic studies of this topic.

1 INTRODUCTION

Testing is one of most important software quality practices and it aims in finding defects before the software is released. Defect bash (or bug bash) is a testing event where a group of people tries to find as many defects as possible from the software. It is widely applied in software companies (Anonymous 2, 2010; Anonymous 3, 2011; Enns, 2004; Fitzgerald, 2012; Powell, 2009; Sagynov, 2011a, b; Whittaker, 2012). For example Microsoft used it quite often (Anonymous 2, 2010; Crowhurst, 2011; Liangshi, 2010; Sahay, 2012). Marick (1997) sees defect bash as an additional testing complementing written automated and manual test cases. Kaner (2011) listed defect bash as a technique for black-box testing. It is used in crowd testing (Anonymous 6, 2012) and open source testing (Grubbs, 2012).

Although, defect bash has been around for decades (Dolan and Matthews, 1993), we could not find a single research article that would have had primary focus on the defect bash, although it is often mentioned as a side note. This is in stark contrast for example with code review practice that has been the primary focus of several academic articles.

In order to shed light to this popular practice, this article analyses the existing information on defect bash with the following fundamental issues:

- The definition of defect bash;
- The benefits and limitations of using defect bash;
- The process of doing successful defect bash.

This paper is a literature review of defect bash. In Section 2 we define the literature review protocol. Section 3 defines defect bash based on the literature review. Section 4 lists the benefits and limitations of defect bash. The process of defect bash is derived from the references in the section 5. Finally conclusion is given in Section 6.

2 SYSTEMATIC LITERATURE COLLECTION

Systematic literature review (Kitchenham and Charters, 2007) is used to identify the definition, benefits and limitations, and process of defect bash.

Combination of string *(("bug bash" OR "bug bashing" OR "defect bash" OR "defect bashing") AND "software")* is used for title, abstract and keywords to identify articles related to defect bash.

Using the search string we searched both the academic and the grey literature. The search engines used were Google (www.google.com) for grey literature and Google Scholar (scholar.google.com) for academic literature. The latter includes the major academic databases such as IEEE Explore, Scopus and ACM.

We found 19,500 items from Google (September 8 2012), but only the first 200 results were checked. This is because in the results from 130 to 200 only

one relevant page was found. Thus, going deeper in the Google search results would have increased the workload with very small likelihood of finding relevant results. Out of the 200 results 36 results were related to research needs and only 27 results from Google are used as references in this article because some results are duplicated. Classification of grey literature results according to the definition on the types of grey literature by GreyNet International (2012) show the following: we found four types of grey literature (27): 3 discussion forums, 12 blogs, and 12 company websites.

66 items were found in Google Scholar (10 April 2013). Among 66 items from Google Scholar, 30 items were unrelated or unavailable (broken links, or books we did not have access); In 20 items defect bash only as a term was mentioned, without information on definition, benefits and limitations of defect bash. This left us with 17 items (academic articles and books) containing information on definition, benefits and limitations. Totally 44 references are used for this article as in Table 1.

The classification is done based on the major content of each reference; this means that one reference may contain different kinds of information, i.e., definition, benefits and limitation, and process of doing bug bash. One reference may be used in several places of this article.

3 DEFINITION

Several definitions for defect bash were found. Next we present three definitions that appeared as the most popular, i.e. these definitions were used in several places.

In Desikan and Ramesh's book (Desikan and Ramesh, 2008), the defect bash is defined as "an ad hoc testing done by people performing different roles in the same time duration during the integration testing phase, to bring out all types of defects that may have been left out by planned testing. It is not based on any written testing case*"*.

The definition in Wikipedia (2012) is "a bug bash is a procedure where all the developers, testers, program managers, usability researchers, designers, documentation folks, and even sometimes marketing people, put aside their regular day-to-day duties and pound on the product to get as many eyes on the product as possible".

ALLInterviews (2012) and QTP (2012) have the same definition of "it is an ad hoc testing where people performing different roles in an organization test the product together at the same time. The testing by all the participants during defect bashing is not based on written test cases. What is to be tested is left to an individual's decision and creativity. This is usually done when the software is close to being ready to release*"*.

Combing the above 3 definitions, we can define the defect bash as follows: *It is a temporally and spatially co-located group testing session, done by people from different roles during the integration testing phase or close to software release to bring out all types of defects that may have been left out by planned testing. It is not based on written test-cases.*

4 BENEFITS AND LIMITATIONS

4.1 Benefits

Table 2 presents the benefits of the defect bash. We can see the most frequently mentioned benefit from

Table 1: Classification of found literatures.

Category	n: references
Definition and benefits of defect bash	15: ALLInterviews, 2012; Aranda and Venolia, 2009; Birkinshaw and Goddard, 2009; ChetanaS, 2011; Desikan and Ramesh, 2008; Dolan and Matthew, 1993; QTP, 2012; Marick, 1997; Nindel-Edwards and Steinke, 2006; Slaughter and Rahman, 2011; Wikipedia, 2012; Williams, 1998; Whittaker, 2012; Wong, 2011; Yüksel, Tüzün, Gelirli, and Bıyıklı, 2009
How to do defect bash	13: Anonymous 1, 2010; Anonymous 4, 2011; Bach,1998; Berkun, 2008; Cruden, 2011; Liangshi, 2010; Haynes, 2009; Kalra, 2007; Khan and ElMadi, 2011; Mey, 2012; Powell, 2009; Pruitt and Adlin, 2005; Spagnuol, 2007
Against defect bash	1: Lyndsay, 2011
Report after defect bash	5: Anonymous 2, 2010; Anonymous 3, 2011; Sagynov, 2011a; Sagynov, 2011b; MarkusN, 2012
Advertisement for doing defect bash	10: Anonymous 5, 2012; Anonymous 6, 2012; Crowhurst, 2011; Enns, 2004; Fitzgerald, 2012; Grubbs, 2012; Kaner, 2011; Sahay, 2006; Sakai, 2012; Sande, 2009
Total	44

defect bashing is finding defects in a short time before the software is released. It can also bring out both functional and non-functional defects. The life cycle of the bugs can be minimized as the reports can be verified and assigned during the defects bash.

Defect bash acts as a basic sort of usability as well as acceptance testing. People can pound the system from the load in the defect bash. Additionally, defect bash can be used to break the system instead of trying to conclude the system works.

Defect bash brings different people from different roles together in the organization for testing. The boundaries between roles are minimized in a co-located session. Different roles also help validating the software from end user perspective. The end users using a software product will be quite different from each other in many aspects such as understanding about the product, the manner of using the software. Defect bash can bring in people who have different levels of product understanding to test the product together randomly, which can simulate the different approaches of the end users. It is also recognized that fresh eyes have less bias and that fresh eyes can uncover new defects.

Learning and competitions are also mentioned as benefits of a defect bash. The built-in competitive instinct of participants should be stimulated to achieve this. Defect bash also helps in learning the product and learning from each other. It can be used as unofficial demo. The learning and competition aspects are also claimed to help in team building inside a company.

4.2 Limitations

Though many benefits are declared in the literature, still there are limitations of defect bash as below by them:

Limitation 1: Defect bash might cause too many duplicate defect reports. The quality of defect reports can be low. Time is wasted in investigating, diagnosing and logging the same problem several times (Anonymous 4, 2011; Berkun, 2008; Lyndsay, 2011).

Limitation 2: The blog (Lyndsay, 2011) claims that in defect bash there isn't much opportunity to learn from each other. This is because many people use the system for the first time, at the same time. Also the limited time period disables learning. We think that the defect bash in the first time would be similar to what Lyndsay observed. However after more experience both organizer and participants will learn how to do a defect bash more efficiently.

Limitation 3: Defect bash can only predict customer behavior for the first few hours (Lyndsay, 2011). Thus, it cannot offer information of long-term product use. We think that usage by different users even once or short period is better than nothing, and we maybe should not expect too many feedbacks on customer behaviors from defect bash as Lindsay's

Table 2: Benefits and the references mentioning it.

Benefit	**References mentioning the benefit**
Finding many defects, bring out both functional and non-functional defects, also shortening the life cycle of the bugs	Anonymous 1, 2010; Anonymous 2, 2010; Anonymous 3, 2011; Aranda and Venolia, 2009; Crudden and Lawson, 2011; Desikan and Ramesh, 2008; Haynes, 2009; Karla, 2007; Liangshi, 2010; MarkusN, 2012; Powell, 2009; QTP, 2012; Sagynov, 2011b; Sahay, 2006; Sande, 2009; Spagnuolo, 2007
The competitive instinct of participants are stimulated and good for team building	Haynes, 2009; Birkinshaw and Goddard, 2009; Wong, 2011; Yüksel, Tüzün, Gelirli, and Bıyıklı, 2009
Saving money (no need to hire group externals)	Crudden and Lawson, 2011
Help in rapid evolution of test scripts	Bach, 1998
Acting as acceptance testing and usability testing	Anonymous 1, 2010; Anonymous 4, 2011; Karla, 2007
Make software more valuable while enhancement done	Anonymous 1, 2010; Crudden and Lawson, 2011
Cross boundary testing	Desikan and Ramesh, 2008
Learn your product and team building	Berkun, 2008; Crudden and Lawson, 2011; Desikan and Ramesh, 2008; Haynes, 2009; Spagnuolo, 2007; Khan and ElMadi, 2011
Fresh eyes have less bias	Anonymous 1, 2010; Crudden and Lawson, 2011; Desikan and Ramesh, 2008
Users in different levels	Desikan and Ramesh, 2008
Not wait for documentation	Desikan and Ramesh, 2008
Testing is also to break system	Desikan and Ramesh, 2008

observation is reasonable.

Limitation 4: Defect bash causes the strain of available resources for setting test environment with a big group of people (Lyndsay, 2011). However, this limitation can be overcome by careful planning of defect bash.

5 PROCESS

Defect bash process is categorized into three phases in this article: preparation, defect bash session and post-process data as shown in Figure 1. There are two kinds of roles in defect bash process, organizer(s) and participators. Organizers plan the defect bash, moderate it and analyze the report from participants. Participants just test the software and report findings.

Phase 1 – Preparation: Panic can be caused (Berkun, 2008) if there is no preparation before defect bash. Usually the defect bash events are advertised in a certain period earlier (Anonymous 6, 2012). Fitzgerald (2012) and Sakai (2012) have a good list of items for the bug bash: goals**,** when to have defect bash, duration, testing target software, testing environment, defect reporting system, participants, instructions. Sagynov (2011a) also declares the evaluation method and rewards in the preparation phase. Additionally, management support needs assured during the proration phase. Table 3 collects all items which might be needed for the preparation. The preparation actions can be as the items in Table 4.

Phase 2 - Defect bash session Analyzing the references and the items for the doing defect bash are collected in Table 5, and the actions in the session can be as the items in Table 6.

Phase 3 - Post-process the data from defect bash Analyzing the references and the items for post-processing the data are collected in Table 7, and the actions in the session are in Table 8.

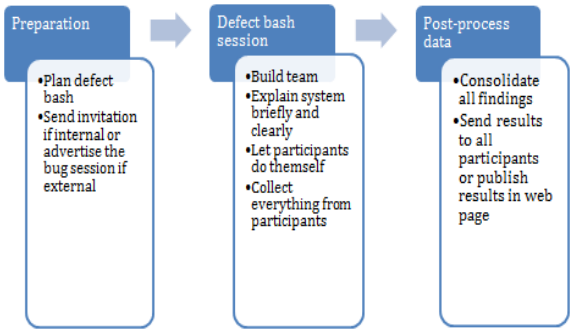

Figure 1: The process of doing defect bash.

6 CONCLUSIONS

Based on a systematic review of literature of defect bash, we make the following conclusions. First, defect bash is defined as a spatially and temporally co-located testing session performed by a group of people. As academic studies of defect bash are lacking, we also decided to search for a grey literature in addition to academic literature.

Second, we found several claimed benefits of defect bash. Among them are finding many defects in a short time period, learning the product, team building, getting many roles to test the software from different viewpoints and getting fresh eyes to search for defects (see Section 4.1). However, Lyndsay (2011) and handful of other authors disagreed with the benefits of defect bash, see Section 4.2. The most serious limitation, however, was the number of duplicated defect reports. Nevertheless, this disagreement calls for empirical investigation on the benefits and limitations of defect bash.

Third, we presented a process for defect bash in Section 5. There are two major roles in defect bash, organizer(s) and participants that can represent various roles from the software development organization. A defect bash is divided into three phases: preparation, defect bash session and post-process data. The actions for executing each defect bash phase in Tables 4, 6 and 8 can be used as the guidelines for doing defect bash.

We think that this work lays out an initial foundation for the future studies of defect bash that are needed to understand this industrially relevant software testing approach. Some examples of future studies are: Improved guidelines on defect bash process, collecting data on the detected defects, factors affecting the efficiency and effectiveness, the spread of knowledge in defect bash and the group sizes for defect bash. Additionally, this literature review should be extended to cover 'team exploratory testing' practice (Bach 2003; George, 2013; Saukkoriipi and Tervonen, 2012) that appears to have many similarities with defect bash practice. Furthermore, similarity between defect bash and software review meetings exist as both defect bash and software reviews are group based quality assurance techniques. The main difference between defect bash and software review is that defect bash consists of individuals *testing* the software while in software review individuals *review* software artefacts' such as requirements, design or code. As software reviews are widely studied group based QA method (Wiegers 2002) comparison of practices of defect bash and software reviews should be made.

Table 3: Items on the phase 1 – preparation.

Items of preparation	**References for each item**
Goals	Flush out the bugs (Crowhurst, 2011; Fitzgerald ,2012; Sakai, 2012)
Time	Informed 1 month earlier (Anonymous 6, 2012), A given time (Fitzgerald, 2012; Sakai, 2012), clear afternoon (Berkun, 2008; Crowhurst, 2011); Schedule in key milestone (Mey, 2012)
Get support from management	Big shot (Berkun, 2008), Haynes, 2009; Pruitt and Adlin, 2005
Where	Have a bug bash headquarter (Berkun, 2008); Haynes, 2009
Duration	3-5 hours (Anonymous 1, 2010), 60 minutes (Fitzgerald, 2012; Sakai, 2012), (Cruden and Lawson, 2011), 30 minutes (Cruden and Lawson, 2011)
Target software or focus	Focus (Anonymous 4, 2011; Fitzgerald, 2012; Sakai, 2012), Freeze the build and a Focus (Berkun, 2008; Crowhurst, 2011)
Testing environment	Server (Fitzgerald, 2012; Sakai, 2012), in the same environment (Cruden and Lawson, 2011)
Participants	Registered users (Sakai, 2012; Fitzgerald, 2012), everyone in the company (Cruden and Lawson, 2011); market person, developers, technical writers (Haynes, 2009)
Defects reporting system	Jiras (Fitzgerald, 2012; Sakai, 2012)
Evaluation method and reward	Give out $50 Amazon gift card (Sagynov, 2011a), criteria to judge the bug (Anonymous 2, 2010)
Instructions	How to connect the testing server, do testing and report findings (Sakai, 2012; Fitzgerald, 2012)
Inform earlier	Participants informed earlier (Cruden and Lawson, 2011)

Table 4: Actions in phase 1 – preparation.

Performer	**Actions**
Organizer	Planning defect bash: • Goals – detect all defects as much as possible; • Time – when to have it? • Support from management; • Where to have defect bash? • Duration – how long a session lasts? Duration can be varied depending on project, 30 minutes, 3-5 hours, and whole afternoon; • Target software – which software build should be used? Optimizing the effort involved in defect bash. For example, it can be classified into: Feature/component defect bash; Integration defect bash; Product defect bash (Desikan and Ramesh 2008). • Testing environment – in what testing environment defect bash can be done? Setting up testing environment; • Participants – who will participate in the defect bash? • Reporting system – how the findings should be reported? • Instructions on how to do it; • Evaluation method, who will win and what is the rewards? Sending invitation to all usually at least one week ago and invite a big manager.
Participators	Not involved yet.

Table 5: Items on the phase 2 - defect bash session.

Items of doing defect bash	**References for each item**
Build team or create a rival teams	Anonymous 1, 2010; Anonymous 4, 2011; Berkun, 2008
Explain what to do briefly	Anonymous 1, 2010; Anonymous 4, 2011
Show a list of known issues and the format of good report	Anonymous 4, 2011; Berkun, 2008
Let participants perform on their own	Anonymous 1, 2010; Haynes, 2009
Keep scores	Anonymous 1, 2010; Berkun, 2008
Encourage reporting	Anonymous 4, 2011
Doing ad hoc testing	Anonymous 4, 2011
Snacks offered	Anonymous 4, 2011, Haynes, 2009; Mey, 2012
Taking pictures for fun	Haynes, 2009;

Table 6: Actions in phase 2 - defect bash session.

Performer	Actions
Organizer	Create rival teams; compose a team having a good mix of experienced, new and untrained people to participate in this exercise. Or each participant does testing him/herself independently. Explain system briefly, show known issues and show a good error report to participants at beginning. Provide focus of testing areas; let them use the software without interruptions. Do not let them discuss each other during the testing. Allow participants report as they like. Anything like issues faced, crashes encountered, questions, comments, and general feedbacks and any suggestions or enhancements. Keep scores who found what. Have snacks and drinks offered and pictures may be taken for fun.
Participators	Doing ad hoc testing for detecting the defects.

Table 7: Items on the phase 3 - post-process data.

Items in post-process	References for each item
Analyse all the issues, filter out the known issues	Anonymous 1, 2010; Anonymous 3, 2011; Haynes, 2009
Criteria to judge the winner	Anonymous 2, 2010
Winners	Anonymous 2, 2010; Anonymous 3, 2011
Appreciation	Anonymous 2, 2010; Anonymous 3, 2011; Mey, 2012

Table 8: Actions in the phase 3 - post-process data.

Performer	Actions
Organizer	Analyse all the issues and suggestions and summarize them for the team. • Issues should be compared with the error tracker to check for duplicates; • All duplicated issues grouped into one issue; • Filter out the known issues. The left ones will probably be new encountered defects, suggestions and enhancement that will be used to improve the software further. Send appreciation email or broadcasting information to all participants to tell the results, how many new defects, who found the most important defects and reward the person or which team (if rival teams) who found the most important defects.
Participators	Receive the acknowledgement from organizer and rewards.

REFERENCES

ALLInterviews, 2012. What is meant by defect bash? In link: http://www.allinterview.com/showanswers/23853.html.

Anonymous 1, 2010. How to do a test bug bash for your software project? In link: http://bettersoftwaretesting.blogspot.com/2010/10/how-to-do-test-bug-bash-for-your.html.

Anonymous 2, 2010. The WDK community bug bash contest 2010 is over. In link: http://www.osronline.com/page.cfm?name=bugbash.

Anonymous 3, 2011. Bug bash aftermath. In link: http://seleniumhq.wordpress.com/2011/01/31/bug-bash-aftermath/

Anonymous 4, 2011. Software bug bash tips. In link: http://programmers.stackexchange.com/questions/47007/software-bug-bash-tips.

Anonymous 5, 2012. Bug bash on Uhuru software. In link: https://www.odesk.com/o/jobs/job/Bug-Bash-on-Uhuru-Software_~~7390d00ba7cef979/.

Anonymous 6, 2012. 99tests Bug bash. In link: http://99tests.com/99tests-bug-bash/.

Aranda, J., Venolia, G., 2009. The secret life of bugs: going past the errors and omissions in software repositories. In *IEEE 2009*. Article ID: 978-1-4244-3452-7/09.

Bach, J.A., 1998. Microdynamics of process evolution. *In IEEE Computer Society, February 1998.*

Bach, J.A., 2003. Exploratory testing explained. In link: http://www.satisfice.com/articles/et-article.pdf.

Berkun, S., 2008. How to run a bug bash? In link: http://www.scottberkun.com/blog/2008/how-to-run-a-bug-bash/.

Birkinshaw, J., Goddard, J., 2009. The management spectrum. In *Journal Compilation, London Business School.*

Chetanas, 2011. What is defect bash? In link: http://www.testingken.com/forum/showthread.php?t=346.

Crowhurst, C., 2011. Bug bash 2011 – A direct response. In link: http://www.marketingarchitects.com/2011/05/bug-bash-2011-a-direct-response/.

Cruden, K., Lawson, N., 2011. The Power of the bug bash. In link: http://www.agilequalityassurance.com/2010/

04/the-power-of-the-bug-bash/.
Desikan, S., Ramesh, G., 2008. *Software testing, principle and practices.* Dorling Kindersley (India) Pvt. Ltd.123-126. ISBN 978-81-7758-121-8.
Dolan, R.J., Matthews, J.M., 1993. Maximizing the utility of customer product testing: beta test design and management. *J PROD INNOV MANAG* 1993;10:318-330. Elsevier Science Publishing.
Enns, N., 2004. It is bug bash day on 28 April 2004. In link: http://blogs.msdn.com/b/windowsmobile/archive/2004/04/28/122435.aspx.
Fitzgerald, K., 2012. Sakai bug bashes. In link: https://confluence.sakaiproject.org/display/3AK/Bug+Bashes.
George, C., 2013. Team exploratory testing: our first experiences... In link: http://mostly-testing.blogspot.fi/2013/01/team-exploratory-testing-our-first.html.
GreyNet International, 2012. http://www.greynet.org/greysourceindex/documenttypes.html.
Grubbs, J.C., 2012. Chicago open source bug bash. In link: http://www.meetup.com/chicago-open-source-bug-bash/.
Haynes, D., 2009. The search for software robustness. In Excerpt from Pacific NW Software Quality Conference.
Kalra, P., 2007. Bug bash. In link: http://rivr.sulekha.com/bugbash_265217_blog.
Kaner, C., Fiedler, R.L., 2011. *Black box software testing. Introduction to test design. A survey of test techniques. In BBST Test Design.* In http://www.testingeducation.org/BBST/testdesign/BBSTTestDesign2011pfinal.pdf.
Kitchenham, B., Charters, S., 2007. Guidelines for performing systematic literature reviews in software engineering. *Software Engineering Group, School of Computer Science and Mathematics, Keele University,* Tech. Rep. EBSE-2007-01, July 2007.
Khan, M.S.A., ElMadi, A., 2011. Data warehouse testing – an exploratory study. Software engineering master thesis no: MSE-2011-65. In *School of Computing Blekinge Institute of Technology SE-371 79 Karlskrona, Sweden.*
Liangshi, 2010. Ce shi za gan, bug bash. In link: http://www.51testinzg.com/html/63/n-225363.html.
Lyndsay, J., 2011. Known ways of managing ET #2 – bug bash. In link: http://workroomprds.blogspot.fi/2011/12/known-ways-of-managing-et-02-bug-bash.html.
Marick, B., 1997. Class Testing mistakes. In http://www.csi-chennai.org/swtws/ws-swt/mistakes.pdf.
MarkusN, 2012. Big bug bashing for GRASS 6! In link: http://gfoss.blogspot.fi/2012/08/big-bug-bashing-for-grass-6.html.
Mey, C.V., 2012. *Shipping greatness*. *Practical lessons on building and launching outstanding software, learned on the job at Google and Amazon*. By O'Reilly Media.
Nindel-Edwards, J., Steinke, G., 2006. A full life cycle defect process model that supports defect tracking, software product cycles and test iterations. In *Communications of the IIMA 2006 Volume 6 Issue 1.*
Powell, C., 2009. ABAKAS bug bash. In link: http://blog.abakas.com/2009/01/bug-bash.html.
Pruitt, J., Adlin, T., 2005. *The personal lifecycle: Keeping people in mind throughout product design*.
QTP Tutorials and Interview Questions, 2012. In link: http://qtp.blogspot.fi/2010/03/bug-bash-defect-bash.html.
Sagynov, E., 2011a. CUBRID bug bash event! In link: http://www.cubrid.org/blog/news/cubrid-bug-bash-event/.
Sagynov, E., 2011b. CUBRID bug bash event results. In link: http://www.cubrid.org/blog/cubrid-life/cubrid-bug-bash-event-results/
Saukkoriipi, S., Tervonen I., 2012. Team exploratory testing sessions. In *International Scholarly Research Network, ISRN Software Engineering, Volume 2012, Article ID 324838*.
Sahay, A., 2006. Microsoft, Appin launch security bug bash 2006. In link http://press.xtvworld.com/article12437.html.
Sakai, 2012. QAE QA. Call for bug bash. In link: https://oae-community.sakaiproject.org/~oae-qa#l=Bug-Bashes/Bug-Bashes.
Sande, S., 2009. Bug-bashing Bento 2.0v5 is now available for download. In link: http://www.tuaw.com/2009/08/19/bug-bashing-bento-2-0v5-is-now-available-for-download/.
Slaughter, J., Rahman, M., 2011. Information security plan for flight simulator applications. *In International Journal of Computer Science & Technology (IJCSIT), Vol 3, No 3, June 2011*. DOI: 10.5121/ijcsit.2011.3301.
Spagnuolo, C., 2007. The bug bash sprint. In link: http://edgehopper.com/the-bug-bash-sprint/
Wang, L., 2011. Master project. Best practice for testing in development and testing groups. In [http://etd.dtu.dk/thesis/277238/]
Whittaker, J.A., 2012. The 10-minute test plan. In *IEEE Software by IEEE Computer Society*. Article ID: 0740-7459/12.
Wiegers, Karl Eugene. Peer reviews in software: A practical guide. Boston: Addison-Wesley, 2002.
Wikipedia, 2012. Bug bash. In link: http://en.wikipedia.org/wiki/Bug_bash#cite_note-0.
Williams, G., 1998. Usability process challenges in a web product cycle. *In HCT'98 Conference Companion.*
Yüksel, H.M., Tüzün, E., Gelirli, E.., Bıyıklı, E., 2009. Using continuous integration and automated test techniques for a robust C4ISR system. *In IEEE 2009*. Article D: 978-1-4244-5023-7.