TECHNICAL TRANSACTIONS | **CZASOPISMO TECHNICZNE**
FUNDAMENTAL SCIENCES | NAUKI PODSTAWOWE
1-NP/2013

PRZEMYSŁAW LEGUŁA*, PIOTR ZABAWA*

BUG FIXING TOOL (BFT) FOR CONTINUOUS INTEGRATION

NARZĘDZIE BUG FIXING TOOL (BFT) DO CIĄGŁEJ INTEGRACJI

A b s t r a c t

The paper is focused on the bug fixing handling business process rather then just on fixing a bug. The tool presented here is dedicated to supporting the business process of bug fixing and not to bug fixing itself. It is addressed especially to small teams having a common testing team.

Keywords: *continuous integration, version control server, bug fixing, bug submission, iterative software development process*

S t r e s z c z e n i e

Artykuł skoncentrowany jest na procesie biznesowym obsługi błędów oprogramowania bardziej niż tylko na kwestiach obsługi błędów. W konsekwencji również narzędzie zaprezentowane w niniejszej pracy służy wspieraniu procesu biznesowego poprawiania błędów, a nie samemu poprawianiu błędów. Jest ono adresowane szczególnie do małych zespołów posiadających wspólny zespół testerów.

Słowa kluczowe: *ciągła integracja, serwer wersjonowania,, poprawianie błędów, zgłaszanie błędów, iteracyjny proces rozwoju oprogramowania*

* Msc. Eng. Przemysław Leguła, Sabre Polska Sp. z.o.o.
** Ph.D. Eng. Piotr Zabawa, IBM/Rational Certified Consultant, Institute of Computer Science, Faculty of Physics, Mathematics and Computer Science, Cracow Technical University.

1. Introduction

The authors of the paper have different and complementary experience from the market. One of them is working in a large international software development company and is responsible for configuration management. The other has long lasting experience from the consulting branch of the same business and thus has some observations from many companies different in size – from very small ones to corporations. The authors agree that the process of bug fixing in many companies is handled mainly on the business process level (by business workers – the staff) with very limited tool support. This is why the process is far from automated. As a matter of fact, the possible process automation is blocked by the process inconsistency, by multiplicity of process variations and by the mess of incompatible tools used to support this process. The possible scale of optimization of the process will result in a relatively large number of attractive proposals that are intended to be addressed in succeeding publications.

A simple but useful tool, the Bug Fixing Tool (BFT), supporting existing bug fixing business processes is presented in the paper. This tool communicates with Subversion (SVN) version control server and is intended to be used in a continuous integration environment. At the same time it constitutes the first stage of the implementation of the defect management process improvement concept presented as a whole in paper [8].

2. Continuous integration

The concept of continuous integration [1–6] was introduced to support the business need to (almost) always have an up-to-date version of a newly released software product. This need appeared as a result of risk minimization achieved by so-called early risk mitigation. More generally speaking, the early risk mitigation is supported by iterative software development processes very well in many disciplines of the process. For example, Rational Unified Process (RUP) expects to have a new release at least at the end of each iteration (excluding Inception phase). But it does not limit the expectation to have releases more frequently – during iterations – in any way. It seams that XP approaches that are promoted by Agile processes stress the necessity of having such frequent (continuous) releases or at least builds more clearly.

There is a slight umbiguity in understanding the subject of continuous integration. In one case, the product should be understood as the installation program (setup) of the product. In the second case, the subject of release is meant as the build (executable of the product) only. The first approach is more general and maps better to the notion of product. However it is worth noticing here that the product is verified by tests, and the tests are different. In order to perform unit tests, the build is sufficient as tests of this kind are not executed on the installed product but on the build of code in a development environment. In the case of black-box testing, the build is not sufficient and the installation should be the subject of test execution.

Nevertheless, the paper is focused on builds only as the direct results of the integration process. It is also assumed that the bug was already identified and located in the source code correctly at the beginning. So the bugs that are not addressed to code (say wrong version of a database file in the product installation) are not the subject of this paper.

3. Bug fixing problem characteristics

This section shows the consequences of finding a bug in a particular project associated to a product for an existing business processes.

A sample configuration tree view for a product's file or directory which is offered by most version control servers is shown in Fig. 1. There are the configuration branches A, B and C. The question of what they represent may appear here. And there are at least two possible answers:

– branches represent different product versions possibly elaborated by different teams,
– branches represent different development or integration branches in a particular project.

So, what should be done when a bug is identified? Let us assume that the bug was identified in branch C during any kind of tests performed by developers (typically white--box testing) or by testers (typically black-box testing). In such a case the special bug-fixing branch should be created from branch C in order to fix the bug just in this newly created branch. The fix branch should not be a development branch as the history of fixing bugs is different to the history of development. Mixing them is a bad practice. That is why Unified Change Management [7] promotes creating special fix branches for bug fixing purposes, nevertheless-manually. The file where the bug was identified is present also in branches A and B. So other teams (or other team members) should be reported to somehow about the bug. They should be able to to pass the report about fixing the bug as well. And this is the place for the subjected tool. Both the characteristic (description) of the bug and fix should be placed somewhere. The best place for it is just the fix branch in the version control server.

Fig. 1. Existing configuration tree view for a file or a directory in version control server

It is worth noting who identifies what in the process of bug identification and fixing. The typical but bad practice is that the team who identifies the bug is responsible for the correct propagation of the bug information to other teams or team members.

This approach is acceptable in a small number of small teams especially when they are supported by a common testing team. In such a situation the team members are able to identify other members of the team or to identify other teams and check if the teams still exist. The question who and how should manage the information of a bug addressed

to already not existing team may appear. The common testing team is promoted here because it has enough knowledge to be able to check if the bug maps to other branches and determine which ones. The last assupmption about the common testing team is crucial as the team must be able to know, understand and have access to tests and code. The knowledge and privilidges mentioned above are necessery to make possible the verification, first if the bug exists, and second if the fix of the bug is correct in all branches involved. It must also be underlined here that the product versions in different branches represent different product functionalities. Applying the same approach to large products, large or many teams and many testing teams is not good. So, another approach suitable for this more demanding situation will be proposed in succeeding papers.

The tool presented in the next sections fits best to the simpler software development process as described in the paragraph above.

4. Problem solution

The problem of a desirable reaction to the bug finding was described in the previous section. This section is dedicated to the description of the role of the BFT tool. How the tool is related to the existing process is depicted in Fig. 2.

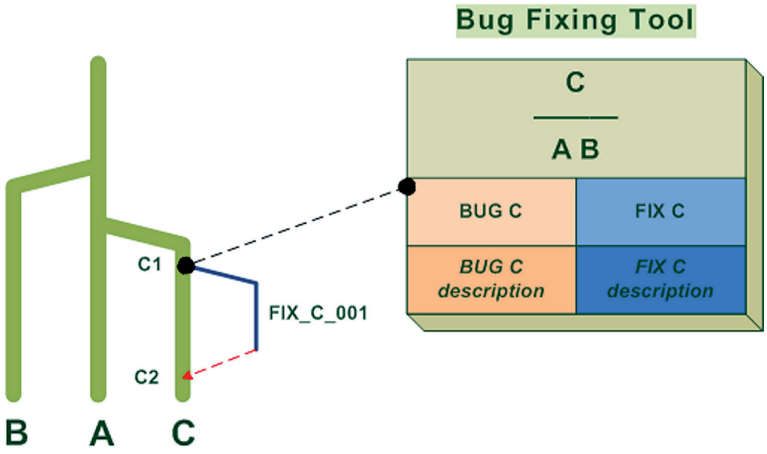

Fig. 2. Location of the Bug Fixing Tool (BFT)

In order to solve the problem with the tool support the following assumptions were made. The tool works for one product but many instances of it may be executed, one for each product. There is one repository per product. The version control server is Subversion (SVN). There are no assumptions regarding SVN clients used by the teams.

The functionality of the product is as follows. When a bug is found by a developer or tester the bug is reported in the tool. The tool is able to store many bug descriptions simultaneously. When the source code file containing the bug is identified the developer responsible for fixing it opens a bug in the tool. At this moment, the automatically tool creates the fix branch based

on branch C directly in SVN. It stores the bug description in SVN as associated to the just created branch. The developer fixes the bug and reports this in the tool by associating an appropriate description to this fix. At this moment, the tool automatically merges the fix to the branch C if possible. If it is not possible, the developer does it himself manually. The manual merge may be necessary if there were merges to branch C in the meantime (that is between C1 and C2). So, the good practice is to do fixes fast. More specifically, in order to maximize the chance for this requirement, the fixing process should be started as late as possible (just before starting fixing the bug). This practice could be called lazy bug fixing per analogy to the notion of lazy initialization well known from programming.

This way the bug fixing was performed on branch C only. But how about bug propagation to other branches? This task is performed by the tool semi-automatically. When the developer or tester identifies the bug, it manually checks on the other branches if the bug has (may have) impact on these branches. If he identifies the possible impact he specifies the possibly impacted branches in the tool (A and B below C in the top-most compartment in Fig. 2). The good practice here is not to assume the lack of impact in unclear situations. As a consequence, the default is to assume the impact to all other branches. And this is the weak point of the process (not the tool – it supports the existing process from the assumption). In the case of the common testers team, the existence of the bug may be verified by running appropriate tests (if they exist). Otherwise, the identification of the impact is a matter of guess work. And this is the reason of assumptions of having common testing team as well as the source for the good practice of assuming impact even if this impact does not exist.

The main role of the BFT tool could be characterized as a tool storing all the information about bugs and fixes identified in the product life, performing simple configuration management tasks like branching and merging as well as storing descriptions in the product configuration repository. This way the tool both simplifies the technical process and supports the existing business process of bug fixing by offering a good communication platform.

The tool described here was already implemented and is used in the environment containing:

- Subversion (ver. 1.6.2) for configuration management,
- Ant (ver. 1.7.1) for build process execution,
- Hudson (ver. 1.306) for build automation,
- Tomcat (ver. 6.0.18) for running the tool.

The BFT tool was implemented in the following technologies:

- Java EE 1.5,
- Hibernate (ver. 3.2.1),
- svnkit (ver. 1.3.0).

5. Tool advantages

The advantages of the tool are of different kinds as is shown below.

The main advantages are that the tool is very cheap due to the fact that it is based on open source tools and it is easy to implement. As a consequence, the source code of the whole

software required by the tool is available which creates the opportunity for more advanced improvements of the BFT tool in the future. The above mentioned advantages make starting the tool usage in a company easy and are not connected to any significant investment at the beginning.

Another group of advantages is strictly connected to the running business of software development. This group of advantages consists of:

– uniform way of bug fixing in the whole company,
– usage of one simple web tool for the bug fixing process which is easily accessible by different teams,
– ease of implementation of the tool,
– small size of the code which limits the likelihood of defects,
– limited but adequate functionality which makes testing the tool easy,
– improvement of company communication regarding bug fixing,
– improvement of statistics that may be performed on version control server via distinguishing between development and bug fixing.

6. Conclusions

The bug fixing tool dedicated to the continuous integration approach to software development process described in the paper is very useful due to the advantages presented in the previous section. However, this tool supports existing business processes that have disadvantages mentioned in section 3. Consequently, the approach described here and the tool itself are a good starting point to the further optimization of both the process and the tool concept. This problem of optimization is a subject of investigations that are taking place at the moment. The results of that different problem defined for the purpose of wide and deep optimization and automation are intended to be published soon.

The situation presented in this paper is quite simple, nevertheless, realistic in many cases. A proposal of a solution to much more complex situations not limited to one configuration repository and consisting of many different additional actions that may be performed on the configuration repository is the subject of current investigations and also will be published soon.

References

[1] Duvall P., *Automation for the people: Continuous Integration anti-patterns*, IBM 2008.
[2] Elssamadisy A., *Patterns of Agile Practice Adoption*, InfoQ 2007.
[3] Fowler M., Continuous Integration (http://www.martinfowler.com/articles/continuousIntegration.html).
[4] Lee K.A., Realizing continuous integration, IBM 2005 (http://www.ibm.com/developerworks/rational/library/sep05/lee),
[5] Lotz C., Continuous Integration: From Theory to Practice, 2007 (http://dotnet.org.za/cjlotz/archive/2007/12/03/continuous-integration-from-theory-to-practice.aspx).

[6] Miller J.D., *Using Continuous Integration? Better do the "Check In Dance"*, 2005 (http://codebetter.com/blogs/jeremy.miller/archive/2005/07/25/129797.aspx).
[7] Wahli U., et. al., *Software Configuration Management. A Clear Case for IBM Rational ClearCase and ClearQuest UCM*, IBM redbooks 2004.
[8] Zabawa P., *Defect Management Process Improvement Proposal, Proceedings of the IADIS International Conferences Informatics 2010*, Wireless Applications and Computing 2010, Telecommunications, Networks and Systems 2010, part of the IADIS Multi Conference on Computer Science and Information Systems 2010 Freiburg, Germany JULY 26-28, 2010, 45-149.