Automating expert-defined tests: a suitable approach for the Medical Device Industry?

David Connolly[1], Fergal Mc Caffery[2], and Frank Keenan[1]

[1] Software Technology Research Centre,
[2] Regulated Software Research Group,
Dundalk Institute of Technology, Dublin Road, Dundalk, Ireland
{david.connolly,fergal.mccaffery,frank.keenan}@dkit.ie

Abstract. Testing is frequently reported as a crucial stage in the software development process. With traditional approaches acceptance testing is the last stage of the process before release to customer. Acceptance Test Driven Development (ATDD) promotes the role of an expert customer in defining tests and uses tool support to automate and execute these tests. Here the challenge is to support such an expert in the reuse of existing documentation. This paper details an experiment in a generic domain while outlining plans for development of an automated testing model that could assist medical device companies to adhere to regulatory guidelines by providing them with a fully traceable testing artifacts.

1 Introduction

A large part of software development expenditure is attributed to *testing*. Traditionally, with plan-driven development, acceptance testing, the process of testing functional requirements with "data supplied by the customer" [1] occurs as the final stage of the development process long after the initial investigation has completed [2]. Many reports, however, highlight that costs can be reduced by detecting errors earlier in development [3]. Also supporting this, in many domains, such as the medical device industry, software is developed subject to a regulatory environment with a tendency for extensive documentation. This regulatory environment features guidelines and standards such as [4] - [9]. Despite many constraints already being specified, this is often ignored with tests written from scratch after implementation is complete. In contrast, agile approaches require constant customer collaboration throughout development, with customer provision of acceptance tests being an important part of this role. Often, it is recommended that tests be identified before implementation commences. In eXtreme Programming (XP) [10], for example, acceptance tests are defined as a part of the User Stories practice and, as such, are written before coding of the story begins. In this context, functional tests are synonymous with acceptance tests [11]. Further, for accurate user stories, Cohn recommends customers themselves specify acceptance tests with developers and testers providing support as required [12]. The XP practice of Continuous Integration, that is, building and testing a system frequently, maximizes the use of the executable and automated

products of Test Driven Development (TDD) [13]. TDD visibly links executable unit tests to the overall development process. TDD is widely practised and has many reported benefits [14] but successful use does rely on tools such as JUnit [15]. ATDD adds to this established test-first philosophy with acceptance testing of an automated and executable nature. In keeping with agile principles, ideally customers write acceptance tests guided by developers. Its practice "allows software development to be driven by the requirements" [16]. A key advantage of ATDD in its wider context is that it leverages existing agile infrastructure supporting continuous integration. As with TDD, support from tools makes ATDD feasible. However, Andrea [17] claims that existing tools exhibit several deficiencies and produce tests that are "hard to write and maintain". To overcome this Andrea also suggests that the next generation of functional testing tools need to support writing (and reading) functional tests in multiple formats. Given the widespread adoption of information and communication technology, in many organisations business rules are documented in numerous formats, for example, including Medical Devices . However, ATDD is currently not well supported with tools that enable reusing such existing documents, without rewrites, to create executable tests. A challenge, therefore, is to support a suitably informed expert to perform the agile *customer role* and in easily creating tests from existing material. However, successful identification of accurate acceptance tests in this manner is not necessarily straightforward.

2 Importance of "Well Tested" Medical Device Software

The risk of patient injury from software defects is a concern due to the manufacture and deployment of increasing numbers of software-embedded medical devices [18] - [20]. There have been a number of major medical device product recalls over this past 25 years that were the result of software defects [21]. Highly traceable testing and change control procedures within medical device software development is important as such modifications can occur frequently and may occur at different levels (e.g. design, interface or code), therefore increasing the risk of software failure [21]. It is therefore important that a medical device company has an efficient software development process in place that include change control practices. According to the Institute of Medicine report "To Err is Human" [22], between 44000 to 98000 people die in hospital from preventative medical errors. The report also says that more people die every year as a result of medical errors than from motor vehicle accidents, breast cancer or AIDS. Like most industries, the medical device industry depends on computer technology to perform many of the functions ranging from financial management to patient treatment [23]. The use of software in medical devices has become widespread in the last two decades. Medical devices with software include those that are supplied and used entirely in hospitals and other health facilities, as well as consumer items such as blood pressure monitors. Many medical devices, and their software, operate in real time - monitoring, diagnosing, or controlling a physiological process as it changes. The complexity and risk profile of med-

ical devices varies widely and range from a consumer digital thermometer for minor diagnosis, and an implanted artificial heart that is critical to preserving a patient's life, to a therapeutic X-ray machine with a computer user interface, programmable software controlled therapy and anatomical and biophysical modelling in the software, which is operated under a high level of professional staff supervision [24]. Analysis of medical device recalls highlights the diverse nature of medical device software failures. The FDA found that during 1983 - 1987 approximately 44% of the quality problems that led to voluntary recalls of medical devices were attributed to errors or deficiencies designed into particular medical devices rather than having been inserted during the manufacturing phase. The study also recognised software quality management practices as a means to prevent failure [25]. In the medical device industry, the software used to control a device takes on an additional role - it must help ensure the safety of the user. There are many challenges to implementing safe software. Software design needs to include deliberate engineering practices and rigorous approaches for software testing such as an expert customer defining suitable tests before development begins.

3 Related Work

Many approaches to conducting acceptance testing exist. Some concentrate on acting as a "recording device" allowing user actions to be replayed against a system, checking for deviations. However, this approach is mainly limited to Graphical User Interface (GUI) testing of a specific version of a system, using a tool such as the Selenium IDE [26]. Tools for writing acceptance tests in a customer friendly format and appropriate for continuous integration exist. RSpec, for example, is a "Behaviour Driven Development framework for Ruby" [27]. It promotes a workflow that involves writing stories in a somewhat prescriptive natural language style and then manually translating these steps into Ruby. While the authors consider this approach interesting for new stories, it has limitations in dealing with pre-existing documents. Other open source tools aimed at supporting ATDD exist including EasyAccept which supports both tabular and sequential styles [28].

Generally, the Framework for Integrated Tests (FIT) is the most widely accepted tool for managing acceptance tests in agile development and therefore practising ATDD [29]. In FIT's simplest workflow a user, places inputs and some expected output into a tabular format, a *ColumnFixture* [30]. The developer then writes code (*fixtures*) that executes this data against the system's production code. Other built-in fixture included in FIT include *ActionFixtures* for testing a "sequence of commands" and *RowFixtures* for "comparing test data to objects in the system" [30]. FitNesse is a Wiki framework developed to support FIT [31]. It facilitates the editing of FIT tables in a browser allowing non-programming experts to add content. While FIT tables can be written in any tool that can export HTML, such as Microsoft Excel, these generic tools do not have any authoring features directly supporting the task domain. Existing

tools that support either FIT or FitNesse include AutAT and FitClipse. AutAT seeks to assist "business-side people" taking a visual approach to building Acceptance Tests [32]. As FitClipse [33] builds on FitNesse tests are entered using its wiki syntax. Mugridge introduces a process based around a library of *fixtures* named FitLibrary, which improves FIT's "business-level expressiveness" to emphasise a "domain-driven design approach" [34]. It supports a type of fixture, *DoFixtures*, which approach natural language in readability. Commercial software also supports such a workflow, with GreenPepper [35] supporting "executable specifications" while providing an expressive library of table types. For clarity, it is important to note that GreenPepper uses code annotations (Java and C#) that are unrelated to the annotations in this paper. However, none of these tools is focused on reusing existing documentation, so unlike the proposed approach these approaches require re-writes of content.

In the requirements authoring process, Melnik and Maurer found that the use of FIT helped students to "learn how express requirements in a precise, unequivocal manner" [36]. In a number of experiments aimed at evaluating the impact of FIT tables on the implementation of change requests Ricca et al. [37], found improvement in the correctness of code produced. The addition of FIT Tables to plain text descriptions had the most impact on more experienced students, and they found no significant increase in time taken to implement the changes. The use of annotations was proposed because it provides users with a simple conceptual framework allowing them to add detail to text descriptions of tests. Annotations are used here to allow for links to be made between descriptions and corresponding FIT Tables. These annotations are based on elements of an acceptance test description recommended by Jain [29]. There are four basic types, covering most elements of an individual acceptance test:

- *Precondition*: event that must occur before a test is run.
- *Actor + Action*: part of system and functionality.
- *Observerable Result*: a verifiable response generated by the system.
- *Examples*: represent the input data given to a test.

The passing or failure of a test rests with variance from specified *Observable Results*. A visual representation of the annotations is contained in Figure 1.

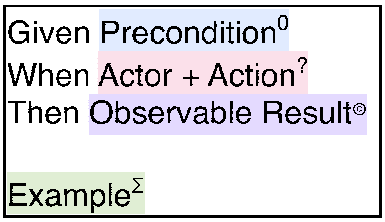

Fig. 1. Annotations

4 Annotations Experiment

This experiment was designed to evaluate the impact of annotations on the process of authoring acceptance tests. The scenario used to write the question descriptions given to respondents concerned the management of software packages on a computer system, such as GNU/Linux [38]. There were six participants, each experienced in computing as either a postgraduate or professional. However, none had prior experience of writing FIT tables. All were given a short, two-hour training session on FIT Tables and ATDD. Participants were tasked to create tests using either annotated descriptions or from non-annotated plain text descriptions. The plain text descriptions serve as a reference for comparison against annotations. The only difference between descriptions was the presence or absence of annotations. Each participant was randomly assigned to Group A and Group B, with each group assigned in total three participants and receiving four questions. Group B started with annotated descriptions while Group A were given a non-annotated version. For subsequent exercises the groups alternated between annotated and non-annotated. Apart from a common assignment of question, to their group, participants worked alone. In providing these descriptions, the first author acted in the role of a customer on an agile project. The experiment considered annotations in paper-based experiment in isolation aside from usability considerations of prototypes.

4.1 Design

For comparison purposes, the first author wrote reference tests, providing an "ideal" test description against which the participants' tests were compared. Each was in the form of high-level descriptions of how a system should function, including handling of error conditions and intended to be of approximately equal difficulty: Question 1 covered *initial bootstrap* of the package management system; Question 2 covered *installation* of new packages; Question 3 covered *removal* of packages; Question 4 covered *upgrading* of packages. The metrics used to assess the experiment were gathered under the following headings:

- *Errors*: elements that should not appear in the test. From participants' answers, all error occurrences counted towards the average.
- *Correct Elements*: From participants' answers an elements first occurrence. Participants were free to reuse structural elements (for example the first row in a FIT Table) as this only affects readability. However, repeated data elements are counted as *Errors*. Presence of a data element irrespective of corresponding structural element was enough for it to count as correct, so two penalising respondents twice.
- *Missing Elements*: defined as elements that were omitted by the participants compared to the reference test.
- *Time*: amount of time taken to complete FIT table.

4.2 Question and Responses

A reproduction of Question 2 with annotated text is presented in Figure 2. This version was provided to Group A while Group B received it non-annotated.

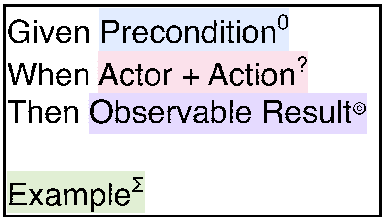

Fig. 2. Sample Question

A simple FIT Table (ColumnFixture) has been transcribed in Figure 3, it represents the text of Figure 2. This acknowledges the flow of events encoded in the text and unambiguously represents the specific package name of the "conflicting package".

package	install()	failReason?	failPackages?
fcron	TRUE		
vcron	FALSE	Package conflict	fcron

Fig. 3. Sample 'Ideal' Answer

For illustration and comparison with the "ideal" response, two respondent answers are transcribed in Figure 4 and Figure 5. Figure 4 the answer attempt from respondent A2, who had been provided an annotated version of Question 1.

package	success?	
fcron	TRUE	
vcron	FALSE	"Package Conflict"

Fig. 4. Respondent answer (annotations)

Here, the respondent A2 correctly identifies the sequence of events, but fails to include the name of the package, "fcron", causing the failure. However, the chosen label heading "success?" does not reflect the action name but this is not considered an error because the respondent correctly labelled the table. Respondent A2 achieved the fewest Errors and both the most Correct Elements and fewest Missing Elements in Question 2.

Install	Result
Duplicate (vcron)	Package Conflict

Fig. 5. Respondent answer (non-annotated)

The corresponding snippet from respondent B1, who had used a non-annotated version, is transcribed in Figure 5. Here, the respondent B1 failed to identify from the text that the "install()" action should fail due to the prior installation of a conflicting package. Indeed respondent B1 didn't correctly identify "install()" as an action at all, instead specifying the package name "vcron" combined with the error detail as data to be verified. In comparing these answers with the reference answer in Figure 4 one element was missed by respondent A2 while four elements were missed by respondent B1 in Figure 5. Finally, it should also be noted that respondent B1 performed better when using annotated texts and respondent A2 performed worse when using non-annotated texts. The next section summarises the overall results for the experiment.

4.3 Results

The results gathered from the respondents answers, are summarised in Table 1. For clarity, the *row number* is included in column 1. Columns 2, 3 and 4 introduce the *question number*, which *group* is responding (A or B) and the *type* of description provided in the group's question. Columns 5, 6 and 7 contain the arithmetic mean of the counts for each group's *Errors, Correct Elements* and *Missing Elements*, respectively. The presence of *Errors* indicates *Over-Specification* while that of *Missing Elements* indicates *Under-Specification*. In all cases, *Correct Elements* plus *Missing Elements* equals *Total Elements* of the "ideal" answer. We analysed both the data element and the structural element of the responses. An *Error* occurs whenever a response is matched against the "ideal" answer and a mistake is identified. A mistake may be identified in either the data element or the structural element. All mistakes that occur in the data element are counted as errors, whereas only the first occurrence is counted as an error in the structural element. For example, if we matched an individual's response against the "ideal" response and discovered that a data element "fcron" had been included by a respondent three times; the first two match the "ideal" response counting as *Correct* but the third element would be incorrect and count as one error.

Table 1. Results from annotations experiment

Row	Q	Group	Type	Errors	Correct	Missing
1	Q1	B	Annotated	7.33	12	14
2	Q1	A	Plain	13	14	12
3	Q1	-	Difference	55.74%	(15.38%)	(15.38%)
4	Q2	A	Annotated	4.67	14	7
5	Q2	B	Plain	9.67	10.67	10.33
6	Q2	-	Difference	69.77%	27.03%	38.46%
7	Q3	B	Annotated	9.67	9.67	3.33
8	Q3	A	Plain	11.67	9	4
9	Q3	-	Difference	18.75%	7.14%	18.18%
10	Q4	A	Annotated	11.5	14	6
11	Q4	B	Plain	14	8.67	11.33
12	Q4	-	Difference	19.61%	47.06%	61.54%
13	-	-	Average Difference	40.97%	16.46%	25.70%

Each row in Table 1 presents the results of one group for a particular question. For example, Row 1 represents the arithmetic mean of responses from Group B for Question 1 (annotated). The use of median would not reverse the overall results.

Further, the percentage difference (55.74%) between Group A and Group B is represented in row 3. This is obtained from as follows:

$$Row3 = ((|Row1 - Row2|)/(Row1 + Row2)/2) * 100. \quad (1)$$

For example, in the case of the obtaining the percentage difference of Errors:

$$55.74\% = (|7.33 - 13|)/((7.33 + 13)/2) * 100 \quad (2)$$

In the case of a worse performance when given annotations, such a result has been enclosed with parenthesis in Table 1. This pattern continues for each question given to respondents. The final row, Row 13, contains the overall percentage difference; these results included the cases of decreased performance in Row 3 as negative numbers. In each case, the occurrence of *Errors* is reduced for the annotated versions. This holds across both groups even with a pattern of Group A taking less time on average compared to Group B. For example, the figure of 55.74% in row 3 indicates that there were 55.74% less errors identified in the annotated version. This means responses with a lower incidence of *Over-Specification* occurred when respondents were provided with annotations. In Question 2 to Question 4, the average number of *Correct Elements* for the annotated version is greater than that for the non-annotated version. A similar reduction in the number of *Missing Elements* occurred. For example, 27.03%, *Correct* in Row 6 means that there were 27.03% more elements identified by the group given annotations. Similarly, 38.46%, *Missing* in Row 7 means that there were 38.46% less missing elements identified by the group given annotations.

As with *Error Rates*, the number of *Correct Elements* achieved by respondents appears unrelated to the amount of time spent. However, the effect of annotations on *Correct Elements* and *Missing Elements* was smaller than on the *Error Rates*, therefore annotations had less of an impact on *Under-Specification*.

4.4 Selection of a Domain

The initial results are promising however the chosen domain used in the experiment is one of largely unregulated innovation; therefore the large tracts of documentation required for the approach do not exist. However, medical device companies must produce a design history file detailing the software components and processes undertaken in the development of their medical devices. Due to the safety-critical nature of medical device software it is important that highly efficient software development practices are in place within medical device companies. Medical device companies who market within the USA must ensure that they comply with medical device regulations as governed by the FDA (FDA - Food and Drug Administration) [39] - [6]. The medical device companies must be able to produce sufficient evidence to support compliance in this area. To this end, the (CDRH - Center for Devices and Radiological Health) has published guidance papers for industry and medical device staff which include risk -based activities to be performed during software validation [4], pre-market submission [5] and when using off-the-shelf software in a medical device [6]. Although the CDRH guidance documents provide information on which software activities should be performed, including risk based activities; they do not enforce any specific method for performing these activities. The FDA have defined the following eleven software development areas:

- *Level of Concern*
- *Software Description*
- *Device Hazard and Risk Analysis*
- *Software Requirements Specification*
- *Architecture Design*
- *Design Specifications*
- *Requirements Traceability Analysis*
- *Development*
- *Validation, Verification and Testing*
- *Revision Level History*
- *Unresolved Anomalies*

The research outlined in this paper with tool support could greatly assist medical device software development companies to have traceability of all requirements throughout the testing phase and to ensure that all requirements are thoroughly tested. In particular, this would assist medical device companies to adhere to the FDA demands in relation to “Requirements Traceability Analysis” and “Validation, Verification and Testing”.

4.5 Conclusions and Future Work

The annotations experiment in this paper was designed to evaluate the impact of annotations on the process of authoring acceptance tests. Future work in the form of case studies will be aimed at measuring the stages of error detection encountered on projects applying digital annotations; this will asses if the approach helps to highlight deficient documents in-place, encouraging correction at source rather than through creation of second generation artefacts (for example, by writing new acceptance tests). While the size of groups in this study has limited the statistical conclusions, the results presented in this paper indicate that using annotated documents helped to identify more elements that are *Correct* with fewer *Missing* elements and *Errors* when creating acceptance tests.

Due to the applicability of this research to medical device software we would now like to re-design this experiment so that it concerned medical device software requirements. This work will specifically help medical device companies to address two of the eleven areas defined by the FDA i.e. "Requirements Traceability Analysis" and "Validation, Verification and Testing".

4.6 Acknowledgments

This research is partially supported by Institutes of Technology, Technological Sector Research Programme, Strand 1 Fund and Science Foundation Ireland through the Stokes Lectureship Programme, grant number 07/SK/I1299.

References

1. Sommerville, I. *Software Engineering*, 8th edition, pages 80-81, Addison-Wesley, 2007.
2. Pressman, R. S. *Software Engineering: A Practitioner's Approach*, European Adaption, 5th edition. McGraw-Hill, 2000.
3. G. Tassey, "The economic impacts of inadequate infrastructure for software testing", National Institute of Standards and Technology (NIST), May 2002.
4. CDRH, *General Principles of Software Validation; Final Guidance for Industry and medical device Staff*. January 11, 2002
5. CDRH, *Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices; Guidance for Industry and medical device Staff*. May 11, 2005
6. CDRH, *Off-The-Shelf Software Use in Medical Devices; Guidance for Industry, medical device Reviewers and Compliance*. Sept 9, 1999
7. ANSI/AAMI/ISO 14971, *Medical devices - Application of risk management to medical devices*, 2nd Edition. 2007.
8. ANSI/AAMI/IEC 62304, *Medical device software - Software life cycle processes*. July 19, 2006)
9. ISPE, GAMP *Guide for Validation of Automated Systems*. December, 2001.
10. Beck, K. and C. Andres. *Extreme Programming Explained: Embrace Change*, 2nd edition, Addison Wesley, Boston, 2005.

11. Sauvé, J. P. and Neto, O. L. A. Teaching software development with ATDD and EasyAccept. In SIGCSE '08: *Proceedings of the 39th SIGCSE technical symposium on Computer Science Education*, 2008, pages 542-546.
12. Cohn, M. *User Stories Applied*. Addison-Wesley, Boston, 2005.
13. K. Beck, *Test Driven Development: By Example*, Addison-Wesley Professional, 2002.
14. Jeffries, R. and Melnik, G. "Guest editors' introduction: TDD- the art of fearless programming". *IEEE Software*, volume 24(3), pages 24-30, 2007.
15. Kent Beck, Erich Gamma, and David Saff. JUnit 4. Website, URL last accessed 16th January 2009: http://junit.sourceforge.net/
16. Park, S.S. and Maurer, F. "The benefits and challenges of executable acceptance testing", in APOS '08: *Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral*, 2008, pages 19-22.
17. Andrea, J. "Envisioning the next generation of functional testing tools". *IEEE Software*, volume 24(03), pages 58-66, 2007.
18. Crumpler, E.S. and Rudolph H. "FDA software policy and regulation of medical device software", *Food Drug Law Journal*, volume 52, pages 511-516, 1997.
19. Munsey, R. R.,"Trends and events in FDA regulation of medical devices over the last fifty years", *Food Drug Law Journal*, volume 50, pages 163-177, 1995.
20. "Medical device reporting: Improvements needed in FDA's system for monitoring problems with approved devices", US General Accounting Office, GAO/HEHS-97-21, 1997
21. Bovee, M.W., Paul, D. L. and Nelson, K. M. "A Framework for Assessing the Use of Third-Party Software Quality Assurance Standards to Meet FDA Medical Device Software Process Control Guidelines", *IEEE Transactions on Engineering Management*, volume 48(4), pages 465-478, 2001.
22. Kohn, L., Corrigan, J. and Donaldson, M., editors, To Err is Human: Building a Safer Health System, National Academy Press, 2000.
23. Wallace, D. R. and Kuhn, D. R. "Failure Modes in Medical Device Software: An analysis of 15 Years of Recall data", NIST. URL Last accessed January 2007: http://csrc.nist.gov/staff/kuhn/final-rqse.pdf
24. Jamieson, J. "Regulation of medical devices involving software in Australia - an overview", 6th Australian Workshop on Safety Critical Systems and Software, Brisbane 2001.
25. Leffingwell, D. A., Widrig, D. R. and Morrissey, W. T. "Applying requirements management to medical devices utilizing software", Rational Software Corporation 1997.
26. Kasatani, S. Selenium IDE. Website, URL last accessed 1st December 2008: http://seleniumhq.org/
27. RSpec Development Team. Website, URL last accessed 1st December 2008: http://rspec.info
28. Sauvé, J.P., Cirne, W., Osorinho and Coelho, R. EasyAccept Sourceforge Project. Website URL last accessed 3rd Dec 2008: http://easyaccept.sourceforge.net
29. Jain, N. Acceptance Test Driven Development. Presentation URL last accessed 30th Nov 2008. http://www.slideshare.net/nashjain/acceptance-test-driven-development-350264/
30. W. Cunningham, Framework for Integrated Test, September 2002. Website, URL last accessed 16th January 2009: http://fit.c2.com
31. FitNesse.org. Website, URL last accessed 7th February 2008: http://fitnesse.org

32. Schwarz, C., Skytteren, S. K., and Øvstetun, T. M. "AutAT: an eclipse plugin for automatic acceptance testing of web applications". In OOPSLA '05: *Companion to the 20th annual ACM SIGPLAN conference on OOPSLA*, 2005, pages 182-183.
33. Deng, C., Wilson, P., and Maurer, F. "FitClipse: A FIT-based Eclipse plug-in for Executable Acceptance Test Driven Development". In XP 2007: *Proceedings of the 8th International Conference on Agile Processes in Software Engineering and eXtreme Programming*. 2007.
34. Mugridge, R. "Managing agile project requirements with storytest-driven development", *IEEE Software*, volume 25 (Jan.-Feb. 2008), pages 68-75, 2008.
35. Pyxis Technologies inc., GreenPepper Sofware, Website, URL last accessed 19th January 2009: http://www.greenpeppersoftware.com/confluence
36. Melnik, G. and Maurer, F. "The practice of specifying requirements using executable acceptance tests in computer science courses". In OOPSLA '05: *Companion to the 20th annual ACM SIGPLAN conference on OOPSLA*, pages 365-370, 2005.
37. F. Ricca, M. D. Penta, M. Torchiano, P. Tonella, M. Ceccato, and C. A. Visaggio, "Are fit tables really talking?: a series of experiments to understand whether fit tables are useful during evolution tasks", in ICSE '08: *Proceedings of the 30th international conference on Software engineering*, 2008, pages 361-370.
38. Free Software Foundation, About the GNU Project. Website, URL last access 16th January 2009: http://www.gnu.org/gnu/the-gnu-project.html
39. FDA's Mission Statement. URL last access 18th March 2009: http://www.fda.gov/opacom/morechoices/mission.html