# A Survey on Testing and Reuse

Richard Torkar and Stefan Mankefors
*Dept. of Informatics and Mathematics*
*University of Trollhättan/Uddevalla*
*P.O. Box 957, SE-46129 Trollhättan, Sweden*
*{richard.torkar, stefan.mankefors}@htu.se*

## Abstract

*This survey tries to give an account of what type of trends exist today in software reuse and testing. The focus was to try to find out how developers use different tools today and what tools are lacking, especially in the field of reuse and testing. The population came from different types of communities and organizations, to better give us a generalized picture of today's developers. We found that a majority of the developers participating in the survey did not test reused code and other testing methodologies were not used to the extent that the scientific community takes for granted. A more automated approach to testing in combination with code coverage analysis and statistical analysis was found to be needed.*

## 1. Introduction

Software engineering today needs best practices and tools to support developers to develop software that is as fault free as possible. Many tools and methods exist today but the question is *if* and *how* they are used and more importantly in which circumstances they are (not) used and why.

A few surveys in this field, e.g. The CHAOS report by The Standish Group [17] which covers software failures in the industry and recently (Aug 2002) the FLOSS survey by Ghosh et al. [7] [8], which is a survey/study about developers in the open source [11] and free software [12] world, do exist. But they either focus on a precise population, with its advantages and disadvantages or cover the result of not testing ones software enough.

We believe that there is a need for a more integrated test methodology together with the traditional configuration management process in order to improve the current situation. This paper covers a survey that took place during late 2002, with the aim to answer some of the questions our research team had with respect to testing and reuse, two areas not usually covered very well in surveys. We wanted to know to what extent reuse was taking place and how frequently reused code was being tested.

In our survey we asked software developers from several companies, both national (Swedish and American) and multinational, as well as open source developers from several projects about what type of problems they faced daily in their work. Not surprisingly the answers varied, but many developers gave us the same basic feedback - the systems designed today are complex and the tools for creating these systems are getting more and more complex as well. This indicated that software developers could, among other things, benefit from more integrated and automated testing in today's software development projects.

Yet, other questions in this survey, focused on reuse and testing of re-usable components and code. We wanted to know to what extent reuse was taking place today in projects, how developers test this type of code and if they use some sort of certification. Unfortunately, this [test of reused code] was not the case among the developers in our population.

All questions discussed in this paper can be found in appendix 1.

### 1.1. Background

Many texts today exist concerning the area of testing. Testing Object-Oriented Systems by Binder [2], The Art of Software Testing by Myers [3] and How to Break Software by Whittaker [4] all give a good insight. For more information about testing - especially unit testing – we recommend reading the IEEE Standard for Software Unit Testing [18] and Using Unit Testing Late in a Development Process [5].

Recently the International Institute of Infonomics at University of Maastricht in the Netherlands [6] published a report (FLOSS) that covered a survey about open source and free software in Europe. This study is interesting in many ways, especially so since some of the questions in their survey touch the areas of software development and software in general. Part IV [7] and V [8] in the FLOSS

Proceedings of the IEEE International Conference on Software—Science, Technology & Engineering (SwSTE'03)
0-7695-2047-2/03 $ 17.00 © 2003 IEEE

study is partly being compared to the survey that we carried out.

## 2. Methodology

The research upon which this paper is based was done in five separate phases. The first phase was to gather good candidate questions that we and other researches in our team would like to have answered. During a meeting several questions came up as good candidates. The second phase consisted of selecting the questions that were of most interest (unfortunately very few developers want to answer 350 questions). The third phase consisted of selecting which population we would use, and finally in the two last phases we established how the questions should be asked and answered and put together additional questions that were not asked in the first questioning round.

The research method we followed during this research was a survey approach [10]. We wanted to conduct a survey that would point out areas that software developers found especially weak and in need of attention. We used empirical inquiries from slightly different populations (open source vs. business) to better examine reuse and testing in today's software projects.

One of the disadvantages of a survey is its time factor. It takes time to prepare and it *steals* time from the population answering the researcher's questions. Gaining access to different company employees, to answer our questions, proved to be the greatest obstacle during this research project.

Another threat to a survey can be the relationship between the questioner and respondent, in our case we estimated this to non-significant as explained later.

Since this research aimed to explain to what extent and how reuse and testing was used today we chose different organizations and type of developers. The main reason for this was that we wanted to make sure the problems we saw when analyzing the answers were in fact problems that more or less all developers - regardless of company or project - found in their daily work.

Since time was a critical factor it meant that a qualitative approach, e.g. interview, was out of the question. The geographic distribution of the population also indicated that we should *not* use a qualitative approach, even though telephones etc. can be used. A quantitative approach was also considered to be the best method, in our case, to more easily draw conclusions in a statistical manner. One must, however, add that a qualitative method probably would have given us a richer set of data on which to base our conclusions upon.

By following the advice in [10] [19] concerning pretests, a first testing round was carried out with four developers participating. Thus we could be relatively confident the questions were of the right type and properly formulated.

The survey used self-administered questionnaires [10] as a foundation, with the addition to a web-based approach. This, in combination with our quantitative approach, made us sure that we did not influence our respondents in any way.

The total number of developers contributing to the survey, during late 2002, was 91 (a further four developers were asked but had no time to participate). Of these 91 developers approximately 43% were from the open source and free software development community and 57% from three different companies; one multinational (approx. 100,000 employees), and two national; one Swedish (approx. 20 employees) and one American with approximately 100 employees. All respondents were either business contacts which have been gathered over time or companies participating in adjacent research projects.

When the survey finished, the answers were checked and if any ambiguous answers were found, the survey participant was contacted and additional questions were asked in order to avoid misinterpretations.

It was stressed, at the introduction of the survey, that the respondent should answer all questions with question number 4 (appendix 1) in mind.

## 3. Presentation

The results are presented in three categories which are discussed; one brief section with general questions and two in-depth sections on reuse and testing. As mentioned previously, all the questions relevant to this paper, are found in appendix 1.

The general questions cover areas such as which development environments or development kits are being used, and the reuse category covers the area of component and general code reuse with accompanying test procedures. Finally, the test category covers questions that more specifically involve different test methodologies and best practices.

## 4. Results and Analysis

### 4.1. General questions

Of the survey participants 55% had an educational level of M.Sc. or higher and only 15% had a high school education or lower. The former number differs from the FLOSS study where only 37% had an educational level of M.Sc. or higher (question 1).

The above variance can be explained by two factors; our survey having a larger degree of developers from the business world as opposed to the FLOSS study Part IV

[7] which only focused on open source/free software developers and the fact [7] that open source developers are younger in general. This was confirmed in question three (appendix 1) which showed us that the population that we used had a higher average age (96% over the age of 21). These numbers, could simply put it mean that we have a more *mature* population in the sense of working experience and educational level.

One of the general questions covered the usage of large frameworks - when asked about different development platforms, such as .NET [13], Enterprise Java [14] and CORBA [15], the majority preferred Java and in some cases even CORBA as opposed to .NET. The main reason was that developers felt .NET being too immature at the moment (mid-2002). Even so the usage of .NET and CORBA was now equal with ~25% each. More recent studies show .NET gaining even more momentum [20]. There is a high probability that .NET will be used even more in the future since it is backed by some major interests in the industry (question 29).

The question "How often do you move deadlines in projects?" (question 11) clearly showed one of the biggest issues in today's software projects (fig 1). The silver bullet has clearly not been found yet.

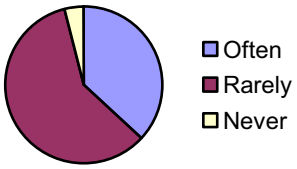



**Figure 1. Moving deadlines**

With 37% of the developers still moving deadlines often and 59% moving them rarely there is still room for improvement. According to [17] 9% of the projects in larger software companies are on-time and on-budget, in our case we have approximately 4% on-time.

Over 53% of the developers (question 17) claimed that the majority of the projects they took part in encompassed more than 10,000 lines of code. According to the FLOSS study [8, p.17] the mean value is 346,403 bytes [of software source code] in the average project. This indicates that the different communities correlate rather well.

As we have seen during the analysis of the general questions not much differs, from other studies conducted in the area of software engineering. This could indicate that we have gathered a good sample population that could answer our reuse and test questions, thus reflecting the average developer, despite us having a smaller population than the FLOSS study. We believe that the validity of the larger FLOSS study with its focus on open source and free software can, in many ways, be generally applicable for the business world.

### 4.2. Reuse

As mentioned previously, the amount of reuse in combination with testing was one of two areas we wanted to focus on since we have not found any surveys covering this area. Never the less, some of the surveys that at least touch this subject are [24] and [25], but they either cover, success and failure examples of reuse in small and medium size enterprises or a particular feature [software repositories] of reuse. Another paper [26] cover large enterprises which are considered to be successful in the area of software reuse.

The developers were asked several questions with regard to reuse in software engineering. Both component-based reuse and clean code reuse (i.e. cut and past). Here one could clearly see a distinction between open source developers and developers from the business world. Almost 53% of the developers said that they usually had some element of reuse in their projects (question 31). But sadly only five of these developers were from the business sector. One of the main reasons for this, we found out when asking the respondents, was that consultants usually do not own the code - the customer who pays for the work owns the code. This, naturally, makes it harder to reuse code later on.

Only 36% of the developers actively search for code to be reused (question 15). The low number is not strange when one considers that developers creating components almost never certify them in any way (either in-house or commercial, e.g. [21]). Only 6% use some sort of certification on a regular basis (question 25).

When it comes to buying components the developers were asked if size or complexity matters the most (questions 32-33), 26% of the developers were of the opinion that size did not matter at all. The complexity aspect of reuse is what makes some developers see a great advantage. Unfortunately, most developers found that components performing complex tasks were hard to find. The reason for this, many developers claimed, is probably that these types of components usually contain business logic made specifically for one company.

### 4.3. Testing

On the question if the developers tested their software, 52% answered yes, and another 34% answered that they sometimes test their code (question 19). Object-oriented testing/Unit testing is without a doubt the most used testing methodology with 78 out of 91 developers (question 20). A previously published study [5] give a good indication on the benefits of said testing. Open source developers and developers from the business world test their code equally according to our survey.

When asked about a unit testing example (question 21), which basically consisted of a simple function, the most common approach (>40% of the developers) tested extreme values only, i.e. boundary value analysis [13, p.1758]. A few developers (~20%) tested random values and almost a third of the developers did not test such a function at all (>30%). The concept of boundary testing seems to be known, both in the industry and in the open source world amongst developers, in general. Even though boundary value analysis only catch some of the faults, it is still encouraging to see that at least this basic test technique is being used, to some extent.

Most developers in this survey used some sort of a testing framework which they themselves did not develop (questions 26-27). A majority of the developers testing their code used some of the unit testing frameworks that exist today, most notably some variant of JUnit [16].

As we showed previously only 4% of the projects the developers took part in were on-time. Sadly these respondents usually did not test their software in any way but instead waited for customer feedback as a form of quality assurance. On the other hand 60% of the developers claimed that verification and validation (V&V) was the first thing that was neglected (question 36). This was mostly common in the business world, unmistakably so since open source developers usually do not have such strict time frames. Could this lead to higher quality in open source software? Some indications exists that this might be the case [28] [29].

Almost 53% of our respondents stated that they had some element of reuse in their code but only 34% of these 53% claimed that they tested the reused code in any way (fig 2) (questions 24, 31).

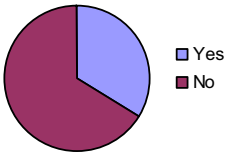



**Figure 2. Dev. with elements of reuse in their projects that also test the reused code**

The reason for this was primarily that they found writing test cases afterwards too tedious. One developer said that while the component was not certified in any way the developers creating it *should* have tested it. The same developer believed that testing software was more or less unnecessary since customer feedback would give him that advantage anyway.

One thing was consistent with all developers. They all wanted better tools for writing tests cases, especially when the code had been written by someone else. Many of them (~75%) also felt that even though they had written test cases they still could not be certain that the tests were good enough (question 37). They, simply put, wanted some statistics on how well their tests were written and how well they tested a given function, class or code snippet, i.e. code coverage [22].

Developers preferring Java had the highest element of testing in their work - this could be explained by developers having the knowledge of JUnit [16] which is considered to be a mature unit testing framework (questions 26-27, 29). The only discrepancy to this was developers in the open source world, they used several different frameworks [16] [23].

Most developers thought that unit testing was tedious and many times not worth the time spent (What is the main disadvantage with the test methodology you use? question 38). As an alternative test methodology, they focused primarily on testing the software under simulated circumstances as soon as possible i.e. a variation of acceptance testing [1, pp.61-62]. We find this to be an alarming sign since unit testing is considered, by many, as being the first vital step in the testing process. By neglecting unit tests many, much harder to find, faults will emerge later on.

If we make a comparison between open source and business, we can see that open source developers in general have a better knowledge of which type of frameworks exists for testing (question 27) - they could, in general, mention several more frameworks they used to cover their needs.

Furthermore, if we combine questions 4, 7-8, 10-11 and 34 it implicates, not surprisingly, that developers in

the industry in most cases have less freedom, higher workload and lack of time. The lack of testing in combination with reuse could be explained by developers claiming that V&V is the first thing being diminished in a project (question 36).

## 5. Discussion

Before we continue, it might be worth mentioning that, our survey had approximately ±5% of *pure* statistical errors, while the FLOSS study with its large number of participants, probably ended up with a ±1% error margin. These numbers are to be considered worst case scenarios but, never the less, must be taken into consideration when comparing results throughout this paper. Even so, we find our numbers being more representative for our purposes than the FLOSS study. We needed a broad population with different backgrounds (business and open source), while the FLOSS study concentrated on open source/free software developers only – with no focus on reuse and testing on the whole.

The results from this survey are in line with other surveys conducted before, when comparing general questions. Some discrepancy exist which can largely be explained by other surveys and studies having a larger contribution from the development community in terms of participation, as well as a different mix of business and open source.

Concerning testing we do not have much to compare with, this is one of the few surveys as of now that have such a strong focus on testing and reuse. Some papers cover some aspect of reuse, as already mentioned, while other [27] cover a combination [quality and productivity].

Simple tools for developing software are still widely used. This is explained by the respondents as being simpler to use while at the same time letting the developers keep the control over their code (question 18). This might also indicate why developers find it tedious to learn new test methodologies and tools – Keep It Simple Stupid (KISS) - is still very much viable in today's projects. This gives us a hint that whatever tools, we introduce to developers, must be kept simple and easy to learn. The best tool would be a tool that the developer does not even notice.

With respect to different development platforms that are in use today - Enterprise Java is holding a strong position. This could very well change soon since already 23% of the developers find (in late 2002) .NET being their primary choice. We believe that this number will rise even more and that this will be one of the two target groups [of developers] where simple and automatic tools could come to the rescue. The second group, of course, being the Java developers.

Developers seem to reuse code to a fairly high extent, but unfortunately they do not test the reused code much. We have already showed [5] the need for testing software, especially software that is reused a lot, in an earlier study.

Here we see developers asking for help to test code that is about to be reused. In addition to that, many developers would like to see a tool that could give them an indication on how well their test cases are written (e.g. test coverage) - again KISS.

Developers today need to have a better knowledge on the importance of unit testing. If the foundation which certain software lies upon is not stable, by not using unit tests, then it risks deteriorating everything. Since the workload is high and deadlines creep even closer, developers must be presented with more automated tools for test case creation and test execution. Also tools that give them an indication on how well the tests cover their software are wanted.

What we found somewhat surprising is the low level of component/library certification taking place. We believed that certification of software had evolved further - beyond academic researchers [30] [31]. This was not true except for a few cases.

In short, to summarize it, some of the key findings in our survey were;

a) developers reuse code but do not test it to the extent we expected,
b) simple to use tools are lacking when it comes to test creation and test analysis,
c) knowledge on the importance of testing in general and unit testing in particular seem low,
d) certification seem to be more or less non-existent.

Since 96% percent of the developers are exceeding their deadlines, at least occasionally, one can claim that there is still much room for improvement.

## 6. Acknowledgements

Prof. Claes Wohlin, Blekinge Institute of Technology, for finding flaws and at the same time seeing the bigger picture. Dr. Christina Cliffordson, University of Trollhättan/Uddevalla, for helping out in forming this survey. Her experience in survey methods gave us some indication on what we should *not* do. Dr. Steven Kirk, University of Trollhättan/Uddevalla, for proofreading this paper over and over again. The European Union regional development fund for funding part of this project.

## References


[1] I. Sommerville, *Software Engineering*, Addison-Wesley, 2001.

[2] R. V. Binder , *Testing Object-Oriented Systems*, Addison-Wesley, 1999.

[3] G. J. Myers, *The Art of Software Testing*, John Wiley & Sons, 1979.

[4] J. A. Whittaker, *How to Break Software,* Addison-Wesley, 2002.

[5] R. Torkar, C. Hansson, A. Johansson and S. Mankefors, "Using Unit Testing Late in a Development Process", Second Conference on Software Engineering Research and Practice in Sweden, 2002.

[6] FLOSS, http://www.infonomics.nl/FLOSS/, 2003-01-29.

[7] A. R. Ghosh et.al., "Free/Libre and Open Source Software: Survey and Study. FLOSS Deliverable D18: FINAL REPORT. Part 4: Survey of Developers", University of Maastricht, The Netherlands, 2002.

[8] A. R. Ghosh et.al., "Free/Libre and Open Source Software: Survey and Study. FLOSS Deliverable D18: FINAL REPORT. Part 5: Software Source Code Survey", University of Maastricht, The Netherlands, 2002.

[9] J. J. Marciniak, *Encyclopedia of Software Engineering,* John Wiley & Sons, 2001.

[10] E. R. Babbie, *Survey Research Methods,* Wadsworth Pub Co., 1990.

[11] Open Source Initiative, http://www.opensource.org/, 2003-01-29.

[12] The GNU project, http://www.gnu.org, 2003-01-29.

[13] T. L. Thai and H. Q. Lam, *.NET Framework Essentials (2nd Edition),* O'Reilly, 2002.

[14] D. Flanagan, *Java Enterprise in a Nutshell (2nd Edition),* O'Reilly, 2002.

[15] F. Bolton, *Pure CORBA,* Sams, 2001.

[16] JUnit, http://www.junit.org, 2002-08-27.

[17] The Standish Group, "The CHAOS Report (1994)", http://www.standishgroup.com, 2003-01-29.

[18] IEEE, "IEEE Standard for Software Unit Testing (ANSI)", 1997.

[19] R. K. Yin, *Case Study Research: Design and Method*, Sage, 1994.

[20] Evans Data Corp., "North American Developer Survey Volume 2, 2002", http://www.evansdata.com, 2003-01-29.

[21] Flashline, "Software Component Certification Program", http://www.flashline.com, 2003-02-07.

[22] T. W. Williams, M. R. Mercer, J.P. Mucha and R. Kapur, "Code coverage, what does it mean in terms of quality?", Proceedings of the 2001 Annual Reliability and Maintainability Symposium, Philadelphia, PA, January 22-25, 2001, pp 420-424.

[23] NUnit, http://www.nunit.org, 2003-02-08.

[24] M. Morisio, M. Ezran and C. Tully, "Success and Failure Factors in Software Reuse", IEEE Transactions on Software Engineering, Vol. 28, Issue 4, pp 340-357, Apr 2002.

[25] J. Guo and Luqi, "A Survey of Software Reuse Repositories", (ECBS 2000) Proceedings, Seventh IEEE International Conference and Workshop on the Engineering of Computer Based Systems, pp 92-100, 2000.

[26] D. C. Rine and N. Nada, "An Empirical Study of a Software Reuse Reference Model", Information and Software Technology, Volume 42, Issue 1, pp 47-65, 1 January 2000.

[27] W. B. Frakes and G. Succi, "An Industrial Study of Reuse, Quality, and Productivity", Journal of Systems and Software, Volume 57, Issue 2, pp 99-106, 15 June 2001.

[28] Resoning Inc, "How Open Source and Commercial Software Compare: A Quantitative Analysis of TCP/IP Implementations in Commercial Software and in the Linux Kernel", http://www.reasoning.com/downloads/opensource.html, 2003-02-27.

[29] J. Forrester and B. Miller, "An Empirical Study of the Robustness of Windows NT Applications Using Random Testing", Proceedings of the 4th USENIX Windows Systems Symposium, Seattle, WA, USA, August 2000, pp. 59-68.

[30] J. Voas and J. Payne, "Dependability Certification of Software Components", Journal of Systems and Software, No. 52, 2000, pp 165-172.

[31] M. Shaw, "Truth vs. Knowledge: The Difference Between What a Component Does and What We Know It Does", Proceedings of the 8th International Workshop on Software Specification and Design, March 1996.

## Appendix 1

*All numbers in this appendix are, if not otherwise stated, in percentage.*

**Q1. What is your educational level?**

| High School | College degree / B.Sc. | M.Sc. or higher |
|---|---|---|
| 15 | 40 | 55 |

**Q2. Are you male or female?**

| Male | Female |
|---|---|
| 98 | 2 |

**Q3. How old are you?**

| <=20 | 21-29 | >=30 |
|---|---|---|
| 4 | 51 | 45 |

**Q4. Do you consider yourself being a; business/industry developer or open source developer?**

| Open Source | Bus./ind. Dev. |
|---|---|
| 43 | 57 |

**Q5. How many developers do you usually have in one team?**

| Less than 10 | Between 10 and 20 | More than 20 |
|---|---|---|
| 77 | 14 | 9 |

**Q6. Is there usually interaction between your team/project and other team(s)/project(s)?**

| Yes | Rarely | No |
|---|---|---|
| 26 | 2 | 72 |

**Q7. Do you find yourself having much freedom in your work as a developer?**

| A lot of freedom | Some freedom | Quite limited freedom |
|---|---|---|
| 75* | 22 | 3 |

* Of the developers experiencing “A lot of freedom” 80% where from the open source world.

**Q8. Are you involved in setting requirements or specifications for software?**

| Yes | Rarely | No |
|---|---|---|
| 94* | 2 | 4 |

* Open source developers seem to be somewhat more involved in this case - although this falls within the ±5% error margin.

**Q9. How do you know if you've fulfilled the requirements, specifications or goals for a particular software?**
(Only a few answers presented)

Customer satisfaction
Time will tell
I just know
Through testing of software and user feed-back
Through customers quality assurance testing

Proceedings of the IEEE International Conference on Software—Science, Technology & Engineering (SwSTE’03)
0-7695-2047-2/03 $ 17.00 © 2003 IEEE

**Q10. Do you have flexible time frames when working in projects?**

| Yes | Rarely | No |
|---|---|---|
| 67 | 22 | 11 |

* Not surprisingly, the vast majority that answered yes in this question, were from the open source world (85%).

**Q11. How often do you move deadlines in projects?**

| Often | Rarely | Never |
|---|---|---|
| 37 | 59 | 4 |

* 34 developers answered yes in this question, almost all of them were from the industry.

**Q12. During a project that is proceeding according to plan, how much of your total time spent on the project do you approximately spend on:** *

| | Lot of time | Some time | No time |
|---|---|---|---|
| Analysis | 33 | 61 | 6 |
| Design | 45 | 53 | 2 |
| Implementation | 70 | 30 | 0 |
| V&V / testing | 35 | 63 | 2 |

*This was unfortunately a question that became very hard to analyze

**Q13. Which part do you find most troublesome?**

| Analysis | Design | Implementation | V and V / testing |
|---|---|---|---|
| 25 | 16 | 35 | 24 |

**Q14. Have the projects you've been taking part in stored and documented components/libraries in a systematic way for later reuse in other projects?**

| Yes, often | Yes, but seldom | No |
|---|---|---|
| 47 | 39 | 14 |

**Q15. How often do you search for re-usable code (i.e. libraries, components, classes) instead of doing it yourself?**

| Always | Rarely | Never |
|---|---|---|
| 36 | 62 | 2 |

**Q16. How many classes does your average project encompass?**

| Less than 100 | between 500 - 10,000 | More than 10,000 | Could not answer |
|---|---|---|---|
| 35 | 41 | 21 | 3 |

**Q17. How many lines of code does your average project encompass?**

| Less than 500 | Between 500 and 10,000 | More than 10,000 |
|---|---|---|
| 4 | 43 | 53 |

**Q18. What sort of development environment do you usually use?**

| Console editor and compiler | Fancier GUI editor + compiler | Visual Studio et.al. |
|---|---|---|
| 35 | 31 | 34 |

**Q19. How often do you test your software?**

| Often | Rarely | Never |
|---|---|---|
| 52 | 34 | 14 |

**Q20. What type of structured approach do you use when testing software?**

| Black-box testing | Structural testing (white-box) | Object-oriented testing/Unit testing | Other |
|---|---|---|---|
| 12 | 10 | 72 | 6 |

**Q21. The function** long foo(int i) **takes an int and converts it to a long, which is then returned. Which approach would you like use to test the above method's ability to convert every possible int to long?**
The absolute majority (~75%), that first and foremost tested their software, only tested boundary values. In some cases this was complemented by random values.

**Q22. Do you test a single component or class in any way?**

| Yes | Rarely | No |
|---|---|---|
| 67 | 22 | 11 |

**Q23. Do you test an assembly of components in any way?**

| Yes | Rarely | No |
|---|---|---|
| 67 | 16 | 17 |

**Q24. Do you test a component in any way before you reuse it?**

| Yes | Rarely | No |
|---|---|---|
| 43 | 41 | 16 |

**Q25. Do you use some sort of certification when you have developed a component within your project/company?**

| Yes | Rarely | No |
|---|---|---|
| 6 | 13 | 81 |

**Q26. Do you use any specific test framework?**

| Yes | Rarely | No |
|---|---|---|
| 35 | 18 | 47 |

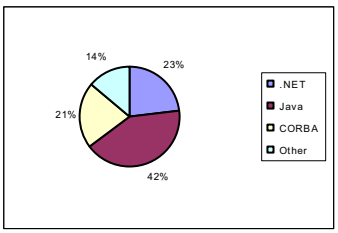
**Q27. If the answer was yes/rarely in the previous question, please stipulate which framework you use.**

Most developers used a variant of Unit testing (usually derived from JUnit).
i.e. NUnit, CppUnit, COMUnit, pyUnit, cUnit, JUnit

**Q28. Does - in your opinion - the choice of framework (.NET, EJB, CORBA) affect the possibility for the software to be easily upgradeable in the long term?**

| Yes | Rarely | No |
|---|---|---|
| 53 | 16 | 31 |

**Q29. Which framework do you use (e.g. Java, CORBA, .NET* et.al.)? ****




* The rather large amount of .NET developers in the open source world was not expected initially. After contacting several of the respondents it became clear that they participated in several open source .NET implementations.
** Open source developers were spread over all four categories – fairly equally. Business developers on the other hand focused mostly on Java and .NET.

**Q30. How often do you rather spend time writing glue code and reuse a class/component, than rewriting it?**

| Often | Rarely | Never |
|---|---|---|
| 46 | 48 | 6 |

**Q31. How often do you reuse a piece of code/class/component from another project?**

| Often | Rarely | Never |
|---|---|---|
| 53 | 47 | 0 |

**Q32. Does the size of (a) component(s) seriously affect decisions on whether you develop it yourself or buy it?**

| Often | Rarely | Never |
|---|---|---|
| 43 | 31 | 26 |

**Q33. Does the complexity of (a) component(s) seriously affect decisions on whether you develop it yourself?**

| Often | Rarely | Never |
|---|---|---|
| 59 | 24 | 17 |

**Q34. Do open source or commercial projects usually enforce a certain technology? (.NET, EJB, CORBA)**

| Yes | Rarely | No |
|---|---|---|
| 55* | 22 | 23 |

* Of the 55% answering "Yes" almost 80% came from the industry)

**Q35. What do you think should be improved in today's component technologies? Do they miss a feature that a developer might need? (i.e. EJB, .NET, CORBA)**
(Only a few answers presented)

- The "[...] ability to concentrate on the business domain. still have to write too much plumbing"
- (1) Easy finding of existing components for reuse. (2) Certification of components. (3) Compatibility between different component technologies.
- Those guys need to agree on ONE standard so you do not need to waste time learning new stuff all the time
- Today's component technologies lack maturity. half of them will be gone in 10 years.
- They are way too bloated.
- I stick with the smaller more specialized components written as libraries or DLL's. They do what they shall, are easier to modify and adapt to special needs.
- Too big, too general, one-fits-it-all will more often fail than help
- Performance/Portability/Speed
- Make the technologies more portable between platforms and app. server vendors.

**Q36. In case of time shortage during a project, which part do you find is being reduced firstly?**

| | |
|---|---|
| Analysis | 16% |
| Design | 20% |
| Implementation | 4% |
| V&V | 60% |

**Q37. When/if you have written test cases for your software, do you feel confident that you have written enough or the right tests?**

| Yes | Rarely | No |
|---|---|---|
| 17 | 10 | 73 |

**Q38. What do you feel is the main *disadvantage* with test frameworks being in use today?**
(Only a few answers presented)

- Hard to get real numbers on how well my tests are written
- Most unit test case generators only do stubs. That is bad...
- I shouldn't need to write even one test case for my software. This should be automated.
- They are not mature enough yet. I don't want to learn a test tool that doesn't give me much help!