

Erlang Testing and Tools Survey *

Tamás Nagy and Anikó Nagyné Víg

Erlang Training and Consulting Ltd, London, United Kingdom

{aniko, tamas}@erlang-consulting.com

Abstract

As the commercial usage of Erlang increases, so does the need for mature development and testing tools. This paper aims to evaluate the available tools with their shortcomings, strengths and commercial usability compared to common practices in other languages.

To identify the needs of Erlang developers in this area we published an online survey advertising it in various media. The results of this survey and additional research in this field is presented. Through the comparison of tools and the requirements of the developers the paper identifies paths for future development.

Categories and Subject Descriptors D.2.5 [Software Engineering]: Testing and Debugging—Testing tools

General Terms Verification, Management

Keywords Erlang, Test Tools, Market analysis

1. Introduction

The aim of this paper is to conduct research on the tool chains used in Erlang software development projects.

- To identify the strengths and weaknesses of the existing practices.
- To point out missing components of a healthy workflow.

The focus is on property and model based testing and test driven development. Furthermore, it aims to create guidelines for further development by identifying the common patterns in this area.

We published an online survey about Erlang tools. This had been advertised on mailing lists and separate emails had been sent out to selected Erlang oriented people. We have got ~200 responses from developers, managers, testers and research engineers.

The research uses the results of the online survey aimed at Erlang developers, but takes into account tools which are currently not used by the Erlang community but have similar functionality to existing ones. This is to provide a cross reference to common practices in other communities.

The next section describes the research methods for collecting information from Erlang users. In Section 3, non-Erlang testing

tools are explored to provide a basis for comparison and determining the functionalities we need (gaps) from Erlang tools. Section 4 focuses on tools used by the Erlang community, using the survey's results to identify the most widely used tools and applications which could be used to create an integrated framework for development. Section 5 rates the applicability of these tools in developing commercial products.

2. Research method

We published an online survey in order to gather data about the usage and spread of Erlang tools and applications in the community. We advertised this via the Erlang mailing list [2], which has around 1000 registered users, and on separate smaller Erlang related mailing lists. For example the trapexit [1] site's mailing list, and the ProTest project's [3] mailing list. This survey was open for everyone, and anonymous submission was accepted as well.

There was a separate survey, with the same questions, open to a selected 200 people who were known to have a background in Erlang but not necessarily in software development.

Overall, for the two surveys, 200 responses were received. The submitters were questioned about their role within their organization. From the answers it is clearly visible that 40-45% were developers and the remaining 55-60% almost equally proportioned between the managers, testers, researchers and students.

2.1 Structure of the survey

The survey contained 20 questions covering four different topics. The questions can be found in the Appendix A. The topics covered were:

1. The development environment of the Erlang users.
2. How widely are the currently available Erlang tools known and used. The participants had the possibility to raise specific problems, if any occurred, which put off the adoption of the tools.
3. The submitter's role within the organization and background in Erlang.
4. Identify common processes which could be helped with tool support.

3. Testing tools

There is a lot of research focused on automated testing with many successful industrial case studies justifying these techniques having a place in a commercial setting.

These tools cater to different purposes. Because of this, it is difficult to show everything in one paper. The aim is to highlight the unique features available in these tools, providing a bridge to Erlang.

* Supported by EU FP7 Collaborative project ProTest, grant number 215868

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Erlang'08 September 27, 2008, Victoria, BC, Canada.
Copyright © 2008 ACM [978-1-60558-065-4/08/09]... \$5.00

3.1 Model based testing

In brief, model based testing means that test cases are derived from a model that describes some aspects of the system being tested.

The Microsoft Spec Explorer [Silva et al.(2008)] is a model based testing tool developed in .NET. There are two specification languages one can use to create the model of a system:

- Spec# - which is an extended version of C# with a possibility to specify pre- and postconditions
- AsmL [Barnett et al.(2003)] - which is an abstract state machine language

One of its features is the possibility to visualise the model of the system. Because the models of large systems tend to be very complex, the visualisation makes it possible to merge states of the system into hyperstates. The idea of hyperstates is merging states together, creating a layered state space, it is possible to de-clutter the model. This technique makes it possible to unfold only those parts of the model that one is interested in.

The tool has a wide range of possibilities to influence the test runs. For example with parameter selection, method restriction and state filtering.

It can be used for offline and on-line testing. Offline means that pregenerated tests are run to test the system. On-line or on-the-fly testing means that the test is generated dynamically as it is running. With online test generation the reproducibility might be lost, but because the test generation can take into account the actual responses of the system, timing and performance also influence the result.

The Spec Explorer has excellent features, but from a commercial point of view it is not applicable because it is only available for non-commercial purposes. There are commercially available model-based testing tools with similar functionalities. For example, Conformiq Qtronic [Huima(2007)] makes it possible to run offline and on-the-fly tests as well.

There are tools which address model based testing through a different approach. They use strictly typed domain specific language to specify the system under test. One of these tools is HOTTest [Sinha et al.(2006)]. Using this approach, assuming that all the domain-specific requirements are available and the model was created, it is possible to automatically generate a test oracle. Furthermore, it is possible to extract domain specific invariants to create additional test cases from the model.

Sinha claims in [Sinha et al.(2006)] that for database systems this technique was the most effective to capture domain-specific requirements either explicitly stated in the documentation/specification or implicit. Implicit requirements are more considered to the writer of the specification, and as a result they are not written down but still part of the model.

There is another group of model based testing tools, closely related to the ones using domain specific languages to specify the model of a system. These testing tools use the UML specification of the system as a model to test against. The difference is that the domain specific languages are, as the name suggests, designed for a specific problem, whereas UML is a general-purpose modeling language. Commercially these tools have wider acceptance, because usually there is already a UML model available to base the testing on. Both Conformiq Qtronic [Huima(2007)], Rhapsody Test-Conductor and TnT [Hartmann et al.(2000)] support testing against UML models. These tools usually provide a graphical interface allowing users to create the model of the system. With complex models however these can become complicated [Sinha et al.(2006)], because of their number of states. There are limitations in graphical systems of how many states can be handled effectively.

3.2 Automated unit testing

Even though unit testing is less time consuming and less complex than system testing, automating parts of the process can help improve quality.

JUnit [Riehle(2008)] and JTest together create a framework for Java which can automatically generate test cases for unit testing. There is functionality to add your own tests to the previously automatically generated ones. In C/C++/C# NUnit provides similar functionality by providing automated testing of classes. What makes these tools highly valuable for the developer is that these tests can be run after every change in the code without major effort.

3.3 Integrated Frameworks

There are directions in the industry/research which aim to create a common framework that provides a well defined interface for different kinds of model based testing tools.

One of these is the AGEDIS [Hartman et al.(2004)] framework. The user has to specify three different things for the testing:

- the behavioral model of the system
- the test execution directives which describe the testing architecture of the system under test (SUT)
- the test generation directives which describe the strategies to be employed in testing the SUT.

Then the system provides three different API for model testing tools to hook into the framework. Using these interfaces the test generation and oracle checks can be specialised without the need of changing the user input.

4. Erlang Tools

In Erlang there are many tools and applications useful during the development and testing process. In this chapter we try to give a short overview with an introduction to the most commonly used tools. The usage and awareness of each tool can be seen in Figure 1.

4.1 Testing tools

Almost every developer uses some kind of test method, either with an existing test environment or with “home grown” functions and applications. From the ratio of the used testing tools it is clear that there is no freely accessible, widely used, flexible and stable tool. Most developers use proprietary tools or manually written test functions.

The available set provides different levels and methods for testing from unit testing to system testing. We will show how well are they adapted to the general requirements to keep the testing methods easy and quick for integration into the development cycle.

4.1.1 EUnit

EUnit [Carlsson and Rémond(2006)] is an open source light weight test server for Erlang. It tries to transfer the main ideas of the existing unit testing frameworks from other languages like SmallTalk (SUnit), Java (JUnit) [Riehle(2008)] applied to Erlang. The main goal is to provide a system where writing test cases is easy, the test cycle is fast and the results of the tests are presented tersely.

During unit testing independent parts of the program are tested separately. The units can be functions, modules, processes or even applications. EUnit allows the developer not only to test functions with assertions, but even to write test generator functions with the support of built-in macros. Test generators provide a representation of tests to be able to run with EUnit.

An important feature is the ability to disable the testing by defining special macros during the compilation or by adding them to the source file.

Erlang Tools

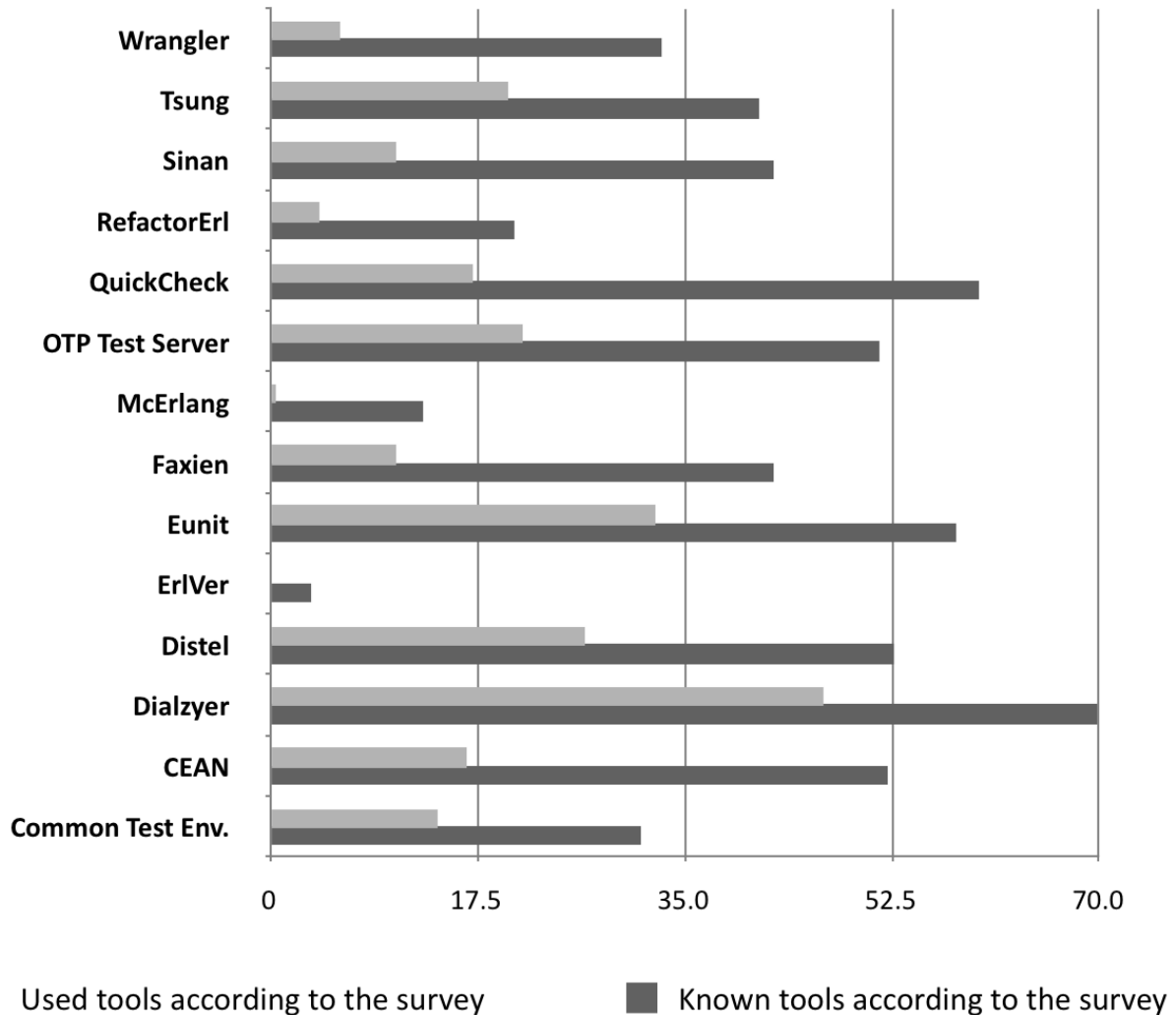


Figure 1. Popularity of Erlang tools among the community(%)

4.1.2 OTP Test Server and Common Test Environment

The OTP (Open Telecom Platform) developers designed a test suite execution environment based on Erlang to support regular automated testing. OTP Test Server [Wiger et al.(2002)] is a portable test server for application testing. The suites can be run on local or remote targets, the progress is logged and the result can be viewed in HTML pages.

Common Test [Blom and Jonsson(2003)] is a framework based on the OTP Test Server application. It is suitable for both black-box and white-box testing. Black-box testing can be done on any type of target system, not just ones implemented in Erlang, as long as the testing can be executed through standard Operation and Maintenance (O&M) interfaces. It provides code coverage analysis by integrating the OTP Cover Tool. According to the survey, the set up of the test environment is complex, suffers lack of documentation and its strict regulation about name conventions makes it hard to use and adopt.

4.2 QuickCheck

QuickCheck [Arts et al.(2006)] is a commercial tool for property and model based testing. It tests running code against formal specification. It can be used at different levels of testing from unit testing to system testing. The test case generation is random but controllable. One of the strongest points of the system is that it has a built in automated test case simplification, called shrinking, to support and facilitate the error.

The system is really efficient for testing and finding problems in complex systems at an early stage of development, especially for testing code against formal specifications. A model can be built according to the specification, and the system can be tested against separate aspects of the built model.

The users would like to have supervision with statistics on what has actually been tested. A dashboard reporting code coverage and executed test cases would rate it user friendly. Another aspect of

QuickCheck is that it has a steep learning curve, even for an Erlang developer.

4.3 Tools for supporting the development cycle

In this chapter we collected projects with different functionalities, which later can be used in an integrated test framework.

4.3.1 Refactoring tools

Refactoring means improving the quality of code through rewriting without changing its external behaviour. The most common program transformations in Erlang are:

- renaming variables, functions, records, modules
- converting data structures (tuples to records)
- function argument operations
- detecting and resolving duplicate code parts

The existing tools are already integrated into the most common editors (in Figure 2). They are based on static analysis of the code and therefore can safely work on big code basis. They can however not support all possibilities of the language; OTP specific behaviours and some of the dynamic function calls are under development, but will not be available for some time.

Wrangler Wrangler [Li and Thompson(2008)] supports both Emacs (with Distel) and Eclipse. The refactoring palette contains seven transformation and two search options.

RefactorErl RefactorErl [Lövei et al.(2007)] has six refactor steps. It stores the analysed source code in a database and provides an interface which can be a base for more applications. It's installation is easy as an installer is provided for Windows, while on Unix systems it is distributed as a source package.

4.3.2 Analysis and checking tools

Dialyzer [Lindahl and Sagonas(2004)] is a static analysis tool for detecting discrepancies in the source or byte code files automatically. Typical errors detected by the tool include

- obvious type errors
- redundant tests
- unsafe virtual machine byte codes
- unreachable code parts

McErlang [Fredlund and Svensson(2007)] is a model checker for verifying distributed Erlang programs. It has excellent support for the most difficult characteristic of the language, namely general process communication, node semantics, OTP component libraries, fault detection and tolerance through process linking.

4.3.3 Automated build tools and package management

Faxien [6] is a package management which helps to find and install or publish OTP applications.

Sinan [5] is a build system for OTP projects. It not only compiles and builds OTP applications but provides a framework for building the documentations, runs dialyzer, checks unit tests, produces output reports and handles all application dependencies.

4.3.4 Load tools

Tsung [4] is an open source multi-protocol distributed load and stress testing tool. Different kinds of servers can be stressed by using the loader, stimulating thousands of virtual users concurrently connecting from several client machines. It is used to test the scalability and performance of the TCP/IP based client-server applications.

Editors

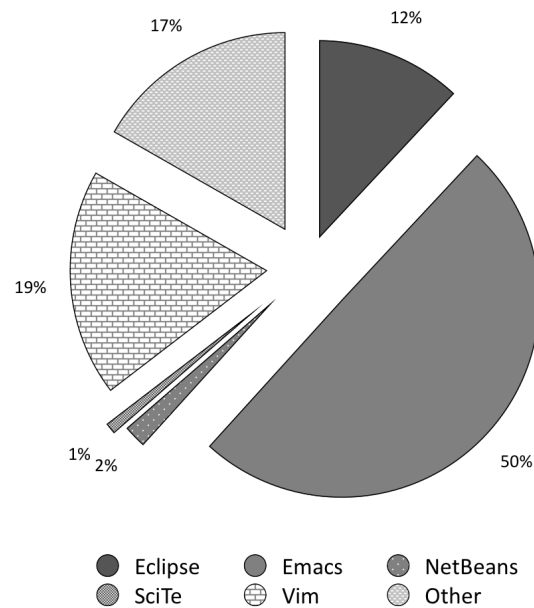


Figure 2. Editor usage among Erlang users.

4.4 Usability of the tools

One of the keys to the popularity of a tool or toolset is user friendliness. The interface should be easy to use, and if it is possible well integrated to the development environment. The layer of the integration can be on the following levels:

1. editors with plugins
2. OS commandline
3. developing software interface (Erlang shell)

It is important to design the interface(s) of the tool, according to the common development environments. Graphs showing the usage of operating systems and editors are in Figure 2 and 3.

According to the survey result the most common editors are platform independent. It is an important result, since the usage of operating systems is quite balanced between the different distributions of BSD, Linux, Macintosh and Windows. All tough Linux distributions are the majority.

The Erlang users' favourite editor is the Emacs with different plugins. The most widely used plugins in descending order are:

- Erlang mode in emacs: included in the Erlang distribution, supports syntax highlight, indentations, comments, function header commands, skeleton templates and compiling. The other Emacs plugins are extensions of this plugin.
- Distel connects to a live Erlang node. It is more than a simple extension of the erlang mode. It can handle dynamic tags, provides the advantages of shell options to the editor such as auto completion of module and function names, process and documentation viewer, a debugger and profiler frontend.
- Eflymake runs a syntax check in the background and highlights the erroneous codeparts

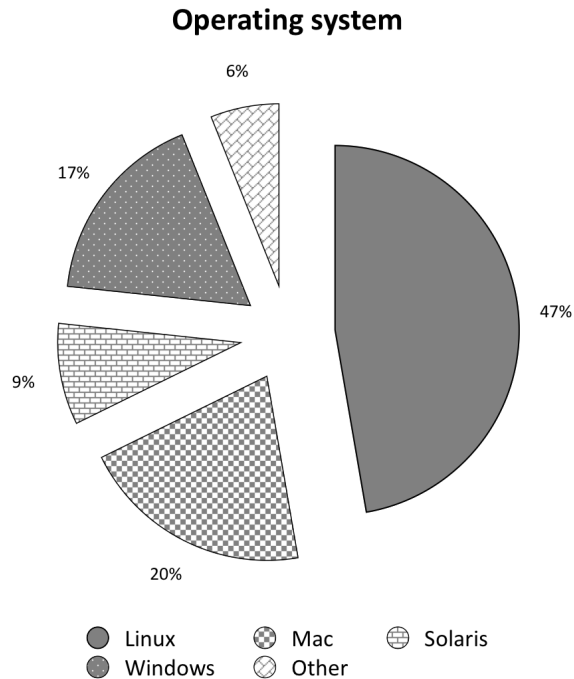


Figure 3. OS usage among Erlang users.

- Wrangler and RefactorErl (see in Section 4.3.1) are refactor plugins built in to the Erlang mode, they can be used with Distel
- Esense or ErlangSense provides features similar to IntelliSense or CodeSense in other editors: completes module names, exported functions, record names, record fields, macros, and shows popup documentation for the above elements.

Most developers use plugins, modes with their editors.

- Eclipse with Erlide is widely used, but some of the programmers found it heavy weighted when compared to Emacs and its Erlang mode. They found it hard to install, and not well tested.
- Vim has syntax highlighting, indentation, and many "home grown" scripts for different functionalities.
- Most popular editors after Emacs and Eclipse are operating system specific with different levels of Erlang support, mostly with syntax highlighting:
 - Textmate with Erlang Bundle
 - Notepad++
 - KDevelop
 - Kate
 - Gedit

There is a need for more complex development helping functionalities such as function argument pop ups, auto complete, function description notes, auto formatting, a good build environment and a debugger.

4.5 Open problems

We asked the survey participants to name problems which make the tools cumbersome to use:

- "Documentation is not enough, and needs some examples"
- "bad documentation"

- "Generally most tools lack the documentation to use them properly without digging into the source code."
- "Generally a lack of tutorials, examples of usage and best practices"
- "Haven't used them enough to comment but cursory opinion seems to indicate that test suites need better management facilities"
- "[any application] is not complete"
- "It is difficult to stub my other modules that are being tested"
- "Unstable and not complete"
- "Most of them are not very thought through. Most of them are badly layered are not extensible in other ways. Most of them organically grown quick hacks without structure"
- "Too much work to set them up"
- "Some of them need a lot of configurations to run, and you need to do it manually"
- "I tried to use dialyzer but it isn't easy to use"
- "It would be great if eclipse plug-ins worked so that I could get the same benefits I get in Java: function argument pop ups, auto complete, function description notes, auto formatting, good build environment and debugger"

The missing applications/functionalities are, identified by the submitters:

- "Specialized Traffic Generators"
- "Create and Interactive protocol tester"
- "It would be nice to be able to somehow generate, at least partially, mock modules to isolate modules under unit test."
- "Continuous integration and hooks for svn, cvs and git"
- "Improve [various testing applications]"
- "A standard harness that runs: a compile check, EUnit tests, Common Tests, the Dializer"

According to these survey results, we would like to highlight the general shortcomings and weaknesses of almost every Erlang tool and project:

1. lack of documentation
2. missing examples and tutorials
3. not completed, tested when published
4. not well designed: badly layered, not extensible, no structure
5. doubts about sustainability
6. hard to install and use especially for non Erlang users
7. sometimes too much configuration is needed and has to be done manually

Even if there are useful tools widely used by the Erlang community, the survey reports that it is still missing functionality and integrated frameworks for:

- The ability to test efficiently on different levels, stubbing functions cause big problems and a lot of effort. At the moment there is no available stubbing tool. No tool can "generate at least partially mock modules to isolate modules under unit test".
- The missing aspect of the testing tools is a high quality display of the results from different interfaces with statistics and graphs.
- Even though there are available load testers for different protocols there is no freely available interactive protocol tester,

framework for testing web applications through http or specialized traffic generators.

- A missing area is continuous integration; hook towards version control systems (svn, cvs and git), integrated into a general framework.
- Existing frameworks do not contain enough functionality. There is a need for a standard harness that runs compile check, unit tests, system tests, xref, dialyzer and tsung performance tests.

5. Applicability of tools in developing commercial products

According to the survey results the testing phases of Erlang projects are poorly supported. Most of the developers use proprietary or manual solutions for unit and system level testing.

From a commercial point of view, the feature set of one tool is less important than other factors namely:

- ease of use
- support
- good documentation
- examples
- ease of comprehending the results

5.1 Ease of use

This covers the ease of initial configuration to non-cryptic error messages related to user interaction. Again the survey showed that the initial configuration is considered hard, manual and error prone work. A lot of test systems tend to return cryptic error messages. Most of these problems could be solved by integrating the tools into one framework, an Integrated Development Environment (IDE). Since this would cut down the configuration needed, the tools would work in the same environment providing similar behaviour.

5.2 Support

Problems with the toolset being used for testing can result in a major loss of development time. It is important to be able to resolve the issue efficiently as soon as possible. This was clearly visible from the survey, because 30% of users would pay for support and training.

5.3 Good documentation

If there is little documentation for a tool the possibility of commercial adoption is low. This was also evident in the survey. Missing or bad documentation was the top problem which came to light with the existing tools.

5.4 Examples

It helps the early adopters if there are available hands on exercises, use cases and examples. If the early impressions are positive the acceptance rate of the tools will be higher. From the survey it is clearly visible that there is a need for training courses.

5.5 Ease of comprehending the results

If it is really hard to interpret the results of a tool it will never pass the evaluation phase. From the survey it seems there is a need for online and offline solutions where more is better. The online means an immediate result as the testing progresses. Offline means a configurable report generation functionality, including browsable webpages, a dashboard or pdf reports.

6. Conclusion

Our research clearly indicates that the tools available for Erlang, commercially or otherwise, have shortcomings in many fields, although they address relevant issues.

One of these shortcomings, for most of the tools in the survey, is the lack of tutorials and examples alongside vague documentation. Even if these tools are useful, addressing existing problems should be a priority of the community. This is clearly visible from the fact that 30% of the survey participants are willing to pay for consultancy on some tools together with a wide adaptation of QuickCheck.

The other field that limits widespread use is lack of automated build and test of software. Even though there are tools solving this problem, they are not integrated with each other, and their output is very diverse. The survey shows that most people yearn after an IDE where building, testing and reporting the results are integrated in the overall workflow.

Although it has already been mentioned, the need for a reporting functionality cannot be emphasised enough. One of the most important functionality is the ability to report. The more structured information available, the better. For example in the model-based testing case (see in Section 4.2) it is really hard to get information about coverage and what has been actually tested.

Many of the mentioned test tools can be used in the development of non-Erlang projects. They are however, rarely used by companies who do not have a history of using Erlang. As Erlang gets wider acceptance and is used to test non-Erlang projects, it is important to tackle issues concerning user interaction, as they will affect the evaluation and adoption of Erlang.

A. Appendix: The questions of the survey

1. Which editor do you use for Erlang development?
 - (a) Eclipse
 - (b) Emacs
 - (c) NetBeans
 - (d) SciTe
 - (e) Vim
 - (f) Other
2. Do you use special Erlang plugin/mode with your editor (Erlide, ErlyBird, Distel, Emacs mode, etc.)?
3. On what platform is the development done?
 - (a) Linux/Unix: Debian, Redhat, Suse, Ubuntu, Other
 - (b) Macintosh: Tiger, Leopard, Other
 - (c) Solaris
 - (d) Windows: XP, ME, Vista, Other
 - (e) Other
4. Have you ever heard of the following Erlang tools?
 - (a) CommonTest Environment
 - (b) CEAN
 - (c) Dialyzer
 - (d) Distel
 - (e) ErlVer
 - (f) Etomcr1
 - (g) EUnit
 - (h) Faxien
 - (i) McErlang
 - (j) OTP Test Server
 - (k) Quickcheck
 - (l) RefactorErl
 - (m) Sinan
 - (n) Tsung
 - (o) Wrangler
 - (p) None of the above
 - (q) Other
5. Have you used the following Erlang tools?

The option list is the same as in the previous question.
6. What Erlang open source applications have you used?
 - (a) CouchDB
 - (b) Ejabberd
 - (c) Erlinda
 - (d) ErlSDB
 - (e) ErlSoap
 - (f) Erlsom
 - (g) Erlyweb
 - (h) OSERL
 - (i) Yaws
 - (j) Mochiweb
 - (k) Tsung
7. Would it help if we generate automated tests for these open source applications?
8. Do the tools you are using have any shortcomings which make them harder to use?
9. Are there any other tools you tried but decided not to proceed with? What were their shortcomings?(Unstable, Not completed, Not resolving my problem, etc).
10. Do you have automated builds and test suites in place? If so, what system are you using?
11. What methods are you using to test your system?
 - (a) Test driven development
 - (b) Develop-test iteration
 - (c) Black box testing
 - (d) Gray box testing
 - (e) White box testing
12. What testing tools are you using?
 - (a) Common Test
 - (b) EUnit
 - (c) Manually written functions
 - (d) OTP Test Server
 - (e) QuickCheck
 - (f) Other
13. Would you be interested in trying out the test tools which will result from the protest project?
14. Do you have time and resources to try new tools?
15. Would you prefer training courses - with hands on exercises - for these new tools?
16. Would you consider short term consultancies?
17. Would you be willing to pay for these courses and consultancies?
18. Imagine that you have three wishes, what Erlang testing applications would you like to: create or improve already existing ones?
19. Thank you for filling out our survey! If we have managed to pique your interest please choose from the following options
 - (a) Do you want a summary of this survey when completed?
 - (b) Do you want to join the protest project announcement list?
 - (c) Do you want us to contact you when the first tools are released?
20. Please give us your contact details
21. What is your role in the organisation?
 - (a) Manager
 - (b) Project manager
 - (c) Software developer
 - (d) Tester
 - (e) Researcher
 - (f) Other

References

- [Arts et al.(2006)] Thomas Arts, John Hughes, Joakim Johansson, and Ulf Wiger. Testing telecoms software with quviq quickcheck. In *ERLANG '06: Proceedings of the 2006 ACM SIGPLAN workshop on Erlang*, pages 2–10, New York, NY, USA, 2006. ACM. ISBN 1-59593-490-1. doi: <http://doi.acm.org/10.1145/1159789.1159792>.
- [Barnett et al.(2003)] Mike Barnett, Wolfgang Grieskamp, Lev Nachmanson, Wolfram Schulte, Nikolai Tillmann, and Margus Veanes. Model-based testing with asml.net. In *1st European Conference on Model-Driven Software Engineering*, December 2003.
- [Blom and Jonsson(2003)] Johan Blom and Bengt Jonsson. Automated test generation for industrial erlang applications. In *ERLANG '03: Proceedings of the 2003 ACM SIGPLAN workshop on Erlang*, pages 8–14, New York, NY, USA, 2003. ACM. ISBN 1-58113-772-9. doi: <http://doi.acm.org/10.1145/940880.940882>.
- [Carlsson and Rémond(2006)] Richard Carlsson and Mickaël Rémond. Eunit: a lightweight unit testing framework for erlang. In *ERLANG '06: Proceedings of the 2006 ACM SIGPLAN workshop on Erlang*, pages 1–1, New York, NY, USA, 2006. ACM. ISBN 1-59593-490-1. doi: <http://doi.acm.org/10.1145/1159789.1159791>.
- [Fredlund and Svensson(2007)] Lars-AAke Fredlund and Hans Svensson. Mcerlang: a model checker for a distributed functional programming language. In *ICFP '07: Proceedings of the 2007 ACM SIGPLAN international conference on Functional programming*, pages 125–136, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-815-2. doi: <http://doi.acm.org/10.1145/1291151.1291171>.
- [Hartman et al.(2004)] A. Hartman and K. Nagin. The agedis tools for model based testing. *SIGSOFT Softw. Eng. Notes*, 29(4):129–132, 2004. ISSN 0163-5948. doi: <http://doi.acm.org/10.1145/1013886.1007529>.
- [Hartmann et al.(2000)] Jean Hartmann, Claudio Imoberdorf, and Michael Meisinger. Uml-based integration testing. In *ISSTA '00: Proceedings of the 2000 ACM SIGSOFT international symposium on Software testing and analysis*, pages 60–70, New York, NY, USA, 2000. ACM. ISBN 1-58113-266-2. doi: <http://doi.acm.org/10.1145/347324.348872>.
- [Huima(2007)] Antti Huima. Implementing conformiq qtronic. In *TestCom/FATES*, pages 1–12, 2007.
- [Li and Thompson(2008)] Huiqing Li and Simon Thompson. Tool support for refactoring functional programs. In *PEPM '08: Proceedings of the 2008 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, pages 199–203, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-977-7. doi: <http://doi.acm.org/10.1145/1328408.1328437>.
- [Lindahl and Sagonas(2004)] Tobias Lindahl and Konstantinos Sagonas. Detecting software defects in telecom applications through lightweight static analysis: A war story. In Chin Wei-Ngan, editor, *Programming Languages and Systems: Proceedings of the Second Asian Symposium (APLAS'04)*, volume 3302 of *LNCS*, pages 91–106. Springer, November 2004.
- [Lövei et al.(2007)] László Lövei, Zoltán Horváth, Tamás Kozsik, and Roland Király. Introducing records by refactoring. In *Proceedings of the 2007 SIGPLAN Erlang Workshop*, pages 18–28, Freiburg, Germany, Oct 2007.
- [Riehle(2008)] Dirk Riehle. Junit 3.8 documented using collaborations. *SIGSOFT Softw. Eng. Notes*, 33(2):1–28, 2008. ISSN 0163-5948. doi: <http://doi.acm.org/10.1145/1350802.1350812>.
- [Silva et al.(2008)] José L. Silva, José Creissac Campos, and Ana C. R. Paiva. Model-based user interface testing with spec explorer and concurtasktrees. *Electron. Notes Theor. Comput. Sci.*, 208:77–93, 2008. ISSN 1571-0661. doi: <http://dx.doi.org/10.1016/j.entcs.2008.03.108>.
- [Sinha et al.(2006)] Avik Sinha and Carol Smidts. Hottest: A model-based test design technique for enhanced testing of domain-specific applications. *ACM Trans. Softw. Eng. Methodol.*, 15(3):242–278, 2006. ISSN 1049-331X. doi: <http://doi.acm.org/10.1145/1151695.1151697>.
- [Wiger et al.(2002)] Ulf Wiger, Gösta Ask, and Kent Boortz. World-class product certification using erlang. In *ERLANG '02: Proceedings of the 2002 ACM SIGPLAN workshop on Erlang*, pages 24–33, New York, NY, USA, 2002. ACM. ISBN 1-58113-592-0. doi: <http://doi.acm.org/10.1145/592849.592853>.
- [1] Trapexit <http://www.trapexit.org> Jun 2008
 - [2] Erlang <http://www.erlang.org> Jun 2008
 - [3] ProTest, property based testing <http://www.protest-project.eu> Jun 2008
 - [4] Tsung <http://tsung.erlang-projects.org> Jun 2008
 - [5] Sinan <http://code.google.com/p/sinan> Jun 2008
 - [6] Faxien <http://code.google.com/p/faxien> Jun 2008