

Test Driven Development In Erlang/OTP

Martin Carlson
IT University of Göteborg
Forskningsgngen 6
Hus Patricia
417 56
Göteborg
Sweden
martin@erlang-
consulting.com

Examiner:
Thomas Arts
IT University of Göteborg

Supervisors:
Francesco Cesarini
Erlang Training & Consulting
Ltd
Marcus Taylor
Erlang Training & Consulting
Ltd

Categories and Subject Descriptors

?? [Software Engineering]: Methodology—*performance measures*

ABSTRACT

Test Driven Development has been around for quite some time, but it has recently re-emerged through the introduction of Agile methods such as Extreme Programming and Internet Speed Programming. Test Driven Development has been proven to reduce faults in software written in Java and C++, but there are no reasons to believe these results are language specific. A case study was conducted during the development of an Erlang/OTP prototype comparing Test Last and Test Driven Development. It was concluded that Erlang/OTP based Test Driven Development improved the quality of the code by reducing faults and increasing overall productivity.

Keywords

Test Driven Development, Erlang/OTP, test methodologies

1. INTRODUCTION

Lately, agile methods [6] such as Extreme Programming [3] and Internet Speed Development [2] have been given much attention as a *more or less* revolutionary way of constructing software, introducing techniques such as pair programming and Test First development. Test First development, also known as Test Driven Development [10], however, is not new; NASA used it in the early 1960s [5].

Test Driven Development is a process where, given a requirement, tests are devised and implemented before the system itself. The system is considered completed only when all tests succeed. Set in a software context, developers iterate in-between testing and implementing the system.

1.1 Can Test Driven Development Increase output of correct Erlang code?

Test Driven Development has been successfully used in software development projects, resulting in fault reductions of up to 40 percent. Research using C++ and Java, presented in [10] and [8], confirm these measurements. There is, however, nothing suggesting that increase in quality is language specific.

Since Erlang/OTP [1] was introduced, the main method of development has been design, implement and test, where testing either is either conducted as a parallel process or after the implementation [9]. This closely reflects the organisation of projects, where separate teams design, implement and test the software constructed.

This paper describes the construction of a prototype system built in Erlang/OTP. Test First development was used to construct the first half of the system and traditional Test Last was used for the remaining half. This provided data allowing us to conclude whether Erlang/OTP based Test Driven Development can increase the productivity of a programmer while maintaining a 40 percent fault reduction.

1.2 Testing

Constructing software that can run without crashing is a major challenge for the software industry [7]. Mili et al. propose three essential ways of dealing with such faults:

1. **Fault Avoidance.** Try to avoid faults from entering the software system in the first place. This is achieved through heavy use detailed design documents, reviews and other related activities.
2. **Fault Elimination.** Uses analytical methods such as formal verification, validation and testing.
3. **Fault Tolerance.** Handles recovery and unspecified states in the run-time environment.

Absolute fault avoidance may not be feasible or economic. There are, however, many proposals on the software processes level that claim to contribute to an increased quality when it comes to the production of fault tolerant software. They include IEEE Std 1028-1997 and Formal Methods [4].

We concentrate on Test Driven development, which falls under the category "Fault Elimination". The prototype was developed in a setting where fault tolerant design was complemented by fault elimination achieved through automated testing.

1.2.1 Test Driven Development

A "unit" (as defined in the software testing literature) is the smallest piece of code that can be tested in isolation. We interpret unit-testing as a test of an Erlang/OTP function.

Given a functional requirement, Test First developers create a design, construct unit-tests and start an iteration between test and development. When all tests pass, the implementation phase is considered completed. This is the basic idea behind Test Driven Development. Research conducted suggests that Test Driven Development gives a number of advantages over traditional test-last development:

1. **Efficiency.** When developing software, unit-tests provide instant feedback. This suggests a highly iterative process of code and test, where the developer efficiently locates faults.
2. **Goals.** Tests serve as milestones. When the test execute correctly, the code is completed. Beck [3, p.41] suggests this has a positive psychological effect on developers.
3. **Test Asserts.** When new code is added, all tests must execute correctly, ensuring new code does not introduce new faults. Refactoring efforts benefit from this, as tests ensures nothing else has been broken by the introduced changes.

2. METHOD

To see if Test Driven Development increases productivity of Erlang/OTP programmers while reducing faults, a case study was conducted. A prototype of a track and trace system was developed. Half of the modules were implemented using the classic design, implement and test methodology. The other half was written using Test Driven Development

2.1 Prototype

The prototype is based on three tiers, the database storage tier, business logic tier and a web based graphical user interface tier. The case study was conducted building the business logic tier, where six modules (database interface, production, certification, tracing and serial) encompassed all the functionality.

2.2 Modules

Size and complexity varied between the modules. In the case study, we divided the modules by complexity, so both development methodologies were equally balanced. Tests in both development methodologies were conducted using the same tools and coverage (See 2.3).

2.3 Testing

Erlang/OTP has built in constructs for tracing program flow and coverage analysis. These constructs were used to trace the executed lines in the modules, making sure all branches were covered. Test coverage was based on.

3. RESULTS

The time spent developing the code and faults found in each module was recorded.

Three modules have been developed according to the Test Driven Development method. These modules include:

1. **database.erl**, is a module which implements a database interface. It is implemented as a `gen_server` behaviour, providing a database independent API.
2. **certification.erl** is a module handling certification information about parties such as producers, refiners and resellers.
3. **cert_dev.erl** is a support module for the certification module, providing a higher abstraction level for database query handling.

The other modules developed using the iterative Test Last methodology include:

1. **production.erl**, a module handling all registration and issuing of product identifiers within the system.
2. **verification.erl** is a module providing tracing and data-warehouse functionality.
3. **serial.erl** is a support module for product identification handling such as encoding and decoding.

A typical test for the first three modules is described in Appendix A. They are similar to the tests applied on the last three modules. The only difference is the time in which the tests have been devised and executed.

For the modules developed, we defined a fault to be a failure of one of the predefined test cases. If a fault was detected, the underlying error was analysed and repaired. In all cases, the code passed all test cases immediately after they were executed the second time. The table shows how many faults were detected when executing the test cases. The same table also lists the number of hours spent implementing and testing (fault reparation included) the number of lines in the individual modules.

Module	Faults	Time (H)	LOC
database.erl	3	3	143
certification.erl	4	4	105
cert_dev.erl	2	2	64
production.erl	5	5	84
verification.erl	2	5	37
serial.erl	2	2	24

4. DISCUSSION

While there was no fault reduction on a module level, faults per line were reduced significantly. The increase in productivity was 37 percent when compared to the Test Last modules.

There are some facts which affected the outcome in unforeseen ways.

1. Gain in knowledge. While implementing the modules in the prototype, we noticed that the modules had similar constructs. Knowledge gained implementing the first module reduced the faults introduced in the subsequent modules. This theory is supported by the decreasing number of faults found in the modules (certification, production and verification).
2. After implementing the certification module, a new architecture was introduced, significantly reducing code needed to implement the production and verification modules.
3. The six modules do not comprise a large amount of data. It would be beneficial to extend this study to large-scale industrial projects with many developers.

4.1 Hidden Faults

There exist faults which appear in the form of runtime errors during execution. These faults are difficult to detect due to supervisors which restart the crashed processes. These faults are hard to detect when testing using test cases. It is therefore important to keep verbose logs on all process activities and monitor all error reports produced. The prototype has now been running for weeks, demoed many times, with no such faults having been discovered.

4.2 Refactoring

Refactoring is a crucial part of projects developed with an agile methodology. Although not measured in this study, we noticed an increased efficiency while refactoring Test Driven Development modules. The test-repository combined with an iterative test and code process was found highly effective introducing the changes in the system.

4.3 Time Used

Although the modules implemented only took 18 hours to implement, much time was used to implement the other tiers in the prototype. These parts are however, not part of the case study. This is because a different testing approach was used for these parts of the system.

5. CONCLUSION

Our case study shows that Test Driven Development increase the output of correct code compared to traditional test-last development. We also conclude that Test Driven Development reduces the faults per line of code.

6. REFERENCES

- [1] J. Armstrong, R. Virding, and M. Williams. *Concurrent Programming in ERLANG*. Prentice-Hall, New York, 1993.
- [2] R. Baskerville, B. Ramesh, L. Levine, J. Pries-Heje, and S. Slaughter. Is Internet-speed software development different? *IEEE Software*, 20(6):70–77, Nov./Dec. 2003.
- [3] K. Beck. *Extreme programming explained: embrace change*. Addison-Wesley Longman Publishing Co., Inc., 2000.
- [4] R. John. Formal methods and digital systems validation for airborne systems. Technical report, 2003.
- [5] C. Larman and V. R. Basili. Cover feature: Iterative and incremental development: A brief history. *Computer*, 36(6):47–56, June 2003.
- [6] R. C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, 2003.
- [7] A. Mili, B. Cukic, T. Xia, and R. B. Ayed. Combining fault avoidance, fault removal and fault tolerance: An integrated model. In *ASE '99: Proceedings of the 14th IEEE International Conference on Automated Software Engineering*, page 137, Washington, DC, USA, 1999. IEEE Computer Society.
- [8] J. U. Pipka. Test-driven web application development in java. In *NODE '02: Revised Papers from the International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World*, pages 378–393, London, UK, 2003. Springer-Verlag.
- [9] C. Williams. Evaluation: Two test projects and a new method of testing systems written using erlang/otp. Technical report, 2000.
- [10] L. Williams, E. M. Maximilien, and M. Vouk. Test-driven development as a defect-reduction practice. In *ISSRE '03: Proceedings of the 14th International Symposium on Software Reliability Engineering*, page 34. IEEE Computer Society, 2003.

APPENDIX

A. TEST CASE

```
%% creating a new company, adding a site and adding a certificate
new_certificate(suite) ->
  [];
new_certificate(doc) ->
  ["creating a new company, adding a site and a certificate"];
new_certificate(Conf) when is_list(Conf) ->
  ok =
  certification:new_company("01",
                            "ChickenFarm",
                            "1 Chicken St",
                            "Farmer"),
  ok = certification:new_site("0101",
                              "01",
                              "Farm A",
                              "1 Chicken St",
                              "Farmer",
                              "Farm"),
  ok = certification:new_certificate("010101",
                                     "0101",
                                     63282124800,
                                     63284803200,
                                     "Certifyer",
                                     "Farm",
                                     2000).
```