

Data Sensitivity and Execution Time Problems in Agile Developer Testing

Hannu Törnroos

Abstract— The highly automated developer testing is considered as Extreme Programming (XP) methods main tool to tackle quality risk cause by frequent changes in requirements. Anyhow this approach inflicts on new kind of problems with the creation of test situations and time used in executing test regularly. These problems are hard to identify in the beginning of the software project and still they may cause remarkable losses during project. This paper presents these problems in a light of the data collected from scientific literature and compares possible solutions to these problems.

1. INTRODUCTION

In his article Martin Fowler (Fowler 2000) present that unpredictability of the software requirements leads to problems in disciplined software processes, which trust in predictability of software development and plan development in long term beforehand. The new software process models to which Fowler refers as agile methodologies are trying to respond to constant changes in requirements by shortening the development cycle.

Kent Beck (Beck 1999) states that evolution of process models can be seen in three steps:

1. Waterfall model
2. Iterative processes
3. XP (agile methods)

Where each step has decreased the size of planned development cycle.

In his article Beck describes how software projects can adapt to inevitable and constant changes using the software process called eXtreme Programming (XP). According (Fowler 2000) XP is the one agile methodology that has got most attention.

When methodology like XP constantly accepts changes to requirements, it has to offer some way to make sure that software works as whole after the changes to certain part of it.

According Beck's article it is clear that XP's main tool to tackle problems caused by constant changes in produced software is highly automated unit and regression testing. Beck states, "if there is a technique at the heart of XP, it is unit testing".

Still it seems that while automated testing helps developers to maintain software quality in reasonable level, they might cause new problems. For example (Elssamadisy and Schalliol 2002) present problems with cost and quality of XP testing. Also (Smith, Meszaros 2001) make same kind of notions

about unwanted results of slow unit test execution and expensive test development of XP testing.

In addition my own experiences in work as test manager have shown me that test automation in environment where test data is strongly time dependent is troublesome task. Meszaros mentions this problem in his article (Meszaros 2003) and uses term data sensitivity problem for it.

Most sources, which describe use of agile testing methods, concentrate on presenting the advantages of these methodologies while they do not pay so much attention to disadvantages. Luckily criticism is raising its head and the problems like the ones presented (Elssamadisy and Schalliol 2002), (Smith and Meszaros 2001) and (Meszaros 2003) are revealed.

At developer level testing above-mentioned problems are closely coupled and solutions to one of them has effect on the other. In this paper I am concentrating on two of the above-mentioned problems. I look answer to following questions:

- According the scientific literature what kind of ways there is to solve the data sensitivity problem (presented in (Meszaros 2003) and test run time problem presented in (Smith and Meszaros 2001)?
- What tradeoffs these solutions have and how the solutions to them are related?

Articles, which present the problems, also offer some solutions to problems they state. Usually offered solutions superficiality of comparison to other solutions and total lack of analysis of negative consequences of offered solution. In this literature study I am trying to gather additional data from scientific literature, present solutions presented in above articles and other applicable articles in a form that would give readers change to compare them and estimate each solution's suitability to their own needs.

The next section presents the data sensitivity problem and possible solutions to it. Section 3 presents test run time problem and its solutions. In section 4 I link studied problems together, summarize the gathered data and discuss about research/business possibilities related to researched problems.

2. DATA SENSITIVITY

In his article (Meszaros 2003) presents problem related to test data. He names problem as *data sensitivity*. He states that all tests assume some starting point; these are often called the "pre-conditions" or "before picture" of the test. In information systems, this is defined in terms of data that is already in the system. If the starting point (i.e., the database contents) changes, the tests may fail unless great effort has been

expended to make the test insensitive to the data being used.

2.1 A standard solution to data sensitivity: Xunit testing framework

As stated in introduction XP encourages its practitioners to automate testing. In XP projects testing is automated using some testing framework, most commonly Xunit framework see (Beck and Gamma 1998,1), (Beck and Gamma 1998,2) for details. Xunit frameworks have “in-build” solution to data sensitivity problem. Xunit framework groups test together in Test Suites. Each test of Test Suite has to make sure that it starts from certain starting point (test preconditions) and after the execution clean up (teardown) the situation to enable next test in test suite set-up itself correctly. Xunit framework offers common interface to handle these tasks semi automatically. Semi automatically means here that developer has to write the actual pre- and post conditions in form of special set-up- and teardown methods, then framework takes care that these methods are called before (setup) and after (teardown) test execution. (Beck and Gamma 1998,2) describes the use of these functions in more detail.

2.2 Problems with Xunit frameworks approach

XUnit’s solution has some drawbacks, which may not be visible in beginning of the project and become apparent during the project. (Elssamadisy and Schalliol 2002) have recognized a problem related to Xunit’s use of setup and teardown methods. They report that many times in setting up proper test cases you need a business object or group of business objects at a specific point of their lifetime with a specific state to write a correct test. This is very recurrent need in all projects with complex business objects. The bad smell¹ is when the you find yourself writing very large amounts of setup code before you actually do the test, or even worse, not writing tests because setup is too extensive or not well understood.

Next I will present possible solutions to data sensitivity problem in agile development environment.

2.3 Possible solutions to XUnit setup/teardown problems

2.3.1 Mock Objects

(Elssamadisy and Schalliol 2002) propose two different solutions to presented problem. The first is the creation of fixtures that will return objects in different states as described in (Mackinnon et al., 2000).

Mackinnon et al.’s idea behind Mock Objects is simple, Mock Object is a substitute implementation to emulate or instrument other domain code. It should be simpler than the real code, not duplicate its implementation, and allow you to set up private state of objects to aid in testing. According them there is a risk that Mock objects become too complex. They have found that warning sign of this is that it (Mock Object) starts to call other Mock Objects.

Mackinnon et al. present numerous reasons why developers should use Mock Objects, including.

- Mock Objects localise unit tests
- Mock Objects have beneficial effects on the coding style of development team.

Anyhow these advantages are interpretations of Mackinnon et al. And they do not offer any scientifically valid proof for claimed advantages.

Use of Mock Objects have certain limitations and Mackinnon et al. mention following:

- Mock Objects help only in unit testing
- Mock Object code might contain errors.
- In some cases the creation of Mock Objects to present types in complex external library is hard.
- In statically typed languages, libraries must define their APIs in terms of interfaces rather than classes so that clients of the library can use such techniques.

There are several tools to facilitate Mock Object use in development work. (Moore and Palmer, 2002) compare these tools and they state that according their experience the automatic generations mocks saves time and produces simple, consistent and predictable Mock Objects.

2.3.2 ObjectMother

Another solution, which reduces the complexity and amount of setup and teardown code, is to reuse the Factory design pattern as described in (Schuh and Punke 2001).

Their ObjectMother pattern has similar approach to solving the problem as Mock Objects. ObjectMother provides developers with a fabrication plant for testable business objects.

ObjectMother approach differs from Mock Objects by two significant ways:

- The purpose of the pattern is to generate business objects that resemble as closely as possible actual objects that will exist in production. (Schuh, Punke, 2001) Where as Mock Objects should be simpler than the real code. (Mackinnon et al., 2000)
- ObjectMother manages the whole lifecycle of test objects, including creation, customisation and when necessary, deletion. Mock Objects do not take up stand on these matters.

(Schuh and Punke 2001) claims that three major benefits, which ObjectMother offers are:

1. Simplified and standardized test object creation,
2. ease of maintenance, because test object creation is entrusted to a specific class or group of classes, and
3. test object clean-up.

As with Mock Objects case, writers do not offer any empirical evidence to backup their ideas. This decreases the scientific value of their document.

¹ Metaphor introduced by Kent Beck to describe early warning signal.

2.3.3 Comparison between MockObjects and ObjectMother

As always with this kind of comparisons the answer is “it depends”.

Interesting point is that Elssamady and Schalliol state that Mock Object solution loses its applicability when the objects become very complex. Anyhow they don't offer any concrete evidence for their claim.

Schuh and Punke claim that the closer the tests data is to real data, the better the unit tests will be able to test for real problems that may surface in application. This might be true, but on my mind there is a trade-off here, creation of test objects that resemble actual domain objects as closely as possible (the ObjectMother's approach), is likely to be more expensive and more error prone than creation of simpler mock objects.

Another important point is the maintainability of solution that technique offers. According (Schuh and Punke 2001) ObjectMothers approach the solution most likely leads to easily maintainable test data.

(Mackinnon et al. 2000) does not provide specific description, which would help in assessing the maintainability of the offered solution. Mackinnon et al. refer to maintainability issues when they mention the common format of unit test when using Mock Objects. Anyhow they don't describe this format in such detail that would make critical comparison possible.

Above comparisons are based on my assumptions and interpretations of literature about these two methods. Thus it would be nice to see empirical evidence from use of both techniques, which would enable scientific comparison and above all give some reliable information whether or not techniques solve the actual problem.

3. SLOW TEST EXECUTION PROBLEM

In their article (Smith and Meszaros 2001) give a clear presentation of developer test runtime problem.

Article underlines importance frequently executed automated test suite. They also state that it is essential for test to run very quickly. Article reports that in practice test execution is normally too slow in developer testing especially if relational databases are involved in system to be developed.

Writers give following present following causes for slow test execution:

- In XP style development developer is expected to run great number of tests accessing the database after each change to code.
- Test accessing backend database suffer from latency in queries and updates caused by disk I/O

Smith and Meszaros argument importance of their findings by presenting:

- economical reasons and
- work ordering reasons,

why slow developer test execution is harmful for the

development.

The simple economical example they present gives clear picture of how small increase in runtime of test execution time is multiplied to considerable time waste because single developer executes tests frequently.

In work ordering they present that long test execution times cause developers to bundle up small changes and test them together. They also state that on extreme cases where test run times are so long that developer leaves the desk while test suite runs, her train of thought may be broken.

Authors impose that above mentioned reasons cause following negative results:

- Debugging the erroneous code becomes more difficult because identifying guilty change is more difficult
- Developers may forget to test some changes
- Fragmented work rhythm becomes recurring pattern of work.

3.1 Possible solutions to test runtime problem

The authors offer three separate solutions to the problem:

- Faster execution environment
- Running fewer tests
- In-Memory testing

Smith and Meszaros present that improvements achieved by altering test execution environment are too small to be beneficial. Thus I will concentrate on two later.

Both of the later are very effective and interesting solutions. (Smith and Meszaros 2001) concentrates totally on in-memory testing and scratches only the surface, when it handles about running fewer tests. Authors note that test selection is the problematic part i.e. which test should be run after specific change. They admit that there is value in this technique and reference to JUnit Cookbook (Beck and Gamma 1998,2) which offers some guidance organizing test in smaller subsets and state that developers have to anyway select the test to be run somehow.

Next I will shortly present Smith's and Meszaros's idea about in-memory testing then I will present some ideas selective regression testing which is quite widely researched subject on the field of traditional testing.

3.1.1 In-memory testing

Smith and Meszaros state that during frequently repeated test runs, the test should avoid querying and updating a database. In their solution this is achieved by replacing the actual database with a simple in-memory “object database”. To obtain rapid feedback from test suites developers run against test against in-memory database during coding. Anyhow they state that before merging the changes to code baseline the developer must run the test against the real backend database. Thus in-memory tests do not replace the developer testing against backend database, but offers faster alternative during coding phase.

Their solution requires quite a lot from developed systems architecture, article list following requirements which system must fulfill to facilitate in-memory testing:

- Database access must be eliminated from the business logic.
- Test environment must offer developers easy way to choose which database (in-memory/backend) they want to use in testing.
- Test environment must be able to mimic the functional features of backend database i.e. the features like stored procedures, triggers and sequences.

Authors present that by using in-memory testing they have reduced significantly the run times of extensive test suites. One example they present is case where test suite containing 53 tests. Executing this suite in memory takes 10 seconds while running this same suite against backend relational database takes about 10 minutes.

Anyhow Smiths and Meszaros's article lacks concrete empirical evidence of advantages offered solution. Other critical issue is that writers do not estimate at all how much time and money the building of the functional framework, which facilitates the in-memory testing, has taken. This kind estimation would be beneficial in evaluation of total cost-effectiveness of the solution.

The in-memory testing solution is most likely effective technique to tackle the slow test execution problem. Anyhow it requires quite a lot from systems architecture to be functional. So there is need for other kind of solution also. As mentioned earlier selective regression testing has been researched quite a lot by researchers of traditional testing era. Next I will present how selective retesting approach could be used to solve slow test execution time problems and whether or not the it's use is feasible.

3.1.2 Selective regression testing

As (Smith and Meszaros 2001) state the (Beck and Gamma 1998,2) offers some guidance how to organize tests in subsets that can be run. Beck's and Gamma's examples are very simple and do not take any stand how the actual selection of which tests/test suites should be run after certain changes are made to code. When compared to method presented e.g. (Harrold et al. 2001) its clearly visible that (Smiths and Meszaros 2001) overlook the issue of reducing the size of test suite to be run.

Before presenting (Harrold et al. 2001) in detail I will present background information about selective regression testing and some common research results gained from selective regression testing.

In scientific traditional testing resources, test automation is mainly related to regression testing. (Li and Wahl 1999) gather regression testing knowledge from many different sources and present it in easily comparable form.

Their article presents and compares selective regression testing techniques. Here selective means that the main goal of

presented method is to minimize the size of run test set and in same time maximize test set's capability in finding bugs. (Li and Wahl 1999) consider retest-all strategy to be wasteful in minor changes to a system. They also address the main problem with selective regression testing following way: "The challenge is to identify the dependencies between test case and the program entities it covers. Determining such dependency information requires sophisticated analyses of both the source code and the execution behaviour of the test cases." The (Harrold et al. 2001) gives following presentation of general regression test selection system:

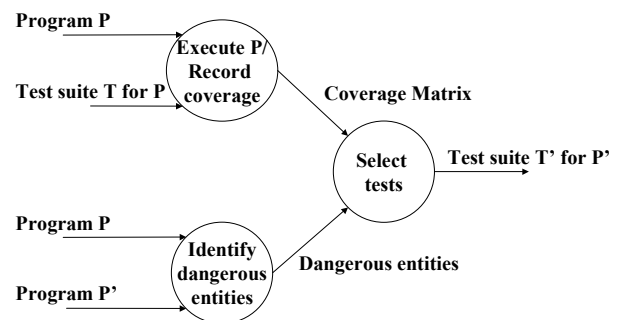


Figure 1: A general system for regression test selection (Harrold et al. 2001)

Where P is given program, T is test suite consisting from set of tests. P' is modified version of program P. Now common regression-test-selection system is trying to find smallest possible test suite T' which is a subset of T. This is done by:

1. Executing program P using Test suite T as an input
2. Collecting coverage information from test run (which methods/statements were executed using test suite T)
3. Compare P and P' to find changed entities
4. Using data collected in steps 2 and 3 select tests from T using certain selection algorithm to form smallest safe test suite T'

In summary of their article Li and Wahl debate about the cost-effectiveness of different regression testing techniques. They state that selective approach usually can reduce the cost of retesting compared to the retest-all strategy. Anyhow they also state that selective approach might not be cost-effective if the cost of test selection exceeds the cost of executing the extra test cases used by the retest-all strategy. (Li and Wahl 1999) also compare the models developed to facilitate the comparison of the cost effectiveness between the selective and retest-all strategy.

Many papers, e.g. (Harrold et al. 2001) and (Rothermel et al. 2000), offer concrete examples of using selective techniques in practice. Next I will present an overview to Java solution presented in (Harrold et al. 2001). The presentation of details of technique is out of scope of this paper, thus I will

concentrate on presenting the assumptions that technique relies on and claimed advantages and disadvantages of the technique because at the level in which my paper handles these matters, this more common information, is more useful for the readers than detailed analysis of techniques implementation.

(Harrold et al. 2001) presents the first safe regression-test – selection technique that based on suitable presentation, handles the features of the Java language. Article also states that unlike other safe regression test selection technique, the presented technique is able to handle incomplete programs. Article also describes the RETEST; a regression-test-selection system that implements presented technique and a set of empirical studies achieved using RETEST system in test selection of for software subjects.

(Harrold et al. 2001) states that regression-test-selection technique is particularly effective in situation where changed software is tested frequently. Article mentions two example environments where technique could be used:

- Environment where nightly builds of the software are performed and a test suite is run on the newly build version of the software.
- Environment where developers run test during coding. After developers modify their software, they can use regression test selector to select subset of the test suite to use in testing.

Authors present main benefits of the approach to be:

- In many cases only a small subset of the test suite is ran, which reduces the time required to perform the testing.
- Technique may also be effective when the cost of test cases is high.

According (Harrold et al. 2001) the presented technique relies on following assumptions about the code under test, the execution environment, and the test cases in the test suite for the original program.

- Reflection; techniques assumes that the program does not programmatically access the information about the fields, methods and constructors of loaded internal² classes.
- Independent external² classes; Technique assumes that the external classes can be compiled without the internal classes, and that external classes do not load any internal class explicitly.
- Deterministic test runs; Technique assumes that a test case covers the same set of statements, and produces the same output, each time it is run on an unmodified program.

The results of the empirical evaluation of technique presented in (Harrold et al. 2001) offer interesting information about techniques efficiency and applicability to specific situations.

Harrold et al. made two separate empirical studies using RETEST. In first study concerned the test suite reduction, In practice they researched how much their technique could reduce the number of test cases to be executed against certain version of software subject. Second study concentrated on test selection granularity, i.e. they tried to determine whether any additional reduction in the size of selected test suite could be achieved by selecting dangerous entities on finer granularity.

From these the former is more interesting for this papers perspective. The studies result shows that algorithm select from 2% to 100% of test cases to new reduced test suite. The authors do not give any average numbers, which makes the evaluation of the studies results bit hard.

(Rothermel and Harrold 1997) gives more detailed information about efficiency and economy of this kind of technique. Empiric study is conducted on C programs using similar technique as presented in (Harrold et al. 2001). Rothermel and Harrold conducted three separate studies:

1. Regression Test Selection
2. Regression Test Selection on a Larger Scale
3. Regression Test Selection on Commercial Software

In each of these studies authors measured the size of selected test suite and test execution time of both retest-all and selective-regression-test approach.

In study 1 the average size of the minimized test suite was 63,6% of the original test suite and the test execution time was reduced 41%. In study 2 numbers were following, the selected test suite was 5% from original and execution time was reduced 82%. On study 3 only savings on test suite size were measured and they were 67%.

In study 1 (Rothermel and Harrold 1997) used modified versions of software, which contained erroneous modifications, and thus they were able to make observations about the fault-revealing capabilities of used technique. In two cases which authors report used test selection technique was able to select all tests that revealed fault in the examined version.

(Harrold et al. 2001) claim that success of the code-based regression-test-selection techniques depends not only on the magnitude of changes to modified software – which, in turn, depends on the frequency of regression testing- but also on the location of those changes and characteristics of the software. Also (Rothermel and Harrold 1997) and (Rothermell et al. 2000) make same kind notions.

(Rothermell et al. 2000) also states that regression test selection technique presented in their paper (which is very similar to Java technique presented in (Harrold et al. 2001) is strictly code based, and thus fulfils the need to employ a white box test case selection technique. According (Rothermell et al. 2000) to achieve adequate confidence in modified software, however, presented technique should be used in conjunction with a black box selection technique that selects test cases relevant to changed specifications. If the specifications for modified software have changed, and the code modifications necessary to implement changed specifications have not been

² (Harrold et al. 2001) refer to set to classes that are analyzed and instrumented by the regression-test-selection system, as internal classes and those that are not analyze and instrumented by the system as external classes.

made, such fault can only be detected by black box test case test selection.

Anyhow there are real problems with this technique especially when you look it from developer testing perspective. (Rothermel and Harrold 1997) report that selecting tests in study 2 (large scale test selection) took 23 minutes and 17 seconds, which is too long time for developer testing purposes. (Rothermel and Harrold 1997) also made conclusion that test selection provides greater opportunities on system test level than in unit test level. They even note that in unit test level test selection may not offer savings sufficient to justify its costs. Anyhow this later notion conclusion conflicts with (Harrold et al. 2001) where authors present developer level testing as one possible target environment for this kind of test selection technique. In addition to this (Rothermel and Harrold 1997) do not give any evidence to backup their conclusion.

As summary for test selection approach I emphasize following matters:

- Selective-regression-test techniques have been under research since 90's
- There is a lot of empirical evidence from efficiency of these techniques.
- Use of these techniques relies on certain assumptions about the code under test, the execution environment, and the test cases in the test suite for the original program.
- The efficiency of these techniques is dependent from magnitude of changes to modify, the location of those changes and characteristics of the software.
- The analysis time of test selection techniques may be too long for developer testing purposes (over 24 minutes for large software)
- The conclusions presented in (Rothermel and Harrold 1997) thrown shadow above use of these techniques in developer testing.
- (Rothermel and Harrold 1997) conclusions suffer from lack of empirical evidence and in (Harrold et al. 2001) Harrold makes totally opposite notions.

3.2 Comparing in-memory and test selection approaches

It is clear that these techniques have totally different approaches to solve same problem. The base situation in their comparison is bit odd, (Smith and Meszaros 2001) claim that In-memory testing approach offers solution to slow test execution times; anyhow they don't offer empirical evidence to backup their claim. On other hand according (Harrold et al. 2001) the selective regression testing approach could be used in execution time problems in developer testing level, but earlier notions presented in (Rothermel and Harrold 1997) state opposite.

Still comparison is reasonable and I will look both methods from following perspectives:

- Approaches prerequisites
- Claimed advantages
- Approaches costs

- Difficulty of the approach

Prerequisites

Both of these techniques have some prerequisites for the software's implementation. On my mind the requirements of the selective regression testing approach are easier to fulfill. In-Memory testing approaches requirements go very deep to software's architectural decisions and thus using In-Memory testing approach might be very expensive if the these architectural decisions are made in way that does not facilitate In-Memory testing.

Claimed advantages

Both of these approaches offer solution to slow test execution times.

In-Memory testing tries to minimize the execution time of individual tests by decreasing the file I/O during test and that way decrease the over all run time of test set. , (Smith and Meszaros 2001) state that run times are reduced by order of magnitudes.

Selective regression testing tries to decrease the size of test suite to be ran after each change by selecting only those tests which execute the changed from test suite. (Rothermel and Harrold 1997) presents that use of selection technique can reduce the size of regression test suite from 41% to 82% depending from the application under the test. The gain in execution time is most likely something between above percentage numbers.

So the numbers seem to favour the In-Memory testing

Difficulty of the approach

From this perspective the In-Memory testing approach is more feasible. The theory behind selective regression testing approach is somewhat difficult. It is hard to see that any project it self would have enough knowledge, time and money to implement system which would enable use of test selection in developer testing level.

Instead the idea behind In-Memory testing is fairly simple and its implementation is quite easy from basis of (Smith and Meszaros 2001)

Cost of approach

Because any of related references do not present explicit numbers about costs of implementing the solution they are proposing, it is hard to compare them from "start up" cost point of view. Below I am stating my own opinion about the magnitude of costs from software engineering point of view and presenting some ways that would make the costs more reasonable.

As the difficulty section already stated, the costs of selective regression testing approach are beyond the scope of the individual project, which main target is implement some software system. Actually it might stand for In-Memory testing approach also. It might be too costly to implement In-Memory testing for purposes of one project only.

One way to improve the return of investment would be to

implement a framework solution of either approach. This framework solution could then be used by many other projects

Day-to-day costs are different thing. From this perspective, selective regression testing seems to be more promising. Use of it seems to be free of cost. With the In-Memory testing the functionality on backend database has to be duplicated to code in order make test run correctly. If application relies a lot on database functionality such as stored procedures, triggers and sequences approach might not be feasible.

4. SUMMARY AND DISCUSSION

Scientific literature offers good solutions to both data sensitivity and test runtime problems. Anyhow when selecting the solution to be used in specific project, it would be nice to know the costs of the solutions implementation. The papers presenting these solutions do not offer any estimates how much work solutions implementation would cost.

Data sensitivity

The comparison between solutions to data sensitivity problems ended up on draw. Neither Mock Object nor ObjectMother approach could claim one's advantage over other.

Both of these solutions are from reference, which are presented originally in agile method conference. Both approaches assume that developers write extensive amount of code just for facilitate the automated testing of the actual production code. The real issue is that who writes the code and where this code written to. Both of presented solutions present architectural solution to ever expanding amount test code. Architectural solution increases the complexity of the development environment. On my mind the unneeded complexity is always bad thing, thus use of these solutions should be considered if the amount of time used in writing test setup and teardown code starts to get too big.

Anyway in problem situations, where time used to handle data sensitivity by pure Xunit way is not enough, my gut feeling says that Object Mother approach could be more efficient in solving the problem, but as mentioned earlier there is a need for empirical evidence about advantages/disadvantages of both of them.

Slow test execution problem

Here the comparison between possible solutions is bit uneven. Selective regression testing approach has been researched much more than In-Memory testing.

The empirical evidence does not favor selective regression testing but on the other hand we do not have any empirical evidence from use of In-Memory testing.

So even In-Memory testing seems to offer advantages, which are orders of magnitude greater than the advantages of Selective regression testing we can not be sure whether or not these advantages will realize all projects.

This is critical issue specially because In-Memory testing

approaches implementation does not build by itself; it requires architectural selections and projects conscious decision to write framework code solely for testing purposes.

Anyhow the promises for In-Memory testing are so much bigger that in real life I would select it despite the lack of empirical evidence.

On my mind the Selective regression testing approach would become real option only if the solution would integrate tightly to development environment. (Nagappan et al. 2003) present interesting way of extending Eclipse development environment with plug-in, which helps developers in their testing tasks. If selective regression testing would belong to integrated development environments tool set, its use would be easier and thus more feasible.

Data sensitivity and slow test execution time together

It is clear that both of these problems are present at the same time. Thus project solving one of them usually has to solve the other one also.

Both of the problems have nature that they are not apparent in the beginning of the project, thus their solutions have to be selected in advance. Planning these kind of solutions is troublesome task, because made decisions may yield either great savings for the project or in case decisions were wrong they may collapse the economy of the project.

Both solutions to data sensitivity problem are similar to In-Memory testing solution test execution time problem. All of them require architectural selections that back up their implementation and use, thus if for example the use of the ObjectMother is selected as solution to data sensitivity, it may be easier to justify the use In-Memory testing approach to solve slow test execution time.

Actually either of proposed solutions to data sensitivity might be use in solving slow test execution times also. If business object used in testing are created using either Mock Object or ObjectMother approach, the database access in object creation can be omitted if solution is planned correctly. Actually object database described in (Smith and Meszaros 2001) seems to be very close to ObjectMother use presented in (Elssamadisy and Schalliol 2002). Thus it seems that using ObjectMother and In-Memory testing together would be efficient.

The selective regression testing approach works on different level than other proposed solutions, thus it is harder to see it working together with one of others. On the other hand because of this matter, using selective regression testing does not preclude any of these other solutions.

Especially if selective technique would integrate to development environment as stated earlier, its use could use to strengthen the use of other solutions.

REFERENCES

- Beck, K., Gamma, E. ,1998, "Test Infected: Programmers Love Writing Tests"Java Report, vol. 3, no. 7, 1998, pp. 37-50.

- Beck, K., Gamma, E., 1998, JUnit Cookbook, <http://junit.sourceforge.net/doc/cookbook/cookbook.htm> . ? <Modified Date>. 28-04-2004 <Reference Date>
- Beck, K., Embracing Change With Extreme Programming. IEEE Computer, 1999, vol. 32, no 10. pp. 70-77.
- Elssamadisy, A. , Schalliol, G., 2002 "Recognizing and responding to "bad smells" in extreme programming.", Proceedings of the 24rd International Conference on Software Engineering,. pp. 617 - 622.
- Fowler, M. , 2000,"The New Methodology. ", <http://martinfowler.com/articles/newMethodology.html> 01-04-2003 <Modified Date>. 07-03-2004 <Reference Date>
- Harrold, M. J., Jones, J, A, Li, t, Liang ,D 2001,"Regression Test Selection for Java Software", Proceedings of the 16th ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications pp 312 - 326
- Harrold, M. J., 2000,"Testing: a roadmap", Proceeding of the conference on the future of Software engineering pp 61-72
- Li, Y, Wahl, N, 1999," An Overview of Regression Testing", ACM SIGSOFT Software Engineering Notes vol 24 no 1 January 1999 pp 69-73
- Mackinnon, T, Freeman, S, Craig, P. 2000, "Endo-Testing: Unit Testing with Mock Objects", XP examined , Addison-Wesley, Canada
- Meszaros, G., "Agile regression testing using record & playback.", Conference on Object Oriented Programming Systems Languages and Applications, 2003. pp. 353-360.
- Moore, I, Palmer, S, 2002, "Making a Mockery" , presented in XP 2002 3rd International Conference on eXtreme Programming and Flexible Processes in Software Engineering
- Nagappan, N, Williams, L, Vouk, M, 2003, "'Good enough" software reliability estimation plug-in for Eclipse", Proceedings of the 2003 OOPSLA workshop on eclipse technology exchange pp 30-34
- Parrish, A, Jones, J, Brandon, D, 2001," Extreme Unit Testing: Ordering Test Cases to Maximize Early Testing." presented in XP 2001 2nd International Conference on eXtreme Programming and Flexible Processes in Software Engineering
- Rothmel, G, Harrold, M, J, Delhia, J, 2000, "Regression Test Selection for C++ software", Journal of Software Testing, Verification, and Reliability, no. 10, pp 77- 109
- Rothmel, G, Harrold, M, J, 1997, "Experience With Regression Test Selection", Empiric Software Engineering Journal 2(2), 1997, pp 178-187
- Smith, S., Meszaros, G., 2001,"Increasing the Effectiveness of automated Testing" presented in XP 2001 2nd International Conference on eXtreme Programming and Flexible Processes in Software Engineering
- Schuh,P, Punke, S, 2001,"Object Mother: Easing Test Object Creation in XP" , , presented in 2001 XP Universe