Leveraging Global Talent for Effective Test Agility

Todd Little
Halliburton, Houston, TX, USA
tlittle@lgc.com

Joe Hughes
Logigear, San Mateo, CA, USA
joeh@logigear.com

Suzanne Elliott
Halliburton, Houston, TX, USA
selliott@lgc.com

Florin Simion
Simco, Bucharest, Romania
fsimion@simco.ro

***Abstract*—A major challenge in agile development is the ability of test teams to keep pace with ongoing development while simultaneously ensuring that new development has not created regression failures. This case study from Halliburton shows how together with two globally distributed outsourcing partners they developed a comprehensive test automation strategy for their agile teams that effectively leveraged both in house and outsourced activities. This approach resulted in a significant quality improvement from prior releases.**

Keywords: Outsourcing, Agile Development, Test Automation, Distributed Teams, Offshore

I. INTRODUCTION

Over the years many agile proponents have come out strongly against offshoring some of the development team, and in particular against having a remote testing team. We had a corporate mandate to utilize offshore outsourcing and decided we were going to make the best of it. In the end we were pleasantly surprised by the overall results. We made use of not one, but two separate outsourcing providers located in two distant locations. While we had many challenges, what we found was that by starting with an overall testing strategy and an understanding of the strengths and constraints, we were able to optimize the problem globally to achieve outstanding results. In particular, we were able to reduce defects found in customer beta testing by 84% and known customer issues at deployment by 97%.

II. BACKGROUND

Landmark Graphics is a wholly owned subsidiary of Halliburton and is the premier provider of software and technology services for the upstream oil and gas industry. Its software solutions help geoscientists and engineers make highly complex technical and business decisions. The specific product line involved in this case study is DecisionSpace Nexus[1][2], a next generation reservoir simulation software suite which utilizes a finite difference mathematical model to allow companies to accurately model hydrocarbon assets, enabling rapid decisions on high-dollar development scenarios.

Like many other software systems, Nexus is collection of integrated applications. This system of systems comprises multiple millions of lines of source code and provides a complete user experience from data preparation to numerical simulation to 3-dimensional visualization. And while many modules of the Nexus family are next-generation, there are several aspects that are legacy and are complicated further by the need to support both the next-generation simulator as well as the legacy simulator.

Figure 1. The Nexus Software Family

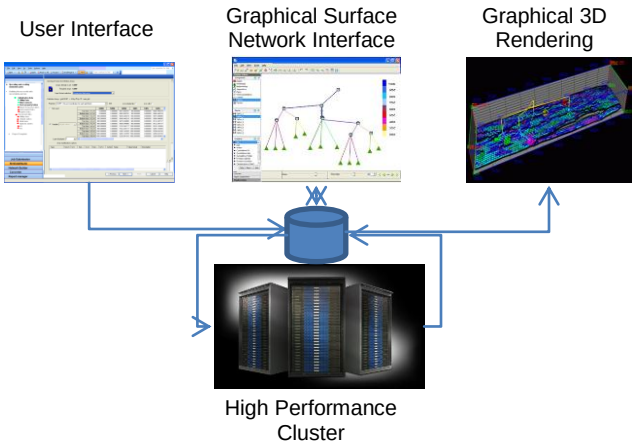


Figure 1 shows a simplified view of the software system and is further described in Table I.

TABLE I.

Component	Language	Started	Summary
Data Studio	C++/MFC	1996	Data preparation, also supports legacy simulator
SurfNet	Java	2009	2D view of wells and production network
NexusView	Java/OpenGL	2004	Enables visualization in 3D of reservoir simulation over time
Nexus	Fortran90 and MPI	2001	Computational engine, often runs on high performance cluster on multiple processors.

III. The Development and Testing Process

The Nexus development team had been doing many things right since they started development in 2001, but had not fully embraced agile development. In 2007 the team moved to a more structured Scrum environment. While not a drastic change for the team, the additional structure seemed to work well and helped identify some of the areas where there were bottlenecks. We will revisit some of the specifics of the agile implementation and team coordination later.

A. Existing Test Approach

One of the most important things that the entire team realized was the importance of having an automated regression test suite which exercised the functionality without going through the graphical user interface. While the team did not have extensive unit tests, they did have a good set of functional tests that provided overall feature coverage. Prior to any commit being finalized the development regression suite was run. In addition to the developer regression suite, the team relied on a customer regression suite, a manual smoke test, and additional exploratory testing. The developer regression suite safety net had paid off well for the team and they had general confidence in their check-ins. However, these developer tests were not finding issues that were showing up in the more complicated customer models. To make matters worse, the big challenge with the customer regression suite was that it took almost a week of computation time on a high-end cluster to determine if the tests passed or failed.

B. Challenges Testing High Performance Computing

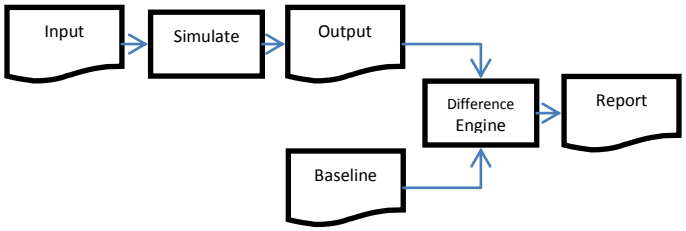
High performance computing software is designed to be able to run on multiple processors, sometimes upwards of 64 or 128 cores. This parallel computing can effectively reduce computation time significantly. The flip side of that is that there is a reason why the software needs to use a lot of processing power – it is doing very complicated scientific calculations that take a lot of computational cycles. While Moore's Law and multicore processors have increased computational power, the complexity that the engineer builds into their models has grown in lockstep. Engineers tend to design their models so that they can get results back with overnight turnaround. Some models like those in our customer suite take several days to complete. One engineer at a customer site joked that the most complex models are measured not in days, but in haircuts.

C. Additional Challenges of Testing Simulation Software

In addition to the challenges of long computation times, reservoir simulation brings the additional challenge that the problem is being solved by approximation techniques. In other words, small perturbations in either data or algorithmic code, or even running on a slightly different processing environment can result in different output results. The development team had been long aware of that issue and had built an intelligent differencing tool to help understand whether generated results were within engineering accuracy of the baselined results.

Figure 2 shows a fairly standard approach to test automation. A given test scenario is run through the system and the results compared against a known baseline. For most software testing the difference is absolute. What is required for our situation is to have a smarter differencing engine that compares results and reports on whether the differences are within engineering accuracy. If they are, the new results then become the new baseline. If not, then there is an issue that needs to be addressed or understood. Sometimes our engineers will find that while the results do not appear to be within engineering tolerance, the software is nonetheless doing the right thing. The differences are artificial and could be considered the results of "butterfly effects" of a poorly conditioned system.

Figure 2. Test Automation



IV. Offshore Outsourcing

In 2005 the organization made a decision to significantly increase the amount of offshore outsourcing of software development and testing. Landmark had been involved with an outsourcing partner since 2001, but the Nexus team had not been involved.

A. Offshore Outsourcing - India

Towards the end of 2005 the Nexus team started working with an outsourcing partner in India to add a team of 6 developers and testers and began searching for a petroleum reservoir engineer to help guide the team with their understanding of the domain. At this time the world market for petroleum engineers was very scarce, and this proved to be even more of an issue in India where there is not a strong petroleum industry. In a period of more than a year, the outsourcing partner could only hire one qualified petroleum engineer, and that individual left after only 3 months. While we did have some success with having the India team run the manual smoke test every day, it became clear to us that we would not be able to make things work effectively in that environment without a domain expert.

B. Offshore Outsourcing - Romania

By 2008, after significant turn-over of computer science staff and the fact that our India partner still could not find qualified domain expertise, we started looking for Plan B. A few years earlier Todd had met an individual at a Petroleum Engineering conference that claimed to have a software development group in Romania. That individual was Florin Simion, one of the co-authors of this paper. Florin not only had a software development team, but was himself a professor of petroleum reservoir engineering with a specialty

in reservoir simulation. We thought we had a chance of making things work by finding the right talent. It also helped that the Romania time shift is 8 hours from Houston versus the nearly 12 hour shift to India. We built a team of 2 software developers and 3 petroleum engineers. While we had typical startup challenges, it was refreshing to talk to engineers that actually understood what we were trying to do. They quickly took over the smoke test, and while it initially took nearly their whole day to run the manual smoke test, they eventually got to the point where they could do it in 4 hours. This allowed time to do additional testing and also to serve as domain experts to assist the development team. The developers were working on Data Studio with one of our long time Houston developers. That developer was originally from France and had recently relocated back to France. That turned out to work well as there was only a 1 hour difference in time zone between France and Romania.

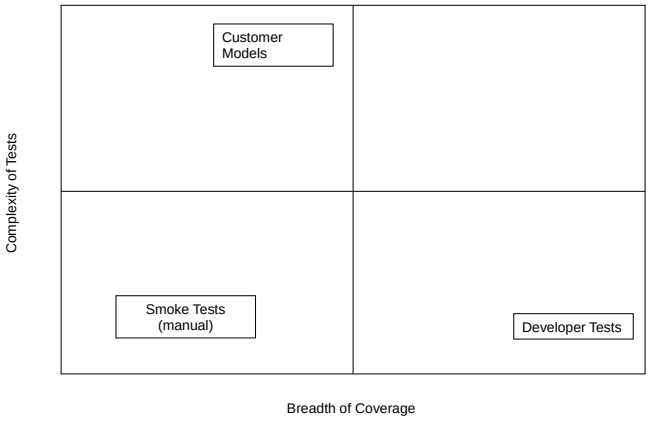
C. *Offshore Test Automation – Vietnam*

While our Romanian partner was working out well, the Landmark outsourcing coordinator met another potential partner while she was attending an outsourcing conference. This partner was Logigear, a company that provides outsourced test automation services primarily out of Vietnam. We were intrigued because they not only provided a low cost labor pool, but provided expertise in GUI test automation including their own test automation software toolkit. While we had been interested in doing more GUI test automation, our team did not have the bandwidth and did not really have the expertise to do it well. Our testers were petroleum engineers and we needed them to have engineering skills in order to know whether the test results were meaningful. We saw this as an opportunity to augment our domain talent with some test automation, thus allowing our engineers to focus on higher value testing. We kicked this off in 2009 with a team of 3 in Vietnam and a part time project manager based in California.

V. The Problem

Despite doing many things well prior to 2009, the team still struggled with quality issues. Developer tests were catching many regression issues, but the more complex customer regressions were still catching a lot of problems. Besides taking a long time to run to provide results, the problems discovered with the more complex customer data were also difficult to debug. Figure 3 shows how the three types of tests map when viewed in terms of complexity of the overall tests and breadth of coverage of functionality. Our developer test suite was very simple but covered most of the functionality. The customer datasets, on the other hand, did not utilize all the potential functionality, but were significantly more complex both in terms of size of the models as well as in the overall interactions with the models. For the overall system, the smoke tests covered workflows from the top six integrated training examples. These tests did not exercise particularly complex scenarios, but did provide reasonable coverage. Since they were manual tests they were both time consuming and monotonous for the testers.

Figure 3. Before: Test Complexity vs. Functional Coverage



As things were, both the developers and the testers were barely keeping up with the defect backlog. And when they did think they had things under control the customers would invariably find issues in beta testing or once deployed.

VI. What We Did

We decided that we needed to evaluate how we could optimize our testing efforts. The developer and customer regressions were working well but did require some maintenance to keep up with new functionality. The team felt that the greatest need was an additional set of tests that were more complex than the developer tests and exhibited some of the complexities of the customer datasets but would provide overnight turnaround. We dedicated one of our Houston petroleum engineer testers to developing what we called the "Mid-Tier" regression suite. This suite of test models was built of synthetic data subsets similar to some of the more complex customer models. Effort was put into making sure that the test suite would run overnight.

About the same time we started working with the Vietnam team to automate our smoke tests. Our objective was to increase coverage through automation, while at the same time freeing up our reservoir engineers so that they could utilize their domain expertise to do more exploratory testing and to design more test cases.

Figure 4. After: Test Complexity vs. Functional Coverage

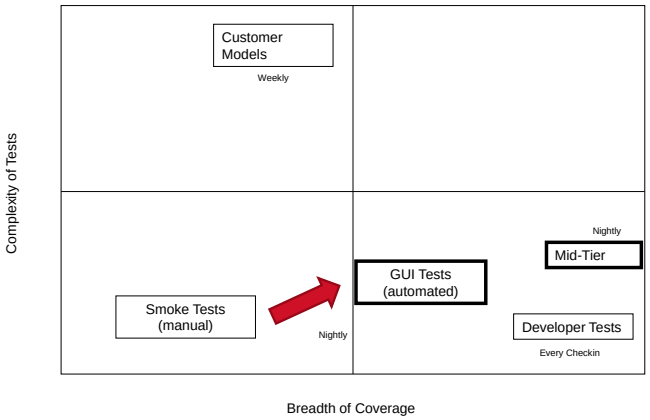
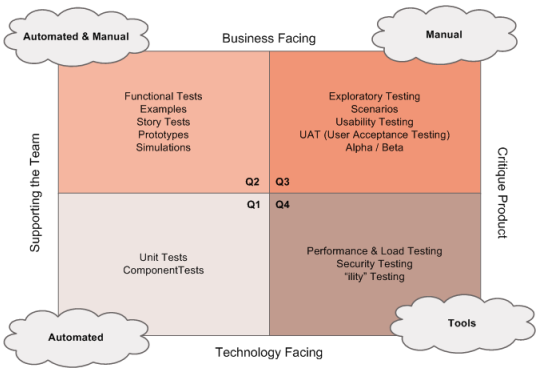


Figure 4 shows the direction that we took with our testing strategy. We looked at what was working and where we had gaps. It was an investment that we hoped would pay out with better coverage and faster feedback. With the help of our outsourcing partners, we set out to make it happen.

VII. Testing Quadrants

One useful way to look at testing strategy is through the Testing Quadrants originally proposed by Brian Marick[3] and then further expanded by Lisa Crispin and Janet Gregory[4]. The Testing Quadrants are shown in Figure 5.

Figure 5. Agile Testing Quadrants



We largely focused the automation effort on Quadrant 2 and Quadrant 4. The mid-tier and customer tests were geared first towards functional accuracy (Q2), but since computational performance is one of our key differentiators we made sure to track any performance changes as we ran all our tests (Q4). It is also worth noting that the Nexus team chose to use a lightweight form of functional testing with the developer regression suite to cover what would often be done via unit testing in Q1. The team did have some unit tests and arguably could have had more unit tests, however their approach to having a developer regression suite of lightweight functional tests worked quite well for them. The nature of the reservoir simulation problem is such the solution of the whole system of equations is necessary to see the full interplay of the complex physics being simulated.

A key aspect of our overall test automation strategy was that it freed our valuable reservoir engineers to be able to spend more time on Exploratory Testing (Q3). This is where the engineers could really utilize their domain knowledge to challenge the system in a manner likely to be used by one of our customers.

VIII. Testing Logistics

The initial smoke test automation pilot project with Logigear targeted the top six integrated workflows used for Nexus software training. Test Leads for the pilot were established at each testing location with workflows prioritized and accountability for initial Logigear automation split across assigned testing resources. Houston engineers found it easiest to provide movies that guided the testers through the integrated smoke test workflows. Our Simco test lead had previously spent time with the team in Houston and being very familiar running the smoke tests manually could answer any questions the Logigear testers had regarding workflow requirements that may not have been clear in the movie clips.

In addition to providing experienced test automation engineers LogiGear also provided the test automation tool (TestArchitect) which they developed. Initially, the testing tool required some development to support Linux as well as some of the legacy application components. The ability to customize the tool was critical so that all of the test cases could be automated.

The tool utilized a methodology developed by LogiGear called Action Based Testing[5]. With test automation a potential pitfall is often the time required to maintain automated tests. Especially in agile development the test team has to keep pace with the ongoing development. If major revisions of the automated tests are required for each new software release then the test team will always have problems keeping the tests up to date. A primary goal of our automation was to utilize a method that would allow the team to maintain and grow the test suite without major effort.

Using the Action Based Testing approach, the remote test engineer viewed the movie and created the test cases as a series of keywords (actions) with arguments. The automation focused not on automating test cases, but automating the actions. Since there are many fewer actions than test cases, and action implementations tend to be shorter than test case implementations, the automation effort is more manageable. This is especially evident when the application under test changes. Using the action based test suite, only a limited number of actions had to be maintained.

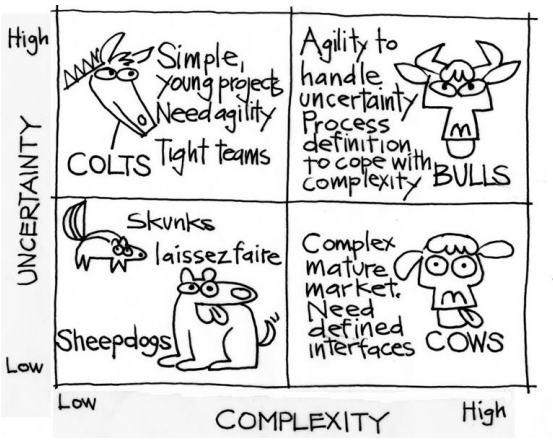
The potential risk with this method is that the Test engineers need to be well trained in test design to ensure productivity through reusability and maintainability of test keywords (actions) and tests. This approach also requires a tool or framework that supports keyword-based automation such as TestArchitect. Logigear is a company that knows their core competency of test automation and focuses on it. They have a very thorough training program to educate new hires on Action Based Testing, the Test Architect tool, and perhaps as important--how to work with different cultures.

Managing the globally distributed teams was challenging but worked out quite well overall. Our primary development was in Houston with some in France and Romania, while we had domain testers in Houston and Romania with the automation testers in Vietnam. Having the project manager for Logigear in their California office was invaluable for the communication required for test tool augmentation as well as any necessary testing automation reprioritization.

IX. Team Logistics

As mentioned earlier, the Nexus product line is made up of multiple applications which are developed by sub-teams. The individual products are quite different in their technology, team size, amount of legacy code and other parameters. The Context Leadership Model[6][7] shown in Figure 6 is a model that we have used to look at projects based on the degree of uncertainty and complexity.

Figure 6. The Context Leadership Model



Complexity includes project composition such as team size, geographic distribution and team maturity. Uncertainty includes both market and technical uncertainty. The four quadrants are named with metaphorical animals described in Table II.

TABLE II. CONTEXT LEADERSHIP MODEL

SheepDogs	Simple projects with low uncertainty
Colts	Simple projects with high uncertainty
Cows	Complex projects with low uncertainty
Bulls	Complex projects with high uncertainty

In Table III we show the four primary subprojects and how they map out with the Context Leadership Model.

TABLE III. TEAM CHARACTERISTICS

Component	Quadrant	Team Size	Iteration Length	Standup
Data Studio	Sheepdog	5	Iterationless	2/week
SurfNet	Colt	7	1 week	1/day
NexusView	Sheepdog	2	1 week	1/day
Nexus	Cow	14	3 weeks	3/week
Overall	Bull	28	3 weeks	none

The Data Studio team, while globally distributed was nonetheless fairly small and as well had very well defined tasks necessary for the update from the legacy simulator to cover the new functionality. With low uncertainty, generally low complexity and a senior team leader, we let the team largely manage themselves.

Surfnet was a new product and was looking to provide a solution that no other commercial product currently solved. This meant that it had high uncertainty. The team was relatively small, although globally distributed. The senior developers were collocated in Houston with 2 remote developers and a tester in Romania. The product manager was in Houston and his proactive involvement was critical. To get the product started he developed a user story board. The graphical description of the results he was looking for worked very well to communicate with both the local and the remote team. Of course the pictures were just an invitation to a further conversation. The team relied heavily on the product manager and he made sure to spend time with each of the senior developers typically daily and often several times per day. The remote team was managed independently by one of the senior developers and communicated as necessary via email and phone conversations with a minimum of a weekly synch up meeting.

The NexusView team was two developers and one primary tester. This project had some overlap with the Surfnet project so we simply merged the team into the Surfnet Scrum meetings.

The Nexus simulator team was by far the largest team but was all collocated within Houston. Nexus is the core engine and must coordinate with the other supporting applications. Overall the uncertainty was moderate and the overall complexity put it into the cow category. As a result we settled on a longer iteration length of 3 weeks. The team started with daily standups, and while they found value in the standups they felt that the nature of their R&D work fit better with standups every other day. The team adjusted and continued to deliver in a highly effective manner.

The overall system of systems required managing all of the uncertainty and even more complexity. The total team size was such that we did not feel the need for a Scrum of Scrums model. Instead, we had two ScrumMasters that covered all of the projects, and essentially had them pair to cover the overall release. Each ScrumMaster had primary accountability for a couple of teams, and the other participated in key Scrum meetings for those projects that they did not have direct responsibility. In that way both of them were up on the overall program and knew what cross team issues needed to be resolved. This model worked quite well as not only did the cross team communications happen efficiently, but when one of the ScrumMasters was out we had the other one help out without missing a beat.

X. HOW DID IT WORK OUT?

The results from the project were impressive. Our concerted effort on improving quality demonstrated significant improvement over the prior year. In both cases we had a 2-3 month beta program with a couple of key customers. Table III summarizes the results and compares with the prior year. The improvement in quality was substantial.

TABLE IV. THE BOTTOM LINE

	2009	2010	Reduction
Defects Found in Beta	222	36	84%
Known Issues at Ship	104	3	97%

XI. CHALLENGES

Although overall things went very well, there were several challenges that we either had to overcome or live with.

A. Proprietary Data

We deal with very sensitive customer data. While customers are willing to share that data with us for our limited use in testing the software, our agreements generally

do not extend to our offshore partners. This limited some of what we were able to accomplish with our partners and required use of synthetic data for much of the testing done by the offshore teams. While we would have preferred to have more flexibility here, this was something that we found we could work with.

B. Time Shift

While the time shift to both Romania and Vietnam created challenges with communication, in the end the time shift and overlap in times between teams actually turned out to work to our benefit. Most of our team was in Houston, while our petroleum engineering partner was in Bucharest, Romania and our test automation partner was in Vietnam. The time shift to Romania is a very manageable 8 hours, and particularly manageable as our partner was very flexible with work schedules. We utilized the Romania team to help with communications with the Vietnam team.

Once we had tests in operation, the Vietnam team would initiate the automation tests during their day and have a time overlap with the Romania team during the Romanian morning. By the afternoon in Romania, the petroleum engineering team would take a deeper dive into any issues raised by the automation tests to make sure that we understood what the issues were. In the end what we got was a daily automation that ran during Houston nights and provided reliable status by the time developers arrived the next morning.

XII. Conclusions: What Did We Learn?

There are a couple of key lessons learned from this experience.

A. Test Automation is Necessary to Maintain Velocity

Prior to this initiative the team was diligently working but nonetheless struggling to keep up with quality issues. Our existing automation testing was invaluable, but we still relied too much on much manual testing. We also realized that some additional automation suites could make a big improvement in our overall productivity. By augmenting our test automation we were able to find some issues faster and also have our domain experts spend more time on exploratory testing.

B. A Testing Strategy Helps to Maximize Efficiency

The team had some good automation and exploratory testing, but knew they could be better. Rather than just randomly add more tests, we looked to see which types of tests would add the most value. For us we found that adding an additional set of functional tests and automating some of our GUI smoke tests could pay off quite well.

C. Outsourcing Can Work When Used Judiciously.

We relied heavily on outsourcing partners to get this work done. While we had some minor challenges in the beginning, we found that it was quite workable. It won't work well if you don't have the right talent or the right attitude. We found that even test automation can be outsourced effectively. The key was the combination of domain expertise provided by our own team and our Romanian partner, with the test automation expertise of our Vietnam partner and the in-house project management that made this globally distributed team work in our agile development environment.

D. Treat Outsourcer as a Partner

By focusing on what our partners were good at and recognizing what we were good at in-house, we were able to leverage our overall talent. We found qualified petroleum engineers that were able to be part of our team and make significant contributions. We found talented testers that arguably understood GUI test automation better than us. Our partners wanted us to succeed and we wanted them to succeed.

E. Cost Effective Global Talent

By sourcing globally we were able to get access to talent that had valuable skills to fit gaps at a fraction of the cost. We needed to make an investment to improve the quality and business conditions would not have enabled us to obtain the same effort had we sourced locally.

In a prior paper by Little[8], savings were computed by:

$$Savings\% = 1 - \frac{c + m}{e}$$

Where

- e = OUTSOURCE efficiency = Equivalent INTERNAL days per OUTSOURCE day
- m = INTERNAL Management overhead = INTERNAL days to get 1 OUTSOURCE day
- c = OUTSOURCE relative cost = cost per OUTSOURCE day / Cost per INTERNAL day

Our estimate for these parameters is shown in Table V:

TABLE V. Cost Savings

e	m	c	Savings
0.5	0.1	0.20	40%

We found that efficiency (e) started out low while management overhead (m) started out high. Over time efficiency has improved while management overhead has declined. These trends continue.

F. Distributed Teams Can be Effective

Our teams were globally distributed and we certainly had some overhead associated with that distribution. We aimed to minimize the overhead of the distribution using a pattern common to software development – loose coupling and tight cohesion. We aimed to have locally collocated teams that had tight cohesion, and recognized that there was coupling and dependencies across distributed teams. We first sought to understand those dependencies and then made sure to monitor and manage the dependencies.

G. Test Automation Does not Replace Exploratory Testing

While test automation is critical to check against regression defects, we found exploratory testing still to be

critical. Because we were able to automate more tests, we freed up our domain experts to be able to do more exploratory testing. Our exploratory testing found more than 70% of the defects.

XIII. Acknowledgements

We would like to acknowledge Mark Kilby for providing valuable shepherding of this experience report. We would also like to thank the entire Nexus development team for their dedication to making the product a success.

XIV. References

[1] B.K. Coats, G.C. Fleming, J.W. Watts, M. Rame, G.S. Shiralkar, SPE 87913: "A Generalized Wellbore and Surface Facility Model, Fully Coupled to a Reservoir Simulator," *SPE Reservoir Evaluation & Engineering*, Volume 7, Number 2, 2004, pg 132-142.

[2] B. S. Al-Matar, et. al. , SPE 106069: "Next-Generation Modeling of a Middle Eastern Multireservoir Complex", SPE Reservoir Simulation Symposium, 26-28 February 2007, Houston, Texas, U.S.A.

[3] http://www.exampler.com/old-blog/2003/08/22/#agile-testing-project-2

[4] L. Crispin and J. Gregory, *Agile Testing: A Practical Guide for Testers and Agile Teams*, Addison-Wesley, 2009.

[5] H. Buwalda, "Action Based Testing," *Better Software*, Volume 13, Number 2, March/April 2011.

[6] T. Little, "Context-Adaptive Agility: Managing Complexity and Uncertainty", *IEEE Software*, May/June 2005.

[7] P. Pixton, N. Nickolaisen, T. Little, K. McDonald, "*Stand Back and Deliver: Accelerating Business Agility*," Addison-Wesley, 2009.

[8] T. Little, "Assessing the Cost of Outsourcing: Efficiency, Effectiveness and Risk," IEEE EQUITY 2007, March 19-21, 2007, Amsterdam, Netherlands.