An Empirical Study of Transition and Adoption of the Scrum Framework in Legacy Products

Peter Howard Coles Yijun Yu Hugh Robinson

Computing Department, The Open University, United Kingdom

6/28/2011

Abstract

Agile[1] software development (ASD) methods attempt to foster agility as one dimension of the complex activity of delivering software solutions to gain business advantage in turbulent environments. It is however unknown whether such methods improve code delivery in software houses producing legacy mainframe software where the source code is not in an object-oriented programming language. In this paper, we evaluate whether there is any impact on the development process by the adoption of the Scrum framework, using a combination of quantitative and interpretive analysis on empirical data. We first test the significance of the code metrics committed by the developers before and after the transition and adoption. Using in-depth knowledge of the commercial project, we then interpret the results of statistical tests on the null hypothesis that the metrics show no significant change. The study has been conducted using real life empirical data gathered from the development of a sample commercial legacy product.

[1] In a software development context, confusion can occur when referring to the term 'agile' (with a small 'a') and 'Agile' (with a capital 'A'). In the interests of clarity, we use 'agile' as the dictionary definition of the term – 'able to move quickly and easily; able to think and understand quickly' http://oxforddictionaries.com; in contrast, 'Agile' (software development) refers to a specific set of processes, techniques and stances that have evolved and are underpinned by, the values of the non-profit organization, the Agile Alliance, http://www.agilealliance.com.

1 Introduction

Agile software development (ASD) methods continue to gain popularity in industrial settings. Possible reasons for their adoption might include a belief of increasing productivity while maintaining or improving quality – i.e. 'doing more with less' [16], 'gut reactions'/'jumping on the bandwagon' [24] or pressure to adopt a new technology or idea [5]. Whilst there is anecdotal evidence published on ASD methods including descriptive articles and lessons learned, e.g., [23, 19], there are not many empirical studies that focus on Agile adoption [15].

Typically, the Agile sweet-spot is the place where the software industry as a whole has the most experience with using Agile methods. Such software is normally developed in an object-orientated programming language for web-based application domains rather than in assembly language for legacy systems. Its developers are normally close to the eventual end-users or actual customers and team sizes are usually smaller than a dozen people [13] . On the other hand, applying Agile processes to legacy systems, whether within maintenance or as new development, could raise numerous issues. Legacy systems generally are difficult to refactor, restructure [6] or reengineer [12, 25] in order to accommodate Agile replacements that need to build capability in increments. Cost may not be justified or resources may be constrained for large organisations running 'lean and mean'. Approaches to Agile adoption range from a large-scale, top-down a-la-carte forced approach, ironically portrayed below by Scott Adams in Figure 1, or attempting cultivation of ground-up adoption to garner grassroots acceptance [10].

Figure 1: The Pointy-Haired Boss Agile adoption approach

In this study, the practices of the Scrum framework were adopted by a legacy software development team in a large company. How does the adoption change the work outputs of legacy software developers? To assess this question, a clear understanding of the maintenance development context is required in terms of organisation, product, process, people and case selection.

2 Context of our empirical study

Organisation *Vendor01* is an established global independent IT management and optimization software company based in the US. It has customers in the Forbes Global 2000, government organizations, educational institutions and many other global companies. Like most large organisations, the company management is hierarchical.

Product and Technology The product *Product01* chosen for the pilot case study can be regarded as legacy, having approximately 18 years of continuous evolution and adaption of the source code. This product forms a large inter-related product suite *Suite01* addressing all facets of mainframe database management and optimization; looking forward, there is potential to study over 60 products of diverse size in our future research. *Product01* is a high-speed utility using data-processing techniques to unload mainframe database tables while maintaining system performance. It provides formatting options that make the output data available in a chosen format for immediate use in other applications or databases. The product is written in High Level Assembler (HLASM), an IBM licensed program running on the mainframe z/OS platform that enables programme development of subroutines and functions not typically provided in other symbolic languages, such as COBOL, FORTRAN, and PL/I [11].

Process and time gaps The adoption of the Agile Scrum framework officially begain at the first Scrum sprint – 30th March 2009. A new release lifecycle for the product begins, as shown in Figure 2 when the latest post-release production code is 'snapped' at point x. Post-release code is the responsibility of a sustaining engineering team who do not follow Agile practices – these activities are shown by dark red arrows after the pre-releases. Release 'A' is snapped at time x to form release 'B'. This new release is the responsibility of the development team following the Scrum framework – these activities are shown by light green arrows before the pre-releases. The pre-release phase consists of the normal Scrum rhythms and practices – new product value is prioritized and developed from the product backlog.

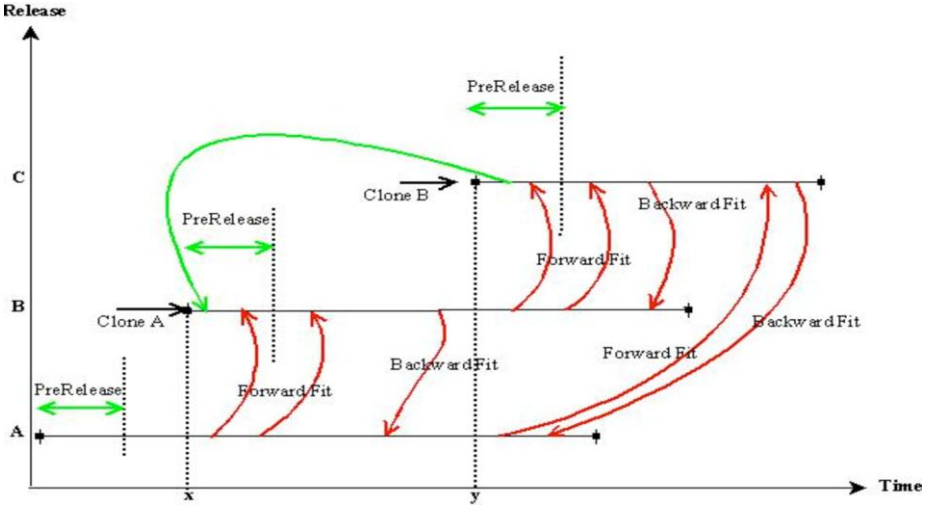

Figure 2: The process changes after the adoption of the Scrum framework

Company policy dictates that all external customer problems found in earlier releases are applied and tested to pre-release code. Customers typically run multiple mainframe database subsystems at different levels of operating system and maintenance; this leads to multiple post-releases of *Product01* requiring maintenance

by sustaining engineering. Occasionally, customers require functionality back-fitting from a newer release to an older release; this can arise for a number of reasons; a common cause is the risk and effort required by the customer to migrate to new levels of system and vendor code is prohibitive. Targeted functional provision in older releases can help mitigate these risks; *Vendor01* considers this part of a partnership relationship with its customers, attempting to balance customer need with internal resource constraints and costs incurred by supporting multiple post-releases of *Product01*. All high-impact fixes are back-fitted to earlier releases regardless of cost.

By analysis of the time stamps of work items, it is possible to track the following time gaps: `PreStgTg` or pre-stage time gap tells how long does it take for the developer to commit the code after they finish the work; `StgIntTg` or stage-to-integration time gap tells the time in hours taken to migrate the change to start the formal testing process; `IntPrdTg` or integration-to-product time gap tells how long it took to test the change and migrate it into production as 'passed'.

Case Selection We are interested in exploring what happens when Scrum is adopted in an industrial setting and we recognize that case selection is crucial in our investigations. The effects of the Scrum framework can only be assessed at a high level of abstraction as there may be many dimensions that come into play that cannot be controlled [14]. We are attempting to reduce the effect of confounding factors by selecting a consistent, typical application domain which is actively evolving (*Product01*) whose developers *Geek01* and *Geek02* are selected on basis that they have detailed organizational and product domain knowledge, and over four decades experience in the software industry and mainframe technology. Also both developers have been members of the same Scrum team since adoption. Our case study only considers time periods for Scrum sprints, that is, pre-release product development from the product backlog. We only consider pre-release development before the Scrum transition to improve the validity of our comparisons and reduce confounding. The official organization date of the first Scrum sprint used in the case studies is 30th March 2009.

The legacy software products of the company are primarily written in assembler code. Studies exist that use different metric suites to assess object-orientated code, for example Sato et al. [22] discuss and evaluate object-orientated metrics from seven projects with differing approaches of Agile adoption. In non-object-orientated environments, we draw on over four decades of research into software size (product) metrics that are grounded in a rigorous approach to measurement theory. We refer the reader to key texts on measurement theory [29] and software metrics [7]. Our research follows the principles laid down by these texts and we use program length or LOC (lines of code), CLOC (commented lines of code) and McCabe [18] (cyclomatic complexity) metrics as the initial metrics forming our analysis.

3 Results and analysis

3.1 Geek01 Change Nature and Change Timing

We found statistically significant differences (at the 5% significance level) between the two periods for both the size of the modules changed (Loc - 0.038 significance level) and the nature (DeltaLoc - 0.013 significance level, TotalChanges - 0.003 significance level) and timing (PreStgTg - 0.000 significance level) of the changes for

this developer. Despite the difference in time periods before and after Scrum, Geek01 changed the same number of modules (81); worked on larger modules, some in the 6000 to 8000 LOC range, and wrote 7 new modules summing 4,365 lines in the Scrum period. The evidence suggests this developers work output seems to have increased since Scrum adoption. Most module changes remain small in the two periods (fitting well with Agile methods emphasis on time-boxed incremental development), however Geek01 produced 5 significantly different changes in the Scrum period that were over 400 added lines, one of these was over 1,300 added lines. These 5 changes were also reflected in the code churn metric (TotalChanges) which was also significantly different; these changes were related to new functionality delivered into the product. The time taken to commit developed changes formally into the change control system (PreStgTg) showed a significant increase in duration before and after Scrum adoption. There is a statistically significant delay for 14 modules which had finished development but were not added to the formal change control system for QA testing; some changes took between 2 and up to 9 Scrum iterations before being formally committed. This evidence raises questions about what 'delivery' entails (in a similar view to the Scrum definition of 'done' discussed previously); the change control cycle illustrated in Figure 6 above allows for both informal testing of code prior to the formal code migration and testing cycle, which address product integration. It suggests that continuous integration is not practiced. Geek01 confirmed this view when asked:

> *I created temporary builds that ... I used during my unit testing and ... QA testing.*

Agile advocates recommend that developers should commit smaller chunks of code frequently rather than delaying and committing several changes at once. However, we found no significant difference in cyclomatic complexity of modules changed by Geek01 before and after Scrum; average McCabe before and after the adoption was 207 and 280 respectively, suggesting there has been no statistically significant change to the overall complexity of product change for this developer, despite increasing module complexity in iterations. The increases in McCabe and LOC before and after adoption suggest that new features are being 'bolted on' to the existing design; whilst average LOC for changed modules increased from 1,728 to 2,462.

3.2 Geek01 Iteration Change Delivery

Scrum iterations are time-boxed - each team member commits to complete agreed functionality of the product that delivers the highest business value (determined by the Product Owner) in each Sprint. We wanted to know if this resulted in smoother, predictable code delivery (Agile sustainable pace). In light of our discussion on the meaning of 'delivery' in the previous section, we look at the iterations where the code was last changed outside of the change control system - we know this code was later migrated through the formal change control system and was deployed in production, but we have found the migration process can span iterations.

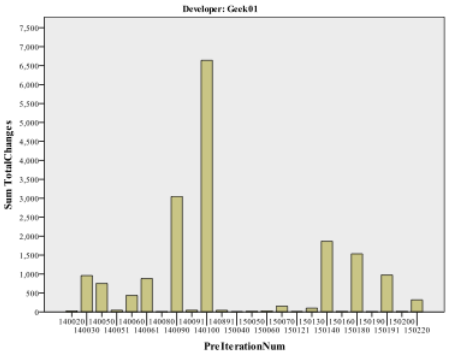

Figure 3: Geek01 Total Code Churn by Iteration

Figure 3 illustrates the total iteration code churn for Geek01. The delivery pattern is punctuated showing variable iteration change delivery, contrary to the Agile principle of sustainable pace. We asked Geek01 to comment on these results:

> *There is no formal estimating techniques used, this area needs a GREAT DEAL of improvement.*

Agile principles of frequent delivery, sustainability and feedback loops are not in evidence based on the data distribution shown in Figure 3. Our evidence for this developer indicate no constant flow of new features into production.

3.3 Geek02 Change Nature and Change Timing

We found statistically significant differences (at the 5% significance level) between the two periods for the size (Loc - 0.006 significance level) and complexity (Mccabe - 0.021 significance level) of modules changed, but no significant difference in the nature of the changes. We did find a significant difference in the timing (PreStgTg - 0.000 significance level) of changes for this developer. Geek02 changed 218 modules before Scrum but only 42 in the Scrum period; Allowing for the difference in time period before and after Scrum, the evidence suggests this developers work output seems to have decreased (from a approximate mean of 43 per year to an approximate mean of 21 changes per year) since Scrum adoption. Geek02 has changed fewer, more complex modules (the largest having Loc 7,584 and Mccabe 879). When we interviewed Geek02 concerning the drop in delivery, he complained of distractions with unrelated sustaining engineering tasks that dominated his development time:

> *"If I can get away from code reviews for sustaining engineering long enough to do ANY development of ANY kind, it's a winâ€ ¦it may take me 2-3 weeks to get enough time away from sustaining engineering to actually make progress on ONE project... Nothing ever just gets fixed without a big cyclical production, and it just sucks off all my time"*

Geek02 also expressed the view that Geek01 did not perform code reviews for sustaining engineering and could concentrate on his Scrum commitments:

> *"Geek01 is VERY organized with his note-taking and recording test cases, has 'very little' general interface with sustaining engineering (I could be wrong...) and REFUSES to EVER do a sustaining engineering code review"€ •*

The time taken to commit Geek02 developed changes formally into the change control system (PreStgTg) showed a significant decrease since Scrum adoption. In contrast to Geek01, who changes individual modules, and waits to commit features in batches, Geek02 ensures all modules involved in a commit have the same date and then commits.

> *"If I'm working on something for awhile, I change the source date stamp to reflect the date I made a particular change - when I'm finally settled on the entire solution, I'll 'try' to make all the change stamps the same date for all the changes in a commit"*

Geek02 Iteration Change Delivery

Figure 4 below illustrates the total iteration code churn for Geek02. It shows no delivery for the first 7 iterations of Release E and spikes of delivery for iterations 9 and 10; Release F shows an 'Epic' feature delivered in iteration 4 and very little else.

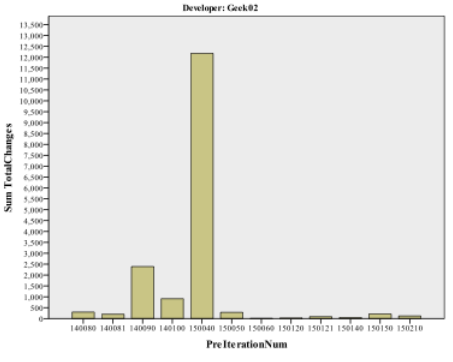

Figure 4: Geek02 Total Code Churn by Iteration

The variable code delivery shown in Figure 8 above suggests product backlog items chosen for iterations do not fit well with the iteration length and there are difficulties with planning and sizing the sprint backlog items; the sprint delivery highlights a need to improve breakdown of product backlog items mindful of sprint duration. Scrum inspect and adapt principles are designed to monitor variations in delivery and

make adjustments based on feedback from previous sprint experience. Figure 8 suggests the feedback from previous iterations is not informing future sprint workload. When asked, Geek02 highlights two concerns:

- breaking down Epic stories

> *"I have done a REALLY bad job of taking the time to break EPIC sized work into stories which neatly fit in a 1 week development + 1 week test cycle of 'deliverable function' when the actual project I'm trying to accomplish takes 2-3 months to code and unit test";*

- difficulty of delivering code that demonstrates customer value within a time boxed iteration

> *"...some things are a) bigger than a couple of days coding and a couple of days unit testing and b) have NO value without a minimum amount being 'done'. Without at least 95% of the code ready, there was nothing to test, by DEV OR QA. Some things are just not practical to break up".*

4 Related work

None of the studies on Agile adoption [1] have applied quantative analysis to legacy software development and maintenance. Agile adoption frameworks and strategies discussed in the research literature show large organizations attempt both bottom-up and top-down, 'Start next Monday' approaches [20], which involve wholesale strategies (where the entire Agile practice is adopted at once) or incremental strategies (where Agile practices are gradually deployed). The research literature commonly considers the use of an incremental adoption approach consisting of an evaluation and introduction stage assessing the ability of the organization to adopt Agile methods, and the selection of suitable Agile practices based on organizational context. Rohunen et al. [20] found this theoretical perspective does not show itself in practice; in all participating industrial case studies , there was no formal evidence-based evaluation stage; the only evidence of evaluation activity were some industrial studies using retrospectives as a proxy initial evaluation process in their initial incremental adoption process. Given adoption of Agile practices in large software development organizations is not straightforward [9], it is perhaps surprising that an evidence-based evaluation stage is not conducted aimed at determining if the software project under consideration is a good Agile fit.

Kelly [13] suggests teams attempting to work outside the Agile sweet-spot are blazing a trail and will face barriers, costs, and constrained benefits when attempting Agile practices to Engineering, Management and Release practices. Despite these concerns, a few experience reports in the research literature show practitioner attempts to use (non-Scrum) Agile practices in legacy object-orientated [26], and non-object-orientated environments [27, 2]. One paper considers how Agile practices might be operationalised in an embedded software domain [21]. Whilst there is anecdotal evidence published on Agile software development methods including

descriptive articles and lessons-learned [23, 19], there are not many empirical studies on Agile adoption.

Studies of Scrum adoption were attempted on small companies as described by Dingsoyr et al. [3] to large multinationals [8]. In addition, the Dyba & Dingsoyr [4] systematic review of research literature found only one empirical study on customer perception of Scrum using a web-based application written in C# [17]; as far as we are aware, no empirical studies consider Scrum in a legacy context.

5 Conclusions

To assess the effects of transition to the Agile Scrum framework in a legacy context, in this work, we have explored how Agile Scrum methods are adopted in industry through a pilot longitudinal (spanning two years) quantitative exploratory case study. A statistical analysis of the quantitative code metrics helped us to evaluate the effectiveness of adoption of Scrum framework in the development and maintenance of legacy software in a large organisation. An initial qualitative analysis was conducted to discover factors hidden behind the statistical significances.

References

[1] Gabrielle Benefield. Rolling out agile in a large enterprise. In *Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, HICSS '08, pages 461–, Washington, DC, USA, 2008. IEEE Computer Society.

[2] G. Coleman and M. McAnallen. Managing the challenges of legacy systems using extreme programming. *Software Process: Improvement and Practice*, 11(3):269–275, 2006.

[3] Torgeir DingsÃ¸yr, Geir Hanssen, Tore DybÃ¥, Geir Anker, and Jens Nygaard. Developing software with scrum in a small cross-organizational project. In Ita Richardson, Per Runeson, and Richard Messnarz, editors, *Software Process Improvement*, volume 4257 of *Lecture Notes in Computer Science*, pages 5–15. Springer Berlin / Heidelberg, 2006.

[4] Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Inf. Softw. Technol.*, 50:833–859, August 2008.

[5] Tore Dyba, Barbara A. Kitchenham, and Magne Jorgensen. Evidence-based software engineering for practitioners. *IEEE Software*, 22:58–65, 2005.

[6] M.C. Feathers. *Working Effectively with Legacy Code*. Pearson Education, 2004.

[7] Norman E. Fenton. *Software Metrics: A Rigorous Approach*. Chapman & Hall, Ltd., London, UK, UK, 1991.

[8] Brian Fitzgerald, Gerard Hartnett, and Kieran Conboy. Customising agile methods to software practices at intel shannon. *Eur. J. Inf. Syst.*, 15:200–213, April 2006.

[9] Paul Hodgetts. Refactoring the development process: Experiences with the incremental adoption of agile practices. In *Proceedings of the Agile Development Conference*, pages 106–113, Washington, DC, USA, 2004. IEEE Computer Society.

[10] M. Huisman and J. Iivari. Deployment of systems development methodologies: perceptual congruence between is managers and systems developers. *Information & Management*, 43(1):29–49, 2006.
[11] IBM Inc. High level assembler for z/os release 6, http://www-01.ibm.com/software/awdtools/hlasm/library.html#assembler, 2008. Accessed 31 March 2011.
[12] Meena Jha and Piyush Maheshwari. Reusing code for modernization of legacy systems. In *Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice*, pages 102–114, Washington, DC, USA, 2005. IEEE Computer Society.
[13] A. Kelly. The limits of agile, 2010. http://www.infoq.com/articles/limits-of-agile, [Accessed 20 March 2011].
[14] B. Kitchenham, L. Pickard, and S.L. Pfleeger. Case studies for method and tool evaluation. *Software, IEEE*, 12(4):52 –62, jul 1995.
[15] M. Lindvall, V. R. Basili, B. W. Boehm, P. Costa, K. Dangle, F. Shull, R. Tesoriero, L. A. Williams, and M. V. Zelkowitz. Empirical findings in agile methods. In *Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods - XP/Agile Universe 2002*, pages 197–207, London, UK, 2002. Springer-Verlag.
[16] Mikael Lindvall, Dirk Muthig, Aldo Dagnino, Christina Wallin, Michael Stupperich, David Kiefer, John May, and Tuomo Kahkonen. Agile software development in large organizations. *Computer*, 37:26–34, 2004.
[17] Chris Mann and Frank Maurer. A case study on the impact of scrum on overtime and customer satisfaction. In *Proceedings of the Agile Development Conference*, pages 70–79, Washington, DC, USA, 2005. IEEE Computer Society.
[18] Thomas J. McCabe. A complexity measure. In *Proceedings of the 2nd international conference on Software engineering*, ICSE '76, pages 407–, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.
[19] Orlando Murru, Roberto Deias, and Giampiero Mugheddu. Assessing xp at a european internet company. *IEEE Software*, 20:37–43, 2003.
[20] Anna Rohunen, Pilar Rodriguez, Pasi Kuvaja, Lech Krzanik, and Jouni Markkula. Approaches to agile adoption in large settings: A comparison of the results from a literature analysis and an industrial inventory. In Muhammad Ali Babar, Matias Vierimaa, and Markku Oivo, editors, *PROFES*, volume 6156 of *Lecture Notes in Business Information Processing*, pages 77–91. Springer, 2010.
[21] J. Ronkainen and P. Abrahamsson. Software development under stringent hardware constraints: do agile methods have a chance? In *Proceedings of the 4th international conference on Extreme programming and agile processes in software engineering*, XP'03, pages 73–79, Berlin, Heidelberg, 2003. Springer-Verlag.
[22] Danilo Sato, Alfredo Goldman, and Fabio Kon. Tracking the evolution of object-oriented quality metrics on agile projects. In *Proceedings of the 8th international conference on Agile processes in software engineering and extreme programming*, XP'07, pages 84–92, Berlin, Heidelberg, 2007. Springer-Verlag.
[23] P. Schuh. Recovery, redemption, and extreme programming. *Software, IEEE*, 18(6):34 –41, Nov/Dec 2001.

[24] Judith Segal. The nature of evidence in empirical software engineering. In *Proceedings of the Eleventh Annual International Workshop on Software Technology and Engineering Practice*, pages 40–47, Washington, DC, USA, 2003. IEEE Computer Society.
[25] Vandana Shah and Ainsley Nies. Agile with fragile large legacy applications. In *Proceedings of the Agile 2008*, pages 490–495, Washington, DC, USA, 2008. IEEE Computer Society.
[26] Vandana Shah and Ainsley Nies. Agile with fragile large legacy applications. In *Proceedings of the Agile 2008*, pages 490–495, Washington, DC, USA, 2008. IEEE Computer Society.
[27] C. Stevenson and A. Pols. An agile approach to a legacy system. *Extreme Programming and Agile Processes in Software Engineering*, pages 123–129, 2004.
[28] Norman G. Vinson and Janice Singer. A practical guide to ethical research involving humans. In Forrest Shull, Janice Singer, and Dag I. K. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 229–256. Springer London, 2008.
[29] Horst Zuse. *Software Complexity: Measures and Methods*. Walter de Gruyter & Co., Hawthorne, NJ, USA, 1990.