2013 Joint Conference of the 23nd International Workshop on Software Measurement (IWSM) and the Eighth International Conference on Software Process and Product Measurement (Mensura)

Measuring Best-in-Class Software Releases

Hennie Huijgens
Delft University of Technology and Goverdson
The Netherlands
h.k.m.huijgens@tudelft.nl

Rini van Solingen
Delft University of Technology and Prowareness
The Netherlands
d.m.vansolingen@tudelft.nl

***Abstract*— In this research we aimed to identify distinguishing factors in software releases. For this purpose we analyzed the metrics of 26 software projects. These projects were release-based deliveries from two stable, experienced development teams. During the measurement period both teams transformed from a plan-driven delivery model (waterfall) to an agile approach (Scrum). Overall, we observed that these small release-based projects differ largely from non-release-based projects. Our research indicates that a combination of release-based working, a fixed and experienced development team, and a steady heartbeat contribute to performances that can be characterized as best-practice. The main contribution of this paper is that we found five success factors (all reducing development complexity) that result in best-of-class performance for small software releases.**

Keywords—IT-metrics; Performance Measurement; Software Engineering; Scrum; Productivity; Time-to-Market; Quality

I. MOTIVATION

A. Problem Statement

From the measurement activities in three large companies (two Dutch banks and one Belgian Telecom Provider) we obtained a repository with primary data of 345 finalized IT-projects. During the analysis of this repository we noticed a special trend: a specific subset of small projects showed a best-in-class performance, that clearly stood out, in comparison with other IT-projects considering their performance on time and cost. When tracing these projects back to their origin, we found that all these projects were performed by two specific development teams.

This subset of best-in-class projects turned out to be a sample of 26 small software releases (all with a size between 5 and 57 FPs) that were performed by two specific software engineering teams. What was done differently here, compared to the other projects in the repository? What can we learn from these two teams? Which best-practices are they applying that lead to these results?

B. Research Objectives

To answer these questions, we decided to dive into these projects and, as such, attempting to reveal which specific conditions makes these small software releases more successful than others. Investigating the literature on small software releases, we could not find research results specifically focused at the performance of small software releases. This in itself is already surprising, because in practice we notice a clear trend towards smaller releases in higher frequencies. This can be clarified by an increasing amount of systems on which evolution takes place, and that are maintained using incremental (release-driven) approaches for software development.

Small software releases seem to become more relevant for the industry due to agile delivery models that all emphasize short-cyclic development and high frequent releases. As such, it makes sense to investigate how to make small software releases perform better. However, when looking, for example, at the publications in recent software measurement conferences, benchmarking and estimation of a specific subgroup of software engineering activities, small software releases, seem to be a relative greenfield activity.

In order to understand what caused the remarkably good performance of these releases, we narrowed our research to this set of 26 best-in-class software engineering releases. A first glance at the two specific teams showed that these teams went through a transition from a plan-driven delivery approach (waterfall) to an Agile one (using Scrum). Stimulated by the possibility to create more insight into the backgrounds of these software releases, and at the same time attempting to support senior IT-management in this company (who were driven by an ongoing search for cost savings, and needed to decide between either deploying Scrum to all their teams or outsourcing parts of the work to an IT-company in India), we decided to focus our study to the following two research questions:

RQ1:"What differences in the 26 best-in-class software releases can be found that explain their over-performance compared to the other measured projects and releases?"

RQ2: "To which extend can the transformation to Scrum in the investigated teams be an explanation for their over-performance?"

C. Context

The analyzed 26 small software releases were part of a larger collection of project data collected over a four-year period in a Dutch banking company. The complete repository did hold a variety of project data from different parts of the company, yet the sample that is analyzed in this research is about 26 releases from two teams that developed and maintained two Finance & Risk systems.

This paper is structured according to a proposed reporting structure for research reports, by Jedlitschka and Pfahl [1]. It subsequently discusses related work, research design, execution, analysis, interpretation, and conclusions and future work.

978-0-7695-5078-7/13 $26.00 © 2013 IEEE
DOI 10.1109/IWSM-Mensura.2013.29

II. Related Work

Various alternative studies have been performed on performance aspects of software enhancements and on small software releases in particular. In this section we discuss these alternatives and relate them with our research.

The staffing profile of both releases – average team size 3 FTE on level load with on average 5 people working in each team - matches with findings on ideal staffing in related studies. Rodriguez et al. [2] refers to Putnam [3] regarding the ideal team size, stating that productivity is higher for smaller teams with an optimum team size of 3 to 5 staff, but with very similar values for team sizes of 5 to 7 staff.

Sutherland et al. [4] studied best practices in a globally distributed Scrum environment and comes up with, as the paper states 'the most productive Java projects ever documented', with productivity figures of 15.3 function points per developer/month. However, a remark is in place here; Sutherland did not count the FPs itself, they are derived from the number of Source-Lines-Of-Code (SLOC). This method, also referred at as backfiring, is assumed to be less adequate then a normal FP-count, not reliable when the starting point is SLOC, and the method is not supported by any organization of measurement specialists [5].

Dybå and Dingsøyr [6] inventoried five aspects that conclude criticism by some practitioners and academics on agile development methods, including agile development is nothing new; such practices have been in place in software development since the 1960s, there is little scientific support for many of the claims made by the agile community, and agile development methods are suitable for small teams, but for larger projects, other processes are more appropriate. Out of 36 empirical studies on agile software development they found 4 studies that compared the productivity of agile teams with the productivity of teams using traditional development methods. The studies showed results that are difficult to fit together of resp. improvements of 337%, 42% and 46% of agile versus traditional, yet also a difference of -44% between agile and traditional. Also with respect to product size the examined studies where not consistent in their findings; results varied from delivery of 78% more lines of code in agile teams to no differences in delivered product size between agile and traditional teams. Finally Dybå and Dingsøyr conclude that in particular, agile project management methods, such as Scrum, warrant further attention. They see that there is a backlog of research issues to be addressed.

Sureshchandra et al. [7] studied the performance of 15 distributed agile and distributed non-agile projects, finding that schedule deviation of distributed agile projects was (X-7.31)% compared to distributed non-agile projects (X), and overall productivity of distributed agile projects was 1.11 better.

Estler et al. [8] compared 66 globally distributed industry projects, classified into structured and agile, on the correlation between process type and overall success, cost-effectiveness, team motivation, importance for customers, and amount of communication. The outcome of the study suggested that the correlations between process type were negligible and without statistical significance. Choosing an agile rather than a structured approach does not appear to be a crucial decision for globally distributed projects.

Ramasubbu et al. [9] analyzed the performance outcomes of 362 globally distributed projects from four different organizations, showing that there is a fundamental trade-off between productivity, quality and profits at the project configuration level. The research provided actionable insights that can help teams to choose configurations that help achieve preferred performance outcomes. Process choice can play an important role here; teams that are originally configured to boost quality could adopt a shorter planning range and more frequent release cycles, where teams configured for high productivity could boost quality and eventually profitability by incorporating structured processes and more disciplined planning.

Tan et al. [10] mention that incremental and iterative development (IID) models are now being used by many organizations to reduce development risks while trying to deliver the product on time. The main advantage of risk-driven incremental and iterative development is spreading and mitigating risks in small steps (meaning small loss). They measure a decreasing productivity over time when measuring six consecutive iterations.

Premrai et al. [11] investigated how software project productivity had changed over time. To do so a dataset of more than 600 Finnish projects, including 65 from the banking sector that were collected since 1978 were analyzed. Overall an improving trend was measured, however less marked since 1990. The trend varied over companies and business sectors. There is no evidence of diseconomies of scale. New projects tend to be significantly larger in size than maintenance projects.

Boehm [12] states that software projects are nowadays growing larger in size and become more and more complicated while delivery requires faster turnaround time. He points out that analysis of the relative home grounds of agile and plan-driven methods found that agile methods were most workable on small projects with relatively low at-risk outcomes, highly capable personnel, rapidly changing requirements, and a culture of thriving on chaos vs. order.

Top et al. [13] mention that primary inputs of an estimation model are software development effort and software size. Although all off the publicly available repositories they evaluated in their research include size and effort; data lacks detail to create reliable estimation models. Effort for different phases of software development life cycle or for each work task in work breakdown structure is not available. Besides that, non-availability of details on size results in validation problems.

III. Research Design

A. Goals, hypotheses and variables

The goal of the analysis is to prove whether the 26 software releases indeed perform better than the average of the repository with 345 software projects, looked at from the point of size, duration and project cost. And, if so, what factors can explain that over-performance. A second goal is to identify whether a transformation to Scrum can be an explanation for that over-performance.

Based at these goals the following hypotheses were defined:

H1: Time-to-market (in Days/FP) of 26 releases was better than average time-to-market in the repository, where Time-to-market is calculated as (1) a weighted average over the total repository and as (2) a weighted average at the average size of the analyzed sample.

H2: Productivity (in Cost/FP) of 26 releases was better than average productivity in the repository, where Productivity is calculated as (1) a weighted average over the total repository and as (2) a weighted average at the average size of the analyzed sample.

H3: Performance of Scrum releases (in Days/FP and Cost/FP) was better than plan-driven releases.

In order to validate these hypotheses variables with regard to size, duration and cost of releases and projects are collected and analyzed.

B. Design

The research was performed as an observational study; during the measurement period we worked directly between the release-teams. We were able to observe the implementation of Scrum in the everyday practice of the development team and ask questions directly when they popped up.

C. Subjects

The method used for subject sampling was based at quantification of both duration and cost of a repository of 345 finalized IT projects that were collected over time by members of three measurement teams within three different organizations. For all inventoried projects size was measured in Function Points (FPs). Project duration (measured from the start of the project initiation to technical Go Live and project cost (all project related cost excluding investments (e.g. software license costs) were measured. We focused the research at the characteristics that could explain the over-performance of a specific group of small software releases within this repository.

D. Objects

The research itself focused at 26 software performed on two Finance & Risk applications in the same organization, that all but one performed as Good Practices, meaning that both duration and cost were assessed to be better than average. One release performed better than average for project cost, yet the duration took longer than average. The reason for this relatively longer duration was the relatively small size of 5 FPs.

E. Instrumentation and data collection

All projects in the repository were collected and analyzed according to the so-called Five Core Metrics method [14]. This method is closely related to the SEI Core Metrics, a standard set of process-based software engineering metrics [15] [16] (Size, Duration, Effort/Cost, and Defects/Incidents). For recording and analyzing purposes the QSM SLIM-toolset was used.

F. Analysis procedure

In the analysis of the sample data trends and specific effects with regard to release size, time, and cost will be inventoried.

TABLE I. OVERVIEW OF RELEASE CHARACTERISTICS

Small software releases in sample	Projects (total sample)
Very limited Project Governance	Project Governance acc. to Prince2
Expected Cost max. 100K Euro	Planned Cost > 100K Euro
Expected Duration < 3 Months	Planned Duration > 3 Months
Fixed, experienced team (Level Load)	Mixed team-size and composition (Usually Rayleigh Load)
Release-based: steady heartbeat (monthly Go Live)	Once Go Live at end of Project
Single application mapping	Single or multi-application mapping
Close feedback loop to business	Feedback loop based at governance

IV. EXECUTION

A. Sample

All 26 analyzed software releases where performed during the period from 2011 to 2012; all releases that were performed during the period of August 2011 to August 2012 were incorporated in the research sample; no delivery activities were left out. Both sets of releases were performed within an IT-department that delivered solutions at group level of a large bank. All releases were performed on two Finance & Risk systems. One release was always mapped on one system (single application mapping). For each system a fixed team of experienced software developers was in place. Both teams worked separately from each other, each at their own location and for different business departments. Development was performed in-house with short feedback-loops with business representatives (the business department for both teams was working within walking distance from both development teams).

During the 13 months that were subject of the collected data, both teams delivered a set of software solutions on a monthly basis (consisting of new developments and enhancements on existing software and maintenance). This heartbeat was tight: once a month a 'Go Live moment' was scheduled and during each month two sprints (or iterations) of each two weeks were performed. Both development teams transformed during the measurement period from a more or less plan-driven delivery model (waterfall) towards an agile approach (Scrum).

A characteristic of the applicable software release-teams was that (compared to non-release-based software projects) only limited project governance was in place. Approval from a project board for each separate release was unnecessary. Instead once every year a budget was approved to have a release-team working for one year. Because formal Go/No-Go milestones where not applicable no formal project documents for every release (e.g. Project Brief, Project Initiation Document) where needed. Instead a decision document was prepared once a year to have the budget for a yearly release-team approved by the senior management. The average team-size of both enhancement teams was 3 FTE each; approximately 5 people working in each team. Here an important difference with non-release-based projects is obvious; the team-size stays at the same level (3 FTE or 5 people) during the whole year (also referred to as Level Load), where in non-release-based projects usually a Raleigh shaped build-up is applicable (also referred to as Raleigh Load) [14]. These two teams were clearly much more stable than regular project teams, both in number of people as well as the specific team members. Each team was coor-

Fig.1 Cost per Size Unit over Time of Release-Team A.

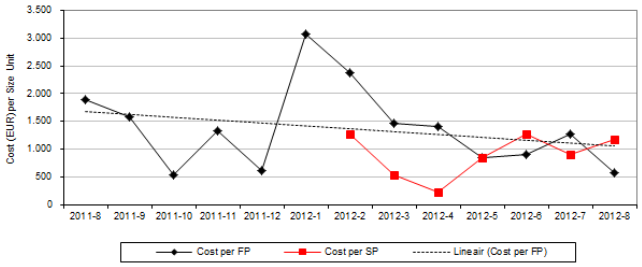


Fig. 2 Cost per Size Unit over Time of Release-Team B.

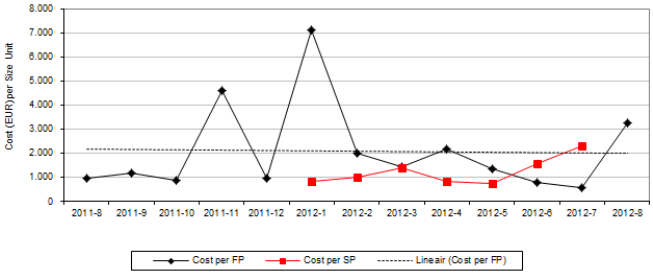


TABLE II. DATA PER SMALL SOFTWARE RELEASE OF SYSTEM A

Release ID	#Req	#FP	#SP	Duration	Cost (EUR)	Cost/FP	Cost/SP
A 2011-08	16	22	-	2.50	41,790	1,900	-
A 2011-09	8	26	-	2.48	41,034	1,578	-
A 2011-10	13	49	-	2.48	26,586	543	-
A 2011-11	14	31	-	2.48	41,370	1,335	-
A 2011-12	25	47	-	2.48	29,358	625	-
A 2012-01	7	16	-	2.54	49,266	3,079	-
A 2012-02	39	14	26	2.48	33,222	2,373	1,278
A 2012-03	20	21	57	2.50	30,912	1,472	542
A 2012-04	14	13	75	2.48	18,228	1,402	243
A 2012-05	18	48	48	2.50	40,530	844	844
A 2012-06	11	41	29	2.48	37,002	902	1,276
A 2012-07	22	32	45	2.48	41,139	1,286	914
A 2012-08	26	55	27	2.50	31,878	580	1,181

TABLE III. DATA PER SMALL SOFTWARE RELEASE OF SYSTEM B

Release ID	#Req	#FP	#SP	Duration	Cost (EUR)	Cost/FP	Cost/SP
B 2011-08	15	52	-	2.50	49,728	956	-
B 2011-09	13	28	-	2.48	33,096	1,182	-
B 2011-10	9	30	-	2.50	26,586	886	-
B 2011-11	12	9	-	2.48	41,370	4,597	-
B 2011-12	4	30	-	2.48	29,358	979	-
B 2012-01	4	5	43	2.54	35,537	7,107	862
B 2012-02	10	18	35	2.48	35,957	1,998	1,027
B 2012-03	7	24	25	2.50	34,823	1,451	1,393
B 2012-04	9	20	51	2.48	44,021	2,201	863
B 2012-05	5	25	45	2.50	33,899	1,356	753
B 2012-06	10	41	21	2.48	32,849	801	1,564
B 2012-07	8	57	14	2.48	32,429	569	2,316
B 2012-08	5	10	-	2.50	32,765	3,277	-

dinated by a coordinator that later performed the role of Scrum-master, and three developers that performed all applicable build and test activities. Once the transformation to Scrum took place the team was enforced with a Product Owner; a representative from the business that worked together with the team on the stories that build the ready backlog and the sprint backlog. A daily standup meeting took place every morning where all team-members (including the Product Owner) where present.

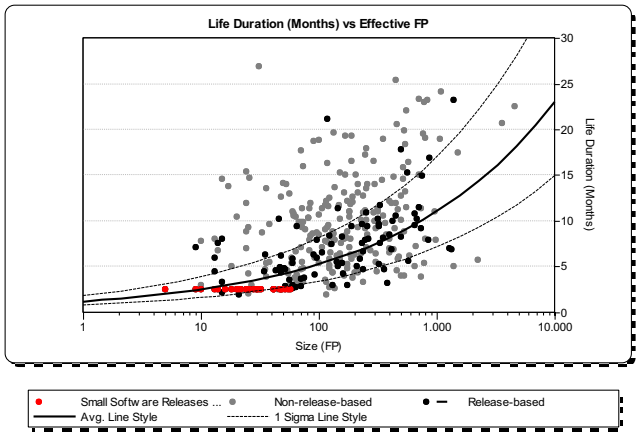
B. Data collection performed

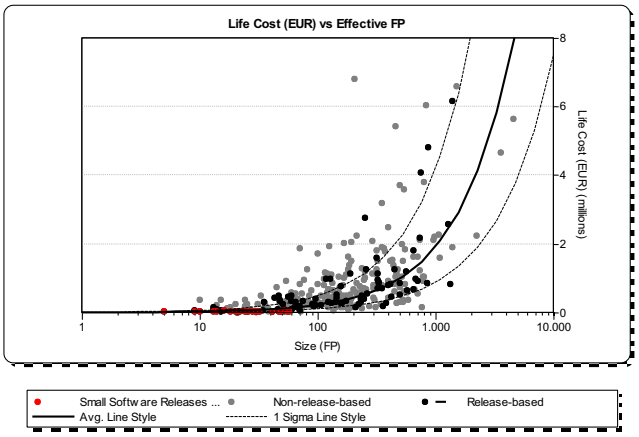
Data was collected during a data collection period from three months at the end of the 13 month measurement period. Data was collected in a structured way: a measurement team member inventoried project data on a data collection form holding project information on among others project name, project manager data, size in FP of the project or release, effort spent, project cost, project duration from Project Initiation to technical Go Live in Months, number of defects found from System Integration Test to User Acceptance Test and number of incidents occurring during the first months after Go Live.

The implemented size of the analyzed small software releases was measured in a number of function points (FPs), based at the set of functional requirements that was prepared for each separate release. For all releases that where performed according to a Scrum delivery approach (2012-01 and on) the delivered size was measured in a number of story points (SPs) too. The SPs where measured by the release-team itself.

Once a draft Project Close Report was prepared this was discussed by the measurement specialist with the responsible project manager. During this interview the project manager was asked to come up with an inventory of reasons that could explain the success or failure of the project. These reasons were in cooperation with the measurement specialist translated to one or more keywords. The reasons and the keywords were included in a final version of a Project Close Report.

Tables II and III give an overview of the number of requirements per release, in relation to the size measured in FPs and in SPs. Looking at both inventories there's no clear coherence between the number of requirements, the number of FPs and the number of SPs. In a number of cases requirements were only about non-functional aspects, not leading to any FPs. However, these requirements could have led to SPs. And on the opposite, some requirements did result in many relatively FPs, where the release-team did probably not measure a lot of SP's. Overall can be said that this analysis does not give any evidence on a correlation between the number of requirements, FPs and SPs, however a connection between the content of a





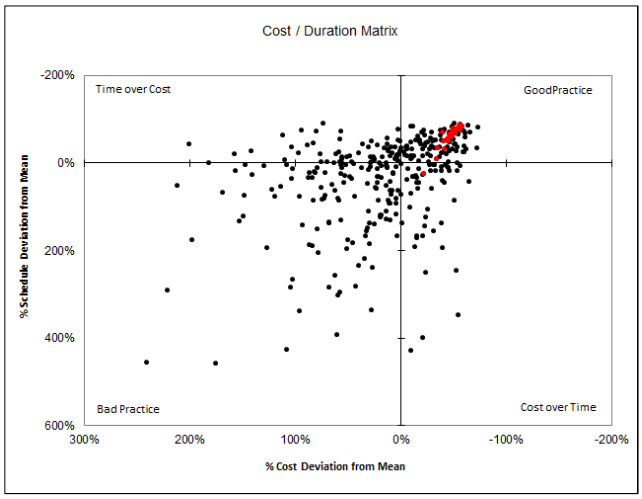


Figure 3: Two plotter charts representing Size (FP) versus Duration (Months) (upper left) and Size (FP) versus Cost (Euros). The software releases that are research subject of in this paper are printed in red. A Cost / Duration Matrix (lower left) shows the deviation of projects and releases within the repository from the average, again the releases that are subject of the research in this paper are printed in red. The graph shows that all but one (a release that was measured as 5FPs of Size) of the subjected releases did score better than average on both Duration and Cost and therefore are categorized as a Good Practice.

requirement and the number FPs is obvious (these can be deducted according to the applicable FSM method).

C. *Validity procedure*

All data used in the research was collected by one or more measurement specialists from several measurement teams within three organizations. The data for the 26 small software releases was collected by an experienced measurement specialist that also performed a Function Point Analysis (FPA). FPA was performed based at sets of final release documentation that were delivered by the project manager. A review on the FPA was performed by an experienced FPA specialist from the organizations measurement team. All FPAs were performed according to NESMA functional size measurement method conform ISO/IEC 24570 [17].

After data collection and performing a FPA for every release a draft Project Close Report was prepared, showing the inventoried data per release and an analysis of time, cost and quality versus size (in FP). In this report the performance of a specific release was compared to the overall performance trends of the organization.

V. ANALYSIS

A. *Descriptive statistics*

As shown in Figure 3 all but one of the 26 software releases performed better than average when measured on size versus duration and cost. One release did not perform as a Good practice. However release cost were below average, the duration per FP of this release was longer than average, caused by a relatively small size of 5 FPs. For the analysis of the 26 releases subsequently the effects on size, time and cost are discussed.

B. *Effects on Size*

Looked at from a functional size point of view all software releases in the analysis (see Tables II and III) can be characterized as small; they ranged from 5 to 57 FP, the mean was 29 FP, the median was 27 FP. The mean number of requirements within one release was 13; ranging from 4 to 39 requirements per release. The content of the requirements varied from major non-functional (e.g. performance improvement of functions or solving a security issue) to building and implementing new user functionality. Requirements were beforehand in close cooperation with representatives of the business (e.g. the Product Owner) documented in so-called user stories [4] in a standard Scrum format (e.g. “as a user of the risk analysis functionality of System A, I need to see monthly reports on usability of the system, in order to steer properly on system security…”). These stories were assessed to be at a too high level for a function point analyst to properly count the size of the requirement (or the story).

Once a user story was put on the backlog for a specific release (e.g. the Product Owner decided to have the specific requirement implemented in a release) the story was documented more in detail by one of the release-team members.

TABLE IV. SUMMARY OF CORE METRICS AND PERFORMANCE INDICATORS

	Releases System A and System B					Total Metrics Repository					
Metric	**A**	**B**	**A+B**	**A+B Structured**	**A+B Scrum**	**Repository at 29 FP**	**Release-based**	**Non-Release-based**	**Structured**	**Scrum**	**Projects (all)**
Number of Projects (n)	13	13	26	10	16	-	107	238	300	45	345
Throughput (FP)	415	349	764	324	440	-	24,204	66,700	77,976	12,928	90,904
Average project size (FP)	32	27	29	32	28	29	226	276	257	287	261
Avg. duration (Months)	2.49	2.49	2.49	2.49	2.49	3,59	6.60	10.16	9.50	6.14	9.07
Avg. effort (PHR)	423	423	423	439	409	6,700	5,593	7,709	7,223	5,960	7,098
Avg. cost (EUR)	35,563	35,571	35,567	36,859	34,346	72,540	590,316	832,440	794,064	515,969	761,620
Avg. Time-to-Market (Days/FP)[d]	2.37	2.82	2.58	2.33	2.76	3,64	0.89	1.12	1.13	0.65	1.07
Avg. Productivity (Cost/FP) [d]	1,114	1,325	1,210	1,112	1,883	2,501	2,610	3,020	3,096	1,795	2,924

[a.] Time-to-Market and Productivity are calculated as a weighted average, with Size as the weighting factor. Time-to-Market is measured in a number of Calendar days per FP.

After building and testing was finalized sometimes an update on these functional documents was performed. In practice these documentation actions were performed by the same team member that did the building and a part of the testing of a requirement. An assessment on the quality of this functional (and often technical) set of documentation showed that a function point count (in the case of this analysis a so-called 'global count' [17]) could be performed without too many assumptions.

Table IV shows the throughput that was delivered by each release-team on a yearly basis was approximately between 330 and 400 FP. However, due to a steady heartbeat this throughput was not delivered in one time to the business yet in small, monthly portions.

C. Effects on Time

All small software releases from the analyzed sample were performed according to an equal schedule. In a so-called Ready stage the Product Owner was building the backlog for the coming two sprints. This ready stage always took about a month and was performed while the enhancement team was working on two sprints of the actual, ongoing build and test stage. When needed members of the release-team supported the Product Owner by working out specifications for the upcoming sprints, based at user stories. At a fixed date of every month (usually the 21st of every month) the release- team started the first sprint of a new release, followed after two weeks by a second sprint. At the end of each sprint a demo was given to business representatives. The constructed solutions were tested (UAT and a Production Test) and deployed once every month (usually in the third week of each month and made available for end users of the systems (Go Live). After Go Live the release-team was for about two weeks available for aftercare work, while they already were working on a next release. On average the release duration was 2,5 months, ranging from 2,4 to 2,6 months.

When looking at the upper left plotter chart in Figure 1 it is obvious that all analyzed small software releases took the same duration of approximately 2,5 months. The graph on duration shows that the releases form a horizontal line-shape due to the varying size in FP on the horizontal axis and the duration that's almost equal for all releases on the vertical axis.

Size was measured in both FPs and SPs. Looking at the outcome of these measurements (see Figures 1 and 2 and Tables II and III) we see that no clear coherence is visual between size in FP and size in SP. As a result of this no cohesion between Cost/FP and Cost/SP is seen.

D. Effects on Cost

Figure 2 and 3 give an overview of the cost per size unit of the small software releases over a period of thirteen months. Cost per size unit is expressed in two metrics here; Cost per Function Point (Cost/FP) and Cost per Story Point (Cost/SP).

E. Scrum versus plan-driven

Driven by the transformation from a plan-driven (waterfall) delivery approach to an agile delivery approach (Scrum), we compared the outcome of the performance analysis of small software releases that were performed in a structured way with those that were performed according to Scrum. Table IV shows that both Time-to-Market and Productivity are on average better within a structured approach (resp. 2.33 Days/FP and 1,112 Euro/FP) than within Scrum (resp. 2.76 Days/FP and 1,883 Euro/FP).

F. Data set reduction

No data was excluded from the dataset. Exclusion of outliers was not applicable for the analyzed release and project data.

G. Hypotheses testing

Table IV shows that the (weighted) average Time-to-market (in Days/FP) of the analyzed sample releases (2.58 Days/FP) is not better than the Time-to-market of the 107 release-based projects from the repository (0.89 Days/FP) and the (weighted) average Time-to-market of non-release-based projects from the repository (1.12 Days/FP). Hypothesis H1(1) (Time-to-market (in Days/FP) of 26 releases was better than (weighted) average Time-to-market in repository) – when measured as a comparison with the Time-to-market of the whole repository - can be concluded not to be true.

However when the Time-to-market of the analyzed sample releases is compared with release-based and non-release-based projects from the repository with a corresponding average size of 29 FPs this view is different. Average Time-to-market at 29 FPs in the repository is 3,64 Days/FP. When compared to releases and projects from the same size category (small releases) the Time-to-market of the sample releases is significantly better. Hypothesis H1(2) can be concluded to be true in that case.

Table IV shows that, opposite to our conclusion on Time-to-Market, the (weighted) average Productivity of the analyzed small software releases (1,210 Euro/FP) is significantly better than the (weighted) average Productivity of other release-based projects (2,610 Euro/FP) and of non-release-based projects from the repository (3,020 Euro/FP). Therefore hypothesis H2 (Productivity (in Cost/FP) of 26 releases was better than average productivity in repository) can be concluded to be true. This is applicable for both H2(1) and H2(2).

We conclude that, looked at from the point of performance indicators, the transformation to Scrum did not lead to better results. Hypothesis H3 - Performance of Scrum releases (in Days/FP and Cost/FP) was better than plan-driven releases (waterfall) – can be concluded not to be true. However, we did not measure any data on user experience (e.g. user satisfaction) or delivered value; maybe a positive effect of the transformation can be found here. In addition to our remark on future research on the subject of user experience in relation to Time-to-Market, we feel that there is a valid ground for additional analysis on user satisfaction combined with added value to the business and end-user with regard to plan-driven versus Scrum.

VI. INTERPRETATION

A. Evaluation of results and implications

A valid explanation for the outstanding performance of both release-teams might be concluded from the analysis showing that they work according to a steady monthly heartbeat, resulting in almost perfectly the same release durations over time. Table IV shows that the average Time-to-Market of the small software releases (2.33 Days/FP) is not as good as the Time-to-Market of other software releases (0.89 Days/FP) in the repository that is used for comparison; yet the steady heartbeat of the small software releases makes that end-users get a portion of new and adjusted user-functionality every month instead of all-in-once after on average 6.6 months. However, on a yearly basis the delivered throughput of the analyzed small software releases (330-400 FP) is higher than the average throughput of other software releases (226 FP).

Furthermore it is worth mentioning that we found (during the 2 month observation period when we worked closely to the teams) that both teams already worked with many practices that are considered Agile before the 'official' transformation to Scrum was made. The major additions from the introduction of Scrum were limited to a better defined role of the Product-owner, the use of Story Points (SPs) as a size metric for estimation purposes, and a more compelling approach on daily standup meetings (before that daily meetings were already present, yet less standardized and formal). The typical Agile practices that these teams already performed before the transformation were:

1. the scope of each team was limited to one single application;
2. both teams were stable and experienced;
3. a planning and estimate capability that was based at a common responsibility of the development-team;
4. a close feedback-loop between user and supplier existed, however rather informal;
5. a release-based way of working was implemented;
6. both teams were delivering developments, enhancements and maintenance according to a steady, fixed heartbeat.

All these aspects are mentioned in the Agile Manifesto [18]. As such, we must be honest that these two teams cannot be considered as a representative example to measure whether a transformation of Scrum gives measurable benefits.

The question can be raised whether the choice of a delivery model is an important success factor with regard to small software releases. Since the outcome of the analysis does not show significant improvements after transformation to Scrum, one might think this is not the case. However, we conclude that both release-teams already before the transformation worked release-based mapped on a single application; they already combined a fixed and experience release-team with a close feedback-loop with end-users, and delivered according to a steady, monthly heartbeat. These success factors are in fact already part of a Scrum approach and showed to be determent for a good performance. Our data did not reveal additional benefits after the transition. Formalizing Scrum for these two teams did not cause any new success factors for both teams within the available data.

Probably the transformation can be assumed to be driven by a wish of the senior management to connect to the ongoing concern-wide advertising campaign on "Scrum-as-the-promise-for-the-future"; not realizing that they actually were doing so already to a large extend, without the formal label "Scrum" put on it.

B. Limitations of study

This paragraph discusses threats to validity; where internal validity refers to whether the study supports the findings. The data used in this research was collected by multiple teams of measurement specialists from various IT organizations. Data was initially derived from formal project administrations which also formed the basis for management reports and enriched by review by project managers and release-team managers of the assessed projects and releases. For the measurement of the size of the project, use was made of a formalized ISO FSM standard [17]. However within the organization was experimented with various size measures FP were chosen used as the least bad alternative [19]. Function Points are independent from programming languages, software engineering processes, delivery models and sourcing strategies, and therefore a good source for cost estimations and benchmarking [20]. Function points are generally consistently and can be used with an acceptable degree of accuracy [21] [22]. The function point analysis for this research was performed by highly trained and experienced specialists, in many cases a CFPA certified expert. The FPA on

the specifically analyzed small software releases was performed by an experienced FPA specialist and reviewed by two IT-metrics team-members (one of them certified CFPA).

The applied primary programming language for both release-teams was Oracle. However, due to the fact that the measurement repository of 345 projects that we used for the analysis did not contain enough comparable projects or releases (most were using programming languages like Java, .Net and COBOL) we could not include the technical environment of the software releases in our analysis. As such, we cannot formally exclude that the higher performance was only technology driven, though this seems unlikely.

C. Inferences

With regard to generalization of our research on the performance characteristics of small software releases given the findings and limitations, we looked at a subset of solution delivery activities performed within one specific large banking organization. As such, our findings are limited in their external validity. Future research within other companies, and other delivery environments, is needed to support generalization of our findings.

With regard to the used research method; our research showed that the used measurement and analysis method supports performance measurement in different delivery models (e.g. structured and scrum). We experienced the model to be a good tool for an organization that strives to continuous improvement. In the studied organization, whereas in many other companies ongoing judgment of the overall performance of suppliers is key, the objectivity and transparency of the method is still appreciated as a valuable tool by CIO's, business managers, senior IT managers, portfolio managers, and procurement officers. We therefore think our approach can be valuable for other organizations too and can work as an example.

VII. Conclusions and Future Work

A. Relation to existing evidence

In this paragraph the contributions of the research in the context of other studies is described.

Our research did not show significant improvements on the performance once transformed from structured to Scrum as was found by Sutherland et. Al: [4]; the Scrum releases in our research even resulted in an on average lower productivity (higher Cost per FP) and a longer Time-to-Market.

Dybå and Dingsøyr [6] conclude that in particular, agile project management methods, such as Scrum, warrant further attention. They see that there is a backlog of research issues to be addressed: a remark that matches the outcome of our study.

Estler et. al. [8] concluded that choosing an agile rather than a structured approach does not appear to be a crucial decision for globally distributed projects, a finding that corresponds with the outcome of our research, however in our case with regard to small software releases.

As discussed by Tan et. al. [10] our research did not incorporate risk behavior and risk mitigation as an aspect of release-based software delivery, this might be an interesting point to address for future research; "In what way do small, iterative software releases help mitigating risks?"

Premrai et al. [11] concluded that new projects tend to be significantly larger in size than maintenance projects; a finding that resulted from our research too; the average size of release-based projects in the total measurement repository (usually with a strong focus at enhancements and maintenance) is 226 FPs, where the average size of non-release-based projects (usually containing a lot of new development) is 276 FPs.

The outcome of our study seems to emphasize the conclusion of Boehm [12] that small, iterative enhancement releases show best-in-class performance results. The effects of economy of scale do result in a better-than-average productivity and however time-to-Market is somewhat higher than average, this is in practice corrected due to a steady monthly heartbeat.

Top et al. [13] state that primary inputs of an estimation model are software development effort and software size. This aspect seems different in a situation like in our research; a fixed team (working according to a level load staffing approach) does not lead to major variations in time, effort and cost. The only variable here is size. Our research showed that an experienced release-team is quite capable in estimating and planning their work by performing planning poker sessions prior to every sprint or release. One might say that in a situation like this estimating did become a release-team capability. Analyzing the performance afterwards, including benchmark the performance against internal and external peer groups, is not part of the process that's performed by a release-team; to do so a specific measurement & analysis capability needs to be implemented in a solution delivery organization. For this purpose SPs are assumed not to be a suitable size metric due to the fact that this metric is not fully objective and may differ over teams. However we think that future research might help to get a clearer look at this issue.

B. Impact

In this study we examined the performance of a series of 26 small software releases, that were performed on two Finance & Risk systems within a large banking organization during a period of thirteen months. The performance of the small software releases was compared with a measurement repository of 345 projects that were performed within three different organizations over a period of four years. We were able to determine several factors that determine why the analysed small software releases are assessed as best-in-class.

Overall, we found that five success factors, all about reduction of complexity of the software development environment, were determent for the best-of-class performance of the analysed small software releases. Four of these factors were assessed in earlier research we performed, the fifth was found specifically as a result of the analysis we performed for this paper, and was about a steady, monthly heartbeat. Once every month a Go Live moment occurred; the average duration of the releases was 2.49 months, with a negligible standard deviation. Summarized, we found that projects that were able to:

1. limit the scope of the project to a single application,
2. work in a series of releases (touching a single application at a time),

3. work with stable (global) software teams to work for longer periods together,
4. increase close cooperation to establish a short feedback loop between user and supplier, and
5. delivered solutions according to a steady, monthly heartbeat,

performed measurably better than other projects, especially where it comes to Time-to-market and Productivity. Besides that, we found that in these specific circumstances a transformation from a plan-driven delivery approach (waterfall) to Scrum did not result in any performance improvements within the available data. Furthermore, one can argue that the above five factors are all found almost literally within the practices and principles of the Agile Manifesto [18].

C. Limitations

Planning and estimating of releases was performed by the release-team itself, based at the results of one or more planning poker sessions [23]. Unfortunately for the analysis no data was available on both planned FPs or SPs at the beginning of a release.

The choice of the senior management to use these small software releases as a research case to justify a company-wide future transformation from structured to Scrum seems to be not very well picked. Small software releases show a different performance pattern from other projects, both release-based and non-release-based. The steady, monthly heartbeat, fixed and experienced release-team, short but steady duration, approximately equal cost per release are characteristics of small software releases, but do not match new development projects at all. As such, the selected two teams are a priori already weakly representative for the majority of development projects in this specific organization.

Limited data was used for the paper. We accept that the choice of the senior management to use these small software releases as a research case to justify a company-wide future transformation from structured to Scrum seems to be not very well picked. This might question the reliability of the limited data.

Our research showed that the formal transition to Scrum for these two release-teams were limited; the differences in performance outcome within the available data are negligible. Our interpretation is that both release-teams already worked in a 'Scrum *avant-la-lettre*' kind of way and that the proposed transformation was a more or less formalized implementation of what they already did to a large extend. In addition did the idea to source parts of the work to a low-wages country (e.g. India) ignored that globally distributed Scrum teams usually face a number of challenges as project distribution impact on communication, coordination and collaboration processes [24].

D. Future work

We did not collect any data on the user experiences (e.g. user satisfaction) with regard to small software releases, yet an assumption is that the end-user might have the feeling that, due to the steady monthly heartbeat of small software releases, the Time-to-Market is quite good compared to releases that deliver all-in-once after more than 6 months. This is, clearly, an interesting subject for future research.

ACKNOWLEDGMENT

We thank Hans Jägers, professor emeritus from the University of Amsterdam and all reviewers for their valuable contributions. In addition, we thank Hans Vonk, Director of QSM-Europe, for the provision of the QSM SLIM tools for analyzing purposes.

BIBLIOGRAPHY

[1] A. Jedlitschka and D. Pfahl, "Reporting guidelines for controled experiments in software engineering," *Proceedings of the 4th International Symposium on Empirical Software Engineering,* no. Noosa Heads, pp. 95-104, 2005.

[2] D. Rodriguez, M. Sicilia, E. Garcia and R. Harisson, "Empirical findings on team size and productivity in software development," *The journal of Systems and Software,* vol. 85, no. Elsevier, pp. 562-570, 2012.

[3] L. Putnam, "A general empirical solution to the macro software sizing and estimating problem," *IEEE Transactions on Software Engineering,* vol. 4, pp. 345-361, 1978.

[4] J. Sutherland, A. Viktorov, J. Blount and N. Puntikov, "Distributed Scrum: Agile Project Management with Outsourced Development Teams," in *40th International Conference on System Sciences*, Hawaii, 2007.

[5] C. Jones, Software, Assessments, Benchmarks, and Best Practices, New York: Addison Wesley Longman, 2000.

[6] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Information and Software Technology,* vol. 50, no. Elsevier, pp. 833-859, 2008.

[7] K. Sureshchandra and J. Shrinivasavadhani, "Adopting Agile in Distributed Development," *IEEE International Conference on Global Software Engineering (ICGSE),* pp. 217-221, 2008.

[8] H. C. Estler, M. Nordio, C. A. Furia, B. Meyer and J. Scheider, "Agile vs. Structured Distributed Software Development: A Case Study," in *IEEE Seventh International Conference on Global Software Engineering (ICGSE)*, Porto Allegre, Brasil, 2012.

[9] N. Ramasubbu, M. Cataldo, R. Balan and J. Herbsleb, "Configuring Global Software Teams: A Multi-Company Analysis of Project Productivity, Quality, and Profits," *IEEE International Conference on Software Engineering (ICSE),* vol. ACM, pp. 261-270, 2011.

[10] T. Tan, Q. Li, B. Boehm, Y. Yang, M. He and R. Moazeni, "Productivity Trends in Incremental and Iterative Software Development," in *IEEE Third International Symposium on Empirical Software Engineering and Measurement*, 2009.

[11] R. Premrai, M. Shepperd, B. Kitchenham and P. Forselius, "An Empirical Analysis of Software Productivity Over Time," in *IEEE International Symposium Software Metrics*, Como, Italy, 2005.

[12] B. Boehm, "A View of 20th and 21st Century Software Engineering," in *IEEE International Conference on Software Engineering (ICSE)*, Shanghai, Ghina, 2006.

[13] Ö. Ö. Top, B. Ozkan, M. Nabi and O. Demirörs, "Internal and External Software Benchmark Repository Utilization for Effort Estimation," in *Joint Conference of the 21st Int'l Workshop on Software Measurement, and 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA)*, Nara, 2011.

[14] L. Putnam and W. Meyers, Five Core Metrics, The Intelligence Behind Succesfull Software Management, New York: Dorset House Publishing, 2003.

[15] CMMI Product Team, CMMI for Development, Version 1.3, Hanscom: Carnegie Mellon, 2010.

[16] S. Kan, "Metrics and Models in Software Quality Engineering," Addison Wesley Longman, Reading, Massachusetts, 1995.

[17] NESMA, NESMA functional size measurement method conform ISO/IEC 24570, version 2.2, Netherlands Software Measurement User Association (NESMA), 2004.

[18] e. a. Beck, “Manifesto for Agile Software Development,” 2012. [Online]. Available: www.agilemanifesto.org.

[19] P. Rahul and S. Martin, "An Empirical Analysis of Software Productivity Over Time," in *IEEE International Symposium Software Metrics*, Como, Italy, 2005.

[20] E. Kulk, IT Risks in Measure and Number, Amsterdam: Free University, 2009.

[21] C. Kemerer, "Reliability of Function Points Measurement - a Field Experiment," *Communications of the ACM,* vol. 36(2), pp. 85-97, 1993.

[22] C. Kemerer and B. Porter, "Improving the Reliability of FunctionPoint Measurement: An Emperical Study," *IEEE Transactions on Software Engineering,* Vols. SE-18(11), pp. 1011-1024, 1992.

[23] M. Cohn, Agile Estimating and Planning, New york: Pearson Education, 2006.

[24] E. Hossain, M. A. Babar and H.-y. Paik, "Using Scrum in Global Software Development: A Systematic Literature Review," *IEEE International Conference on Global Software Engineering (ICGSE),* vol. DOI 10.1109, pp. 175-184, 2009.