Measuring and Monitoring Agile Development Status

Martin P. Boerman[1], Zeeger Lubsen[1], Damian A. Tamburri[2], and Joost Visser[1,3]
The Software Improvement Group[1]
Department of Computer Science, Faculty of Sciences, VU University Amsterdam[2]
Institute for Computing and Information Sciences, Radboud University Nijmegen[3]
{m.boerman, z.lubsen, j.visser} @sig.eu[1]
d.a.tamburri@vu.nl[2]

Abstract— **Applying agile methods in large, and complex organizations requires effective status reporting to external stakeholders in order to facilitate communication, inspire confidence, and allow appropriate project steering. We defined a practical model for measuring and reporting the quality, progress, and predictions of agile development towards stakeholders that are not involved in the actual development activity. To design this metrics-based reporting model, we used the Goal-Question-Metric (GQM) approach to derive a set of 11 metrics from the goals of project owners. The selection of this perspective was part of the GQM process. The defined metrics in our model are measured using mostly product backlog data. We validated the reliability, usefulness, and feasibility of the model with a case study. We show the practical value of the individual metrics, and of the monitoring method as whole.**

Keywords – monitoring; agile; product backlog; project owner; quality model

I. INTRODUCTION

Over a decade after their proclamation in the agile Manifesto [1], agile software development methods have gained widespread adoption, with as many as 88% of organizations practicing agile in some form and as many as 57% doing so with 5 or more agile teams [2].

Adoption of agile methods is not without challenges. Among the most often reported concerns are lack of management control, management opposition, and lack of predictability [2]. Often cited critical success factors are effective communication between the team and its environment [3], [4] and appropriate involvement of the "customer" (i.e. users and/or project sponsors) [5].

Ineffective status reporting for agile development activities may be a root cause underlying reported difficulties in adopting agile methods. When external stakeholders (i.e. those not involved in the actual development activities) do not have a clear understanding of the current status of an agile project, this leads to a lack of confidence regarding successful completion, followed by attempts to re-introduce traditional, plan-based control mechanisms and subsequent retreat from agile principles and practices.

To counter this tendency, we set out to devise a reporting mechanism that – while respecting agile principles, such as the self-organizing nature of agile teams and the positive attitude towards changing requirements – provides external stakeholders with insight into, confidence in, and sense of control over agile development activities.

We first used the Goal-Question-Metric (GQM) method to derive a set of metrics from the goals of project owners (i.e. project sponsors), based on a literature review on project success factors. We then validated the resulting measurement and reporting model in a case study at a Dutch government organization that was in the process of renewing their core service according to Dutch law changes. The focus of this validation was assessing each metric's practical feasibility, usefulness, and reliability.

The remainder of this paper is organized as follows. In section II we will give background information about the adoption of agile development methods and practices. In section III we describe common issues emerging in agile development. Then, in section IV we elaborate on monitoring agile development using product backlogs. Section V describes the appliance of our method in a case study, after which we discuss limitations and some lessons learned in section VI. Finally in VII and VIII we respectively present our conclusions and future work.

II. BACKGROUND

Agile development has been, and currently is being, adopted by many organizations [2]. The adoption of agile practices can lead to improvements in quality, requirements management and both customer satisfaction as well as team satisfaction [6].

The term 'agile' is an umbrella term for software development methods that are organized in such a way that they adhere to the agile principles [1]. There are thus multiple methods that can be called 'agile'. Each method has its own characteristics, process implementations, and tools to support the development process. In practice, organizations often choose to combine parts of different methods.

In Scrum, which is a widely used agile methodology [2], a so-called *product backlog* is applied to manage work items. A survey across more than 3,500 individuals from organizations [2], showed that 73% of the respondents used Scrum, or Scrum-variant agile methodologies. This supports the observation in the field that agile projects very often include the usage of a product backlog.

A product backlog is a prioritized list of work items that have to be completed in order to complete the end product. We refer to these items as *product backlog items* (PBIs). The PBIs are preferably weighted in terms of needed effort. Aside from that, PBIs are expected to have a description and a certain "status", i.e. its completion level.

To maintain the product backlog, organizations often use undedicated tools. In this case, by undedicated tools we mean tools that have not been designed to maintain a

backlog with (e.g. Microsoft Excel, or whiteboards). Dedicated tools specifically focus on digitally managing an agile environment as a whole, in this case including support for product backlogs. Examples of such tools include JIRA, VersionOne, and Microsoft Project.

III. Problem Statement

During agile development, teams are supposed to track their progress and update their planning. Contact between developers, team leaders, product owners, and other stakeholders, is meant to steer the continuous process of designing, planning, and implementing PBIs.

Overall, agile development has shifted responsibilities of SD projects to project owners. The nature of agile development calls for a heavy focus on continuous synergy between internal and external stakeholders. Although most organizations are aware of this need, and are using agile methods and tooling to improve this synergy, project owners often still sense a loss of control [5][2].

Earlier studies have stressed the importance of collaboration, communication, and status reporting in order to increase the chances of project success [3][4]. This is in line with the high adoption of agile methods, which seems to have caused a number of agile-specific problems and difficulties whilst at the same time addressing issues related to traditional development methods [6]. In fact, five of the top six reasons for failure given in [6] had to do with communication, which is something agile development specifically relies on.

By relying more on face-to-face communication, rather than on documentation, 'being agile' can be a burden, instead of an improvement during SD activities. Over time, the increasing complexity of the matter at hand can push communication to its limits, or even over them.

Projects owners should thus play a very active role in the development process, which is one of the factors that can lead to a successful project outcome [5]. At all times it is important for a project owner to have confidence in the developers, and to believe that the project will indeed have a successful outcome.

The absence of the project owner's confidence (which could be the result of bad communication) leads to dissatisfaction and even more communicational issues with developers, resulting in a worsening relationship [7]. In short, communication between project owners, developers, and other stakeholders, is calling for an improvement so that disagreements, unclarities, poor status reporting, and lack of feeling in control can be addressed.

To address the aforementioned problems, we devised a model to monitor agile development processes. We wanted to create a practically feasible, reliable, and useful model that would be generally applicable in agile environments, based on commonly available data. The resulting monitoring method is meant to support communication between developers, managers, and sponsors. Ultimately it should provide external stakeholders (specifically project owners) with an improved sense of control, by providing objective analyses that support decision-making, while respecting agile principles.

During our study, the main research questions were:

1. Can we derive a measurement model based on the main goals of agile development project owners?
2. Is the measurement model practically feasible, and are its results reliable, and useful in practice?

IV. Monitoring via Product Backlogs

In section II we addressed the widespread use of product backlogs in agile development environments. Organizations have various approaches when it comes to implementing product backlogs, and these approaches can be divided into two categories.

The first category covers the "undedicated tools" (e.g. Microsoft Excel or whiteboards). By undedicated we mean that the tool in question was not originally designed to maintain a product backlog. In practice this means that the tool requires a manual implementation of the backlog, which leads to an organization-specific backlog instance, often without any ownership or substantive constraints.

Such implementations perform poorly, if at all, at providing historical data, or at ensuring data integrity and reliability. A resulting problem from this way of working can be the inability to properly communicate the project's status between stakeholders (either internally or externally).

The second category involves "dedicated tools" (i.e. JIRA, VersionOne, and Microsoft Project). These tools often provide standardized backlog functionality, with additional customizations within certain constraints. Often, the backlog functionality is only a portion of what the tool has to offer, meaning that the same tool supports a plethora of other agile practices and artifacts, each with their own set of customizable features.

All in all, dedicated tooling is more reliable than undedicated tooling. However, because of their extensiveness, high customizability, and detailed reporting features, dedicated tools can cause their own communicational troubles. For example, because certain tools offer more than 50 project metrics it can become rather difficult to select the right metrics for status reporting, or to ensure that related metrics are not selectively reported on, which can lead to overlooking side effects of improving on only a subset of related metrics. Aside from that, the scientific background, rationale, and business value of such metrics are not made explicit.

With the vision of our model in mind, as it was described in section III (i.e. in terms of applicability, usefulness, and perspective), we set out to select an approach to guide the process of describing the model. Such a method was required to support us in finding a well-ordered set of metrics that are relevant in the context of our research.

Therefore we adopted the Goal-Question-Metric (GQM) approach [8]. This top-down approach consists of four phases, starting with 'planning' and 'definition', during which metrics are defined based on relevant goals and questions from a certain perspective. The final two phases - 'data collection' and 'interpretation' - entail applying the

metrics and feeding the results back to the GQM model in a bottom-up manner. This would be done to answer the questions defined in the planning phase, and thus serve their corresponding goals.

A. *The GQM method planning phase*

An important step of the GQM planning phase is the selection of an *object of measurement*. We chose to use the product backlog as the main source of input. Naturally, there are many things that can be measured in agile projects, but we wanted to delineate our focus and keep our method concise, feasible and generally applicable. In agile SD, product backlogs are often in place by design, meaning that the data we need is already there, embedded in a certain backlog implementation.

The constraint of focusing only on the product backlog also helps the method in being valuable for project owners, since it should result in a graspable and clear picture of the status of an agile project, stemming from a platform that is actively used by the developers. It thus creates a shared platform for communication.

Product backlog properties varying across organizations, forced us to create a product backlog model. This was also required before defining a set of metrics later on. We consider a *Product Backlog* (PB) with a number of items on it, called *Product Backlog Items* (PBIs). A PBI can have a variety of properties, but we selected the properties that are fundamental to agile development, namely 1) a unique ID, used to trace the PBI, 2) a name, 3) a description, that describes the requirement 4) an effort-estimation, indicating the amount of effort needed to realize the requirement 5) a priority, used to favor certain requirements over others during iteration planning 6) a status, and 7) a parent PBI, which is optional.

These properties are supporting the pillars for agile projects, i.e. that developers should reflect - at regular intervals - on how to become more effective, and how to adjust accordingly [1]. For example, the execution of a PBI is supposed to fit in one sprint. So if the granularity of PBIs is too low, making them too complex for one sprint, the developers should adapt and split the remaining work into smaller units of work (i.e. smaller PBIs).

As shown in Figure 1, a PBI in our model can have one of five statuses. A PBI always starts out as 'designed' (D) at the time it gets added to the backlog. In this phase, minor effort is put into determining properties such as a description a first effort estimation. After a certain amount of time, a PBI is to be picked up by a development team (T) that starts implementing it. Once the item has been implemented it goes through an acceptance stage (A), which usually involves testing, before it goes into production (P). At any time before the 'produced' stage, a PBI could be rejected (R) if it is decided that the item is no longer needed. A more detailed description of these status distinctions is shown in TABLE I.

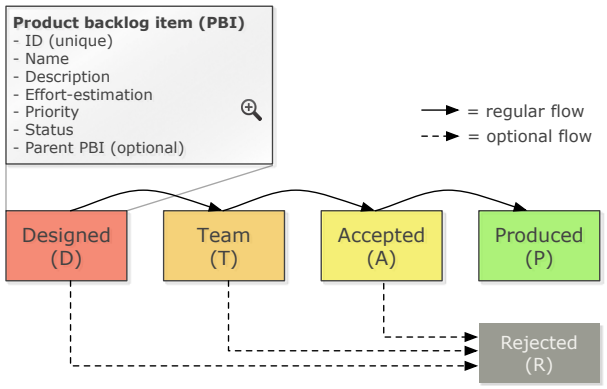

Figure 1: Product backlog item properties and statuses

TABLE I. PBI STATUS DESCRIPTIONS

Status	Description
Designed	Can be a vague idea of a (non) functional requirement. Preferably has initial effort estimation.
Team	Whenever the PBI is in a sprint, and thus being worked on by one of the teams in the project.
Accepted	Includes the testing phases used in the project. The item can be in a test, or have been successfully tested and waiting for a release in the production environment.
Produced	In production, meaning that the requirement is part of the progressive product and used by the end users.
Rejected	Any PBI that has been removed from the backlog should be flagged as such. Actually removing the entry would mean the loss of information.

B. *The GQM method definition phase*

To derive a number of relevant SD project owner goals, we used literature about SD projects success and failure [9]. We also performed unstructured interviews with IT practitioners at the Software Improvement Group (SIG)[1], who are experienced in both IT consultancy and low-level technical analysis of software. Using the goal-template provided by [8], we described our model's goals. This goal description template is structured as follows: *"Improve (purpose) the reliability (issue) of product X (object) from the viewpoint of the (perspective) within organization Y (context)"*. In our case, the perspective and context are equal for each of the goals; respectively the project owner and any organization performing an agile SD activity.

We concluded this phase by defining questions and metrics corresponding to the goals. The mapping of goals to questions and metrics can be seen in TABLE II. Certain metrics correspond to multiple questions.

Note that we have defined two questions that are not explicitly covered by a common product backlog. During GQM definition, we decided to include these metrics nonetheless, because they are needed to answer questions related to cost and quality.

[1] The Software Improvement Group (www.sig.eu/en/)

TABLE II. GOAL, QUESTION, METRIC MAPPING

Goal	Question	Metric
1. Achieve the functional compliance of the software system from the viewpoint of the project owner/sponsor[a]	1.a. To what extend are the functional requirements being implemented?	Enhancement Rate
		Scope Prognosis
		Project Size Remaining
	1.b. How much scope churn is there?	Changed PBIs
		Added PBIs
		Rejected PBIs
		Project Size
2. Fulfill the expected schedule of the SD activity from the viewpoint of the project owner/sponsor[a]	2.a. When is the delivery of the software system expected?	Enhancement Rate
		Time Prognosis
		Project Size Remaining
3. Optimize value for money of the SD activity from the viewpoint of the project owner/sponsor[a]	3.a. What is the quality of the development process?	Enhancement Rate
		Estimation Shift
		Priority Shift
		PBIs At Risk
	3.b. What is the quality of the product?	Software Quality[b]
	3.c What is the financial status of the project?	Expenses Prognosis[b]
4. Minimize the risk of wasting SD effort from the viewpoint of the project owner/sponsor[a]	4.a. What is the amount of effort at risk?	Effort At Risk

a. For each goal, the context is: any organization performing an agile SD activity

b. These metrics cannot be measured using the backlog only

These two aspects – cost and quality – are part of the so-called *iron triangle* [10] and extensions to it [11][12], that describe project constraints which are important to keep an eye on when trying to achieve project success. Therefore, we believe that they cannot be neglected in our model.

C. Descriptions of the metrics

In this section we describe the metrics that we have defined during the GQM definition phase. Note that we do not describe each metric, for the following reasons.

Firstly, in TABLE II we noted that two of the metrics were added even though they require input other than the backlog. The *Expenses Prognosis* metric has to do with analyzing financial expenses and extrapolating the expected project costs by combining the foreseen expenses with the either of the two other prognosis metrics (i.e. *Time Prognosis* and *Scope Prognosis*). The *Software Quality* metric refers to measuring software quality following quality properties described in ISO/IEC 25010. At SIG, analysis tools and methods have been developed to measure several of the sub-characteristics in ISO/IEC 25010.

Secondly, we do not describe the *Enhancement Rate*, *Scope Prognosis*, and *Effort At Risk* here, since they are described in detail in section V, where they are discussed in light of our case study.

We use a *completion ratio* to quantify the completeness per status and give more meaning to some of the metric's results. This ratio is used for example in the *Enhancement Rate* metric. What it means is that PBIs with the status 'accepted' are considered to be 95% complete, and items with the status 'produced' are complete. TABLE III shows the *completion ratios*, as well as the *rejection impact ratios*.

These ratios are the result of non-structured interviews with IT experts at SIG. They serve as an extra layer of detail, and improve the sensibility of certain metrics. The *rejection impact ratio* is discussed in V.C. There it is used to describe the *Effort At Risk* metric, the only metric that makes use of this ratio.

TABLE III. COMPLETION RATIO AND REJECTION IMPACT RATIO PER PBI STATUS

Status	Completion ratio	Rejection impact ratio
Designed	-	-
Team	-	0.70
Accepted	0.95	0.95
Produced	1.00	1.00
Rejected	-	-

1) Added PBIs

"Added PBIs" is the percentage of PBIs that are new on the backlog of a given development iteration, which were thus non-existent on the backlog of the iteration before. The metric compares PBIs by means of their unique ID, which means it does not count PBIs of which the description has been changed.

PBIs with descriptive changes are handled by the *Changed PBIs* metric, and are thus not counted as newly added items.

2) Changed PBIs

"Changed PBIs" measures changing descriptions of existing PBIs. If a PBIs description differs from the description that the same PBI had in the iteration before, the item is added to the number of changed items. Ultimately, the result is given as a percentage of the total PBI count.

Changes in existing PBIs are can be overlooked, because the total number of PBIs and their effort estimations may not change accordingly. However, changing descriptions can have an impact on the amount of effort that is needed to implement the corresponding PBIs. In such a case, a project would be growing in terms of needed effort, even though this is not directly visible if not measuring description changes.

3) Estimation Shift

"Estimation Shift" measures changes in estimations. One way of quantifying effort estimations is by using *story points*. These translate to a certain amount of man-hours, depending on the development team that assigned the *story points*. In other words, one *story point* might translate to 1 hour for team A, but to 10 hours for team B.

This metric takes any negative or positive differences in effort estimations of an iteration compared to the iteration before, and gives the sum of these as a result. For example, if there would be a total decrease of 100 *story points* in one

iteration, and a total increase of 150 *story points* in the same iteration, the resulting value for this metric would be 50.

If the amount of positive shift were equal to the amount of negative shift, the combined result of the *Estimation Shift* would be zero. To make sure that the metric remains insightful in such cases, the visualization (i.e. graph) of this metric's result is split into positive and negative shift. The results are given as a percentage of the total *Project Size*. The reason for combining the final result is the fact that a leveled out shift in effort estimations would mean that the sum of the effort estimations was not altered due to estimation shifts. Note however, that the total estimated effort could then still have changed due to the addition or rejection of PBIs.

4) Priority Shift

As with the estimations, we would like to gain insight into the shift in priorities of each of the PBIs on the backlog. An example of backlog prioritization is the MoSCoW principle, which breaks down to a categorization into four types of PBIs, called 1) must haves, 2) should haves, 3) could haves, and 4) would/won't haves. The "Priority Shift" metric takes the relative change in priority and adds this to the overall result. A change from priority 1 ('must have') to 4 ('would have') would thus numerically mean a shift of 3.

5) Project Size

The size of the backlog can be measured in two ways, and the applied method is reflected in the implementation of most of the other metrics as well.

Generally, we expect that effort estimations are available, meaning that we weigh each PBI accordingly. In that case, the "Project Size" is given by the total effort estimation (e.g. the total number of *story point*). If there are no effort estimations given, this metric counts the number of PBIs.

6) Project Size Remaining

The "Project Size Remaining" metric closely related to the *Project Size*, as it takes the result of that metric and subtracts from it the amount of work that has been completed. Here, the *completion ratio* for accepted items (TABLE III) is used to reflect the effort that is invested into getting PBIs into acceptance. PBIs in with the status 'produced' are subtracted.

Project Size Remaining bears a resemblance to the commonly used *burn-down* chart. A burn-down is a visualization of work left to do compared to time. Varying implementations of the burn-down chart are used in practice, some of which use the number of PBIs left, while others consider the effort estimations to quantify the work left. With *completion ratio* we try to give another layer of depth to the view on remaining work. Part of our future work would be to further refine such details.

7) Rejected PBIs

In agile, a certain level of re-scoping is expected. This does not only mean adding or changing PBIs, but also removing them from the backlog. A low priority can be used to indicate a low importance of a PBI, but if it is clear that the implementation of a certain PBI is not required anymore, it is usually removed from the backlog. Unfortunately – due to varying handling of rejections by different tools, the removal of PBIs is often not visible in the project – which can result in an incorrect picture of the development process.

We argue that it is important to keep track of rejected PBIs. That means that PBIs should never be physically deleted, but rather flagged as 'rejected' or 'deleted'. If this is done, our "Rejected PBIs" metric can measure the trend of PBI removal.

Again, this metric can either be weighted or it can use an absolute count of the number of PBIs, depending on whether effort estimations are available. The *Rejected PBIs* metric shows, for a certain iteration, the percentage of the *Project Size* that has been rejected since the preceding iteration.

8) Time Prognosis

The "Time Prognosis" metric is very closely related to *Scope Prognosis*. Depending on environmental constraints, the end date of an agile project might be fixed, which requires adjusting the scope during the project so that it fits the desired schedule. This means that by the end of the project, the product might not be completed, but following the agile principle of iteratively releasing a working product, it can be made operational albeit with limited functionality.

For each sprint, *Time Prognosis* shows the expected end-iteration (i.e. the iteration at which the product is likely finished), by combining the average *Enhancement Rate* with the *Project Size Remaining*. By changing the input value of *Project Size Remaining*, thus simulating a scope cut, one could visualize the effect of cutting the scope on the expected end-iteration. This can help in deciding if a planned scope cut is big enough to get the project back on schedule.

V. APPLYING OUR METHOD IN A CASE STUDY

Our goal with this case study was to evaluate the backlog metrics in practice by assessing their practical feasibility, reliability and usefulness. We assessed these aspects for each backlog metric by performing semi-structured interviews with the client's IT program manager, as explained earlier. He was fulfilling a sponsoring position outside the development teams, but still had enough internal knowledge of the project and the teams to be able to link our observations to actual events in the development timeline.

We have not evaluated 3 of the 13 metrics, since two of these metrics involve more than just the product backlog (i.e. *Expenses Prognosis* and *Software Quality*), and the third metric required data that the client did not have available (i.e. *Priority Shift*).

In collaboration with SIG we applied the aforementioned subset of 10 metrics of our method at a client, using an automated implementation of our model in spreadsheet software. We chose to do so, at least for the length of this case study, in order to be able to swiftly produce visualizations, to maintain a certain level of flexibility, and because of time constraints.

The client in question was in the process of developing a renewed version of a legacy system. This solution was needed to cope with law changes. The project consisted of sprints with durations of two weeks each. This SD project was estimated to take 13 sprints. At the moment we got the backlog data, the client was active in sprint 10. The data we received existed of 7 separate files, each being a historical

backup of the product backlog as it was at one of sprint numbers 3 through 9. Unfortunately, the backlogs of sprints 1 and 2 were not available.

Together with the client, we transformed the received data so that it fitted our product backlog model (e.g. by mapping PBI statuses onto our representation of PBI statuses). When the data transformation was complete we imported the data into our model, after which the metrics' results were calculated for each sprint. We included empty data sets for the two unavailable sprints 1 and 2 so that the sprint numbers correspond with the actual sprint numbers of the project, to prevent unclarities regarding the prognosis metrics (e.g. *Scope Prognosis*).

In the remainder of this section we will show the results of 3 of the metrics that we find to be the most representative in this case. We discuss our observations on the results of these metrics. At the end of the section we present the client's specific feedback on the three examples, as well as the overall client feedback on our model's metrics in terms of practical feasibility, reliability and usefulness.

A. Enhancement Rate example

This metric shows the throughput of PBIs in the project, based on their status and effort estimation. For every sprint, the *Enhancement Rate* shows what percentage of the *Project Size* of the preceding sprint – thus including items with the status 'produced' – has gone into acceptance or production at the current sprint. For this metric, and many of the other metrics, we favor PBI weight over the more naïve PBI count. This means that, if the data is available, we use effort estimations (e.g. *story points*) to weigh each PBI. This gives a more representational and detailed indication of the amount of work that is associated with a PBI.

The result of this metric is the trend line shown in Figure 2, which starts at ~9%, moving up to ~12% after a dip to 6%. The main observations here are the dip around sprint 6 and the steep increase in the sprints thereafter. The *Enhancement Rate* can change due to a number of different factors.

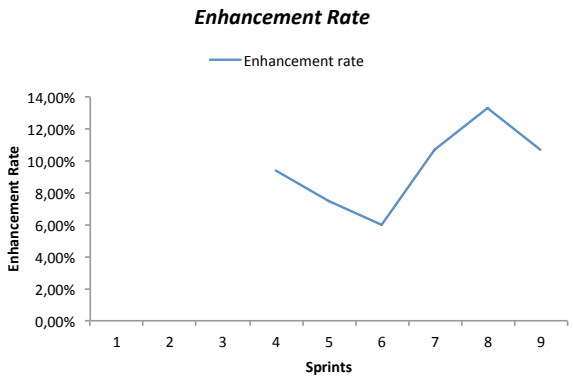

Figure 2: Enhancement Rate metric results

For example, the rate can increase when the development teams become more effective.

The reason for the trend line starting at sprint 4 instead of sprint 3 is that the metric's result is acquired by comparing the throughput with the *Project Size* of the iteration before. That is because it is not likely that PBIs are designed and developed in the same iteration. We thus look at the throughput as a percentage of the backlog size as it was at the start of the concerned sprint, which is why there is no result for sprint 3.

B. Scope Prognosis example

To address our observation of organizations not always sticking to a fixed end date – either deliberately or as a result of continuous development – the *Scope Prognosis* turns the principle of the *Time Prognosis* around by taking the scope of a certain sprint and fixing that scope instead of a desired end-iteration. We then determine the expected level of completeness for an arbitrarily given end-iteration.

The *Scope Prognosis* takes into account the average of the *Enhancement Rate*, and combines this with the *Project Size Remaining* and the given end-iteration. With this metric we want to give the project owner an indication of the expected level of functional completeness. In practice, a desired end date for the project would most likely be used as input, to see if it is realistic.

As with the *Enhancement Rate*, if effort estimations are available, then these are used to weigh the PBIs accordingly. This gives a more fine-grained and more reliable view on the amount of remaining effort needed to complete the project.

The graph in Figure 3 shows the results from our case study. For each sprint that we have analyzed, estimations for the level of completeness at the given end-iteration (in this case sprint 13) are shown. Although the trend-line seems to starts out a bit off at 30%, it starts following a more reliable path soon after. This is most likely caused by the rough result of the average *Enhancement Rate* for the first data point.

The trend is in line with the increase in the *Enhancement Rate*, causing the *Scope Prognosis* to improve from ~60% to ~80%. This would mean that, according to the metric, 81% of the *Project Size* would be completed at sprint 13.

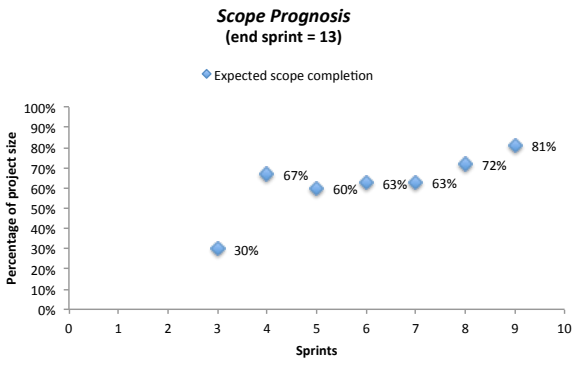

Figure 3: Scope Prognosis metric results

Since the average *Enhancement Rate* is used to assess the speed with which PBIs are being realized, the effect of an increasing *Enhancement Rate* is not immediately visible. The first notable increase in *Scope Prognosis* in this case, is

therefore in sprint 8, one sprint after the first increase of the *Enhancement Rate*.

C. *Effort At Risk example*

Whilst getting PBIs from the 'designed'-phase to the 'produced'-phase, the amount of spent effort increases. It requires effort to describe a PBI, to implement it, to test it, and to integrate it into a working software product. In our model, this effort is turned into product value gradually per PBI, but only definitely when a PBI actually goes into production. By that time, it has become part of a working product that holds value in terms of functionality, non-functional requirements, and technical quality. Before production, any effort made is at risk of being lost, for example due to rejection of the concerned PBI(s) or if a PBI gets stuck in the acceptance process.

To provide an overview of the amount of effort that has not yet been turned into product value, as well as of the relative risk of losing this effort, we defined the *Effort At Risk* metric. The metric divides effort into three risk categories, namely 1) low risk, 2) medium risk, and 3) high risk. The categories map to the three statuses 'accepted, 'team', and 'designed respectively. This means that PBIs that are in the 'designed'-phase have the highest chance of being rejected, and the items in the 'accepted'-phase have the lowest chance, due to them being more mature.

To put the actual risk into perspective, the *rejection impact ratio* (TABLE III) is used when summing up the PBIs per category. Again, if available, the effort estimation is used for weighing the PBI. For example, a PBI with an effort estimation of 20 *story points* with status 'designed', has a relative high chance of becoming rejected. However, since it is in the design phase, the *rejection impact ratio* is low. Since we consider designed items to have consumed about 10% of their total needed effort, we would add 2 *story points* (0.10 times 20) to the high risk category.

The graph in Figure 4 shows the results from the case study.

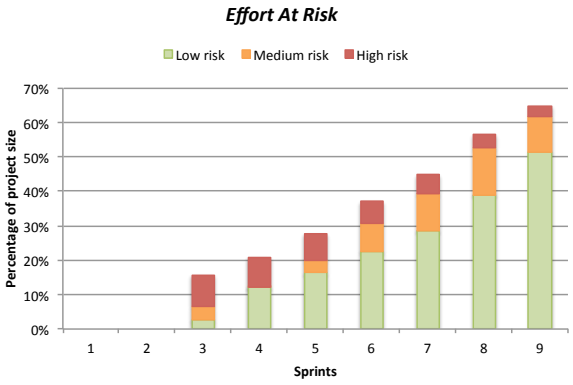

Figure 4: Effort At Risk metric results

The most important observation here is the continuous increase in total *Effort At Risk*. Aside from that, the relative amount of high risk decreases, which is compensated by an increasing amount of medium and high risk. Apparently, PBIs are getting stuck in the 'accepted' status where they form a large portion of the total risk, albeit in the low risk category (i.e. having a low probability of being rejected).

D. *Case study results*

After our analysis of the product backlog, we validated each backlog metric in a semi-structured interview with the client, focusing on practical feasibility, reliability, and usefulness. Assessing the reliability of the metrics was important, because it indicates the extent to which the metrics are capable of reflecting actual project events. This is required, when the metrics are being used to report on active projects. The semi-structured interview entailed going through each of the measured metrics and having the client link project events to our observations of the metric trends. After that, each metric was discussed again to get feedback on each metric's practical feasibility, and an indication of its usefulness had it been used in an active project.

By letting the client link project events to the observations that we got from the metrics' results, we assessed the reliability of the model. We also asked the client about the feasibility and usefulness of each metric. Note that we were not able to measure *Priority Shift* and *Expenses Prognosis* in this case, because the client was not able to deliver the data needed for the calculation of these metrics.

From the results of the *Enhancement rate*, the client recognized the dip at sprint 6. During the sprints leading up to that moment there had been a deliberate focus in the project on designing new PBIs. At that time, it was already decided that a new team would be added to the project around sprint 6. Therefore, it was expected that the new team would compensate for an increased workload caused by the addition of PBIs.

The decrease could thus be explained by the addition of PBIs. The steep increase of the *Enhancement Rate* after sprint 6 could be explained by the addition of the extra team. It seems to be the case that the extra team was more than productive enough to cope with the added work, as the *Enhancement Rate* moved to and over its original level.

The *Scope Prognosis* was found to be particularly useful by the client. It is closely related to the other two prognosis metrics (i.e. *Time Prognosis* and *Expenses Prognosis*), which is something the client noted as well, but it was found to be the most insightful and useful of the three prognosis metrics. The client stated that this metric clearly showed the effect of the *Enhancement Rate* on the remaining work, and that it could be very valuable when deciding whether to cut (or maybe even increase) the project's scope. The results were apprehended as reliable, and in retrospect deemed to be reflective project progression.

The *Effort At Risk* metric was also perceived as highly useful. The categorization into low, medium, and high risk was said to be very insightful and valuable for monitoring an agile SD process. The client noted that there had been a deliberate decision to postpone putting PBIs into production to the end of the project. This was in line with our observation of the *Effort At Risk* results. The client stated that, since the *rejection impact ratios* were based on expert opinions, it might be fruitful to conduct further research

effort into determining the actual loss off effort per status when rejecting PBIs.

TABLE IV shows the feedback that we got from the client in terms of practical feasibility and usefulness of the metrics. Some metrics were not measured in this case study, which is why certain entries are missing. Still, for some of these metrics we did get partial feedback, either because we explained the metric (in the case of *Expenses Prognosis*), or because the client had experience with the metric through other engagements with SIG (in the case of *Software Quality*).

Overall, the model was found to be feasible and useful. The client stated that in an active project, our method could help initiate changes, that it could support decision-making, and that it could serve as an important communicative tool when reporting to higher management and external stakeholders. The client was able – in retrospect – to link our results and observations to project events (e.g. the addition of a development team).

TABLE IV. METRICS' FEASIBLITY AND USEFULNESS ACCORDING TO THE CLIENT

Metric	Feasibility	Usefulness
Added PBIs	High	Medium
Changed PBIs	Low	Low
Enhancement Rate	Medium	High
Estimation Shift	High	High
Expenses Prognosis	-	-
Effort At Risk	Medium	High
Priority Shift	-	-
Project Size	High	Medium
Project Size Remaining	High	Medium
Rejected PBIs	High	Medium
Scope Prognosis	Medium	High
Software Quality	High	High
Time Prognosis	Medium	Medium

The client least valued *Changed PBIs*. The most important reason for this was that the client found this metric to be too 'naïve' in the sense that minor changes to PBI descriptions were counted as heavily as major changes.

VI. DISCUSSION

The presented model provides a non-intrusive and lightweight method to bridge the gap that typically exists between agile development teams and external stakeholders. By observing the state of the backlog, we have defined useful indicators that provide more control and confidence around agile teams for external stakeholders. Teams themselves can maintain their autonomy, while the backlog administration gives them insights internally. By means of our method, the same administration can now also serve as a communicative tool that provides stakeholder with understanding, and can be used for discussion with the project sponsor(s).

Although we are positive about the initial results of our approach, the metrics and their relations are still somewhat basic, in the sense that a high level of expert understanding is needed to interpret the measurements in a balanced way. For example: we have no empirical data on how much scope churn, or growth and dynamic in project size is typical for agile projects, or beyond what threshold it becomes a reason for concern.

We need additional data points to empirically derive thresholds that enable us to better interpret the metric results. We might come to the conclusion that such thresholds are project specific. Further application of the model should also give us more data on the robustness of the individual metrics. For example, the average of the *Enhancement Rate* takes the project's overall average, though a sliding window approach might give a more realistic view when monitoring during long periods. We want to improve our metrics without increasing the amount of effort needed to supply data to our model, thus maintaining its non-intrusiveness.

Aside from that, more case studies are required to remove the subjective bias that might be the result of the one case study that we performed. In that sense, other case studies could yield different results or problems that were not found in the case study we performed.

Additionally, an explicit baseline for agile development project could help as well to give more direct meaning to the metrics. We feel that project stakeholders at the least have a high-level idea on what they expect to be delivered, in what timeframe and against what effort. It is then up to the agile team to deliver within those boundaries, or discuss the expectations.

Many of our metrics are deemed to be more meaningful if they consider PBIs in a weighted manner (i.e. weighing them using effort estimations). We learned that many organizations use *story points* for effort estimations. A risk of incorporating *story points* in our metric calculations is the volatility of this unit due to its subjectivity, and its therefore varying relation to objective units such as time. For example *story point* inflation might occur, which could cause unreliable results for *Scope Prognosis*.

Approaches to address problems that can arise when using *story points* might be 1) tracking the age of *story point* estimations, and requiring teams to reconsider older estimations to improve overall consistency, or 2) linking initial effort estimations of each PBI to actual time spent on that PBI, or 3) using other units for weighting PBIs. These approaches might require extra effort from the average development team, which is something we want to minimize in the end.

An important factor in applying our method is the availability of data and its quality. In our case study we already found that extensive manual effort was needed to clean the project data and map it onto our model. On the other hand, we feel that the availability of the model supports better data quality in project administration as it explains what needs to be administered in what way, and why. We experience that many projects keep insufficient administration for facilitating discussion and interpretation. We already see in ongoing cases that the data is more structured when using dedicated tools (e.g., JIRA), and that gathering snapshots for each sprint is much easier and robust when this can be automated.

VII. CONCLUSIONS

We presented a reporting method for agile projects that can improve agile project stakeholders' communication, reporting processes, and stakeholders' sense of control. Our method consists of 11 metrics based on the product backlog, and 2 additional metrics.

By using the Goal-Question-Metric approach to define metrics, we were able to determine a measurement model that is linked to questions and goals that were formulated from the perspective of project owners/sponsors. This perspective was chosen because existing literature often mentions the sense of losing control and miscommunication in the same sentence as the need for increased project owner involvement.

We based most of our metrics on the product backlog, and we added two extra metrics that require extra input aside from the product backlog. As we explained, these metrics are not required for the model to work, but they provide valuable additional insight into software development (i.e. financially, and qualitatively). Concerning the metrics that purely rely on the product backlog, very little effort from the side of the client was required to perform our method, which was one of our goals.

During our case study, we learned that the method is practically feasible, useful, and reliable overall. An important finding was that the client was able to link actual project events of the concerned project, to notable metric results. For the metrics that this applied to, the client noted that they could have been used to actively steer the project, had the method been used when the even occurred during the project. This supported the reliability of these metrics.

Some of the metrics were deemed more valuable than others, and the client of the case study made some remarks regarding the implementation of certain metrics. These remarks helped us in improving the model before we used it in other cases. An example of a metric in need of more refinement is *Changed PBIs*, which was considered to be the least feasible and useful due to the way description changes are measured (i.e. with no consideration to the amount of change that occurs).

VIII. FUTURE WORK

The case study presented here, as well as further case studies that are still active, are helping us to further evaluate and improve the individual metrics and the method as a whole. We are working with new clients to continue putting our model into practice, and improve our metrics along the way. Another important focus area of our current and future case studies is – and will be – determining risk thresholds for each metric, so that the metrics' outcomes can be more easily linked to strategic decisions.

Similarly, we are aiming to implement our method in an automated tool, that will be able to extract and transform backlog data from commonly used backlog tools and automatically calculate the metric results from this. This is being done to limit the time required to analyze large, evolving data sets, and to minimize the chances of errors occurring during the analysis. In light of this automated tool, we are focusing on JIRA for now, as this is one of the most used tools in agile development environments.

During our research we encountered several software development projects in which the implementation of the product backlog was not done properly. Often, required PBI properties were incomplete – or even non-existent – even though they are fundamental to the agile principles. Aside from that, data was unreliable or inconsistent at times. In most cases, the low data quality was found in backlogs that were maintained in undedicated tools, but low data quality was also apparent in dedicated tooling.

This finding suggests that checking data quality during a project's start-up – but at least before applying our method – is important to ensure the reliability of the eventual outcome of our metrics framework. We are performing additional studies with regard to a maturity model for backlog data quality. These studies might suggest a minimum quality threshold required to apply our framework in practice, and indicative of the reliability of our backlog analysis.

REFERENCES

[1] M. Fowler and J. Highsmith, "The Agile Manifesto," *Softw. Dev.*, vol. 9, no. 8, pp. 28–35, 2001.

[2] VersionOne inc., "8th Annual State Of Agile Survey." VersionOne Inc., 2014.

[3] R. N. Charette, "Why software fails," *IEEE Spectr.*, vol. 42, no. 9, pp. 42–49, 2005.

[4] R. Turner and R. Müller, "On the nature of the project as a temporary organization," *Int. J. Proj. Manag.*, vol. 21, pp. 1–8, 2003.

[5] R. Turner and R. Müller, "Communication and Co-operation on Projects Between the Project Owner As Principal and the Project Manager as Agent," *Eur. Manag. J.*, vol. 22, no. 3, pp. 327–336, Jun. 2004.

[6] M. Ceschi, A. Sillitti, G. Succi, and S. De Panfilis, "Project management in plan-based and agile companies," *IEEE Softw.*, vol. 22, no. 3, pp. 21–27, 2005.

[7] Y. Leau, W. K. Loo, W. Y. Tham, and S. F. Tan, "Software Development Life Cycle AGILE vs Traditional Approaches," in *International Conference on Information and Network Technology*, 2012, vol. 37, no. Icint.

[8] H. Koziolek, "Goal, question, metric," in *Dependability Metrics*, I. Eusgeld, F. C. Freiling, and R. Reussner, Eds. Berlin: Springer Verlag, 2008, pp. 39–42.

[9] F. Castri, D. A. Tamburri, and P. Lago, "Success and Failure in Software Engineering," 2012.

[10] P. W. G. Morris and G. H. Hough, *The anatomy of major projects: A study of the reality of project management*. Chicester: John Wiley & Son, Ltd., 1987.

[11] J. Wateridge, "How can IS / IT projects be measured for success ?," *Int. J. Proj. Manag.*, vol. 16, no. 1, pp. 59–63, 1998.

[12] K. Jugdev and R. Müller, "A retrospective look at our evolving understanding of project success," *Proj. Manag. J.*, vol. 36, no. 4, pp. 19–32, 2005.