Governance of an Agile Software Project

David Talby
Hebrew University of Jerusalem
davidt@cs.huji.ac.il

Yael Dubinsky
IBM Haifa Research Lab
dubinsky@il.ibm.com

Abstract

Effective governance of agile software teams is challenging but required to enable wide adoption of agile methodologies, in particular for large-scale projects. In this paper we apply a full lifecycle governance model to agile projects, focused on the iteration level. The concept is demonstrated via a case study of a large-scale, enterprise-critical software project that implemented agile practices. We analyze three governance events, including the metrics that triggered the event, the decisions taken and the follow-up to ensure resolution. We conclude that governance iterations can be naturally unified with agile development iterations, resulting in a lean governance mechanism that identifies and resolves issues in an effective and timely manner.

1. Introduction

One of the prevailing points of criticism against agile software development is that it leaves little place for external oversight of projects. The *Whole Team* practice [1] short-circuits common communication barriers by putting the customer, developers, business analysts and testers all in the same room and team. The team is then empowered to decide what to deliver and even how to adjust its development process over time using a retrospective practice [11]. This changes the traditional roles of project and program managers, by making some of the communication and coordination aspects of these roles redundant.

Senior managers and program managers usually supervise multiple projects simultaneously, cannot dive into the details of each project, but are still expected to keep all projects on track by identifying and resolving alarming issues as they occur. Providing effective governance mechanisms that enable them to monitor projects effectively is thus a key success factor for the adoption of agile methodologies and the realization of their business value. This is particularly critical in large-scale software projects that are not released to end users in short time intervals, regardless of being developed in short iterations.

Successful large-scale agile software development must therefore reconcile agile principles – including whole, self-organizing teams – with enabling proper governance, which in the scope of this paper involves answering the following questions:

- How do we know on an on-going basis that an agile project is on track, and effectively identify and resolve alarms?
- How do we know if and to what extent the agile practices improve the software development productivity and quality of a given team?

This work presents an application of a recently proposed full-lifecycle model of software project governance [7] to agile projects. The subject of research is a large-scale software project developed by the Israeli Air Force [12]. Our data comprises of quantitative and qualitative data, from interviews and questionnaires assessing current state and goals at the outset, through to metrics collected by the team. Focusing on the level of the single iteration, we show that agile practices can be implemented in a way that naturally incorporates the full governance lifecycle.

This paper is organized as follows. Section 2 provides background of software project governance, and describes its key stages in an agile context. Section 3 presents a case study for which we describe the governance of one agile team using metrics that were designed for that project. The data was collected in its first three releases (12 iterations). Moreover, data from the retrospective process was used in order to add the team's perspective. In Section 4 we analyze the case study through the roles and responsibilities that were involved. In Section 5 we conclude.

2. Governance of Software Development in an Agile Environment

Governance of software development is a relatively new and evolving field [3,5,8,9] that in a sense has emerged from the IT governance arena [2,4,13,15].

Governance in general is defined by the Webster dictionary as exercise of authority, control, and arrangement [14]. This definition relates well to top-down 'command and control' situations that are common in traditional organizations. In the last decade, the agile approach to software development [1,10,11] promotes the use of self-managed teams, thus enabling the exploration of software development governance and its effectiveness in bottom-up 'empower and reflect' situations [8].

Given a specific software project, governance of software development is an iterative process concerned with the project goals and how they are expressed in the decisions made by the different roles involved with the project. Such decisions are based on policies that are defined and metrics that are collected accordingly. Using the metrics data gathered, role holders can participate in decision making as per their decision rights. Adjusting the governance iterations to the short development iterations enables on-going reflection on the governance events – events that highlight key issues in development, and how governance mechanisms such as policies and metrics assist relevant roles in making decisions to solve and follow up on these issues.

An agile iteration, in all major agile methodologies consists of these main parts [1,10]:

- **Planning session** – the project team listens to the customer's requirements that are prioritized for this iteration and distribute the development tasks that are derived from a high level design among teammates. The development tasks are estimated to assess the iteration scope.
- **Development days** – the project team works to complete the committed tasks. During this time a re-planning session may occur in case there is a need to change the iteration scope – for example defects that should be fixed during the current iteration, or wrong estimates that lead to reprioritization of customer requirements.
- **Presentation and feedback** – the project team presents the customer with the developed features of the iteration. Metrics are also presented – the most common are ones that relate to team velocity and progress.
- **Reflection session** – the project team reflects on the ending iteration and decides on action items to improve accordingly. Reflection can relate to technical, social or organizational issues.

Each iteration is part of a complete release, so each reflection session is typically immediately followed by the next iteration's planning session.

It is a natural "lean" idea to implement governance of an agile software project by unifying the agile iterations and the project's governance iterations. The *definition* stage of the governance lifecycle happens first before development begins, when the team first decides on its work procedures and metrics. The *activation* stage happens during the development days, in which work is done and metrics are captured. The *assessment* stage happens in the presentation and reflection sessions, with the reflection having the formal role of being the *definition* stage for the following iteration.

In the scope of this paper we do not address governance across releases. Governance at the iteration level has two goals:

- **Ensure alignment with the iteration goals**: The level of governance abstraction is low and is focused on the current development iteration that is 2-3 weeks long.
- **Effectiveness for a specific team and problem:** Effective governance must deal with current, relevant issues in order to show improvement in the short term – the very next iteration. Key traits are flexibility and appropriateness.

3. Case Study: Governance of an Agile Project

The software project under research in this paper is a large-scale, enterprise-critical system, intended to be used by a large and varied user population. The project was developed within the Israeli Air Force for internal use. In the first release the team had a coach and 15 members, out of whom 7 were full-time developers and 8 were business analysts, architects, developers and testers who devoted between 30-60% of their time to the project. By the third release the project team grew to 15 full time members.

In what follows based on a brief description of the definition stage of the governance of this project, we present three specific governance events. They show how data that is gathered by governance mechanisms can be regularly assessed after each iteration, affecting the team's behavior right afterwards.

3.1. The Governance Definition Stage

The decision to use Extreme Programming (XP) as this project's development methodology was the result of a one-year effort – meant to change the established waterfall development process in order to enable more rapid response to customers' requests, required changes and new requirements. Following training, a workshop and a pilot, this was the first large-scale agile project to launch, and while upper management fully supported the change, it was considered risky and thus closely supervised.

XP practices contradicted many of the existing work procedures and templates used in the organization, and so the team was given a waiver on most of the existing regulations. However, this required the team to establish a strong governance framework: it was allowed to design its own work processes, as long as it could show management that the project *is* managed – effectively run and under control. The team's success to do so not only enabled better external oversight, but also addressed the challenge of internal governance – helping the team members make the switch to the agile practices and way of thinking.

In addition to establishing a planning, development, presentation, feedback and reflection cycle as described in the previous section, the project's leadership defined a set of metrics [6,12]. The formal role of tracker was established in the team, as responsible for the quality and continuity of measurement. Metrics were presented to the customer as an integral part of each iteration's summary meeting.

Since the metrics – whose details are elaborated in the next section – were designed for the iteration level and for the needs of this specific team, they achieved multiple governance goals. First, they ensured external observers that the project was making good progress and that key risks were under control. Second, it educated team members on what aspects of their day-to-day behavior were deemed most critical to focus and improve on. And third, it enabled data-driven decision making, making that stage of the governance lifecycle easier for a new, diverse team.

3.2. Governance Events

We present three events in which an alarm was triggered and resolved by the project's governance mechanisms defined above. Each event demonstrates one of the core metrics that were used to track healthy progress. In each event:

- The alarm was raised during the iteration summary meeting when metrics are reviewed.
- Action items to resolve the issue were discussed and decided at that iteration's reflection meeting.
- The next iteration's summary meeting two weeks later is used to verify that the issue is resolved, and find out what needs to be addressed next.

Event I: Re-plan the release scope

Burn down is a well-known metric used to measure a team's progress towards commitments made to the customer during the planning stage. In the project under research the risk that this metric addressed was that the team would get bogged down in perfecting minor features – adding more complexity and increasing the product size, at the expense of other major requirements.

To address this, the project was divided into two-month-long releases – a variant on the XP quarterly planning practice. Each release began by allocating resources to it, and then planning the deliverables for that release based on a high-level estimate of how much effort each one would take. The sums of estimated remaining work and remaining resources define the starting point of the release's burn down chart – see for example week 0 in Figure 1, which shows the project's burn down for the first release.

Once the release planning is done, the plan becomes a commitment to the customer. It is also used to coordinate with other teams – for example, state that the project will be ready for integration in two months' time. Therefore, when remaining work does not shrink at the same rate as remaining resources – as happened for example at the end of the second iteration (week 4 in Figure 1) – an alarm was raised.

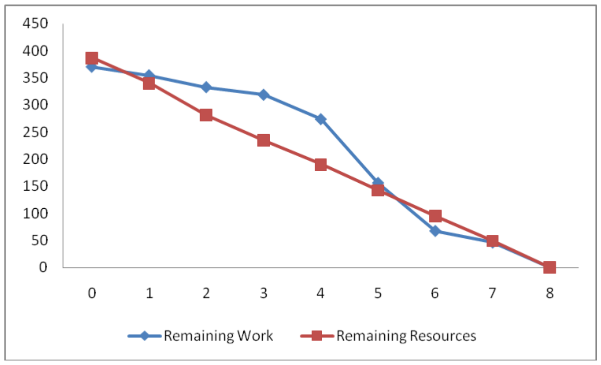

Figure 1: Burn down by week in the first release

Such a growing gap in a burn down chart can mean that the team is not delivering as fast as expected, or that there was a significant estimation error when a commitment for the release was made. In the case of the second iteration, it was a combination of both: The four week old team was still evolving the development and build environments, and inefficiencies still existed; on the other hand, some of the high-level commitments which were assumed (and estimated) as simple experienced feature creep during the release.

These facts were uncovered during the iteration summary meeting in which the customer was present. The discussion and reflection that followed resulted in two decisions: One was to reach for the help of senior engineers from other projects about the development environment and continuous build system, and the second was to reevaluate some of the new feature requests with the customer to make sure that the team is not realizing the risk of perfecting minor features.

As the burn down chart shows, these combined actions had the impact of reducing the remaining work to within the available resources within two weeks. This was reviewed by the team at the summary meeting of the third iteration, at the end of week 6.

Event II: Extend testing resources

The pulse metric was designed to monitor the risk of people sticking to their "pre-agile" habits of coding on a private branch and integrating with the rest of the team only late and when a delivery is due.

The practice of continuous integration means that in addition to maintaining a continuous build, the developers check in code into the shared integration environment at least once per day. Since this was a change in personal work habits for a majority of the team, the number of check-ins per days into the integration branch – defined as the *pulse* metric – was measured and discussed in each iteration summary.

The team's goal was to maintain a healthy, steady pulse – checking in regularly during the iteration, in contrast to having quiet periods followed by a surge of integration efforts towards the end of the iteration.

Figure 2 shows the pulse metric on a daily basis during the third iteration of the first release. While the pulse for software code is reasonable, an alarm was raised regarding the automated acceptance tests. Basically, one person was writing acceptance tests in the team, and was doing the majority of that work in the last two work days of the iteration, with check-ins happening on the very last day. In this case, that person raised an alarm even before the metrics were viewed – after working over the weekend still not all new code was tested, and even if it were such a work style was not sustainable.

This urgent issue triggered the project's governance mechanism – after the metrics were viewed in the morning, this was chosen as the subject of that afternoon's reflection meeting. In this case the team decided to fight for more testers to be available to it, train some of the developers to write acceptance tests, and allocate tasks in the next iteration to complete the testing of the new features which were not yet fully tested. These tasks received priority not only by the team but also by the customer, who is present when metrics are discussed and shares the interest of maintaining a fully tested code base, especially when the project is young.

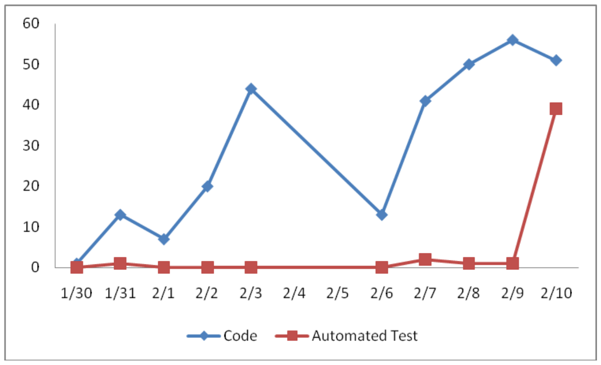

Figure 2: Pulse metric, daily in the third iteration

Figure 2 does not show the problem being solved in the following iteration. To see this we can look at Figure 3 that is explained as part of event III and presents the product size metric. After the minor increase in product size in iteration 1.3 there is a major leap in 1.4, which is the cumulative result of catching up on the missing 1.3 tests plus new fully tested features that are developed in 1.4.

Event III: Insist on delivery every iteration

The product size metric was designed to monitor the risk of not producing a customer-ready deliverable at the end of every iteration. Up to this project, the standard in the development organization was that new projects took months before delivering the first customer-facing features – and even then, those were "code complete" and not fully tested and ready for delivery.

Therefore, it was decided that the key velocity metric measuring the team's progress will not be in story points, work days or lines of code – and instead be based on the size of the automated acceptance tests suite. This was measured by the number of test steps that passed an automated test run on the integrated code base. This metric was intended to send several messages to the project team: Only tested features count, only customer facing features count, and only integrated code counts. When the project started, one professional tester was assigned to it, and the team's main goal for the first iteration was to deliver a greater than zero product size.

Figure 3 shows the *product size* metric over the first three releases. This metric raised an alarm once during the project: in iteration 2.3, the only one in which the metric declined compared to the previous iteration.

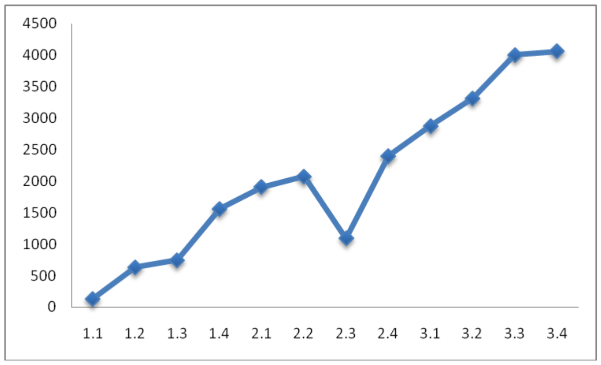

Figure 3: Product size metric across three releases

This was the obvious subject for the reflection meeting at the end of iteration 2.3 – a team's inability to deliver more value to its customer every iteration is a severe issue. The reason for this decline was the team's dependency on one high performing tester, who over time took it upon herself to write, maintain and run a majority of the team's acceptance tests. In the middle of iteration 2.3, this person was reallocated to help with an emergency in another project. At that point the team had less than a week – in which they had other committed work – to find a solution to this problem, which didn't happen until the iteration ended.

That reflection meeting focused on discussing ways to remove the team's dependency on that one tester in order to deliver. Developers and business analysts committed to write, maintain and run acceptance tests on their own. One person was appointed to distribute this workload and tackle roadblocks. The team decided to invest in getting trained on writing and running tests.

As Figure 3 shows, in the following iteration the product size was once again up, surpassing iteration 2.2. From that point on the team regained its velocity and continued a steady increase in product size.

Towards the end of the third release another tester was allocated to the team on a full time basis. At that point the team members still chose to do some testing on their own and train new developers on acceptance testing, to prevent recreating a dependency on one person to achieve the team's goals.

4. Role and Responsibilities

The role scheme and the responsibilities of each role holder differ among teams and organizations. Roles and responsibilities are continuously used and sometimes shaped according to governance events that occur. In this part we relate to the roles and responsibilities that were involved in the three governance events described in Section 3. Using this perspective enables a closer look at decisions that are made as part of the development process and the decision rights that are used.

In *Event I*, two decisions were made with respect to the release scope of requirements. The first decision is to add more role holders – experts from other projects – to serve as consultants. This means that role holders who are responsible remain, and can still be consulted in addition to the new role holders who do not have direct responsibility. The second decision was to re-plan the release scope. This was done by the business analysts and team leaders together with the customer who was accountable for the release requirements.

In *Event II*, the decision was to increase the resources available for developing automated acceptance tests. Before this decision was made, the main tester was solely responsible to develop and run the complete set of acceptance tests. After this event, developers were assigned to write tests and were responsible to develop and run them.

In *Event III*, it was discovered that the decision made in Event II was not enough. Although more people developed and ran tests, they did not take full responsibility for the entire set of tests, in particular those that were maintained by the main tester. So it happened that when that tester was urgently asked to help in another project, the tests were not maintained and run and this resulted in a poor delivery. The decision therefore strengthened the previous one by specifically emphasizing collective responsibility to maintain and run the tests before the end of each iteration, in order to deliver a high quality product on time, every time. Making this decision increased the commitment of the other team members and eliminated the dependency on the tester.

5. Summary

This paper presents a case study on governing an agile project. We relate to the different governance stages and mechanisms, including an examination of the roles and responsibilities that are involved.

We found that governance is tight and effective when performed at the level of each development iteration. The governance mechanisms are used to steer the development process and increase the quality of the developed product. In addition, we found that governance events can shape the responsibilities of role holders and hence improve the role scheme in use.

In this paper we examine governance at the level of single iterations. In future work we intend to analyze governance at the release and full project level. This would require looking at a longer time scale and at release- and project-level metrics and events.

6. References

[1] Beck, K. & Andres, C. (2004). *Extreme programming explained: second edition*. Boston, Massachusetts: Addison-Wesley.

[2] Broadbent, M. (1998). *Leading governance, business and IT processes: the organizational fabric of business and IT partnership*, Findings Gartner Group, 31 December 1998, document #FIND-19981231-01.

[3] Cantor, M. & Sanders, J. (2007). *Operational IT Governance*. Retrieved from: http://www.ibm.com/developerworks/rational/library/may07/cantor_sanders/index.html

[4] Chulani, S., Williams, C., Yaeli A., Wegman M. N., & Cantor M. (2006). *Understanding IT governance: definitions, contexts, and concerns* (Research Report RC24064). IBM. Retrieved from: http://domino.research.ibm.com/library/cyberdig.nsf/papers/38905EEA124CDDFB852571FE00569CCE/$File/rc24064.pdf

[5] Dahlberg T. & Kivijärvi H. (2006). *An integrated framework for IT governance and the development and validation of an assessment instrument*. Paper presented at the 39th Hawaii International Conference on Systems Sciences, Kauai, Hawaii.

[6] Dubinsky Y., Talby D., Hazzan O. & Keren A. (2005). *Agile Metrics at the Israeli Air Force*. Agile 2005 Conference, Denver, Colorado.

[7] Dubinsky, Y., Hazzan, O., Talby D., and Keren A. (2007) Transition to Agile Software Development in a Large-Scale Project: A System Analysis and Design Perspective, in the Advances in Management Information Systems (AMIS) Monograph Series, AMIS Systems Analysis and Design (SA&D) volume 1, chapter 5.

[8] Dubinsky, Y., Yaeli, A., Feldman, Y., Zarpas, E., and Nechushtai, G. (2008) Governance of Software Development: The Transition to Agile Scenario, *IT Governance and Service Management Frameworks and Adaptations*, Idea Group Publishing, Information Science Publishing, IRM Press.

[9] Ericsson, M. (2007). *The governance landscape: steering and measuring development organizations to align with business strategy*. Retrieved from: http://www.ibm.com/developerworks/rational/library/feb07/ericsson/

[10] Hazzan, O. and Dubinsky, Y., (2008). *Agile Software Engineering*, Springer-Verlag London Ltd.

[11] Highsmith, J. (2002). *Agile software development ecosystems.* Boston, Massachusetts: Addison-Wesley.

[12] Talby, D., Hazzan, O., Dubinsky, Y. and Keren, A. (2006) Agile software testing in a large-scale project, IEEE Software, Special Issue on Software Testing, pp. 30-37.

[13] Van Grembergen W. & De Haes S, (2004). IT governance and its mechanisms. *Information Systems Control Journal*, 1. Retrieved from: http://www.isaca.org/Template.cfm?Section=Home&Template=/ContentManagement/ContentDisplay.cfm&ContentID=16771

[14] Webster's Revised Unabridged Dictionary (1913). Edited by Noah Porter.

[15] Weill P. & Ross, J.W. (2004). *IT governance*. Watertown, Massachusetts: Harvard Business School Press.