Agile Metrics at the Israeli Air Force

Yael Dubinsky[1], David Talby[2], Orit Hazzan[3], and Arie Keren[2]

[1] *Department of Computer Science*
Technion – Israel Institute of Technology
yael@cs.technion.ac.il

[2] *MAMDAS – Software Development Unit*
Air Force, IDF, Israel
davidt@cs.huji.ac.il
ariekk@netvision.net.il

[3] *Department of Education in Technology & Science*
Technion – Israel Institute of Technology
oritha@techunix.technion.ac.il

Abstract

It is a significant challenge to implement and research agile software development methods in organizations such as the army. Since it differs from organizations in the industry and the academia, data gathered in the army and its continuous analysis may enrich the community knowledge abut agile methods. This work describes a research, conducted during an entire release, about one development team at the Israeli Air Force that works according to Extreme Programming. The establishment of this team and the investigation of the first release is part of a long-term process, started last year, aiming to reduce delivery time while raising communication and customer collaboration. Among several themes this research is concerned with, we focus on agile metrics and provide a metrics mechanism that was established and refined along the release development.

1. Introduction

Experience gathered by the community of software development practitioners indicates that the introduction of the agile software development method ([3]) Extreme Programming (XP) [1] into an organization goes along with conceptual and organizational changes that are an integral part of the process ([4], [8]).

The army is a large and hard-to-change organization with respect to fixed regulations, project approval, management method and organizational structure. Therefore, software development according to traditional software development methods, such as the water fall model and its variations, is still common at such organizations. These facts challenge the transition to agile software development methods. Further, though it seems that there are XP practices that fit the military culture (e.g., the *planning game* and *sustainable pace* with respect to time management and *collective ownership* where there is a special team spirit), and the benefits of XP projects are well known ([7]), the embracement of the complete XP method is a large-scale change, both to managers and developers.

"Betting" on the success of XP for a large-scale project, such as the one on which this paper focuses, was therefore considered a risk. Accordingly, the project began under close management supervision, with high hopes for being a prototype for the implementation of XP in other teams on one hand, and fear of incompatibility of XP to the army environment on the other hand. In order to cope with this dual perspective, presenting the method's benefits and pitfalls, a tight and continuous measurement of the development process was established. Therefore, it was a straight forward decision both to constantly evaluate the development process as part of the XP tracker role ([1]) and to conduct a systematic strategic research about the unique development environment (a specific military unit) in order to enrich the knowledge of the community of the agile software developers.

This paper focuses on a metrics mechanism that was established and refined during the first release of the project. In Section 2 we describe the research setting, elaborating on the research tools and methods. Section 3 describes the project setting, and the rational behind the metrics used. Section 4 presents the data that was gathered and its analysis. We conclude in Section 5.

2. Research Setting

This section describes the research background and flow. In Section 2.1 we describe the process of introducing the agile approach in general and XP method in particular into one of the Israeli Air Force units. In Section 2.2 we delve into the details of the research method specifying the research tools we used.

2.1 Research Background

The software project we deal with has been developed by MAMDAS - a software development unit in the Israeli Air Force. The project is developed by a team of 60 skilled developers and testers, organized in a hierarchical structure of small groups. The project develops large-scale, enterprise-critical software, intended to be used by a large and varied user population.

The forth author, who is in charge of the system engineering group of this project, was requested to lead a change in the current development process. This change aimed at implementing a new software development process that would enable a rapid response to customers' requests and requirement changes, and would obtain feedback with respect to released features. This change-oriented sub-project was named "Short-Cycles". Its duration was set to one year, in which a new methodology had to be suggested, and a pioneer team should start implementing it. It was clear that an organizational and conceptual change should take place. Since the entire team was relatively large and teammates had different individual interests, such a change could not be performed over night, but rather in a gradual, stage-based process which was to be planned very carefully.

It is important to note that the Air Force leadership supported the Short-Cycles initiative. Furthermore, the leadership specifically declared that, while a reduction in quantity might be accepted, quality and fitness to customers' needs were not to be compromised.

One of the decisions of Short-Cycles management was to learn about the agile approach in general and about XP in particular. As a result of this decision, an XP workshop, facilitated by the first and the third authors, was conducted in the summer of 2004. This workshop was attended by project members, who will be able later to evaluate and decide upon the sequel of Short-Cycles. A report about the workshop which includes participants' reflection is presented in [2].

During December 2004, the first XP team was established with the second author as the project leader. Besides the coach, among 15 teammates, 7 teammates are developers who work full time and 8 teammates are system engineers, system analysts, developers and testers who work between 30% and 60% on this project. This team should evaluate the effectiveness of XP for the said project. It was encouraging to observe that after the first two weeks iteration "managers were very surprised to see *something running*" and everyone agreed that "the pressure to deliver every two weeks leads to amazing results" [quotes from team members reflections]. Still, accurate metrics are required in order to take professional decisions, to analyze long-term effects, and to increase confidence of all management levels with respect to the process that XP inspires.

2.2 Research Method

Two main research approaches are used by us in the investigation of the process of implementing XP in the team's work. The first one is a *qualitative* approach in which we seek to understand the process from the participants' point of view. Accordingly, we ask the team members questions such as "How do teammates conceive the change?", "What process characteristics can and should be measured?", "In what frequently should each metrics be measured?" and so forth. The second approach is a *quantitative* one, in which we aim at measuring the effectiveness of the process. Accordingly, we look for ways to answer questions such as "What is the work progress?", "What is the status of the project resources?", "What is the quality of the product?", "How continuous is the integration?", etc.

Two perspectives are used to analyze the gathered data. The first one is the *insider* perspective performed by the second and the forth authors who play key roles in the process itself, were familiar with the technical aspects of the project as well as with its managerial and social ones. The second author - the team leader - acted also as the tracker during the first release. The second perspective is an *outsider* perspective carried out by the first and third authors who, as consultants, analyze the process without being part of it.

Following are the research tools that are used for data gathering:

Observations: Every two weeks one day is observed from both insider and outsider perspectives. This day includes a presentation of the work of the previous iteration and the planning game for the next iteration.

These activities are attended by the customer, all teammates and representatives of the project management. The consultants participate in some of these days.

Reflections: Every two weeks, before the next planning game is started, the consultants conduct a reflection session in which the participants of the above mentioned days are asked to reflect on the process and on their activities within the process. The discussions raised are documented.

Questionnaires: Participants answer open and close questions on process related subjects.

Interviews: Two interviews with the team leader (the second author), which focused on the evolution and implementation of the quantitative metrics mechanism, were conducted by the first author.

Quantitative metrics: Quantitative data is gathered by teammates' reports as well as by automated data retrieved by the development environment.

Some of the research tools, like observations, reflections and quantitative metrics, are used in an iterative, cyclic manner in each iteration. This enables us to gather and analyze extensive data that can be analyzed throughout the process time scale.

3. Project Setting and Origin of Metrics

3.1 Project Setting

Like many organizations making their first attempt to introduce XP, this project started at an organization with a rigid, well established, highly managed development process. While this process lacks agility and enforces high overhead, it does provide answers to many managerial and developmental questions. There are ready-made tools, document templates and tutorials to support planning, progress tracking, formal analysis & design artifacts, fault management, configuration control and so forth.

The XP principles contradict many of these existing work processes, and so the XP team was given a waiver on most of the existing regulations. This meant that we were able to design our own work processes, as long as we could show our management that the project *is* managed – effectively run and under control. One of the fortunate results of this waiver was that the metrics presented in this paper are highly integrated with the way the project is actually managed. The next section describes the rational behind each metrics, and elaborates on this issue.

In contrast to the relative freedom in the process aspect, we are bounded by a unique set of technical constraints. Our project is built based on a large-scale in-house object-oriented framework [6], which handles many of the underlying technical aspects of the system. The framework contains rich functionality, and enables the delivery of usable application features in record time, as well as a significant cut of the entire project's cost. The framework also has a profound effect on the development process, in two major aspects.

The first aspect is **formal detailed specifications**. The framework relies on a metadata repository [10], which contains most of the system's specifications: data entities, data types, actions, transactions, user types and privileges, messages, external interfaces and so forth. This data is edited in the repository, in formal forms – in contrast to free-text documents – and much of it is used to automatically generate code and other files. This has the benefit of eliminating the need to manually code these specifications, and to test this manual translation. The framework maximizes this benefit, by formalizing as much of the detailed specifications as possible. The result is that our development process does not start with ordinary design and then coding – it starts with design, continues with writing the detailed specifications in the metadata repository, and only then in coding those parts of the specs that are not automatically generated.

Automated acceptance testing [9] is the second aspect in which the framework affects our process. This is a tool that supports writing a test scripts, such that each test step – for example "login", "do action" or "verify field value" – is written in a formal yet human-readable way. The framework can execute these formal test scripts, supporting both batch and interactive modes of test execution. This saves a very large amount of effort, spent on running acceptance tests manually, and also provides other benefits to the development process (see [9]). In large organizations, such as ours, acceptance tests are the responsibility of a dedicated QA group, but in this project the acceptance testing framework enabled moving this responsibility into the XP team – making it responsible for building the product all the way from detailed specifications to production-quality testing.

It is important to note that the above paragraphs are presented to explain the environment in which we operate; they do not impair the wide applicability of our results. Each project has its technical constraints, and must tailor and constantly refine XP metrics to its specific needs. Due to this perspective and the size of the project, existing tools like for example XP-EF (Williams et al.) seem not suitable in this case. The next section defines the metrics we used, and starts each one by analyzing *why* it was needed. We suggest that these considerations, and the results that follow, are typical to XP introductions in large and conservative organizations.

3.2 Origin and Goals of Metrics

As mentioned earlier, the project was the first large-scale attempt at XP in that military unit, deviating heavily from existing work policies, and thus closely watched as risky. Therefore, the design of the right metrics began with a project risks analysis; a metrics was added where it seemed valuable in reducing a risk. Although existing software metrics were studied [5], we preferred to design our own metrics, to fit the exact needs as dictated by the risks list.

Metrics can be used for three purposes:

1. To communicate to the team which behaviors are most valued, or most problematic.
2. To enable faster and more accurate decision making by the project's leadership.
3. To communicate information about the project to upper management.

While all three reasons are important, our experience from the first release is that the first one is the most important at this initial stage, when XP is introduced to a team. To an extent, people and teams do behave as they are measured. We believe that this by itself is a strong motivation for any XP team to define and refine own metrics.

3.3 Definition of Metrics

In this part we describe four metrics and the kinds of data that are gathered to calculate them. These four metrics present information about the amount and quality of work that is performed, about the pace of the work progresses, and about the status of the remaining work versus remaining human resources.

Product size, initially just called 'Product', is the first metrics. It aims at presenting the amount of completed work. The data that was selected to reflect the amount of work is the number of test points. One test point is defined as one test step in an automatic acceptance testing scenario ([9]) or as one line of unit tests. The number of test points is calculated for all kinds of written test and is gathered per iteration per component. Additional information is gathered with respect to the number of test points for tests that pass, the number of points for tests that fail, and the number of points for tests that do not run at all. As was presented by the project leader - "This is not a quality metrics. For quality metrics we count faults."

The initial risk that the product size metrics was designed to reduce was the inability to measure the progress of the XP project, and thus the inability to compare its velocity to that of the organization's 'traditional' development. The advantage of test points is that the amount of acceptance tests for a given feature is usually proportional to the feature's size and complexity. This cannot be said on the count of lines of code, or lines of specifications.

Another high risk that the product metrics deals with is un-thoroughly tested product, caused by people not writing or running enough tests. Most people do not like writing tests, and many of the team's members had experience in other projects in the organizations, in which they developed artifacts which were tested later (for example by the QA team). This product size metrics was very effective in making two important points to the team. First, test points are the *only* metrics of productivity in the project – nothing partial (like running code, for example) counts. Second, regular regression testing is a must – since a test point must be run (and pass successful) each iteration again to be counted as part of its product.

Pulse is the second metrics, which aims to measure how continuous the integration is. The data is automatically gathered from the development environment by counting how many check-in operations occur per day. The data is gathered for code check-ins, automatic-test check-ins, and detailed specifications check-ins. When referring to code in this paper we mean code plus its unit test.

The risk that the Pulse metrics was designed to monitor is high overhead due to lack of continuous integration. XP requires a mindset that is very different than what people were used to: instead of completing a two-week specifications task, now an entire iteration is just two weeks long. This means that a full cycle of specification-coding-testing must be completed within these two weeks, and usually more than once per teammate. In the way the configuration tool was used, checked-in changes in files are visible to all , so check-in is the basic method of integration.

Therefore, the initial role of the Pulse metrics was not to measure the amount of work, but instead to verify that it spread evenly across iterations. "Steady" pulse means that pulse is even across many days; it is the good status. "Spiky" pulse means that most check-ins are grouped at the end of iterations, meaning that people don't integrate enough during iterations; it is the bad status.

Burn-down is the third metrics. It presents the project remaining work versus the remaining human resources. This metrics is supported by the main planning table that is updated for each task according to kinds of activities (code, tests, or detailed specifications), dates of opening and closing, estimate and real time of development and, the component that it belongs to. In addition, this metrics is supported with the human resources table that is updated when new information regarding teammates' absence arrives. This table also contains the product's component assigned

to each of the teammates and with the percentages of her/his position in the project. By using the data of these tables, this metrics can present the remaining work in days versus the remaining human resources in days. This information can be presented per week or for any group of weeks till a complete release, both for the entire team or for any specific component.

The burn-down graph answers a very basic managerial question: are we going to meet the goals of this release, and if not, what can I do about it? The risk behind the metrics is the opposite – the risk of missing release goals due to the lack of a clear view of progress during the release. The burn-down is useful for both the team's leaders, for example changing teammates' tasks according to high priority components during the release, as well as for upper management, to easily verify whether the release is on track.

Release goals were set before each release – each goal is a high-level feature. Goals are defined by the user, and are verified by matching a rough estimate of the effort required to complete each goal (given by the development team) to the total available resources. Once goals are defined and estimated, both remaining work and remaining resources are based on this initial estimation, which is refined as the release progresses.

Faults is the fourth metrics, which counts faults per iteration. During the release on which this paper focuses all faults that were discovered in a specific iteration were fixed at the beginning of the next iteration. The faults metrics is required to continuously metrics the product's quality. Note that the product size metrics doesn't do it, since although it metrics test points, it does not correlate between the number of failed or un-run test steps to the number of actual bugs.

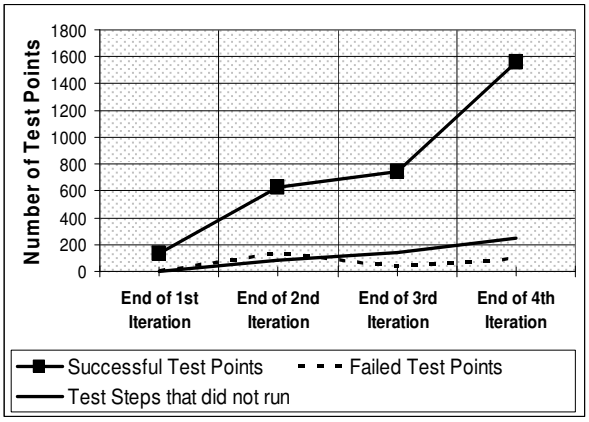
4. Data and Analysis

Data was gathered throughout the first release, which had four iterations. The metrics were presented once in two weeks before the planning game, and were also continually available on the project's intranet. Most planning-game participants have reported that they are viewing the metrics status only when presented once in two weeks. The project leader however stated that the metrics cause change in behavior especially in the matter of writing more tests.

4.1. Product Size Metrics Analysis

Usually we are not used to view software product size using its tests capacity. However, this is most interesting aspect of the Size Metrics. Figure 1 shows a global view for all four iterations presenting an interesting situation of growing numbers of test points as the product is developed. One of the reasons for this growth is the relatively small number of testers' hours for automatic test writing that were allocated to the project at first, and soon turned to be a bottleneck. In the third iteration, for example, not all coded features were tested, and accordingly the Size metrics showed only a small increase.

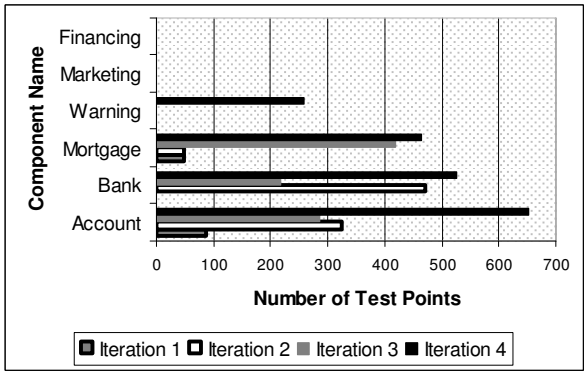
Figure 1: Size Metrics during the release



Consequently, it was decided that at the beginning of the fourth iteration the main tester will teach the developers to write automatic test scenarios for their code. During the fourth iteration she taught developers, so wrote fewer tests by herself. The end result was a sharp increase in product size during the iteration.

During the first week of the third iteration many teammates were absent since they participated in a routine training. Indeed, we can see low number of tests considering that more developers have started to write tests. The tests are written for every additional code and there is an option to drill down in data in order to observe the work capacity according to components. Figure 2 presents the number of test points per component per iteration.

Figure 2: Size Metrics according to components



It can be observed that the forth iteration was the most fruitful. New components were dealt with, while increasing number of test points of other components as well. This shows a mature stage of the process. Indeed, after the development environment is stable, the testing process and knowledge caught us with the rest of the development activities. The components' names were changed for security reasons.

4.2. Pulse Metrics Analysis

When the Pulse Metrics was first presented a slight of resistance to this metrics was expressed. Several teammates said that it does not reflect continuous integration. One team member said that it is a twisted metrics that searches for rating like web surveys, and, accordingly, he suspects that teammates will simply click for check-in operation just to raise the count. The value of courage was in the air when others insist on understanding why someone would do this. All sounds of resistance were gone when teammates observed that first, only real check-in operations are counted – meaning there was a change in the part that is integrated – and second, that this metrics says even more than continuous integration, that is, actually continuous work. Figure 3 shows the Pulse Metrics for the entire release.

As can be observed the first week of each iteration always has fewer check-in operations than the second week of the iteration has. Also, the forth iteration looks as if the work was best distributed among iteration days. We can see the reduction in work at the third week and the pick as a result at the forth week when teammates were back at work.

Figure 3: Pulse Metrics during the release

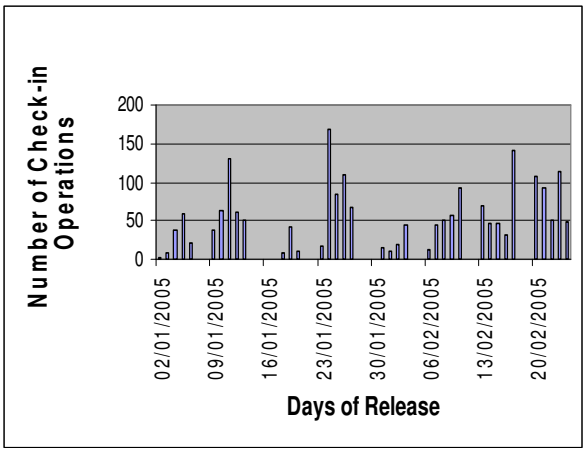
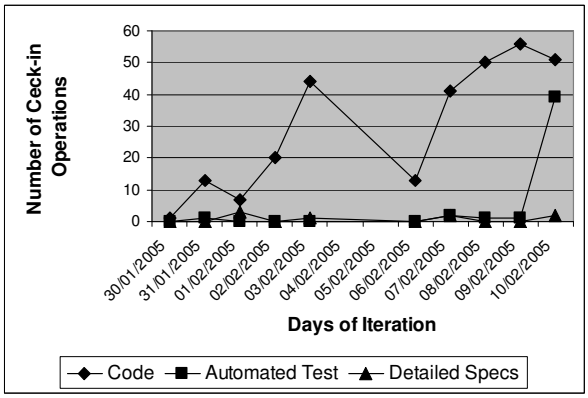


Figure 4 presents the Pulse Metrics of the third iteration with the details of the different check-in operation kinds. Note that check-ins of code files are the most prevalent, because code tends to be spread over more files than specifications of test scenarios.

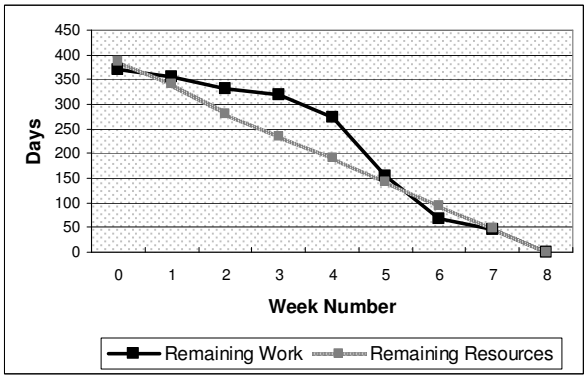
Figure 4: Pulse Metrics of iteration 3



4.3. Burn-down Metrics Analysis

The Burn-down Metrics is a classical managerial metrics that shows whether our plans can actually be performed. Figure 5 presents the Burn-down Metrics for the entire release. As explained in section 3.2, these are derived from the estimates of the release goals, and from the total resource allocation for the entire release. Each two columns represent the data that was known at the beginning of the specified week. At the beginning of the first week, before the release has started, there were 387 days as resources for the entire release and only 370 days of estimated work. During the release the number of days as resources was reduced while development work was performed. The data on the last eighth week shows the eighth week itself where number of days as resources is 49 and the estimated work is 46.25 days.

Figure 5: Burn-down Metrics during the release



This kind of Burn-down Metrics gives a two-months view on the development process and is very successful as a plan chart. Figures 6 and 7 presents drill down data of the fifth and sixth weeks that are part of

the third iteration. These figures present the inner data of the Burn-down metrics that shows the remaining human resource days versus the remaining work days for each of the product's components.

In addition to the product's components, overhead was also referred to. Overhead means training sessions, the planning game days, coordination meetings, and other activities which are not development. People were not allocated specifically for the 'overhead' component – this is why its remaining resources are zero all along – instead, it is meant to be spread relatively equally across all team members. The simplest and most effective way to achieve this was to require a small positive gap between resources and work in each component, which is reserved for the shared overheads. Note that the cumulative burn-down chart (shown in Figure 5) does contain the gap and is therefore exactly accurate.

As can be observed the variation is significant when looking according to components, illustrating the strength of this metrics with respect to the ability to decide on human resources mobility. For example, in Figure 6 the 'Mortgage' component was way behind its goals in week 5, and this was identified and fixed in week 6, by a combination of added human resources and reduction of features for the release. Also note that the problem was not visible from looking at the cumulative burn-down chart alone, since the 'Bank' and 'Account' components have a surplus of resources in week 5 that cancels the lack in the 'Mortgage'.

Figure 6: Burn-down Metrics at Week 5

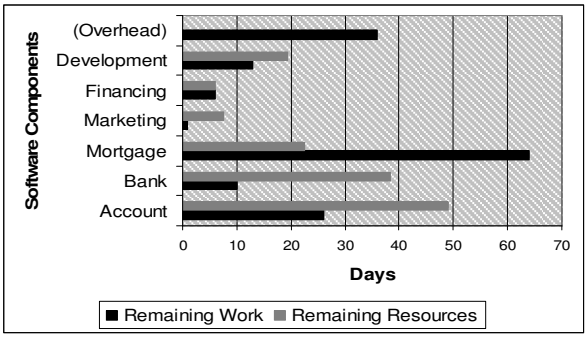
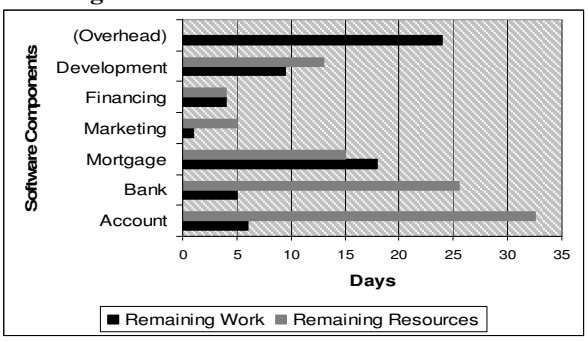


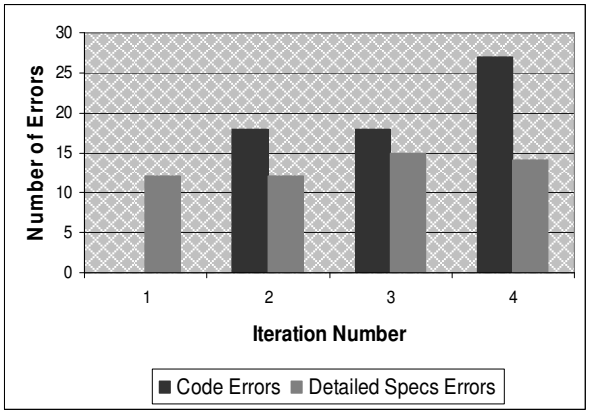
Figure 7: Burn-down Metrics at Week 6



4.4. Faults Metrics Analysis

The Faults Metrics is a standard quality metrics that presents the number of faults that are found and their kind. It can be a coding error or a detailed specifications error. Figure 8 presents the number of faults per iteration and their distribution. Note how spec errors are common at the start of the project, when many team members were inexperienced with using the framework, and slowly reduce their relative rate.

Figure 8: Fault Metrics during the release



4.5. Reflection

In this part we present teammates and management reflections on the process. The most impressive observation retrieved in a reflection meeting which took place after the first release, was that experienced participants were very satisfied from the XP process – more than younger ones. This observation can be explained by the fact that the experienced participants had previous experience to relate to. The experienced participants emphasized the real feedback they get every two weeks, the fixed dates of delivery, the ease of combining inexperienced people in the project, and the way they are aware of problems almost immediately when they occur. Younger participants were satisfied from the direct communication and connection with the customer and from the process itself. Most developers wrote the word "people" answering what they liked most thus XP was really good in melting the team in a very short period.

The feedback on the Size Metrics was that it motivates writing tests and that it can be referred also as complexity metrics. If a test scenario has twice as many step as another scenario, it is considered to be more complicated, about twice as much. However, as for unit tests, the developers who did write them said that each unit test line should be worth 2 test points, since unit tests are often more difficult to write and test

more subtle bugs (multi-threading issues, for example) than do acceptance tests. This issue has been left as an open question in the project so far.

The feedback on the Pulse Metrics was that it did not influence the development flow mainly since the main risk that it was designed to monitor – that teammates will integrate only at the end of the iteration – did not happen in practice. Still, it was decided to continue monitoring this metrics.

The feedback on the Burn-down Metrics was very interesting yet expected. Managers said it is important, and helps in making decisions, while others said it is not important. Managers also claim that this metrics can help scaling XP, for example to manage several teams developing a single large project.

5. Conclusion

In this paper we present the results of a research conducting at one of the software development teams of the Israeli Air Force. The use of the presented metrics mechanism increases confidence of the team members as well of the unit's management with respect to using agile methods. Further, these metrics enable an accurate and professional decision making process for both short- and long-term purposes.

In future work, we plan to continue refining the metrics mechanism while investigating its scalability, both on the time axis with more releases to come, and on the size axis, as more teams under this project join the XP method.

6. References

[1] Beck, K., *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2000.

[2] Y. Dubinsky, O. Hazzan, and A. Keren, "Introducing Extreme Programming into a Software Project at the Israeli Air Force", *6th International Conference on Extreme Programming and Agile Processes in Software Engineering,* 2005.

[3] Highsmith, J., *Agile Software development Ecosystems*, Addison-Wesley, 2002.

[4] K. Johnsen, R. Stauffer, and D. Turner, "Learning by Doing: Why XP Doesn't Sell", *Proceedings of the XP/Agile Universe*, 2001.

[5] Kan, S., *Metrics and Models in Software Quality Engineering (2nd Edition)*, Addison-Wesley 2002.

[6] Mohamed, F., Schmidt, D., and Johnson, R., *Building Application Frameworks*, Wiley 1999.

[7] D.J., Reifer, "How to get the most out of Extreme Programming/Agile Methods", *Proceedings of the 2nd XP Universe and the first Agile Universe Conference*, 2002, pp. 185-196.

[8] H. Robinson, and H. Sharp, "The Characteristics of XP Teams", *5th International Conference on Extreme Programming and Agile Processes in Software Engineering,* 2004, pp. 139-147.

[9] D. Talby, et al, "A Process-Complete Automatic Acceptance Testing Framework", *SwSTE*, 2005.

[10] D. Talby, D. Adler, Y. Kedem, O. Nakar, N. Danon and A. Keren, "The Design and Implementation of a Metadata Repository", *INCOSE/IL,* 2002.

[11] L. Williams, L. Layman, and W. Krebs, "Extreme Programming Evaluation Framework for Object-Oriented Languages - Version 1.4", *North Carolina State University Department of Computer Science*, *TR-2004-18*, 2004.