Implementing Program Model with Agile Principles in a Large Software Development Organization

Maarit Laanti
Nokia Corporation
Maarit.Laanti@nokia.com

***Abstract* - Organizations developing software have increasing interest towards deployment of agile methods. However, there is a problem in scaling up scrum and other agile methods, since these were originally meant for individual team scope. It is not enough to deploy agile methods only on the team level – because of the dependencies that teams have between each other when developing large software systems, and because the software engineering teams need input for their decision making (or actual decisions) and need to be synchronized with other activities. We present one way of scaling up scrum to program level consisting of several scrum teams, concentrating on the changes done in the process front-end. We also describe a new artifact called Agile Policy that was created for large-scale agile deployment.**

1. INTRODUCTION

Software industry has increasing interest towards the deployment of agile methods [1]. However, it is not a straightforward task to start the deployment of agile methods, because of scaling problems. Agile methods have a history of being applicable in small teams [2] yet it might be that larger organizations will actually benefit more from using these methods, since in a large program the probability of change is bigger– assuming that requirements change at a constant rate. Previous research does not offer much help in scaling up agile methods. Few research papers exist on this subject, and agile methods in general lack proper research [3, 4].

Before agile deployment in our organization, some teams had applied scrum on the team level, despite the surrounding organization that was working with a waterfall software engineering model. According to some informal interviews we carried out in early 2007, deployment of scrum only at a team level had led to some problems, as the overall program decision making was not synchronized with the team decision making. As an example teams were making up priorities since no absolute prioritized list of requirements was available for them, they were not able to drop less important features, and they did not actually have a product owner. Shortly, the scrum teams were lacking external program-level guidance that would have supported the teams' operation in scrum mode. In scrum, the product owner makes the decisions [5]. In a large organization, especially when use of scrum has originated from the development team itself, the product owner may be hard to find. In practice this often results in use of a proxy instead of a real product owner. Sometimes there has been a solution that the program manager has a dual-role also acting as a product owner. Experiences from this kind of solution have not been good.

Using scrum only on the team level had led to additional problems. Software interfaces were not properly synchronized with other development teams. Larger features were causing synchronization problems due to dependencies in code. Synchronization to the main waterfall program was considered problematic. Deploying scrum on the team level had brought flexibility in the team, but the ability to embrace the change was not truly benefiting the whole program as it was lacking capabilities to systematically embed the changes in all teams and levels. Agile, by definition, means the adaptability and ability to respond to change [6] as well as more frequent communication among the different stakeholders plus measuring the progress with the amount of code rather than the documents produced. For these and several other reasons our organization decided to start deploying agile methods along with the program model.

2. A PRACTICAL APPROACH TO DEPLOYING AGILE PROGRAM MODEL

Few published experiences on scaling agile existed at the time when we decided to deploy agile with program model in spring 2007. In her book Jutta Eckstein [7], describes how communication can be solved in a large agile program. Dean Leffingwell describes a practical approach to the question in his book Scaling Software Agility [8]. Few research papers exists on this subject [9, 10] with some more journal articles on experiences [11, 12] but according to our experience, every company or organization needs to find out its own adaptation based

0730-3157/08 $25.00 © 2008 IEEE
DOI 10.1109/COMPSAC.2008.116

on their own business, operative model, and needs. Shortly after our decision was made, Ken Schwaber published his book Enterprise and Scrum [13], which made us believe more strongly that we were on the right track.

We often hear claims that scaling agile is the last thing to do [2]. When scrum is scaled up, the practical advice is to use scrum-of-scrum masters instead of program managers, a single product owner, and a single backlog for the whole program [15].

Despite all this, we decided to take a different approach. We saw scrum as a software team management method affecting mostly on the team level although deploying agile would likely have impact in almost everything we do. In our organization we had teams serving many programs, and programs consisting of many teams, we decided to use two sets of backlogs on the team level and two sets of backlogs on the program level (Figure 1). The teams serving several programs were picking up their user stories from several sources and prioritizing those.

We decided to call the topmost backlog on the program level a "program content backlog" containing the highest level of requirements. The program content backlog would contain all the requirements defined for a program when a program was started in priority order. During the program, the requirements could be changed and re-prioritized, if necessary. For each program release, a program backlog based on the requirements would be created containing all user stories needed in the next release. The program backlog would be different to program content backlog in a sense that it would contain all program tasks (such as test environment creation and documentation) that would be needed in accomplishing the program mission. No Gantt chart would be needed any longer in our programs (Figure 1).

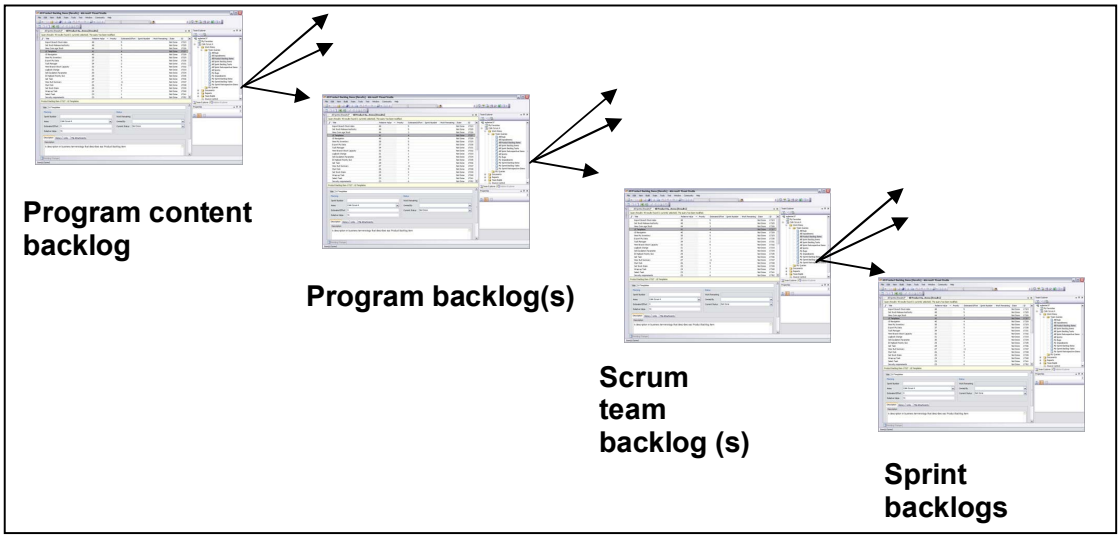

Figure 1. Program and team backlog structure consisting of four levels of backlogs

On a team level there would also be two sets of backlogs, namely Scrum Team Backlog containing user stories for that team (just like Product Backlog in scrum) and Sprint Backlog containing tasks (like in scrum) [15]. The Scrum Team Backlog was needed, since we had some teams that were working for multiple programs (though we decided that we will aim at dedicated resources for programs in the future). All these backlogs needed prioritization, so a hierarchy of product owners was needed to support a single program. We decided to call them Program Product Owner (having the ultimate responsibility of a program) and Team Product Owners (following orders from Program Product Owner). And we decided to have a Program Manager, responsible for the program execution and playing Srum-of-Srum Masters role to the (team) Scrum Masters.

We tried to maintain and multiply the scrum ceremonies [15] as much as possible: each team would have their daily scrums, sprint plannings, retrospectives, reviews, and demos. Each program would have a program scrum once or twice a week. We decided to repeat the other scrum ceremonies on the program level also: release planning would be participated by all scrum masters as well as the Program Manager and a Program Product Owner. A program release planning meeting would always precede the team level sprint planning meeting so that the team scrum masters could familiarize them with the user stories in that program release before the team level sprint planning. After the release there

would also be a review and retrospective sessions on the program level as well as a demonstration of the working software.

All these new practices were institutionalized with the Agile Policy (Appendix A).

3. DEPLOYMENT RESULTS

In order to support programs in this new approach, we implemented spreadsheet-based tools for all these backlogs. The feedback received from the programs has been mainly positive. The best benefits seem to be 1) better visibility on what is happening in a program via the program content backlog and the program backlogs and, 2) the adjustment of program against the existing reality, i.e. the ability to better steer what is happening in scrum teams by continuous planning. Figures 2 and 3 show the deployment rate of program and team level backlogs respectively.

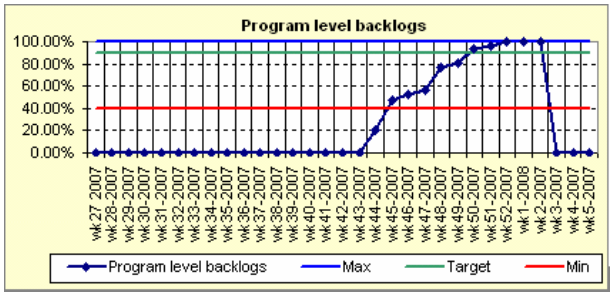

Figure 2. Backlog deployment statistics at program level

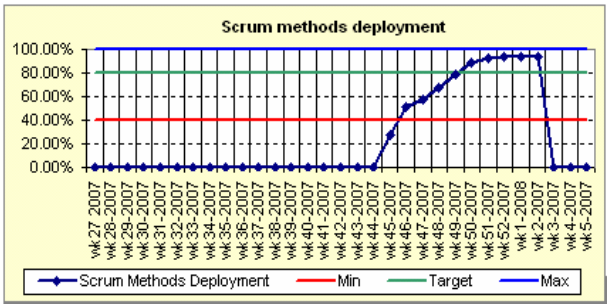

Figure 3. Backlog deployment at scrum team level

4. LESSONS LEARNED

Agile deployment in our organization was hampered by some questionable sidetracks. Some practitioners misunderstood agile to denote hacking; some associated it with no obligation to document what has been implemented. Some teams were suffering of the old habits, like the scrum master dictating the tasks the team members should be doing. We tried to combat these false understandings by arranging public lectures about agile in all our development centers.

One of the management concerns when transitioning to agile was the conflict between long-term and short-term planning. One of the threats seen was that backlogs and agile planning would give us only short-term visibility. We had many experienced program managers, who were able to estimate the size and duration of new programs based on their earlier experiences, and we saw no reason to leave that experience unutilized. Thus we decided to keep those both: long-term vision based on experience, and short-term "strategy" based on backlogs. This has resulted in more time spent on actual planning but also in improvement in the plans themselves.

For practical reasons, some scrum teams needed to work for several programs. Sharing resources has not been seen as a good option by the programs, as it results in non-steady output level and makes estimation harder. Programs with less dedicated resources have not made very good progress. In fact, agile program model deployment has revealed the overload of tasks to programs and teams and the true development velocity and capability.

Scrum teams have a sprint demonstration of working code at the end of each sprint. Programs arrange working code demonstrations of new features too, which has been received with great interest by program customers. Working code demonstrations have also been a visible place to communicate the new agile program mode to our customers, and let them really see the change and the fact that they can truly have an influence on the program outcome. Despite the popularity of the program demonstrations, in practice we have learned that we should not arrange too many of those but rather more sparingly. There should be working code demonstrations no more often than once every three months in a program. That is because arranging a proper demonstration takes time and effort from both the program and the attending customers. Most of the programs make enough progress in two months for the new features to really interest the demonstration audience.

5. DISCUSSION AND FURTHER DEVELOPMENT

The metaphor for scrum is the one of a control loop [5]. We see our agile programs as two nested control loops, where the shorter inner loops mean scrum teams and the longer outer loop is the control loop of the program. This semantic model is simple enough to promote the understanding of what we would like to achieve with the different backlogs on program and team level. It has astonishing similarity with the famous *Sashimi model* [16]. In fact, we have come to believe that they are more or less the same.

In the mid of the deployment of our agile program model we found out that a quite a similar approach had been taken in use in one program at Nokia-Siemens-Networks led by Petri Taavila. He has managed three successive completed programs with a similar approach that has been very successful in terms of productivity and the accuracy of the program outcome (i.e. how accurately the program outcome fits the customer needs). We also found a remark by Paul E. McMahon that in large programs it is necessary to have multiple task lists. [17] In fact, Paul McMahon is actually also advising scaling agility based on multiple task lists.

We are now collecting actual metrics on the validity of the new agile program model. A survey with 96 respondents from one agile program shows that people feel more productive in this new mode and the subjective feeling of quality has increased (Figure 4).

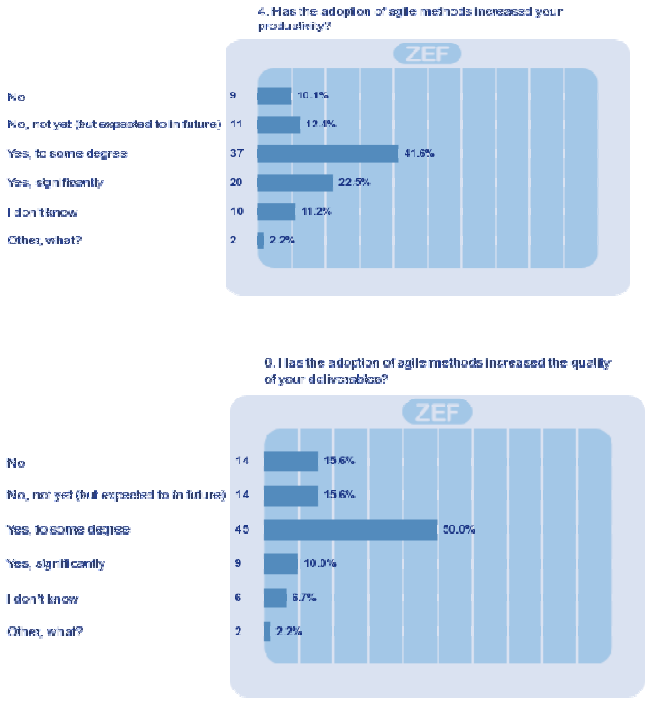

Figure 4. Agile Survey results.

6. CONCLUSIONS

At the time of writing, none of the started programs in this new mode has yet been completed. Thus there is no measured evidence except the qualitative data mentioned above of the superiority of the improvements generated by the agile program model. Anecdotal evidence from program managers and their superiors has been positive.

The deployment of agile program model has been wide, covering more than 60 independent programs, their size varying from tens of developers up to 150 developers. Thus we believe the agile program model could be applicabile in other large software organizations.

Software crisis [18] has led to growing need of software leading to longer and longer software development projects. If changes to software requirements happen at a constant rate and probability, longer development programs need agile methods more than shorter ones. Software business has become also more competitive [19]. This means that companies must resolve the gap of productivity with some means, for example with agility. Traditional requirements freeze and change management will become too costly and slow. This paper has presented one practical method of deploying large-scale agility in a large software development organization.

We would like to thank Petri Taavila for his input in improving our model and the change of using his program as a reference. We would also like to thank the agile community, from whom we have been privileged to learn the essence of agility. We would also like to thank Petri Kettunen for his constructive suggestions to this paper.

REFERENCES

[1] Schwaber Carey, "The truth about agile processes". *Forrester research*, August 2007.

[2] Kähkönen Tuomo, "Agile Methods for large organizations – building communities of practice". Proc. *Agile Development Conf.* (ADC), 2004, pp. 2-10.

[3] Salo Outi, Abrahamsson Pekka, "Empirical Evaluation of Agile Software Development: The Controlled Case Study Approach", *Product Focused Software Process Improvement*, p. 408, Springer Berlin / Heidelberg, Volume 3009/2004.

[4] Abrahamsson Pekka, Warsta Juhani, Siponen Mikko T., Ronkainen Jussi, "New Directions on Agile Methods: A Comparative Analysis," *icse*, p. 244, *25th International Conference on Software Engineering* (ICSE'03), 2003.

[5] Schwaber Ken, Beedle Mike, "*Agile Software Development with Scrum*". Prentice Hall, 2001. ISBN-10: 0130676349.

[6] Conboy K., Fitzgerald B., "Toward a conceptual framework for agile methods: a study of agility in different disciplines," *Proc. ACM workshop on Interdisciplinary software engineering research* (WISER), 2004, pp. 37-44.

[7] Eckstein Jutta, "*Agile Software Development in the large: Diving into the deep*", Dorset House Publishing, 2004. ISBN 0-932633-57-9.

[8] Leffingwell Dean, "*Scaling Software Agility: Best Practices for Large Enterprises*", Addison-Wesley Professional, 2007. ISBN-10: 0321458192.

[9] M. Lindvall, et al., “Agile software development in large organizations,” *IEEE Computer*, vol. 37, pp. 26-34, December 2004.
[10] Kettunen Petri, Laanti Maarit, “Combining agile software programs and large-scale organizational agility”. *Software Process: Improvement and Practice*, vol 13, nr 2, pg 183-193 2008. http://dx.doi.org/10.1002/spip.354.
[11] McMahon Paul E., “Extending agile methods: a distributed program and organizational improvement perspective,” *CrossTalk*, vol. 18, pp. 16-19, May 2005.
[12] Highsmith J., Agile for the Enterprise, “From Agile Teams to Agile Organizations”, *Cutter*, 2005. http://www.cutter.com/program/ fulltext/reports/2005/01/index.html .
[13] Schwaber Ken, “*Enterprise and Scrum*”, Microsoft Press 2007, ISBN-10: 0735623376.
[14] Kähkönen Tuomo, “Agile methods for large organizations – building communities of practice,” *Proc. Agile Development Conf. (ADC)*, 2004, pp. 2-10.
[15] Schwaber Ken, “*Agile Program Management with Scrum*”, Microsoft Press 2004, ISBN-10: 073561993X.
[16] Takeuchi, H., Nonaka I, “The New Product Development Game”, *Harward Business Review*. Jan 1, 1986.
[17] McMahon Paul E., “Lessons Learned Basics Affected When Using Agile Methods on Large Defense Contracts”, The Journal of Defense Software Engineering. STSC *Cross Talk*, May 2006 Issue.
[18] Feller, J., Fitzgerald, B.. “A framework analysis of the open source software development paradigm”. In *Proceedings of the Twenty First International Conference on Information Systems* (Brisbane, Queensland, Australia). International Conference on Information Systems. Association for Information Systems, Atlanta, GA, 58-69, 2000.
[19] Highsmith, J., Cockburn, A., "Agile software development: the business of innovation," *Computer*, vol. 34, no.9, pp.120-127, Sep 2001.

APPENDIX, AGILE POLICY

Agile Policy

1. ABOUT THIS POLICY

1.1 Purpose

This policy provides the criteria and guidelines for applying Agile Principles, Values and Practices in our organization. It starts where Agile Manifesto and principles ends, defining the level at which we may value:

- *"Individuals and interactions over processes and tools.*
- *Working software over comprehensive documentation.*
- *Customer collaboration over contract negotiation.*
- *Responding to change over following a plan.*

That is, while there is value in the items on the right, we value the items on the left more."

1.2 Goal

The goal of applying the Agile Principles, Values and practices is that:

- Agile methods have been taken into use and the development is organized to Scrum teams executing through iterative sprints.
- All operations have been implemented as a system that is self-correcting and self-steering by implementing short feedback-loops. On the lowest team level these loops are very short, and the feedback almost immediate. On program level the loops may be a bit longer.
- Deliverables are made as complete during each short cycle as possible in order to drive for maximal predictability and quality.
- True visibility will be gained to pipeline loading and development capabilities.
- The organization has learned how to best adapt its operations based on reflection and continuous planning.
- Agile principles and values are integral part of all operations and culture.

These rules are based on the fact that change is constant and exact long-term estimation in this extremely complex environment is mission impossible. These rules tell how we can drive our development in value - and risk driven mode.

1.3 Scope

This policy applies all teams and all programs. It defines how agile principles and values are scaled in relevant processes. However, criteria in this current version will not intentionally cover all issues that are relevant when scaling agile principles and values. This policy will evolve over the time while we learn based on the implementation of this policy.

1.4 Ownership & approval

This policy is owned by the Agile Steering Group. Policy was approved by the Agile Steering Group on November 12, 2007.

1.5 Terms

Continuous Integration - Developers run all their unit tests, and integrate. Large programs often require a longer build and test cycle, so these programs use Automated Continuous Integration. A server process or daemon monitors the version control system for changes, then automatically runs the build process (i.e. build script) and then runs test scripts. The build script does not only compile binaries but also generates documentation, web pages, statistics and distribution media.

Definition of Done- Defines what is the maturity level of the code that team/IDO/program will release when a sprint is finished.

PMT- Program Management Team.

Program Backlog - Programs maintain and prioritize their work items in **Program Backlog** according to business value as user stories. Does not just contain requirements, but also other tasks the

program must take care of, e.g. testing, error correction, localization.

Program Content Backlog - Programs identify their mandatory and optional content and put them into **Program Content Backlog** in absolute order by business value. A program demands capacity for mandatory and optional content. On average, mandatory should represent 60-80 % of the whole scope.

Program Product Owner- Product Owner for a program. A program may have multiple product owners, forming a hierarchy.

Retrospective- Team Retrospective. At the end of every sprint a retrospective should be held where the team asks the question: "What worked?" and "What do we want to do different?" Based on these questions the team creates concrete actions which they will do during the next sprint. This way the used software process will continuously and forever be improved.

Program Retrospective - After each release and before next planning session scrum masters will hold a retrospective together with program or program managers for improving the software process used in that specific program.

Scrum Team- A small team of 5-9 people with cross-functional skills to do the actual work listed at Scrum Team Backlog.

Scrum Team Backlog- Backlog consisting of user stories which Scrum Team analyses and upon which the tasks at Sprint Backlogs are created. Scrum Team Backlog is created from Program Backlog by Team Product Owner.

Scrum Master- A person leading the Scrum Team. Scrum master ensures that the team works according to scrum rules, correct implementation and maximizing the benefits. Scrum master arranges sprint planning and review meetings, demos and retrospectives.

Team Product Owner- Product Owner in a scrum team. A person may act as product owner in multiple teams.

Sprint - A time period (usually 2 to 4 weeks) in which development occurs on a set of stories that the team has committed to.

Sprint Result- The functionality that scrum team was able to complete during the specific sprint.

Sprint Review- A time-boxed meeting at the end of each sprint at which the scrum team demonstrates to the team product owner and any other interested parties what it was able to accomplish during the sprint. Only completed product functionality can be demonstrated.

Sprint Backlog - Team level backlog contains tasks to be implemented in next Sprint (iteration). Shared and dedicated component scrum teams prioritize their Sprint Backlogs optimally for programs within committed allocations.

2. IMPLEMENTATION OF THIS POLICY

This policy is a collection of minimum set of requirements for the organization to implement agile-in-large. The implementation is the responsibility of program categories and each function's operational management.

Compliance with this policy is evaluated by going through the criteria in Chapter *3. Guiding Principles and Criteria for Large Scale Agile Deployment.* Organization is compliant with the issue stated by a criterion if it 1) has an approach to deal with the issue being mentioned and 2) has no significant problems or deficiencies originating from the issue.

3. GUIDING PRINCIPLES AND CHECK POINTS FOR LARGE SCALED AGILE DEPLOYMENT

3.1 All Development work organized to programs

All development work organized to programs (C1).

3.2 Scrum methods

All development (unless specific reason why scrum team set up is not effective) is done in scrum teams, delivering to one or multiple programs (C2).

- All development teams utilize iterative, incremental and agile methods and practices in their work. Recommended program management practice is based on scrum practices.

Each scrum team's work is organized via scrum team backlog (optional) and sprint backlogs (C3).

Every Scrum team measures their SW development & delivery capability (i.e., velocity) and plan sprints accordingly (C4).

Every agile scrum team has a defined and published definition-of-done that fulfills at least the definition of done criteria (for scrum teams) (C5).

Each scrum team performs testing according to criteria set in team's definition of done (C6).

- Definition of done is based on product owner's expectations of the quality of features or functionality.

- Teams continuously aim at better quality by adjusting the criteria of Definition of Done after each sprint.

- High quality is promoted by keeping the feedback cycle fast on scrum team level, and by following continuous integration practices.

Each scrum team has a retrospective meeting at the end of each sprint in order to continuously improve the process used by the team, and publishes the results on the intranet. (C7)

Every scrum team must follow the existing SCM policy and Codeline and Component Delivery Policy (C8).

Scrum Teams apply scrum methodology and follow-up these additional basic scrum rules:

- Have a team product owner to prioritize scrum team backlog.

- Have a scrum master to remove the impediments.

- Have a planning meeting at the beginning of each sprint which output is the public sprint backlog.

- Have a daily scrum meeting.

- Base their sprint planning on scrum team backlog.

- Base their (scrum team) backlog on program backlog(s).

- Every Scrum team reports its sprint and product burn-down charts, and estimates remaining work days left for product and/or features to respective program(s).

- Have a demo and sprint review after each sprint where team product owner approves or disapproves the sprint result.

3.4 Program management

All Programs shall have a prioritized program Content Backlog (C10).

- Program Content backlog is prioritized list of requirements.

- Requirements that are mission critical for the program, are explicitly separated from mission non-critical items.

- Program product owner is responsible for the Program Content Backlog prioritization.

All major program tasks are organized and prioritized by PMT according to Program Backlog (C11).

- The program backlog contains, besides requirement based user stories, also items for creating test cases, correcting errors like performing testing, preparing milestones, and creating language variants etc. – all major task items of a program.

- Program Backlog is created and updated by PMT in program planning meeting based on Program Content Backlog. In the planning meeting, it is recommended to ensure that all relevant stakeholders are presented, e.g. architect, UI specification and localization participation besides (team) scrum masters.

- Programs measure their velocity from Program Backlog, based on the progress done in a time-boxed increment from 2 weeks to 4 months.

- Programs report their progress as a burndown chart. This burndown chart is based on Program Backlog.

Programs have a review and demo after each release. Review and demo are either approved or disapproved by Program Product Owner. Possible corrective actions are injected and prioritized in program backlog and/or program content backlog.

Each program has a published definition of done defining the quality criteria (C12).

Programs must base their testing to criteria set in definition of done (C13).

- It is recommended that programs continuously aim at better quality by adjusting the criteria of Definition of Done after each sprint done.

Every program follows the latest codeline and component delivery policy (C14).

3.5 Scrum team self assessment

Each scrum team conducts a self-assessment against this policy periodically at least twice a year (C16).

- Teams are encouraged not to limit the self-assessment only to the criteria in this policy, but to include other items, relevant for team, in the scope of the assessment.

3.6 Program self-assessment

Each Program shall conduct a self-assessment against this policy periodically at least twice a year (C17).

- Programs are encouraged not to limit the self-assessment only to the criteria in this policy, but to include other items, relevant for program, in the scope of the assessment.

End of the policy.