ARCHITECTURE AND TECHNICAL DEBT AGILE PLANNING METHODOLOGY FOR SOFTWARE PRODUCTION

Aya Elgebeely and Amr Kamel

Department of Computer Science, Cairo University, Giza, Egypt.

ABSTRACT

This paper shows an empirical study for a new agile release planning methodology. The case study includes the application of the methodology by two teams in different software business domains (Game development and medical software development). The suggested methodology showed clear improvements in teams' productivity, enhanced software quality and better handling of the overall software architecture and technical debt. It allowed software teams to have more predictable release plan with fewer technical uncertainties. Results are showed in comparison with the traditional scrum release planning approach.

KEYWORDS

Agile, Technical Debt, Release Planning, Software Architecture, Software Engineering

1. INTRODUCTION

Accumulated technical debt and technical uncertainty are some of the most common complains of software teams. Business features always tend to get priority over the technical ones that have no clear direct business value. The absence of adequate technical planning tools is one of the factors contributing to this problem [1]. Technical debt according to Martin Fowler [2] happens because of some bad and quick design decisions that were made during the release of software. This debt incurs more development efforts in future to fix the bad code and smells injected in the code base. SEI [3] describes how architectural technical debt should be handled in this short paragraph: "A delicate balance is needed between the desire to release new software features rapidly to satisfy users and the desire to practice sound software engineering that reduces rework. The notion of technical debt creates a concrete way for software development teams to discuss the value and priority of system quality, maintainability, evolvability, and time-to market issues."

The method presented in this paper promises an enhancement in managing technical debt and uncertainties in software planning and implementation. However, it doesn't totally eliminate it, as there will always be unexpected changes throughout the course of the development cycle. The suggested release planning methodology can be considered as an extension to the well known Scrum release planning process. The traditional Scrum release planning focuses on clearly defining business features that should go into a specific version of software. All stakeholders meet together to define the requirements details , business priority and give a relative size using story points or any other estimation technique to each user story. However, discussing the technical implementation details early were always a missing dimension in the release planning meeting [4]. It was found through this empirical study that using only the 'risk' factor to consider uncertainties while planning a software release is insufficient. The large number of interdependencies between issues included in a specific software release is one of the major challenges in large-scale software development, which involves large number of contributors as well [5]. Previous studies [6] had shown that a balance is needed between adding new business value and eliminating technical debt to maintain stable software and reduce the increasing risk of technical debt accumulation.

Therefore, this method helps development teams to explore technical debt, architectural changes, and research and development efforts lying on the critical path of delivering the required business value. The suggested method helps development teams to visualize all the technical dependencies (through a dependency matrix) so that an implementation strategy is manifested to follow through sprints. The suggested release planning methodology considers both business priorities and the technical dimension as well. It also helps the development team to justify the time be spent in laying down technical infrastructure (that may not have direct business value) for other future releases to come.

2. RELATED WORK

Eltjo Poort introduced the iterative approach of addressing architectural changes and technical debt through his talk “Architecture Roadmapping” [7]. Wolfang Trumler and Frances Paulisch [8] coupled unit and integration tests with the low level details of requirements’ specification to avoid technical debt and enhance release predictability. Nelson Sekitoleko [9] found that improved knowledge sharing between technical and managerial staff on different levels can significantly improve the ability to create a good plan that considers interdependencies between features and addresses both business and technical challenges. Robert Phaal and Gerrit Muller [10] discussed the value of having what they called “Linked analysis grids” that is used in directly linking business requirements with technological aspects of these requirements, then use a scoring technique that ranks product’s backlog according to technical priority and complexity. Wojciech and Dorota [11] investigated the inherent risks in different agile methodologies, and concluded that a special risk mitigation process is mandatory. They found that these risks won’t be addressed by merely following agile best practices. This proves that additional investigation and analysis are required on all phases of the software development process for teams adopting agile methodology. Marjan [12] presented mathematical formalization of flexible release planning, using integer linear programming models and methods. They believe that using this ILP method in release planning will help in selecting the most appropriate features backlog based on revenues and costs, which takes into consideration both the business and technical perspectives. Jason Ho and Guenther Ruhe [13] introduced a method called Goal-Questions-Metrics (GQM) which helps in quantifying the technical debt coupled with a specific release in order to be able to give more accurate release dates.

3. RESEARCH METHODOLOGY

The methodology presented in this paper was applied via empirical experiments involved two development teams from different domains, but all of them were following the Scrum methodology. The goal was to include the real technical complexity in deciding on the features of a given product release. And the other goal was to reduce the technical uncertainties and waste elimination from the software development process.

One of the main activities that are done in a typical Scrum release planning meeting is the story points’ estimates of the user stories which is named as (sizing). The estimation is done by the technical team, with the presence of the product owner and Srcum Master. The technical team discusses the implementation details, risks and complexity. Story points have many usages during scrum iterations ‘sprints’ and future releases as well. Story points help the team know the relative size and development efforts needed for a user story against each other. It helps the team select their stories wisely and put a realistic time estimate for this story accordingly. It also helps in measuring and maintaining team's *velocity* over releases and iterations.

However, the technical teams' ability to analyze and predict the technical complexities of user stories and relative risks is affected by many factors. Some factors to mention are the teams' experience level and seniority, domain knowledge, project size and complexity, the business volatility and the novelty of development tools and frameworks.

In many occurrences, teams' judgment in the beginning of the project drastically changes afterwards, which yields to inaccurate story points sizing and time estimates during the later sprints. One of the major advantages of the new methodology presented in this paper, that it contributes in enhancing the design of the product and enables the team to address architecture concerns, technical debt and other dependencies continuously and early in the process. It also helps in envisioning the business impact and importance of such architectural changes to all stakeholders.

As shown in the related work, most of the methodologies revolved around going to the lower level of technical details and do more through technical analysis in the planning process. And they found that just following the basic release planning technique is not enough to mitigate the technical risks and in correctly judging the release backlog from the business side due to their unawareness with inherent implementation complexities and risks.

The suggested methodology is similar to the traditional release planning by beginning with the product owner and stakeholders explaining the new features of prospect products. Yet, a series of technical analysis meetings are done before the sizing event takes place. These technical analysis meetings should be correctly facilitated by the scrum master or the facilitator the team chooses, to avoid falling into analysis paralysis trap, ensure reliable outcome and that everyone in the team has contributed.

3.1. Release Planning Stage 1 (Pair Analysis)

The technical analysis starts after the business perspective (new set of requirements) being shared with the team. Then the team is allowed a time box (1-2 days) to think solely or in pairs about the features suggested by the business side. Inside this time box every single team member is allowed the opportunity to reflect, search and write down all her technical concerns and questions.

3.2. Release Planning Stage 2 (Swarm Analysis)

The next step is that the whole team is gathered again in an assembly point, where the scrum master explains the meeting process and expected outcomes. Then the team is re-distributed into sub teams (3-5 members) each. Sub teams should be as diverse as possible. For instance, a senior developer, a junior developer, a tester and architect will form a perfect sub team. Each sub team should have the business backlog with them, and they would exchange together the concerns and questions they had collected in stage 1. Each sub team should compile a list of technical considerations and thoughts they agreed upon about each feature in the backlog. The time allowed for sub teams discussions varies from (1 to 3 hours) according to the size of the backlog and sub teams. However, it's suggested that sub teams' size do not exceed 5 team members to reduce conflicts and communication channels opened within the sub teams' discussion. Sub teams should nominate a representative who will talk on behalf of them when teams are gathered again in analysis stage 3.

A typical scrum team will have about 3 to 6 sub teams. In this stage, the facilitation role of Scrum Master is mostly needed. She should be rotating over sub teams to make sure they are all progressing with their discussions, not blocked by minor conflicts, or over analyzing a feature. She should also make sure that every team member is actively sharing in the discussion, and no one is taking a passive attitude. Also, she will be the time keeper for sub teams, by reminding them of how much time is remaining for them to maintain their momentum.

3.3. Release Planning Stage 3 (Analysis Consolidation)

In this stage, all the technical teams are gathered again to share their collected concerns about the new backlog. The team goes through the backlog, and for each user story, every team representative shares the sub-team's concern about it if they had any. The concerns and thoughts being shared are categorized into one of five categories, (New feature request – technical debt – Architectural change – Risk – R&D "Research and Development"). A board is created for each category and issues are written on sticky notes with different color representing the five areas. This helps the team keep track of issues being shared without repetition and help them maintain the big picture physically.

3.4. Release Planning Stage 4 (Technical Analysis Matrix)

In discussing the value of the Technical analysis, we will use a quotation from Stojanovic & Dahanayake [14] "By its definition a component hides a complexity of its interior so that components can help in easier making an architecture metaphor and architecture prototypes (spikes) as the main design-level artifacts. Components as a mechanism for organizing business requirements in cohesive business service providers, and at the same time a blueprint of future implementation, can provide bi-directional traceability between business needs and software artifacts. That certainly helps in better understanding and communication across the project, and more straightforward iterations and increments. Good business-driven, component-oriented architecture design can reduce the need for refactoring (that can be also time -consuming task), as well as permanent customer presence, in the case it is not feasible."

3.4.1. Example of Technical Analysis Matrix

This is a sample view of the technical analysis matrix and the dependencies domains included inside it with sample data.

Table 1. Example of technical analysis matrix.

Feature	**Dependent Features**	**Confidence Level**	**Risk (1-5)**	**Bugs ID**	**Pending UI/UX**	**Technical Debt**	**Architecture Change**
A	(B,E)	Medium	3	546	Progress bar	Thread-Safe implementation	Threading module, Job Queues
B		High	3				
C	(A)	Medium	2	342	New CSS, log In page	Refactoring of Login pages	Security Module
D		Low	4	321, 542			
E		High	1				

3.5. Release Planning Stage 5 (Technical Intuition)

At this stage, each new business feature is linked with its related decencies in the dependency matrix. The team discusses the related dependencies and accordingly set the confidence level of implementing this feature. The confidence level is a qualitative metric (High – Medium – Low); It depicts the overall confidence of the team of implementing that feature, after the dependencies and details of the features had been discussed [15].

Although it's a qualitative metric, yet different pervious experiences mentioned that this qualitative approach of simply asking the team their confidence level was much better approach than only looking to the reports and resource allocation to determine the confidence level.

3.6. Release Planning Stage 6 (Requirements Sorting)

After all the technical details and dependencies have been discussed the team meets again with the business side to share the relative risks and complexities associated with the release suggested features. Hence, the business side and product owner have a chance to reprioritize the release backlog after they got technical insights. This is very suitable for projects with high technical complexity or niche technologies. At this stage the team is capable of sorting the backlog with respect to business priority and technical complexity as well. Spikes, technical debt, architecture can now added early in the sprints as the team as well as non-technical partners knows the business impact of these technical tasks.

4. RESULTS DISCUSSION

An empirical experiment was made with two teams in different domains, first team composed of 12 developer and 2 testers. The development team has 3 of them dedicated to work on R&D tasks, while the others shared in the overall development process. The suggested methodology helped in reducing the overall release duration by 4 times less than the time used to deliver similar release backlog with the traditional release planning approach. It also allowed the team to address critical architecture concerns that allowed them to enhance different areas in the design including testability, logging, modularization and optimization. The number of production bugs reported decreased from an average of 10 bugs reported after the software release to 2 bugs per release after production given a release backlog of relatively similar complexity.

Table 2. Data comparison for the first team.

Factors	**Release 12 with traditional release planning**	**Release 13 with technical perspective workshop**
Developers	13	11
Average years of experience	2.5 yrs	1.5 yrs
Release Backlog size (features/story points)	230 story point	210 story point
Changes during release	Massive	Minimal
User stories	Shallow	Detailed
Testing team	3	2
Release time challenges	4 months after due date	3 weeks after due date
No. of Blocker and critical bugs after release	17	4

The second team composed of 4 developers, 1 tester and 1 designer. The time to release the release under test was enhanced by 6 times faster than the traditional working scheme they used. The team tended to tackle the technical debt in an ad-hoc manner, once they started a new feature that includes a technical debt, they work on it regardless its risk, cost or added value. There were a lot of cross dependencies between team members that caused idle time for the designer, tester or other developers. By following the suggested approach, dependencies were clear from the release start date, the team was focused; tasks were distributed in a lean way that minimized cross dependencies between team members. And it drastically reduced the delay caused by having any of the team members waiting for someone else in order to be able to start progressing in a given task.

Table 3. Data comparison for the second team.

Factors	**Version 1 with traditional release planning**	**Version with technical perspective workshop**
Developers	5	4
Average years of experience	2 yrs	3 yrs
Release Backlog size (features/story points)	100 story point	120 story point
Changes during release	Massive	Minimal
User stories	Detailed	Detailed
Testing team	0	1
Release time challenges	2 months	No challenges
No. of Blocker and critical bugs after release	21 and a failed client site deployment	7 and a smooth deployment

5. CONCLUSION

This paper presented a methodology that increases the confidence of estimates and works on eliminating many wastes presented inside a software development process of a given team. These enhancements are basically linked with the inclusion of a deep technical analysis with the release planning process. The application of the methodology was described and the role of every stakeholder during the release planning event was discussed. Performance indicators measured during the development and after production showed substantial enhancement in team's performance and software quality.

The key for reducing technical uncertainty is doing enough upfront analysis prior starting a software release. Technical debt, new architecture changes and software development research can be addressed by following the above-mentioned release planning method that protects the whole development team from taking wrong decisions and underestimation of technical complexities.

REFERENCES

[1] H. C. Benestad and J. E. Hannay, "Does the prioritization technique affect stakeholders' selection of essential software product features?," Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement , pp. 261-270 , 2012.

[2] M. Fowler, "Technical Debt," October 2003. [Online]. Available: https://martinfowler.com/bliki/TechnicalDebt.html.

[3] "Architectural Technical Debt," [Online]. Available: http://www.sei.cmu.edu/architecture/research/arch_tech_debt/.

[4] K. Schwaber, "SCRUM Development Process," in Business Object Design and Implementation , London, 1997.

[5] C. R. B. de Souza, D. Redmiles and G. Mark, "Management of interdependencies in collaborative software development," in 2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings., Rome, Italy, 2003.

[6] K. Power, "Understanding the impact of technical debt on the capacity and velocity of teams and organizations: Viewing team and organization capacity as a portfolio of real options," in 4th International Workshop on Managing Technical Debt (MTD), San Francisco, CA, USA, 2013.

[7] E. R. Poort, "Agile Architecture Roadmapping," in SATURN 2016, San Diego, CA USA, 2016.

[8] W. Trumler and F. Paulisch, "How "Specification by Example" and Test-Driven Development Help to Avoid Technial Debt," in IEEE 8th International Workshop on Managing Technical Debt (MTD) , Raleigh, NC, USA , 2016.

[9] N. Sekitoleko, F. Evbota, E. Knauss, A. Sandberg, M. Chaudron and H. H. Olsson, "Technical Dependency Challenges in Large-Scale Agile Software Development," in International Conference on Agile Software Development, Rome, Italy, 2014.

[10] R. Phaal and G. Muller, "An architectural framework for roadmapping: Towards visual strategy," Technological Forecasting and Social Change, pp. 39-49, 2009.

[11] W. Walczak and D. Kuchta, "Risks characteristic to Agile project management methodologies and responses to them," Operations Research and Decisions, pp. 75-95, 2013.

[12] M. d. Akker, S. Brinkkemper, G. Diepen and J. Versendaal, "Software product release planning through optimization and what-if analysis," Information and Software Technology, pp. 101-111, 2008.

[13] J. Ho and G. Ruhe, "When-to-release decisions in consideration of technical debt," in Managing Technical Debt (MTD), 6th International Workshop, Victoria, BC, Canada , 2014.

[14] Z. Stojanovic, A. Dahanayake and H. G. Sol, "Modeling and Architectural Design in Agile Development Methodologies," in Evaluation and Modeling Methods for Systems Analysis and Development, Velden, Austria, 2003.

[15] P. A. Beavers, "Managing a Large Agile Software Engineering Organization," in Agile Conference, Washington, DC, USA, 2007.

[16] P. A. Beavers, "Managing a Large "Agile" Software Engineering Organization," IEEE Computer Society , 2007.

AUTHORS

Aya R Elgebeely is a masters student in computer science department in Cairo University, who has over 9 years experience in software development and leading development teams. She worked in IBM Egypt for 5 years and currently working as a technical consultant with many tech startups in Egypt as software delivery manager to boost software team's performance.