2021

Three Levels of Agile Planning in a Software Vendor Environment

Ramesh Lal
Auckland University of Technology, ramesh.lal@aut.ac.nz

Tony Clear
Auckland University of Technology, tony.clear@aut.ac.nz

Three Levels of Agile Planning in a Software Vendor Environment

Full research paper

Ramesh Lal
Dept. of Computer Science and Software Engineering
Auckland University of Technology, Auckland, New Zealand
Email: Ramesh.lal@aut.ac.nz

Tony Clear
Dept. of Computer Science and Software Engineering
University of Technology Auckland, New Zealand
Email: Tony.clear@aut.ac.nz

Abstract

There is a misconception that agile development requires minimal planning effort. In reality, an agile approach for market-driven software development requires highly disciplined, reliable, and accurate planning practices to swiftly plan and develop high-value innovations in a software vendor environment. This study investigated a highly successful international software vendor in Melbourne, Australia, to provide a case study on agile planning practices. Five planning practices were identified which underly successful agile software development for software vendors. These planning practices were driven by agile concepts such as adaptation, self-organizing, cross-functional collaboration, and empowerment/delegation. We constructed a conceptual framework for agile planning practices, the APP (Agile Planning Practices) Framework illustrating the three levels of agile planning.

Keywords Agility, market-driven software development, agile planning practices, software vendors

1. Introduction

The agile approach for software development is based on having a barely sufficient process (Xiaocheng, Paige, et al. 2010). However, it can be interpreted as requiring minimal planning effort in projects. Agile's goal to deliver business value by software is achieved through a solid planning effort. Software vendors need to have reliable planning practices to create well-thought-out plans. They ought to build agility into their planning approach to develop software for the market-driven environment.

Product and project planning are two critical activities, one done at, what is termed here, the business level and the other at the project level (Lal, 2011). Product planning provides a solid foundation to make first time-right decisions on the development of new software or features. Hence, the project plan must be reliable and provide certainty for successful implementation.

While the visibility of product planning is vital for project planning, it also requires fluid boundaries for a collaborative effort between the business and software engineering units. However, agile methods have been limited to project-level planning only. There is a lack of understanding of agile method fragments integrating product and project planning in a software vendor environment. Hence, a planning framework for market-driven software development is investigated.

The rest of this paper is organized as follows: Section 2 identifies key literature on agile planning practices and constructs for the research. Section 3 describes the research methodology. Section 4 provides the case study, while Section 5 discusses the findings and identifies a conceptual framework for agile planning. Section 6 provides the conclusion.

2. Literature Review

2.1 Agile software development concepts for projects

Several important and established agile concepts are presented here. The first is that projects are implemented using *short development cycles* (Vanhanen, Itkonen, et al. 2003). Hence, the capability to produce reliable plans is required throughout the projects. Another vital conception of the agile approach is *self-organizing* work effort in projects (Moe, Dingsoyr, et al., 2008). This requires adopting mutually accepted planning practices for an organization-wide contribution. It also highlights the next essential agile view, a *cross-functional effort* (Schwaber & Beedle, 2002). Hence, fluid boundaries with functional units and roles are critical for a collective planning effort.

Empowerment and delegation are other critical agile development concepts (Boehm and Turner 2003). Individuals in development teams have the responsibility for producing plans for their development work. Another essential concept is the flexibility to adapt development practices (Conboy and Fitzgerald 2010). Hence, the adaptive mindset for creativity and improvisation is vital for a rapid response in a market-driven development environment (Highsmith, 2002).

2.2 Planning practices for agile software development

In this section, the key planning practices for developing software products and projects are identified.

2.2.1 Vision planning

A vision plan (business level) plan has information on target users, a few key features (high-level requirements), benefits and reasons for clients to buy them, how it differs from other competing products, and how it will provide the competitive advantage at the marketplace (Vähäniitty & Rautiainen, 2008). While vision planning provides the link between business decisions and requirements engineering, it is also used as a tool for communicating ideas to the stakeholders (Lehtola et al., 2005). This planning practice is not part of any agile method, although advocated as requirements engineering practice by Highsmith (Highsmith, 2009, p.92) but adopted by software vendors with their agile approach (including our case study organization) (Lal & Clear, 2018).

2.2.2 Roadmap planning

The roadmap planning, another business-level planning practice, identifies features (high-level requirements) that will be in different market releases (Bagnall, Rayward-Smith, & Whitley, 2001). This plan identifies the number and dates of each release for a period with a list of high-level requirements, with their rough implementation estimates (Wilby, 2009). This planning practice has not been part of any agile method (until more recent frameworks such as "ADAPT" by Vallon et al.,

(2016) and in some scaled agile frameworks) but adopted by software vendors with their agile approach (Lal & Clear, 2017, 2018).

2.2.3 Backlog planning

This project-level planning practice establishes tasks or user stories (low-level requirements) from the high-level requirements to create a product backlog (Lehto & Rautiainen, 2009). Hence, project backlogs enable product goals to be met (Raatikainen, Rautiainen, Myllärniemi, & Männistö, 2008). Project backlogs have prioritized tasks or user stories with estimates ready for implementation in short development cycles (Schwaber & Beedle, 2002).

Backlog planning is time-boxed (Rosenberg, Stephens, et al. 2005). It also provides the opportunity to discover emerging requirements (Highsmith, 2009). Therefore, a product backlog is not a static list of tasks (Rising & Janoff, 2000). Features can be removed, or priorities changed in the backlog by the product managers (Sutherland, 2005). Tasks in product backlogs are in their smallest form (Kalliney, 2009). They are referred to as tasks with XP and user stories with Scrum methods, functional and non-functional requirements with the DSDM method, and features with ASD and FDD methods (Highsmith, 2002). The software engineers' involvement in backlog planning enables them to understand features better (Dinakar, 2009)

2.2.4 Short development cycle planning

This project-level planning practice requires in-depth preparation on specific user stories or tasks to be implemented in the following short development cycle (sprint or iteration) (Liu, Erdogmus, & Maurer, 2005). This plan provides a subset of estimated and analyzed tasks from the release backlog (Klein & Canditt, 2008). This planning happens a few days before the next short cycle (Kinoshita, 2008). This plan is often reliable and realistic (Engum, Racheva, & Daneva, 2009).

The sprint/iteration planning meeting is between the backlog owner and the entire development team members (Abrahamsson, Salo, Ronkainen, & Warsta, 2002). During the planning meeting, epic tasks are identified, split, and estimated into smaller tasks, and others are re-negotiated for their estimates (Frank & Hartel, 2009). This re-negotiation is vital for teams to commit to delivery by the end of the sprint or iteration (Cohen & Thias, 2009)).

2.2.5 Daily planning meeting (stand-up meeting)

The next significant project-level planning practice is the daily planning meeting, lasting no more than fifteen minutes and requiring each team member to answer three questions; what they did since the last meeting, what obstacles they had, and what they will do before the next meeting (Schwaber & Beedle, 2002). It helps teams organize for the day where the members decide which tasks, they will implement from the set of tasks they have committed to deliver in the current cycle (Rubart & Freykamp, 2009). This planning practice also encourages team members to communicate more outside the meetings to help solve identified issues (Paasivaara, Durasiewicz, & Lassenius, 2009). The daily stand-up meetings help keep solving problems and deliver results (Shaye, 2008).

2.3 Constructs for the agile planning framework

The following were the constructs to investigate the planning framework (as identified above through the literature review): (1*). the high-level or business level planning for the product,* including the vision and roadmap planning, and (2). *the low-level or project level planning* includes release or backlog planning, short cycle (sprint or iteration) planning, and daily (stand-up meeting) planning. The agile concepts such as *cross-functional effort, empowerment/delegation, self-organizing, short development cycles, and adaptation* impacting the planning practices (constructs) were also part of the investigation. These agile concepts enable organizations to achieve competitive advantage through development agility (Highsmith, 2002; Hitt et al., 1997, Lal & Clear, 2018). Hence, for this research, the constructs provide the basis for data analysis of agile planning in a software vendor environment to create a planning framework.

3. Research Method

To provide an understanding of method fragments integrating product and project planning in a software vendor environment, creating an agile planning framework involved data gathered through a longitudinal empirical investigation (since 2006) with a software vendor based in Melbourne. This case organization (name anonymized) had adopted a hybrid agile approach in 2003 based on the agile manifesto with practices adopted from FDD, XP, and Scrum methods, including the UCD (user-centered design) approach. Currently, their agile approach is driven by a scaled agile DAD (Disciplined Agile Delivery) framework. They have had an ongoing adaptation of their organizational structures,

method, and practices, including global development teams based in 9 countries. This study reports on the situation in the case organization up until 2017.

Data was collected through semi-structured interviews conducted at the Melbourne site by the first author of this paper. In total 62 interviews were done up to December 2019: 2006 to 2009- 40 interviews; 2015- 1 interview; 2017- 8 interviews; and 2019- 13 interviews focused on their scaled agile method. In 2006 a week was spent observing and understanding their application of agile practices. These practices were further investigated with interviews. The individuals for the source of information were the Director of Software Engineering, Software Engineers, Product Manager, Project Managers, Team Leader, Quality Assurance Manager, Quality Assurance Engineers, and Technical Communication Engineer. Interviews were conducted with a mixture of engineers selected by their job titles, including graduate, post-graduate (engineer title), senior, and principal level engineers. Each interview session was planned for an hour and held in meeting rooms at their production lab. The participant had anonymity, and they had the right not to answer any uncomfortable questions. Data was coded into the various planning practice categories under two levels of planning identified in a literature review (Lal, 2011). They were reduced to codes to use for quotes to identify and describe the agile planning practices.

4. Case - Agile Planning Practices

Meldevelopment, founded in 1990, has offices across North America, Europe, India, South Africa, and Australia. The following describes its planning practices for developing software that sells as on-premises software and is offered as Software as Service (SaaS) through cloud.

4.1 High - (business) level planning

At Meldevelopment, there was a significant change in the product planning process with agile adoption in 2003. This change established a team approach involving their engineering and sales/marketing departments to develop vision and roadmap plans. This model was based upon face-to-face interaction, collaboration, and working together as a group. Their high-level planning involved creating vision and roadmap plans. These two artefacts show their initiative to produce a strategic product identifying a prioritized list of highly innovative features as high-level requirements (vision plan), including a set of short and long-term release goals of these features (roadmap plan).

> The product manager works with engineering, sales, and marketing [to] propose vision plan … roadmap, list features, how big they are, the release they are in … dates you would deliver things … the vision is what the roadmap is trying to achieve with new features in the market (P1)

Here, it is the responsibility of their product manager to compile, coordinate and communicate the vision plan. The product manager presented this plan to the product planning team (management level stakeholders) for implementation approval. Based on the vision plan, the product planning team collectively formulated a roadmap plan for strategic market releases for a period of up to eighteen months. Software engineers were not directly involved in the final decision making but had the Director of Software Engineering representing them for product planning.

> Spend two- or three days doing product planning, … looking at the market and the roadmap … bouncing some things around, feel what is realistic, what we can do as a complete team … have consensus and agreement … have a roadmap for the next 12 to 18 months of projects. (P1)
> This aspect of our work is not agile … developers aren't involved. (P2)

However, the implementation priority of the features and release plans at Meldevelopment were subject to change due to changes in the marketplace.

> Always have changes … requirements [high level] change, the needs of the business change. (P7)

In 2003, the product manager (part of their sales/marketing group) adopted practices to elicit and set priorities to high-level requirements through a series of meetings with various stakeholders, including the engineering group. With their previous RUP (Rational Unified Process) approach, software engineers faced severe challenges to elicit low-level requirements, often causing delays in implementation. With this change, the product manager became part of their design phase in projects, helping engineers swiftly produce low-level requirements from high-level ones. Hence, the product manager provided engineers and project managers with information to understand the high-level requirements.

> Product manager, responsible for defining the requirements … works with the marketing & sales, consultants, documentation, QA, and support team … see customer issues … does all the trade-offs and priorities. (P1)
> As an engineer … so my communication can be straight to the product manager. (P3)

The product manager role has adapted to have a reliable technical background to determine features for implementation from many ideas and concepts. Engineers had experienced requests coming in that

were technically not possible. The product manager role was required to effectively learn and communicate the proposed features with user organizations, understand specific market needs, and share the limitations of what could be achieved, including software engineering.

> Our product is technical … current product manager drives a lot of our projects … seeing what the market wants … puts together [what] we would be able to do … does culling. (P1)
> Must have a technical understanding …would not need to down to a code level … simulate a user [role] looking at the system … the technical knowledge should be more of the customer domain. (P2)

Meldevelopment enhanced their vision planning with prototyping to get feedback on proposed innovations. It was challenging to communicate the vision using documents only for highly technical products. The vision planning practice was adapted to include prototypes to transform the high-level requirements into something that looks like a feature that individuals can interact with to understand the proposed features better. At the engineering level, prototypes created an awareness of the likely technical hurdles allowing the architects to think of potential architectural problems.

> Articulate the vision … prototype to make sure everybody understands … any technical hurdles, start to resolve … senior management can look at and understand what we are trying to communicate. (P1)
> We must get input from all stakeholders … [the product manager] will work with us early on to get a prototype. (P4)

4.2 Low-(project) level planning

At Meldevelopment, the design phase through backlog planning elicited low-level requirements (user stories & tasks), for implementation in a project. The project manager drove the design phase and was responsible for planning and implementing the backlog. It was one of the most critical phases for development teams as it enabled them to get a deep understanding of implementation tasks. For the quality assurance team, the design phase provided insights into test cases, and for the documentation team, it helped them plan the write-up of the supporting materials. The product backlog let the marketing team know when features could be shown to potential customers or included in their marketing campaigns.

> Get high-level requirements from product planning … project manager calls meetings to flesh things out. (P5).
> Project managers ensure that implementing is based on the priority, we're in session constantly … project managers engage with the field … our sales teams go directly to the project managers. (P6)

Their design phase had a team approach involving the development, quality assurance, and documentation teams, helping swiftly to create a product backlog. Their complex high-level requirements required different approaches to break them into implementation tasks. The software engineers, project manager, and product manager collectively decided the process to take to elicit low-level requirements. If they agreed to accept the test-driven approach, then QA engineers provided guidance based on how they would test the feature. If the project was interface-driven, then the documentation team gave guidance to elicit low-level requirements.

> Have QA and documentation team sitting with us during the design session. Project manager, product manager, all to get the product vision … get a rough sort of skeleton on what the system is to look like … the whole team is involved. (P2)
> Have a brainstorm on different ways of testing it [QA driven] … decide the best way to understand requirements. (P1)

How the team captured the low-level requirements ha usually been adapted. They applied use cases to capture features, but written descriptions were used to capture more technical features.

> A lot of end-user interactions employ use case analysis … when more algorithmic, drive through stated. (P2)

But using use cases did not enable them to identify their products' end-users different roles and objectives. They also experienced an increase in defects reported by the customers and support team. Analysis of the defects identified that most of the issues related to integrating their product with other systems at client sites, requiring Meldevelopment to provide helpful features. Hence, they collectively agreed to adapt their design phase with the user-centered design (UCD) approach. The UCD approach enabled them to put user experiences at the forefront by learning from the actual or proxy users how they would use the product and fit it into their work environment. The UCD approach involved inviting individuals who could provide the best insights into the likely users and creating prototypes to get feedback on the features and usability requirements.

> Building with an engineer's mentality … not thinking the company had administrators and managers, with different needs. Defects reported by customers, the user-centered design would have solved the problem. (P1)
> UCD starts with a face-to-face meeting with experts … define personas … brainstorming, whiteboard sketching, mock-up prototypes … send that around, get feedback and have a few iterations. (P7)

The design phase was further improved by incorporating prototyping to understand the features they were developing through a time-boxed approach. This helped to avoid overspending their time on developing prototypes. Senior engineers identified prototypes as an effective tool to get better feedback on features and architecture.

> Put a bit of software in front of someone … people just understand so much more clearly … some prototypes are for the UI, throw them away … some are architectural prototypes, become evolutionary, turns into a real product … just a quick way to mitigate risk, time-box it. (P7)

4.3 Development- (short development cycle) level planning

Meldevelopment projects were between one to four months in duration. They had three major releases annually and delivered 4 to 5 patches, including a feature pack before a major release. Incremental and iterative practices drove their development. Thus, all their projects were feature-driven with short development cycles, known as iterations. The iterations enabled them to package features that were implemented up until the major release date. Their projects had two-week iteration cycles. The implemented features from an iteration cycle (iteration release or IR) were delivered to the QA team for various quality assurance checks and then to the documentation team to produce manuals and online help materials. The QA and documentation teams had a two-week iteration cycle within which they carried out their tasks and got any bugs or issues fixed by the engineers.

> Series of projects emerge at the same time … 3 major ones… minor releases and patches, some feature packs. (P1)
> Features are driven by sales opportunities or customer demand … using an iterative approach … better result. (P6)
> 2-week iteration … 2 weeks development, then test for 2 weeks… fixed any bugs found … got a feature completed. (P7)

At Meldevelopment, an iteration plan identifies the schedule and individual ownership of the tasks to be implemented in the next iteration. Iterations start after the design phase once a prioritized product backlog has been established for the project. The priorities were assigned by the product manager, which was market-driven. Iteration planning meetings lasted an hour and took place a few days before starting an iteration cycle. The iteration plan was an outcome based on agreement by development engineers on the individual ownership of tasks and their estimates with their project managers. Their QA function, documentation function, and marketing and sales teams were dependent upon the iteration builds. The visibility of the iteration plan was extremely important, dependent upon the reliability of the task estimates provided in iteration plans.

> Established a feature list [product backlog] …work out rough estimates … put together an iteration plan … engineers come up with their estimates … work out the schedule. (P5)
> It can be the project manager … it will be the engineers [putting iteration schedule] … the developers to be. (P2)

At Meldevelopment, there were three points for estimation. The first was with the product planning when vision and roadmap plans were developed, estimating on a large scale for the next 12 months for feature releases. This was a high-level estimate, providing the likely implementation duration of a feature, and at this stage, for estimation, only a few senior software engineers were involved. The next point for estimation was the design phase, which determined the number of iterations needed to implement a feature. Their design phase required software engineers to split and estimate high-level features into individual independent tasks. Then at iteration planning, the engineers evaluated and re-estimated tasks against which they will work. This inbuilt refinement and re-estimating practice enabled the team to have an in-depth understanding of design decisions and implementation strategies.

> A small number of people [engineers] involved in the high-level [estimation]… doing design, we must commit… need every person … knowing that they can deliver against it. (P5)
> If you had estimated, will also do the development … moved away from that [so] anyone can work on it. (P2)

Despite having three different points for estimation, it has been a constant challenge for their development teams to deliver against the schedule. On occasions, to deliver on a specific date required crowding out some of the tasks from the product backlog. Other times, iterations may have had more than the accepted bugs due to insufficient time appropriately spent implementing tasks. These were primarily due to estimation errors. They now have adapted their agile estimation practice, which involved not revealing estimates until all engineers had their estimates ready to reveal to the team, enabling more discussion to understand design decisions and implementation strategies. The previous practice was based on someone suggesting an estimate and everyone agreeing to it. Later, they also adapted their agile estimation approach to capture, track and evaluate iteration performance.

> Estimating in the group, not showing until they are all done … reveal at the same time … not influenced by another person … prompts discussion … record what we do, how our estimates compare … how many bugs we end up with. (P1)
> Working on the estimations … one of our weakest areas … most developers tend to take on more than they can handle. (P2)

At Meldevelopment, the daily stand-up planning meeting required providing status updates on the implementation tasks. Their meetings had an informal structure, not being time-boxed, and not held each day simultaneously. Later, the engineers experienced not much value when they provided task updates as they were frequently aware of one another's progress when co-located. They adapted the stand-up meetings to have quick discussions around the implementation issues the engineers faced.

However, engineers realized that they did not achieve much value in having daily stand-up meetings since engineers discussed problems in their teams as soon as they encounter them. Since then, the stand-up meeting has been held on an as-needed basis. Some projects have daily stand-ups, while others have adapted to weekly stand-ups, depending on the project complexities.

> Done it informally in teams, without having a set time ... everyone says what they've been working on ... didn't seem to be effective ... then tried, what issues do we have ... have a quick discussion, how to solve it ... at the moment, sometimes we don't depend on how well the project is tracking. (P1)
> We did for a time had dailies ... know what everyone was up to every day ... so we stopped it after a while. (P2)

5. Discussion

Lessons learned on planning practices from the case study are discussed below. A Conceptual Framework for Agile Planning Practices is also provided based on our findings.

Our findings support the literature on planning practices such as *vision, roadmap, backlog, short development cycle,* and *daily planning*. These planning practices were part of the product development lifecycle involving two functional units and three planning levels in a software vendor environment. First, *high-level or product planning* driven by the business function included *vision and roadmap plans*. Second, low-level planning driven by the *software engineering function* consisted of *project* and *development* (short cycle) *plans*. These planning practices were adopted to develop and enhance software products that also help to deliver a minimum viable product (MVP). Product planning helped software vendors decide what to build, including their strategic market release dates, while project planning targeted software with appropriate user experience functionality (Yang, 2016). Development plans were created to produce and monitor the implementation results of a project plan, whether planned for two weekly, monthly, or even shorter development cycles.

We identified that agile concepts such as *short development cycles, self-organizing, cross-functional effort, empowerment, delegation, and adaptation* drive planning practices for product development. It is unclear what influence these concepts have, particularly at the business level, with other software vendors. Adaptation of product and project manager roles is critical since these roles have responsibility for planning. Adopting the software engineering practice for high-level requirements to enable vision and roadmap planning in a software vendor environment shows the mindset for short development cycles at the business level. Hence, product planning requires capturing high-level requirements as specific individual features, by product managers with the technical knowledge, and understanding to do that effectively. These findings are consistent with that of the increasing need for dynamic business-IT alignment in digitally driven businesses (Horlach et al., 2020).

The cross-functional collaboration for business-level planning (product planning team), including supporting engineering level planning (product manager supporting backlog planning), further suggests an agile mindset at the business level. A limitation of agile methods is not having practices beyond the project level. Software vendors will significantly benefit if clearly defined agile planning practices and roles support product planning activities. Scaled-agile methods counter this limitation by identifying portfolio and product management aligning with software engineering (program and project management) (Lal & Clear, 2018).

Two vital elements in a software vendor environment are the product manager role and the product planning team. The product manager's role is to interface between the business and software engineering to propose, develop and deliver features. This requires effectively working with the industry sources, clients, business units, and engineering department to compile and submit product innovations (high-level requirements via vision and roadmap plans) that are economically and technically feasible. The product planning team brings responsibility and accountability for making decisions on product innovations. A cross-functional membership (including software engineering representation) ensures well-thought-out and 'first-time-right' decisions on high-level requirements for development and market release dates. It democratizes the decision-making on product innovations, which are more responsive to the market needs, and development capacity and capability. While the product planning team makes the implementation decisions, the product manager is responsible for allocating and implementing high-level requirements through projects.

Low-level planning helps deliver high-level requirements as product features. Agile methods serve well for product development, providing product backlog and short development cycle (sprint/iteration) planning practices for undertaking projects. However, a software vendor environment also requires adopting appropriate structures, such as a design team, based on cross-functional membership to ensure instant product, client, and end-user environment-related information to support the product backlog planning practice. In addition, an upfront design phase in projects is essential to break down the high-level requirements into low-level ones (i.e., user stories, provide estimates, and set priorities),

creating product backlogs for projects. The design team should consist of product and project managers, stakeholders with vital product knowledge, software engineers, quality assurance engineers, and technical (documentation) writers. Participation and empowerment for engineers and technical writers are critical for product backlog planning to ensure collective responsibility to deliver the backlog items for projects.

The product backlog planning practice requires the adaptation of skills to enhance roles with product and project management skills. Project managers need to be both business- and project-focused. Hence, they have been adapting to work as product development managers with end-user and product knowledge, including as proxy to product managers. Therefore, project managers must act as 'on-site customers' to implement high-level requirements and be responsible for project delivery. As proxies to product managers, project managers may co-locate with engineers in projects representing product managers who can't work full-time in projects. Software engineers require extensive design and planning abilities, while quality assurance and documentation engineers need a collaborative mindset to provide test and interface insights to support the product backlog planning practice.

The development-level planning practices, like the product backlog planning practice, are also essential for undertaking projects in a software vendor environment. A critical aspect of agile methods is that product backlog implementations are done in short development cycles. Therefore, development level planning is separate from the product backlog planning and repeated throughout the project for each short development cycle to further plan and implement the product backlog in blocks.

The short development cycle (sprint or iteration) planning is a crucial development-level planning practice for product development in a software vendor environment. This practice allows software engineers to re-plan, i.e., re-estimate and re-evaluate the user stories allocated for implementation in the upcoming short development cycle. The on-site customer (project manager) assigns specific user stories from the product backlog based on their implementation priority. Notably, the short development cycle planning practice helps to clarify if the user stories are well-thought-out for implementation. The project team must do the re-planning with the on-site customer (project manager) to enable a reliable plan. This empowerment for re-planning also delegates a collective responsibility to the team for the timely delivery of features. Essential skills now required by project managers and software engineers for this short development cycle planning practice are negotiation, empathy, and trust with testing knowledge and understanding.

The daily (stand-up meeting) planning, which is the next development-level planning practice, may not be required every day to plan and inform the implementation status of user stories by individual engineers while working on a current short development cycle in the project. Organizing projects for small-sized teams, co-location in a common work area, and working together enables the engineers to constantly discuss one another's progress and raise issues as implementation progresses in a short development cycle. Hence, this planning practice is adopted on an as-needed basis in projects.

High-level planning enables software vendors to learn and make well-thought-out decisions on product development. Low-level planning helps provide visibility in projects. Product backlog and short development cycle plans provide certainty, clarity, and reliability including confidence for implementation since working software regularly emerges in vendor product environments. Echoing the work of Vlaanderen et al., (2011), we see the need for continuous refinement of requirements in product management and the iterative process of converging on product goals.

Figure 1 (page 9) captures the five different planning practices relevant to a software vendor environment based on the high-level (business) and low-level (engineering) planning driven by an agile approach. While agile methods provide planning practices for software engineering, business-level planning must adopt an agile mindset in a software vendor environment.

6. Conclusion

We established that product development by software vendors driven by basic agile methods requires business and engineering level planning based on cross-functional collaboration to identify and build business value software. As a result, five agile planning practices have been identified; *vision, roadmap, product backlog, short development cycle (sprint/iteration), and daily (stand up meeting) planning*. The agile mindset for short development cycles, cross-functional effort, adaptation, empowerment, and self-organizing must be adopted by vendors since 'basic agile methods' (Scrum, XP, etc.) provide planning practices limited to the engineering level.

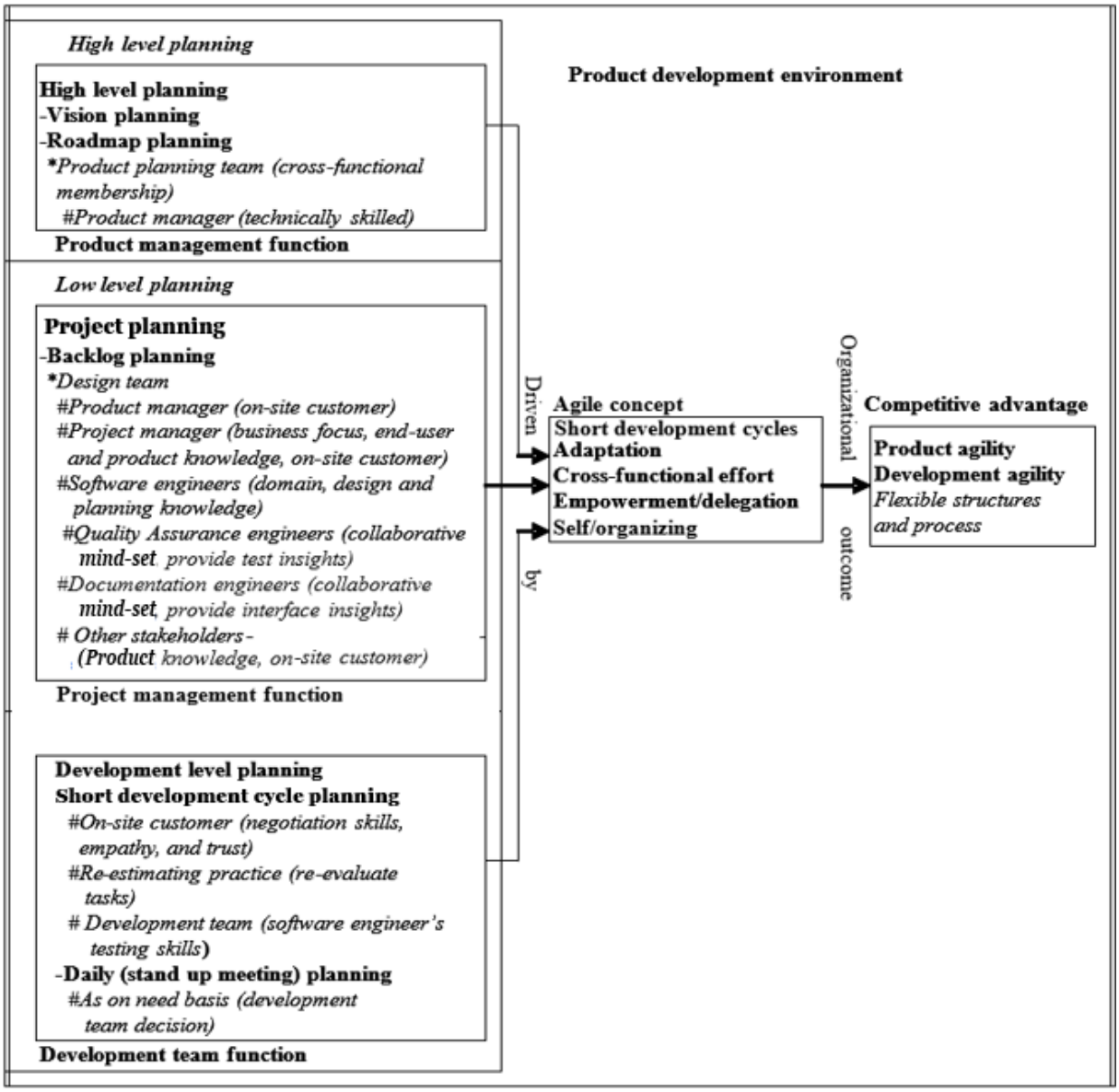

Figure 1 APP (agile planning practices) Framework

We developed an empirically grounded conceptual framework for agile planning practices, the *APP (agile planning practices) Framework* (Fig. 1 above), based on our findings. This Framework demonstrates that planning is vital and critical in a software vendor environment for product development. It also shows that an agile approach for software development is plan-driven, and no development happens without planning. Planning goes through three levels and provides certainty for achievable implementation. We also believe that the APP (*agile planning practices*) Framework provides a basis for further research to enhance understanding of planning practices based on 'basic agile methods' adopted by other software vendors. We intend to use the APP Framework to investigate further planning practices with the 'scaled agile' framework subsequently adopted by our case organization in 2017. The three levels of agile planning and their relevant planning practices integrate a product development environment for a team effort and provide an organizational capability for planning high-value product ideas. The APP (*agile planning practices*) Framework shows that planning for business value through software requires dynamic organizational structures (planning teams and practices, roles, and skills), adaptable on a needed basis to enable product development agility.

7. Limitations and Future Research

This research is based on a single, albeit longitudinal, case study, limiting the ability to generalize the findings to a broader software development community of in-house software development and other software vendors. However, our case organization is a mature agile development practitioner, and a highly successful global software vendor, with that agility permeating the organization, so it may

provide a blueprint for others. We believe our research which uses interview data from our longitudinal study (2006 till 2017) of a software vendor provides a sound understanding of planning practices driven by 'basic agile' methods. A future investigation into agile planning practices may consider replicating this study with another software vendor organization, including large-scale software production driven by a 'scaled agile' framework. The results will provide valuable insights for software vendors and potential in-house development teams into the required changes to be successful with agile planning at the product, project, and development levels. Another study into agile planning involving in-house development teams may be helpful to compare the agile planning approach with this study and provide insights into the specific changes to adapt to be successful with agile planning.

8. References

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile software development methods: review and analysis. www.vtt.fi/inf/pdf/publications/2002/P478.pdf.

Bagnall, A. J., Rayward-Smith, V. J., & Whittley, I. M. (2001). "The next release problem". Information and Software Technology, 43(14), 883-890.

Beck, K. (1999). Extreme programming explained: Embrace change: Reading, Mass., Addison-Wesley.

Boehm, B. and R. Turner (2003). Observations on balancing discipline and agility. IEEE Computer Society (pp. 32 - 39).

Cohen, B., & Thias, M. (2009). The failure of the off-shore experiment: a case for co-located agile teams. IEEE Computer Society (pp. 251 - 256).

Conboy, K. and B. Fitzgerald (2010). "Method and developer characteristics for effective agile method tailoring: A study of XP expert opinion", *ACM Transactions on Software Engineering and Methodology 20(1)*: 1-30.

Dinakar, K. (2009). Agile development: overcoming a short-term focus in implementing best practices. http://doi.acm.org/10.1145/1639950.1639952. (pp. 579-588).

Engum, E. A., Racheva, Z., & Daneva, M. (2009). Sprint planning with a digital aid tool: lessons learnt. IEEE Computer Society (pp. 259-262).

Frank, A., & Hartel, C. (2009). Feature teams collaboratively building products from ready to done. IEEE Computer Society (pp. 320-325).

Highsmith, J. (2002). Agile software development ecosystem. Boston: Addison-Wesley.

Highsmith, J. (2009). Agile project management: creating innovative products: Pearson education.

Hitt, M., Ireland, D., & Hosikisson, R. (1997). Strategic management: competitiveness and globalization, concepts, and cases (2nd ED.): West Publishing Company.

Horlach, B., Drews, P., Drechsler, A., Schirmer, I., & Böhmann, T. (2020). Reconceptualising business-IT alignment for enabling organizational agility. In *Twenty-Eighth European Conference on Information Systems (ECIS2020) – A Virtual AIS Conference*.

Kalliney, M. (2009). Transitioning from agile development to enterprise product management agility. Paper presented at the agile conference, 2008. IEEE Computer Society (pp. 209 - 213).

Kinoshita, F. (2008). Practices of an agile team. Paper presented at the agile conference, 2008. IEEE Computer Society (pp. 373-377).

Klein, H., & Canditt, S. (2008). Using opinion polls to help measure business impact in agile development. In proceedings of the 1st international workshop on business impact of process improvements (pp. 25-32). Leipzig, Germany. ACM.

Lal, R., & Clear, T. (2017). Scaling Agile at the Program Level in an Australian Software Vendor Environment: A Case Study. In *Australasian Conference on Information Systems* (pp. 1-12). Retrieved from https://aisel.aisnet.org/acis2017/81/

Lal, R., & Clear, T. (2018). Enhancing product and service capability through scaling agility in a global software vendor environment In R. Espinoza & D. Šmite (Eds.), *Proceedings 2018 IEEE 13th International Conference on Global Software Engineering* (pp. 59-68). Los Alamitos, California: IEEE. https://doi.org/10.1145/3196369/3196378

Lal, R. 2011. "Strategic factors in Agile Software Development method Adaptation: A Study of Market-Driven Organisation", http://mro.massey.ac.nz/handle/10179/2496.

Lehto, I., & Rautiainen, K. (2009). Software development governance challenges of a middle-sized company in agile transition. http://dx.doi.org/10.1109/SDG.2009.5071335. (pp. 36-39):

Lehtola, L., Kauppinen, M., & Kujala, S. (2005). Linking the business view to requirements engineering: long-term product planning by roadmapping. IEEE Computer Society (pp. 439-443).

Lindgren, M., Wall, A., Land, R., & Norstrom, C. (2008). A method for balancing short- and long-term investments: Quality vs. features. IEEE Computer Society (pp. 175-182).

Liu, L., Erdogmus, H., & Maurer, F. (2005). An environment for collaborative iteration planning. Paper presented at the agile conference, 2005. IEEE Computer Society (pp. 80-89).

Moe, N. B., T. Dingsoyr, et al. (2008). Understanding Self-Organizing Teams in Agile Software Development. IEEE Computer Society (pp.76 - 85.).

Paasivaara, M., Durasiewicz, S., & Lassenius, C. (2009). Using scrum in distributed agile development: a multiple case study. IEEE Computer Society pp. (195-204).

Raatikainen, M., Rautiainen, K., Myllärniemi, V., & Männistö, T. (2008). Integrating product family modeling with development management in agile methods. http://doi.acm.org/10.1145/1370720.1370728. (pp. 17-20).

Rising, L., & Janoff, N. S. (2000). The scrum software development process for small teams. Software, IEEE, 17(4), 26-32.

Rosenberg, D., Stephens, M., & Collins-Cope, M. (2005). Agile development with ICONIX process. Berkeley, CA: Apress.

Rubart, J., & Freykamp, F. (2009). Supporting daily scrum meetings with change structure. http://doi.acm.org/10.1145/1557914.1557927. (pp. 57-62).

Saliu, O., & Ruhe, G. (2005). Supporting software release planning decisions for evolving systems. IEEE Computer Society (pp. 14-26).

Scott, A. (2017). Defining MVP, MBI, MMF, and MMR. Retrieved from https://www.projectmanagement.com/blog/blogPostingView.cfm?blogPostingID=61937&thisPageURL=/blog-post/61937/Defining-MVP--MBI--MMF--and-MMR# =

Shaye, S. D. (2008). Transitioning a team to agile test methods. IEEE Computer Society (pp. 470-477).

Schwaber, K., & Beedle, M. (2002). Agile software development with scrum. Upper Saddle River, N.J: Prentice Hall.

Stapleton, J. (1997). Dynamic systems development method: the method in practice. London: Addison-Wesley.

Sutherland, J. (2005). Future of scrum: parallel pipelining of sprints in complex projects. IEEE Computer Society (pp. 90-99).

Vähäniitty, J., & Rautiainen, K. T. (2008). Towards a conceptual framework and tool support for linking long-term product and business planning with agile software development. http://doi.acm.org/10.1145/1370720.1370730. (pp. 25-28).

Vallon, R., Strobl, S., Bernhart, M., Prikladnicki, R., & Grechenig, T. (2016). ADAPT: A Framework for Agile Distributed Software Development. IEEE Software, 33(6), 106-111.

Vanhanen, J., J. Itkonen, et al. (2003). Improving the interface between business and product development using agile practices and the cycles of control framework. IEEE Computer Society (pp. 71 - 80).

Vlaanderen, K., Jansen, S., Brinkkemper, S., & Jaspers, E. (2011). The agile requirements refinery: applying SCRUM principles to software product management. *Information and Software Technology, 53*(1), 58-70.

Wilby, D. (2009). Roadmap transformation: from obstacle to catalyst. IEEE Computer Society (pp. 229 - 234).

Xiaocheng, G., R. F. Paige, et al. (2010). An Iterative Approach for Development of Safety-Critical Software and Safety Arguments. IEEE Computer Society (pp. 35 - 43).

Yang, R. (2016). The Product Plan vs. the Project Plan. Retrieved from https://www.aha.io/blog/the-product-plan-vs-the-project-plan.