Understanding the Relations Between Iterative Cycles in Software Engineering

Henri Terho, Sampo Suonsyrjä, Kari Systä and Tommi Mikkonen
Tampere University of Technology, Tampere, Finland
{henri.terho, sampo.suonsyrja, kari.systa, tommi.mikkonen}@tut.fi

Abstract

Iterations are one of the most successful mechanisms in software development to ensure that the resulting system is satisfactory. Due to its strengths, various kinds of iterations have been integrated to software development with varying goals. In this paper, we consider different types of iterations related to software development, including prototyping, incremental development, sprints as in e.g. Scrum, and iterations as defined in *Lean Startup. The goal is to understand the relations between the types of iterations, and to find out what kind of similarities and differences they have with each other. As a result, we find that while the goals are different, it is possible for the iterations to coexist, so that one form of iteration is used as a tool to complete the goals of another.*

1. Introduction

While often considered as a modern approach compared to plan-driven, waterfall-style approaches, iterative development has a long history – the application of iterative and incremental development dates as far back as the mid-1950s [1]. While no single iterative approach was dominant and numerous differences between methods existed, they all shared the view to avoid a single-pass, sequential, document-driven, gated-step process [1].

Different iterative methods and techniques for different phases of software development have been proposed by the software engineering community. For example, prototyping [2], Scrum [3] and, more recently, Lean Startup [4] share an iterative way of working. However, these techniques have born from different viewpoints, and they are to be used at different stages and for different purposes in the development process. For instance, while sprints are used to manage weekly tasks [3], Lean Startup is used to test initial product viability [4].

Since the term iteration is used in so many contexts and meanings, ranging from a minimum viable product that can be used to test business hypothesis to full-blown new versions of software products, it is not surprising that the overlapping use of methods can cause confusion in the process. The situation is further complicated by the fact that numerous stakeholders, with different terminology but partly the same terms, often participate in software development activities in different roles, such as customer, domain specialist, project and product manager and developer, to name some common ones.

The communication problems between the stakeholders of the software development process are a major issue in software development. The different goals of different stakeholders can result in conflicts between priorities even though all are in their own opinion speaking the same language. These problems are exacerbated in large organizations, where communication between stakeholders is already a larger issue in its own. If the knowledge of the different types of iterations and their targets, attributes, and stakeholders would be improved, the strengths of all the cycles could be better utilized. This in turn would lead to more integral working between projects and organizations, and creating common tools and vocabulary to the whole development team.

Some authors claim that in the end, the cycles culminate in running code that is continuously maintained [5], but we assume a wider view. We claim that iterations also serve other purposes, and that iterations proposed by different approaches are inter-related but not the same. We believe that when understood properly, these different cycles could actually result in better overall view of the product development and communication between the different stakeholders in software development. This better view can be utilized to optimize the usage of resources, understand feedback better and make better decisions on the development track of the project as a whole, resulting in higher quality products.

URI: http://hdl.handle.net/10125/41874
ISBN: 978-0-9981331-0-2

In this paper we analyze similarities, differences, and other relations between the different forms of iterations used in software development. The paper has been inspired by earlier work regarding how software startups handled product development [6]. Extending this work to cover the different types of iterations instead of simply product strategy introduces more possibilities to apply the results in practice.

The rest of this paper is structured as follows. In Section 2 we introduce the iterations in the selected approaches together with a brief comparison of their characteristics, goals, and motivations. In Section 3, we shift the focus to the various targets of software development cycles. In Section 4, we discuss the stakeholders of software development, and in Section 5 we address the attributes of the various cycles. In Section 6, we provide a synthesis of the results. In Section 7 we draw some final conclusions.

2. Background and Related work

The researchers and practitioners of software engineering have introduced several ways to iterate in software development. These different types of iterations are used in different context but have several similarities. Managing these iterations takes work and specific attention, as well as balancing between time to market [7].

As also discussed by Berente and Lyytinen, iteration is actually a multi-dimensional issue where different levels of iteration always happens in a software project, be it cognitive or guided by the process [8]. In this paper, however, we focus more on the different forms of iterations as methods and on their various characteristics as such. More precisely, we analyze similarities, differences and other relations between the different forms of iteration in four different setups. The analyzed iteration types are the following:

- **Prototyping**. Prototypes enable a high degree of user evaluation and initiates a learning process for the end users and developers of the system [2].
- **Incremental development**. The features of the software are grouped so that the most important features are implemented first, and the subsequent iterations complement the software [9].
- **Sprint**. Popularized by Scrum [3], sprints contain time-boxed sets of features selected for implementation.
- **Lean Startup**. Popularized by Eric Ries, Lean Startup is an iterative development method for creating products that users actually want and are ready to pay for [4].

As the starting point of our study, we next briefly review the different iteration types together with the drivers behind these approaches.

2.1. Prototyping

Software development approaches that are based on prototyping have been developed for situations where the work steps of a project cannot be clearly detailed before execution [10]. Prototyping incorporates many styles, including iterative, rapid, evolutionary, throwaway incremental, and mock up prototyping [11]. Stephen and Bates [2] define the prototype through two common characteristics:

1. The prototype enables a high degree of user evaluation, which then substantially affects requirements, specifications, or design.
2. The prototype initiates a learning process for users and developers of the system.

Hierarchically, prototypes can be divided into throwaway and evolutionary prototypes. Throwaway prototypes are discarded after their initial use, but evolutionary prototypes are used as a basis for further development. Thus, development based on evolutionary prototypes goes through sequences of re-design, re-implementation and re-evaluation without knowing the complete set of requirements beforehand [11]. Although the exact requirements for further development might be unclear, the implementation choice still matters as large parts of the code will be reused. In contrast, the intended further use of throwaway prototypes is clear from the beginning – its code will not be used.

2.2. Incremental Development

The *Guide to the Software Engineering Body of Knowledge* defines incremental development as *"An incremental model produces successive increments of working, deliverable software based on partitioning of the software requirements to be implemented in each of the increments. The software requirements may be rigorously controlled, as in a linear model, or there may be some flexibility in revising the software requirements as the software product evolves*." [12].

While incremental development is often considered a somewhat modern technique, Craig Larman and Victor Basili argue that its application dates as far back as the mid-1950s [1]. In incremental development, completed increments are deployed and taken into use. A particular feature of incremental development is that all increments are planned according to the needs of the users and the development gets feedback from the real usage for designing and deploying later increments. Still, a plan over multiple increments to come is commonly made, so that each increment can be used to drive the design towards future requirements. In terms of concrete realizations, RUP introduces four distinct project life cycle phases:

- **Inception**: Scope the system so that there is a valid baseline initial budgeting.
- **Elaboration**: Mitigate the key risks; execute problem domain analysis; define baseline architecture.
- **Construction**: Build the system.
- **Transition**: Take the system to production.

While it is possible to advance in iterative cycles within each phase, the above phases, when repeated, form the incremental development cycle as defined by RUP. Consequently, each cycle is planned almost to the extent as a one-off product would be planned, thus resembling the waterfall model. However, it is common that some of the features are pushed to subsequent releases, in particular if they do not contain near-term value for end users.

2.3. Sprints in Agile Methods

Two core values in the Agile manifesto are "Customer collaboration over contract negotiation" and "Responding to change over following a plan" [13]. For obvious reasons, these values conflict with rigorous control and up-front definition of the requirements that are often associated with development methods where a longer-term view is used, or, in general, with any precise interpretation of a pre-made project plan.

Many concrete incarnations of agile methods, for instance XP and Scrum, include time-boxed sprints where technical activities take place. The primary purpose of these sprints is coordination and management of work. Furthermore, Scrum proposes the production of potentially deliverable products in all sprints, where the content of each sprint is usually defined according to development and customer collaboration aspects – not based on the incremental needs of a user.

2.4. Lean Startup

Lean Startup is a methodology for building enterprises, not software [4,13]. The methodology has been crafted in software context and it shares the idea of frequent iterations with many software engineering methods. In a nutshell, building a successful product for a software startup consists of multiple short iterations each of which surveys systematically the context of the conceptualized product.

The iteration in Lean Startup starts with an idea that includes hypotheses about the customer behavior or the context of usage. When the first hypotheses have been validated, the first *minimum viable product* (MVP) can be built. This product is a version that enables a full turn of the build measure learn loop with minimum effort. For each loop the main goal is to learn if the business and product hypothesis is valid – in other words, whether the product is actually something that someone needs or wants and can it create a scalable business.

Based on the above, the goal of the process is to evaluate the business validity of the proposition. However, technological development is required in most cases to do such evaluation, in particular in the context of software development [15].

3. The Changing Targets of Software Development Cycles

Many challenges in software development have led to different kinds of iterations. Firstly, there are many unknowns related to technologies, requirements and business. This means that iterations are needed to *manage risks and learn from feedback*. For complex systems, these challenges exist even if the context of the project is stable. However, the context – customer needs, technologies, and so on. – usually change. Thus iterative approaches are used to *effectively respond to changes*. The fact that the challenges vary in their nature means that we must use iterations for several purposes. In the following, we formulate the types of problems that match the types of iterations addressed in this paper. In addition, the role of iterations within the larger scope of development is addressed.

3.1. Prototyping

The goal of prototyping is probably the most straightforward, when considering the different types of iterations – build a prototype simply to figure out what is doable and what is not. Prototyping is often needed to get started with something new, be it implementation technique or domain. In addition to testing or trying a technology, prototyping is used to communicate ideas to stakeholders.

The value of a prototype is not primarily in the developed software or its use. The value is in the learnings and communication. The developer organization learns from the issues in development, benchmarks and stakeholder feedback. In addition, the prototype helps in communicating the idea or product.

Prototyping methods have a long-standing tradition also in the field of human-computer-interaction (HCI)

[16]. In general, prototypes range from high to low fidelity, i.e. from low-cost methods such as paper sketches to more detailed propositions like interactive web applications. Low-fidelity methods have proven to be highly efficient in validating designs and predicting large problems. On the other hand, high-fidelity methods have been used for example for assuring management and other stakeholders.

While often considered as small experiments, prototypes of considerable size also exist. For instance, the Cloudberry project [17] – obviously beyond a simple, single-developer experiment of a particular detail – can be regarded as a prototype for demonstrating the feasibility of web technologies in the development of a mobile device. In fact, although seldom mentioned explicitly, many totally new software systems can be traced back to prototypes created to test technology, which, when deemed mature and applicable, are eventually refined to products. Clearly, organizing such complex prototypes needs different kinds of iterations to help the development.

3.2. Incremental Development

Almost any computing system we are accustomed to is a result of several evolutionary steps. These steps, reflecting the understanding of user needs at a particular moment, as well as development capabilities available at the time, are used to create a product in such a way that changing technology during the development can be integrated into the process to create simpler, better results which are easier to maintain and develop further.

While often associated with new features introduced in each iteration, it is sometimes in the eye of the beholder how much iterations have in common. For instance, while one can consider the different Microsoft Windows versions as increments, it is questionable to what extent the different iterations share their code base. Thus, incremental development can be considered from various angles, one angle considers the technical origins, and others focus on the development organization and end users of the system. While the last angle is often overlooked, keeping customer happy with new and improved features is an important part of incremental development – indeed, if no new versions emerge, the users may think that the development has ended and there is no maintenance left, encouraging them to start using another, competing system.

3.3. Sprints

Sprints as understood in Scrum [18] most likely have the most concrete, unambiguous definition of any cycle in software engineering. Simply put, sprints are time-boxed, repeated cycles during which software development takes place. Each cycle contains a number of events, such as Daily Scrum and retrospective, which help in execution and coordination of the work, as well as enable improving the ways of working. Thus, sprints can first and foremost be considered as a way to organize software development, and to associate the work with fixed starting and ending points. What happens during the sprint is up to the Scrum team that can independently decide how to meet the targets of the sprint. Since the focus and commitment is on one sprint at a time, the team can respond to change only in the next sprint. However, since the sprints are usually short, between 2-4 weeks, it is usually enough to shift the focus to next tasks only in the next sprint.

Based on the above, sprints can be regarded as a project management mechanism for the development. Advancing in increments enables frequent evaluations as well as forces the developers to verify and validate the system each time a sprint terminates, making it a solid starting point for the development.

3.4. Lean Startup

In Lean Startup, iterations consist of three phases – build, measure, and learn as illustrated in Figure 1. Each phase plays a role in gathering justifiable evidence if profitable, scalable user needs exist – and what is a feasible business model or a product to fit to the model. The goals of the phases are presented in the following:

- **Build**: Create the simplest possible version of the system that fulfills the intended mission of the system, based on hypothesis of the users need.
- **Measure**: Collect data from the use of the system, preferably so that it gives statistically significant evidence that either validates or rejects the hypothesis.

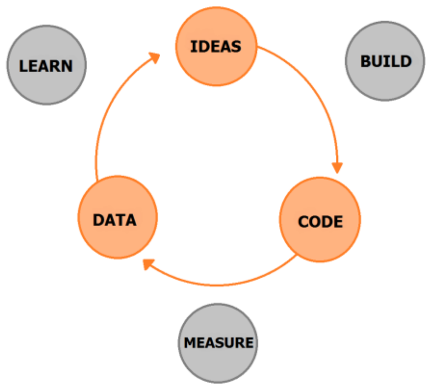

Figure 1 Build Measure Learn Cycle

- **Learn**: Based on measurements, determine whether or not the mission was accomplished in accordance to the hypothesis. If the mission was not accomplished, redefine the hypothesis and initiate a new build measure learn cycle.

It is important to notice that while software may be built as a part of executing the Lean Startup process, its goal is to validate a business hypothesis, not to be a full-fledged product. Hence, the notion of Minimum Viable Product (MVP) is used to denote a version of the system that includes enough elements to judge its business potential, but which by no means is a complete product.

3.5. Summary

The targets of the different cycles are presented in Table 1 and briefly summarized in the following.

For prototyping, the main focus lies in turning ideas, thoughts, and intuitive designs into something concrete. The target is communication: either to get feedback or to communicate the idea to external stakeholders. This is achieved by turning ideas, thoughts, and intuitive designs into something concrete. Although the produced solutions can be small and cover only one perspective, prototyping is a great way to take the first steps towards the final product.

The main target of Lean Startup's build measure learn loop is to learn by creating something concrete and validating the learning with a specified audience. In contrast to prototyping, the context of learning is business driven although metrics such as amount of new users can be seen software driven as well.

However, both incremental development and sprints emphasize the software and its production. In the incremental development new version are delivered to users one after another and in extreme cases the software development is seen as a continuous flow of new software versions. With such premise, the software team can take advantage of new emerging technologies that become available during the software development. On the other hand, the team can also respond to the changing user needs faster and easier than with more traditional methods.

Although sprints might guide the software teams into the same kind of benefits as incremental development, one of their core targets is to freeze at least some parts of the user needs and requirements. In this sense, sprints help the teams in execution and coordination of the work by providing time-boxed

Table 1 Targets and Attributes of the Cycles

Cycle	Targets	Attributes
Prototyping	• Figuring out what is technically doable • Validating designs and predicting large problems • Communication, assuring management and other stakeholders	• Cycle length: From hours to months • Team size: From one developer to a team of developers • Termination condition: Full stop once a technological solution is proven to be feasible.
Incremental development	• Provide value to the customers already during the project. • Taking advantage of new technology • Assuring the stakeholders that the development is continuous and on-going	• Cycle length: Any given time that is needed to get a new increment done • Team size: Software team (and the related stakeholders) • Termination condition: When the new software asset / increment is considered done.
Sprints	• Responding to emerging user needs • Helping in execution and coordination of the work • Improving the ways of working • Guiding to frequent evaluations of new parts of the system	• Cycle length: Evenly one to four weeks • Team size: Software team • Termination condition: Calendar deadline
Lean Startup	• Gathering justifiable evidence if profitable, scalable user needs exist. • Evaluating if a hypothesized business model is feasible to satisfy the user needs. • Learning by creating MVPs.	• Cycle length: From days to weeks • Team size: From a single developer to a whole software team. • Termination condition: Once the learning goal can be validated with statistically significant results.

segments with clear targets.

4. Role of Stakeholders

Almost all software development projects involve various stakeholders. At least the following roles are commonly identified:

- **Individual developers** that participate in the development in different roles, like designer, programmer, and tester. Together, they form the development team, which can sometimes be considered as a separate stakeholder as well.
- **End-users** are the individuals and organizations that eventually use the designed software system. Most commonly the developers and the end-users have different backgrounds and therefore have a different view to the system.
- **Customers** represent the organization that make the investment decision, provide the requirements and decide if the software system is to be taken to use. The relation between end-users and customers is often overlapping – you first buy a system, and then you use it – but at times the roles are distinct.
- **Sponsors** are investors that help development team to start their work, when a paying customer is still to be found or if the current revenue stream does not yet cover the development costs.
- **Software organization** provides support for the developers. For instance, they may provide support for product management, marketing, sales, and number of other things that fall beyond the actual development. Obviously, each specialized actor inside an organization can be considered as yet another stakeholder, but for the purposes of this paper, they can all be treated similarly.

Stakeholders of the different iterative software development cycles are described in the following subsections and summarized in Table 2.

4.1. Prototyping

Prototyping involves several stakeholders. Prototypes may be used to collect feedback from any of the above stakeholders. End-users and customers can give feedback on usability and feature set of the developed product. Sponsors and organization can give feedback about profitability and other business aspect.

In addition, prototypes are used to communicate the content of the designs and to gain commitment from any of the stakeholders. Based on the information the stakeholders can plan their own activities and increase their interest and trust in developed software and the development team.

4.2. Incremental Development

In incremental development a software organization repeats its development activities one

Table 2 Stakeholders of the Cycles Summarized

Cycle	Developers	End-users	Customers	Sponsors	Organization
Prototyping	Learn about the tested topic	Get early information about the forthcoming software	Get early information about the forthcoming software	Get confirmation about the progress	Get early information for supporting actions
Incremental development	Can concentrate on manageable set of tasks. Reduced risk with early feedback.	Early value: can start using SW and features earlier. Can give feedback.	Early value: can start using SW and features earlier. Get information of the progress. Can give feedback.	Get reliable information of the progress.	Get early revenue. Get reliable information of progress.
Sprints	Can Concentrate on manageable set of tasks. Reduced risk through early feedback.	Can give early feedback at the end of each sprint.	Can give early feedback at the end of each sprint.	Can give early feedback at the end of each sprint.	Can give early feedback at the end of each sprint. Ability to change direction due to changed business situation.
Lean Startup	Get fast feedback to minimize waste	Early value: can start using SW and features earlier. Ability to give feedback.	Get early information about the forthcoming software	Get fast feedback on the business potential.	Get fast feedback on the business potential.

round after another. In the case of RUP, these activities include inception, elaboration, construction, and transition that are further decomposed to smaller increments. Also, different kinds of variants can be derived for company-specific use. Feedback from end users, including also usage data collection, as well as marketing and sales can be taken into account as a part of the development, and in general the approach is comprehensive in the sense that it involves almost any possible stakeholder of the software, including developers and testers, organizational support functions, as well as end users and customers.

The overwhelming range of interest groups makes it sometimes difficult to determine all the consequences the introduction of a new version produces. Obviously, phasing of the project means that the set of involved stakeholders is not the same in each phase. Furthermore, since the different phases in themselves include several activities – such as alpha and beta testing – defining the precise set of stakeholders for the life-cycle is next to impossible as every stakeholder is somehow involved at some point.

4.3. Sprints

Sprints are executed by software teams, so software developers and testers are obvious stakeholders. However, any outside communication with the team takes place via a product owner, who acts as a proxy for all other stakeholders. Therefore, the number of stakeholders in the middle of sprints remains low. However, after each sprint, feedback from stakeholders is requested. Preferably an executable version of the system is then demonstrated to other stakeholders, such as product managers, customers, and end users to gather feedback and foster mutual commitment to the development. In these demonstrations stakeholders both learn about the developed software but also have possibilities to give feedback.

4.4. Lean Startup

As long as the decided end-result of the build measure learn cycle is a software artifact, individual developers are obviously entwined in the loop. However, the software organization is likely the most influential of the stakeholders, because the bottom line target of an MVP is commercial. Consequently, the *software* from the defined *software organization* term above can many times be eased out, because it is not uncommon that the organization for example subcontracts the software development of their MVP.

Although the software organization might be the one calling the last shots when building an MVP, the influence of potential customers and investors cannot be emphasized enough. As the main idea in the development of an MVP is to get feedback from other stakeholders, refining it towards something that customers want intrinsically requires their input to the subject. Additionally, or in some cases even with the heaviest focus, MVPs can be developed to assure and engage investors.

5. Attributes of Software Development Cycles

To understand the software development cycles and their nature more deeply, we select three dimensions that are continuously present with them. These descriptive dimensions are cycle length. work effort or team size per cycle, and a termination condition for a cycle or how is each cycle validated

5.1. Prototyping

Prototyping can have the shortest length of the development cycles, if the low-fidelity paper sketches are considered – such can be completed with a minimal work effort and team size of only one developer or designer. However, be it paper sketching or technological try-outs, the work efforts of prototyping usually stop at once when the required result is reached. In this sense, the amount of work effort and time can be difficult to define in advance.

On the other hand, prototyping can involve much more of the development organization than just one developer. In these situations, the devoted time and work efforts typically require far more careful planning, i.e. risk management by the organization. This, again, can have an impact on the required result of the prototyping cycle as well, because the decision whether the result is sufficient enough is not for only one person to make. For example, a paper sketch or an experimental design can be done by only one designer. In contrast, when a whole organization is devoted to prototyping whether a technological solution is feasible, opinions on termination conditions are bound to raise debate, thus requiring careful planning.

5.2. Incremental Development

Incremental development relies on well-planned, established process, where each of the phases in the life cycle form a solid basis for the subsequent phase. For instance, only after inception it is possible to start to elaborate the project into an implementation form, and only an elaborated enough project can really result in an implementation. Due to such planning, the life cycle of a RUP project can take considerably long time to run – up to years for each iterations in the case of complex products such as telecommunications systems. Due to the extended period of the life cycle, also the development effort can be considerable, up to 1000 man-years in the case of large systems.

Since each iteration in incremental development produces a real system, the outcome for each release includes almost any possible feedback one can imagine. These include technical data such as code quality measurements, test and bug reports as well as business data such as user evaluations, sales reports, and market research studies. The overwhelming amount of feedback can at times be so extensive that it is difficult to utilize all of in the design and planning of the next version of the system.

Since the time it takes to execute a full project life-cycle may be so long, it is not uncommon that the personnel changes in the course of the project. This in turn calls for a procedure to involve new persons in the project in a planned, controlled fashion.

5.3. Sprints

One of the most important constants in sprints is the stability of the development team, followed closely in importance by the fact that the sprints are always of the same length and executed to the end. The fact that the team works together for extended periods of time results in the ability to create realistic time and work estimates for problems at hand, forming the key enabler to meet the time-boxed deadlines.

5.4. Lean Startup

Although a wide range of artifacts from paper sketches to fund raiser campaigns could be seen as MVPs, we scope this paper to include only MVPs with some sort of technological solutions. Even with this limitation, however, the time and work efforts required in each build measure learn loop can vary quite significantly. On one hand, a landing page describing a product idea and a built-in analytics solution can be made in a matter of hours. On the other hand, a detailed user interface that allows customers to act the same way as is intended with the actual product (but for example the real business logic is still done manually), can take weeks only to build.

The decisive point for the length of the cycle in these situations is the wanted end-result. With the landing page example, the organization has to wait in the measure phase as long as the quantitative data, such as the page visits is statistically significant. With the second example, however, the organization can have a very short measure phase and gather qualitative data from a few specific customers sufficiently to advance to the learn phase.

5.5. Summary

Of the described iterative development cycles, prototyping has he most variable cycles ranging from hours with paper sketching to months spent with more difficult technological evaluations. The team size can vary as well, but once the prototype is evaluated as sufficient, the development with the same learning objective comes to an end. Similarly, cycle times vary in the build measure learn loops with Lean Startup. They are dependent on the set learning goals and therefore on the MVPs under construction. The development time of different MVPs obviously varies case by case, but roughly the times range from days to weeks. Obviously, the different types of MVPs need different amounts of staff to work on them, but typically this amount ranges from a single developer to a software team. If the learning goals are clearly set, the termination condition of a build measure learn loop is clear as well – once the learning is validated.

In contrast to the varying cycle times described above, sprints have a fixed time period, which is usually something between one to four weeks. Incremental development is somewhat similar to this, as it also has fixed goal with which the cycle terminates. However, the needed time depends so heavily on the work effort, that the cycle time varies dramatically from minutes to months or even years. The same obviously applies with the needed work effort and team size.

6. Synthesis

Table 3 presents a summary of the different types of iterations. When considering the focus of the described iterative cycles, prototyping and Lean Startup share a similarity in creating a method for experimentation. However, prototyping distinguishes itself with a clearer focus on feasibility and implementability rather than Lean Startup focusing on the business side. Incremental development and sprints, on the other hand, have their focus more on the way the work is organized - incremental development chopping it feature wise and sprints scoping it in time.

The motivation for using the described cycles clearly distinguish them from each other. Incremental development takes into account a wide mix of background ranging from business reasons to technical aspects and from risk management to evolving customer needs. Sprints, on the other hand, aim to exclude almost all of the aforementioned and liberate developers to focus on only the technical aspects.

Table 3 Summarized Characteristics of the Cycles

Cycle	Focus	Motivation	Goal	Developed by
Prototyping	Feasibility and implementability	Almost always technical in nature	Commonly executed to explore design space for a particular solution.	Can involve an individual developer, or a team of developers if a more complicated system is being explored.
Incremental development	Scoping the technical work feature wise.	A mix between business reasons, technical aspects including risk management, and customer needs.	The goal is to organize company operations as a whole in terms of releases.	Most commonly affects the whole organization, including obviously the developers but also sales, marketing, customer care, and so on.
Sprints	Scoping the technical work time wise.	Mechanism to liberate developers from constant changes to a fixed set of features to implement during the sprint.	Considers mostly development aspects and overlooks others, in particular if following Scrum interpretation.	Traditionally executed by a Scrum team up to 12 people; variations that enable synchronization between different teams exist.
Lean Startup	Learning and experimenting.	Business oriented in nature.	Validate or invalidate a business hypothesis with minimum amount of invested effort.	Usually executed only by a minimal team.

Similar to this, prototyping scopes the development into specific problem solving cases. Lean Startup is something of a mix in this sense, since its motivation is ultimately business oriented, but it surely has to take a wide range of different aspects into account in the end.

Lean Startup and incremental development have a similarity regarding their goals and the people they affect. In both of them, the intention is to scope the development work of the whole organization. The final goal is different, however. With Lean Startup the aim is on validating or invalidating a set business hypothesis with a minimum amount of invested effort and staff - this learning is the ultimate key and the produced software artifact is almost irrelevant. On the other hand, an organization probably does not want to waste any work efforts either with incremental development, but the produced software artifact is the most important thing in this case. Therefore, also the amount of people and different parts of the organization can be a lot greater than with Lean Startup.

With sprints, the goal changes again. Although the produced software artifact is unquestionably of high value, the main intention is to make sure that the defined technical and work management related aspects, such as the amount of people, stay the same during a fixed time period. In a way, prototyping is somewhat of a mix from each of the others. It scopes the work into a specific problem solving case like incremental development, but its main outcome is learning from an experiment as with Lean Startup. In addition, its focus is usually sharply on technical aspects as with using sprints.

7. Conclusions

In this paper we have presented an initial analysis of the different types of iterations. The cycles encompass the whole of product development and its different levels from business planning to product refinement. The higher abstraction level cycles such as Lean Startup and RUP can be achieved using sprints and prototyping. This way the software development process as a whole consists of iterations within iterations producing an interlinking whole that is more than the sum of its parts.

An interesting topic for further research is the developer perspective and psychological aspect of different types of cycles. For example, the motivational aspects of the cycle types may be rather different. In addition, we aim at creating a comprehensive conceptual model that covers the different iterations. With such, we see a lot of potential in industrial collaboration to help us validate the model as well as to test it in practice.

Acknowledgments

The authors wish to thank Need for Speed program funded by TEKES, the Finnish Funding Agency for Innovation, for its support.

References

[1] C. Larman and V. R. Basili, “Iterative and incremental development: A brief history,” Computer, vol. 36, no. 6, pp. 47–56, Jun. 2003. [Online]. Available: http://dx.doi.org/10.1109/MC.2003.1204375

[2] M. Stephens and P. Bates, “Requirements engineering by prototyping: experiences in development of estimating system,” Information and Software Technology, vol. 32, no. 4, pp. 253–257, 1990.

[3] K. Schwaber, “Scrum development process,” in Business Object Design and Implementation. Springer, 1997, pp. 117–134.

[4] E. Ries, The lean startup: How today’s entrepreneurs use continuous innovation to create radically successful businesses. Crown Books, 2011.

[5] T. Mikkonen and K. Systä, “Maximizing product value: Continuous maintenance,” in Product-Focused Software Process Improvement. Springer, 2014, pp. 298–301.

[6] H. Terho, S. Suonsyrjä, A. Jaaksi, T. Mikkonen, R. Kazman, and H.-M. Chen, “Lean startup meets software product lines: Survival of the fittest or letting products bloom?” 2015.

[7] M. Meboldt, S. Matthiesen, Q. Lohmeyer et al., The dilemma of managing iterations in time-to-market development processes, 2012.

[8] Berente, Nicholas, and Kalle Lyytinen. "Iteration in systems analysis and design: Cognitive processes and representational artifacts." *Sprouts: Working Papers on Information Environments, Systems and Organizations* 5.4 (2005): 178-197, 2005.

[9] P. Kruchten, The rational unified process: an introduction. Addison-Wesley Professional, 2004.

[10] C. Sandor and G. Klinker, “A rapid prototyping software infrastructure for user interfaces in ubiquitous augmented reality,” Personal and Ubiquitous Computing, vol. 9, no. 3, pp. 169–185, 2005.

[11] C. Floyd, “A systematic look at prototyping,” in Approaches to prototyping. Springer, 1984, pp. 1–18.

[12] P. Pierre Bourque and R. e. Fairley, Guide to the Software Engineering Body of Knowledge, Version 3.0. IEEE, 2014.

[13] M. Fowler and J. Highsmith, “The agile manifesto,” Software Development, vol. 9, no. 8, pp. 28–35, 2001.

[14] S. Blank, The four steps to the epiphany. K&S Ranch, 2013.

[15] A. Maurya, Running lean: iterate from plan A to a plan that works. ” O’Reilly Media, Inc.”, 2012.

[16] M. McCurdy, C. Connors, G. Pyrzak, B. Kanefsky, and A. Vera, “Breaking the fidelity barrier: an examination of our current characterization of prototypes and an example of a mixed-fidelity success,” in Proceedings of the SIGCHI conference on Human Factors in computing systems. ACM, 2006, pp. 1233–1242.

[17] A. Taivalsaari and K. Systä, “Cloudberry: An html5 cloud phone platform for mobile devices,” Software, IEEE, vol. 29, no. 4, pp. 40–45, 2012.

[18] K. Schwaber and J. Sutherland, “The scrum guide,” Scrum Alliance, 2011.