BEYOND AGILE: HOW SOFTWARE DEVELOPMENT CAN EVOLVE TO BE ANTI-FRAGILE

SAMUEL WASSWA

10/12/17

Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213

Table of Contents

Introduction

In this paper, we study and analyze the software development process of DynamicTech. We then assess the weaknesses of their processes and propose improvements that can adopted by DynamicTech and any other organization which develops software.

DynamicTech is a Software Company which spun-off Dynamic Trust, a company which specializes in providing agricultural services for commercial farmers. During its early years, Dynamic Trust began growing a big clientele of farmers. The company had to track the crop production process of the farmers right from planting to harvesting and finally to selling. To carefully monitor these activities, they deployed field agents who would manually collect information using paper- based forms. As the volume of information grew, it became increasingly error-prone and arduous to organize all this information. The company instructed the IT team to develop a web-based system to manage the whole process with specialized software developed for mobile phones such that field agents can easily collect data. What started as an in-house team with only two developers, slowly grew into a full-fledged team of eight developers.

The software which was called the Comprehensive Farmer Tracking and Consolidation System (CFTS) became a wild success. Soon, other Farmer Organizations took notice and became interested in adopting the system for their own operations. Dynamic Trust soon realized that their product could stand on its own. The management decided to spinoff the product into a software company which they called DynamicTech. DynamicTech had full autonomy from its parent company. Dynamic Trust only retained majority shareholding in the new company.

Software Development at DynamicTech

At the time of its formation DynamicTech had eight full-time developers. Within a year they had increased to twenty developers. The Chief Technology Officer(CTO) of DynamicTech was very forward thinking and quickly adopted Agile Software Development techniques[1]. Specifically adopting Scrum[2].

DynamicTech generally uses a combination of Scrum and Kanban which is known as ScrumBan[3]. Scrum is an iterative development method that ensures a “piece of working software is delivered at the end of each development operation called a sprint”. Roles, meetings and “process artifacts” are clearly defined. Kanban on the other hand is based on “Just-In-Time and Lean Production”. Kanban is very flexible compared to Scrum and doesn’t define roles, meetings or “process artifacts”. It instead focuses on “visualization of workflow, limitation of work in progress and measurement of lead time”[4].

When the team is about to begin a new sprint, they spend a whole day planning. The sprints last for two weeks. The CTO maintains a product backlog where he continually

adds features that may be requested by clients, features he thinks will add value to the system and bug fixes.
At the beginning of each Sprint. He will move items from the product backlog to the sprint backlog. It's from this backlog that the team begin estimating work for the sprint.

Because the team is fairly big it will be divided into five groups who will each carry out an estimation exercise using planning poker[5]. Planning poker involves the team making estimates by "placing numbered cards face down" and revealing each one's estimates and then discussing about the best estimate[5].

Tasks are then updated with the agreed upon estimates. Units called "story points" are used to estimate the size of a task and develop an idea of how long it could take. This is important because it helps keep track of the team's "velocity" which is how much they can deliver in a sprint versus the team's "capacity" which is how much time they have in the sprint[2]. After these planning meetings work begins. All the tasks are organized on a Kanban board. The team utilizes a Software project management tool called JIRA [6] . The tool links the tasks to the source code version control system. The main feature of the Kanban board is Work-In-Progress limit where "explicit limits" are set on "items that maybe in progress at each workflow state"[4]. This is to ensure that team members commit to finishing tasks they take on before taking up new tasks. Developers must log the number of hours they spend on a task in whatever "workflow state" it is. Daily "standup" meetings are held every morning for strictly five minutes where each team member talks about what they did the previous day, any impediments they are facing and what they plan to do on that day.

At the end of the sprint a sprint review is done. All developers are required to stop their tasks and any unfinished work is pushed to the next sprint. During the review, the developers present their work to the CTO and any stakeholders that may be present. The CTO then displays the team's velocity for that sprint and a burndown chart which shows the "cumulative time it takes to complete outstanding tasks for deliverable software"[7].

The CTO will then comment on how the team performed in the sprint and point out if the sprint velocity went up or down. A sprint retrospective will then be carried out where developers discuss the individual problems they faced and how they can improve in the next sprint. The CTO is not present in these meetings such that the developers communicate freely. These meetings are conducted by a chosen scrum master for the team. The scrum master is responsible for taking feedback to the CTO.

The Problem

For a number of months, the CTO began to notice worrying trends. The velocity of the team was inconsistent and it was becoming difficult to assess the amount of work the team could deliver. Some developers were also significantly underperforming compared to their counterparts. The sprint reviews became showcases for developers to brag that they had done more significant work than their colleagues. He kept getting reports that some of the developers were losing motivation. As a result, some of the features began traverse several sprints because they were always unfinished. Planning became an ego clash as several developers disagreed on the estimates. The CTO was also finding it difficult to cope with feature requests that came midway through the sprint. And the product backlog began to grow with each sprint. It began to feel like they were spending a lot of time following a methodological approach with little progress.

Scrum and the failure of agility

The "basis of an agile approach is to embrace change". "To be aware of changes to the product under development, the market and the environment" to name just a few[8]. Agile methods encourage adapting to change but we can see that DynamicTech is struggling with process. What is adapting in this case? Can the CTO begin changing this process which has worked well and is now part and parcel of the company's culture?

So is Scrum really agile when it is very prescriptive of process and provides little room for change. It seems Scrum is in direct contravention of the core tenets of the agile which state "Individuals and interactions over processes and tools" and "responding to change over following a plan"[1]. Scrum has created a rigorous strict framework. Utilizing practices like Kanban doesn't make it more agile. As long as the core process is maintained. It becomes difficult to change. With good reason, it will be hard for the CTO to change a process he has invested time and effort in. "Agile methods ask practitioners to think when it is far easier to follow the rules and claim you are doing it by the book"[8]. We tend to be comfortable with rules despite their restrictive nature.

Scrum has led to misconceived belief that agile is about product backlogs, standup meetings, burndown charts or team velocity[2]. All these principles while exciting at first quickly develop into dreary routines. This is also not helped by the fact that iterations are described as "sprints" which suggest that you run as fast you can, rinse and repeat.

Software development is really more of an obstacle course because it is impossible to foresee anything that can happen during this process. The process lends its self to adapting and reacting rather than following methodological process.

So the team has found itself stuck in rut. Following rules and prescriptions month after month "without gaining the experience they need to get to the point where they understand they need to move beyond the rules"[8]. It is easy to tell the CTO, that he should evolve his practice or he should fire the incompetent developers. But how does he make these decisions without understanding the underlying cause. Perhaps it's a case of lack of motivation. One of the agile principles states "build projects around motivated individuals"[1]. How does the CTO go about embracing change without significantly affecting the project? We aim to identify ideas that DynamicTech and any company which feels its stuck in a rut or it has failed to outgrow rigorous process that so called “agile methods” introduce unwittingly, can leverage to be able to respond to change tactfully.

Anti-Fragile: Beyond Agile

Anti-fragility, a concept developed by Nassim Nicholas Taleb means that a system increases in “capability or resilience as a result of stressors, faults, attacks or failures”[9].

Agile methods like scrum have failed to deliver agility as seen with DynamicTech. How do we overcome the failures of agile. We must evolve beyond the overloaded term “agile” and begin describing our ideas as anti-fragile. Anti-fragility is a better metaphor because “software is not designed and built”. This is “too deterministic and linear”[8]. Software evolves and so does its development process. To attempt to introduce a deterministic prescription for this process is a recipe for disaster. Agile in its purest form embraces this constant evolution by “harnessing change” and “welcoming changing requirements”[1]. Unfortunately, Agile methods have failed to capture this essence. Hence the need to embrace antifragility when developing software.

When our teams are in a rut the process can evolve to compensate for these new changes in the environment. When new requirements emerge, we must be able to accommodate them seamlessly. We are resilient to any stressors in the environment, the product, the market and the competition. We build fully self-organizing teams that adapt and develop with the changing requirements while staying motivated.

Anti-Fragile Software Development

How then does one ensure their software development process is anti-fragile? We believe an anti-fragile process should focus on

- Growing Skills
- Developing awareness
- Disposable software[10]

Growing Skills

At DynamicTech, one of the problems they had was some developers were significantly underperforming. One option they can look at is identifying skills gaps and making sure they address them through training or pair programming with their more skilled counterparts. Unfortunately, Scrum doesn’t address the problem of skills gaps within the team and instead assumes that one size fits all. A good way to be prepared is to make sure the team keeps learning and keeps growing their skills.

Developing awareness

Boiling Frog Syndrome

It is said that when you take a frog and drop it into boiling water it will immediately jump out. However, if you place the frog in a pan of cold water and gradually heat it, the frog will stay put and won't notice the dangerous temperature change until it is too late because of its ability to adjust it body temperature[11].

At DynamicTech, the CTO began noticing things were going awry after a long period time much like the frog and by the time he realized change was needed, the situation was already festering and change was going to be painful. He needed to develop awareness about the process and its shortcomings. Scrum standup meetings are good at helping the team develop awareness because they encourage constant communication.

Conversely, static communication tools like the product backlog fail to convey this constant stream of communication and lead to unwieldy backlogs that become unmanageable as was the case was with DynamicTech.

Disposable Software

The premise of disposable software is to encourage the concept of reacting to change by not attaching a lot of value to work that has already been done regardless of the effort that has been put in. Some of the things that make it hard to have disposable software are the notions of "maintainability, extensibility or reusability"[10]. These are not wrong per se but depending on the emphasis placed on them can reduce the antifragility of the software.

Effort should instead be invested in making software "replaceable" because we simply cannot predict the future. This surprisingly improves the properties we like in software like loose coupling and encapsulation[12].

Why teams get it wrong

The CTO of DynamicTech discovered Agile and was excited about the potential it has to change the way they conduct Software development. He immediately began to research "agile practices" and discovered Scrum was one of the most popular and adopted practices. He immediately started studying Scrum and he began implementing some of its processes in the team. He fell into a trap teams fall into when they are beginning agile. He focused on practices not principles.

The problem is focusing on practices and losing sight of the underlying principles. Also, beginners of Agile can't really apply principles because they don't have enough experience to objectively think about what they are applying[10].

The major problems of trying to apply “agile practices” can be summarized as

- Misunderstandings and misapplication of practices.
- Lack of enough support for beginners
- No support for growing the team’s skills or continuous learning
- Little support for other members of the team like managers, testers etc.
- **Agile methods are not really agile** because they don’t change or encourage change[10].

The agile manifesto states that “At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly”[1]. Yet ironically, very few teams do this and agile practices like scrum purport to do sprint retrospectives but offer little room for change in the process.

You are not really agile if you follow practices “by the book” and you do the “the same thing the same way without changing”[10].

So where do these agile practices get it wrong. Let’s expand on some of the principles of the Agile manifesto

Agile Principle	What it says
Individual and interactions over processes, tools, “tradition” and “canon”	Ideally something that is changing should not have a rule book
Working software over comprehensive documentation	These also includes plans, charts, estimates, wikis etc.
Customer collaboration over detailed contracts	These includes Lawyers, ignoring users etc.
Responding to change over following a plan	“Clinging to pre-conceived notions”, repeating the past, ignoring change

Table 1 : Taking Agile a step further[10].

Practices

We want to convey principals through practices but practices boil down to following a set of rules. The reality is some rules shouldn’t be broken like placing your code under version control but in other instances we have to know when to go beyond the rules and examine when a practice is good and when it is going to be dangerous. Specifically, what context will the practice be good.

We can safely say that one practice does not fit all situations because of the following reasons.

- We have a variety of project types.
- Programmers of different skills
- Managers who don't know anything about programming
- People who don't know how to work with each other.

The practice is therefore going to be different depending on the skill level[10].

A modern practice should

- "Be empirical" and it should be able to be tested and adjusted accordingly
- "Offer different approaches based on participant's skill level" i.e. different teams have differing skillsets. A common theme in agile practices is to tell people just reflect and fix it (sprint retrospectives). Again beginners just don't have the experience to be able to "self-reflect".
- "Sensitive to human needs and cognitive limitations"
- "Be inclusive and integral" i.e. every stakeholder should be able to be involved[10].

Stop Practicing and Start Growing

Grows Method

Andy Hunt came up with a methodology which they believe can tackle most of the issues we have highlighted in this paper. He calls it the Grows method[13]. In fact, most of this paper is really based on his work in identifying and assessing the weaknesses of Agile methods like scrum.

Grows method is based on four pillars

- Skill model
- Empirical, Experimental approach
- Inclusive i.e.it involves everyone not just the development team
- Self- determined i.e. the team decides what to do and how much to do

Skill model

The skill model is inspired by the dreyfus model of skill acquisition[14] where they concluded that there are differences in how people perceive the world, solve problems, acquire new skills and how they perform based on their skill level.

The skill stages are loosely defined as shown below

Skill Level	Characteristics
Stage 1: Novice	Little or no previous experience. Not interested in learning and just want to accomplish a specific goal. Don't know how to respond to mistakes well i.e. they need context free rules to follow.
Stage 2 : Advanced Beginner	Don't want the big picture because it is confusing to them. Need small, frequent rewards
Stage 3 : Competent	Develop Conceptual models, Troubleshoot problems on their own, Seek out expert user advise.
Stage 4 : Proficient	Want to understand the larger conceptual framework. Are frustrated by simplified information, will self- correct previous poor task performance, Learn from the experience of others.
Stage 5: Expert	Primary sources of knowledge and information. Continually, look for better methods, Work from intuition not reason/rules. Rules actually degrade performance to the level of a novice.

Table 2: Dreyfus skills model[10].

So how do all these people work together because

- Manager is likely a novice at software development
- Developer is likely a novice at the domain or business
- Everyone is likely a novice at process [10]

This skill model is not specific to a person, it is specific to a skill. You can be an expert in one skill and a novice in another.

Mapping Practices to skills

Skill Level	Practice
Stage 1: Novice	Should focus on safety, Context-free rules, checklists, version control, Continuous integration etc. Concrete feedback loops
Stage 2: Advanced Beginner	Iterations, team synchronization, Kanban, standup meetings, checklists
Stage 3 : Competent	Modern technical practices, Scrum
Stage 4: Proficient	Invent new approaches beyond current practice, Move toward intuition/expertise
Stage 5: Expert	Intuition and expertise ,not rules. Teach new solutions. Keep a beginner's mind.

Table 3: Skill Practice mapping[10]

Directed Empiricism

Here we want to make decisions with actual outcomes and backed by evidence. We avoid estimates, theories and charts.

We use feedback loops for everything.

The feedback loops are
- Setup before any activity is started
- Short i.e. they must be near real-time
- Real world i.e testing systems in production
- Iterative and incremental
- Anti-Fragile(improve them when they break)[10]

When we are presenting this framework to DynamicTech, we shouldn't force people to change or adopt this. We should instead encourage them to experiment and adapt where they see fit and emphasize the fact that it is just an experiment.

The feedback itself must be
- Concrete and measurable
- Must be short – Real time wherever possible.

Experiments in GROWS method are

- Cheap to run
- Very short term with a time-boxed deadline
- Generate Specific, measurable outcomes of value
- Conditions of test and outcomes must be agreed by the team or stakeholders
- No experiment fails and experiments only generate data to inform the next experiment[10].

Conclusion

In this paper we have identified some of the shortcomings of Scrum and how DynamicTech started facing these obstacles . We then presented a framework of ideas that can help guide decision making for the next steps when you are stuck with scrum or any other agile practice.

Note that we don't insist that DynamicTech should throw Scrum out but use this guiding framework to direct the CTO's next direction. Wholesale changes may be counterproductive and we believe adopting this thought process can rapidly improve the Software development process and make it anti-fragile.

In a nutshell we can summarize this framework as growing software through a form of iterative or incremental model, growing the skills of the team members and making sure the software development practices are "self-determined not imposed"[10].

References

[1] K. Beck *et al.*, “Manifesto for Agile Software Development,” *The Agile Alliance*, 2001. [Online]. Available: http://agilemanifesto.org/.

[2] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*, vol. 18. 2001.

[3] N. Nikitina, M. Kajko-Mattsson, and M. Stråle, “From scrum to scrumban: A case study of a process transition,” in *2012 International Conference on Software and System Process, ICSSP 2012 - Proceedings*, 2012, pp. 140–149.

[4] H. Kniberg, *Kanban and Scrum - Making the most of both*. 2009.

[5] M. Cohn, “Agile estimating and planning,” in *VTT Symposium (Valtion Teknillinen Tutkimuskeskus)*, 2006, no. 241, pp. 37–39.

[6] J. Fisher, D. Koning, and A. P. Ludwigsen, “Utilizing Atlassian Jira For Large-Scale Software Development Management,” *Proc. 14th Int. Conf. Accel. Large Exp. Phys. Control Syst.*, pp. 1–7, 2013.

[7] S. Berczuk, “Back to basics: The role of agile principles in success with an distributed scrum team,” in *Proceedings - AGILE 2007*, 2007, pp. 382–387.

[8] Andy Hunt, “The Failure of Agility,” 2015. [Online]. Available: http://growsmethod.com/articles/the_failure_of_agile.html. [Accessed: 17-Nov-2017].

[9] N. N. Taleb, *Antifragile: Things that gain from disorder*. Random House, 2012.

[10] A. Hunt, “Anti-fragile and feedback. Trying to make up for the failures of ‘agile.’ - Andy Hunt on Vimeo,” 2015. [Online]. Available: https://vimeo.com/131410262. [Accessed: 13-Dec-2017].

[11] A. Hunt and D. Thomas, *The Pragmatic Programmer*. 1999.

[12] M. Keen *et al.*, “Patterns: Implementing an SOA using an enterprise service bus,” 2004.

[13] A. Hunt, “Stop Practicing and Start Growing,” 2016. [Online]. Available: http://growsmethod.com/articles/stop_practicing_and_start_growing.html. [Accessed: 13-Dec-2017].

[14] S. E. Dreyfuss and H. L. Dreyfus, “A five-stage model of the mental activities involved in directed skill acquisition,” *Oper. Res. Cent.*, no. February, pp. 1–18, 1980.