Vol.2017-IS-141 No.5
2017/8/25

# Taking the Step from Software to Product Development when teaching PBL at a Business School

Marat ZHANIKEEV[1,a)]

**Abstract:** Many people in the software development community today do not realize that techniques like SCRUM were originally proposed as tools for general product/project development and problem solving in a traditional (non-specific) business environment. This misunderstanding is mostly due to the fact that SCRUM, Agile, Waterfall and other techniques are used mostly for software development today. Another indicator of this bias is that generic PBL-related certification, namely EXIN and PMP, remain under-proliferated in Japan. This paper shares experience from trying to bridge this gap while teaching PBL at a business school from the prospective of software development. Without ever actually touching software, the well-known SCRUM techniques are generalized to development of any product or service. The paper focuses on several specific techniques and concepts like Backlog, Burndown chart, strategies related to per-task budget allocation, and others.

**Keywords:** Problem-Based Learning, SCRUM, Generic Product Development, Innovation, Active Learning

## 1. Introduction

The **SCRUM** method, both as the keyword (actually, just English word) and the concept, was originally proposed in [1]. Notedly, the method was not about software development but discussed innovation in general. The paper was mostly about a new approach to developing innovative products like printers, cars, etc. Since both authors were from Japan, the examples also mostly came from Japanese companies (most of which are still around at present time).

SCRUM was not the only such tool at the time. Similar ideas were expressed and discussed in academia at the time. For example, the *systems thinking* method in [2] is not as much rival as a neighboring method in the same general area.

SCRUM and similar methods obtained the strong academic backbone several years later through the analysis of *team efficiency* in [3]. The paper models teams as loosely coupled systems, which receive and process signals from the outside. Although the paper does not formulate the point, this approach is derived from *signal processing*, where teams can be modeled as nodes in a distributed signaling network whose purpose is to generate a steady stream of new ideas. Some similarity can also be drawn with *Social Network Analysis (SNA)* were teams are participants in a larger social network. Academic depth of the topic aside, the focus in [3] was on how teams receive and internally process outside signals.

Closer to the end of the last century, the SCRUM method was firmly adopted in *software development* community [4]. Note that the method is referred to as SCRUM, which has further solidified its name. The software version of the method reached its maturity with the *Agile Manifesto* in [5] which traced its history back to the original paper on the SCRUM method [1], but otherwise focused only on software development. Note that SCRUM is not the only popular technology of its type. Methods like *Agile* and *Waterfall* are about at the same level of popularity and share many of the same features.

SCRUM is very easy to apply to software development. Primarity because majority of software products have *well-defined goals*. However, this also means that such products cannot really be defined as *innovation*. For example, Fig.1 shows a visualization of a daily scrum for a single (arbitrarily selected) team member. The visualization is a timeline of *source code* uploads (with *diff* size marked in bullets). Obviously, this scrum features a relatively high activity rate which is not representative of the general innovation process.

The generic applicability of SCRUM was already considered when the method was establishing itself in software community [6]. Again, this research also heavily builds on the original SCRUM paper [1] and focuses on Japanese companies at the time. However, this paper is a good indicator of the fact that SCRUM was considered for general innovation.

There are more specific attempts to generalize the SCRUM method in recent years. For example, SCRUM for hardware is considered in [7], with the conclusion that the method can be suc-

1 School of Management, Tokyo University of Science
Fujimi 1-11-2, Chiyoda-ku, Tokyo, JAPAN 102-0071
a) maratishe@gmail.com

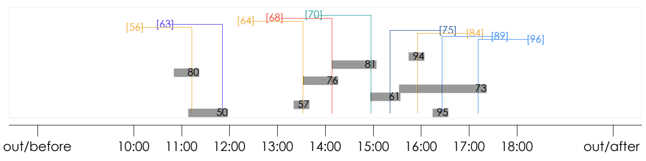


**Fig. 1** Visualization of a randomly selected *scrum* in software development, in form of a timeline of source code edits.

Backlog

| Task (specific details) | Budgeting game (numbers) | PO | Assigned to... | Budget | Actual |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |

**Fig. 2** A standard form for management of *scrum* tasks. Also known as *Backlog*.

cessful in such environments.

This paper takes one further step away from software development – this author has ample experience with the software version as per Fig.1 – to generic-nature products developed using the SCRUM method within an academic setting of a business school. Specifically, this paper discusses the experience of a class on Problem-Based Learning (PBL) implemented specifically using the SCRUM method. This paper discusses applicability of the standard SCRUM toolkit, namely such tools as *Backlog*, *Budgeting Game*, *Burndown Chart* and *Definition of Done* (DoD). Based on practical experience, this paper concludes with a practical advice for generic SCRUMs in form of the **decision algorithm** which can help with selecting an innovative product as the development target.

Several disclaimers are in order. First, this paper assumes that the reader has basic knowledge of the SCRUM method and specifically its main roles (Scrum Master = SM, Product Owner = PO, and Stakeholder) and tools. This paper is written in English in order to maximize proliferation, but all forms, algorithms, etc., were handled in Japanese in class.

## 2. Backlog and Task Management

*Backlog* is at the core of a SCRUM project. The form used in class is shown in Fig.2. As obvious from the figure, *Backlog* is a fancy word for a *list of tasks*.

**Definition of a task.** Task is defined as a *maximally specific atomic job* that contributes towards fulfilling the project's final goal. *Atomic* here simply means that the task cannot be split further. Tasks, however, are not limited in time and can be shared among multiple team members.

Here are some example tasks which are valid in an educational setting. Conducting a *web survey*, *researching* a given specific topic, development of a software or hardware component (probably not at a business school), planning an/or taking an interview, writing *documentation* in any form (manuals, surveys, reports, presentations, etc.), *analysis* of collected or otherwise acquired data. Work which *is not considered* as task includes anything which starts with *deciding...* or *considering...* or *discussing...*. Teams should avoid ambiguity and vagueness in tasks as much as possible. As a general rule of thumb, only *time consuming* work can be made into a task, where the *time* itself has a fixed and well-defined budget (more on this further on).

Back to the form in Fig.2, **Task** column takes specific details (not just the title) of what is to be done under the task. **Budgeting Game** is for leaving the record of the game which helps identify the correct (time) budget for the task – more details on the game are offered in the next section. **PO input** is the column where POs can indicate delayed or even removed tasks, ideally specifying the reasons for such a decision. Note that POs are supposed to have the long-term vision for the product and this form of task management is part of their normal duty. **Assigned To** is used to specify the team member(s) in charge of the task. **Budget** column is derived from the Budgeting Game with more details on the process offered in the next section. Finally, the **Actual** column is the actual time it took to implement a task – this column is filled post-factum, but is very important as it plays role when visualizing and managing team performance. Specifically, the two last columns are used to generate **Burndown Charts** which are simple indicators of team performance.

Note that Budget is always about time! At a business school, where money is often part of class discussion, it proved very difficult to persuade teams to abandon the concept of money completely and just focus on the *time dimension* of the budget. Note that both in English and Japanese, the term budget is applicable both to money as well as any generic resource, which deepens the confusion.

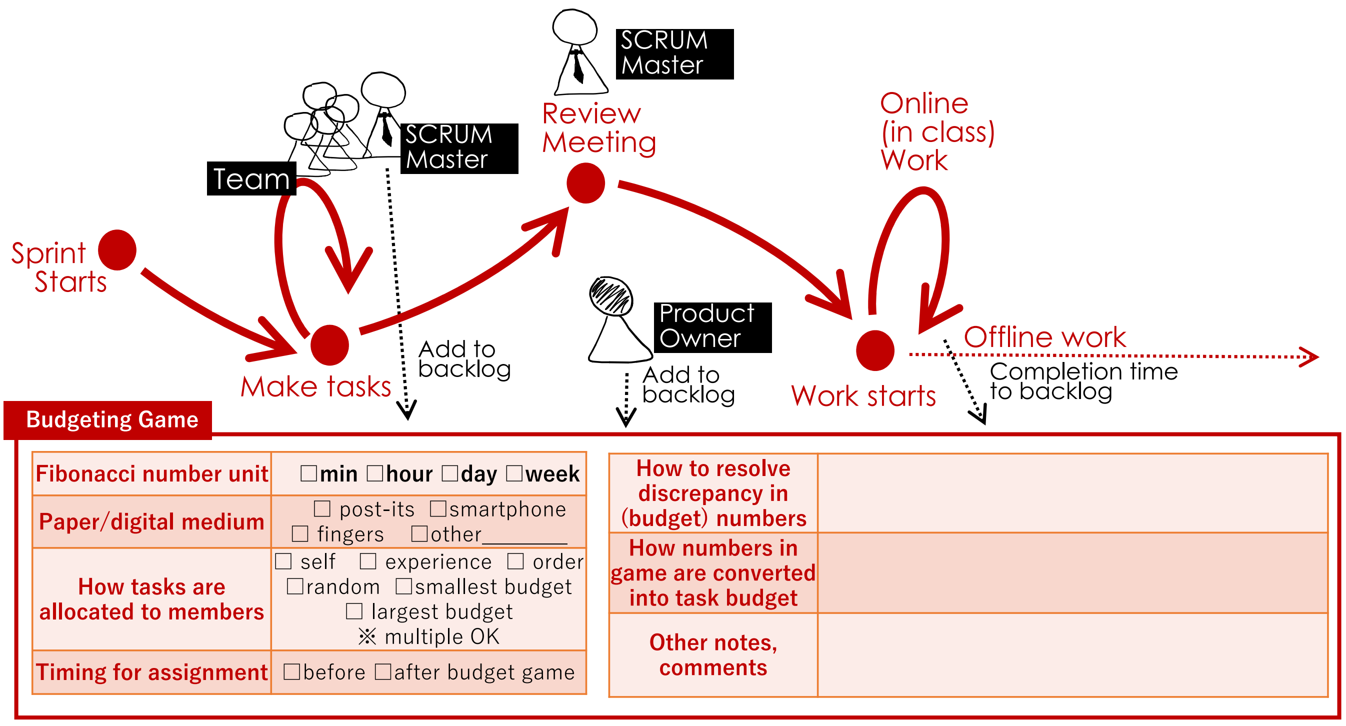
There is a logic behind focusing only on the time when discussing budget. First, there is the **Elevator Pitch** stage which teams undergo prior to entering the developing phase. In Elevator Pitch, teams take their ideas to potential sponsors. The actual SCRUM method is applied *after* the Elevator Pitch has been successful and the financial budget has been ensured. Past that point, the team should forget about money and focuses purely on the time resource during the development.

A disclaimer is due at this point. The above statement about budget is not necessarily true for all existing SCRUM projects. In advanced scrums, budget is often defined as a *resource* comprised of both money and time.

## 3. Signature SCRUM Techniques

Granted, the Budgeting Game is not part of all scrums in practice, it is a very useful tool for general team-based activities, including product development.

The following academic justification can be offered for the Budgeting Game. When playing the game, the team functions as a small **decision market** with the purpose of selecting the right budget for a given task. This function resonates with the signaling-by-interaction idea in [3]. Another related idea is the *Wisdom of Crowd* concept, which has already been connected to decision markets in academic literature. The core idea of a decision market is that *diversity* in experience and knowledge in each



**Budgeting Game**

| | |
|---|---|
| Fibonacci number unit | ☐min ☐hour ☐day ☐week |
| Paper/digital medium | ☐ post-its ☐smartphone ☐ fingers ☐other________ |
| How tasks are allocated to members | ☐ self ☐ experience ☐ order ☐random ☐smallest budget ☐ largest budget ※ multiple OK |
| Timing for assignment | ☐before ☐after budget game |

| | |
|---|---|
| How to resolve discrepancy in (budget) numbers | |
| How numbers in game are converted into task budget | |
| Other notes, comments | |

**Fig. 3** Learning the *budgeting game* by setting its parameters and discussing the core strategies within the team.

team member results in an emergent phenomenon when the total sum is greater than the all the individual member contributions put together. Conversely, having your SM or PO decide the budget is a very bad idea. Having the member in charge of the task decide his/her own budget is equally bad.

Fig.3 shows the form that walks the team through the process of solidifying the rules of its own Budgeting Game. Why not adopt a fixed set of rules? As the explanation below shows, there is a rich variety of components to pick from, which allows each team pick and choose, and finally arrive at its own unique configuration, suitable and comfortable for its current members.

**Budgeting Game Basics.** The game starts once a new task is decided and is about to be added to the Backlog. Everyone on the team votes on the budget for the task (decision market moment). It is common to use Fibonacci numbers, namely $\{1, 2, 3, 5, 8, 13, 21, ...\}$ to avoid unnecessary spread in budget estimates. Even with the fixed set of numbers, some scattering is natural as team members perceive the same task differently. This is resolved by discussion until the budget is fixed.

The form in Fig.3 tells the team to pick the *unit of time* (minute or hour is best for educational scrums), forces the team to fix a physical medium (post-its, smartphones, etc.) for the game, and draws the focus to the time when the task has to be assigned to a team member for implementation, etc. Note that there is no correct choice here, the team is expected to configure its own unique game environment, using the form as a guide.

The right side of Fig.3 focuses on the strategic (in academic way) component of the game. There are many possible strategies to pick from. For example, the team can set as a rule that members that come up with lowest/highest numbers for budget estimates, are asked for details. Or the team can decide to assign the task to a member prior to the game and exclude the member from budgeting discussions (fairness).

There are also several methods for converting the numbers that come up during the game into the actual budget estimate. Some teams can pick lowest or highest numbers. Some can do the opposite – remove numbers at both extremes and pick the average of the remaining numbers. Experience in class shows that this level of freedom results in a wide range of configurations generated by individual teams.

The upper part of Fig.3 depicts the overall timeline of activities during each Sprint – development is split into sprints each with its own set of tools (Backlog, Burndown, etc.). First, the team has the *Planning Meeting* for which SM serves as facilitator. The meeting is expected to generate tasks, and play the Budgeting Game for each. All such activities are immediately reflected in current Backlog. Once the scope of tasks for the current sprint has been decided, there is another, either team-wide or SM-PO or PO-only *Review Meeting*, at which the PO, who it expected to have the overall vision for the product, removes or delays some tasks if needed. Past that, the work on current Backlog starts and completion times for each task are marked under *Actual* in the Backlog.

SCRUM offers several tools for analysis and management of **team performance**. Arguably the most popular tool is the *Burndown Chart* in Fig.4. As per its name, the chart visualizes the rate at which the work *burns out*. The chart can be derived from many metrics in practice. In this paper (and in class), the vertical axis was used to plot the *sum of all existing Actual (completion) times minus the moving sum* which gradually advanced towards the end of the current Sprint. The horizontal axis is used to plot the *sum of all Budget values*.

The common shape found in Burndown Charts is a curve that hovers slightly above the diagonal line. In the SCRUM commu-

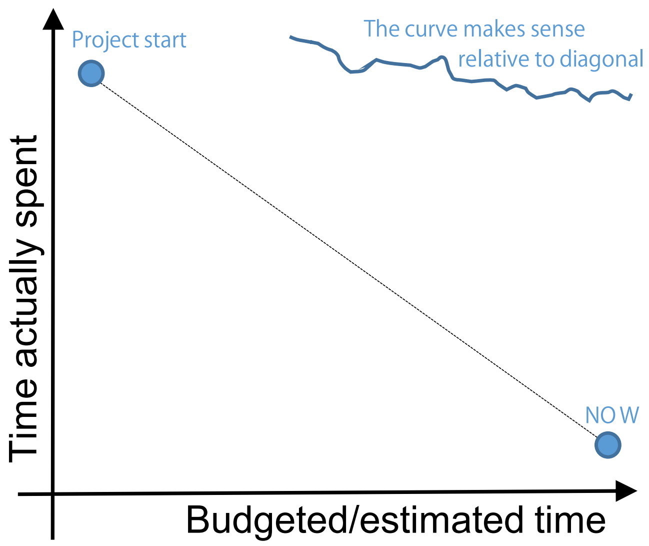


**Fig. 4** The form for writing *Burndown charts.*

nity it is common to discuss the performance by classifying the curve into several distinct types and offering advices as to how to rectify performance problems when detected.

## 4. DoD: Definition of Done

Repeating the earlier statement, software scrums are relatively easy to conduct [5]. In generic scrums, even if the ultimate goal is clearly defined, the final shape of the product may not be clear. For example, if a team's goal is – quote – to rectify the problem of long queues in front of women restrooms – endquote – stating the goal itself does not help much with its implementation. In generic scrums, products can be basically anything including physical things, services, manufacturing processes, etc. Which is why the Definition of Done (DoD) tool is extremely important for generic scrums.

**Definition of Done** is defined as the thorough description of *the conditions*, having achieved which it should be clear for an *outside observer* (including product user) that the product has reached its implementation target. The part about the outside observer is a must-have for DoD. Good parallels for DoD can be drawn with user manuals for commodity devices, service agreements for services, etc.

Stipulating parts of product's functionality as *out-of-scope* and/or *future work* within the DoD is a useful productivity tool. Such power is normally given to POs. It is important to do it as early as possible into the development in order to avoid crushes and disappointments closer to the release date. There is no shame in specifying part of your product as *future work*. In fact, software community has long adopted the concept of **Perpetual Beta**, which simply means that a product is never complete but is developed and used in parallel.

Having a well-defined DoD adds clarity to the project and improves team performance. The clearer the immediate next step in the development process, the better.

## 5. Conclution: What makes a valid Product?

First, this paper formally states that generic SCRUM-based PBL in academic classes can be (and was for this author) a suc-

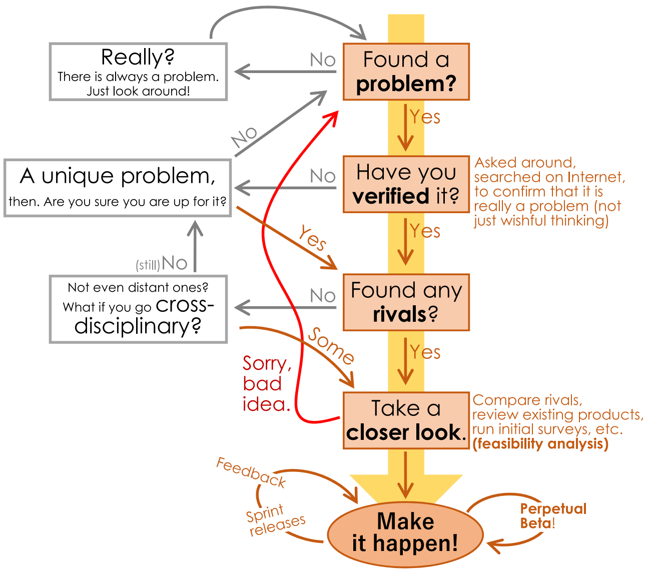


**Fig. 5** Decision algorithm the team can walk to identify whether it has selected a valid (in PBL definition) product for development.

cessful experience. However, there is much room for improvement. Specifically, the rest of this paper offers and important advice on the product-selection phase of generic scrums. In this author's experience, product selection is the weakest part in most teams.

In class, selecting the team's target for development starts with the *Elevator Pitch* (EP) tool, as the first step towards solidifying the product. EPs are 1-minute speeches in which the team presents its product to a potential Shareholder. Teams are expected to rapid-fire summarize all the main features of the product, compare it to rivals, etc. If the team comes up with a poor EP – it is useful to have a board of industry professionals to judge EPs – then it might be better for the team to look for another product altogether as the current product is very likely to fail.

Now, the above is a very very vague formulation. Added to the general vagueness of a generic scrum, we end up with majority of the teams confused about what to do and, specifically, about how to tell a good product from a bad one. This is when the decision algorithm in Fig.5 can help by guiding the team through the selection process. Fig.5 can be viewed as a better version of the EP. It covers several common (in this author's experience) mistakes made in product selection (in educational environments) and offers advice on how to avoid and/or rectify them.

Let us walk through Fig.5. First, the team is asked whether it has identified the P in Problem-Based Learning (PBL). Note that it is very difficult *not* to find a problem in today's rich environment of tools, services and even industries. *Mimi wo sumaseba* translated roughly as (if you listen closely) is a fit expression for such cases.

However, having identified a *seemingly* interesting problem does not necessarily mean that it is suitable for a SCRUM-based development. The second step in Fig.5 is to *verify* the problem. One can search on Internet, study academic literature, or simply ask around in order to verify that the selected problem is not the case of *wishful thinking* on the part of the team. If the answer is *NO*, then there is still a chance that the product is some-what unique but can still be the subject of a successful SCRUM project. In this case, as the algorithm shows, the team has to decide whether *it is up to this task*. If not, it is advised that the team picks another problem.

The next important step is to identify existing or potential **rivals** for your product. Rivals are not necessarily defined as products that compete in the same area and pursue roughly the same goals. As the figure shows, rivals can be cross-disciplinary – this is when some shared features or components are found in products from drastically different areas. In short, cross-disciplinary products are welcome as major drivers of innovation.

If the team has come this far in the algorithm, it is asked to take a closer look. In business this is known as the *feasibility analysis* but can be defined as tasks (Backlog and the full sprint) which compare rivals, review existing products, conduct surveys, etc. It is not rare that too close a rival or a major problem with the product are identified at this stage, leading to the necessity to abandon the product altogether. Granted some time has been wasted, it is better to start from scratch than to develop a product which will not receive any attention in the market.

Having reached the end of the algorithm, the team is cleared for the development process proper. This happens in accordance with the core SCRUM method, using the tools and techniques explained earlier in this paper.

## References

[1] Takeuchi Hirotaka, Ikujiro Nonaka, “The New New Product Development Game”, Harvard Business Review, February 1986.
[2] P.Senge, “The fifth discipline : the art and practice of the learning organization”, Doubleday/Currency, New York, 1990.
[3] P.Wegner, “Why Interaction Is More Powerful Than Algorithms”, Communications of the ACM, vol.40(5), pp.80–91, May 1997.
[4] M.Beedle, M. Devos, “SCRUM: A Pattern Language for Hyperproductive Software Development”, Pattern Languages of Program Design, Addison-Wesley, pp.637–651, 1999.
[5] M.Fowler, J.Highsmith, “The Agile Manifesto”, Software Development, vol.9(8), pp.28–32, 2001.
[6] Jeff Sutherland, “Retrospective on SCRUM and Its Implementation in Five Companies”, PatientKeeper, Inc., July 2001.
[7] “Application of Scrum Methods to Hardware Development”, Blackblaze, July 2015.