
© 2006 Xiaglow Institute

Reprinted by permission of Foresight Academy of Technology Press.
Cost Modeling Agile Software Development

Maarit Laanti  Petri Kettunen

Nokia Corporation, P.O. Box 321, 00045 Nokia Group, Finland
Email: {maarit.laanti; petri.kettunen}@nokia.com

Abstract: Many New Product Development (NPD) software projects use nowadays agile methodologies. These methodologies date back to the ’90s, and the Agile Manifesto was declared in 2001. However, already before that the concept of Agile Manufacturing was discovered to describe a corporation’s ability to quickly adapt to changing requirements. There is surprising amount in common between these two fields. This paper investigates the question of whether NPD software development companies could learn from the cost management research done for agile manufacturing. An industrial case example is illustrated. The observations suggest that there is still much room for improving the cost awareness in software product development in general.

Keywords: agility, agile software methodologies, software engineering management, software cost model, new product development.

1. Introduction

In the ‘90s several new lightweight software process models such as Extreme Programming (XP), Scrum, Feature-Driven Development (FDD) and Rational Unified Process (RUP) were emerging. The common denominator of all these agile methods, as they are now called, is the ability to adapt the change in product requirements in order to maximize the customer-value of the actually delivered software. Instead of forming a fixed plan at the beginning of the project, these methods allow the evolution of requirements during the project, and add the mechanisms and control how changes and new tasks are brought in the project. In new embedded software projects the agile methodologies are widely taken into use [12]. Because of the gap in productivity and the software needs, mastering agile technologies has become a key success factor in the software industry.

Meanwhile, Agile Manufacturing principles and techniques have successfully been taken into use in many other fields of industrial product development, such as the automotive industry [5]. Since the needs and goals of software product development are mostly the same, consequently, we could potentially adopt similar solutions even at the New Product Development (NPD) company level.

A key feature of the agile software methods is that they attempt to keep the cost of fast design iterations low. Managing the costs has always played a key role in the manufacturing industry. In this paper, we investigate the applicability of Agile manufacturing cost management to Agile Software Development.

Research question. The background discussion brings us to our exploratory research question: Are general agile cost models applicable to software development?

The rest of the paper is organized as follows. Section 2 explains the background of agile cost models. Section 3 then investigates the cost of agility, and the benefits and costs of achieving agility in software product development. An industrial case example is examined. Section 4 draws the conclusions and gives pointers for further research.

2. Agile Cost Models

Basically the total value of a product, as perceived by the customers, can be defined by the well-known formula [10]:

\[
VALUE = \frac{Quality \times Service}{Cost \times LeadTime}
\]  (1)

Agility strives to increase the Quality and Service factors by meeting the current customer requirements by being flexible to customer demands and market changes. Furthermore, the Cost and Lead Time factors can potentially be decreased by being responsive yet decisive with the market forces, and by improving the overall productivity of the product development workflows (lean).

In general, agility requires investments. Allowing agility costs more [7]. An agile response bears the following cost factors:

(i) Building the reaction capabilities (e.g., software product platforms) in advance anticipating future changes
(ii) Utilizing those capabilities to rapidly implement the responses to the actually realized changes

The business logic here is that by investing in preparations for the most likely changes beforehand, the cost of then actually implementing the changes becomes smaller. Without any proactive preparations, the cost of responding to changes becomes higher, and the net result is less profitable (for instance due to lost market opportunities caused by slower response time).

If the company is able to anticipate the right changes in advance and thus implement the consequent responses quickly when they are really needed by the customers, the initial investment to create such capabilities pays off as the fast response is likely to be a competitive advantage. On the other hand, if the changes are anticipated incorrectly, the investments in building the reaction capabilities based on false assumptions leads to waste.

Conboy and Fitzgerald propose a general cost model of agility [2]. The model is divided into (i) Cost of Change Creation during development; (ii) Proaction Cost; (iii)
Reaction Cost and (iv) Learning. The essence of their model is that making more changes consumes more resources, which can be measured in terms of money, time, etc. ‘Change’ in this context is a broad concept covering not only market and technology factors, but also for example social and organizational factors. The later the change needs are realized, the more they cost. Proaction is thus less costly than reaction. A robust organization can accommodate changes without taking corrective actions. Over time, the company can learn and thus reduce the cost of changes in the future. Over here, they have found the very essence of the agile methodologies. It pays to be agile.

Whether the investments in agility are justified (ROI) depends in particular on the nature of the products. If the end-product is very well defined – i.e. functional – it is beneficial to optimize the production just with a short feedback cycle (lean). However, the more innovative the end-product is, the more agile the production must be [6][9][14].

Heikkilä presents a model of an Efficiency Frontier, see Fig. 1 [7]. The performance of any production system can be measured in terms of its agility and cost efficiency.

The point is that, up to the efficiency frontier, there is room for improving both the agility and cost efficiency of the current production system. On the efficiency frontier, no longer can both dimensions be improved, and we must make a trade-off choice between the two. Notably, however, if we are able to move the whole frontier forward, there is again more room for improvements in both dimensions. This requires typically additional investments, though (e.g., completely new technology development).

An investment in agility could pay off in the future as the company is able to sustain profitability, whilst the non-agile competitors could eventually lose their business. However, it is often not straightforward to determine, how much focus should be given to each activity. This is a balancing act. In addition, staying agile requires constant attention and focused improvements, since in turbulent business milieus the operating environment is in flux, and the company has to tune its operations accordingly. There are often other agile competitors around, and the competitive advantages gained by agile responses are likely to be only temporary until the competitors catch up [3]. Note that Heikkilä sees agility as only one possible source for a company’s competitiveness [7].

Over the years, the software production industry has been able to make some leaps in profitability but apparently there is still room for much more. The fact that many software projects have experienced huge improvements in productivity with agility suggests that software projects typically are still far from the efficiency frontier (see for example [4][13]). Note that in manufacturing, companies competing on the efficiency frontier are also rare. These include only such giants as Wall Mart, Amazon.com and Dell, which have made their supply chain both fast and cost efficient as well as agile and adaptive [7].

As the Agile Manufacturing focuses on the cost of agility (what it costs to become agile) the agile software process is often focuses on the rewards of being agile (by improvements of efficiency) (Note 1). Conboy and Fitzgerald have developed an Agility Assessment Framework, where the reward (in euros!) is defined as a formula based on how well one is able to adapt to various kind of changes (in creation, proaction and reaction) [2]. Anderson proposes a systematic method of calculating the financial metrics of agile software production [1].

Lean software thinking also mostly concentrates on reasoning why using an agile method is less costly than using a plan-driven method, not the cost of the method itself. It even goes so far as claiming the cost and schedule are unimportant in new product development [11].

3. Application of the Cost Models to Software Development

One should take a more holistic view on agility in large-scale software product development than that provided by the basic agile software process models. In a large-scale software development the cost model of agility should include the business model as well.

3.1 Reflecting

Fig. 2 illustrates how we have mapped those investments and cost factors into the cost models described in Section 2. Moving to higher levels of agility and/or expanding the Efficiency Frontier incurs costs (I in Fig. 2(a)). At any point of time, the cost levels of different companies can be very different, depending on the past history (Fig. 2(b)). A more agile company (A) is able to implement more changes or implement the same changes with lower cost than the less-agile competitor (B). Any company can, however, improve its agility with some investments (I). On the other hand, the company could also lose its agility for example with inappropriate reorganizations. The amortized costs of improving agility in different ways can vary considerably, and the effects may be visible over different time periods (c.f., learning gradually vs. hiring an expert). The costs of attempting too many changes, or to strive for minimal costs beyond the current Efficiency Frontier become prohibitive. Striving for maximum agility brings the cost efficiency down (Fig. 2(a)).

A real-life example of this is a user interface (UI) developer’s investment in a framework. A large, purpose-fitting framework enables quicker implementation changes, thus the framework investment (I) pushes the efficiency frontier down. In the long run the effect becomes negative when the efficiency frontier is crossed. First, maintaining strict cost-policy in software development regarding also the employee salaries will first decrease the maintenance costs.
After some time, especially if put into extreme (i.e. pushed beyond the efficiency frontier), it will lead to increased framework maintenance and renewal costs because when the cut-down budget makes the former expertise to leave the company. The company may even be forced to create a new framework, if there is not enough knowledge to maintain and repair the former one.

With respect to Conboy’s and Fitzgerald’s cost model presented in Section 2, most agile software methodologies concentrate only on reaction to changes (Cost of Change Creation) during development and Learning [2]. The development team may swap and switch almost daily the product features. In software literature less emphasis is given to robustness, and anticipating the future changes. For example XP simply discourages anticipating future requirements proactively. Flexible software architecture design is not a concern.

Conboy’s and Fitzgerald’s model becomes even more interesting when considering large embedded projects. In large projects swapping features (change cost during the development) becomes more costly, as it is not just one team the change typically affects to, but many interdependent teams. However, investing in proaction (such as making more generic components, robustness and prototyping features before actually accepting them into project contents) becomes a more essential factor of overall cost saving.

Lean Thinking proposes that one should make a value stream map of the software production process [11]. That idea could be taken one step further, suggesting that a larger organization should compose a value stream map of the whole organization, where the (embedded) software production function is one element of the NPD value stream.

### 3.2 Example Case

The following examines an industrial case example reflecting these considerations. The example illustrates that it is possible to quantify the costs and benefits of agile NPD software development, with exact questions to assess when preparing investments.

Many telecommunications network element products have to support different transmission frequencies. Often new customers need new frequency support. In typical implementations the reaction to this means new hardware plug-in unit types with the associated embedded software support. Because of stringent time-to-market requirements the development of such new products (variants) is typically done with concurrent engineering of the new hardware and the embedded software functionality.

Two different cost scenarios can be characterized:

A **non-agile product development organization** does not realize the need for close co-operation between the related hardware and software projects. The different functions are in “silos” with little collaboration. The hardware development proceeds independently of the software project. The software developers cannot influence the new hardware design, and the knowledge sharing between them is scarce.
The software architecture has not been designed for accommodating new hardware unit types. Consequently, the new unit support requires extra design and testing efforts, and is error-prone. A Waterfall-based development model reveals major hardware-software integration problems at a late stage, while document-based milestones indicated illusory progress. All these factors cause time-to-market delays, leading to poor responsiveness as well as lower productivity.

A more agile way of doing this product development is to begin with a common systems engineering, which creates a modular, flexible product architecture (excellence in technology). The need for different hardware plug-in unit variants is proactively anticipated from the beginning (possibly with software platforms). The software architecture is designed for easy incorporating of new hardware unit types, based on standardized hardware/software interfaces (flexible software architecture). The related hardware and software projects are managed in close co-operation, with daily face-to-face interactions with the hardware and software developers (multi-skilled, flexible workforce). The product integration proceeds smoothly in an iterative fashion, avoiding major breakage delays (agile software methods). The net effect of all these contributing elements is good responsiveness of the NPD company.

Fig. 3 models typical behaviors in the cost structures of these different development strategies over time. A larger initial investment to proaction (b) pays off in the long run due to the resulting better responsiveness and shorter time-to-market. However, this depends on the future business cases.

The questions a company may ask are:
• Under what circumstances is \( b_x < a_x \)?
• What is the ROI of investing to proaction (b)?
• What is the effect to time-to-market?

The following value stream maps (Fig. 4) illustrate the example case described previously. The more agile approach (Section b of Fig. 4) leads to shorter time-to-market and thus responsiveness by proactively preparing to the new customer need, and by performing the different engineering activities concurrently with intense collaboration between the systems, hardware, and software engineering functions. The time values shown here are not based on any particular project case, but in our experience those kinds of ranges could well be observed in real product development.

4. Conclusions

In this paper, some agile manufacturing cost models have been investigated in respect to software development. These have been compared to other cost models, and developed further.

With respect to the Research Question in Section 1.1 we can conclude that
• Certain agile manufacturing cost models can be successfully applied also in software development (as seen in Section 3). The closer companies move towards the efficiency frontier, the more important cost management becomes. However, we must keep in mind that new innovations may move the efficiency frontier, and that agility requires some investments.
• Most agile manufacturing concepts have been applied in some form in software development. As the costs become of more importance, product placement on the market becomes of greater interest.

However, this paper leaves room for further study:
(i) Further developing the cost models of agility for analyzing and visualizing the benefits and associated costs of improving the different dimensions of software product development agility. This could possibly be done based on general-purpose strategic management tools (e.g., Balanced Score Cards [8]).
(ii) Collecting empirical evidence for supporting our propositions. This could be done by making a thorough cost analysis of an agile software project or enterprise.

References


Notes

Note 1. This is obviously due to the fact that many product development organizations often work far from the efficiency frontier.