Doing Small-scale Agile Projects Efficiently and Profitably

Ilkka Laukkanen <ilkka.laukkanen@futurice.com>

Futurice Oy

1 Introduction

Futurice is a software agency of about 140 people with offices in Helsinki and Tampere, Berlin and London. We offer consulting, design, development and training services to clients from diverse industries.

Our projects are generally done on a tight schedule and without complete, concrete specifications up front, instead evolving quickly from a concept to a polished product. In terms of project outcomes and customer satisfaction we have been very successful in working in this agile manner. The aim of this paper is to document some of the methods we've used to be agile and to keep projects on time and in budget while helping our customers with their business, and keeping both them and our employees happy while doing it. These patterns that are key parts in our way of working and contribute greatly to being successful and profitable while working on small-scale projects in an agile way.

In the context of this discussion, efficiency is taken as a qualitative measure of the time during which employees assigned to a project are actually able to do useful project work, in proportion to the billable hours they spend on that project. Efficiency is decreased when developers or designers have to hunt for information, write exhaustive documentation instead of new features–unless documentation itself is the aim–or generally do anything that isn't directly advancing the project towards its stated goal, and is instead waste [5]. Some inefficiency is inevitable, and some is necessary: sticky notes have to be moved on Kanban boards, project email has to be read and meetings attended and so on, and this is well and good, but increasing efficiency would allow to shorten project timespans, which would make for more compelling quotations.

Profitability is defined in the short term as avoiding overruns–in the case of black box projects–and in the long term as being able to charge realistic rates despite heavy competition. Overruns are caused by many things, but generally are due to either solving the wrong problem, or solving the right problem but in the wrong way.

Small-scale here means both brief and of limited scope. Agility is on one hand an umbrella term for the methods we use, but it is also a desirable element of the customer's perception of us.

2 End to End

In our projects customers often present only a preliminary concept, which we then start to develop and design further. They look for us to help develop the idea into a concrete,

useful product or service and see this plan through to completion. This requires a sense of ownership of the product, and pride in the work that we do. We want customers to have the sense that their idea is in good hands and that we care about what comes out of it. The more we show we care, the more trust is placed in us.

From the customers' viewpoint, getting everything from consulting and design through implementation and training to life-cycle management from one place is beneficial, because it avoids expensive knowledge transfer phases between subcontractors. When projects are short, the extra effort this requires toward comprehensive documentation starts to become a prohibitively large fraction of the total project workload.

Internally there needs to be cohesion between design and development throughout the project. For a given total project budget B, an overrun is guaranteed if a designer spends $\frac{1}{2}B$ coming up with a design that will cost $\frac{3}{4}B$ to implement.

For an agency such as ours, building repeat business and keeping customer satisfaction high is important, because a lot of business depends on referrals, recommendations and reputation. When we take an interest in the customer's business and actively help them develop it, we make an investment towards our own future. To achieve this it is necessary not just to cultivate a capable team, but also an environment where communication towards to customer is open.

Therefore: when pitching and planning, get input from designers, developers and people with experience in maintaining systems; come up with the next chapter in the product's story and pitch that as well. During the project get developers and designers working closely together. Use Periodic Demo so everyone knows what is being built and shares the vision.

This is related to the Cross-Functional Teams pattern [1] in that tight co-operation between different parts of the organization is needed to keep everybody on track, and the Vision pattern [4] in that a a shared vision is required, although it is more a joint effort than a product owner creation.

3 Periodic Demo

When a product is taken from a seed idea to a complete implementation, a lot of communication is needed to ensure that the thing that is being built matches with the customer's perception and need. Verbal or written communication is fraught with the danger of misunderstandings, omissions and other sources of errors. There is no better way of showing progress, getting feedback and building a common understanding than handing the customer a working product as soon as possible, and keeping on handing them when the product improves and develops.

The customer's unspoken and undocumented preferences will quickly become apparent as they get to use even a rudimentary demo version of the product. It will also help determine future direction, as not all ideas that sound good in meetings and on paper end up translating well in actual use. Furthermore, since the basis for the changes we end up making is the working product, we have confidence that we're using the best, most accurate information available, instead of modifying unrealised plans whose validity we cannot fully know.

Also, by striving to keep the product in a state where it could conceivably be released at any time, we force ourselves to do integration at an early stage. Integration is a major source of technical risk for any project, and doing it early is an effective way to manage it, as Reinertsen writes in "Managing the Design Factory", pages 224–229 [7].

Therefore: commit to having a working first version of the product ready for the customer to use at the end of every iteration.

This pattern is similar to Build Prototypes [2] in that work is being done to better understand the requirements, especially latent ones, but these products are not thrown away, instead becoming product increments [3]. In small projects we can refactor very aggressively to keep the product from becoming prototype spaghetti.

4 No Handoffs

The communication overhead discussed in End to End applies not just between many subcontractors collaborating on a project, but naturally also between teams and individuals within one company. If a customer comes to a representative with their ideas and problems, and that representative talks to a team–and in extreme cases maybe that team then talks either directly or via a proxy to e.g. an overseas team–the process becomes a game of Chinese whispers. This often resembles the situation described in Periodic Demo, where consensus cannot be reached due to poor communication, and quality suffers as a consequence.

This is best remedied by bringing the team close to the customer, both in a geographic sense and in terms of communication. In quick, small projects it is important that the team has direct access to the customer, and conversely the customer to the team. This way the feedback and guidance can be given unimpeded and all the important questions asked directly from the people that give the answers.

Intuitively it would seem that this approach results in much of the developers' time being spent handling feedback, but this is not necessarily the case. In "Leading Lean Software Development" Mary and Tom Poppendieck cite two examples of removing handoffs resulting in major benefits for the customer (pp. 20–21, 222-223) [6]. Instead of the team being swamped with feedback, the mental models of the team and the customer were more aligned, and as a result the customer was more satisfied with the end product.

The same goes internally too. The less rigid the product definition, the more communication is necessary (see "Managing the Design Factory", pp. 113–115) [7]. This means that team members must be in contact at all times when working with the kind of vague specifications we usually have.

Therefore: remove impediments to direct communication and foster team engagement with the customer, but make sure this does not swallow too much time by:

- having regular face-to-face meetings;
- having team members receive and respond to feedback;
- having the customer provide contacts for direct technical questions.

This requires a somewhat active customer, but our experience is that they are often willing to get engaged in these less rigidly defined projects, being invested in them and excited about them to begin with.

References

1. Cross-functional teams. ScrumPLoP published patterns. Website, referenced Feb 9 2012. https://sites.google.com/a/scrumplop.org/published-patterns/team-pattern-language/cross-functional-team.
2. James O. Coplien and Neil B. Harrison. Organisational patterns: Build prototypes. Wiki, referenced Feb 2 2012. http://orgpatterns.wikispaces.com/BuildPrototypes.
3. Lachlan Heasman. Regular product increment. ScrumPLoP published patterns. Website, referenced Mar 5 2012. https://sites.google.com/a/scrumplop.org/published-patterns/value-stream-pattern-language/regular-product-increment.
4. Lachlan Heasman. Vision. ScrumPLoP published patterns. Website, referenced Feb 9 2012. https://sites.google.com/a/scrumplop.org/published-patterns/value-stream-pattern-language/vision.
5. Mary Poppendieck and Tom Poppendieck. *Lean Software Development*. Addison-Wesley, 2003.
6. Mary Poppendieck and Tom Poppendieck. *Leading Lean Software Development*. Addison-Wesley, 2010.
7. Donald G. Reinertsen. *Managing the Design Factory*. The Free Press, 1997.