Prototypes are forever
Evolving from a prototype project to a full featured system

Hugo Corbucci[1] and Mariana V. Bravo[1]

Agilbits, Sao Paulo, Brazil,
{hugo,marivb}@agilbits.com.br

Abstract. This paper shows you how to handle prototype projects.

Key words: prototype, agile methods,

1 Introduction

Prototyping is one activity that most developer have heard about. Fred Brooks mentioned it in The Mythical Man-Month [**?**] and most agile methods adopt various ideas related to prototyping. Nevertheless, it is also one of the most feared practices by those who had some experience with it.

Most of the times, using software prototypes is traumatic because successful prototypes look very much like full features and customers usually want to integrate the prototypes to the working system and move on. However, prototypes are frequently created in a "quick and dirty" fashion and the result is not adequate to be incorporated in a full featured system. It is, however, quite hard to explain this fact to the stakeholders who usually do not want to spend any more money in this "already working" feature.

They, therefore, switch priorities and work efforts to other parts of the system and let the rough prototype lost within the code base. Months or years later, the prototype becomes an importat part of the system but is filled with bugs, unhandled corner cases and, frequently, crappy code. Nobody remembers what it was supposed to do or whether it is really important. Maintainability gets deeply affected and developers have that natural and unpleasant I-told-you-so feeling.

Developers that have been through the pain of maintaing those dirty prototypes hardly accept to work with prototypes anymore. If they are forced to, they make it so that there will be absolutely no way to integrate the prototype to the existing system by either using a different platform, language or even creating prototypes in other medias. That inflexibility can reduce the capacity of responding to changes quickly and therefore harm the clients' interests.

This paper presents how a four colocated people team managed to start a prototyping project and evolve it naturally to a full featured software. The organization of this work follows a chronological order as the project evolved softly from a series of prototyped features to a full featured software. Section 2

will present the project as the team received it. Section 3 presents the working system established by the team to create the software based on prototypes. After some time, the team felt that the customer was shifting to a full featured idea as described in Section 4. The following section (Section 5) shows how the team adapted to those changes to accomodate both ideas. Finally Section 6 presents the current status of the project and Section 7 concludes with a summary of practices that were useful to pass through this experience without that much pain.

2 Starting the project

Back in March 2008, Agilbits was hired to do some consulting with one of the biggest movie producing companies in Brazil. The client had a fantastic idea for a software to write movie scripts and needed some help to evolve this idea. The company's job at the time was to scout the market, discover competitors and provide an estimation of the work needed to turn the client's idea into working software.

For such work, one consultant was assigned to understand what were the client's needs and desires and two developers were asked to analyse the existing software and evaluate the possible development paths. After about 3 weeks of consulting and studies, the team handed a deck full of story cards estimated with the use of two platforms. The first one was an existing open source software with several features and a copyleft license. The second one was an Eclipse Rich Client Application using Eclipse's open source framework as the basis for a new software.

That first estimation suggested that a four people team with a half-time dedication would be able to build a partially working prototype of each feature described in about nine months of development using the existing open source software and about one year using eclipse's platform. The open source solution had the advantage to provide full functionality of several other features. For a full featured system, the estimation was over 2 years of work on the Eclipse version and about a year and a half for the open source one.

After some discussion, the client opted for the eclipse based solution because his business plan was to keep the solution closed source and sell the project to external investors. He also chose to have just a prototype version since 2 years seemed like a too heavy investment for him alone.

Such choice terminated the consulting contract and created a new one with a majority of development time and a few hours of consulting. Originally, this new contract mentioned a 4 developers team working on an open scope providing 160 hours of work each month. It specifically stated that the developers would work on pairs all the time and that the developed system should have automated tests to the production code.

The project goal was to create a software prototype with most faked (or simplified) features and a few working ones. The client would use this prototype to present his ideas to some investors in Hollywood by October 2008. This

meeting would either boost the project's development to a full featured system if the investors liked the idea or end its development in case they rejected the proposition.

That was the team's vision of the project when the development begun. A short seven months project whose fate would be decided by its capacity to impress Hollywood investors. Therefore, the main goal was to provide an excelent support for the client's show case to ensure the project's growth and success. The next section (Section 3) describes how the team organized itself to achieve this goal.

3 Developing a work system

Given the project's situation, the customer was always pushing for new features as fast as possible considering only one specific use scenario. This meant that, for most features, there were several cases which the team was asked **not** to handle. Regarding the source code, this meant a lot of conditionals, several spikes becoming permanent solutions and a fair amount of unhandled exceptions, ignored errors or non functional regular behaviors.

The team knew since the beginning that the client would change his mind over time. After all, it was partly to better understand his idea and its aplicability that he wanted to build this prototype. So things were going to change and features would be developed to later be thrown away while code produced only for a quick spike was going to become part of the system. Therefore, the team started investing a little on design, automated tests and refactoring since the beginning and made it clear for the client that there would be some work done on features after he accepted them to polish the work.

The first few month went quite smooth. The main features developed were to integrate with a text format coming from competitor software, provide a simple text marking feature and a visualization feature to manipulate and visualise the marks. For those features, it was quite simple to avoid gaps since there were not many business rules involved. Problems started to appear once the script writting business rules started to show up.

The client's presentation script was evolving as the software did and the team soon started to add conditionals to ignore cases he would not enter in. By October, the main features were ready but new discovered features were still incipient and the client was not feeling confident to present the software to the investors. However, he started to make contact with a few people to schedule a meeting by the end of November 2008 and December 2008. Those dates became our new deadline until which all efforts should be focused in making those incipient features available for the demonstration.

A new feature pressure installed itself since the projects fate would be decided at the demonstration and it was close! The customer wanted the team to ignore corner cases, speed up delivery and ensure the demonstration would run smoothly. The excitment from the important presentation to other people let the development team highly motivated to deliver all features the client had asked

for. Although unit testing and pair programming was a mandatory rule on the team, the general will to quickly deliver the features decreased the code quality.

4 Changing the rules

December 2008 arrived and passed without any meeting. The company that the client was in contact with had just been acquired by another one so any project presentation was useless until things settled down. That news pushed the deadline away for another 3 or 4 months at least. Along with this news came the information that the client had formed a dramaturgy experts group to help him understand better how to structure the software.

This new context relieved a 4 months pressure of upcoming deadline over a team which was beginning to feel the burden of unhandled technical debt. All members of the development team agreed that the code was getting complex and the quality was decreasing which was affecting productivity and speed. The software was now going to have a set of beta testers and it needed to perform decently to allow the users to suggest improvements in the work system.

The general feeling was that the project was no longer aimed at a simple presentation to investors. It was softly switching to a more elaborate and end user oriented software. The current development approach would not be able to support this new use of the system. The change had to be clear to the client so that development efforts would be directed to address this new way of working.

The warnings came surely from the dramaturgy study group. They started having troubles with several known and unknown corner cases, behaviours and just plain old bugs. The client started to notice that the users were having several troubles with the software and decided we needed to invest more in usability and user experience. To which the team replied that it would also mean less new features.

At this point, the client started to understand the dilema that the developpers had felt so far. How to keep a good rhythm of new features and still cover most use cases of existing features? Another critical issue in the software was that, so far, most features aimed at visualization and insertion of meta data in the movie script but users were claiming for basic text editing features ignored so far.

The team estimated that to have an editor with the basic features expected by the client would take at least three full iterations. This was completly unacceptable to the client since it would mean no new feature until the moment when he would possibly be able to show the software to investors.

5 Adapting to the new rules

6 Current status

7 Conclusion

References

1. Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. J. Mol. Biol. 147, 195–197 (1981)
2. May, P., Ehrlich, H.C., Steinke, T.: ZIB Structure Prediction Pipeline: Composing a Complex Biological Workflow through Web Services. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 1148–1158. Springer, Heidelberg (2006)
3. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
4. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid Information Services for Distributed Resource Sharing. In: 10th IEEE International Symposium on High Performance Distributed Computing, pp. 181–184. IEEE Press, New York (2001)
5. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: an Open Grid Services Architecture for Distributed Systems Integration. Technical report, Global Grid Forum (2002)
6. National Center for Biotechnology Information, http://www.ncbi.nlm.nih.gov