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 an activity that most developers have heard about. Fred Brooks mentioned it in The Mythical Man-Month [1] as one of the best ways to provide a quick view of a feature to the clients or users to help them make a choice. Dynamic System Development Method (DSDM)[2] is heavily based on prototyping and other agile methods also adopt many ideas related to it. However, those who have had some experience with this practice feel uncomfortable with it.

Successful software prototypes look very much like complete features given a certain execution path. Therefore it is common that the customers get so happy with it that they want to integrate the prototypes to the working system and move on. The problem is that prototypes are frequently created in a "quick and dirty" fashion and the result is not adequate to be incorporated in a full-featured system. Yet it is quite hard to explain this fact to the stakeholders who usually do not want to invest any more money in this "already working" feature. The consequence is that they switch priorities and work efforts to other parts of the system and leave the rough prototype lost within the code base. Months or years later, the prototype becomes a 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 are not entusiastic to work with prototypes anymore. If they have 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 ability of responding to changes quickly and therefore harm the clients' interests.

This paper presents how a four-people collocated 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. Section 2 will present the project as it was first presented to the development team. Section 3 presents the work process 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 acommodate 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 much pain.

2 Starting the project

Back in March 2008, our company was hired to do some consulting for one of the largest movie producing companies in Brazil. The client had a great idea for a software to write movie scripts but had absolutely no knowledge about software development. He wanted to mature the idea and understand how much investment it would take to turn it into working software in order to establish his business plan. The company's job at the time was to scout the market, discover competitors and provide an estimation of the work needed to develop the client's idea.

For such work, one consultant was assigned to understand what were the client's needs and desires and two developers were asked to analyze the existing script writing programs and evaluate the possible development paths. After about 3 weeks of consulting and studies, the team handed a deck full of story cards with two estimates each, based on the use of two possible platforms. The first platform was an existing open source software with several features and a copyleft license. The second one was an Eclipse Rich Client Application developed from scratch using Eclipse's open source framework.

This initial estimation suggested that a four people team with a half-time dedication would be able to build a 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 complete system, the estimation was well 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 due to the license restriction of the open source one which conflicted with his business plan. He also chose to develop only a prototype of the idea since 2 years seemed like a too heavy investment for him alone.

After the investigation phase, the consulting contract ended and a new one with a majority of development time was established. 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 toinvestors 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 it.

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 investors. Therefore, the main goal was to provide an excellent support for the client's demonstration 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.

Things were getting quite unpleasant from a development perspective but since the client satisfaction was still high, there was little that could be done. However, external interference was about to change a bit the situation. Section 4 will explain how the project got affected and what new direction those changes pointed to.

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. By February 2010 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 quickly 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. So he took another action which indicated changes in the business plan of the project. He decided

he wanted to have another team working on an editor feature that would allow all the features he wanted.

The team did some research and ended up discovering an open source Eclipse Rich Client WYSIWYG (What You See Is What You Get) HTML editor[1]. The editor relied on a reimplementation of Eclipse's StyledText component which is responsible for rendering text within Eclipse editors. This kind of knowledge was close enough from the one needed to implement the features our client wanted on our application. So the client outsourced the development of this underlying infra-structure as a way to keep the feature producing speed while attending the users' requests.

It took about two iterations to establish the new rhythm and adapt to the idea of a new work process. After that, the development team was concerned with the increasing complexity so they started to track some data from the source code. The first metrics was the amount of `FIXME`, `TODO` and `XXX` marks in the code. As previously mentioned, the development team added those marks everywhere the felt a corner case or a behavior existed but was not handled. Each mark had a small comment associated and the kind of the mark determined the criticality of the problem. Figure 1 shows the evolution of those marks during the project.

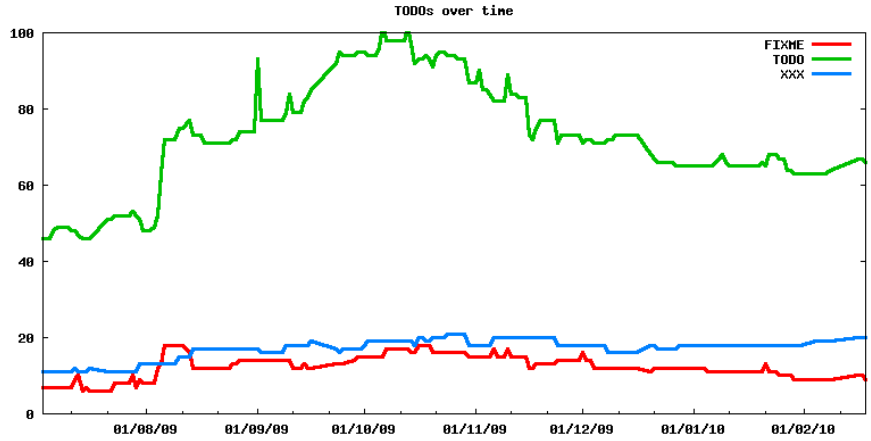

Fig. 1. Evolution of FIXME, TODO and XXX marks in the source code

5 Adapting to the new rules

Investing in reducing instanceof uses, TODOs and FIXMEs as well as increasing line of tests. Noticing most issues are coming from UI. SWTBot?

[1] `http://onpositive.com/richtext/` - Last accessed on 20/02/2010

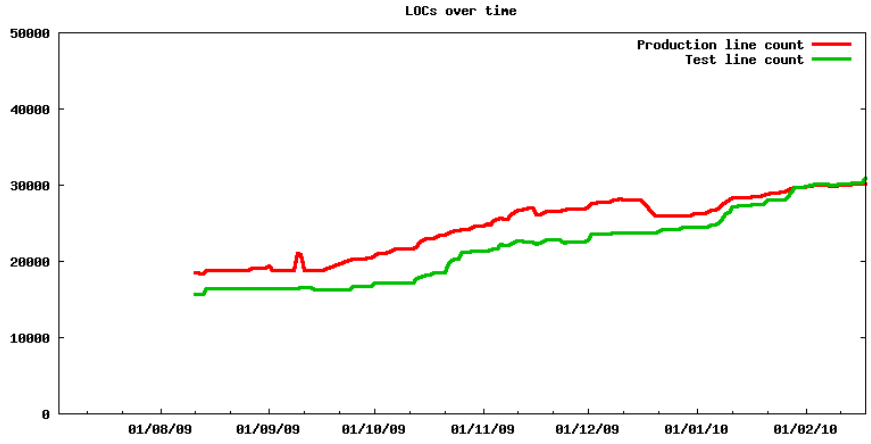

Fig. 2. Evolution of production

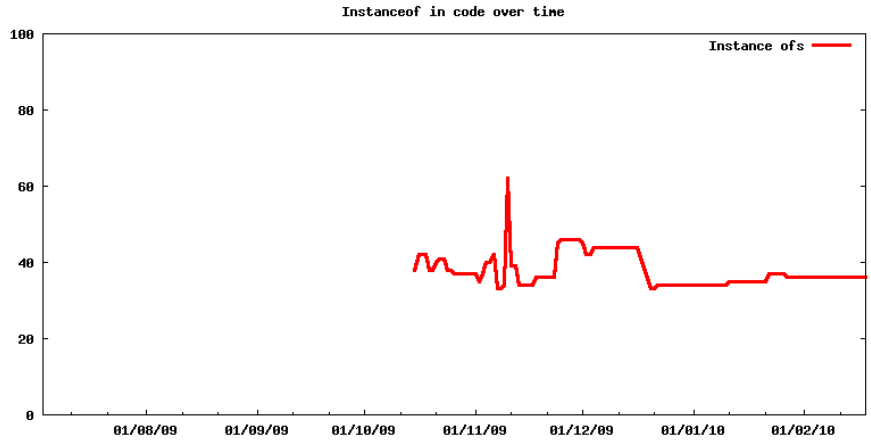

Fig. 3. Evolution of instanceof in the code

Effort dedication to integrate the new outsourced component developed.

Always handle (even if it means throwing an error to the log) corners cases or unexpected input/paths. Always refactor before completing any task. Tolerance 0 to bugs. Bug means highest priority.

Introducing another developer in the team.

6 Current status

Test coverage

Testing in other platforms

Merciless decoupling

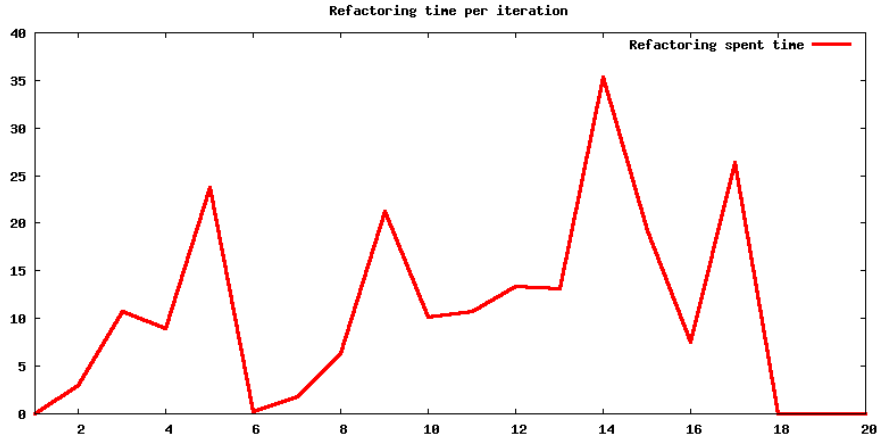

Fig. 4. Explicit refactoring time by iteration

7 Conclusion

Team's effort to keep tests level (ignoring known issues not solved).
Refactor tests.
Refactor prototypes (their code must be good).

References

1. Brooks Jr., F.P.: The Mythical Man Month: Essays on Software Engineering. Addison-Wesley (1975)
2. DSDM Consortium: DSDM: Business Focused Development. Addison-Wesley (2002)