Saving a Failing Project

Adrian Kosmaczewski

2008-08-11

In 2006 I had the opportunity to work as a "project leader" into a small failing project. Three developers were working in an ad hoc basis, creating a software application for an important client (a government office in Lausanne), without any kind of detailed formal specification, without any kind of design documentation, and with strong pressure from the management to release the application, even if not in an usable state. Needless to say, the project was also beyond budget.

I had just joined this company a couple of days ago, and the management asked me to take the project in charge. Not an easy task, particularly because it was my first experience of this kind.

The client was pushing to get the software it had paid for (it was a desktop reporting application for the Police department), and had not got any kind of preview yet. So the first thing I did is to pick up my copy of "Leading a Software Development Team" book and read chapter 2, "I'm taking over the leadership of an existing project // where do I start?" and get a thorough read:

> The first thing that you should start to do is to review the situation. This involves more than just absorbing impressions; you need to organize these impressions into a framework. Try to organize your thoughts into the following areas, and in each area try to separate technical issues from personnel ones:
>
> - Where is the team now? (…)
> - Where is it supposed to be getting to? (…)
> - How does the team currently intend to continue?

(Whitehead, page 17)

Another highly pragmatic resource was Joel Spolsky, and his "Joel Test":

> The neat thing about The Joel Test is that it's easy to get a quick yes or no to each question. You don't have to figure out lines-of-code-per-day or average-bugs-per-inflection-point. Give your team 1 point for each "yes" answer.(…)

> A score of 12 is perfect, 11 is tolerable, but 10 or lower and you've got serious problems. The truth is that most software organizations are running with a score of 2 or 3, and they need serious help, because companies like Microsoft run at 12 full-time.

(Spolsky, 2000)

The "Joel Test" result for this team was 2 when I joined the team (they just had source control and good tools). When I left the company, they were running at 9 (we just did not have candidates writing code during interviews, nor testers, nor hallway usability testing).

For this project I took the following decisions:

- Since the priority for the client was to see results, I asked the developers to concentrate on stabilizing "visible" features, particularly on a visual report editor, that used a complex set of controls, similar to those of a drawing application, to create reports. Doing this, we could have a stabilized preview version that we showed to the client as early as one week after my arrival to the project.
- In agreement with the developers, we set up a daily build procedure, and I also asked them to provide a "client build" every Wednesday, that would be placed in a public directory available to the client. It turns out that the client never downloaded the binaries, but they liked to see the version numbers grow, and the binaries being delivered. Every week, Wednesday was the "public build" day, Thursday was the "bug correction" day, and Friday, Monday and Tuesday were "new features day". Small stand-up meetings every day allowed us to know what was going on.
- Another important concern from the developers' side was to have a quiet environment to work. They were constantly interrupted by the (quite nervous) managers to see their progress, and as such, I decided to stand in between both; I asked them to not to interrupt the developers for any reason, and to ask me for updates. I became a "proxy" between both, which reduced the tensions, and brought some peace to the developers.
- I created a fast project plan in our Intranet (there wasn't any, so tracking the project was next to impossible) by asking the developers about the tasks they needed to do to finish the project, with the estimated time to do them, and setting some milestones. Since the project was in a wiki page, the developers could change the time estimations in case that they felt they had made a mistake; the only condition being to notify me of these changes.
- Using that information, I could create a couple of reports for everyone to see, and bring more visibility to the project:
    1. I wrote a weekly report stating the week's achievements, the status of the project (number of open bugs, new functionality available, etc).
    2. In the intranet, I set up a couple of graphs and report tables, which were automatically updated every day.

- I did not take any technical decisions about the project; I gave full authority on this matter to the lead developer, who in turn appreciated this trust and took spontaneously the decision of documenting and unit testing the system thoroughly doing extra hours every day. This boosted the morale of the team, and the quality of the application as well. The other two developers contributed to these tasks as well, and the rhythm of releases and their quality increased in a couple of months. It turns out that the architecture of the system was particularly well done, and as such, adding new features was a relatively simple task, once the underlying framework was done; of course, during that time no visible results were available, which made everyone nervous.

Looking backwards, the only technical decision I've needed to take during this project was to use the company wiki; there I could add information pages that everyone contributed to, reducing the number of communication channels and reducing the misunderstandings between project team members. I cannot stress how much this helped; it provided a complete dashboard for everyone to refer to.

The most important problems in this project were human and customer related ones. By providing more visibility to the project, and by reducing the signal-to-noise ratio in the communication channels between developers and management, the team was able to provide the customer with a more reliable and full-featured product.

References

Joel Spolsky, "The Joel Test: 12 Steps for Better Code", Wednesday, August 9th, 2000 [Internet] http://www.joelonsoftware.com/articles/fog0000000043.html (Accessed June 8th, 2007)

Whitehead, R.; "Leading a Software Development Team - A Developer's Guide to Successfully Leading People & Projects", Addison-Wesley, 2001, ISBN 0-201-67526-9