TO CONTAIN COST, LET'S NOT OVER BUILD OUR SOFTWARE SOLUTIONS

Jie Liu
Department of Computer Science, Western Oregon University, Monmouth, OR, U.S.A.

Keywords: Project management, Software engineering, Cost estimations, System requirements, Quality control.

Abstract: In the software industry, many deployed projects suffered one or more of the following: they had fewer features than planned, they were late on their deployment, or they were over budget. We participated in a project that suffered all of these. More significantly, it overran the budget by at least 400%. Looking back, many wrong decisions were made, such as misjudged users' expectations and their environments, subscribed over complicated backend architecture, and selected a different programming language that was unable to reuse existing code, etc. In this paper, serving as a case study, we argue that an effective approach to contain the cost of a software project, especially internal software, is to build a system that answer the core requirements with room for improvement, not to build the best system in the market.

1 INTRODUCTION

Many software projects need to cross the bridge of "requiring additional funding to continue." Most of the times, the added costs were justified assuming the software under developed is what the clients want and need. However, this assumption may not always be true for many reasons. Rather than discussing why this assumption may not be true, this paper uses a multi-million dollar project as a case study to identify a few areas where the difference between what the clients really need and what we are building may be the main reason for the increased cost.

We participated in a project to replace an existing one that was still in use and functioning. However, this is not a typical "next generation" project. The new software has more professional GUI and is capable of supporting, estimated, 100 times more users than the one it replaced. The problems are (1) the users are all internal (so more professional GUI is nice but is not necessary to increase the productivity) and (2) the number of users remains roughly the same (so the added capacity is not necessary and can never be utilized).

A proposal of building a replacing system with a few added features over the functioning system was only prices at $120,000 and was reject for business reasons. The initial budget of the replacement software project was budgeted at $350,000. The final product was built with an estimated cost of $2,000,000. Clearly, the cost of the project was unnecessarily high.

In this paper, we will discuss our analysis by sharing some details of the project and listing several major factors and decisions that, we believe, had contributed to the project's high price tag. We surely hope that our readers can benefit from the lessons we have learned. We are in a unique position to offer this analysis because we actively participated in the development of both the replacing and the replaced systems.

2 THE PROJECT HISTORY

Microsoft Excel is a common tool used by many to perform calculations involving a lot of numbers. However, it lacks the capability of tracing and debugging. When numbers do not reconcile or when accuracy is questioned, it is very hard to isolate the problems or to verify the results.

This happened to our client. After a few incidents, our client realized that they needed a software solution to help them collecting necessary inputs and propagating them to all the calculations, automatically performing complex calculations, providing certain level of tracking ability, and generating a list of predefine reports. The system was an internal one with only about 200

Liu J..
TO CONTAIN COST, LET'S NOT OVER BUILD OUR SOFTWARE SOLUTIONS.
DOI: 10.5220/0003562102690272
In *Proceedings of the 6th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE-2011), pages 269-272
ISBN: 978-989-8425-57-7

geographically distributed users to support highly cyclical activities every three months. Purchasing such a tool is not possible because most calculations are proprietary and may need to be calibrated on the fly. We were fortunately selected to be the solution provided.

After several discussions with the clients, it became obvious that they needed a web-based system. We then built the first version, functional but not flashy. The entire project took less than four person-months with a total cost of $25,000. The development process was a classical evolutionary prototyping one. The system went into production on 1/3/2000, the first working day of the Y2K, and survived the first forecasting cycle without any major glitch and users like it.

In the three years after our initial deployment, several enhancements were made including a stronger reporting component and performance improvement. Gradually, the software became one of the essential tools of the division's operations, and many decisions were made based on its reports.

Because the original developers were the only ones knew the system well enough to provide effective support and enhancement, attempts were made to train others to provide the necessary support and to add new features. The attempts failed because identifying engineers with the similar skill set as our original developers was difficult. This worried the upper level managers. Consequently, the lack of the ability of effective supporting the tool was identified as a business risk. Finally, at the juncture of adding a few new features, managers decided to enlist their company's own IT team to build a replacement.

We submitted a proposal and scoped the project at 12 person-months. Our proposal was rejected because it would not resolve the initially identified business risk. However, we were asked to join the new project team mostly due to end users strong request. The final product, with many requested new features moved to later releases, was deployed with a procurement costs exceeded one million dollars under a new PM and took close to two years to complete. If we add the costs of internal personnel and other overhead, the unaudited estimation of overall cost easily exceeded two millions US dollars.

The rest of the paper lists factors, contributed to the high cost of the project and other lessons we have learned through the execution of this project.

3 PROJECT RETROSPECTIVES

Since we are not be able to conduct any scientific experiments by repeating the process with one or few changes, we have to claim that the following are only our opinions and may or may not reflect the opinions of the project managers, our client, project sponsors, or other project team members.

3.1 Lesson 1: Not Every System Needs to be Enterprise Level Software

Our PM had a solid back ground in building enterprise level software for large companies. It should not be a surprise that he announced during our first project meeting that we would be building an enterprise level software solution. The only issue was that the system only had couple of hundreds internal users, and every one would access the application using Microsoft's Internet Explorer. We believe that this should be capitalized as an opportunity to cut development costs. In addition, the system was not a mission critical one.

This "enterprise level software" mentality greatly contributed to the decisions followed, especially the using of extremely sophisticated backend infrastructure. Developing software on the complex platform increased the complexity of the software, resulting in higher costs in design, development, testing, and support.

3.2 Lesson 2: Do Not Add Features that Will Never be Used

With only a couple hundreds of internal users and no addition users in sight, a piece of software should not be built as if it would support hundred concurrent external users accessing the software under diverse operating environments, which is what many software engineers had in mind regarding enterprise level software systems.

Because the users would all be internal, we had several advantages. First, they are more forgiven. In our case, this translates to that we had a great deal of freedom to plan our down time, and our GUI budget could take a little break. Second, users' operating environment was prescribed by our IT department and was extremely uniform -- every user access to our system through IE 6.0. Third, the users were readily available to define the correct behaviour of the system, which calls for development processes that could benefit from the opportunity.

Still, despite many hours of meeting and discussions, the architecture team selected Java as the programming language for our project, ORACLE as the DBMS, and WebLogic as the application server. The main reason given for selecting Java was that it was the best language

supporting platform independence. In our situation, this was a feature not in the requirement and would never be utilized because we require our users to use IE 6.0 or later. Clearly, this decision was heavily influenced by the PM's "enterprise level software" mentality without closely examining the real needs.

3.3 Lesson 3: Development Tools Matter

We believe that the selection of Java increased the duration of our development cycle in several ways. First, none of the business logic related code from the previous version, which was in C#, could be reused without converting into a different programming language; doing so required a much deeper understanding for the algorithms and resulted in expanses on a large number of avoidable tests. Second, many of the component that came with Visual Studio .NET 2.0, such as the data grid, had to be re-implemented using Java. Third, all the above not only extended the development time, but also increased the time necessary for testing and debugging. The decision of not using component based software development approach future contributed to the project's cost (Microsoft, 2002).

Heard our questioning about the PM and architecture team, our clients questioned IT's decisions about using Java and complex backend infrastructure. Interestingly, the end users were told to let the IT team to make IT related decisions. It seems to us that only internal IT team would use this line of reasoning. Let's say we were building a house, would the decision on what grade of lumbers to use be a decision solely made by the builder?

We are not here to promote one stack over another. We have participated successful projects where Java were used. What we want to emphasize is that programming language selection can affect the backend infrastructure and programming tools. As a result, this decision will affect the final project cost and is not as simple as just a "preference issue."

3.4 Cheaper Rate ≠ Lower Overall Cost

The backend infrastructure decisions were made without considering existing engineers' skill set. The next logic solution was to outsource. The company procurement division forwarded two software companies: Company X, a CMMI level three certified, and Company Y that was not certified and was now to us. Our PM selected the Company Y because its initial quote was much lower than Company X. Most importantly, Company Y accepted most of our conditions. One of them stated that we would contract a third company to perform quality control at all levels, including conducting some unit tests. The interesting observations were (1) if we did not trust Company Y to generate solid code, hence the necessity of third party quality control, why would we hire it, and (2) if we were to outsource our development, why not select a programming language already proven to be suitable by the previous version of the system?

3.4.1 Quality Control Should be Part of Software Development

It is clear now that this approach of separating software development with quality control did not work well, especially in terms of costs. The reasons are simple. First, high quality software hardly ever results from testing.

Second, the QA company and the software vendor have very different agendas. The QA company wants to conduct more tests, finding more defects (not necessary significant), and has an overall goal of spending more time testing the software so it could realize more revenue. On the other hand, the software vendor was constantly bombarded with defects not necessarily significant. So, the two companies spent a lot of time in meetings discussing and reclassifying defects. At the same time, our client is paying parties on their hours.

Third, the QA company engineers, as superior as they were, did not have a good understand regarding the end users' needs and preference, nor did they understand the code provided by the software engineers. So, they cannot create elegant test cases. As a result, they can only come up with some simple tests and did not have the ability to interpret the testing results without the help of end users or software engineers. This hindered the effectiveness of their tests and generated a great deal of extra work to both end users and software engineers resulting in increased costs in several areas.

At the end, the cost for QA costs counted for about one-third of the total procurement expenses, partially because the QA company is located in North America and commanded a higher rate. Did the QA company positively contribute to the quality of the final system? I believe it did. However, we recommend project managers search for alternatives before commit to this approach, especially when the system specifications are not detailed enough.

The bottom line is that if you outsource, then select a company you can trust that will do a good job, even it may cost you more hourly.

3.4.2 Cannot Just Follow Some High Level Specifications to Build Software

We all know the importance of SRS (Software Requirement Specifications). Generally, the bigger a system is, the more difficult to produce a SRS that is complete and accurate (Laird, L. and Brennan, C. 2006)

Company Y sent its PM produced the SRS by combining several initial documents such as RFP, studying the existing system, and interviewing end users. Clearly, the SRS only collected key use cases. This resulted in a great deal of communication efforts to fill in the details for the developers. Initially, we relied on emails and weekly telephone meetings. However, due to time zone difference, often some simple questions would take more than a working day to be answered.

The lesson learned here is that we cannot expect a software vendor to build a piece of software through just a high level SRS. A very detailed SRS is essential when project needs to outsource. Without it, any quote on cost estimation is likely a much lower figure than the final one.

3.4.3 Leadership can be Very Important

Soon after the budget increased close to a million dollars, the upper management replaced our PM. The new PM made many changes to quickly move the project forward. The most important change was that she listened to her engineers' and trusted them in her decisions. She also was willing to work with the end users to answer their concerns truthfully.

The system was eventually successfully deployed with fewer features than what was originally requested, six months late, and cost way more than initially budgeted. The clients commented that they would surely not have started the project had they known its final cost.

3.5 K.I.S.S

When developing a software system, we have to consider the supporting costs. In this case, the client had to outsource the support because the complexity of the backend. Its engineer who managed the outsourced supporting teams suggested to rewrite the system again using more suitable server architecture, namely SQL Server 2005 and C#, because the supporting costs were too high. His estimation on the effort was 6-12 person months. Unwilling to spend more money on the project, this re-write request was rejected. After a few more forecasting cycles, the client decommissioned the software due to its high supporting costs. Lesson learned: if some simple architecture and tools can meet the requirements, use them.

4 CONCLUSIONS

We have discussed many factors that may have negatively affected a project's costs. We agree that, there will be cases some of the costs are necessary. We argue that we need to evaluate the necessarily and benefit before commit on features or tools worked with other projects. In our case, the lessons were learned from real costly mistakes. We do hope that readers can benefit from our mistakes.

There are many reasons caused the PMs to request additional funding. Expecting initial funding to be accurate is unrealistic. However, when the budget has increased several folds with no major change in requirements, project owners should look into the few factors we have discussed here, and possibly others, to see if the similar mistakes have been made and to take necessary corrective actions. We are certain that there are other factors that can negatively affect a project's cost.

ACKNOWLEDGEMENTS

We would like to thank the Division of Computer Science at Western Oregon University, especially Dr. Marsaglia and Dr. Morse, and Oregon Engineering and Technology Industry Council for their continuous support of our research.

REFERENCES

Microsoft. (2002). Using .NET to Implement Sun Microsystems' Java Pet Store J2EE Blue Print Application http://msdn.microsoft.com/en-us/library/ms954626.aspx.

Lewis, W. (2004). Software Testing and Continuous Quality Improvement (2nd ed.). Auerbach Publications

Laird, L. and Brennan, C. (2006). Software Measurement and Estimation: A Practical Approach. Wiley-IEEE Computer Society Press.

Liu, J. and He, J. (2002). Web-Based Software Development for Today and Tomorrow. Proceedings of International Conference of Internet Computing.

Liu, J., Marsaglia, J and Olson, D. (2002). Preparing Software Engineering Students To Be Successful In The Real World. Proceedings of International Conference on Software Engineering Research and Practice.