Reusing an open source application – practical experiences with a mobile CRM pilot

Jyrki Akkanen[1], Hunor Demeter[2], Tamás Eppel[3], Zoltán Iványfi[2], Jukka K. Nurminen[1], Petri Stenman[4]
1 Nokia Research Center, P.O. Box 407, 00045 NOKIA GROUP, Finland
jyrki.akkanen@nokia.com, jukka.k.nurminen@nokia.com
2 Nokia Research Center, P.O.Box 392, H-1461 BUDAPEST, Hungary
hunor.demeter@nokia.com, zoltan.ivanfi@nokia.com
3 Budapest University of Technology and Economics, Budapest Pf. 91, H-1521 Hungary; peletomi@gmail.com
4 Nokia Ventures Organization, P.O. Box 407, 00045 NOKIA GROUP, Finland; petri.stenman@nokia.com

Abstract. We discuss experiences in extending an open source CRM application to develop a new server-based mobile business application. Combining the application code reuse with incremental development process allowed successful development of a pilot application in a tight schedule. In particular, it enabled a quick start for customer-driven development, diminished risks related to the baseline application itself, and provided the flexibility needed in experimental pilot development.

1 Introduction

New software is often developed over a platform. In many senses platforms are ideal for developing new applications: they are usually well-documented and designed for reuse and easy development. However, in this paper we explore a different approach: we discuss the benefits and challenges associated to reusing an existing open source application in order to create a new product. This approach has potential since it allows a quick start and, when combined to an agile process, also an early user involvement. Furthermore, using an open source product as the development base reduces risks due to associated information openness.

Our study grew up from practical experiences. Our vision was to create a business software solution for small and medium-sized companies, a solution that could support both multiple business domains and mobility. To begin with we implemented a pilot system for the Finnish real estate agencies by extending an open source CRM application to a new business domain and by integrating mobile phones in it. The solution was a new concept and included a lot of uncertainty both on the technical and business side. To deal with the uncertainty the venturing literature suggests testing the underlying assumptions at the lowest possible cost (see e.g. [1]). So we were not implementing an application fulfilling a fixed specification but in a

very dynamic fashion exploring for the right combination of business and technology to create profitable business.

2. Course of the development project

A lot has been written about motivations to participate in open source development; both from individual and corporate viewpoint [2]. One of the frequently cited benefits is that reusing open source reduces development cost [3,4]. This was also our main motivation: using an open source business application as development base would save development effort and thus allow us to rapidly test our concept and assumptions with a real case.

On the Internet we found roughly 40 open source CRM and group work suites. With a set of criteria (e.g. functionality, quality, license, environment) we finally selected vTiger [5] as our base application. Most solutions were dropped due to lacking functionality or immaturity, and none of them fully satisfied all the criteria. With vTiger we were mostly concerned about the product quality and maturity. It is a web-based business management software written in PHP and running over Apache web server and mySQL database. The product is developed mainly by an Indian company vTiger, which is getting revenue from support and subscription services. Our baseline version was vTiger 4.2.

We used a very light-weight and incremental development process along the common open source guideline "release early, release often" [3]. Our initially rather vague understanding of customer requirements grew during the development, and we also became more and more aware of our abilities to fulfill them. It was only after two months, in the middle of the project, when we finally were able to estimate how much functionality we actually can deliver, and were able to even roughly pin down the final feature set for the deliverable:

1. Basic CRM functionality: contacts, messages, phone calls, notes etc. that could be cross-linked to each other.
2. Real estate business specific functionality: property management and search, bidding process, checklists, brochure creation.
3. Mobility support: synchronize CRM data (e.g. contacts) with phone, rich context to phone calls.
4. PC-phone interoperability: dock the phone to PC, respond to incoming calls on the PC with proper CRM context, and initiate calls by clicking special hypertext link in PC browser.

As depicted in Figure 1, all the business functionality was implemented in an Application Server that provided a web UI for a browser. That consisted mostly of vTiger CRM augmented with Realtor Extensions, the set of specific modules for real estate business. The mobile phones connected to the Application Server through a proprietary Mobility Server that took care of all the mobility-related issues. A small adaptive module, Mobility Connector, in the Application Server provided a Web

Services interface for the Mobility Server. This design made the mobility framework independent from the Application Server.

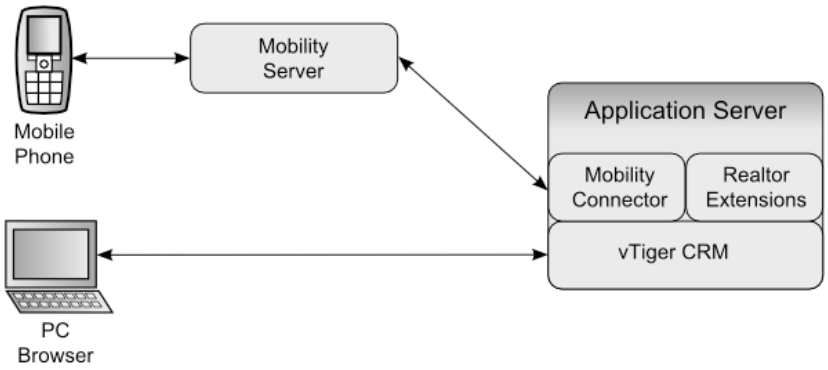

Fig. 1. The solution architecture

We managed to deliver the pilot to our customers according to the original schedule and they were satisfied with the quality of the deliverable. There was no need to fix bugs during the 2 month pilot testing period. The table 1 below sums up the code size in the Application Server divided into different functions.

Table 1. Deliverable size: lines of code including empty lines and comments

	Base	New	Modified	Release
vTiger CRM	361 487	8 000	7 915	373 838
Realtor Extension		12 080		12 080
Mobility Connector		2 979		2 979
Third-party Libraries	163 020	902	380	164 266
Total	524 507	23 961	8 295	553 163

The Base column shows the size of the original vTiger 4.2. The New column contains the lines in the files that we created ourselves while the Modified column tells how many lines in the Base code files were modified. The Release column contains the final size of the deliverable.

The figures show that reuse was very efficient: only 6% of all the code lines in the deliverable were written by us. Furthermore, we used only about 15 man-months to enhance the vTiger CRM into our Application Server, obviously an order of magnitude less than what would have been required for developing everything.

Of course a considerable effort would still have been required to finalize our prototype into a product. In addition, the fact that the vTiger modifications spread over 236 files complicates future maintenance. For instance, merging the changes to a newer version of vTiger would be harder.

Despite the successful pilot we eventually decided to discontinue developing the Application Server. The reasons were mostly related to business: as our revenue would come from mobility support it might be better to rely on existing business applications rather than develop new ones. We thus re-focused our resources in developing the mobility support into such form that creating the necessary Mobility Connector modules for existing business applications would be as easy as possible.

3. Lessons learned

Compared to mature application platforms complete applications have a harder learning curve, especially if they are weakly documented and do not have a clean architecture. Our experience showed that the benefits of complete applications easily outweigh their weaknesses: we got a lot of complete functionality for free. In our case the base application already provided most of the necessary software infrastructure and one third of necessary features. Furthermore, we could develop new functionality on top of the old one with relatively small effort.

Basing our development on an existing application suited well to incremental customer-driven process. Customers easily caught a definitive role in the development process as the lead users could try out working software from the first day of development. We could also quickly provide new functionality: this kept customers active and motivated to provide domain knowledge and feedback. Active weekly customer feedback was crucial to maintain the development speed and right focus. In the software engineer viewpoint, incremental prototyping felt as well the best approach to get familiar with the base application: the source code was readily available, but we did not know it at all and it did not have too much documentation. Making small modifications was a natural way to learn the software piece by piece.

Maybe we could even state that reusing an existing application is an optimal approach only when combined to an incremental process. A careful and deep design phase before implementation requires learning both the existing software and requirements. This is time-consuming and, as a result, one loses the quick start. In worst case one ends up re-designing large parts of the base software in such a way that writing completely new software turns out to be an easier solution.

Incremental prototyping approach is, however, not a silver bullet: it may lead to architectural problems. The architectural decisions in the base software concern also the new features but, in practice, deviations from clean architecture easily occur. In the beginning one simply does not know the software enough to follow its architecture. It is also difficult to control the overall architecture if new features are inserted one after another with least possible effort. In the end some architectural refactoring may need to take place.

A further benefit of open source is that not only the source code, but all other data concerning the software is public and accessible. For us the open information was useful for the software quality. The open source advocates eagerly remind us from the elements in open source development model that promote quality [3] and based on experience with mature top-quality open source products like Linux and Apache, such claims are valid [6]. On the other hand, during our selection process, it became clear that all products were not mature enough for real production use. We encountered security vulnerabilities, weak usability, low performance, lots of dead code, and poor documentation.

Such weaknesses are also common in commercial non-mature software products. However, the benefit of open source development is that it is practically impossible to keep quality problems hidden. To look behind the hype on the front page you can install and try out the software. An experienced developer can recognize low-quality code by looking at the implementation. The discussions in various Internet boards also reveal if users are having problems with the software and indicate what the developer community attitude towards the possible weaknesses is.

The situation is totally different with commercial software. Information can only be accessed "in drops" and getting beyond the front page advertisements often requires commitment. This holds especially if you think about reusing the software in order to further develop it: commercial software is usually not sold or advertised for that purpose.

We were bad citizens in that respect that we did not contribute to an open source community at all. The basic reason for not doing so was that we were unsure about what to contribute, where and how. The realtor-specific features were very domain- and country-specific, so we did not expect them to have much generic interest. The mobility features would surely have been welcomed but the solution depended on proprietary components that could not be open sourced.

An additional difficulty in joining the community was that our own targets were changing as our experience with the business and technology grew. As we were not mature enough to clearly specify our needs we concluded that attempting co-operation in so early stage would just confuse the base product community.

Open source application reuse causes complicated licensing situation inside companies building commercial solutions. When you modify open source code, you end up intermingling your code with the open source code. Depending on the open source license this may lead to many concerns related to commercial usage of the software. You may also end up with problems with your IPR: this is very complicated matter and so difficult to analyze to even understand all the consequences [7]. In our case this led to situation where we had code without clear strategy how to use it later on.

4. Conclusions

The target of our project was quickly and with small resources to provide a pilot of a server-based business application, which supports realtors in their daily work and allows use through a mobile phone. Building on top of an existing open source application allowed us to successfully release the pilot in a tight schedule.

Reusing an open source application gave us a quick start for the development. As the base product already contained lots of functionality, we could concentrate on satisfying the actual needs of the user. Many risks related to the product could be easily managed since open source software came with open information: all relevant knowledge of the base product was easily available.

We also noticed the benefits of combining agile, incremental development process to reusing an existing application. Straight from the beginning of the project the customers were able to use the weekly releases. Their feedback of the features and their usability steered the project and allowed flexible prioritization of the implementation tasks.

On the other hand, for the pilot, we ignored some critical questions concerning the software licensing and our position in the open source community. These must be dealt with before basing a commercial product on an open source application.

References

1. McGrath, R.G and MacMillan, I., *The Entrepreneurial Mindset* (Harvard Business School Press, 2000).
2. Rossi, M., Decoding the 'Free/Open Source Software Puzzle': a survey of theoretical and empirical contributions (2004), in: The Economics of Open Source Software Development, edited by J. Bitzer and P.J.H. Schröder (Elsevier 2005); http://opensource.mit.edu/papers/rossi.pdf.
3. Raymond, E., *The Cathedral & the Bazaar* (O'Reilly, February 2001). See also http://www.catb.org/~esr/writings/cathedral-bazaar/.
4. Bessen, J.E., Open Source Software: Free Provision of Complex Public Goods (July 2005). Available at SSRN: http://ssrn.com/abstract=588763 or DOI: 10.2139/ssrn.588763 (http://dx.doi.org/10.2139/ssrn.588763).
5. vTiger CRM; http://www.vtiger.com/.
6. Boulanger, A. Open-source versus proprietary software: Is one more reliable and secure than the other? *IBM Systems Journal* 44:2 (June 2005), p.239.
7. Vetter, G.R. "Infectious" Open Source Software: Spreading Incentives or Promoting Resistance. Rutgers Law Journal 36:1 (Fall 2004), p.53; http://opensource.mit.edu/papers/vetter2.pdf.