JASDM: Just Another Software Development Methodology

Juan Diego Tascón

The most important thing that Widely [1] development leaves me is the experiencing, at first hand, of the current software development methodologies and to see the high degree of inmaturity, even on methodologies widely used like the proposed by IBM Rational, the RUP [2] (Rational Unified Process).

Based on my experience is humanly impossible to think on the full set of consequences that a fairly developed prior design can bring with sequence and colaboration diagrams and wisely to explain the inner workings of the application, obviously, putting on the table as a restriction that the design should be at least equal to the construction time, we see ourselfs in the painful task to say that our very cutes sequence diagrams are only a terrible waste of time, regardless the frecuent software diagrams synchless problems, very common in a product life cycle.

Due to a number of factors involved in the development process, the real design consequences are only measurable with acceptables heuristics when the real implementation of the software occurs, these factors are highly stochastics and leads to problems which are not had in mind in the design just because of our forgetful nature, they also suffer a mooring effect with our application desing, molding it and forcing it to have a closed approach to several features, these factors are:

- Programming language: Is possible to find a lot of programmming languages, actually there is almost 2500 [3], regardless the personal use languages and the very specific environments languages, besides this there are programming languages in all colors and flavors characterized by their skills, tools, support, paradigms, and the most important of all, the approach, that may change the solution in a radical way, and in some cases, makes the solution something impossible to reach, from this we conclude that the solution design of a problem varies according to the language characteristics.

- Tools: This concept can be better seen in the field of the web applications which have gone through a revolution oriented to the aids and facilities to the developer and have seeing the emerging of major tools called frameworks which automates process like: database connections, deployment (development, tests and production), unitary tests, project administration, etc, each framework creates dependencies and requirements of doing things in a certain way (I.E: in ruby on rails you must have a model-view-controller architecture) once again dramatically changing our application design.

- Available libraries: A library is just an implementation of a programming concept, with more and better available libraries for our language we can make things easier; a very simple example of this is the authentication system that usually requires the having of a cipher, when we don't have a good implementation of it, we may seek other alternatives like using a library from different language and this requires the making of a communication system between both languages or just to add the implementation of the cipher to the initial development increasing the software complexity; these changes are reflected in our design in one way or another.

- Associated Technologies: There is generally a lot of ways to do things in software programing, depending even in the era they are done, the clearest example is that 10 years ago most of the programing was made using a structured paradigm, today the standard is to do object oriented systems, and it's very likely that in the future, process concurrence based tools will be used, like the ones that follow agent or restriction paradigms, the reader now can imagine the conclution of this item.

Of course these assertions has their own limits because of simply launching into solution programming will bring several problems such as waste of work and closed and poorly extendable resulting software.

For the Widely development i proposed a metodology and it was followed faithfully, thanks to it i got very oportune and practical results. this is summarized in the declaration of an intermediate point where the problem and the espected results are described and where most of the work is focused in the real development of the application.

The design must be adaptable, easy to modify and it's detail limit must be up to the point where you can pick up any of the variant of the before mentioned factors without it being reflected in a design change. It's main characteristic is to diferentiate and describe, but not to detail the systems that make the application (I.E. user control system, information sending system, etc).

I proceed with the development of the application (programming) after this, where the implementation details were described not in colaboration or sequence diagrams but inside the own code, this implementation suffer a reingeneering process until i get a stable solution that satisfies

the espected results.

Having always in mind the reingeneering process that might begins because of requirements changes, platform changes or integration errors, by this reazon the design must be prepared for these changes to affect, as posible, the lowest number of subsystems.

The only diagrams made are the really necesary and usefuls, in which highlights two types: the division diagrams which are those that explain the way of separation of the application subsystems and the system requirements and the external diagrams which explain separated processes from the business logic but that are necessary for the proyect completion, for Widely i made the following diagrams:

- Division Diagrams:
  - Use Cases
  - Architecture Diagram
  - Conceptual Model
  - Packages Diagram

- External Diagrams:
  - Navigation Diagram
  - Interfece Design
  - File System Hierarchy

As additional point, the Widely webservices system, which is a little more tangible, was built following a test driven development (TDD [4]), what this means is the tests cases were made before building the systems in order to successful pass these test.

The proper application of the proposed methodology made possible for Widely to be finished in a timely and effective way having as main result a very well made, clear and third party continuable (because of the documents obtained and its free license the GPL [5]) application.

The final question to the reader is: this methodology could really works on differents developments? or the good results in Widely was just a coincidence, just the practice and the time will tell us.

References

[1] Juan Diego Tascón, Tesis de grado Widely: Sistema Web de Desarrollo Integrado, Available at: Biblioteca Universidad del Valle, 2007

[2] IBM Rational, Rationa Unified Process (RUP), Disponible en: http://www-306.ibm.com/software/awdtools/rup, 2007

[3] Bill Kinnersley, The Language List, Available at: http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm, 2007

[4] Wikipedia, Test Driven Development (TDD), Available at: http://en.wikipedia.org/wiki/Test-driven_development, 2007

[5] Free Software Foundation, GNU General Public License (GPL), Available at: http://www.gnu.org/licenses/gpl.html, 2007

[6] Creative Commons License, Available at: http://creativecommons.org/licenses/by-nc-sa/2.5/co/, 2007