International Journal of Soft Computing 10 (5): 336-339, 2015
ISSN: 1816-9503
© Medwell Journals, 2015

Approaches to Organization of the Software Development

P.R.A. Valiyev, L.A. Galiullin and A.N. Iliukhin
Branch of Kazan Federal University in Naberezhnye Chelny, 10A Sujumbike Avenue,
The City of Naberezhnye Chelny, The Republic of Tatarstan, Russian Federation

Abstract: During the last 40 years the software development industry has not become the engineering one. The main issue of the software development is the related risks. Great amount of the out-of-the-box software solutions accelerates the development process, however, generates a variety of solutions of the same tasks which in its turn increases the uncertainty and risks related to the software development. The main objective of work on a software design is reduction of risks; one of such approaches to organization of the software development is Extreme Programming (XP). In its turn, the common property of the software code also allows the programmers to quickly improve their skills. Besides, a common software code completely eliminates the probability of situation when one of the programmers leaves and takes a part of the code with him, killing the long-term and sometimes almost completed project. Pair work at the same computer, continuous testing, continuous design and code improvement (refactor), integration immediately after implementation of the new functionality all of these XP techniques are aimed at achieving that both the design and the software code itself is easily modifiable at any point of the project life-time both at the stage of the primary system design and years after the software commercialization. Simple design and commonly accepted coding standards will allow the new team members to puzzle out the project quickly. And continuously developing and supported unit tests will ensure secure operational performance.

Key words: Programming languages, software, design, development, information architecture, information systems, programming methods (techniques), extreme programming, pair programming, refactor, risk mitigation

INTRODUCTION

The software design industry that appeared about 40 years ago due to standardization of the communication protocols between the software products gradually breaks through the original chaos, continuing booming. The high-level programming languages, data-description and management aids, integrated software environments are created and evolving; the design workbench becomes more and more sophisticated. Hundreds of thousands of specialists all over the world for the international software market daily (Abdullah and Abdelsatir, 2013).

However, during the last 40 years the software development industry has not become the engineering one. Upon availability of the developed methods of organization of the software design (RUP, XP, etc.) and fundamental approaches to arrangement of the software itself (object-oriented, procedure-oriented programming, etc.) there are still no exact formalized coding procedures for particular kinds of tasks. A huge number of various components and out-of-the-box software solutions significantly accelerates the development process, however, generates a variety of solutions of the same tasks which in its turn increases the uncertainty and risks related to the software development (Wood *et al*., 2013).

RISKS OF THE SOFTWARE DESIGN

The main objective of work on a software design is reduction of the related risks. These are the main of them:

- Failure to comply with the terms the date of the software delivery comes and it will be necessary to tell the customer that the program will not be ready for another 6 months
- The project is closed after multiple delays the project is cancelled without being completed
- System clogs the software is being successfully operated, however, after a few years the cost of the system modifications or number of errors causes the necessity to change the program
- The error level the project is completed, however, the error level is that high that software can not be used in practice

Corresponding Author: P.R.A. Valiyev, Branch of Kazan Federal University in Naberezhnye Chelny, 10A Sujumbike Avenue, The City of Naberezhnye Chelny, The Republic of Tatarstan, Russian Federation

- Lack of understanding of objectives the project is completed however, it does not solve the tasks set
- Changes in the objectives the project is completed however the issue for solution of which it was designed six months ago is substituted through another, more topical issue
- Achievement of false goals the software product features a variety of potentially interesting options that could be efficiently implemented however neither of them nor the combination thereof will not provide much benefit by solving the task set
- The staff left after 2 years of development all skilled programmers started hating the software being designed and left

RISKS OF THE COMMERCIAL SOFTWARE DESIGN

By considering the risks of design of commercial software for the purposes of the engine testing automation one should take into consideration both the state in the software design industry and the specifics of the process itself.

In the software design industry over the recent years a significant misbalance is observed the demand for highly-skilled specialists by many times exceeds the offers available, at the same time, the number of the poorly qualified specialists exceeds the demand and with each year this misbalance becomes more and more strained (Putra *et al*., 2012). Speaking of automation one should mention the tendency of the last year to increase in the remuneration of labor of young specialists in the scientific area which may have a positive effect on the software design for commercial purposes.

However, there are no grounds for supposing that situation will change that fundamentally to attract the young highly-skilled staff to the industry. The gap between the remuneration of labor of specialist in the industrial area and let's say in trading or banking one (which in their turn are far from being the most well-paid areas of the software design) is so significant that even the outlined positive trends will hardly result in the substantial growth of interest in such activity even on the part of the poorly qualified specialists. On the other hand, the market flooding with poorly qualified specialists creates pre-requisites for graduates to come to the industry for the purpose of gaining experience, practical skills, improving their qualification to leave further on for working in other more attractive companies.

On the basis of analysis of the situation certain conclusions may be drawn as to the risks of the software design in the industrial area. Engaging as contractors mainly poorly qualified specialists one may state for certain that nearly all risks relating to the software design are increased. Taking into account the high staff turnover one may draw a conclusion that the risk of the 'staff leaving' is increased significantly with all the consequences arising from it and namely:

- Delay in the project turnover
- Long period of adaptation of the new employees resulting in more delays
- Intricate, unclear design of the software systems
- Complete 'death' of the project when the specialist gone has not left the software source codes or left the codes that no one can puzzle out

Examples show that the 'death' of the project in case of leaving of one particular specialist despite the seeming absurdity of the situation is more than real and such a situation occurs quite frequently.

APPLICATION OF XP TECHNIQUES AS A METHOD OF ROSK MITIGATION

Of course, one of the main goals of any project management is first of all, the risk mitigation. In case, when the risks are high (and by design of the software for the purposes of the industrial sector they are especially high) the project management and organization of works upon the project shall be assigned the special priority. One of the methods of such organization of works on the software aimed at mitigating most of the risks is extreme programming.

To some people the principles of extreme programming resemble the common sense. XP is based on the principles of common sense and applies them to the extreme extent (Sohaib and Khan, 2011). The examples of such common sense are:

- If the code reviews are good then we will review the code all the time (pair programming)
- If testing is good then each will test all the time (unit testing), even the costumers (functional testing)
- if design is good then we will make it to be a part of the daily routine (refactor)
- If simplicity is good then we will leave the system with the simplest design that would support all functional capabilities (the simplest thing that is able to work)

- If architecture is important then each will work determining and improving the architecture all the time (metaphor)
- If the integration testing is useful then we will integrate and test few times a day (continuous integration)
- If short cycles are good then we will make the cycles to be minimal (seconds, minutes and hours and not weeks, months and years) (scheduling game)

XP provides two sets of promises:

- It promises to the XP programmers that each day they will work on the things that really matter. They will not need to face the unpredictable situations alone. They will be able to do everything possible to make their system to be successful. They will take decisions they are the best at and not vice versa
- XP promises to the customers and supervisors that they will get the maximum effects from each week of programming

Every few weeks they will be able to see certain progress and what is especially important they will be able to change the direction of the project in the middle of the progress without incurring enormous costs.

In short, XP promises to reduce the project risk to improve the feedback options in case of changes in the objectives, to improve the performance of the working process during the entire system lifetime and to increase interest in creation of the software within the teams-all of that at the same.

These are the main XP techniques:

- Scheduling game the scale of the next release is determined by combining the tasks set and engineering estimates. If the reality does not correspond to the plan, the plan is upgraded
- Small releases a simple system is created within a short time then the new versions are released within very short intervals
- Metaphor guides all the system design process by the simple plain description of how the system operates
- Simple design the system is designed in the simplest manner possible at any given moment of time. Extra complexity is removed as soon as it is detected
- Testing the programmers continuously code the unit tests that should run faultlessly to ensure that the development is continued. Customers compose tests demonstrating that functional features are completed
- Refactor the programmers restructure the system without changing its behavior in order to eliminate redundancy, improve communication, simplify or and enhance flexibility
- Pair programming the entire software code is written by two programmers at the same machine
- Common property anyone may modify the code in the system at any time
- Continuous integration as soon as any task is fulfilled it is integrated in the system immediately, integrations are performed many times a day
- The 40 h long working week not >40 h a week, as a rule
- Customer availability a real user is involved in the team that is able to answer the questions during the entire working week
- Coding standards for the purposes of improvement of communication between developers at the software code level the programmers write the entire code according to the accepted rules (Zhai *et al*., 2011)

Having summarized these techniques one may say about development in the XP style that:

The programmer pairs work together. The development is guided by continuous testing. First you test, then you code. Until all the tests are completed you can not consider the task to be fulfilled. When the tests are completed and you can not invent a test that could fail this means that you have completed adding functional options.

The programmer pairs do not only invent tests and make them work. They also continuously improve the system design. Any code can be reviewed and modified by any person in the team. Pair work increases the efficiency of analysis, design, implementation and testing of the system anywhere where it is required by the system.

Integration follows implementation immediately, including testing of integration itself. Pair work at the same computer, continuous testing, continuous design and code improvement (refactor), integration immediately after implementation of the new functionality all of these XP techniques are aimed at achieving that both the design and the software code itself is easily modifiable at any point of the project life-time both at the stage of the primary system design and years after the software commercialization.

SUMMARY

The methods described in this paper have been used by development of the application for an automated information system for diesel engine testing (Zubkov and Galiullin, 2011; Biktimirov *et al*., 2014).

CONCLUSION

In order to realize the potential benefit from application of the XP techniques the XP techniques and main factors characterizing the software design in the industry have been compared. Poor skills of contractors, high staff turnover rate, contractors' interest no so much in the successful completion of the project (for which they paid little) as in improvement of qualification and changing job for a well-paid one it is possible to manage all those negative factors by using the XP.

Thus for example, pair programming allows improving the contractors' qualification much faster than alone (Musa *et al*., 2011). It is better to hire two graduates and ask them to code a unit than trying to parallelize the work let the same graduates code the two different units in a parallel way. In the second case, the poor qualification of contractors will determine the high probability that either of units will be delayed or not completed at all which will bring into challenge execution of the entire project.

In its turn, the common property of the software code also allows the programmers to quickly improve their skills (Ji and Sedano, 2011). Besides, a common software code completely eliminates the probability of situation when one of the programmers leaves and takes a part of the code with him, killing the long-term and sometimes almost completed project. And significantly reduces the effects of the similar situation when one of the contractors leaves, the code is left but no one can understand what this code does and how it can be modified (Fojtik, 2011). Indeed, the common property of the code along with pair programming completely excludes the probability of such situation in XP the remaining team members will be able to handle the entire software cone which will allow them to quickly become clear of the segments of code designed primarily by another team member.

Simple design and commonly accepted coding standards will allow the new team members to puzzle out the project quickly. And continuously developing and supported unit tests will ensure secure operational sperformance.

REFERENCES

Abdullah, E. and E.T.B. Abdelsatir, 2013. Extreme programming applied in a large-scale distributed system. Proceedings 2013 International Conference on Computer, Electrical and Electronics Engineering: 'Research Makes a Difference', ICCEEE, Art. No., 6633979, pp: 442-446.

Biktimirov, R.L., R.A. Valiev, L.A. Galiullin, E.V. Zubkov, A.N. Iljuhin, 2014. Automated test system of diesel engines based on fuzzy neural network. Res. J. Appl. Sci., 9 (12): 1059-1063.

Fojtik, R., 2011. Extreme programming in development of specific software. Procedia Comput. Sci., 3: 1464-1468.

Ji, F. and T. Sedano, 2011. Comparing extreme programming and Waterfall project results. 2011 24th IEEE-CS Conference on Software Engineering Education and Training, CSEE and T, Proceedings, Art. No. 5876129, pp: 482-486.

Musa, S.B., N.M. Norwawi, M.H. Selamat and K.Y. Sharif, 2011. Improved extreme programming methodology with inbuilt security. ISCI 2011 IEEE Symposium on Computers and Informatics, Art. No. 5958997, pp: 674-679.

Putra, I.P.E.S., A. Yuliawati and P. Mursanto, 2012. Industrial extreme programming practice's implementation in rational unified process on agile development theme. 2012 International Conference on Advanced Computer Science and Information Systems, ICACSIS Proceedings, Art. No. 6468769, pp: 137-142.

Sohaib, O. and K. Khan, 2011. Incorporating discount usability in extreme programming. Intl. J. Software Eng. Applicat., 5 (1): 51-62.

Wood, S., G. Michaelides and C. Thomson, 2013. Successful extreme programming: Fidelity to the methodology or good teamworking?. Informat. Software Technol., 55 (4): 660-672.

Zhai, L.L., L.F. Hong and Q.Y. Sun, 2011. Research on requirement for high-quality model of Extreme Programming. Proceedings 2011 4th International Conference on Information Management, Innovation Management and Industrial Engineering, ICIII, 1, Art. No. 6115089, pp: 518-522.

Zubkov, E.V. and L.A. Galiullin, 2011. Hybrid neural network for the adjustment of fuzzy systems when simulating tests of internal combustion engines. Russian Eng. Res., 31 (5): 439-443.