Been a Long-Living Software Mayor — the SimCity Metaphor to Explain the Challenges behind Software Evolution

Submitted to the CHASE 2005 Workshop on Software Evolution

Stéphane Ducasse, Tudor Gîrba

Software Composition Group - University of Berne - Switzerland

Abstract

When explaining the true dynamics of legacy systems, we are often short of good metaphors that provoke on the audience the correct understanding of the situation. While war stories are entertaining and fun, the audience often summarizing them as that the original developers were not good and that they would not do the same mistakes. Sometimes managers miss the point that a software system is a living organism for which caretakers are important and that such a system can simply die. We propose to use the SimCity game as a metaphor to explain the challenges of maintaining or making a system evolves.

Keywords: software evolution, metaphor

1 Introduction

Once upon a time, there was a successful project that got sixty percent of the world-market. For the american market, clients wanted to stack processors from 12 to 200 to get "speed increase". This could have been possible if the architecture of the system would have been document, flexible, covered with tests, if the original developers were available and . . . As a normal successful project, the software was not documented, not covered by tests, following an old and obscure architecture and nearly all the developers left the company since salaries were better on the other side of the border. Still there was some hope. One developer resisted against dark forces. This bold knight of the new century was ready to slice bugs with his refactoring axe even when he was surrounded. Full of grace, he asked his manager if they could hire a couple experimented code fighters to "clean the system". "But, we do not need to clean the system! This a highly successful system and we have no problem with it. This is our system!" was the answer of the expert and wise manager . . .

In software development, it happens that decision making people do not get the reality of software development in the sense that software is a *living* organism. Often-times even the developers believe that they will not redo the same mistakes other did because they are much smarter and they use the brand new caffeine-based languages and cutting edge wonder technologies. Still reality have proven that them wrong: new languages, new standards and frameworks, too much up front design, not getting client point of view do not prevent to fail. If the system is not ready to change in unexpected ways, changes will happen inevitably. This is why the Figure 1 is a good appertizer for presenting the problems related to changes.

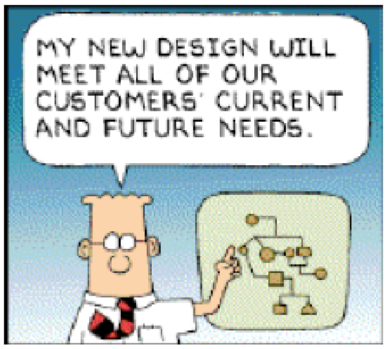
MY NEW DESIGN WILL MEET ALL OF OUR CUSTOMERS' CURRENT AND FUTURE NEEDS.

Figure 1. Building software in wonder land.

The end of our story was lost in the dust of the future past and we do not know whether our brave knight of the new century manage to convince the manager or not. Yet, we like happy endings. And as we were knights in a previous life, we decided to come to the rescue of our brave knight and endow him with the solution that leads to a happy end.

Our solution is to arm the developer with the metaphor of SimCity, so that he can explain his manager his problems.

2 SimCity

The goal of SimCity is to be a long-living mayor of a prosperous city. To be reelected the mayor should take care of polls and the citizens can vote and express their opinions

Figure 2. SimCity: Simulating and controlling the evolution of a city

concerning their environment. Multiple factors come into play such as the balance between production area and living places, social facilities, infrastructure setup ... In SimCity is difficult to pay attention to all the aspects upfront, that is what you originally planned does not always work the way you thought: a nice city full of small houses and gardens can get deserted if you do not have factories; a successful city attracting too many people at once can turn rapidly into a nightmare and shantytowns.

The laws of software evolution have stressed that evolution is inevitable [2]. Sotware must change or become obsolete. It should adapt to new requirements to survive. And the same holds for SimCity.

2.1 Continuous Changes

SimCity is a large dynamic systems that forces the mayor to take corrective actions, foresee the evolution of the situation, analyze the problems and find fast remedies. Sometimes entire parts of the city should be changed: bridges need to be built, industry needs to be moved, quarters need to be rearranged etc. Not changing the city leads inevitably to problems.

Similarly, in software, not analyzing the evolution, looking as symptoms clearly indicates that the state of the system is in danger. Here is a list of such a symptoms classified in three groups [1].

Lack of knowledge

- obsolete or no documentation,
- departure of the original developers or users,
- disappearance of inside knowledge about the system,
- limited understanding of entire system.

Code Smells

- duplicated code,
- code smells,
- large complex controlling classes,
- complex conditional logic [**?**].

Process Symptoms

- difficulty to get simple things done,
- difficulty to understand why problem occurs,
- difficulty of communication between subteam,
- large build time.

Not taking the time to analyze the trends and refactoring or cleaning the system leads to fragile systems that cannot change to adapt to new requirements. Rethinking and refactoring the system after an expansion phase leads to brittle and fat systems.

2.2 Complex system with multiple stakeholders

As in software development, when building a city different stakeholders should be considered. For example, a mayor should satisfy the citizens need of a clean environment while at the same time attracting the industry to sustain the economical value of the city.

2.3 Past solutions were suitable for the past situations

Oftentimes developers think too easily that original design was simply wrong, that previous developers were not smart enough. While this may be true, it is much more probable that the original developers took the right decisions within their context at that time. However, the fact that new requirements are coming into play may be inadequate with the old architecture. SimCity really stresses this aspect by its intrinsic dynamic nature.

2.4 Growth speed

Building a city too fast leads to sure failure. The same goes for software development. Investing too much in the upfront grand vision is often heavily challenged by the fast changing environments. One should not grow faster than the system can assimilate.

3 Conclusion

The urge of taking the evolution of a system into account is often misunderstood. The SimCity Metaphor applied to software evolution is one way to present this urge and to help reengineers and maintainers of valuable legacy systems to communicate their problems and the importance of evolution in complex dynamic systems.

Afterwords

Once upon a time there was this bold knight of the new century armed with his refactoring axe ready to fight the bugs from the obscure structure of a highly successful system. He knew his fight and he knew he needed help. So, he faced his manager and ...

“Do you like SimCity?” asked the bold knight.

“Who does not,” asked the intrigued manager.

“Then you know that if you want a long-living system, you should always change it, you should always update the infrastructure, as the condition change.”

“Of course I do! What, do you think I am stupid?”

“Then I tell you: a software system is like a city of sims.”

...and everyone lived happily ever after.

Acknowledgments. We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project Recast: Evolution of Object-Oriented Applications (SNF 2000-061655.00/1).

References

[1] S. Demeyer, S. Ducasse, and O. Nierstrasz. *Object-Oriented Reengineering Patterns*. Morgan Kaufmann, 2002.

[2] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Addison Wesley, 1999.

[3] M. M. Lehman and L. Belady. *Program Evolution – Processes of Software Change*. London Academic Press, 1985.