Continuous Care Vs. Initial Design

Robert C. Martin
13 February 2002

Abstract

Although the initial quality of a software design is important, software designs suffer more from lack of continuous care than they do from poor initial design. Lack of continuous care allows a good design to degrade over time, whereas a dedication to continuous care will correct a poor initial design.

The design of many software applications emerges as a vital image in the minds of its designers. With luck and skill it may achieve a state of cleanliness, elegance, and beauty that fills its developers with deep satisfaction. Sometimes the developers manage to maintain this purity of design through the initial development and into the first release. More often something goes wrong. The software starts to rot like a piece of bad meat.

At first the degradation isn't so bad. An ugly wart here, a clumsy hack there, but the beauty of the design still shows through. Yet, over time as the rotting continues, the ugly festering sores and boils accumulate until they dominate the design of the application. This degradation makes the system ever harder to change and ever easier to break. Eventually the sheer effort required to make even the simplest of changes, and the ever increasing tide of reliability problems, become so onerous that the developers and stakeholders cry for a redesign.

Such redesigns rarely succeed. Though the designers start out with good intentions, and create an good initial design based upon their long experience, they find that they are shooting at a moving target. The old system continues to evolve and change to serve current customer's needs. The new design must keep up with these changes. The warts and ulcers accumulate in the new design before it ever makes it to its first release. On that fateful day, usually much later than planned, the corruption of the new design may be so bad that the developers are already crying for another redesign.

Why do designs degrade?

Designs degrade because requirements change in ways that the initial design did not anticipate. Those changes introduce new and unplanned dependencies between the modules of the system. As the dependency structure degrades, the system becomes ever more rigid, fragile, and imobile.

Initial Design vs. On-going care.

It has long been thought that a good initial design will eliminate, or at least mitigate the accumulation of these symptoms. Many of us have grown to believe that it is only bad designs that degrade. If the design is good enough it will be able to tolerate change without suffering il-effects. Therefore it is not uncommon for development teams to expend massive efforts on the initial design of a system in the hopes that those efforts will prevent the degradation of the software. Interestingly enough, those same teams often consider their job done once the initial design is in place; leaving the on-going maintenance of the system to someone else.

Agile development has become something of a watchword lately. The principles of this movement suggest a different approach to software design. Rather than placing a huge emphasis on a comprehensive initial design, they recommend a more modest initial design followed by continuous care. The goal is not to create a comprehensive design that cannot degrade. Rather the goal is to create a small but capable initial design, and then maintain and evolve that design over the life of the system. Indeed, more emphasis is placed on this subsequent care, than upon the initial design..

Attention to Change

According to the agile view, the design of a system requires constant attention. The forces that act to degrade a design are ubiquitous, and therefore the developers of a system must apply continuous effort to keep the design simple and flexible. They must aggresively attack any accumulation of dependencies, duplicate code, or unecessary infrastructure. They must constantly be on the lookout for localized areas of rigidity, fragility, and imobility.

The tools that agile developers use are the same as those applied to large initial designs. They apply principles of OOD and dependency managment to break or reroute unwanted dependencies. They build dependency firewalls to prevent the propogation of change. They use design patterns to solve well known problems. They use UML to communicate with each other and play what-if games. However, they do this *all the time*, throughout the life of the project.

This approach strongly mitigates the importance of a comprehensive initial design. Even if the initial design of the system is poor, the on-going care of an agile team will steadily improve that design.

Of course it's not the goal of agile development to create poor initial designs. However, it is not uncommon for the initial design of a system to be thwarted by changes to the underlying business and requirements. Indeed, many non-agile projects stall in the design phase because the designers cannot keep up with the changing business. Indeed, it is this ability to be responsive to change that gives agile development it's name.

Agile Project Structure

An agile project is developed as a series of very small releases. The development time allocated to each release differs for each team and method, but is typically less than a month. At the start of each release, the stakeholders and developers negotiate over the

features they want to see implemented by the end of the release. The developers then design and implement the release. As part of this design and implementation, they make changes to the existing structure of the system to accomodate the new features. Immense attention is paid to migrating the design of the *whole* system so that it is appropriate for the new features.

This constant migration of the design requires intimate knowledge of the system. Though documentation can help, initimate knowledge of a system is gained by working with that system for long periods of time. Thus, an agile team usually works on a system from beginning to end, instead of handing it off to a maintenance team after initial delivery.

This does not mean that individuals cannot transfer into or out of the team. Indeed, an agile team is very good at integrating newcomers because of the emphasis they place upon communication. However, an agile team changes personnel gradually. At any given point in time, most of the team will have long experience with the system. After ten years there may be no original team members left, but most of the existing team will have been around for awhile.

Agility

By placing an emphasis on continuous care, rather than initial design, agile teams are able to keep the design of the system at a very high level of quality, while also being able to migrate the functionality of the system to respond to business needs. For example, at Object Mentor we have a web based system that helps us manage our business. This system produces real-time forecast reports, opportunity listings, employee time reporting, consultant scheduling, invoice tracking, etc. This system has grown from very humble beginnings as a simple calendar manager. We had no idea that it would grow and change the way it has. And yet, the design of the system remains robust. We have migrated that design many times to keep it appropriate for its current function.

It should be possible for an agile team. over time, to migrate a system written for one purpose into a system that achieves a completely different purpose. Indeed, the team would barely know that such a change in purpose had taken place. Rather, from month to month, they would simply add new functionality, remove old functionality, and maintain the design to be appropriate for the current behavior.

Conclusion

The design of a software system is a living thing. It is not imposed all at once, but gradually over time as the function of the system changes and evolves. To keep the design appropriate to the current needs of the system, it must be cared for and evolved during the entire lifecycle of the project. Indeed, that care is far more important than the initial design itself. A poor initial concept can be corrected by on-going care; but without on-going care, a good initial design will degrade.