Software and House Requirements Engineering: Lessons Learned in Combatting Requirements Creep

Daniel M. Berry
Computer Science Department
Technion
Haifa 32000
ISRAEL
FAX:+972-4-822-1128 or +972-4-829-4353
dberry@cs.technion.ac.il
http://www.cs.technion.ac.il/ dberry

Introduction

Anyone who has built or remodeled a house and has developed or enhanced software must have noticed the similarity of these activities. This paper examines these two processes from the points of view of budgeting, scheduling, and requirements creep. It is admitted from the start that some of the arguments and conclusions are based on popular perceptions and personal observation over small populations, that is, the houses the author and some close friends have remodeled and built and software projects in which the author has participated as an analyst, designer, programmer, or consultant.

The Two Activities

When a customer wants to have a house built, she approaches an architect who puts his knowledge of house construction together with his creativity to try to come up with a plan for a house that will meet the customer's requirements. The customer describes her dream house and the architect asks her questions, although not necessarily in that order. The architect draws some possible plans or builds some models to show the customer his ideas. The customer gives feed back on these plans or models to allow the architect to better understand her needs and desires. Finally, the architect produces a final plan that meets the customer's requirements to her satisfaction and that is in a form that any builder will understand and will execute properly to build what the customer expects. A budget and schedule for building the house are drawn up and are bound together with the plans in the house building contract.

When a customer wants to have a program developed, he approaches a requirements engineer who puts her knowledge of software construction together with her creativity to come up with a specification of a program that will meet the customer's requirements. The customer describes his blue sky program and the requirements engineer asks him questions, although not necessarily in that order. The requirements engineer writes up some possible specifications or builds some prototype to show the customer some of her ideas. The customer gives feed back on these specifications or prototypes to allow the requirements engineer to better understand his needs and desires. Finally, the requirements engineer produces a final specification that meets the customer's requirements to his satisfaction and that is in a form that any programmer will understand and will program properly to implement what the customer expects. A budget and schedule for implementing the program are drawn up and are bound together with the specifications in the software development contract.

Similarities

The above descriptions of the two activities were deliberately framed to enhance the similarities of the house and software requirements engineering activities. There are elements of similarity that run deeper and include:

1. the relationship between the customer and the architect/requirements engineer,

2. the relationship between the customer and the builder/programmer,

3. the relationship between the architect/requirements engineer and the builder/programmer,

4. the importance of a graphical notation in the plans/specification, especially to the customer who is not in a building/programming profession,

5. the importance of the client's understanding the plans/specifications in validating that the plans/specifications capture his/her intent,

6. the usefulness of a model/prototype in the client's understanding the plans/specifications,

7. the degree to which the reason that the house/program does not satisfy the client is in the plans'/specifications' not matching the client's intent or needs rather than in the house's/program's not matching the plans/specifications,

8. the relative costs between changing the plans/specifications and changing the house/program,

9. the propensity for the client to think of new requirements as the house/program is being built/programmed, and

10. the debilitating effect of requirements creep on building/programming schedule and budget.

Key Difference

A key difference between house and program construction is that it seems that building contractors routinely charge extra and announce schedule delays for changes in the plans requested by the client. However, for some inexplicable reason, it seems that we software engineers are often reluctant to do either, routinely promising to handle new requirements within the confines of the original contract's budget and schedule [1]. We software engineers feel guilty even thinking about doing what contractors routinely, and logically, do without batting an eyelash.

In the immortal words of Tom DeMarco, "We're being had!" [2] We have been conditioned by the customers and programming managers of the world to accept utterly impossible schedules and budgets for software production [1] and to even accept adding more requirements to these schedules and budgets to make them even more ridiculously utterly impossible. We act as if it would be unpatriotic, unhumanly, unprogrammerly not to take on these few extra requirements or to insist on on a new contract, especially if the new requirements are delivered a slice at a time. After all, software is *so* flexible.

Indeed, in the house building trade, there is so much confidence that the client will change the requirements during the building that contractors routinely underbid on the original plans in order to win the contract, counting on recovering costs and gaining time with inflated cost and time estimates for each change requested by the client during the building. Most clients accept that the changes are more expensive per unit of area and require more time

than if they had been included in the original plans. After all, the support structure of the building must be changed at the cost of much material and time, especially since already completed parts of the house must be torn away or changed. Even with this understanding, most clients push ahead with the changes regardless of the costs.

A Personal Experience

On this particular issue, I, being aware of the costs of requirements creep in software, managed to outwit two building contractors, one for a remodeling and one for a build from scratch. I made sure before construction started that the plans both captured our (my family's) intent and were implementable, and I resisted all temptations to change the plans once the building started. In the end, we finished within budget on both plans, although not on time. Apparently, the contractors had bid impossible budgets and schedules. While the contractual budget can be enforced, a schedule is, in the last analysis, unenforceable. When the schedule slips, nothing can be done to recover, and stopping the building would only hurt us. In other words, I did not act like the normal client and the contractors got stuck into an unrealistic contract, unable to complete the building according to the schedule and unable to use a new requirement to excuse a slip in the schedule. Likewise, the contractor was unable to collect more money from us. In both cases, soon after completing our construction, the contractors faced serious financial difficulties, and in one case, bankruptcy, perhaps because they did not get from us the expected additional revenues that they had internally planned on assuming my behaving normally and worse than that, they were assessed penalties against their income for being late.

In the remodeling case, the contractor had started the building before I approved the plans and before I finished an inspection of the plans that showed a serious inconsistency in the plans. During this inspection, I found that the length of a wall determined by the scale of the plans differed by 6 inches from the sum of the widths of the built-ins attached to that wall, and the placement of the window in that wall when measured from the outside differed by 6 inches when measured from the inside. The contractor had started making the hole for that window based on the wrong measurements. I ended up having to go to a lawyer to force the contractor to stop building and to recognize his error in not waiting for my approval. The lawyer, himself a contractor, put my contractor on notice that he could lose his license. My contractor removed the foreman who had given the order to start building and replaced him by another, a former programmer! When I explained to the new contractor that I did not want to start building until I had finished verifying and validating the plans, he smiled a smile of mutual understanding. He and I were on the same wavelength!

Another Possible Factor

Glenn Jennings, after reading an earlier draft of this paper wondered if part of the difference is that people expect to live in houses for 10 or more years, but the lifetime of some programs is expected to be one or two orders of magnitude less! It is as though the next house were impossibly far into the future, but the next program is so near already, that the programmers start writing it before the first is finished, thus leading to requirements creep.

Conclusions

It is time that we software engineers were as smart about not accepting requirements creep as building contractors. It is not necessary for us to go so far as to underbid, counting on requirements creep, as we could find ourselves outwitted by clients who read this paper. Furthermore, we need to wait until the customer has thoroughly inspected the specifications and has validated that they meet his/her intentions before proceeding with very expensive implementation. Finally, as Tom DeMarco exhorts, we must learn not to accept anything but realistic budgets and schedules.

References

[1] DeMarco, T., “Why Does Software Cost So Much?,” in *Why Does Software Cost So Much? and Other Puzzles of the Information Age*, Dorset House, New York, NY (1995).

[2] DeMarco, T., “Requirements Engineering: Why Aren’t We Better at It?,” pp. 2 in *Proceedings of the Second International Conference on Requirements Engineering*, Colorado Springs, CO (1996). Keynote.