The CRUD Methodology

Akiva Leffert
Pomegranate Inc.

I know thy works, and thy labour, and thy patience...
- Revelations 2:2

Recent years have seen the rise of a number of software engineering methodologies with a variety of silly names like Extreme Programming(XP)[2] and Scrum[7] grouped under the umbrella of *agile development*. These methodologies vary wildly in approach, but have several aspects and goals in common: A focus on rapidly developing working code, keeping requirements clear, and aggressive unit testing. However, these approaches are designed to produce software that works, at least that is their claim. Few have tackled programming processes for software that doesn't work and isn't meant to.

In the remainder of this paper, we present Creating Rubbish Using Developers(CRUD)[1]. A methodology for producing bad code. We first present and elucidate the principles of CRUD. We then present examples of the CRUD methodology in action. Finally, we talk about applying CRUD to your software process and how you can reap the benefits of CRUD.

1 Principles of CRUD

It is important for a software development methodology to have principles. Otherwise, the other methodologies would consider it a weak-kneed spineless jellyfish[5] hack of a process and will steal its lunch money, leaving it penniless and sad[4]. Also, lists of principles look good on Wikipedia[8] pages and glorified blog posts[6] promoting its merits. The principles of CRUD are simple:

1. Don't think.

2. Don't act.

There are many ways to produce software that doesn't work e.g. too few programmers, too many programmers, incompetent management, incompetent programmers, contracting to a sketchy guy on the street, bubonic plague, unclear requirements, extremely clear but impossible to implement requirements, demonic possession, dependence on outside code, bad tools, and getting involved in a land war in Asia. However, most of these methods are unreliable, programmers occasionally thrive despite bad management, the demon possessing your lead engineer may turn out to be a serious hacker.

The advantage of the CRUD methodology is that it basically guarantees your project will fail. Even if only one of the CRUD principles is followed, and a good CRUD programmer should be entirely unprincipled, the software project is still likely to fail. In the rare circumstance where

[1]Not to be confused with whatever CRUD it is the database people talk about

the goals of a project are met using the CRUD methodology, the code is often unmaintainable, impenetrable, and difficult to extend.

2 Real CRUD

Here we present several examples of the CRUD process in action or inaction as the case may be.

2.1 Don't Think

A developer has come up with two ways to implement a feature. The first way involves creating a variety of infrastructure, which will take more time to produce up front, but will be useful later in the project. The second way involves writing a much smaller, but still decent amount of code, which will be much harder to change later. Most developers would choose the former, knowing that they will save time later. Developers on a deadline would typically choose the latter, but would feel bad. CRUD developers would sidestep this problem entirely, by grabbing some code from the internet, changing a few bits, and calling it a day. The problem is solved in a tenth the time, is wholly unmaintainable, because no one involved in the process knows how the code works, and mostly importantly used almost no neural energy.

2.2 Don't Act

A piece of code is like any buffoon: The bigger it is, the harder it falls. Large code bases are hard to understand and difficult to modify. The foundation of agile methods is that simpler programs and simpler processes are much easier to maintain and succeed with. This is like bringing a knife to a gun fight. At the same time, at the point where you've gotten into a gun fight, you've made a wrong turn somewhere. Any piece of existing code restricts implementation choices, restricts design possibilities, and generally makes the task of adding new functionality more difficult.

A wise piece of software once said, the only winning move is not to play[1]. The CRUD methodology takes this idea to heart. The Don't Act principle of CRUD articulates that its always easier to not do something than to do something. If your goal is to produce incorrect software, the easiest incorrect software to produce is the empty program. Experienced CRUD programmers eventually attain a zen-like understanding that written code is buggy and hard to manipulate and that the best code is that which is unwritten.

3 Conclusion

It is well known that half of all software projects fail[3]. We think that this isn't good enough. All software projects should fail. The software crisis should become the software apocalypse. After the software apocalypse comes the software post-apocalypse. In this post-apocalypse, the software wasteland is a barren mess of barely started sourceforge projects, increasingly decrepit, but still functional COBOL programs, and the occasional bit of code that only builds on alternate Tuesdays after the sacrifice of a goat.

Developers willing to brave this wild frontier will be hailed as heroes. Adoring crowds will follow them through the otherwise abandoned streets, tossing wilted and mutant flowers in their wake. They will be as gods amongst the twisted men of this future age, rationing code like a stern

quartermaster deep in enemy territory invading Russia in the cold winter of 1812-1813. We can hasten the arrival of this day by adopting principles of CRUD and ruthlessly terrorizing those who don't.

References

[1] Wargames, 1983.

[2] Kent Beck. *Extreme (Xtreme!!!!! For serious!) Programming Explained.* Addison-Wesley, 2001.

[3] Nels Beckman. The g-unit testing harness: Achieving source code street cred. *Proceedings of the Workshop about Symposium on Robot Dance Party of Conference in Celebration of Harry Q. Bovik's 0x40th Birthday*, 2007.

[4] Stephen R. Covey. *The 7 Habits of Highly Effective People.* Free Press, 1990.

[5] Ernst Haeckel. *Art Froms in Nature: The Prints of Ernst Hackel.* Presetl Publishing, 1998.

[6] Andy Hunt and Dave Thomas. Orthogonality and the DRY principle.

[7] Ken Schwaber (teehee Schwaber). *Agile Project Management with Scrum (teehee Scrum).* Microsoft Press, 2004.

[8] Tom Murphy VII. Wikiplia: The programming language anyone can edit. *Proceedings of the Workshop about Symposium on Robot Dance Party of Conference in Celebration of Harry Q. Bovik's 0x40th Birthday*, 2007.