Measure Once, Then Cut?

October 12, 2020 By Brad Evert

"Measure twice, cut once." This age-old English and Russian proverb originated in carpentry and needlework[1]. It refers to taking special care in understanding requirements and exercising due diligence in execution of work. This proverb permeates modern business practices, from strategic decision making to the smallest manufacturing processes. Modern software development methodologies, like Waterfall[2] and Agile, stem from an intense focus on nailing down requirements to more than 80% understanding before writing a single line of code.
This "get ready to prepare to get started" approach front-loads the cost of the project in hopes of decreasing time spent coding. This is often the case in endeavors with well-known, quantifiable requirements. Unfortunately, a lot of software efforts do not enjoy such a stable business problem domain. So the question remains, lurking in the dark, should we always measure twice before we cut?

When I was growing up, I was fortunate enough to enjoy the trifecta of living in a town suburb near my great grandparent's large farm, where my grandfather and uncle had also established an industrial manufacturing business. For a young boy, this provided a wealth of opportunities for trial-and-error learning (ok, getting into trouble). Through this, I developed a tendency to "go with my gut" or "jump in head first," taking calculated risk shortcuts to find out information or get results. While in college pursuing an Engineering degree, every lecture was accompanied by lab time, where theories were understood through practical application in the real world. Both electronic hardware and software were covered in this Computer Engineering program, with interesting conclusions. When working with hardware, mistakes were extremely costly – sometimes frying components in a puff of smoke, other times completely starting over etching a new printed circuit board because of one misplaced trace. When writing the operating programs for the hardware created, even more mistakes were made as my inexperience led me into unknown traps of software development. Interestingly, I was able to recover quickly, rip up and rewrite code as new requirements revealed themselves. The results were generally more complete solutions.

When dealing with physical raw materials, we get less do-overs and so need to be very judicious in their consumption. With software, yes, we want to have as many requirements as we can, but there is no replacement for the journey of discovery as a requirements refinement tool. Now, I am not advocating jumping into writing code with no pre-work since that leads to both scope creep and bunny trails. Each project should be approached with the idea of a good-enough (minimum viable product) requirements position to prevent analysis paralysis. Yes, it's ok to break a few eggs. Yes, it's ok if it's not perfect. "Getting it right" with software is more than the basics (which should be right of course). Great software feels natural, and is intuitive. It resonates with users.

At this point, a lot of users will cry foul. I know I have been frustrated when encountering software issues. But, with risk comes reward, and software is ever-evolving. The truth is software is alchemic, conjured from nothing, and then forged (not cut) into something useful.

1. Oxford Dictionary of Proverbs 5th Edition 2008, 2009
2. Practical Software Development Techniques, O'Reilly Media, 2020
3. Do You Have Analysis Paralysis? Psychology Today, BobTaibbi, L.C.S.W, 2019
4. Lean Product Management, Mangalam Nadakumar, 2018

About the Author

Brad Evert

Dr. Brad R. Evert is the Director of Development at Automated Systems Inc. He is a business and technology evangelist with over 30 years of experience in the financial sector. Mr. Evert has served as a board member for 8 companies delivering products and services in financial, technical, real estate, travel and marketing call center markets.