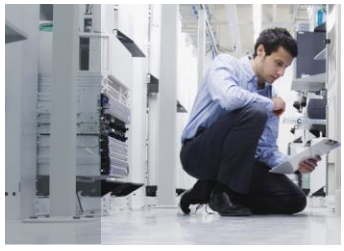
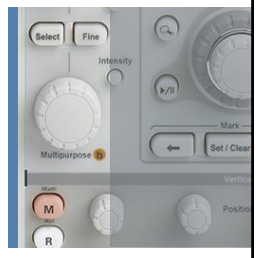
Baker's Dozen of Inconvenient Truths about Software Engineering

Tom Feliz
tom.feliz@tektronix.com





Author Biography

Tom Feliz is a Lead Software Design Engineer at Tektronix Corporation in Beaverton, Oregon. He has been engineering software since he bought his first computer, a Commodore VIC-20, in 1983. Prior to joining Tektronix, Tom founded multiple technology startups and has worked in a variety of Software Engineering environments. His professional interests include Embedded Software Engineering, Hardware-Software Integration, and Software Quality.

Tom Feliz has a Master of Science in Computer Science and Engineering (MSCSE) from OHSU OGI School of Science and Engineering. He was awarded the IEEE Certified Software Development Professional (CSDP) credential in 2007, one of only a handful in Oregon.

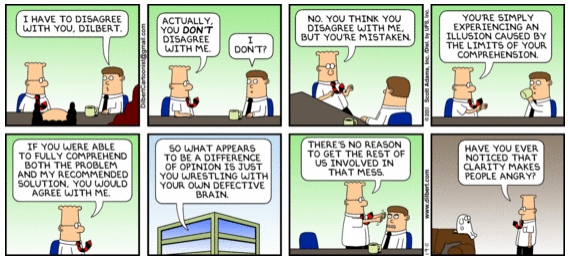
Introduction and Caveats

Software developers tend to value intuition above all else. This list is an attempt to summarize the inconvenient "truths" of Software Engineering that may not be immediately apparent or intuitive.

These "truths" are based on my experience, though most appear repeatedly in Software Engineering literature.

The relevance of these "truths" is somewhat context dependent and are aimed at medium to large commercial software organizations.

And, of course, your mileage may vary…



13. Software Engineers are generally terrible estimators and often underestimate by a factor of 2-3x or more.

Worse, estimates, if done at all, are typically done once at the beginning of a project, when uncertainty is greatest (i.e. cone of uncertainty). Despite this, most well-reasoned estimates are better than having no estimates at all.

Source: McConnell 1996, pp.168-169

12. Solving software problems by adding people has limited efficacy.

The incremental productivity gain from an additional Software Engineer diminishes with each person due to the increased overhead of project coordination and communication. This is particularly true for software engineers added late to a project.

Sources: Brooks, pp.25-26, Glass, pp.16-17,

11. The most complex technical problems are almost always architectural in nature.

Coding efficiency and optimization are not nearly as important. Therefore, proper attention must always be given to good architectural design and review practices, preferably early in a project.

Sources: Brooks, p.42; Glass pp.139-141

10. Software projects are becoming exponentially more complex, yet the intelligence of the average Software Engineer has not changed (Capers Jones).

To compound matters, the way software systems are built hasn't fundamentally changed either (i.e. there are no silver bullets).

Sources: Brooks, p.181;

9. Complexity is the enemy of Software Quality.

Consequently, it's imperative to maintain a strong preference for simplicity (e.g. simplifying architectural designs, minimizing supported platforms, removing legacy features, applying the Pareto Principle to requirements, reducing the frequency of releases, and avoiding the "new shiny thing" if it doesn't add immediate value.

8. Surprisingly, the average developer has little or no formal Software Engineering or Process background.

In fact, many undergraduate Computer Science degrees don't even require an introductory Software Engineering course. The situation is worse among Computer Engineers and Electrical Engineers.

"[The] Average developer reads less than 1 professional book/year and subscribes to no professional journals."

- Steve McConnell

Sources: Ford and Gibbs 1996; Miller 2002

7. The key difference between junior and senior Software Engineers is that junior Software Engineers naively believe they can deliver defect-free software by their sheer brilliance and grit.

Senior Software Engineers, in contrast, know that defect-free software is nearly impossible, avoid over-commitment, always seek to reduce complexity, and aggressively employ practices to minimize defect injection.

6. When developers are incentivized solely based on the quantity of code they produce, regardless of quality, there is an inherit conflict of interest.

In general, developers want to have pride of ownership and produce quality software.

Software Engineers must also be empowered to produce high quality work products and utilize well-known best practices to drive defect detection upstream.

Source: McConnell 1996, p.269

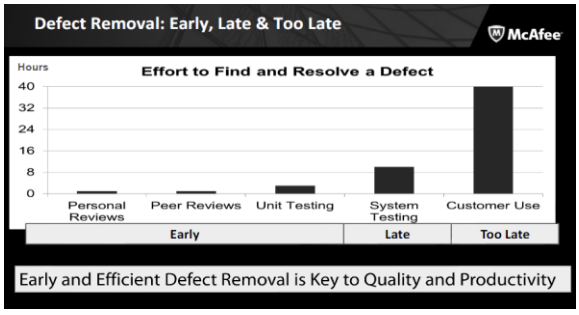
5. There is such a thing as negative productivity for individuals, as well as teams.

Latent defects that escape early detection, often multiply into a 10-20x impact on productivity later in a project. The potential impact is even worse for long-lived codebases.

Sources: McConnell 1998; DeMarco and Lister 1999, p.133; Lopp, pp.147-153

4. The bulk of debugging effort is not in fixing defects (10%), but in finding the root cause of defects (90%).

Therefore, it's essential to find defects as soon as possible after they're introduced while still easy to find.



Source: Sartain 2011

3. The actual coding phase of a software development lifecycle (SDLC) is less than 20% of the total effort required.

Experienced Software Engineers know that coding is only one activity of many and seek to develop expertise in all aspects of the SDLC.

Source: Brooks, p.20; Glass p.90

2. Ultimately, Software Engineering is about Risk Management.

The primary purpose of all development lifecycles, software processes, and software best practices is to control risk and improve the predictability of delivering high quality software in a timely manner with the relevant features required by customers.

"Risk Management is Project Management for Adults"

Tom DeMarco and Timothy Lister

Waltzing with Bears

Source: DeMarco and Lister 2003

1. If a software organization wants to improve schedule predictability, it must focus on improving software quality.

Quality always leads productivity because defects discovered late in a project frequently lead to significant schedule delays later in a project.

"If you do up-front inspections and if you use static analysis before testing begins, your schedule will be shrunk by at least 50% and your software quality will be above 95% [Defect Removal Efficiency]."

– Capers Jones

Source: Diaz and King 2002; Jones 1994

References

Brooks, Frederick P. *The mythical man-month : essays on software engineering*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1995.

DeMarco, Tom, and Timothy R. Lister. *Peopleware : productive projects and teams*. New York, NY: Dorset House Pub, 1999.

DeMarco, Tom, and Timothy R. Lister. *Waltzing with bears : managing risk on software projects*. New York: Dorset House Pub, 2003.

Diaz, Michael, and Jeff King. "How CMM Impacts Quality, Productivity, Rework, and the Bottom Line," Crosstalk, The Journal of Defense Software Engineering, vol. 15, no. 3, March 2002, pp. 9-14.

Ford, Gary, and Norman Gibbs. *A Mature Profession of Software Engineering (CMU/SEI-96-TR-004)*. Pittsburg: Software Engineering Institute, Carnegie Mellon University, 1996.

Glass, Robert L. *Facts and fallacies of software engineering*. Boston, MA: Addison-Wesley, 2003.

Jones, Capers. Assessment and Control of Software Risks. Englewood Cliffs, N.J.: Yourdon Press, 1994.

Jones, Capers. *Software engineering best practices lessons from successful projects in the top companies*. New York: McGraw-Hill, 2010.

Lopp, Michael. *Being geek : the software developer's career handbook*. Farnham: O'Reilly, 2010.

McConnell, Steve. *Rapid development : taming wild software schedules*. Redmond, Wash: Microsoft Press, 1996.

McConnell, Steve. "Dealing with Problem Programmers," IEEE Software, vol. 15, no. 2, March/April 1998.

Miller, Dave. "Software Quality Requires Professionalism and Fortitude," Software Quality Professional, vol. 4, no. 4, September 2002, pp. 28-33.

Sartain, Jim. "Inspiring, Enabling and Driving Quality Improvement," PNSQC 2011 Proceedings: 193-202.

Thank You!

Do any of you have inconvenient "truths" you've discovered?

tom.feliz@tektronix.com