How the Web Resuscitated Evolutionary Design

Jeff Offutt
George Mason University

October 2015

One of my favorite oldies, The Design of Everyday Things, discusses evolutionary design. It caused me to consider what this concept means to software design, development, and testing. I want to start with cost. All technological artifacts, hardware and software, come with costs. Not being an economist or systems engineer, I may leave some out, but at least four types of costs help us understand some of what's happening in software:

1. **Design**
2. **Production**
3. **Distribution**
4. **Support**

The relative amounts of these four costs are continuously changing. Back "in the day," we evolved our designs very slowly. For centuries before the industrial evolution, most things were crafted by hand. A carpenter took days or weeks to build a rocking chair with hand tools. After each chair was built, the carpenter considered ways to make the next one better. The design evolved over time, but each new chair was a little bit better than the last.

Hand-crafting incurred very little design costs, but had very high production costs. Craftsmen worked for days, weeks, or months on each product. The distribution cost was usually low–most craftsmen before about 1850 sold directly to customers, usually in the same village or town. Support cost was also low. Buyers used the product until it wore out; usually years if not decades.

Then manufacturing and the industrial revolution changed the equation. Assembly lines allow many people to work on one product, letting the same team produce many products quickly. The same design was put into hundreds or thousands of products, bringing production and overall costs down dramatically. But to ensure quality, designs had to be created more carefully, and encoded into a factory to allow the assembly line to operate. Thus, design costs went up. Because more products could be created in the same factory, products were distributed more widely, increasing distribution costs. More complicated technological objects, in turn, meant that support costs started increasing. They were often outsourced to people like car mechanics or plumbers, so that customers bore the support costs.

We developed automated manufacturing in the 1900s. Automation increased the speed and efficiency of production, so production cost continued to decline. Design costs now included creating expensive automation hardware, including robots. At the same time, distribution continued to expand, increasing costs.

After World War II, we globalized our manufacturing. In particular, cheap energy, large ships, and global air freight dramatically increased our ability to distribute. As automation increased with electronics and software, production costs continued to decline, and in turn, design costs continued to increase. As costs decreased, support also decreased because people replaced instead of maintained.

By the end of the 20th century, free trade accelerated globalization. Oil became cheaper and shipping costs declined. Ultimately, design became the dominant cost of many engineering products. In other words:

Manufacturing defeated evolutionary design!

Automation and globalization allow people to buy thousands of incredibly cheap products, but many are low quality and built to last a few months instead of decades. Instead of evolution, we have replacement. We have lost something precious and wonderful: **craftsmanship**.

What does this have to do with software?

Traditional software development such as the waterfall process has very low production cost, and substantial distribution cost that includes marketing, sales, and shipping. Support costs escalated as software became larger and more complicated. Design activities split into software design and software implementation. That is, what software engineers call implementation is more akin to design in hardware, and software production is simply compiling and copying files. In software engineering, both design and implementation is very expensive.

In the 1990s, having millions of customers skewed costs to the back end, where support costs and distribution costs steadily increased. New versions of major software products shipped every five to ten years, which meant that software problems affected users for years. This forced a mentality that software had to be "perfect out of the box." Both design and implementation became very expensive—including testing, which was 50% or more of the total cost. As a result, software evolved very slowly.

The need to be "perfect out of the box" has heavily influenced decades of software engineering research, creating such goals as describing software formally, modeling large systems, creating processes that led to perfection, testing products after completion, and looking at maintenance in terms of years. Until recently, most of our research focus and results have assumed that design costs, implementation costs, support costs, and distribution costs are high.

But a countercurrent had been slowly but steadily forming as storage technology drove software distribution costs down. We moved from shipping large disk packs to floppy disks, to diskettes, to CDs, to thumb drives. At the same time, engineers responded to high support costs by increasing usability, which decreases the need for customer support.

Then the web changed everything, creating a tipping point: First, the web created a new way to deploy and distribute software. Second, the web rearranged the importance of quality criteria, making usability and reliability crucial as competitive advantages.

The web is used to distribute desktop software nearly instantaneously and with near zero-cost, allowing more frequent updates. Even more impressive, software that runs on the web (web applications) are not distributed in any meaningful sense. Web applications reside on servers where updates can be made weekly, daily, hourly, or even continuously! Even more extreme, applications on mobile devices allow the craftsman to come into your "home" to improve that rocking chair at any time.

Almost magically, near-zero production costs … immediate distribution … and near-zero support costs … resuscitated evolutionary design!

In summary, before the web, design was expensive, development was expensive, new versions were deployed infrequently, and evolution was very slow. Post-web software engineers design and develop "pretty good" initial versions, then gradually make it bigger and better. Evolution is faster as we make immediate changes to web applications, automatic updates to desktop applications, push out software upgrades to mobile devices, and replace chips in cars during oil changes. This changes all of software engineering!

As researchers, we have the opportunity to look at many novel and interesting problems. Software is not so much designed and built; it grows. Testing must focus on evolution, not new software. The waterfall process is now, finally, thankfully, completely dead. Yes, the web really does change EVERYTHING.

These changes affect software testing in deep and fundamental ways. Test-driven design uses tests to drive requirements—every step is evolutionary. We must stop thinking of regression testing as something special done "late in the process," and think of virtually all testing as regression testing. Model-based testing is so widely studied and used because it allows test design to quickly and easily adapt to changes. Test automation is essential to running tests as often as software changes occur.

This new world brings many new questions. How do we translate from test models to automated tests? What is our best strategy for creating test oracles? How do we test continuously, when requirements evolve too often to track? The meta-question that all software testing researchers should ask is: Does our research support evolutionary design?

References

[1] The Design of Everyday Things, Don Norman, Basic Books, 1988.