In 1859, Charles Dickens began his serial publication of "A Tale of Two Cities," which starts out, “It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity.”

It sounds an awful lot like the story of the past 50-year introduction of computers to business. On one side — the worst of times — an exceptionally messy experience, fraught with missteps, exploited by shameless hucksterism and punctuated by colossal failure, massive losses of time and money; and many a damaged or derailed career. And on the other
— the best of times — computers have, on occasion, provided existing organizations sustained competitive advantage, exceptional increases in efficiency and productivity, and vast new capabilities in marketing and data collection; not to mention entirely new services and business models.

Few business tools can claim a comparable track record. And the question is, of course, why?

This three-part series aims to provide a number of what we feel have been missing pieces in the story of the development of information technology in the enterprise. These missing pieces justify the adoption of a DevOps approach, and in turn, provide a rationale that improves overall probability of success. This first part outlines the context within which this technology has developed and explains why you need to consider adopting DevOps. Subsequent parts will develop some important measures of success that give insight into how you might go about implementing DevOps.

**Once Upon a Time**

So much of what we do today is a result of, and often an imitation of, what we’ve done in the past. And the past was often as not, not particularly pretty.

Computers have been a part of business since at the very least the 1950s, more likely earlier. The Lyons Electronic Office I (LEO I) was supporting business applications in a British company as early as 1951; GE declares that it was “the first company to tap the ENIAC for solving hard engineering problems” some 66 years ago; Computerworld cites a programming team from GE, Arthur Andersen and Remington Rand as responsible for the first payroll application “that worked” in 1954.

The early years and those early players were critical in shaping how the technology would be seen and how it would ultimately develop. The business machine companies like IBM and Remington Rand regarded the new devices as a boundless stream of wholly new future earnings. Firms like Arthur Andersen, meanwhile, were looking for ways to expand their audit practices and searching for new sources of revenue.
Computers served these companies and companies like them extraordinarily well. And these three companies can serve as reasonable proxies for the major early influences in how information technology was adopted and introduced: they were engineers and auditors/accountants; but above all else, they were salesmen.

**Random Walks and Winding Roads**

Before we get too much further, let’s acknowledge that the summary above is greatly simplified, to say the very least. There are quite literally volumes of detail and nuance that could be added to the above that provides a far richer and more accurate history of the development of information technology in business. Where, for instance, are the titans like Larry Ellison or Bill Gates or Steve Jobs accounted for? The rise of Microsoft and Apple? Cisco? Or even Xerox PARC? How about the DEC story, remember that one? IBM was certainly successful with more than just "big iron."

There’s no denying that what's been outlined is a gross simplification. Nonetheless it does the specific job of providing part of the necessary context: when new capabilities are introduced, mishaps occur, mistakes are made, missteps are taken. And they result from any number of market forces and conditions. The market does not instantaneously produce the singularly most efficient solution any more than evolution delivers a finished human being from a single cell.

And nowhere has that been more apparent than the introduction of computing devices into the enterprise.
It's relatively easy from this very simplified starting point to call out early missteps in the development and evolution of computers in business. The first is how we’ve chosen to “see” systems. Very early in the introduction the market more or less split into those who produced hardware components and those who produced software applications. And while individual firms might organize to produce their products and services in such fashion, the customers for those products and services would have been better served if they had not allowed themselves to be influenced into organizing internal replicas of such. After all, for McCormick & Dodge, an early maker of financial software, the machine was something of an afterthought. For IBM, the machine was the primary mission with the provision of software seen as ancillary. But for the customer, you needed both. And you needed them to work well together. The idea that these “elements” could somehow be managed separate and apart or that they were two separate and independent things, tends to be more a result of early market players, and their business interests, than any natural or inherent division.

So, thus far, we can say this — if we simplify the concept of DevOps to the basic idea of fusing various parts of the enterprise that had previously acted virtually autonomously, then we can say that it’s not so much something new as it is an acknowledgement of early missteps and the need for a course correction.

**And That’s Not All Folks**

There’s yet a second early misstep. There was little business experience with the creation of highly complex and intangible products prior to the introduction of large-scale business software. It’s absolutely true that we could engineer an aircraft carrier as well as the aircraft that took off from it. It’s equally true that we had little experience engineering anything that you couldn’t put a wrench on or create a blueprint of. In such cases, business did what business typically does: it relied on experience — with decidedly mixed results.
For software producers, taking the known processes from business and engineering created several serious issues and outright failures. Everything from early releases of IBM’s relational database product DB2 to early releases of Oracle to even such fundamental components like initial versions of MS Windows were fraught with product errors that can reasonably be assigned to the process used to create them. Proving that business processes that may work for some manufactured products are clearly stretched to the limits when considering complex intangible products that require integration with equally complex and tangible platforms.

When the techniques and methods of the marketplace were adopted by firms focused on creating custom software internally, the results were nothing short of disastrous. By some estimates, fewer than a third of custom software projects were ever completed on time or within budget. And those two phrases are crucial.

First, it's because they are "top level" measures of success. In essentially two parameters we have, in the past, decided whether we accomplished our objectives or not. By extension, they are the method for determining the project boundaries. So, once again in the past, we decided there was an “end point” or in the parlance of consultants a “final deliverable” that needed to be accomplished within the bounds of the resources (or the budget and time) allocated.

So, while manufacturing is served by the defined boundaries (on time, within budget) and well-defined and understood specifications, these arbitrary constraints, more often than not, do not apply to the construction of software. And while they may have provided the Big Six consulting firms as well as numerous others with significant revenue growth, they proved woefully inadequate and ineffective in individual firm efforts to construct custom software. And equally inadequate to simply delivering product.

Clearly something else was necessary. The very first of which is a simple declaration: Software? Is only done, when it’s retired and taken out of service.

In the next part of the series we’ll discuss the critical first step in what that ‘something else’ that’s necessary is. And likely as not, it isn’t perhaps what you think. While metrics are vital to the success or failure of most — if not all — endeavors, it’s not what should come next in the quest to embrace the cultural change prescribed by DevOps.

This was adapted from an original blog post by Al DuPree, who is an information systems professional with international experience in strategic planning, software application development and delivery, data center operations and information
technology related service delivery. His past experiences include chief of innovation and solutions division at the National Institute of Standards and Technology and CIO at the Congressional Budget Office. You can hear more from DuPree in an upcoming episode of our new podcast, The Agile Advocate.