"What is more important: Software quality or productivity?"

November 2013

Scope of this Report

There are a variety of ways to look at this question. The first response that comes to mind is to simply say, it depends. It depends on who is asking the question. For example, quality may be far more important than productivity if you are talking about a customer who is using the software. It depends on how we define quality and define productivity. Finally, it depends on what we mean by important.

Definitions

For purposes of this article we will assume the following definitions or descriptions for quality, productivity and importance:

**Quality** - the absence of defects and conforming to user requirements. So a quality deliverable is a working piece of software that is bug free and provides features and functions that a customer requested. It is understood that there are additional ways to characterize quality, such as the quality of code; but for our purposes here, we will use what we think of as a general description of quality.

**Productivity** - Output (real output metrics such as function points are preferred but this is often measured as cost or effort – sometimes speed of delivery for unit output is used) divided by input (usually effort). If we are productively producing software then we are working effectively and efficiently thereby minimizing the level of effort and ideally minimizing the time it takes to deliver.

**Importance** is a bit more challenging to define and should be looked at from different perspectives. For this discussion, we will assume that there are three main players that are involved or impacted by the development and deployment of a software deliverable. There is an end user (customer), there is a developer and there is an individual or a job function that is responsible for paying for the development and ongoing support of the software. We will refer to that person/position as the owner. It is worthwhile to be explicit here: under our definition, the customer (end user) is usually not the owner (investor).

Before we leave our definitions section, it is worth considering the picture that most software practitioners carry around in their heads in one form or another (shown below as Amblers “Iron Triangle).
The diagram’s familiarity means that it requires little explanation but we’d ask you to look at it again more carefully in the context of the question at the heart of this paper: quality or productivity? As we can see, Ambler’s version of the triangle assumes that quality is a given – “non-negotiable” perhaps? Odd - we all know that quality is sometimes compromised to get a release out of the door. Where is productivity? “Scope” or “Schedule” could be the output with “Resources” as the input. Perhaps the real point here is that while “quality” always has a place in the Developers mind, “productivity” does not get as much consideration compared to Scope and Schedule. Some but not all practitioners realize that improving productivity is the key to better results on scope and schedule.

**From a Customer Perspective**

Customers want it all! They want working software that provides the features and functions they need to perform their work. In certain circumstances, it will also be important that the software be delivered in a timely fashion to meet the demands of a competitive marketplace. Only if the Customer is also the owner (paying for the software themselves) will they prioritize an economical solution (see issues associated with Owner perspective below).

If the Customer is working in an industry that is highly competitive, speed of delivery is often the highest priority and a Customer may be willing to compromise some software features and functions if it means getting a faster delivery.

If there is a typical scenario, experience tells us that a customer can usually find ways to afford a project implementation even when it is coming in “reasonably” over budget. Further, a late delivery is soon forgiven and forgotten if the software is feature-rich and performing flawlessly.

So, regarding the question of which is more important to the Customer...advantage quality.

**From the Developer Perspective**

In IT shops everywhere, what is fundamentally important to the Developer is simply what is important to the company or perhaps more specifically their boss and/or their bosses boss. Budgets and schedules are often dictated to the Developers. Often times, at the Develop level, speed is of the essence; getting a product out the door ranks higher than the cost or the quality of the software or the underlying productivity. That said, most Developers and, by extension, their immediate supervisors (who were once Developers) are consciously or unconsciously aware that some patterns of work that they must sometimes endure (e.g. being assigned to more than 2-3 projects or incessant interruptions) negatively
impact their productivity. Ultimately, an unmitigated focus on speed almost always impacts the budget so the Owner insists that some of the features and functions being asked for by the Customer are cut back. As an industry, we have come to accept that moving some features to phase II is a fact of life! We all know that story. Not only have we not delighted our customer, we have most likely created software that will have quality issues somewhere down the line. However, this usually is not the Developers concern or responsibility since they will be turning the system over to a maintenance support team. Rarely are Developers (or their managers) held accountable unless the shortfall in scope is a complete disaster.

To be fair, software Developers, for the most part, want to do a good job. They want to produce high quality code that they can be proud of. They also understand the ‘pay me now or pay me later’ adage of creating bug free code versus leaving technical debt. However, they also want to keep their jobs and so they are wise to heed the directives from above and if speed and costs are the focus...well then they best follow orders.

It is important to remember that higher productivity developers work faster but developers working at their fastest are not necessarily the most productive

So, regarding the question of which is more important to the Developer (company)...advantage speed (which is often but not always synonymous with productivity).

From the Owners Perspective

Clearly speed may be of the essence and certainly quality is a concern based on the desire to have a satisfied customer, but at the end of the day, it comes down to cost. The issue of cost is interesting because there can be a short term view of cost and long term view of cost, particularly from an owners perspective.

The short term view of cost considers the cost to develop and deploy the software. Within an organization there is a great deal of emphasis on annual budgeting for newly developed or enhanced software. Estimates are developed, discussed and debated. Once the budget is formalized and development of software begins there is the ongoing monitoring and tracking of budget dollars being spent on development. Of course software development projects get off course and incur additional expense and often need additional time to deliver. These are the projects that are viewed organizationally as failures. On the other hand, projects that come in under budget and are delivered early are considered major successes - perhaps regardless of quality.

The longer term view of cost considers the cost to maintain software. Here too, the budgeting process accounts for the resources that will be required during the coming year to support and maintain production software. Typically the maintenance costs include system upgrades, bug fixes and minor change requests. Any major enhancements or newly added features and functions are budgeted under the development dollars.

So, what's more important here, quality or productivity? Although there are estimation techniques available, few organizations attempt to predict how many defects a delivered piece of software is going to have – perhaps because they distrust the accuracy of the available techniques compared to the “baseline” of the Developer’s eternal optimism about quality. Hence, the immediate focus is on the
initial delivery and therefore on cost and time to market. However, if a software deliverable is poorly developed, either technically or logically, than it may fail to deliver the necessary functions and features and/or it may simply fail to function at all. In those situations, the cost to maintain that software escalates. Also, there can be a direct impact on the business when a dissatisfied customer chooses to do business with a competitor who has a better working product. So when organizations are focusing on speed and dollars, and potentially compromising quality, they are taking a big risk. Of all the three perspectives, the Owner is likely to have the longest time view and to be the most risk averse. Advantage --- quality

**Rework**

One way to reconcile the three perspectives of Customer, Developer and Owner, or at least to measure how far apart their priorities are in reality, is to assign the costs of code rework, before and after release to Developers. This is rarely done because it is hard. Defects in unit testing and pre-release integration testing are usually fixed by the Developers on the project and, as such, this time is included in retrospective productivity calculations. However, post-release defect fixing is hard to trace back to particular releases and so is bundled into maintenance productivity not development productivity. Interestingly, hurried testing, due to poor productivity on development or design cutting available time for testing, finds fewer defects for development to fix pre-release. Few defects found pre-release means less time bug fixing for the Developers and better productivity numbers – except that the maintenance team gets all the bugs to fix.

Another perspective of rework and its structural companion, technical debt, is that buggy or complex code makes future development harder causing productivity to drop. This can be especially true in an environment where application release are “pipelined” so that development hands one release over to test and then immediately start work on the next release – in effect building on the defects that test has not found yet.

Best practice here is to find ways to track post-release rework and include it in productivity calculations for Developer teams – even if it means limiting the post-release defects counted to some arbitrary time after the release e.g. three months.

**Is there an optimum balance between quality and productivity?**

Let’s consider how we might best maximize both productivity and quality. The three factors, all representing a certain amount of risk, that are considered when planning to develop and deliver software are; speed, cost and quality. Developers are often heard saying, “pick two”. In other words, you can’t have all three. If you want it fast and cheap, then quality is naturally going to suffer. If you want high quality it is going to cost you time and money.

Software development is a very new discipline and other disciplines have faced the challenge of optimizing for productivity and quality. Almost every profitable factory in the world has solved the problem to a greater of less extent through rigorous definition and implementation of repeatable processes and statistical process control to measure and optimize for quality – key principles in the six sigma approach.
Despite efforts to apply this approach to software development over the years, software development does not lend itself to the factory approach. While repeatable processes are vital, a different product is being produced in every project – in that sense, the process is not repeatable. Hence, the application of statistical process control to optimize quality is arguably a non-starter.

Instead, various studies have shown that the sweet spot for quality and productivity for software development is a small team of 6-9 people working collaboratively to carry out requirement definition, design, coding and testing. This is conclusion is one of the key drivers for the movement towards Agile development. The Agile philosophy goes further by prioritizing to Customer satisfaction over other considerations and acknowledging that the Customers’ needs and desire’s (against which they will measure satisfaction) change over time and in response to the software delivered.

Wonderful! Problem solved! We can have optimal productivity and quality by doing agile. Well, yes – and no. As software projects get larger, the number of teams grows and the overall productivity and quality fall. This is often identified as a communications problem. Clearly, communications between teams is harder than communication within a team and more teams makes this worse. However, we suggest that the problem is as much one of control. The agile philosophy on self-organizing teams and the need for some sort of central command and control function on large projects are in tension. This tension can be managed well with care and creativity (and good coaches) but all too often it causes the failure of otherwise valuable agile initiatives. It must be said too that larger projects can generally afford more rigorous testing. Specifically, development projects that are repeated enhancements to long-standing applications can start to approach the repeatability conditions for statistical process control.

The lesson here is that there is an optimal balance between productivity and quality for any project but it might be different for any given project and is likely to be a lower optimum for larger projects than smaller projects.

Conclusion

The overall conclusion, considering these three perspectives is that in theory quality is more important in the long run than productivity but in practice speed trumps quality. Productivity is only important as a proxy for more speed and measurements of productivity often don’t include the consequences of poor quality.

In this article we have made the case that quality is more important than productivity; therefore, organizations should take better care to properly budget their projects and to properly set expectation regarding the cost, schedule and quality of the software deliverable. The answer here is clearly a need for improved estimating and planning practices.

Sources