Journal of
Information Technology & Software Engineering

Mancas, J Inform Tech Softw Eng 2014, 4:1
DOI: 10.4172/2165-7866.1000e123

Editorial Open Access

Will Software Development Projects Always Risk Delays?

Christian Mancas*

Department of Computer Science, Bucharest Polytechnic University, Romania

State of the Art

It is true that, generally, mankind is still not mastering estimation, although lot of effort has been done in this field too. However, accurate estimations are currently rather the norm than the exception in most domains, from missiles manufacturing to cooking. How comes then that so many software development projects are still behind their schedules, despite the fact that many of the involved people have solid computer science background [1-3].

Generally, depending on available data, they say that between 50% to 66% of the software development projects fail to meet their deadlines. For example, [4] found that more than half of 2011 SE projects failed or were considered at risk for a bunch of reasons, including resource conflicts, insufficient data, and timelines that are too tight. Even worse, delayed projects have sometimes to be abandoned [5]. As another, more recent example, [6] reported that only 39% of the worldwide software projects in 2012 were considered a success by standards of budget, cost, and expected functionality, a whopping 43% were deemed as challenged (late, over budget, and/or with less than the required features and functions), and 18% as failures (cancelled prior to completion or delivered and never used). Put another way, about two of every three software projects are simply not ready when they are released to the customer or an unsuspecting public.

Fortunately, the trend is towards improvement: for example, three years before, a similar report showed that some 66% of the software projects failed to meet their deadlines, as compared to this 61%.

Reasons of Delays in Software Engineering Projects

Main product development delay causes (which also apply to software products) were and are constantly scrutinized by researchers all over the world. Thomke [7] identifies six involved fallacies (and suggests corresponding countermeasures):

(i) High utilization of resources will improve performance.

(ii) Processing the work in large batches improves the economics of the process.

(iii) Our plan is great; we just need to stick to it.

(iv) The sooner the project is started, the sooner it will be finished.

(v) The more features we put into a product, the more customers will like it.

(vi) We will be more successful if we get it right the first time.

Lot of other books and papers [8-11] were devoted to this topic. In my opinion, after nearly 40 years of experience in this field, the following 16 causes of delays in software development projects, in increasing order of their relevance, are the top ones:

- Force majeure
- Gold plating
- Customers not trusting developers
- Working on too many projects at the same time
- Securing project approval
- Choosing an improper platform
- Absence of analysis and design principles
- Cuts in quality control
- Customer delays
- Poor architecture and/or design
- Underqualified personnel
- Overly optimistic schedules
- Functionality expansion
- Poor business analysis
- Too much research involved
- Project complexity

Possible and almost Impossible Solutions

Here are, in my opinion, some possible solutions to part of the above issues:

- Trivially, project managers should allow for gold plating if and only if the customer asks and pays for it or if no other task is assignable to the corresponding developers.

- More than in everyday life, tensions between customers and developers should be always avoided; whenever they appear, however, they should be dismounted immediately.

- Obviously, organizations should always consider whether they have enough resources to complete all projects successfully. They need to consider whether they need to work with an outsourced service provider, especially when they are short on specific skills and expertise.

- To secure faster project approval, we should build a sense of urgency by helping management understand the risks of a lengthy approval timeline. While some things may be out of our control, we can help educate and communicate with stakeholders to get projects started more quickly, thus increasing the chances of on-time completion.

- Whenever choice of platform is possible, we should spend enough time in order to choose the best one. Moreover, we should always encourage and assist customers to upgrade to their best new versions.

- It is obviously necessary that both analysts and designers have up-to-date knowledge of emerging business analysis and software design principles.

- Cuts in quality control should never be allowed!

- Customers should always be explained that it is in their best interest to timely cooperate such that they do not cause delays.

***Corresponding author:** Christian Mancas, Department of Computer Science, Bucharest Polytechnic University, Romania, Tel: 40722357078; E-mail: christian.mancas@gmail.com

Received February 19, 2014; **Accepted** May 01, 2014; **Published** June 09, 2014

Citation: Mancas C, (2014) Will Software Development Projects Always Risk Delays?. J Inform Tech Softw Eng 4: e123. doi:10.4172/2165-7866.1000e123

• Enough time should always be spent for the architecture and design phases.

• Developers should be very carefully selected, on qualitative not quantitative (e.g. x years of Java and y years of PL/SQL experience) grounds, motivated, and continuously trained for the latest methodologies and technologies.

• Developers should always be encouraged to ask others for opinions about their issues whenever they don't manage to solve by themselves within reasonable time (some half an hour, for example). While explaining your issue to a colleague, most often you will find the solution yourself. We should not waste too much time: together, as a team, we just know more. Be proud you dared to ask for help!

• Even the rough estimations should be done based on enough data, only by qualified and experienced personnel, and optimism should be tempered by sharp realism!

• Especially for immature customers (but not only) projects requirements should be taken in writing and some legal bond be filled for requirement specifications. It is necessary that all hidden and explicit requirements are mentioned in detail and contracts be modified accordingly for any functionality expansion (for all related legal provisions, see [12], for example).

• Research should be carried by the best qualified personnel, who should also possess best possible didactic skills for explaining their findings to all involved colleagues in the minimum possible time and with maximum efficiency.

• Complexity should never be underestimated and always tackled with the most advanced methodologies and technologies available.

An example of such a methodology is the Lean Software Development (LSD) one [13,14]. Adapted from the Toyota Production System in 2003, LSD is a translation of lean manufacturing and IT principles and practices to the software development domain. Consequently, a pro-lean subculture emerged from within the Agile community [15,16]. Iterative software development methods (e.g. Agile) do not develop complete software in whole cycles, but develop prototypes and iteratively improve upon them. They suppose shorter software development cycles; hence, software is quickly delivered and then improved immediately. They allow customers to make better decisions after having better understood their requirements. They encourage reducing the number of errors, finding and fixing them as early as possible: otherwise, software defects accumulate and multiply manifold times.

SE teams should be empowered: managers must listen to developers, so that they can explain what actions might be taken, as well as to provide suggestions for improvement. Developers (generally, people) should never be considered as tools: they need to be motivated and be inspired to a higher purpose.

Companies that have employed Service Virtualization (SV) consider that it helps improve the quality of their products, lowers costs, and helps getting to market faster. SV is using in-house software processes to simulate real-world situations that can also be used during development as a reliable stand-in for critical components, such as mainframes and third-party systems. SV provides the ability to simulate how software products will perform in the wild, which allows making improvements on the fly in a secure environment; as it's less expensive to simulate processes, SV impact can be dramatic: it allows for quick, inexpensive test cycles to ensure that projects are ready for production.

• Unfortunately, not only major force is sometimes impossible to cope with; there will always remain other issues with no possible solution:

• Generally, once lack of customer trust occurred, it is very rare that trust is regained, even after tensions disappeared. As they say, building trust is a long and painful process, while for losing it one mistake or even misunderstanding may be enough.

• Almost regardless of its size, no company may generally afford to ignore a new project possibility, even when its workforce is fully occupied: managers will always tend to hire more personnel, even if in hurry, rather than losing a contract to competition.

• Market pressure from ever increasing competition will continue in the foreseeable future to determine managers proposing and/or accepting unrealistic deadlines, causing poor business analysis, software architecture, and design, hiring underqualified personnel, cuts in quality control, and overly optimistic schedules.

• Brooks' law ("adding manpower to a late software project makes it later" [8]) will always remain true (as "Nine women can't make a baby in one month"![8]).

• Even if some customers grow mature in time, there will always spring lot of immature ones; moreover, both of them will generally always ask for functionality expansions in all project stages, very rarely accepting corresponding additional time and fees.

• Nothing can replace research.

• Complexity is inherent to almost any large software development project.

Conclusion

Iterative methods of software development (e.g. Agile) that correct mistakes in the earliest possible stages do help to avoid or at least minimize delays. While Lean springs from manufacturing, the insights that are so relevant there are even more critical in SE.

Dually, note that developmental methods (be them waterfall lifecycle, agile process, or any other) alone are not guaranteeing by themselves either observing deadlines or software quality. Fortunately, there are "little" details we should always commit to for saving time and heartache down the line:

• Truly understand customers' needs.

• Get requirements right the first time.

• Implement the correct architecture.

• Employ simulation tools that help working out kinks before integration.

• Examine the testing and integration phase for ways of reducing cycle time.

Developing high-quality software saves our money in multiple ways, as customers ask for quality: anything that falls short can result in the need to rework code, fix bugs, and sometimes even revisit requirements, architecture, and/or design (by far the most costly of them all). These "fixes" inevitably lead to delays that put off deliverables (and corresponding payments) and cost extra hours that must be invested in repairing the product. The biggest impact of bad software is damage to your brand: even only one screw-up gives customers a great excuse to choose the competition next time around.

There is certainly great room for improvement, but it is undeniable that the SE industry has come a long way in a reasonably short period

of time. There is lot of factors that make SE truly unique. As opposed to any other industry, SE is truly ubiquitous: it is used in almost every aspect of our modern life. It is almost a miracle that SE achieved this much progress and is capable of delivering the results it does.

Obviously, changing SE companies from "artisans" to "factories" is not possible: factories are executing a (sometimes long time ago) established process; their workers execute their process with not that much human thought (if any: when none is needed, workers are replaced with robots). Software development is not executing, but creating processes: SE is much more akin to designing the factory and its processes rather than running it. Although software creation benefits from standards, it cannot fundamentally become a factory.

The software development delays problem is one of expectation: just because methodology and technology are there doesn't mean using them is going to be successful (or wise to use) for a while; if other industries behaved like the SE one did, we'd have black hole powered cars for sale by the end of this year or some "visionary" would have the resources to build a Mars base, also including a McDonald fast-food. The problem is not the SE industry, but the expectations placed on it. While we should continue striving for excellence, which includes avoiding or at least reducing delays, our customers should also understand that software development is creation and, as such, is perpetually condemned to delays: no composer, for example, may be always expected to meet any imposed deadline for writing a new symphony; moreover, it is not in the interest of any commissioner to cause the death of neither a composer (like most probably was the case of Mozart) nor a SE (or any other industry) company.

References

1. U.S. Congress (2013) House Report 113-102 - National Defense Authorization Act for Fiscal Year 2014.
2. DOT&E (2014) FY 2013 Annual Report.
3. U.S. G.A.O. (1992) Software Development Problems Delay the Army's Fire Direction Data Manager.
4. PM Solutions (2011) Strategies for Project Recovery.
5. Ewusi-Mensah K (2003) Software development failures: anatomy of abandoned projects. MIT Press.
6. The Standish Group (2013) CHAOS Manifesto 2013. Think Big, Act Small.
7. Thomke S, Reinertsen D (2012) Six myths of product development. Harvard Business Review
8. Brooks F (1995) The Mythical Man-Month: Essays on Software Engineering, anniversary edition, Addison-Wesley.
9. McDonell S (1996) Rapid Development: Taming Wild Software Schedules. Microsoft Press.
10. Knight J, Thomas R, Angus B, Case J (2012) Project Management for Profit: A Failsafe Guide to Keeping Projects On Track and On Budget. Harvard Business Press.
11. Dewar RBK, Schonberg E (2008) Computer Science Education: Where Are the Software Engineers of Tomorrow? CrossTalk, The Journal of Defense Software Engineering.
12. Stephen FJD (2007) Legal Guide to Web & Software Development, 5th edition. Delta Printing Solutions Inc.
13. Poppendieck M, Poppendieck T (2003) Lean Software Development: An Agile Toolkit. Addison-Wesley.
14. Ries E (2011) The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Crown Business.
15. Schwaber K, Sutherland J (2013) The Scrum Guide.
16. Langr J, Ottinger T (2011) Agile in a Flash: Speed-Learning Agile Software Development (Pragmatic Programmers). Pragmatic Bookshelf.