Delivering Complex Software Projects in large organisations.

A pragmatic approach to Project Management for large scale software projects.

Written by Mario De'Cristofano

Disclaimer

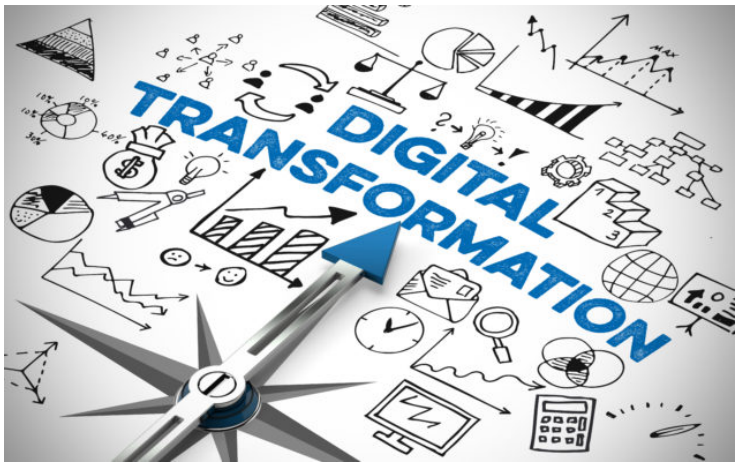
Unless otherwise stated, the article below is written originally & in full by the author & remains the property of the author. All referenced & third-party content is linked with appropriate credit given where applicable. For questions about this article or its content, please contact the author at **mariodecristofano@googlemail.com**

Contents

1. Why this paper?

I've been involved in a lot of web projects whereby some of which have been on a considerably large scale. When I'm faced with new challenges, I like to write my experiences down in a more structured way than a regular blog post. This is one such example. You should note whilst the project is recent, it's been entirely desensitized to protect client privacy. The papers structured into the background, the problem and my recommendations. Some of the topics aren't deep and aren't meant to be, they are subject to follow up blog posts or outside the scope of the nature of this paper. If you have questions or comments, please get in touch with me directly. Issues with any of the content, email me on the address above.

2. The Scenario



The scenario for this whitepaper covers a complex IT landscape amongst a transforming business in a non-technology 1st sector. An old-fashioned company culture then, with somewhat out of date opinions & lack of direction are embedded into the DNA of an extremely complex environment, that of 'doing what we always do' and 'change is tough' type of mentality rules. Business size is in the billions of dollars. Both an extensive bricks & mortar network along with some online capability exists in multi-regions throughout the world & the business is looking to compete aggressively in the UK market with a couple of direct competitors with an e-commerce solution to ship product in a B2B wholesale vertical.

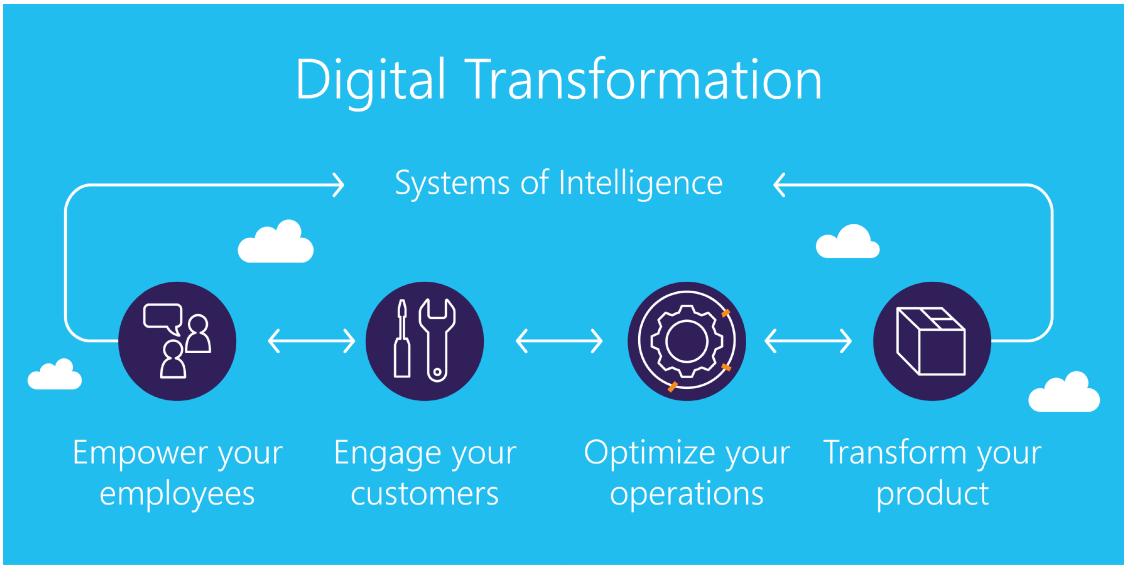
3. The Background

The business runs its centralised ERP, Infor's M3 with the SAP Hybris e-commerce platform serving its website plugged in straight over the top. Situated in between are two middleware layers, IBM's TIBCO and an iterative proprietary web services layer (*referred to by the authors name*) which governs communication to various third parties including PIM's, invoicing & supply chain management tools & receipt and delivery note generation, electronic document interchange (EDI) and other such services & workflows.

The UK part of the global business has its own unique UK market requirements in terms of feature need. Thusly these have been developed by both UK in-house teams & outsourced software teams as standalone code features, webservices and code adjustments, to be then retro-bolted into the company group-wide software solution. This solution sits in a global data centre and is deployed to, in true agile fashion - every fortnight (*or so*) with a new codebase containing core features for the overall solution used by multiple territories. Further this code base is often updated daily with minor changes as matter of routine. As both the overall solution and UK specific features are developed out of sync and independently by different teams rarely

when merged does this solution work correctly. Think both Agile & Waterfall being melded together (*see later in this document*) badly. We'll call this methodology '(F)ragile'

The environment is a mixture of UK & European tier 1-3 data centres hosting production and 'pre-prod' (*the universal term the company uses for anything not production or live*) environments sitting on a few years old and a few licenses back (*for cost reasons*) Windows estates, (x86) with AWS (*I presume Puppet maybe*) self-deploy and cloudfront (AWS) to distribute content via a CDN. Hybris in one data centre, M3 in another, alongside the TIBCO middleware.



Further, and in terms of testing, this is often done in the same environment that development occurs on (*assumed for cost, simplicity and legacy purposes they don't have a structured Sandbox > Development > UAT enviro*) meaning a constantly changing core environment impacts the ability to conduct effective end to end testing in any such case. Automated testing rarely works because on-page elements constantly change or features are switched on and off to a constantly rolling timetable, meaning effective testing is difficult. Local network restrictions throughout the group mean even web developers & coders can't see on-page rendering accurately because everything from JS to other web based scripts are all blocked making visual testing next to impossible.

Finally, the overall solution is developed by a giant 3rd party vendor in a 'perfect' non-real-world environment with a subsequent perfect M3 install nothing like the working environment. Often, when that code is signed off and deployed to the actual real-world M3 environment issues occur, which highlight an issue with the current production environment & the sanity of the data within. Or so I believe!

The UK development facility is somewhat entrenched with both a jaded fatalistic attitude and a lack of internal governance means procedures are unmanaged & out of control. Code is developed straight in Visio Studio, compiled on the fly and uploaded via FTP. No version control, no centralised repository and high risk of code overwrites each and every day.

As this project has several components, the delivery of a content management system driven website to sell products being one of them, the whole web development cycle also needs to be

followed (at the same time via an offshore team), through wireframing, prototyping, UX and UI consideration to actual design, page template build, device, regression & usability testing. This is happening (*in various degrees of competence)* simultaneously inline with the overall code build with an effort to marry the two later down the line. M3 and supply chain/logistics being seen as a separate collective workstream to front end website, static content & marketing & SEO.

4. Core problems

Like any business going through transformation, change is tough. A mixture of cultures & generational expectation, language gaps, skills and experience disparity, political complexities along with market strains such as digitally savvy competitors all piles the pressure on. Further, pressure changes people's behaviour, and collective culture breeds similar behaviour to cultivate a hot bed of bad practice & poor behaviour. You have here then a perfect environment to fail.

Leaders squabble as everyone grabs land, trying their best to both protect their position and attack others if it means getting ahead. This poison is rife and in part due to the lack of direct leadership from the top.

A point to note is the spectacularly bad blending of both agile methodology used by the overall group, for software development, and the waterfall six sigma style approach of the UK business. Melding of the two is akin to trying to graft a pig's ear onto a horse.

5. Current state of play

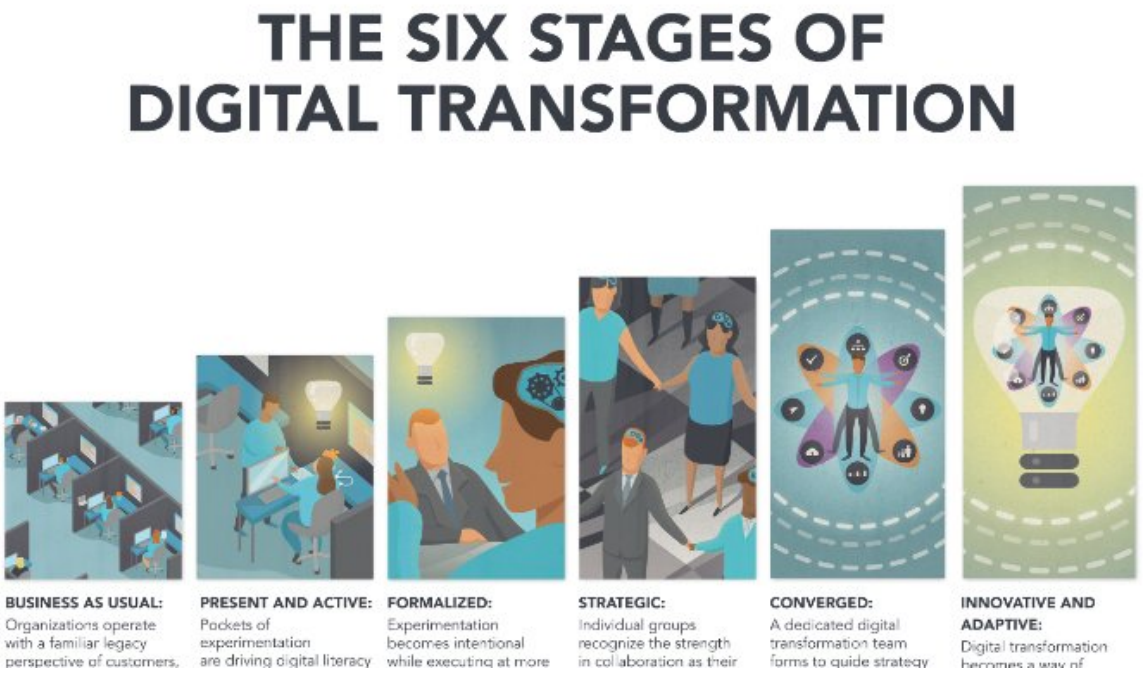
A UK market version of the product is due to be released imminently with people drafted in to facilitate that delivery, however is in no way near to time or budget. Resources are burned through, being ejected at the other side like a hot spent shotgun cartridge. Morale and the health & wellbeing of those involved is at an all-time low with very little in the way of authorative decision making capability from any one in a leadership position. Not entirely their fault, complexities of multi-headed management breeds unmanageable complexity and a risk appetite of the people in charge of the strategy along with realising that strategy vs how the business (*as a large beast*) works I don't believe are in tune. Project perspective is also often lost in translation as generally the project is relatively low risk/impact commercially speaking, but very high noise throughout the organisation. That very noise impacts the project massively and how people behave within it. PR is needed, the project has very bad PR ongoing.

6. Cultural & business expectations

The culture appears to be that of 'get it done' and 'compete as soon as possible with the immediate competitor' along with the favourite old 'we need to be digital' line, but with no real investment into doing things at reasonable time scales or to best practice. Some say the result is more important than how we get there and rumours such as performance related bonuses and back room agreements are all quipped around various members of the team into the why we are in this position. Mostly though, quick decisions with lack of detail up front is now burning the organisation as considerable flaws in the plan (*or as I like to call dead bodies*) appear with alarming frequency. The business doesn't have the long sight at the minute to consider using OPEX capital to plan (a bit like the lazy squirrel story – *one spends the summer gathering nuts for the winter only for the other lazier squirrel to go hungry through lack of longer term preparation*) instead choosing to move down a path of what is the most resistance, using the excuse that 'change is tough' and 'if we don't do it now we never will'. Even at the n'th hour, the business

continues to pour resource & hemourage cash into a project almost seemingly destined to fail. Quasi inter-country battles over who should be responsible for delivering the solution also rings loud, in particular cultural sensitivities between both the French & the English are genuine problems in this project. Foreign language exchange is poor, the French off-shored against the Americans, off-shored against the English, off-shored against the Indians…off-shored against….you get the picture. Again however, the size of the project and its relatively low risk/impact are often missed and/or mis-communicated. Generally communication is bad.

7. So what to do next.

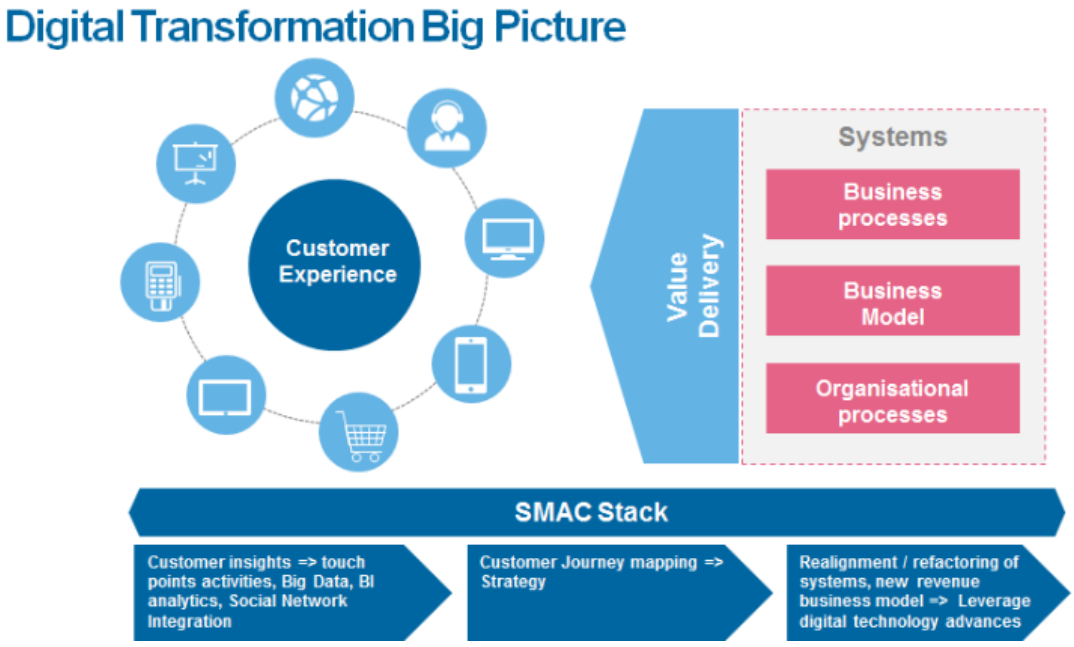


Stop. Stop with the braking force of Lewis Hamilton smashing his Apline stars covered foot into his anodised aluminium pedal of the Mclaren MP1 going from 200mph to 0. (*congratulations on the GP championship by the way Lewis!) Take stock. Breath.* Understand cultural and political divides needs to be strengthened, build bridges. Ignore those issues at your peril. *Communicate. Communicate more.* What are we trying to achieve? Recognize we are not experts in this field but there are experts in the business. Utilise them. Give them autonomy and build a team with the right people in the right roles with the right levels of autonomy to get stuff done. **Set up a skunkworks**. Protect the business from its new digital requirement (*for now*) and build in a silo to free up time to breath, play and learn. Ultimately the ability to make mistakes at this stage is important as there's no other opportunity to do so. Use the existing project as a case study. *Learn from it. Invest in internal auditing, invest in external auditing. Benchmark your workflow.*

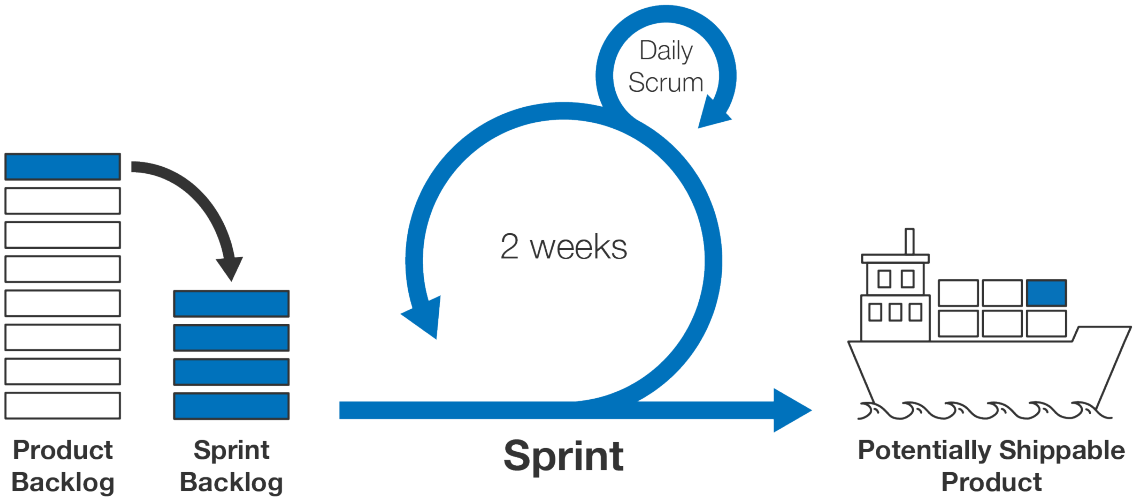
My plan falls into several distinct parts;

6.1 Set the team up, communicating the mission and the requirements. Set up a decent PMO facility with structured governance, and ensure minimum documentation is created (*regardless of the agile world we like to convince ourselves we work in*) so that's SOWs, Functional & Technical spec's, wireframes, the whole nine yards are all completed BEFORE work starts. Get both internal & external auditing in place. Always sensible for a public company with shareholders to feed.

6.2 Structure the people, the right people in the right roles with a single leadership from the top. Remove dead wood, political fighters and those who are incompetent. One person. In charge. Buck stops with them. Ring fence this team, make it a skunkworks. At this scale, this will help. Silo it. Protect the business, protect the people, take the pressure off. Use a properly blended agile methodology for day to day software deployment (see fig1) but waterfall/Six Sigma toll gate for the business communication as that what the big business craves. Weekly calls, minimum decision-making stakeholders from each section of the entire business, follow up notes, recorded minutes. Quartley summits and monthly town hall communications tracking progress to plan. Everyone participates. No option to duck out. No option to pass the buck. Small teams. No excuses.



6.3 Create infrastructure first ready to accommodate two versions of the group software. If we already know every two weeks a code drop will change the overall core code base, let's encompass that in one environment, (we can call it UK1) and let's have a UK centric version which contains the latest version of the group software but doesn't receive the ongoing two weekly updates, and we'll call that UK2. UK1 continues to receive the code releases, these are managed, tested & the agile process interfaces with the way the group want to work without an issue. Use GIT, a private centralised Repo, or get everyone working in Visual Studio with TFS – whatever the workflow, use Version control & proper collab. Tools such as Basecamp. Enforce it. Audit it.



6.4 Build & code the UK centric features on UK2, with each feature being a micro-project on it's own. Don't run them side by side, run them waterfall to ease pressure on the team. Simultaneously whilst this technical workstream is happening, ensure the entire supply chain, logistics and internal operational processes are all fully mapped, understood & with any touch points audited. Change the processes or modify them ready to accept the digital product at this point. Horse first, Cart after, like it should be.

6.4.1 Pass the Baton (in the right order)

One of the biggest issues in complex software development is the issues around managing multiple groups of people all working on the same engine but in different parts. For simplicity, we'll use a car analogy, one team working on the air conditioning system, one team working on the electronics and one team rebuilding the piston chambers. All key to the engine actually working, but all very different teams. Assuming there's a natural order to this, so electronics mean diddly squat if the piston chambers aren't built, then you assume do that first, before you call the team in to work on the electronics. Same with software development, ensure the group deployment is fully communicated & tested, before the baton is passed to the UK team to code individual features. Sounds simple, but missing this clear communication really impacts multiple development streams. So I'd want to facilitate this multi-party development work using something like a perm. Google Hang Out, Slack Channel or a Skype channel set up. People can dip in and out, but the overall narrative is all captured in one channel.

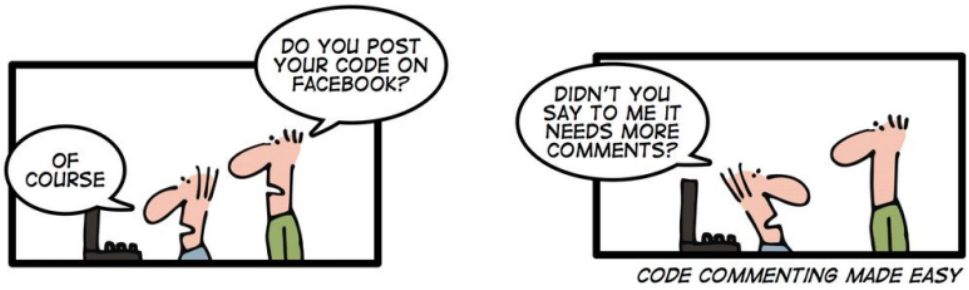
You know who get this large scale collaboration over a complex solution right? F1 teams. See how they do things. Even better, get people with experience working in teams like that. I know people. You know people. Make it work.

6.5 Despite point 3 (products), assuming an e-commerce project involves selling stuff, get your product catalogue ready at this point – the one thing you can do simultaneously whilst developing features as it's somewhat standalone and non-reliant. Using a PIM? Now's the time to populate it. Product categories, descriptions, photographs. Get that done. This is a milestone. We don't progress until your products are ready. Make no mistakes on this simple work stream, photographs & such like are all easy quick wins to do properly, go & get them done, and get that work completed right, 1st time.

6.6 Content. Write it, author it, proof it, sign it off. This step falls into general web design (& you can read here about that). Get your prototyping complete, make your page templates work, build your content and get it all ready to be populated. This is a milestone. Map the site structure. Document it.

6.7 Testing – Build your testing environment. Get native devices, don't rely on emulation. Like making your own network cables, it's a false economy. Get a sandbox environment. Get a UAT environment. Stabilise and harden it, and back it up. Get your individual features tested. Unit, functional and regression. Then automate them. Each feature, deploy it as a standalone feature into the last version of the group software. Test it again. If it breaks, rinse repeat (*but you've not broken your core environment and you also know where the problem lies. i.e – the last UK feature*). Do this with each feature. Time intensive, but saves time later down the line. No one likes mixing a bunch of code bases developed by different people and then picking apart what caused the issue. Simplify the process. This process is also greased well by passing the baton (see 6.1).

At this point you have all your web features developed and now is the time to start writing any interfaces with third parties, web services and re factoring anything which needs it, to work with those web services. Document those services, calls and structures. Now's the time for SIT testing, more regression testing and as a milestone, you want to finish here with a working solution for the UK on UK2, encompassing the last weekly update from the group deployments plus all your webservices integrated & documented. You can choose at this point to deprecate UK1, or move the contents of UK2 (*your success platform*) to UK1 and keep them mirrored for redundancy prior to deployment. At this point, both UK1 and UK2 now receive the two weekly core code updates and the UK specific features all work and behave well.



DOCUMENT YOUR CODE. Enforce it. All code, documented. Subroutines, APIs, everything. If there isn't comments, it gets bounced back to the developer & revisioned. Do NOT skip this step. The code needs to live on beyond the individual. Especially important when dealing with segmented code from third parties.

The next micro project is payment gateway integration. This can always be done standalone. Pick your gateway, (*there's not that many globally*), define your rules and integrate it. Anything which breaks now, it's the payment gateway, you know this because everything else was working up until this point. Fix it on UK2, you now have UK1 as a recent backup continuing to take two weekly deployments, and you can roll back to that until you get things right.

Test with the business. Test each operational area with the now barebones of the site which should have been plugged in. Page templates populated and UX and UI signed off. Now you can start doing closed user group working groups. Testing the UI, testing on mobile devices, seeing how the product interacts with the business. Tweaking the process here, there until everything works. You're now working at the visual layer, so CSS, layout & UI changes are possible and can be done cleanly without impacting the core code base.

Plan your phased BETA approach. Select your project champions. Sort out your project comms to the entire group. Deploy as a BETA, for X period of time. This is a topic all on its own so I won't speak more on this here.

The overall process for this particular scenario is above. It's designed not to be super-lean, (*although it is structured*) but it's designed to allow for error and allow that error to be easily tracked in a large complex organisation, just like how F1 teams work. Software is released in a structured narrow way, with certain aspects not tackled until others have been addressed. *You can't have a baby in a month by throwing nine women at the task* & it's the same with this type of project.

Here's some summary do's & don'ts;

Don'ts's

- Don't develop too many complex features simultaneously

- Don't be defensive, work together collectively and use Agile to work quickly through software development tasks
- Don't ignore issues and short term plan.

Do's

- Deal with language & geographical barriers from the outset
- Plan for long term success by doing time intensive tasks correctly to start with.
- Define your acronyms and language, and stick to it consistently
- Get auditing in place both internally & externally.

Software product launches are fairly straight forward and again out of scope of this whitepaper, but let's talk briefly about tools for a project this size.

8. Tools of the Trade

7.1 Software Tracking

JIRA, no other choice really. Works well. I'd arguably also run it with Microsoft Team Foundation Server utilising the code release & feature management tools. In fact, Team Foundation server is a must in any .Net project at this scale considering its tight link to Visual Studio.

7.2 Testing

Webload, Selenium with Cucumber, Loader.io, Soap.ui all good for testing, stick with a number of these and you'll be golden.

7.3 Project Management

With Microsoft Project being the forerunner, tie that up with Basecamp. Great tool for Comms and with independent access it's great for working with third parties & covers all work streams. Use Trello day to day for stand ups and task summary management. You can read about tools a little bit more in my other white paper here and also, if you want to learn how to use MS Project, read my guide here.

9. Finally, a note on Project Management.

You'll need a bloody good orchestrator for this symphony so don't skimp on Project Managers. Good ones, those who know software & are comfortable ball's deep at the busy end of deployments. Make them '*people*' people, this project (*like all technology projects*) are more about people than the technology, so get that right. Strong planning & documentation along with governance and process should all be followed. Following the process will help stop the project from getting out of shape. You can either skimp on process if you have people who can work together well amidst chaos, or skimp on people if the process and documentation is robust & appropriate. You can't skimp on both.

Get a lead PM or Programme lead, with sub PM's managing the various work streams. Make them web-disciplined with several years of ideally hands-on experience. That means the Digital PM for the content has actually built content, the Digital PM for the code base has actually coded, spend time up front finding these people or calculate the net loss of every month you're delayed through incompetence. Then make the call. Good luck with the shareholders!

You can read about my Project Management manifesto here, and what to do if you're implanted into this type of project half way through here and if you'd like to get in touch please do. Looking for a Project Manager, ***I know a few. Maybe I can help.***

@mariodc on Twitter – grumpy, sweary & always ready to have an opinion.

External Links

How to plan Large scale Technology Projects on Time & on budget – McKinsey

https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/delivering-large-scale-it-projects-on-time-on-budget-and-on-value

Why your next large software project needs to start with change management

http://www.ellucian.com/emea-ap/Blog/Why-Your-Next-Software-Deployment-Needs-to-Start-With-Change-Management/

Agile Software Delivery Methodology of Large-Scale Software Projects

https://pdfs.semanticscholar.org/ea63/55028d31f3842affaf80b9274c4f53f5bab4.pdf

How to Document a large software project

https://www.smartics.eu/confluence/display/PDAC1/How+to+document+a+Software+Development+Project