

The Top 10 Ways Software Projects are Different

By James Bullock

Max Wideman responds

James Bullock:

I have spent a lot of time "explaining" software projects to non-software professionals: customers, support organizations, project sponsors, sometimes project managers. A couple years ago I helped a PMI trained project manager, a mechanical engineer, who ran plant floor install projects for years - recover a floundering mission-critical software project. Now that it's shipped and routinely in the Media Metrix top fifty, she manages most of the software development projects in that shop. She primarily needed a little insight into how software projects are different from other projects. That plus some software vocabulary and an occasional sanity check on a task, technique or work product was all it took to help her succeed. Software projects are a little different, but not all that much.

Just last month an experienced project manager assigned to his company's software Project Management Office asked the NewGrange discussion list "How do I learn about software projects?" (Note 1) Although I have been explaining software projects to non-software people for a long time, I never recorded my answers until then. My initial four-item reply grew to ten. So, here are The Top 10 Ways Software Projects are Different for non-software project managers.

Max Wideman:

It is always valuable when people are willing to come forward and take the time to document their practical experiences. Note, I emphasize "practical".

James Bullock:

Software is Different Stuff. The biggest difference between software and the products of other kinds of projects is it's not physical. Software consists of ideas, designs, instructions and formula. Creating software is almost entirely a cognitive activity. The stuff we can see and measure, from code files (How strange is it to use a collection of arbitrary symbols visible through a computer as an example of something relatively real?) They stand in for the real stuff, vs. the other way around. Still, software only matters when it appears as something real, even as barely real as colored squiggles on a computer screen. Maintaining that connection from thought stuff to real stuff is one of software's peculiar challenges.

Max Wideman:

Yes, we all like to think that our particular specialty is "different". And indeed, for the most part they truly are, otherwise they would not be a "specialty". But are they really so different to the project manager, relative to all other types of project as Jim suggests? Let's take a closer look.

The first trap that many people fall into is that they do not differentiate between managing the technology and managing the project process. Certainly there is a lot of difference between managing how concrete is placed and how the interface in a software development is placed on a computer screen. But how much difference does that make to managing either project in terms of scope, quality, time, cost, risk and so on? You can see both, even though you may not be able to see the technological processes behind either. (Has anyone ever sat around watching concrete

harden by any chance?) Nevertheless, yes, there are differences but they have to do with the sophistication of the technology involved and the types of people needed for the work. For example, construction is considered to be "well established" while software development is "recent, and technologically advanced", but is certainly not unique in this regard.

So, on this basis, let's examine each of Jim's points.

James Bullock:

1 - The artifacts in software projects often aren't as visible or well understood as in other projects. Since you can't kick a chunk of software like a brick, you have to create ways to see that it's there. Tracking progress is often the hardest thing in software. Some big-bang software projects fail because early artifacts like designs weren't as done as we thought they were. We're clearer on whether a factory blueprint is correct, complete and useful, or even present than with a software design. Some so-called agile projects fail because they don't have a repository or any meaning to "done". Producing transparency in software projects requires two things that we get for free with a brick. First we need a surrogate for the invisible thought-stuff. Then we need visibility into the state of the surrogate. So, put stuff that's "done" somewhere consistent and visible. Create a customer for each item and check whether each "brick" meets their standards of doneness. Successful software methods, tools and management are as much about maintaining transparency as about getting the work done. Because software is thought stuff, intelligent testing is harder and more valuable in software than with many other kinds of artifacts. Testing software projects creates surrogate customers and checks for "doneness." Incremental delivery and early tests can force more visibility into the state of bricks and blueprints along the way.

Max Wideman:

Well, item 1 is all about the technology and not about project management. So if we are comparing with construction, the need is for brainwork rather than craftwork and that certainly needs a different approach to managing the people. But I would say that exactly the same point might be made for research and development, for example.

James Bullock:

2 - The end state of a software project is often a lot more speculative than with other projects. Sometimes we have at best a vague notion of what we want. Sometimes our notion is clear but we forget to tell the other guy (often with implicit requirements). Often, we change our mind along the way. Because software is a relatively new field, software projects almost always involve new art - applying software to a new or expanded problem. We are learning what we want to build as we build it. There are good cognitive reasons why learning what software can do provides a platform for discovering additional uses. The goals of a software project will change along the way for many reasons, but it is a bit harder to notice changes with software than with a wall that's halfway built. So, software projects must aggressively manage what's being built. Incrementalism and iteration help a project include new values and new challenges discovered along the way. They also help flush out implicit requirements while we can still address them. Implicit requirements can have huge impacts. Along with incrementalism, testing helps expose new uses, implicit requirements, and changes to what is being built. Often, it makes sense to develop and test software on a defined-investment vs. defined-deliverable basis and consider another bounded investment based on what we learned.

Max Wideman:

We are still talking about managing the technology. True there is an element of uncertainty, perhaps risk that is greater than might be expected in construction, although in construction the stakes are typically much higher. However, the same arguments can be advanced for, say, film making. You shoot a scene and upon review have to come back and modify it. Even then you don't know how viable the final result is until you roll it out before the paying audience.

James Bullock:

3 - There is an incredible variability in what we call "software" and projects called "software development." Designing a Formula-1 racer is different from designing next year's Camry, which is different from designing GM's new fuel-cell multi-vehicle platform. They all have to do with designing cars yet we treat them like vastly different activities. Somehow we try to treat making software as the same when the products are as different as Camrys, racers, and platforms. Software isn't really like making a car anyway. Software is more like steel - it's a material. However, making software is more design than production, and designing software for different systems can be as different as designing a racer, a Camry, or a new platform. The systems solve different problems, to the software design process is different for each. With all that variability I think universal software methodologies reduce to the tautologies: Do what makes sense. Track the work. Ship things. Your software project is possibly very different from the model project or the last project. When your project isn't tracking the model, believe your project.

Software Production: Even with visibility into delivery and stable targets software production often varies wildly and often invisibly, in part because "software production" can also mean many different things. So a software project manager must work actively to understand how production is behaving on their particular project.

Max Wideman:

I have worked on many different types of building projects (as well as software projects) and I think I can advance exactly the same arguments so far as building projects are concerned – especially when you are dealing with "innovative" architects who don't seem to know when to let go!

James Bullock:

4 - The production chain from feature to code to executing stuff varies wildly in throughput, availability, reliability and even variability itself. Software production varies from one application technology to another, from one platform to another, and even between deployments of the same production chain. Software production varies wildly between individuals and with changes to the environment they work in. With software, we are not guaranteed that we can make bricks from clay at any given rate even when we were making bricks just fine last week. A software project can also involve discretionary or forced changes to the production tools and methods, which change production as you go. The Hoover dam used a novel method to speed up curing concrete by a factor of one hundred or more. That kind of change in production processes is normal with software projects (although sometimes in the wrong direction.) So, tune project projections to the production you are getting including rate, rework, and additional activities. Incrementalism pays off here again - you get to calibrate as you go and notice when things change. As a project manager, replanning a software project isn't a formality, it's required.

Max Wideman:

Believe me, I've experienced the same sort of inconsistency on projects in the developing world – and there we were only building earth dams and brick weirs, all by hand, of course.

James Bullock:

5 - Producing code varies wildly in the character of the work. Code production can be mostly any one of: discovery, investigation, design and creation, generation of code, integration, or something else. If your project is mostly remediation like Y2K, "code production" is production-like, and process-manufacturing models can apply. If your project is mostly new system development, "code production" is more like what we call design for manufactured products. Manufacturing models don't apply to design even though you are getting lots of code. Design and discovery in code production can be repeatable in the sense that writing a newspaper column is repeatable, but not in the sense that typesetting edited copy is repeatable.

So, realize and deal with the kinds of code production you have - probably several, probably changing during the life of your project, and maybe a couple kinds at once. Most methodologies assume one particular kind of production process, which fits reality only some of the time.

Max Wideman:

Indeed, this is very much akin to research and development of pharmaceutical products. It takes years before you know if you have a winner and then only after going up any number of blind alleys.

James Bullock:

6 - Software production involves a lot more than producing software. Production processes on the front-end (before features) and back-end (after something that executes) are more variable than code production, and often dominate the project schedule, risk and variability. Non-production activities can also dominate the project, like organizational change, team formation and technology adoption. Worse, "complete" code doesn't mean the same thing. For example, some environments handle deployment for you (more or less) while others don't, so code that is "done" involves different back-end work. A software project may involve changes to the way we produce software - like inventing a new way to pour concrete that sped up building the Hoover dam. Often when we change the scope or reach of a piece of software, our methods for testing and deployment have to change - which doesn't show up as production proper. To succeed with software production, you must manage the parts of the project that aren't software production. When replanning software projects whole tasks or chunks of scope may appear or disappear, along with adjustments for productivity. When an activity comes up that's not part of the official approach don't exclude it, but feature it most visibly.

Max Wideman:

Infrastructure projects can become very complex, too, with complex and highly integrated work breakdown structures. Now there's a thought. How many software architects or engineers develop a work breakdown structure for their software development projects? If they did, would it help?

James Bullock:

Managing the Project. Managing a software project is more managing and less administration than in less variable domains. Managing a software project is a continuous negotiation with customers and sponsors, but also with the technical team, suppliers, and support organizations.

Max Wideman:

True the stakeholders involved in software projects require a lot of "handling", but so do the stakeholders impacted by linear projects like highways and transmission lines. Just let them try putting a new road, rail or pipeline through your neighborhood!

James Bullock:

7 - There aren't any software-only projects. At minimum software needs some hardware to run on, and somebody to use it. The hardware must be specified and deployed, and sometimes invented. The people using the software have their world changed - that's the point. This applies to shrink-wrap and embedded systems as well as the obvious IT systems. So, if you are managing a software project you either have to own the non-software parts of the project and be responsible for the whole value proposition, or you need clear interfaces and hand-offs to the other parts. Interfaces and hand-offs are a bit harder because software artifacts aren't so obviously physical.

Max Wideman:

Exactly the same applies to any construction project. You have to have land to put it on, and you have to make sure that land is suitable in all respects and then prepare it to receive the constructed edifice. Similarly, you have to test the final works, commission it, and often market it as well.

James Bullock:

8 - Software development capability is usually a limited, shared resource. Rather than get a bunch of cement trucks to pour our factory floor on our schedule, we get what we get. Often developers are involved with several systems. Add the fact that we're always thinking of some other thing we could do with this system and software production is always oversubscribed. As with most thought-workers, multitasking developers devastate productivity and morale as well as being untraceable. So, divide the work and treat even a team dedicated to a single project like a job shop. Within a single product or system we use iterations and releases to make chunks. Software is uniquely suited to incrementalism because it can often deliver incremental value. A manufacturing plant may not be of much use until it is mostly there. Software can often be of use when it's only 10% there, if you pick the right 10%.

Max Wideman:

The same can be said of "business" or administrative projects where software is not involved, but a change in culture, e.g. attitude, definitely is.

James Bullock:

9 -The tools and methods in software projects are tunable. So, if rebuilding the software every five minutes helps code production, adjust your environment to do that. If you are dependent on data in a database, but you can't get at the database finesse that - take a copy or fake a copy. If pushing the software into 10,000 ignition systems is slow and expensive, tune your methods to get it absolutely right before you burn a couple million dollars and months of market time. You might want to pilot burning chips so when the real deal happens you have a scaling problem - 10,000 is a big number - but know that the function is right and the burning works. You might want to invent a way to burn 10,000 chips in parallel. Software tools are more malleable than the tools for bricks or cement, so pay attention to what's hurting and fix that. Inventing new art in software tools and techniques is both a common need, and a common failure. It's just as wrong to leap on the sexy new way as it is to stick with the failing old way. I like to iterate processes and methods with each project increment - "How can we do the next one better, faster, cheaper?" then track process or method changes explicitly on the project plan.

Max Wideman:

We seem to be back to managing the technology rather than managing the project process.

James Bullock:

The Software Project Manager's Job. So, software projects are sort of uber-projects - just like non-software projects but more so. With software we don't have a lot of the crutches that help make other kinds of projects successful, like most people pretty much know what a factory looks like, can count bricks, and we have a lot of experience with producing and building with concrete (2,000 to 2,500 years for sure, arguably more vs. less than 60 years for software.) Given all of this, item 10 should be obvious.

10 - A software project manager is employed for a reason. Projects this challenging need leaders and managers, not just administrators. You are there to manage the project and support the people who will have to deal with changing targets, multiple demands, variable production, accidental (or deliberate) changes to their world, multitasking, new methods and so on. There is a tremendous temptation for project (and line) management to retreat into administration. The methodology said . . . Those bad and evil customers, developers, vendors, agitated ghosts said. . . The PMO requires . . . It's too hard, complicated, confusing, obscure . . . It's your job to make it happen. If you can't figure that out (maybe by

finding a trusted adviser) then your job description is inoperative.

Max Wideman:

I don't think I have ever met a project manager that was not employed for a reason. Nor do I think I have met one that was successful yet not a leader and manager, and a good administrator and facilitator as well. True, project managers suffer a lot of abuse, but part of their role these days, is to "educate" their principals in the art and science of project management. And, for that matter, also educate them in program management and project portfolio management.

James Bullock:

This advice is also for software project sponsors, and the managers of software project managers. Put your best, most engaged and flexible people on managing software projects. Let them manage. Expect a lot from them but support them, or their job description is inoperative.

Max Wideman:

Amen to that! In fact, this befits all areas of project management application.