Hundred Person Project

The purpose of this little thought experiment was to express in an amusing way the notion that projects with large staffs may not get any more work done than projects with small staffs. Large projects (as opposed to large problems, with thanks to Alistair for the notion) do seem to have very high overhead. This would suggest that the same amount of actual development could be done by a smaller project in the same amount of time.

ChetHendrickson and I were talking about this today while preparing our Smalltalk Solutions talk. Chet, fix this to be better if you like ...

Imagine a hundred person project, scheduled for a year, or 200,000 person hours. Of the 100 people on the project, how many are actually programmers? We guess maybe 40. So there will be 80,000 programmer hours.

How much programming will these programmers actually do? (In XP terms, what is their LoadFactor?) In a conventional project,they clearly spend the bulk of their time preparing for meetings, attending meetings,writing documents, writing comments, and reading documents to figure out how to write their code. We think their LoadFactor is about ten, that is, in a typical week, they actually spend a half-day coding.

AlistairCockburn asserts that the LoadFactor couldn't possibly be that bad, and has visited projects that report more programming than that by far. 30 or 40 hours in a week, even. If this is true, the reasoning below is not valid.

Don't all programmers REPORT 40 or more hours of programming a week?

Therefore there are 8,000 actual hours of programming, per year, done in a 100-person project.

Alistair believes that we can't convert our measures of actual development time to real time into estimates of how long development will take. However, that's what we do all the time. There is some communication mismatch here, I believe. If we can't multiply programming time times LoadFactor to get elapsed time task estimates, then what's below isn't valid.

Our XP developers have load factors of 2 to 2.5 by actual measure. Therefore you can get 8,000 hours of programming done in 20,000 real hours. That's ten people, one year.

A HundredPersonProject is a ten-person project, with overhead. Doctor, it hurts when I do this. -- RonJeffries

---

I like the math, but I'd like you to try it another way. What if the question was not how much programming you get done, but how much you learn (this fits with HowardBaetjer's ProgrammingIsSocialLearning idea)? How fast can a 100-person society learn compared to a 10-person society? -- KentBeck

See LargeProjectLearning. (And fill it in ... I can't, yet.)

(Indented stuff is InterpolatedComments. I guessed the attributions. I inserted new ones because I think every change of author needs a fresh attribution, to emphasis that change. I removed some formatting. This is something of an experiment, to see if it is any clearer. -- DaveHarris)

Suppose the project is to produce a Smalltalk framework for other programmers to use. Or suppose you are in the position of Sun, having to create a raft of libraries for a new language or a new operating system. Say you have 10 programmers on the project but 100,000s are going to be working with the code you produce. How do you deal with that?

My guess is that you see the code as the product, and you do produce things like documentation (and maybe even comments?) although it's for the external client programmers rather than for the internal team. -- DaveHarris

- The product is whatever the customer wants. Certainly we'll do a manual when we need one. And if comments were useful (as in your scenario they might be) we would do them. -- RonJeffries

A flip side of the question is, how do XP teams deal with the 3rd party frameworks that they buy in? Say you've got this GUI library full of neat widgets. Hey, at the end of the first 3 week iteration, 95% of the widgets are not being used! You don't need 'em. Delete the source. -- DaveHarris

- Of course there is a difference between how we treat the code we write: we refactor it, pat it, prick it, mark it with a C[3]; and code we buy. We don't refactor the base image, we extend it. Ditto other 3rd party code. Reasons available on request, but I suspect they're obvious to us all. -- RonJeffries

- Would it make sense to split a group of 20 into two groups of 10, with each group treating the other as "3rd party" and/or client? Is this the way to scale XP? -- DaveHarris

- *The difference is that 3rd party code comes complete and ready to run. If both sides are evolving rapidly, the interface might (read will) become the problem. My customer (the other team) needs things now. In XP, we are the other team, and we fix the other class if it needs it. So this would be a big deviation from the XP "philosophy", I suspect. -- RonJeffries*

By the way, I'm putting forward the (old) idea that the way to deal with a 100 person project is to split it into 10 10 person projects. -- DaveHarris

No, the way to deal with a 100 person project is to fire the bottom 90% of the staff.

I wonder about that also. Is XP suitable for production software as opposed to in-house development? Seems like you need clients there (when you develop code for retail sale), or do you DogFood it yourself? -- MichaelFeathers

ClientPresence is an important part of XP. It replaces huge requirements documents, with their large cost in time and clarity, with presence of someone who actually knows and can decide. This is consistent with Baetjer's notion that software development is very much about mutually discovering, with the client, what is needed.. -- RonJeffries

I have more than a dozen years experience in packaged business software. It never worked unless we had real clients that were really available almost any time for consultation. Only difference is we usually had more than one flavor of real client, which has its own problems - they seldom agree. --Bob Haugen

---

I have seen two different groups that did the same thing in Smalltalk. Both were interested in making a framework and products that used it. Both the frameworks and the products were similar, at least they solved the same problems. One project started with two gurus who gradually grew the core group to five or six, but added on dozens more to make applications. They spent a lot of their time teaching people Smalltalk. The other started out as a 100 person project, with 15-20 people devoted to the core frameworks and the rest to building applications. Although both groups had plenty of problems, the first group was more successful. The people I talked to on the second group said they would have been done long before if they had just done it with their best 15 people instead of with the entire group.

One problem is that it is hard to get a large group of really good people, so the average experience and skill goes down on large projects. Also, large groups require several managers, and they can start having political battles with each other. In a large group, some people are sure to leave before the project is over, and that slows things down a lot. If someone leaves a small project then the damage is even greater, but you can work harder to make sure that nobody leaves and you have a good chance of keeping them all

sure that nobody leaves and you have a good chance of keeping them all.

Large groups produce more code than small groups. The small project produced half as much code as the large project, but it seemed to me that their system did at least as much, if not more. The small project had a more sophisticated design at its core, and built simple designs for things that were not at its core. The larger project had a less sophisticated design for the core business objects and had a more sophisticated design for technology objects that did things like persistence and workflow. I think this was because the designers on the big project had a harder time figuring out what the core problem was, so they would carve out pieces that they understood and work hard on them. Even though the small project was not really following ExtremeProgramming, in this case it was following DoTheSimplestThingThatCouldPossiblyWork.

It is an exaggeration to say that a HundredPersonProject is a ten person project with lots of overhead, but sometimes it is true.

-- RalphJohnson

RalphJohnson's example sounds like an example of ConquerAndDivide.

-- SteveAlmond

FredBrooks in The MythicalManMonth talks about overhead in large projects. He also points out that good programmers can be 10 times as productive as average ones. (I've known programmers whose productivity was negative.) -- DaveHarris

---

Seems to me that "Client presence" is important enough to be "ClientPresence" and warrants some explanation of its own.

I got into a bit of a disagreement this morning with another developer about XP and the minimal requirements documents it advocates, and I don't think I really convinced him. If anyone could provide some more background on why ClientPresence is superior to a BloatedRequirementsProcess I, and probably others, would find that very helpful. -- DavidRosenstrauch

Actually, after further reading, it looks like ExtremeProgrammingMayScaleUp provides some ammunition against a BloatedRequirementsProcess. -- DavidRosenstrauch

---

OK, lets work the numbers. ChryslerComprehensiveCompensation was using traditional methodologies, 18 months invested we were 1/2 functionality complete. KentBeck and RonJeffries came in and we started over using ExtremeProgramming methodology. It took 9 months to get to functionality complete. But wait, we used half as many people including overhead (non-combatants). But wait again, the rewrite used some information collected during the original go. I would say about 1/3 of the total effort was saved. But wait yet a third time. That 9 months also included training of all the people in a new methodology, which was also changing as we went along too. I would estimate we back out 1 month. Now lets do the math.

((18months x 2 functionality) x 2 head count) / (9 months + 3 scavenging - 1 training) = 6.5

Each person on the project (including overhead people) resulted in about a 6.5x performance gain. This number is not inconsistent with TomKubit's experiment on the VcapsProject. The same functionality was added to two systems. One Built extreme, one built traditional. Tom showed a 10x improvement, but that did not include overhead.

So now take your 100 person project and divide by 6.5. We can do it with about 15 people and that includes overhead. -- DonWells

I think that Extreme Programming is a confession that "traditional" design methodologies don't work, yet XP doesn't solve the problem either. If you had a real workable technology for designing a software system, it would result in a flexible, reusable and extendible program. If you get lost using a map, you make a better map, you don't start out in whatever direction looked the simplest. -- GregFoxx

And how do you make a map? Travel the terrain and record what you encounter!

"No, you take pictures from a satellite."

In what way does XP not solve "the problem"? I presume the map thing is an analogy suggesting that you can't write quality software by DoTheSimplestThingThatCouldPossiblyWork. That suggests that you haven't tried it it yet. Those of us who propound it have tried it. And we're serious guys. Or we're hallucinating. Get an XP person to hang with and try simplicity. We think you'll like it. -- RonJeffries.

P.S. When lost in the woods, walk downhill. It tends to lead to a river. When you reach a river, walk downstream. It tends to lead to a town. Hmmmm ...

The "better map" suggestion makes perfect sense if you know what is going to happen. In the case of uncertainty about the future, you are better off going in a promising direction that you can easily change. The greater the uncertainty, the greater the value of fancy footwork and the greater the risk of relying on maps. -- KentBeck

---

There seems to be a hidden assumption on this page - that a HundredPersonProject is going to have WorseManagement than a ten-person project.

From above: *No, the way to deal with a 100 person project is to fire the bottom 90% of the staff.* Good point! Programming productivity shows the 80/20 (or 90/10 if you prefer) rule in action. The other thing to note is that there always is the productive 10% - even when the stars are gone, someone else rises to the challenge. When the JobControlLanguage can't just be duped anymore, someone will learn JCL!

only one problem - the people who decide who gets fired are usually a significant fraction of those that need to be fired (i.e. the overhead and management)

Another way of thinking about why projects are better broken down ... SmallWorld network theory. In this theory, the distance across graphs stays small as we scale up when we have mostly nodes in small clusters and a few random non-local links. Think of teams as clusters, and the links as communication channels, and you can see that breaking down the 100-PP into 10x10-PP might actually be a sensible and even optimal approach.

For more links on small world theory see http://www.santafe.edu/sfi/organization/annualReport/econSocial.html

I'm just waving my hands here. But it may be a justification for the 10-person approach. In my own company we have CentresOfExcellence for various technology areas. This sounds good on paper but in practice it means we have formed IslandsOfExcellence all over the country with little communication going on. With this setup it is virtually impossible to form a 10-PP to complete a large project - you cannot gather the diverse skills required in a single physical location. Drives me nuts. -- BrianEwins

---

As I look through this page I'm trying to find something useful but I keep thinking that I've wandered into a slashdot reunion. Sure there are projects out there that are wildly overstaffed, but what about projects that work on big problems?

I work on a forty person project (i.e. forty developers who touch the code). It's a CAD system of 2 million source lines compiled into a single huge executable (yours for $400,000/seat). There is no way that we could cut it down to a 10 person project. The combined scope of the tool covers enough disciplines to keep 10 people busy just keeping up with their fields, much less editing any code.

Of course we are split into groups of 5 to 10 people to handle subsections of the tool, but it's still one big project. -- JeffBell

Isn't that the point? The groups of 5 to 10 people which actually do the work? Composed to result in a big product? --WilliamUnderwood

---

P.S. When lost in the woods, walk downhill. It tends to lead to a river. When you reach a river, walk downstream. It tends to lead to a town.

Alas, projects often find themselves lost in swamps.

Hideous, hideous! Can somebody with actual experience working on a Hundred Person Project do some refactoring here? I'm afraid to touch this thing for fear of injecting a large quantity of bias. -- MartySchrader, under the guise of AnonymousCoward

---

The premise of this page is incorrect and incomplete. A "scheduled person-hour" is not the same unit of measurement as an hour of actual programming. An average employee completes about 40 scheduled person-hours per week, regardless of how the time is spent; an average 100-person project completes about 208,000 scheduled person-hours per year. There is insufficient information to support the hypothesis that projects become less efficient as they become larger, and so no conclusions are possible.

One of the first projects I worked on out of school was a large one for a US defense contractor. I don't know the total head count (it was 25+ years ago), but it was 50+. The system was a new development and consisted of multiple hardware platforms, running multiple operating systems, and coded in multiple computer languages. Each system had its own development team and although communications within teams seemed adequate, communications between teams required more effort and reducing the amount of time for purely technical work. Short of drastically extending the development deadline, I do not think it would have been feasible to reduce the number of people in order to cut the additional communications overhead. Due to the varying skill sets required, I am not certain there would even be a 10 person team that could have done it all.

I think there are large projects whose scope requires more than a single 10 person team. The best way to minimize the communications overhead is to divide the work into many smaller pieces, but even so, the ratio of time spent on communications versus that spent on purely technical work is going to be higher than in a project consisting of a single 10 person team. Unfortunately, there is no way to "buy back" that extra effort, one can only minimze it.

- WayneMack

The notion that a 10-person team, even one consisting of gurus who get along very well and are operating at peak productivity, could develop a project of the scope of WindowsXp, LinuxKernel, or other large-scale applications within any reasonable time-frame, strikes me as ludicrous. We're talking things on the order of *tens of millions* of lines of code, with extremely complicated user requirements (that require this many lines of code to specify).

See: WhatIsaProject, IdealProgrammingTime, XpProductivityMeasurementProblem

---

Last edit January 19, 2012, See github about remodeling.