Ideal Programming Time

There seems to be some confusion regarding LoadFactor and its relationship to the clock or calendar. The discussion below the line was taken from HundredPersonProject.

My take on ideal programming time (or ideal engineering time, if you're feeling insecure), is not that it represents anything on the clock at all. Instead, it is a unit of predicted effort. You look at a task and say, "Hmmm, I think that would take me three days if all I had to do was work on it."

A related concept is LoadFactor, which is the observed relationship between IdealProgrammingTime and the clock. This tells you how much you should commit to in a given number of days, weeks, or months.

IdealProgrammingTime is valuable to me because I get much better estimates from a two-stage process of "ideally how long would this take" * "observered factor" that I do from a one stage process of "how long will this take (I have a dentist appointment on Thursday and Susan is going on vacation next week and there's a network shutdown scheduled this weekend and ...)". That's all.

IdealProgrammingTime has nothing to do with how many minutes a day I spend typing, so all the discussion below about how many hours a day do you "really" program in this style of project vs. that style of project seems completely off base. Using LoadFactor to conclude "XP programmers get to spend ten times more time programming than COBOL drones" is bad argumentation and probably hurtful. However, most of the big methodology-oppressed programmers I know would say something like, "If we could just get on with this, we could do this in a tenth the time with a tenth the people," which sounds like a LoadFactor of 100 to me. --KentBeck

Because C3 measures estimated and used IdealProgrammingTime, as well as elapsed time, I believe that it does have something to do with how many minutes per day are spent developing on the task. While I may be in disagreement with Kent on that, I'm not in disagreement on the utility of what's below. --RonJeffries

---

Plain text is AlistairCockburn. Italics is RonJeffries.

a) Yes, they pay themselves using their own system... that was in the first year's activity on the project. They have graduated to paying union people (a tougher group).

b) I think I disagree with the load factor of 10 up above for the 100 person team. I agree with 40 programmers, actually 45 on a project of 90 I visited, but they were relatively light test staff and managers. The actual programmers actually programmed 30-40 hours per week, i.e., spend 2 hours a day doing non- programming activity. They also didn't document much, whatever might be required in the big books. They also worked overtime, so a day might be 10 hours averaged over the project. So really 30-40 programming hours per week or nearly 8 times your estimate. The time spent writing comments is tiny compared to writing and debugging code.

75% of 80,000 programmer hours is 60,000 hours spend programming, at your 2.5 load factor is 2.5 x 60,000 = 150,000 time clock hours = 75 XP work years = 10 XP programmers 7.5 years. This is obviously a wrong number

If they really spent those hours coding, then our math is wrong. (Our math is doubtless wrong for other reasons, as we're just trying to create controversy/discussion with the original model.) I've been a programmer for many years, and managed many more, and have never gotten 30 to 40 hours a week **programming** *out of myself or anyone else. Are you sure that's what they were doing? --Ron*

see LoadFactorArithmetic

The other thing I notice is that the load factor goes into the fact that *other* people are hired on the project. The fact that there are 55 non-programmers, busy writing requirements, test plans, schedules, documentation, user guides. This is where the XP project efficiency shows up, not in the lives of the programmers, so terribly much. I'll hazard a guess that actual programming time is not more than 1.5 different.

Certainly the other 55 are a big overhead issue. We don't think we get a lot of time programming in XP (we only get one hour out of 2.5). We **are** *suggesting above that in big projects programmers don't get even that much time in real programming. Probably you have the data, what do they say? --Ron*

LoadFactor does not say that your team gets one hour of sit-down programming time per 2.5 hours of clock time. It says that when someone guesses that a bunch of tasks add up to 6 days of IdealProgrammingTime, that's all they can accomplish in 15 work days. It doesn't predict or say how much of those 15 days are actually spent sitting at the workstation in "programming mode" vs. how much is spent in meetings, or carding, or getting drinks. So you can't infer "time spent programming" from clock time divided by LoadFactor. --AlistairCockburn

I'm sorry I've had so much trouble making it clear what we do. We track estimated IdealProgrammingTime (Estimated) and developers' reports of IdealProgrammingTime (Reported) actually put in. Developers do remove time spent in meetings or getting drinks from their actual reports. They do include time spent carding or interrogating customers related to THAT task. We also track clock time (Clock).

Developers generally wind up with Report equal to their original Estimate. So we have data that says:

- *Estimated = Reported*
- *Estimated * LoadFactor = Clock.*
- *Reported * LoadFactor = Clock.*

Your concern would be valid if we didn't report actual IdealProgrammingTime, or if it didn't track with estimated. But developers estimate and report consistently. --RonJeffries

Where does PairProgramming show its influence in these calculations? I've assumed that each member of the pair will report time against the same task - so does that imply that each individual has an effective LoadFactor of 1.25 on your project? -- Kevin Rutherford

---

We have done it both ways. Our present practice is that one person signs up for a task. That person reports his time against it. Pairing time is not reported against anything. So it's a hidden part of an individual's load factor.

We've tried having a PAIR sign up for tasks. Some folks think we should do that again, though my own view is that it dilutes the feeling of responsibility. We'll try it one day. We'd usually do the accounting by taking the (e.g.) 4 day task signed up by Chet and Ralph and count two days against each. --RonJeffries

---

Ron, how do you balance the concept of responsibility against ownership, since XP has no CodeOwnership?

Note, XP has CollectiveCodeOwnership (all developers own all classes), not none.

I'm not sure I understand the question. How are responsibility and ownership ever out of balance? So if the following isn't the answer, ask again.

A developer is responsible for an EngineeringTask. When she releases, she must have implemented the capability implied by the task and written tests for it. Her tests, *and all other UnitTest's in the system,* must be running at 100%. Since all tests ran at 100% before she started, any test that breaks was broken by her code.

Interesting... That completely changes my perceptions about some of the potential benefits of pairing. So one Programmer is the responsible party, while the other is a helper? Do you find helpers sometimes lose interest? -- kr

Generally not, because you have to give partnering to get it. However, we've seen that in times of stress sometimes people will press on without partners, and next thing you know there's code being written w/o partner. We aren't allowed to kill the offenders in the US, so some people argue that a pair should sign up together. We tried that once, stopped doing it, will probably try it again and pay more attention to how well it works.

In what way does it change your perceptions about pairing? Sounds interesting! --rj

---

MattRickard suggests elsewhere that it takes more than a lifetime's experience to learn IdealProgrammingTime well enough to use it. It inspired me to write IdealProgrammingTimeHomeworkAssignment. --RonJeffries

---

Moved from: PlanningGame

"Ideal time: "I've interpreted it is a measure of "work": how much work is the implementation of a story. But is this interpretation correct, or is "ideal time" something else instead? -- LarryWinkler

---

CategoryExtremeProgramming

Last edit May 13, 2004, See github about remodeling.