Tim Ottinger's thoughts on Software Development.

Monday, September 29, 2014

Programming Is Mostly Thinking

> Pretend you have a really great programming day.
>
> You only have to attend a few meetings, have only a few off-topic conversations, don't get distracted or interrupted much, don't have to do a bunch of status or time reporting, and you put in a good six hours of serious programming [note: this RARELY happens in an 8-10 hour day].
>
> I want to review your work in the morning, so I print out a diff of your day's work before going home.
>
> Sadly, overnight the version control system crashes and they have to recover from the previous day's backup. You have lost an entire day's work.
>
> If I give you the diff, how long will it take you to type the changes back into the code base and recover your six-hours' work?

Programming is 11/12ths Thinking

I've been touting this figure for some time now, and people keep asking me where the study is that produced such an odd number.

Well, it's not pulled out of thin air and it's not the result of a thorough scientific study.

I have done informal polls now for a few years, though I've not kept good records. My goal was not to become the scientist who cracks the statistical/mathematical code for programming activities. I was looking for a reasonable answer to a reasonable question.

However, this answer surprised me. In a long Quora post titled "How do programmers code so quickly?" one responder offered that it was a combination of physical skills (muscle memory, skill with tools, debugging skills, typing skill) and knowing where to search for info. His post was swamped and overwhelmed by posts explaining that typing and tools are not the most important aid to quick code production.

Software Factories

I have seen the stickers and slogans on stickers and social media for a long time that "typing is not the bottleneck" (though every once in a while the inability of some programmers to type is **a** bottleneck).

I am keenly aware that most management still subscribes to the idea that *motion is work*. They are fairly convinced that a lack of *motion* is a lack of *work*. That makes sense in a lawn care service, a factory assembly line, or a warehouse operation.

Nearly all of the visible work done in producing physical goods is motion. People roll steel, stamp, press, mill, pick and place, bolt/screw/rivet, and on.

Grab The Feed

Posts

Comments

Tell Me Anonymously

In addition to comments (which everyone sees) you can also leave me an anonymous comment at SayAt.Me

Buy Me A Taco!

You can send a donation, perhaps the price of a taco, via paypal.me/TimOttinger

Industrial Logic

We at Industrial Logic have taken our practice beyond run-of-the-mill Agile or Scrum or even XP. We are now "anzeneers." We build safely to build safety for all.

Popular Posts

Programming Is Mostly Thinking
Pretend you have a really great programming day. You only have to attend a few meetings, have only a few off-topic conversations, don&...

Preserving Wasteful Practices
Habit is powerful. Inhabitants of intersection set People who place symbolic value on the wasteful practice People who have bu...

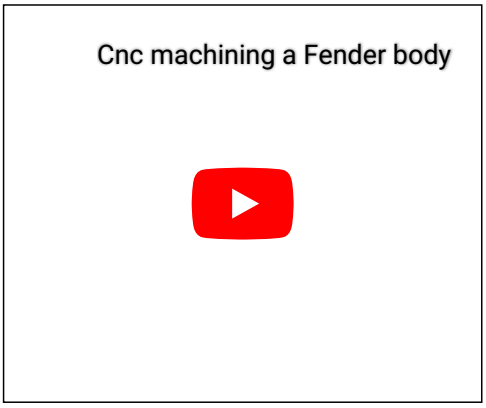
Is It My Fault You Can't Handle The Truth?
You can't handle it! In 2018 or 2019, I was introduced to the idea of hyper-rationality. I think it was under another name (to ...

Maximize Value, not Quantity

Modern factories produce goods with Computer Numerical Control machines, which produce perfect copies of an original model that may not even exist in real life. These machines work from abstract models -- just data, really -- and perform perfect motion. Humans tend the machines, rather than working the wood by hand.

I have some great guitars that were produced at affordable costs because of the degree of automation brought by such machines.

Great boutique guitars are produced entirely by hand at higher cost and I don't put down that effort either. The world has room for both.



Software developers have perfected the factory. It runs flawlessly bit-perfect copies. You just click the "copy" or "download" button. It's so cheap that the purchasers happily cover the costs of the factory. Those who are cautious will double check the checksums that come with the download, but most people don't bother. The machines are reliable and efficient and quick and cheap.

Once the initial model (really, just data) exists, then the marginal cost of all the bit-perfect copies is essentially zero. Yes, this is just copying and not creating, but that's what factories do. Custom shops might produce unique items (like guitars) but factories create copies of originals.

The software factory tends to give you a progress bar, so you can visualize the motion of bits, but in many ways you can say that the product doesn't really exist. It's a pattern of tiny charged v. uncharged areas of metal on a plate (well, probably) and you don't even pay for the plate or the magnet or the laser when you create the copy. It's already there.

Software is an intellectual good.

The Design Shop

In my years of working with Uncle Bob Martin, I heard him continually tell customers and students that software development is not a fabrication operation, but a design operation. Once the initial design is done, all the duplication is done by machines at nearly zero cost.

So what programmers and testers and POs and Scrum Masters and software management area all doing (if they're doing it right) is designing the data model that will later be used by the factory to create copies for use by customers, patrons, and other people in the community the software is intended to serve.

I was chatting with a manager who was once a PO on a team I coached many years ago. This is only my best memory of the conversation (I didn&...

14 Weird Observations About Agile Team Velocity

(note: I added a 15th, but was worried that changing the title would invalidate links, so you get a bonus observation at no extra cost) I ...

Bug Teams v. The Nature Of Defects

How it Happens You realize that you're not getting as much done as you expected to get done. It's troublesome because you have pl...

I Want Agile Back

Note: this was originally all plain text and a little shorter. As more people have joined the conversation, and other supportive materia...

Preplanning Poker: Is This Story Even Possible?

The story says "attach an e-commerce server." Well, maybe it says "As a product manager I want my system to incorporate an ...

Splitting Stories - A Resource Listicle

I've noticed that for several years now, one of the most frequently asked questions in agile forums deals with the splitting of stories...

Defending Scrum Against Stupid Arguments

I'm not a big scrum promoter, but I am VERY familiar with scrum and have coached many teams and always been able to improve their succes...

Agile In A Flash

Get your copy of Agile In A Flash today! Maybe outfit your whole team...

About Me

Agileotter

View my complete profile

Yet the mechanistic, Industrial-Age idea of software development as a factory persists, and developers dutifully try to make it look like they're doing physical labor at the detriment of the process.

All intellectual activities are hard to observe and monitor. An idea that is 80% complete has no physical manifestation. It's an idea, and it's not done yet. Sometimes we have experiments or proof-of-concept code or notes, but they don't give an accurate "% complete" number as does physical work.

A chair being manufactured looks about 50% done at the 50% mark. When it's done, it looks done.

A design for a chair may not exist on paper until it is more than 70% complete. And we don't know that it's really 70% done, because it's not finished being designed yet.

The Answer: Really?

I have asked this question at conventions, client companies, to my peers, to colleagues, and to strangers I have met for the first time when I find out they are programmers.

The answer I receive most often is "about half an hour."

I could use the 8-hour day, ignoring meetings and interruptions and status reports, but that feels like padding the answer. I stick to the six hours doing things that programmers identify as programming work.

There are twelve half-hours in six hours. One half-hour to retype all the changes made in six hours of hard programming work.

What in the world can that mean? How can it be so little?

The Meaning Behind the Answer

Right now I suspect a bunch of managers are about to go yell at their programmers for putting in a half-hour's work in an 8-hour day! That would be a horrible misunderstanding of what was actually happening.

What is really happening?

- Programmers were typing on and off all day. Those 30 minutes are to recreate the ***net result*** of all the work they wrote, un-wrote, edited, and reworked through the day. It is not all the effort they put in, it is only the *residue of the effort.*
- Programmers are ***avoiding defects*** as best they can. In order to do that, they have to be continuously evaluating the code as they write it, hypothesizing the kinds of defects or security vulnerabilities they might be introducing. After all, they receive their harshest criticism for introducing defects into the shared code base.
- Programming is a kind of ***lossy compression***. The code only says what the program must do when it is running. Why a programmer chose one particular way over others, how it influences the rest of the system, what errors were introduced and removed, and what pitfalls it avoids are not (generally) present in the text of the program.
- Most of the work is not in making the change, but in ***deciding how to make the change***. Deciding requires us to understand the code that already exists. This is especially time-consuming when the code is messy or the design is not very obvious in the source code.

- Programmers work in a ***social context*** since all their results are integrated into a shared code base (and most use pair programming or other "many eyes" techniques). Programmers may be helping other programmers or testers or operations people get a handle on their work. Connecting and communicating with others has benefits and costs that don't appear in the code.

Six hours of intellectual work (reading, researching, deciding, confirming, validating, verifying) translates to about 30 minutes worth of net lines-of-code change to a code base.

That's not additional lines of code. We often have weeks when we fix bugs and add features and have fewer lines of code at end of the week than we had at the beginning of the week. I once got in trouble for having multiple weeks where we had negative lines of code -- we didn't know the 'grand boss' over our team was reporting SLOC as if it measured progress. Sigh.

Programmers will gladly explain that the work they did was reading, learning, understanding, sometimes guessing, researching, debugging, testing, compiling, running, hypothesizing and disproving their ideas of what the code should look like. In short, they were thinking and deciding.
Most of what goes on is intellectual work.

One of the quora responders wrote:

> You see the fingers flying over the keyboard; you don't see the hours spent in talking to users, discussing the problems with coworkers, doing research and thinking the problems through.

Another suggested:

> I achieve it firstly (to the extent that I do) by 'helping' the customer to eliminate the unnecessary notions from their idea, which they often mistakenly call 'requirements' and sometimes even say they are 'must have'. This is the biggest possible acceleration in the delivery of a solution because I can do an infinite amount of no work in no time at all.

And yet another:

> Really good developers do 90% or more of the work before they ever touch the keyboard; really understanding the requirements and devising an appropriate solution.

These are not unique unusual answers. I find that most of the time, "knowing what not to write", "doing less," "working in smaller steps", and "having first figured out what to do" are common answers. Programming is much more about thinking than about typing.

I have examined a lot of the change logs (diffs). It has consistently looked like 30+/-10 minutes of change on a good day (at least to me).

I'm confident enough to tout this number as effectively true, though I should mention that no company I work with has so far been willing to delete a whole day's work to prove or disprove this experiment yet. Remember, I have only estimates and examinations of daily diffs to work from. The result here is not scientific.

I should also let you know that people who do more typing or more cut/paste are often doing less thinking and understanding, which results in more errors and more burden

on other programmers to understand and correct their code.

Code is just the residue of the work.

So What?

If programming is 1/12th motion and 11/12ths thinking, then we shouldn't push people to be typing 11/12ths of the time. We should instead provide the materials, environment, and processes necessary to ensure that the thinking we do is of high quality.

Doing otherwise is optimizing the system for the wrong effect.

What if we changed our tactics, and intentionally built systems for thinking together about software and making decisions easier to make? I think that productivity lies in this direction.

So I invite you: how can you experiment with learning on-the-job to create systems where the thinking is optimized?

9 comments:

A J Mishra June 29, 2023 at 12:09 AM

Thank you so much for such an informative blog post! Check out more about How to Ensure Quality in Software Development and share your thought about it.

Reply

Buonarotti April 21, 2024 at 1:15 AM

thanks for the post - diseño web vitoria

Reply

sergioMadrigal April 21, 2024 at 3:12 AM

Thank you very much for this interesting and thoughtful post!

Reply

Daniel K April 21, 2024 at 7:08 AM

Agreed from my coding experience too. There are things / artifacts you can produce while doing the thinking, though. See literate coding by Knuth, commenting code before writing, data modelling and other uml diagrams can help.

Reply

Bob Lockwood April 22, 2024 at 5:22 AM

Let's face it, most programmers can't stop thinking about how to solve a problem. So many programmers are also thinking outside the work day. So the employers are getting much more than a normal day from their developers. Agree/disagree?

Reply

Replies

Ronnie Overby June 14, 2024 at 11:18 AM

Agree 100%. Most days, my commute and my bed are where I'm doing real problem solving for my employer.

Reply

MikeHynz April 23, 2024 at 9:15 AM

This aged well.

Reply

Teki Chan May 7, 2024 at 5:10 AM

Well said. Thanks for the thoughtful article. Developers do think a lot

Reply

Nick June 15, 2024 at 12:21 PM

This is a phenomenally good article.

Reply

To leave a comment, click the button below to sign in with Google.

SIGN IN WITH GOOGLE

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)