Twitter Thread by Santiago

Santiago
@svpino

I sat next to one of the smartest, most successful software engineers in the country for over 10 years.

Here's what I learned.

Fast is 10 times better than good.

9 out of 10 times, a "good enough" solution is all you need to unlock time, money, and attention.

Focus relentlessly on being first. Then take the time to be right.

Fast, then right.

Ship more frequently.

The faster you show your work to your users, the faster you'll hear their feedback, you'll adapt, and you'll ship again.

A question you should ask every day:

What can I do to ship this one day before planned?

Technical debt is a good thing.

Everyone hates technical debt. I think they don't know how to take advantage of it.

When used appropriately, technical debt means working on what truly matters and deferring anything that can wait.

No debt ■ You waited too long.

Ask, don't think.

People burn their time every day instead of asking for help.

You don't get extra points for working harder, especially when a little help will make you faster.

Time is your most valuable resource. Don't waste it.

On the same topic:

There aren't stupid questions. Only stupid people that don't ask them.

"He who asks a question is a fool for five minutes; he who does not ask a question remains a fool forever." — Chinese Proverb

Communication > technical skills.

You won't get far unless you can convince people, share your ideas, and make connections.

Invest twice as much time learning how to talk and write than how to program.

Not everything you can do is worth doing.

If you want to make a difference, learn to prioritize and delegate.

Invaluable skills:

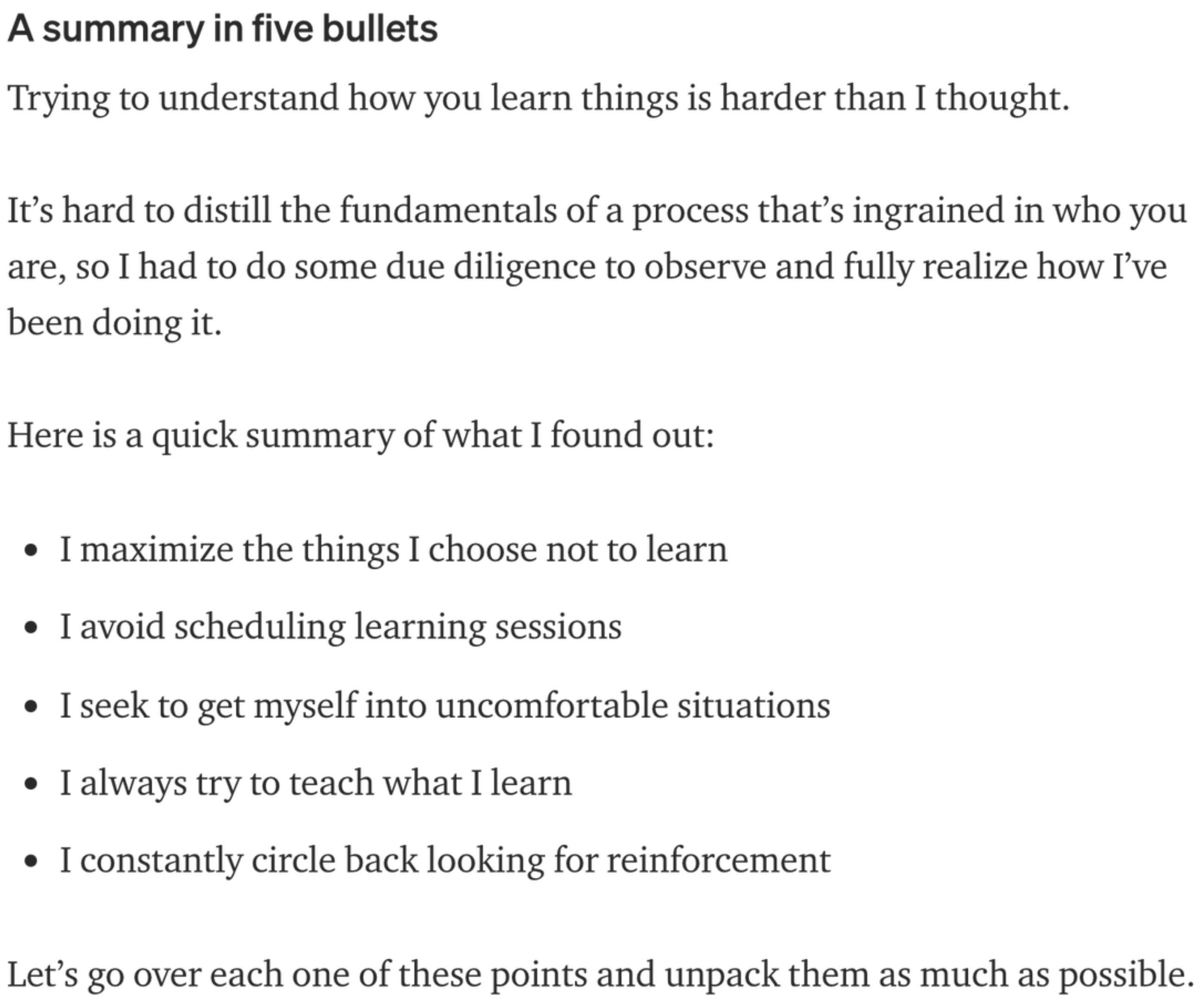
• Recognizing high-impact work
• Recognizing what wastes your time
• Learning to delegate effectively

Learn all about learning.

You need a process that helps you learn efficiently and effectively.

I wrote an article with my 5-step process to learn new things. You can now go and make it yours:

https://t.co/jYw7WGy6Ht



A summary in five bullets

Trying to understand how you learn things is harder than I thought.

It’s hard to distill the fundamentals of a process that’s ingrained in who you are, so I had to do some due diligence to observe and fully realize how I’ve been doing it.

Here is a quick summary of what I found out:

- I maximize the things I choose not to learn
- I avoid scheduling learning sessions
- I seek to get myself into uncomfortable situations
- I always try to teach what I learn
- I constantly circle back looking for reinforcement

Let’s go over each one of these points and unpack them as much as possible.

Share what you learn.

Many people think that hiding information makes them a linchpin. 99% of the time, the opposite happens.

People want to be around those who lift them.

The quickest path to becoming the angular stone of your team: share your knowledge indiscriminately.

Take full responsibility.

A question you should ask every time a problem happens: "What can I do differently next time?"

Look inward. Every time, no matter what.

Finding justifications is easy. It's also a sure way to stay mediocre.

The best code is the one nobody wrote.

Code is a liability. Learn to solve problems by writing as few lines as possible.

An underappreciated superpower: No-code solutions.

Your worst enemy: complexity.

If you don’t test, it doesn’t work.

Any code that can break will eventually break. If you don't have automated tests, you'll suffer the consequences.

Embrace failures.

If you don't fail, you don't learn. If you haven't failed yet, you aren't working on interesting problems.

You fail, you adapt, you try again.

Welcome imposter syndrome.

If you feel inadequate, it means you are stretching yourself and growing. You don't know how to do it yet, but you will.

(People who are working on easy, irrelevant things have no reason to doubt themselves.)

I post threads like this every week. You can find them here: @svpino.

Follow along for a good bunch of practical tips and epic stories about my experience in the field.