STEVEN TALCOTT SMITH

Level Up

HOW TO BECOME A GREAT PROFESSIONAL SOFTWARE DEVELOPER

Level Up!

How to Become a Great Professional Software Developer

Steven Talcott Smith

This book is for sale at http://leanpub.com/level_up

This version was published on 2014-11-15

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Steven Talcott Smith by spreading the word about this book on Twitter!

The suggested hashtag for this book is #secretsoftheaelogicians.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#secretsoftheaelogicians

This book is a waypoint on a journey I started long ago. I do not yet know where it will lead. I have many people to thank for inspiring me, for teaching me, for encouraging me. The person I want to thank most of all is my wife, Celda. Thank you for your Love and support. Thank you for not being too hard on me when I am present in body but my mind is elsewhere, thinking about software, business or writing.

I want to thank our staff in the Philippines, in particular, Nestor Pestelos for reading and reviewing drafts and for taking an interest in this project.

Contents

Pair Pragmatically

Pair programming is still one of the most controversial techniques to come out of the Agile movement. Non-technical managers often doubt its effectiveness. Reclusive programmers accustomed to a high degree of autonomy and very little scrutiny or oversight, may resist opening up and sharing their work and thought process so completely.

It is very different from the cliched developer working alone with a door shut or in a cubicle or bullpin with headphones on. Among those who practice it properly, there is little doubt about the utility and value of the technique.

What is Pair Programming?

Pair programming as we practice it at ÆLOGICA, consists of two people sitting at one powerful computer with a single large 27" or 30" monitor. Developers may opt to keep a personal laptop to the side of the main computer for use in looking up documentation or performing quick research.

Two keyboards and two mice are connected to the single computer. Either person may control the computer though only one person is typing or controlling the mouse at any

given time. The two developers will continuously discuss the problem at hand and will take care to vocalize their thoughts to share with the other person. Control passes back and forth naturally in the course of the work or possibly according to an agreed cadence. Both "hogging the keyboard" and partner-disengagement is frowned upon.

Ramon and Cecille pair-programming. Ramon is navigating, Cecille is driving.

The person who is actively typing or who has their hand on the pointing device may be referred to as the "driver." The person who is not typing may be referred to as the "navigator." The phrase "let me drive for a second" or "you drive" is commonly used to switch roles.

Why We Pair

The main benefits to pair programming include:

- fewer defects, better designs
- knowledge is spread around the team
- mutual accountability
- better adherence to conventions and best practices
- faster convergence to solutions for complex problems
- less time spent "stuck"
- higher engagement and enthusiasm
- less time wasted on non-development tasks

The cost of pair programming according to some studies, seems to be about a 15% in the form of an up-front decrease in the apparent velocity as compared with each developer working solo. However, the higher quality leads to reduced re-work, lower ongoing maintenance costs and perhaps delayed obselence. This increases the value of the software asset. Higher employee satisfaction leads to lower turnover. When turnover occurs, knowledge is more distributed throughout the organization. Pairing also makes it easier to bring new developers on and make them productive right away. This means any given person less essential to a particular project. Managers should appreciate all of these things.

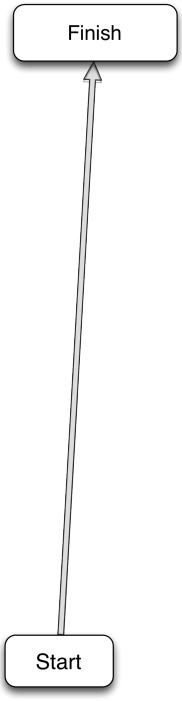
I submit that significant commercial software development is rarely undertaken without the expectation of 10x or more return on investment. If a given project cannot accept a 15% pay-it-forward tax to "do things right" and produce a more maintainable, lower-defect system, with employees who are happy to continue to work on it, then one might wonder if the development should be undertaken at all. This holds true whether the tax is paid through pairing or through other techniques aimed at addressing the same problems.

We believe that by pairing pragmatically as opposed to dogmatically, we can realize most of the benefits of pairing without necessarily paying the 15% tax.

How Pairing Saves Time

Non-technical managers may have trouble visualizing how pair programming can be anything other than twice as expensive as "solo" programming. The naive person observes two people sitting at one computer and imagines that their production will be halved. An illustration and analogy from sport will help explain the fallacy of this thinking.

As we discussed earlier, there is a notional Minimum Essential Effort (MEE) in the development of any feature or the accomplishment of a software development task. Let us represent this on a 2 dimensional surface as the straight line from the starting point to the ending point.



Ideal path of Minimum Essential Effort

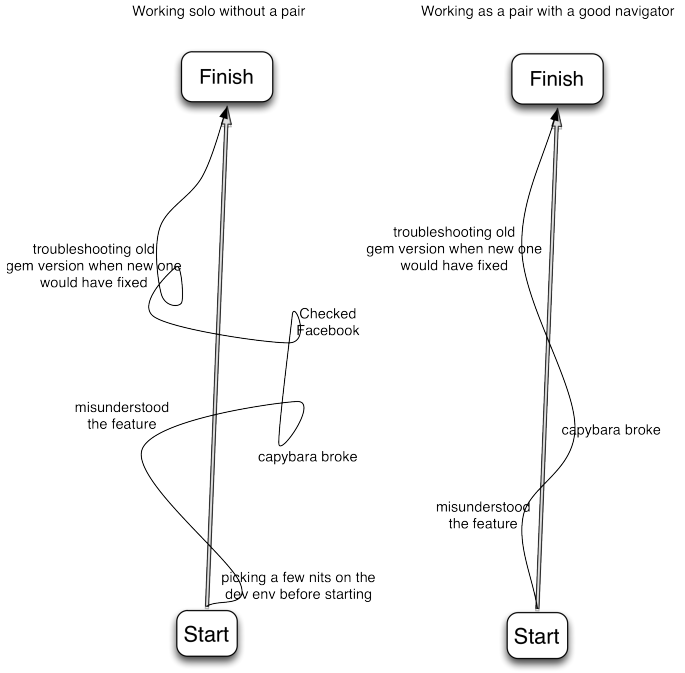
This is much like a leg of a sailboat race between two marks

placed in the water. The MEE is the "perfect course" which would lead to the shortest distance traveled.

We measure our rate of progress in sailboat racing with the term "Velocity Made Good" (VMG) – that is speed toward the mark. One rarely can steer a course "straight for the mark" – the wind may shift, other boats may get in the way, current may push the boat one way or another, the fastest point of sail may not be very close to the ideal course, etc. There are many factors.

A sailboat racing crew may consist of at least several people: one or more to handle the sails, one to steer, and one to navigate. A few more may be handy for "rail meat" but here the analogy breaks down. Suffice to say that a boat with at least two or three crew focusing on their individual roles – ceteris parabis (all other things being equal) – may be more speedily sailed toward the mark than a single-handed vessel where one individual performs all the roles.

Software development is a multi-dimensional activity and forward motion does not always imply progress toward the desired end. Direction is very important. The following diagram shows how a pair may stay on course and drastically shorten the actual calendar and clock time expended on a task.



Solo vs. Pair Programming

With this visualization and the benefits we hope to obtain

by pairing firmly in mind, let us proceed to discuss guidelines for pairing pragmatically.

When to Pair

Here are the occasions when we almost certainly want to pair:

- When two heads are better than one

  Complex problems with a lot of context may benefit from pairing. If you you feel like you need to "talk it out" you probably should pair. If you are confused about a requirement or need to make a guess about something and the customer or product manager is not available, pair. If you are unsure about how to properly test or structure something, pair. If you are working with an unfamiliar system or subsystem, pair. If you are struggling with your tooling and you suspect other developers have resolved the problem or that your problem affects everyone, pair.
- When you intentionally want to transmit knowledge or technique
- When deploying to production

  Depending on the complexity of the system, how many users it has, and how much money can be lost with an outage, less experienced people generally should not deploy to production alone. Two heads

are more likely to spot a problem quickly before irreversible steps are taken. Discussing the plan explicitly with someone who bears shared responsibility for failure may highlight problems or risks before they occur.

- When reviewing code or pull requests or making a large commit

  Someone will have to look at the code later one way or another. It is better to inspect it now.

- When on-boarding a new team member

  There is no faster way to get someone new up to speed.

- When you are having trouble staying focused

  Your pair can help you focus on the chore. People hesitate to interrupt a pair of people who are obviously engrossed in a task, whereas people frequently feel justified in interrupting someone who is working alone. Discussing the overall plan with someone can help clear the road ahead.

- When you are not sure what you should do

  Your pair probably knows what to do or at least can help you figure it out quickly. Two people can more easily "saturate" the design space or come up with varied approaches to a problem.

When Not to Pair

Pairing pragmatically means that while we prefer to pair, we recognize there are some occasions where pairing may not be beneficial. Here are some times we recommend breaking off:

- When you need to cover ground as quickly as possible

  Quality isn't always an overriding concern. Occasionally you just need some stuff working quickly for a demo. Or occasionally a critical item comes up but you do not want to disrupt progress at the main front of development. Split up.
- When you can divide and conquer the problem space

  For example, suppose you have a long list of user interface glitches to clear up. You both know how to do it. It's straightforward and you could go twice as fast independently. Divide and conquer.
- When you need to perform UI Design

  6-up techniques and others allow for collaborative UI ideation but design sometimes benefits from the thought and coherent vision of a single individual.
- When you need brain or physical space

  We all get sick of each other from time to time and pair programming can involve a great deal of time sharing someone's personal space. Get some time apart.

Driving vs. Navigating

Role of the Driver

The driver typically has hands on the keyboard and will be thinking of how to solve the immediate problem. When practicing Test-Driven Development (TDD), the driver will write a test or implement code with the objective of making a test pass. If you are driving and you do not know what to do next, your pair has probably let you down. In this case, suggest writing a test and maybe switching roles.

Most everyone who can program to some degree knows how to play the driver role in a pair. Everyone can benefit from practice. The most difficult aspect of driving is speaking about what you are doing while you are doing it. The second most difficult thing to learn is how to give up control and switch.

Navigator

Most developers need to practice navigation in a pair context to become skilled. Experienced developers who have never paired will have learned to navigate by themselves but may not be accustomed to speaking about it, to guiding someone else or to performing the role so consciously.

Navigation usually consists of asking (aloud!) and often answering such questions as:

- "Where are we?"

- "What is the next step here?

The Navigator constantly assesses progress toward the feature or objective and helps correct course when necessary.

The Navigator may also:

- help the driver get unstuck
- research needed information (eg. api usage, candidate gems, resolve confusion about requirements)
- question the approach
- handle interruptions (run interference)
- point out unseen problems or errors

Switching Roles

Two skilled developers who are evenly matched in terms of ability will often fight for control. You know a pair is really working well when keyboard control shifts back and forth by the minute. If you observe a pair over 5 minutes and do not see the control change hands, something is probably wrong.

If you are the one on the keyboard and find that you have been typing or using the keyboard for 5 minutes straight, or if you notice your pair has disengaged, back away from the keyboard for a moment. Say, "hey, you do this!" Or, "can you take over?"

If you are the one who is supposed to be navigating but you are losing the plot and having trouble seeing the big

picture, ask to drive and let the other person navigate for a bit.

TDD Ping-pong

A more structured approach which works well with Test-Driven Development is to take turns writing a test and switching roles to let the other person write the code to implement the test. They would then write the next test and pass control back to you to implement the code. Rinse and repeat.

Respect Your Pair

Pairing requires etiquette. You need to be nice. Most of this is "common sense" but some requires practice – especially if you have never worked in a very collaborative environment.

- Check your ego at the door

  This is the first rule. Don't put anyone down or insult them. You are not a genius and even if you are, you probably don't want to act like one. Put your ego away. If you do a great job pairing, you both did a great job. If you did poorly, blame is shared.
- Ask your pair to explain his or her thinking

  If your driver has gone quiet, ask them to explain what they are doing. If you don't know why the

driver is doing something or why the navigator is recommending a certain approach, ask them to explain their thinking.

- Give good criticism

  Be constructive. If you see a problem with the code, say something. Don't just sit their like a lump on a log.

- Take criticism well

  Do not be defensive. All code is collectively owned. It is not "mine" or "yours" – it's ours.

- Do not be silent

  Silence is deadly. When observing a room of pair-programmers, one should hear constant chatter.

  Old habits die hard and most of us learned to program in absolute blissful solitude. Most of us cultivated techniques for tuning others out. That doesn't work here.

- Do not hog the keyboard

  Even if you are not aggressive, your pair may be too timid or intimidated to ask for control. If you've been typing away furiously like you're the only one there, stop. Take your hands off and ask the other person to pick up and carry on.

Change Position!

Pairing involves sitting at a slight angle to the center of the main screen. At ÆLOGICA we have special custom made

tables which encourage each member of the pair to direct their chair properly toward the common focus of attention.

If you sit in the same pair position (on either of the left or right side) day after day, the slight asymmetry in your posture and seating position will accumulate. Even if you carefully avoid asymmetrical postures, you will cancel out any accumulated asymmetry by switching positions often.

I believe that moving positions may offer other benefits besides those related to posture and purely physical concern. Changing positions or workstations or even rooms, can help make things feel "fresh" and can help break poor habits or modalities of interaction that set up between team members.

Use Proper Sitting Posture

If you notice that your neck is slightly turned, adjust your chair so that your body faces the screen and your head is positioned straight ahead most of the time.

Over-use of the adjunct or personal laptop can cause asymmetrical posture. When you turn to use a laptop adjacent to your workstation, turn completely to face the laptop using the rotational capability of the chair. If you simply twist your body or your neck, you will end up with neck and back pain. If you twist this way in one direction for a week, you will be miserable. Don't do it.

If you find you need to use your laptop extensively, it may be time to "split" from your pair for a while. Go find a

comfortable place to use your laptop where you have more space or where you won't be tempted to sit at an angle or in a twisted position.

Pair Flow

Earlier in the chapter on Maximizing Productivity, we discussed the concept of Flow as it relates to the activity of programming. When you work as a pair, you no longer have freedom to get in the "zone" quite the same way as you would if you were working alone. However, some have described a state called "Pair Flow" – wherein the pair is deeply absorbed in the task. This can be very powerful. You know you are in pair flow when the work feels sticky, and you do not want to get up and you see that your partner feels the same.

Structuring Breaks

When pairing, you want to be diligent about taking breaks. Absorption in the task waxes and wanes for either partner and when you feel like taking a break, the other party may be deep in the problem and not want to break. Out of concern for the pair, you may stick with it and by the time you are absorbed, the your pair may be ready for a break. He or she may then be reluctant to leave and you can find yourself "stuck" to the workstation much too long.

If this happens, try practicing Pomodoro together. We

describe the Pomodoro technique in Maximizing Productivity.

When you take a break while pairing, it is especially important that you get up and get out of the room. Get a little space and time away from your pair. Let them take care of personal business or visit the restroom.

Estimate your work in terms of Pomodoros. Use the chance to co-estimate with your partner. The better you can get at estimating granlar tasks in terms of well-defined blocks of time such as Pomodoros that fit within your break schedule, the more accurate your overall estimates will be. Hours often seem to be too large to use for estimation while minutes are of course too small.

Bookend your end your breaks with with the questions advised in the chapter on Maximizing Productivity. When working with a pair, you really have no excuse. Before you break, ask, "What did we just do?" or "Where are we now?" Then state to your pair what you will start with when you come back. If you are really deep in a problem, leave a note on the screen or on paper as to where you left off.

Keep Each Other Accountable

Mutual accountability is one the great reasons we pair program. With someone programming shoulder to shoulder with us, we have to stay on our toes.

Be explicit about your intentions. Speak out loud. Do not just start typing away. Tell your pair what you are planning

to do and why. This alone will have a positive regulating effect on your efficiency.

Keep your momentum up. When you feel like you are slowing down or losing steam, ask your pair to take over. Usually the feeling will pass. You know you did a great job when you are both slightly exhausted and amazed at what you accomplished at the end of the day.

Mind your pair. If your pair is goofing-off too much, wasting time, arriving late, or doing something that detracts from your mutual enthusiasm for the work, or from the productivity of the team, call them out. Do not suffer in silence or try to pick up their slack and make up for their deficiency.

Go to your manager or ask another person outside your team for advice if you have trouble with a difficult pair partner. Chances are you will have some good suggestions. Ultimately if you just cannot stand your pair, your manager may be able to reassign one of you.

What to Do if Your Workplace Does Not Pair

Pairing is a cultural thing. Some cultures do not support it. Some will be actively hostile to the idea. If you work in an office where developers all wear headphones or close their doors you many not have much luck with pairing at first. Management may not be the only obsticle here.

I suggest starting by making sure your own workspace can support pairing and even looks inviting to a potential pair-partner. Make sure you have a large enough desk to accommodate an extra chair. See if you can pull an extra chair in or keep it to the side so you can pair opportunistically with a minimum of fuss. Keep an extra keyboard and mouse handy. If you are running multiple smaller monitors or even a single small one, request a single 27" or larger monitor and swap any little ones out. Pairing works best with a single screen. Keep your desk clean of personal clutter so that someone does not feel that they are invading your personal space too much.

If you do have an office with a door that closes, you're in luck. You may be able to arrange your desk to better support pairing by placing it against a wall rather than facing the door.

If you want to ease into it, volunteer use of your workstation for code review – either to review your own code or that of a colleague.

More suggestions can be found in the chapter "Debug Your Workplace."