A Software Development Process for a Team of One

by Philippe Kruchten
Rational Fellow

For some, the phrase "software engineering process" evokes an image of a huge set of dusty binders full of policies, directives, and forms, all saturated with administrative jargon. But, these are materials that would probably be used only by very large companies that deliver software at a snail's pace to government agencies and Fortune 500 companies -- software developed by armies of programmers aligned in giant cubicle farms and herded by "pointy-haired managers," like the one in the famous Dilbert cartoons by Scott Adams.

In reality, however, a software engineering process does not need to be such a monster. It can be as lightweight or heavyweight as the job at hand and the size of the development organization requires. Whether the project is a 200-developer mastodon or a short, solo gig, a good process can be tailored to fit the job.

The purpose of a software engineering process is not to make developers' lives miserable, or to squash creativity under massive amounts of paperwork. Its only real purpose is to ensure that a software development organization can predictably engineer and deliver high-quality software that meets all of the needs and requirements of its users -- on schedule and within budget.

To understand the essence of a software engineering process, let's look at a very simple software project developed by a team of one.

A Solo Software Project

Although Nick, a software engineer with twelve years of development experience, prefers to work alone, he deliberately and conscientiously follows a well-defined process. Here is a diary he kept of a one-week project that he recently completed for Gary, an old friend of his.

The Seminal Idea (Saturday Night)

Tonight I met my friend Gary in our favorite watering hole. He's the software development manager in a small company. As part of an effort to improve their process efficiency and predictability, they recently went through Personal Software

Process training.[1] But he has a problem: A key element of the approach is that individual developers keep track of the time they spend on each type of activity, such as requirements capture, design, testing, and administration. Each of his developers uses a different tracking strategy, and at the end of the week, it is quite a nightmare to gather and consolidate all the data. An assistant has to go around gathering all the random Post-It notes, e-mails, and voice mails that team members use to estimate how they spent their time. This is discouraging, because for a software organization accurate measurements of the effort expended on various activities are key for monitoring productivity, identifying potential areas for process improvement, and, above all, planning future projects effectively.

I suggested that Gary try automating the tedious job of tracking activity effort. Developers could have little timers on their screens that they could activate, associate with an activity name, suspend when they stop for lunch or some other interruption, resume when they return, and close when the task is completed. The data would be stored somewhere safe, and then retrieved and consolidated in a spreadsheet at the end of the week. "Great idea!" said Gary. "Nick, can you crank that out for me? It would save me a lot of hassle, and therefore a lot of money. I'll pay you whatever you want. Well, sort of. How much do you want to develop this?" I told Gary that I needed to think twice about it. I had no engagement for the following week, so maybe I could crank out something in a few hours. But I quickly revised that: "Hmmmm, better make it a couple of days. Come to my office Monday morning around 11 a.m., and I'll have a proposal for you."

The Proposal (Monday Morning)

I thought about the timer project a few times over the rest of the weekend, and by the time I woke up this morning, I had a "mental concept" of it, as well as one or two possible implementation ideas. But this was a serious business proposition, so I needed a serious business case. What would I build, and how many resources would I need to throw at it? Mostly, this would require my time and maybe some software acquisitions. And finally, how much would I ask Gary to pay me? So I arrived here at my office early this morning, cleaned my desk, and laid out four sheets of paper. Then, at the top of each one, I wrote one of the following headings:

- Vision
- Plan
- Risks
- Business Case

The Vision

I start with the *Vision*. I need to describe, for Gary and myself, what exactly we want to achieve: the fundamental need he is trying to address, what the tool will look like, and how it will be used.

Here's my first stab at it:

Personal Timer Tool: VISION

Problem
For Gary's organization, the inability to gather consistent data about time spent on various software development activities hampers the ability to monitor a project's progress against estimates, invoice customers properly, pay contractors, and, ultimately, accurately estimate work for future projects.

Vision Statement
A Personal Timer Tool (PTT) that measures time spent, and collects and stores this data for later sorting and extraction, would (unlike Post-It notes and wild guesses) allow Gary's organization to easily make systematic, consistent assessments of where effort is spent, track actual time spent versus estimates for a project, and do a better job of estimating future development workloads.

Main Parties Involved

- individual developers
- administrative assistant
- project managers

Use Cases

- measure time for an activity
- extract weekly time sheet
- consolidate data for a project
- set up tool and database for a project

In the *Plan*, I'll rough out a schedule for the next few days, mainly identifying major milestones -- the points at which I'll have to make a decision, either on my own, or, more likely, together with Gary.

At lunch today, I'll need to reach an agreement with Gary to go ahead, to get some commitment from him to at least start paying me for the job. I'll have to have him agree on the *Vision*, the *Plan*, and my estimate. For myself, I need a private *Business Case* that Gary won't see, detailing how much I need to spend on the project. If I can accomplish all this -- getting agreement on an overall vision, a plan, and a price, and ensuring that I won't lose money, then I'll achieve my first milestone, the Lifecycle Objective (LCO) Milestone, and bring the Inception phase to a close.

To make the pill easier for Gary to swallow, and also to cover myself in case I run into some unforeseen difficulty, I'll suggest that he commit only to paying me for producing a rough prototype, which I'll show him Tuesday evening. Only then, if he likes what he sees (and if I'm confident I can finish the project by Friday), will I ask him to commit to the total amount.

The Plan

It's only 9:30 a.m., so I can work next on the *Plan*. No need for heavy artillery planning software to do a GANNT chart, but I want a rough plan that will decompose my time into major phases.

After sketching it out on that piece of paper, I decide to transfer it to my pocket agenda. This is what my first phase plan looks like:

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
INCEPTION Vision Plan Business Case Risks	Prototype Mitigate risks	**CONSTRUCTION** Design Code Test	Design Code Test		
LCO: OK from G.	LCA: OK from G.		IOC: show the first beta version		
ELABORATION Prototype	Use cases Tests	Design Code Test	**TRANSITION** Improvements?		
			Delivery		

Inception. I've been working on these activities since early morning and hope to wrap them up just after lunch with Gary. If I get his commitment to pay for a demonstration prototype, then this phase will represent a day of work on the project. If he won't commit, then we'll quit there and remain good friends.

Elaboration. I think I could conclude this phase by Tuesday lunch. I'll build a rough prototype that will allow me to "elaborate" on the requirements, the solution, and the plan, and to explore some of my ideas. Then, I'll ask Gary again to validate everything with me over lunch. The prototype will give us several things:

- Something more concrete to show Gary so I can get more feedback from him on the requirements (you know, the "I'll Know It When I See It" syndrome). After all, so far all he'll have had to go on is a discussion in front of a glass of pale ale and some rough plans.
- More important for me, I can validate that I really have all the bits and pieces needed to do the job on my hard drive, and that I do not underestimate the amount of effort. While shaving this morning I thought I had an idea for an architecture and the various parts I would use, but now I am less sure. Can I use this little database I used on a previous project, and will I be able to interface it with a Java applet? Where did I put the user guide for the API? Will that run on their OS?
- More information to do a much better plan and a more detailed schedule.
- A starting point for building the real thing, with a chance to scrap everything I did wrong.
- An opportunity to refresh my database definition skills, as well as my Java style.

I think of this crucial Tuesday lunch as the Lifecycle Architecture (LCA) Milestone. At that point, both Gary and I will have the option to bail out, significantly recast the project, or go ahead with confidence.

Construction. I figure that if Gary gives me his go-ahead, helped along by a fine Beaujolais, then on Wednesday morning I'll start early to build the real thing, all neat and clean, and thoroughly test it. Then I'll ask Gary to come at around 2 p.m. on Thursday and bring one of his programmers to try it out, on *his* laptop. That will give me the afternoon to fix whatever they don't like.

This Thursday session I think of as the Initial Operational Capability (IOC) Milestone, because it will be the first time real that actual end users try the software.

Transition. This will be the last run, and I hope it will last just a couple of hours. It'll conclude with a release -- I'll probably e-mail the software to them -- accompanied by my sweet invoice, all by close of business Thursday.

The Risk List

I've already mentioned that I have a few doubts and worries. Rather than putting my head in the sand, I'll jot them down on that piece of paper headed "Risks." I'll include anything I can think of that might make this little project fail, get delayed, or go over budget. And I'll use a pencil, because a *Risk List* always requires reorganization and sorting out again and again.

Here's what's on my list:

Personal Timer Tool: Risks

- License for the development tools I need has expired.
- Database is too expensive.
- Mechanism for internode communication is not supported in Gary's organization.
- Some of Gary's programmer machines not connected to the Net.

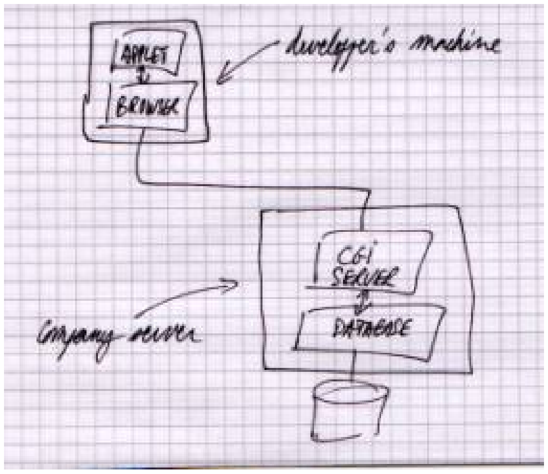
The Business Case

It's now 10:30 a.m., and I have all the information I need to build my initial *Business Case*, so I can begin filling in that last piece of paper. I've already estimated that the project will take four days of my time. I might need to upgrade both my Java compiler and the database software, so I'll mark those things TBD (To Be Determined). I figure that with my usual loading factor and a bit of padding for any bug fixes that might come later, it should be a reasonable deal.

If Gary is reluctant, I could even build a convincing business case from his perspective. If he were to save a half hour per week per developer, plus two hours of data entry and consolidation time for his administrative assistant, he would get his money's worth in less than six months. (I'm even thinking about how I could sell this little program to others through a profit-sharing scheme with Gary, but I'll focus on this some other day. Time is short).

The Architecture

Since Gary had not shown up yet, I go a step further. On a fifth sheet of paper labeled *Architecture*, I do a quick pencil diagram of the major components of the timer software: Java applet, browser, database, and data extractor. It looks like this:



Then, I add a couple of other components, using the Unified Modeling Language (UML). Since the diagram looks cute, and since Gary is himself somewhat software literate, I'll show it to him, too.

The Commitment (Monday Lunch)

To make a long story short, Gary likes it. He pays for lunch. I've brought my five sheets of paper (and my pocket agenda), and we scribble while waiting for the main course.

The bad news is that I did not completely understand the requirements. Gary wants all of his developers to accumulate data in a single database, over their local area network; he doesn't want to have each of them accumulate data in his or her own database, because it's not that easy to merge the data. Also, they don't always work from the same machine, especially when they do testing. We make a few other touch-ups and clarifications to the requirements, but that network feature has me worried. It has consequences for my architecture and requires much more setting up and testing. Plus, we have to identify an administrator to maintain the database.

So, more or less on the fly, I adjust the documents that I prepared this morning.

The Vision, Take Two

I fix the *Vision*, adding that network feature. I also add a couple of ideas for future development that we discussed when I touched on the idea of making a business of this. Although I won't implement them in this round, they might constrain some design choices.

The Plan, Take Two

I decide not to take too many chances. To mitigate the big architectural risk, I shift the LCA milestone (end of Elaboration) to dinner on Tuesday. I plan to do Construction over two days, with two iterations. For the first iteration, on Wednesday I'll make sure the "single person" version works fine and test it, and on Thursday I'll develop the client-server feature over the network and test it. This will shift Transition to Friday, for a final product delivery Friday evening. Gary also wants me to come to his office Friday morning to install the beta version and try it *in situ*.

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
INCEPTION Vision Plan Business Case Risks	Prototype Mitigate risks	**CONSTRUCTION** **Single Person** Design Code Test	**CONSTRUCTION** **Client-Server** Design Code Test	**TRANSITION** Improvements? IOC: show the first beta version	
LCO: OK from G.					
ELABORATION Prototype	LCA: OK from G. Use cases Tests	Design Code Test	Design Code Test	**Delivery**	

The Risk List, Take Two

Now there are five new risks to add:

Personal Timer Tool: Risks, v2

- Synchronization of updates to the database.
- Consistency of activities, projects, and users across multiple machines.
- Access rights policy for administrator and regular users.
- Same user connected from two different machines: Can it occur? What are the consequences?
- Dialog with one user dies for some reason and locks out all other users.

My biggest risk? If things go wrong, I'm jeopardizing the hiking trip I've planned for this weekend.

The Business Case, Take Two

Now we're talking about a full week of work, so I raise my estimate. Gary will have a return on his investment in only eight-and-a-half months, but he thinks a reasonable commitment is to pay me two-fifths of the project fee if I get to the LCA milestone by Tuesday night. He promises to send me a purchase order for the Elaboration phase as soon as he is back in his office.

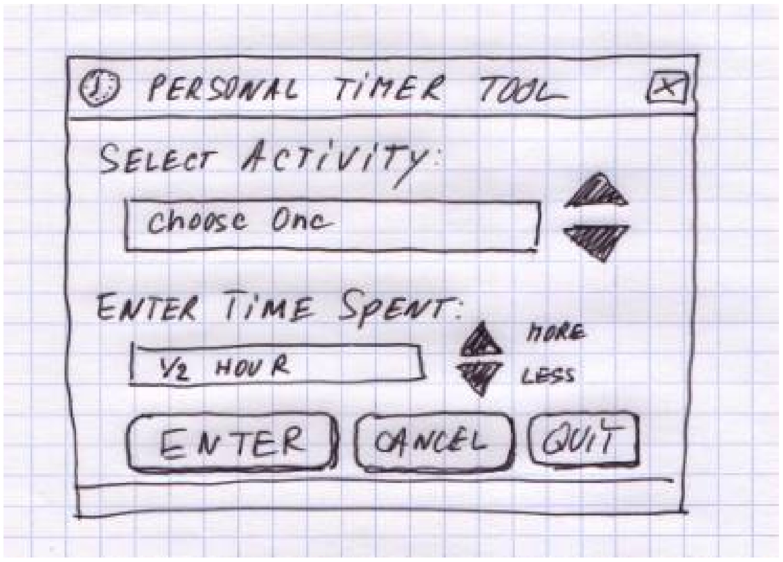
Digging In (Later Monday)

Back in my office, I start looking at more details for the two major use cases:

- Timing an Activity
- Getting a Tally of the Data

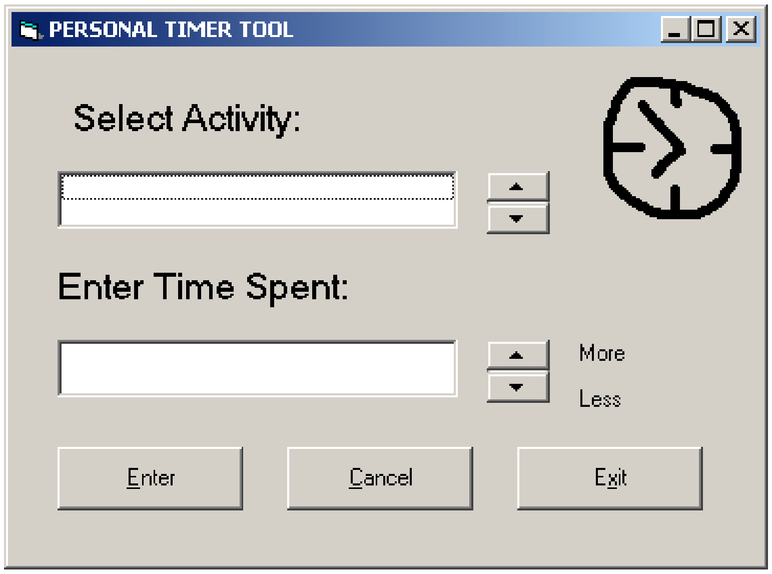
I expand them a bit on two other sheets of paper and build a sequence diagram on my whiteboard.

I'm also starting to have an idea of how I'll design the code in the applet, using three classes. I draw a sketch of what the timer will look like on the screen:



And as I go, I think of more and more questions and issues: Is the activities list predefined and static? (Probably not.) Can each developer create a new list, or only access an existing list? Gary is unavailable, and I can only get hold of his voice mail, so I write down my questions for tomorrow.

By evening, I've built an applet that looks like this on the screen:



I've also succeeded in writing some data for an activity on a text file, with all the testing I could think of (in true XP style). Not too bad for a single day of work.

Pressing On (Tuesday)

This is a real "cold shower, hot shower" day. Every time I cross one risk off my list, I have to add two more. I've made quite a few discoveries today. For lunch I stay in and order a pizza because I can't interface with the database. The thing crashed because my version of the software is too old. Also, I didn't read the API specification carefully enough. I lost an hour with tech support, then downloading the right version, then studying the documents.

I haven't made much progress, and I'm starting to think the whole thing was a bad idea. There are so many things that can go wrong, even with such a ridiculously small app!

I have built a UML model, though. Just a single class diagram and two collaboration diagrams. While I was at it, I also built a component diagram based on my architecture sheet, which I can discard now. I pin all my remaining sheets on the wall.

Since my *Risk List* was starting to look like a mess, I put it on my PC in an Excel file.

I am not changing the Vision, but it is now surrounded by seventeen Post-It notes, all with questions and issues. I start adding constraints, such as:

- The code shall run on Windows NT or Unix.
- The database runs under Windows NT 4.0 or above.

When Gary arrives for dinner with his colleague, Eric, I am putting the latest touches on a not-too-unreasonable prototype. All the data is canned, there is only one user ("Gary") and one activity ("Think"), and I was able to do an on-the-fly "suspend and resume" as an extension to the Timing an Activity use case. The database runs on my desktop with the applet on my laptop, through the Net. My *Risk List* is now down to a few simple ones.

We spend about five minutes playing with the prototype, until Eric crashes it. He's shocked, but I explain that robustness and completeness were not the objectives. Then we discuss the look and feel of the applet and reorganize the fields a bit. We look at my list of questions. Eric is very concerned about losing data if someone has to reboot a machine while a counter is running. I promise to look into it, although I was hoping he would say to forget it. Maybe I shouldn't have brought up the issue.

I end up adding a couple of new risks to my *Risk List* and half a dozen more requirements to my *Vision* document, but that is not too bad. I decide to leave the *Plan* as is. To the *Vision* I add more use cases for "system administration":

- Clean Up Database
- Add a User
- Clean Up Activity List

The good news is that Gary is happy with what he saw, and he says to move ahead with the whole project. He does not object to the constraints.

More Progress, More Changes (Wednesday)

I have found a solution to Eric's problem. Yes!

As I work, I put all my code and tests in a configuration management tool, because I'm afraid I'll make a mistake and lose track of my changes. The plan is simple: Take a complete snapshot of each iteration.

Also, from the use cases I make a more complete list of tests to run.

I now work on a dialog for extracting the data, sorting it, and presenting it in a way that can be digested by Excel to make nice graphs.

Around 11:30 a.m. Eric calls. He forgot one requirement: A person may be working on more than one activity and need to have several counters active at the same time.

Ouch. Change the *Vision*. This one may be difficult, so I add it to the *Risk List*.

Nearing Completion (Thursday)

Testing. Do the network thing. Problems.

I renegotiate requirements with Gary, trading the new one Eric dropped on me yesterday for another one that was on my list. I *have* to do activity+project, because most of Gary's people do work on several projects at a time.

Based on the use cases, I start building a little Web-based *User's Guide* in HTML.

I have so many little things to fix that I need to get better organized. I make a list so I can sort them out. I merge the change requests from Gary and Eric and also add several ideas for improvement.

More testing. First I try capacity testing. No problem. Then I try some concurrency: updating the database from two machines at once. Not good. Synchronization needs some serious rethinking. Error: same user from two machines on same activity+project; there is one entry missing.

Late at night I find the problem. Now, almost everything works.

Beta and Ship (Friday)

In the morning, I go to Gary's company with my first beta version. We install it on several machines, I set up the database and brief his people, and they start playing with it. I run from one to another with my clipboard, writing out suggestions for improvements.

I add to my bug list two major problems and twelve minor ones (mostly matters of taste).

At lunchtime I'm back in my office. I fix all the critical problems and ignore three minor ones. I find another four issues myself, mostly through systematic testing. Finally, the next release is ready. It is numbered 0.9 in my configuration management system. It looks like I will have to go through 0.91, 0.92. I start to despair.

Late at night, I take a break to write some *Release Notes* and prepare a little installer tool. By 1:00 a.m. I am done. I burn a CD-ROM and scream, "Ship! V1.0!!"

I open a pale ale (there's no champagne in the fridge).

The End. Well, for this round anyhow.

Nick's Process

What can we learn about process from Nick's story? Let's take a closer look at what he says and does.

Nick is very aware of ***risks***, both technical (technologies, languages, interfaces, and performance) and business (schedule, expenditure, and missed expectations). He uses an ***iterative process*** to mitigate these risks, rapidly trying out ideas to validate them, to get feedback from his customer, and to avoid painting himself into a corner. He also sets up a ***plan*** with a few well-defined milestones, although the project is only one week long.

Nick has a simple ***business case*** for embarking on this project, with reasonable estimates for both his expenditures and the potential revenue. He revises this business case when he discovers that the basic requirements have changed.

Nick develops a ***design***, beginning with an overall ***architecture***, which he tries out very early. He also does more detailed design in areas for which the right way to go may not be obvious.

Nick tries to make sure he fully understands Gary's needs and vice versa. He tries to ensure that Gary knows exactly what he will get. Rather than jumping right into what he *thinks* Gary needs, Nick dedicates some time to writing down ***requirements***, features, and constraints. Then, he validates this with Gary several times to make sure they share the same ***vision*** of the product.

Nick tries to spend his time on the highest-priority tasks, and he sees that no issue is left aside or ignored for too long. Whenever he finds a problem with the product through a failed test, or whenever Gary comes back with a new request or a better idea, Nick captures this in the form of a ***change request***, and he keeps managing and reprioritizing the change requests to drive his hourly schedule.

Besides the ***code*** of the product, early on Nick develops a set of ***test cases*** that match many of the requirements, and thanks to the iterative process he uses, many of these tests are matured, refined, and improved over time, as he develops the product.

To avoid losing some code by accident (for example, through a disk crash) or under pressure through his own mistake, Nick uses a simple strategy to manage the software, keep a history of versions of his files, and make snapshots of the version sets he tests. He tracks the evolutions and changes that he makes relative to these versions, so he can backtrack to a known ***configuration*** if he makes a mistake.

Finally, Nick writes simple ***user documentation***, with a section that describes the release, how to install it, and its known limitations, one with the ***release notes***, and one describing how to use the product, that is, a ***users' guide***.

This is the essence of the very lightweight engineering process Nick uses. It is a "low ceremony" process that focuses only on a small number of artifacts or workproducts. It does not involve a huge amount of paperwork, since many of the artifacts are stored in various development tools: the configuration management tool, the design tool, office tools, and so on. These few development artifacts produce value not only during the initial product development, but also later on, for additional releases. If Gary were to come back in three months asking Nick to develop a version 2, these artifacts would provide Nick with invaluable information to help him do a better job this time. They'll offer him important cost estimates, partial design or code to reuse, and a few lessons learned -- mistakes he can avoid repeating.

Believe it or not, the simple process Nick followed includes every important step in the Rational Unified Process®(RUP®), and you can easily duplicate it -- without poring over a lot of complicated instructions. Who said that the RUP could only be applied to mammoth projects with hundreds of developers? Here is an example that uses the RUP very effectively for a team of one. Nick refers to this affectionately as his "Personal Unified Process" -- PUP for short.

References

Kent Beck, *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000.

Rational Unified Process, Rational Software, 2002.

Leslee Probasco, "The Ten Essentials of RUP." *The Rational Edge*, December 2000, http://www.therationaledge.com/content/dec_00/f_rup.html

Philippe Kruchten, "What Is the Rational Unified Process?" *The Rational Edge*, January 2001, http://www.therationaledge.com/content/jan_01/f_rup_pk.html

Philippe Kruchten, *The Rational Unified Process: An Introduction*, 2nd ed. Addison-Wesley, 2000.

Humphrey Watts, *Introduction to the Personal Software Process*, Addison-Wesley, 1997.

Notes

[1] Humphrey Watts, *Introduction to the Personal Software Process*. Addison-Wesley, 1997.

For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!