Trading Systems Programmer's Checklist

☑ Step 1: Write out the requirements for the program.

☑ Step 2: Write an outline of the basic flow of the program.

☑ Step 3: Clearly define the definition of a successful program.

☑ Step 4: Ensure most of the features of the program are things you already know how to do.

☑ Step 5: For parts of the program that are new, write smaller working proof-of-concept programs.

☑ Step 6: Get your program language reference guide and other documentation handy.

☑ Step 7: Write out the trade setups and triggers for the trading rules you will be coding.

☑ Step 8: Ensure you know how to use all of the indicator functions you will need.

☑ Step 9: Ensure you know your stop levels, risk levels and profit target levels for your trades.

☑ Step 10: Trade the system manually (or back-test it manually) so that you understand its behavior in real markets

☑ Step 11: Write your code with clear structure, descriptive variable names and good comments.

☑ Step 12: Write out your plan on how to test your software.

☑ Step 13: Get your back-testing tick-level data ready.

☑ Step 14: Divide your back-testing data into an in-sample portion and an out-of-sample portion.

☑ Step 15: Plan your walk-forward testing.

☑ Step 16: Ensure you know which metric you are optimizing for: net profit, win percentage, consecutive wins, minimal drawdown

☑ Step 17: Ensure you know important measurements for your system such as maximum losses in a row, maximum draw-down, maximum loss per trade.

☑ Step 18: Ensure you know how to determine if your system is behaving normally in the markets.

☑ Step 19: Ensure you know what you will do if your system isn't behaving normally. Do you know how to pull the plug?

☑ Step 20: Create a plan for re-optimizing the system every month.

General programming preparation

Step 1: Write out the requirements for the program.

With written requirements, you'll know exactly what you intend to accomplish with your software. Even it if it is as simple as a list of things your program **must** do, it's important to have your goals written out.

Step 2: Write an outline of the basic flow of the program.

An outline or pseudo-code sketch lets you specify clearly what the program will do, and in what order. By breaking down the program into smaller steps, you will be able to determine clear definitions of functions, modules and other parts of the program, as well as general flow.

Step 3: Clearly define the definition of a successful program.

By defining success for the program, you can determine when you are finished. For a trading program, this might match up certain metrics for wins and losses, or profit achieved over a given period of time. Knowing when you are done keeps you working until that point, but not any longer than that.

Step 4: Ensure most of the features of the program are things you already know how to do.

When writing software, usually the majority of the features are things you already know how to do; as you go through your pseudo-code or outline, make a note all of the parts you have done in other programs, or already know well.

Step 5: For parts of the program that are new, write smaller working proof-of-concept programs.

Most new programs you write will have something new in them – the Research and Development part. If you separate these parts out, and write a smaller proof-of-concept program first, you will be able to figure out the hard parts without the complexity of the rest of the bigger program around it. These proof-of-concept or "R&D" programs let you explore a new idea in an isolated context where you can figure things out. Once that's done, you can easily write the hard parts of the larger program using what you have learned.

Step 6: Get your program language reference guide and other documentation handy.

Get your manuals within easy reach, whether they are PDFs, eBooks or paper manuals. Often, you'll want to look up how to do something, or a better way to solve a programming problem. Having the manuals nearby will let you keep going when you are on a roll. When you find something you know you'll need to come back to, bookmark it or copy it to a cheat-sheet so you'll have it handy when you need it again.

See http://www.tradertechtalk.com for more great information on programming

Trading software specific preparation

Step 7: Write out the trade setups and triggers for the trading rules you will be coding.

When coding trading systems, have **all** of the rules written out in front of you; likely this will be a part of the pseudo-code or outline, but it bears emphasizing that you don't want to be struggling with the rules of your system while you are trying to code it. Having the trade setups, triggers, filters, and system rules written out will ensure you get all the pieces written in code. Give yourself enough time for this step. This is the hard work part.

Step 8: Ensure you know how to use all of the indicator functions you will need.

When writing trading systems, you may want to use a new indicator or function that you haven't used before. Make sure you know how the indicator or function works, and that the values it produces make sense. Knowing how to use the indicator manually **before** trying to code it will help your programming a great deal.

Step 9: Ensure you know your stop levels, risk levels and profit target levels for your trades.

Your trading system will have specific stops and profit target levels that work best when trading the system manually. Know what these levels are before coding, and know what your risk tolerance is, so that you know what to expect from your software.

Step 10: Trade the system manually (or back-test it manually) so that you understand its behavior in real markets

Make sure you know the system you are trying to code; if you have not traded it (or parts of it) manually, then use back-testing software to trade the system on historical data. That will give you a sense of how the system will behave in the real

markets. Without this knowledge, you won't know if your system is behaving as it should or if something has gone wrong.

Write your code!

Step 11: Write your code with clear structure, descriptive variable names and good comments.

Code written with good comments and descriptive variable names is **readable.** And readable code is easier to work with than unreadable code. The more descriptions and comments you can put in your code, the easier it is to come back to later to figure out what you were thinking.

Coding is iterative. You'll write some, run the program, check the results, and then come back and refine it, and test some more. At each iteration, the code will become a bit better.

See http://www.tradertechtalk.com for more great information on good coding techniques

Back testing your system

Step 12: Write out your plan on how to test your software.

Writing the program is only half the battle. Knowing how to back-test it is the other half; in fact, it's more than half. A written testing plan ensures that you will test each feature or section of your code. Plan on testing your software in up-trending markets, in down-trending markets, in volatile markets and in flat markets. And know which months or weeks of market data correspond to those different market types. Also plan on how you will keep track of the resulting data. The better your back-testing records are, the faster you'll be able to get to the optimal settings.

Step 13: Get your back-testing tick-level data ready.

Many trading platforms provide you with data to back-test with, but the data is not of the highest quality. In many cases, the data is only minute-level data, rather than tick-level data, which will give you less than accurate results in your back testing. Consider investing in a tick-data service to give you the best chance at testing your program before you run it live.

Step 14: Divide your back-testing data into an in-sample portion and an out-of-sample portion.

When you test your program, you will optimize parameters using some of the historical data, and then test the optimized parameters on new data that your

program hasn't processed yet. The optimization data is "in sample" and the new data is "out of sample". Know the difference, and know why this is important!

Step 15: Plan your walk-forward testing.

As you work with in sample data to tune your program, and out of sample data to test the parameters, ensure you have a plan for how you will roll the window of in sample and out of sample data forward. This is the crux of good back testing.

Step 16: Ensure you know which metric you are optimizing for: net profit, win percentage, consecutive wins, minimal drawdown

When you optimize your trading system, are you looking only at net profit? Or do you consider draw-down as well? How many losses in a row are acceptable? Do you want to maximize profit while minimizing drawdown, and keeping consecutive losses small? Know what you are optimizing for and what your goal is. This goes back to your definition of success above.

Monitoring your system

Step 17: Ensure you know important measurements for your system such as maximum losses in a row, maximum draw-down, maximum loss per trade.

When you back-test, you need to keep track of measurements that will tell you much about your system. How many losses in a row are typical? How much will the system lose before it gains it back? How big of loss is typical? What is the worst case scenario in the back testing? Many trading system testing tools will create reports which will give you some of these numbers. Keep a printed copy of the report handy to refer to when you are monitoring your system's trading behavior.

Step 18: Ensure you know how to determine if your system is behaving normally in the markets.

When you finally run your system in the live markets, you will need to know if it is behaving normally. What is normal? Knowing the metrics from above will help! If you know your system might have 7 losses in a row, and you see 8 losses in a row, is that ok? If in back-testing the worst loss lost $500, and it just lost $750 in a trade, is that ok? You must keep track of all trades that your system makes in the markets in order to determine these numbers. Save the trading history data! Keeping track of this will save you from a ruined account!

Step 19: Ensure you know what you will do if your system isn't behaving normally. Do you know how to pull the plug?

Have a written plan for your "not-to-exceed" metrics. Decide now how many losses in a row you will accept. Decide now how large a loss on one trade you will accept. Decide now how many days without seeing a trade are acceptable. When the system is clearly not behaving as it did in back testing, turn it off, and get back into development mode.

Step 20: Create a plan for re-optimizing the system every month.

While your system is performing well in the live markets and you are collecting the trading data, schedule a monthly tuning day to re-run the parameter optimization using the most recent month's data. By continually optimizing, you ensure the program is running the best it can run for the markets right now.

See http://www.tradertechtalk.com for more great information on back-testing.

Sign up to receive updates at http://www.tradertechtalk.com . I post weekly tips and valuable information on setting up your trading system. Look for my weekly podcast on iTunes! If this has been helpful to you, let me know!

Follow me on Twitter @TraderTechTalk or email me at john@tradertechtalk.com