10 Tips
FOR WORKING WITH CODERS

Understanding how to successfully manage a software development project

BY BOB DAVIS

Ten Tips for Working with Coders

Understanding how to successfully manage a software development project

By Bob Davis

10/10/2017

Table of Contents

Ten Tips for Working with Coders

With the onset of globalization, more and more software development companies overseas are completing more and more projects successfully. However, the land of software development is also fraught with disaster stories of projects gone wrong with missed deadlines and costs soaring over budget. One entrepreneur worked with a software development company for 3 months spending $11k in services, without producing a single line of code! The whole time was spent going back and forth talking about the brand of the company, what they wanted the user flow to be even though he already had the layout mapped out.

Most software development companies are well meaning and don't intend for this type of fiasco to occur. It often stems from poor communication and misguided direction, but there's hope! Below are a ten best practices to ensure your software project is a success.

1. **Find Developers you can TRUST**

As Stephen Covey highlights in his book, "Speed of Trust," business always goes better and faster when you have confidence others will deliver and follow through on their word. Of course, the first component of this is finding a company with integrity. Not only is this essential for believing what they tell you, but is important when they tell you something you don't want to hear. For example, your next favorite feature will take weeks instead of days to build as you thought. The magic of software is that very complex tasks can appear very simple because the end result just happens in the blink of an eye. For example, Google's PageRank algorithm displays the top webpages almost instantly, but it has taken thousands of people-hours of development time. Yet the result is available at your fingertips at any moment throughout the day and appears as a simple list of websites.

The second component of trust which is often overlooked is the value of competence. The team you're working with could have the highest degree of integrity and wouldn't lie if their job depended on it, but if they aren't capable, then you could still end up being left hanging. The world of software is rapidly changing, and there is always a shiny new piece of code that developers want to learn and use. It is very valuable to stay up on the latest trends, but if a developer hasn't completed a very similar project in the past, there will necessarily be some time spent figuring out how to piece together the right tools. Finding a development team with the right skill set is very valuable to reducing the overall time it takes to build a mobile or web app. This goes beyond just

having experience with a particular programming language but rather having worked on similar projects with the same toolset.

Finding a development team who has your best interest in mind, a great work ethic, coding quickly while maintaining optimal quality can make or break a project. A lot of time and money can be saved by identifying the minimum feature set or removing non-essential items in the first build. However, without significant technical knowledge, it can be difficult to undercover those potential time savers. Therefore, having your own trusted technical advisor can be very advantageous. Someone who is able and willing to work closely with the development to make hundreds of those micro-decisions such as what to include within each feature to balance speed and quality.

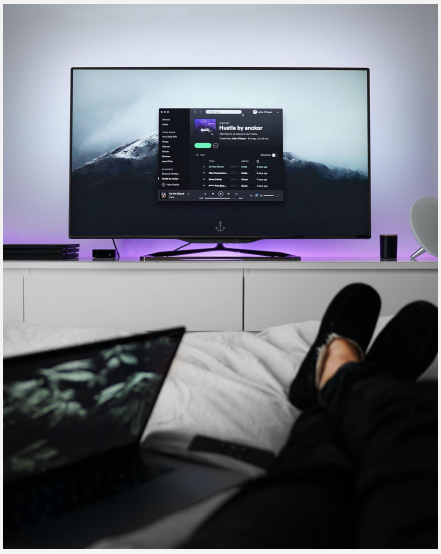
2. Clarity is Power

> Cat: Where are you going?
> Alice: Which way should I go?
> Cat: That depends on where you are going.
> Alice: I don't know.
> Cat: Then it doesn't matter which way you go."

Many people working on software development teams' end up feeling like Alice lost in wonderland. The sky's the limit when it comes to what is possible to build with software making it very difficult to explain exactly what you want. Most people have a clear picture in their mind of what they want to create but lack the words to express those ideas to the people who can bring their vision to life.

This is because the ideas are very complex and abstract making it difficult to create. Imagine trying to describe to an architect how you want to build your house without drawing anything. How difficult would that be?

Furthermore, imagine beginning construction without any blueprint and just making it up as you go? Well-meaning people with great ideas for software can see their dreams pushed off by meetings and endless abstract discussions about features before any code is written.



The corollary to a building architectural blueprint in software is a mockup and requirements specification. A good mockup contains an outline of what will be on each screen and tell a story of what each users experience will be. For example, a user story for bestbuy.com could be:

1. Enter homepage: bestbuy.com
2. Type: Big screen TV into the search box
3. Browse through a list of available TVs
4. Select 3 to compare the features
5. View each in detail
6. Purchase one TV

This is a high-level overview of a user story, and in reality, each and every screen should be listed. For the mockup, you would then layout what information is needed on each screen everything from the sale item details to what should be displayed in the search results or comparison page. Not to mention, what other ads, banners or links should be included and what information do you want to collect from the user. As you can see, this can get quite complex very quickly.

The power of a mockup is that it takes a nebulous idea from one's mind and puts it in the concrete form on "paper" for all to see. Once this is done, you can actually interact with the mockup and see if the flow makes sense and discover any details that were left out.

Like Google's PageRank, many apps have a lot going on behind the scenes which are not captured by a good mockup. Instead, this is outlined through the technical requirements specification. These specs outline how the software works behind the scenes, often referred to as the "backend" opposed to the "frontend" which covers the items that can be seen and interacted with directly by the user. To continue with the construction analogy, the backend is analogous to the foundation, framing, roof, etc. A good backend design is like a proper structural engineering design to ensure the support beams are strong enough to hold up each floor and stand up to inclement weather. Just like no one wants a building to fall, it's important you find excellent backend engineer(s) to keep your app up and running. Beyond a detailed mockup outlining how the app should look and feel, a detailed requirements specification is helpful to ensure the app won't fall apart or be plagued with errors when all complete.

In short, the more prepared you are with a detailed plan of what you want to build before you meet with a software developer(s) the more likely your project will be completed on time and within budget.

3. Embrace the Dynamic Nature of Software Development

One difference between constructing a house and creating software is the ability to make changes after the initial setup is completed. After a building is finished, it is very difficult to change the place of the kitchen or bedroom. With software, the parts which are difficult to change are highly dependent on how it is architected and which parts of the code are dependent on other parts. The more dependencies, the more difficult it can be to modify the code. Nonetheless, the code is more dynamic than a physical building and therefore allows for more flexibility. At risk of overusing the analogy, with software after you complete the kitchen you could easily add a bathroom or extra bedroom without much overhead.

The main point is to recognize that things change and you will learn more as progress is made. Therefore hold on loosely to the overall timeline and deliverables and focus on the tactical week by week how can the team most effectively march toward the end goal? What are the features that you can put off to focus on obtaining a fully functional prototype as fast as possible? You always want to be asking, what is faster to complete and what is the easiest way for the developer to get the job done.

4. Welcome to Agile Development

If you've spent much time searching for what you need to know when working with software developers, you've probably heard of the project management style known as Agile Development. It took the software industry by storm in the early 2000s, allowing teams to break away from a very detailed plan and work more on an ad-hoc basis. Eventually, they even put together an Agile Manifesto (agilemanifesto.org).

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan
That is, while there is value in the items on
the right, we value the items on the left more.

As highlighted in the prior section having a clear vision and plan for what you want to accomplish is essential to ensure you get there. However, in the 1990s, the software development industry went a little overboard attempting to create very detailed and precise project timelines. A lot of time was consumed trying to plan out every attribute of a large multi-year project. It became very difficult to hit deadlines and meet specific requirements that were six months out. Oftentimes, more time was spent on the planning phase and on documentation than actually writing code.

Hence the birth of Agile development, which attempted to add more flexibility to the process. There is no "silver bullet, " but an important balance should be struck between spelling out a clear vision and getting your hands on working software which can be improved upon as you go. There are many opinions about how to manage a project and implement an agile environment, so one key to working with a development company is learning what works well for them and what systems they have in place to ensure progress is made. One popular agile

implementation is the Scrum, which consists of sprint planning, sprint review, and daily standups.

5. Establish clear weekly goals

Although the origins of the Scrum process, mentioned above, are not clear, the first book was published in 2001, called Agile Software Development Scrum Series. The Scrum outlined a framework or process to help drive projects forward and ensure the right things are prioritized and everything is accomplished in a timely manner.

One of the key components of this process is the Sprint Planning meeting. This meeting occurs every one to four weeks, depending on the organization, and biweekly meetings are the most common timeframe. The work to be completed each week is broken into “User Stories” which explain what the code will do from the user’s perspective and are the specific deliverables that the developer needs to complete at the end of the sprint. The meeting workflow goes as follows.

Sprint Planning Overview

- The Product Manager (or Scrum Master) will review the list of prior user stories (aka tasks) to be completed and prioritize them
- Developers will review the list and individually estimate how long each user story will take to complete and then reach a consensus
    - This is often done in terms of complexity (aka difficulty) rather than specific hours
    - Then as the development process progresses, the number of complexity points completed per sprint can be measured and refined over time
- Everyone meets to review this list, agree on the contents and priority, thus solidifying the plan for the upcoming sprint
    - This provides a great opportunity for any questions to be answered
    - It is also an excellent time to level set expectations and ensure progress is being made fast enough to hit the long-term goals
- Meeting notes are sent out to summarize what was agreed on and document the plan

One benefit is clear visibility to all the stakeholders about what is being worked on and what can be accomplished over the given timeframe. It also provides clear, realistic goals for the developers to pursue and an easy way to hold them accountable. A best practice is to determine what the consequences will be if all of the user stories are not completed by this timeframe. If people are not delivering on what they say they will accomplish, you need a plan of action to ensure that any incorrect systems or behavior is remedied as fast as possible. It can be devastating to a project if a habit of not completing tasks in the specified timeframe is formed.

Along those lines having a properly prioritized list of additional tasks to complete (often referred to as a "backlog") can be valuable to make it clear what to work on if the items for the sprint are finished ahead of time.

There are many software tools available to help keep track of the project status and the list of items to be completed. A few of the most popular ones are Atlassian's JIRA, ProjectTracker or Github Issues (see http://blog.capterra.com/agile-project-management-software/ for more options). They each have their own pros and cons, so it is often best to go with whichever the software development company prefers.

6. Daily Standup

As in many areas in life, rarely will a project get completed on its own without outside help. You need to water your plants every day, or they won't grow. Since things are frequently changing and many small issues can arise while working on a software project, it is very important to have open pathways of communication to answer any questions that come up. That is where the daily standup comes in. The idea is to have a regular review of the progress and provide a time to answer any questions which arise. It should be short, simple and to the point:

Standup Meeting Agenda

Each person highlights:

1. What they accomplished yesterday?
2. What are they going to tackle today?
3. Are there any questions or things holding them back from making progress?

The meeting should take no more than 5 minutes and have no more than 5 people in attendance. Anything larger and the work scope is too large, so people often don't need to hear about what everyone in the room is doing, so they lose interest and become

unmotivated. It will take significant discipline to build a habit of having a short meeting, so you might want to start by using a timer and setting it to 1 minute for each person for the first couple of weeks until a habit is formed.

This can also be a good time to ask if the developers are on track to accomplish all the tasks (user stories) for this sprint. If not, it can be advantageous to discuss ahead of time what to do if they fall behind. Which items can be dropped? Can we pull in more resources to ensure we meet the deadlines, etc.?

7. Sprint Retrospective

If you've recorded a clear list of user stories to be completed each sprint and met daily to track progress, then the retrospective will likely go very smooth. It is a time to review the work previously accomplished and ask, did we get done all the items planned for this sprint? If not, why not? Then talk about what went well and what can be improved upon. It is always good to highlight the positive things first, so each person should come to the meeting with at least 2-3 things that went well that week. If there is anything that could be changed this is a good time to bring it up. Perhaps someone wasn't responsive, or some areas turned out to be more challenging than expected, etc. The important part is to ask, what will we do differently this time to improve in this area? Is there something we need to stop or start doing? What worked well that we should continue doing?

The retrospective meeting can usually be completed back to back with the sprint planning meeting or at least within 24 hours of each other, so the learning easily transfers to the next sprint.

It can also be useful to have a project overview meeting every one to three months. This is where the long-term goals of the project can be reviewed, and the pace can be assessed to ensure the team is on track to reach those goals. Also, it can be helpful to take a step back from the tactical sprint planning process and ensure the day-to-day details align with the big picture. This meeting could be an extended sprint planning meeting or an additional meeting on its own.

8. Asynchronous Communication Methods

In addition to these daily meetings, it can be very useful to have an easy way to communicate and ask questions about each user story or technical requirements without requiring a meeting. If the only time, the team speaks with each other is in a

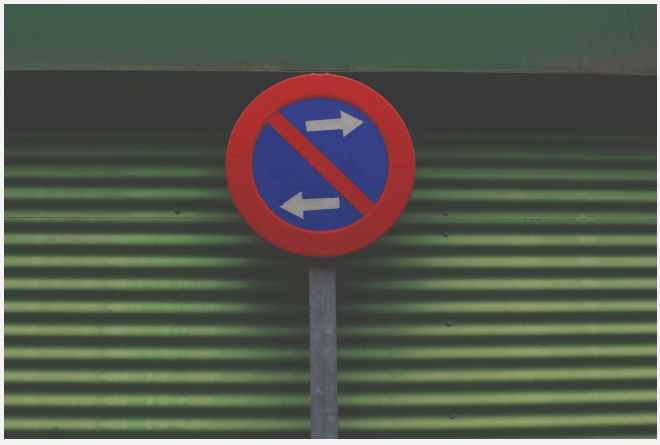
meeting, then they become long and laborious. Email is the defacto choice for communication but how many people like their inbox being overloaded? It can be preferable to have a tracking system which provides more context for each user story but integrates with your current communication system. For example, PivotalTracker can send you a slack message or an email if your name is mentioned within the list. Slack is another popular choice for ways to provide easy communication back and forth. A separate place for discussions also makes it easy to search through old conversations.

Having a way to communicate that doesn't clutter up your inbox is really helpful to ensure work is progressing and any roadblocks are being eliminated quickly. Without an easy, effective way to talk to someone, small problems can build up into tantamount issues that carry on for weeks or months.

9. Be Consistent

Every new idea evolves with time, especially as you see it working in real life. The simple act of using the software can lead you to discover many new insights about the way it should work. For example, a button here or too many options to fill out there or the address should be pre-populated when filling out a form, etc. It's nearly impossible to anticipate all these minute details hence the value of iterating quickly and operating in short sprints so you can test out new features as soon as possible.

Nonetheless, this is not an excuse to constantly change your mind about what you want to build. There are some matters of preference such as the color of buttons or the layout on a screen that can feel really important at the time but can drastically slow down development if they keep changing. Remember that ***done*** is better than *perfect* and each change costs time and money, no matter how small. Therefore, if it's good enough then leave it for now and feel free to make a note of it so you can come back to it later.



In general, it's best to focus on the large items first, like what content should be displayed on each page and what the layout of the content should be, before worrying about the finer details. Start big and then zoom into the details as work progresses.

10. Expect Excellent Code Quality

Not all code is created equal. Cutting corners, in the beginning, can lead to compounding problems in the future. When just interacting with the software or using it at first, it can be difficult to tell how good of quality the code is behind the scenes. It's important to find experienced developers who appreciate some of the best-known methods in the industry. Here are a few concepts to ask your team about and expect them to deliver on:

Test Coverage - If you ever worked with buggy software you see the result of poor test coverage. The beauty of code is that each part can be tested individually before they are put together, much like the parts of a car are tested separately. There are various levels of testing depending on the scope ranging from unit tests, integration tests to functional tests. The important thing is to ask your developer, how high is the test coverage? In other words, what percentage of the lines of code are covered by automated tests? They should tell you a number between 0% and 100%. The costs of writing more tests can increase significantly going from 95% to 100%, but in general, 85% to 95% is a solid target.

Code Linting - Although a piece of code might function well and get the job done, if no one else can read it then you'll suffer tremendously in the long run. It will be more difficult to debug (fix problems when they come up) and bring on other developers. Similar to checking for Test Coverage there are libraries which will check how well the code format conforms to language specific standards.

Code Reviews - Another way to improve the quality of the code is to ensure that no piece of code is committed without a peer review. Code is written cleaner when the developer knows someone else will be examining it. Also, it helps answer any questions and ensure other people understand what a piece of code does besides the one person who wrote it. The person paying for the software can hold people to this standard and ensure it's followed.

Continuous Integration - Although not required, it can be very beneficial to set up an automated testing and delivery system known as continuous integration. These frameworks

will automatically test, build and release the code to the server as soon as the developer commits it to the code repository (such as Github or BitBucket). This saves a lot of developer time debugging why their code works on their own machine but not on the server. Also, it helps ensure the tests are run after each change. Tests aren't helpful if they don't get run.

Technical Debt - Writing code can be like storing stuff in your garage. Each time a new toy is purchased, it gets placed in the garage somewhere, and some are taken out and put back in. Over time the garage can get filled with stuff with no more room for a vehicle! Technical debt is when code gets disorganized and cluttered resulting in a lot of overhead to get anything done. It's like trying to find that set of china buried behind all the boxes in the garage. Just like with a garage, you can let the junk pile up for months or years and then spend weeks clearing it out, or it can be cleaned little by little each week and organized as new stuff comes in. As more code is written more time needs to be dedicated to cleaning up the old code and organizing it otherwise someday those looking to add new features might get lost in the tangled web of bad code. In short, give developers time to organize and clean out that old pesky technical debt through activities like refactoring (think to organize a messy closet), rewriting and adding tests. There might not be any visible changes on the frontend, but in the long run, it will pay dividends!

In summary, it is easy to get lost in the wild west of software development, so it is invaluable to have solid principles to guide you along the way. Remember to provide a clear vision, be flexible, stick to proven methodologies and don't settle for the stinky code. If this territory feels a bit overwhelming, it can be helpful to bring in a third party with proven development skills to act as an advisor through the process.

Best of luck on your next project!