The Effective Developer

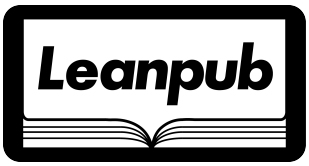
The Habits, Practices, Skills & Disciplines of the Modern Software Developer

Chris Edwards

The Effective Developer

The Habits, Practices, Skills & Disciplines of the Modern Software Developer

This version was published on 2012-10-03



This is a Leanpub book, for sale at:

http://leanpub.com/TheEffectiveDeveloper

Leanpub helps authors to self-publish in-progress ebooks. We call this idea Lean Publishing. To learn more about Lean Publishing, go to: http://leanpub.com/manifesto

To learn more about Leanpub, go to: http://leanpub.com

Contents

Preface

Several years ago, I attended a presentation at the Austin Code Camp that impacted me so much that I This book is the result of a series of events that began when I attended a presentation at the Austin Code Camp several years ago here in Austin, TX. It was in introduction to Test Driven Development (TDD) given by Mahendra Mavani. Honestly, I wouldn't have attended the session if I knew it was an introduction. I had used TDD for years, so an introductory course wasn't going to be that helpful to me. But I am sure glad I was there.

The questions everyone asked shocked me. They wanted to know the basics–things I assumed everyone already knew. I took it for granted that everyone knew TDD, but these developers had never tried it; some had never even written unit tests before. In this flood of sessions on advanced topics and new technologies, this room-full of developers desperately thirst for the basics. The irony hit me like a ton of bricks. What good were all our advanced sessions to them? Sure, I love to give talks on new stuff just as much as the next guy, but this talk opened my eyes to a greater need.

Over the next couple of years I started giving a presentation called *"Padawan to Jedi - A Developer's Jump Start"* (Yes, I admit itâ€¦I am a Star Wars fanboy). I designed the talk to cover a broad spectrum of all the things that I felt a modern developer needed to know to be effective. It didn't take long for me to realize that's a lot to cover! I only had two hours to talk, so I couldn't go into much detail. To make up for it, I decided to provide links where the listeners could learn more about each topic. That way they could get the detail if they wanted it, and I could cover more in my talk.

The first time I gave the talk I finished in just under two hours. I must have had a ton of caffeine because when I listened to the recording afterward, I was talking really fast. I barely finished.

A year later I gave the talk again. However, I had learned a lot in that year–things I felt were valuable–so I added them to the talk. To my dismay, I only got half-way through the talk before I ran out of time. I talked for two solid hours and wasn't even half done! But the audience was engaged and asking questions the entire time. When it was over, many of them said they really wanted moreâ€¦

So here it is! This book is a distillation of things I have learned in the last twenty years of working as a professional software developer. I have tried to include everything I feel a developer needs to be effective. It is by no means complete, and definitely not perfect. But this is the book I wish someone had handed me the day I started developing software. And so now I hand it to you.

There is something here for everyone–whether you are an architect, senior developer, team lead, or newcomer to programming. But it's primarily written for those who want to learn the basics of modern development practices and get a taste of how effective developers work. Some of the topics are very basic. Some are just plain common sense. If you already understand something, feel free to skip it. You don't have to read this book in order. Read only the parts you need, when you need them, or read it straight through. Whatever works best for you.

It is my desire and prayer that this book helps you become a more effective developer. If you don't even know where to start, don't fear, you have taken a good first step.

God bless you,

Chris Edwards

Part 1: Soft Skills

In part 1 I am going to discuss what I feel is the most important aspect of an effective developer: *Soft skills*. Soft skills are the behaviors, habits and personality traits that characterize your relationships with other people–sometimes referred to as *people skills* or *social skills*. Soft skills complement your hard skills, which are your technical proficiencies and the capability to do your job.

To be truly effective, you have to have both types of skills. Hard skills are easy to attain. Soft skills are hard to attain. Better to hire someone with soft skills. Can train hard skills. Soft skills are severely undervalued. Soft skills enhance effectiveness. More than hard skills. Soft skills result in teachability. Soft skills benefit others. Hard skills benefit self, and only benefit others if taught (using soft skills).

Because most developers focus solely on hard skills, it is easy to find technically proficient people. Yet finding those equally proficient in soft skills is difficult.

But they will enhance your effectiveness more than any technology you can learn.

I believe so strongly that soft skills are important that I have placed them at the beginning of this book. *First thing first!*

Chapter 1: What makes a great developer?

So what makes a developer great? Here are some attributes I find common in truly great developers.

- Passion
- Continuous Learning
- Humility
- Communication Skills

Passion

Passion is a developer's strongest asset. It motivates all kinds of positive behavior. Nothing drives effectiveness like passion.

Meet Bill. Bill works 9-5. He's a hard worker. After work he spends time with his family then reads some fiction or raids a dungeon with his online friends. Most often the latter. Last night he reached level 80 with his mage. He fell asleep thinking about how close he is to getting the next level armor for his mage.

Now meet Bob. Bob also works 9-5 and is a hard worker. Bob listens to software development podcasts on his daily commute. He spends lunch reading blogs or technical articles, except Wednesdays when he meets with co-workers who are going through a TDD book together. After work he spends time with the family then either reads a programming book or writes some code for an open source project he's contributing to. Last night he researched a new mocking framework because he heard a co-worker mention it. He even got some sample code up and running. Bob fell asleep weighing the pros and cons of using the framework on an upcoming project.

(Might be overdoing Bob..sad person? Needs other interests. People can't relate to Bob, but more to Bill. People don't want to be like Bob. Make Bob more like me. Not exclusive to software development. Too nerdy. Need to express personality. Perhaps carry character through scenarios and stories through the book. Maybe start Bob off with less technical skills but more soft skills like passion…we follow Bob on his journey grow into an effective developer. Depict him as a likable guy. Can show Bob working for a bad leader with poor soft skills.)

Consider John and Bob. Who would you say is more passionate? Why?

Who would you want on your team?

Did you even notice I didn't say anything about their code quality? Did you assume one is better?

Benefits of passion

Passion drives so many positive behaviors. People with passion perform better without even thinking about it. The benefits of passion are great..and contribute to making the person great.

The developer with passion has a great advantage over other developers because he enjoys what he does. When something becomes fun, it ceases to be work. He will put in the extra hours to learn something without thinking about it, but it's not work. It's fun. Rather than playing computer games, he will further his career. You see, the passionate developer's hobby is his job.

Passion also provides a great advantage in times of difficulty. When the going gets hard, passion provides the motivation to hang in there and tough it out. It stops you from giving up on becoming great.

What is passion?

So you want to be like Bob. You want that kind of driving passion? First let's look at what passion is. It's pretty simple really. Its a desire to do something. It's a burning desire to do it. (does this even belong)

Do I have to be born with passion?

You are probably wondering, "Do I have to be born with passion? Or can I cultivate it?" I think the answer is obvious. No one is born passionate about something. They acquire that passion somewhere along the way. It wasn't until I was 12 or so that I began to have a passion for computers (and later, software development). Prior to that, I didn't know anything about computers, so how could I be passionate about it? So no, its not something you're born with. And yes, you can cultivate your passion.

How do I cultivate passion?

So we see that we aren't born with passion, so how to you get it? How do you cultivate it? Let's begin with a story.

When Billy was little, he took guitar lessons. He liked the guitar, but when he played it, it didn't sound to good. It was a lot of work, and he quickly got bored of it and gave up. Later in life, he had a friend that played very well, so Billy decided to try again (with a little encouragement from his friend). The first few weeks weren't so fun. However, after learning how to play his favorite song, (song?), he found it was much more fun. Anytime he got discouraged and had a hard time playing something, he would just break out singing the chorus (to his dog's dismay) as he played the song again. This got him though the more difficult lessons and Billy became a very accomplished guitar player.

So, did you notice the turning point in Billy's guitar career? Everything changed when he started to enjoy playing it. It's hard to enjoy foul notes and poor rhythm. But once you acquire some skills and get a little better, your enjoyment increases. When that happens, it drives you to practice and learn more. The effect is cyclical and builds upon itself. This is how passion works. It's a self-feeding (something).

Knowing this, I think we can now look at some logical ways to cultivate passion.

Things to do

Find something fun. Because enjoyment is a key part of passion, you need to find some aspect of software development that you really enjoy. Do you like working on the UI? How about writing tests? Try to spend some time on the things you enjoy. If you can't find time to do those things on the job, try them at home or over your lunch hour. Put some joy in your job.

Experiment Try some different ways of doing things you already do. Finding a better way to do something gives you a boost of excitement and also helps expand your understanding of whatever you are doing.

Learn something new. Learn a new framework, or new language. This helps get you interested and engages your thinking. It may change the way you think about existing problems you have to solve at work. Learning breeds excitement…which feeds passion in a huge way.

Change projects/job Probably the first thing you need to ask is "Am I happy here?" or "Do I even like software development". While passion doesn't require you to love it to start, if you don't even like it a little bit, you will be hard pressed to gain the kind of passion I am talking about. Perhaps you need to change careers. Think about where you work now. Do you like the company? Do you have opportunities to work on something "fun"? If not, maybe you need to find a new place to work.

Deliberate Practice This topic could fill a chapter in itself, but I present it here because it helps feed passion. If you practice your craft, you will get better. Getting better feeds your confidence which feeds your enjoyment which feeds your passion which feeds your desire to practice more. You get the picture.

The bottom line is what my dad always used to tell me (though I was tired of hearing it), "Practice makes perfect." Well, we may not be seeking perfection, we do want to get better. Practice makes you better. If you want to be effective, you have to constantly improve.

Deliberate Practice refers to your ability to intentionally practice with the goal of getting better. Practice those things that are impeding you from being more effective. There are many tools to help you. There are Code Katas available that have you solve a problem in code, as a practice exercise. You can find some at (urls).

Know/Avoid Draining Weaknesses Know what drains you Minimize time on actions that drain you. Difference between weakness you need to work on and weakness that drains you. Examples: - Difficult Co-workers - Mundane Tasks (automate) - Working in an area you aren't excited about (UI vs. Business Logic)

Be Enthusiastic Let your passion be seen by others. Its infectious. You may influence others to be as excited as yourself.

Try to be around other enthusiastic people Try to be around other enthusiastic people, because it will help you be more excited. Passion feeds passion. A roomful of passionate people is a sight to behold. Here are some ideas of ways to surround yourself with other passionate people. - Attend user groups or conferences - Read blogs - Follow passionate people on twitter.

Seek role models Look for someone who can serve as a role model. Someone who has achieved what you seek to achieve and learn from them. Someone you can look up to. How did they get where they are? Can you follow in their footsteps.

```
Find others to encourage/praise you
Being around these kind of people motivate you
The opposite kind of people will drain you.
```

Be obsessed?

Focus on strengths I used to always think I had to focus on the areas I am weakest in. And I do feel this is appropriate in some areas, but one thing that feeds passion is working in an area of strength. If you feel confident, you are more likely to feel fulfilled. Try to work on things that fulfill you. Try to work in an area of strength.

If you are unsure where your strengths lie, here are some questions that may lead you to it. - What fulfills you? - What to others tend to compliment you on? - What are you feel most confident doing?

The areas I am talking about here (for focus) are smaller subareas of your career focus. For instance UI vs. backend code, Security vs. application code, etc. All of these areas may fall under the role of a software developer, but there is leeway in that title that allows you to focus on many different areas. Some will feed your passion. Some will drain it. Sometimes you will have a choice. When you do, choose wisely.

As a side note, be aware that working in an area of weakness is necessary to grow into a fully effective developer. You need to constantly be learning new things. The areas of weakness I am referring to here are areas that drain your passion, not areas of ignorance. Be deliberate about eliminating ignorance by learning. Expand… What fulfills you What to people compliment you on? Not necessarily what you are already good at.

Things to avoid

Just as there are things that feed your passion, there are some that drain it. Be aware of these things and avoid them when possible. Here is a possible list:

Working in an area of weakness You need to feel confident to spark your passion. At the least, you need to feel as if you are growing in the area you are working in. However, some areas are just plain weaknesses that you are uncomfortable and unhappy working in. For instance, if I were placed in a

sales position where I had to talk to people day in and day out to make sales, I would fail. I may learn to get better…but why would I want to do that? Its an area of weakness that I have no intention of improving in. It saps my passion.

Don't confuse what I am saying here. There are weaknesses you will have that you want to work on. And working on them will take you out of your comfort zone. This is called learning and growing. It's good for you. This is not a weakness to avoid. The kind of weaknesses I am referring to here are those that really pull you down and you have decided not to work on. Like taking tech support calls, or writing pure database code. If you really don't want to do these things because you dread going to work every day when you know you have to do them…its time for a change.

Difficult relationships at work This could be a coworker or your boss. One difficult personality can bring down a whole team. Seek to change teams or managers if you need to. Worst case scenario, its time to get your resume ready.

Too much pressure/ stress We all find ourselves in this situation at one time or another in our career. Under stress, you don't have time to do anything but work. There is no learning, and you aren't given time to do a a good job you can feel good about. Hence, your passion will suffer.

If you find yourself in this scenario, try to step back and objectively analyze the situation. Is it temporary? If so, just hang in there and keep positive. If it's a persistent problem then determine if there's something you can do to change the organization. Talk with your manager/team lead about it. As a last resort…yep…you guessed it…resume time.

Working on something you don't enjoy So you hate working on UI code, and wouldn't you know it? That's what you are stuck doing day in and day out. I have come across many people in the same situation. I have met many of them in interviews…as they were looking for a way out. Do what you can to get reassigned. Realize that sometimes you have to work on stuff you don't like…just don't let it become your permanent job description.

Lack of control over your work We need freedom to express ourselves in code. Sometimes we have oppressive co-workers or managers that tightly control every aspect of our job that we feel more like slaves than employees. Flee this. Some control is ok, but if it stifles your ability to work, seek a remedy. What you work on How you work When you work How long you work

Boredom Sometimes, you just aren't challenged. Perhaps you have solved all the problems so well and automated everything so perfectly that the project practically runs itself. What was once exciting is now so simple its boring. Time to move on.

Negativity/Discouragement You really need to be around people that encourage you and push you to be better. If you are in a negative environment, or around negative co-workers, it will affect you. Do all you can to get away from these kind of projects/people.

Warning: Passion can lead to imbalance

> *The effective developer is passionate about his work. He loves what he does and is therefore driven to do it well.*

If you really want to be great at something, you have to love doing it. How can you be motivated to excel at something you don't enjoy? A healthy passion for your work provides a wellspring of motivation. It drives you to do your best, and constantly improve your best. Just think of the advantage this gives you.

You see, we tend to do the things we enjoy. We think about them, read about them, and practice them–because we like them; we call them hobbies. A passionate developer's hobby is his job. Because he loves it, he is driven to do it well. As you can guess, I am passionate about software development. I love reading a good tech book or blog, writing code or writing this book. These activities sharpen my skills, but they don't feel like work. I enjoy them and they come naturally to me. I love what I do and I do what I love. This is the biggest secret to my success.

Sidebar: A warning about passion!

Be aware that passion can lead to an unbalanced life. Its easy to spend too much time on something you love doing. Resist that temptation. Don't neglect the important areas of your life, like family, friends, church, etc. These are essential for happiness, and they are far more important than work. It's tempting to believe that happiness can come from work alone. However, that kind of happiness is fleeting; it's a lie; burnout and sadness soon take its place.

I will never forget what a wise friend once told me. He said, "I work to live, I don't live to work". This should be true for all of us.

When I interview developers, one of the most important things I look for is passion. I will hire a passionate developer who is lacking technically. I know their passion more than compensates for their deficiencies. Because passion can have such a profound affect on the ability to learn and grow as a developer, I believe it is one of the strongest assets you can have.

Continuous Learning

> *The effective developer is committed to continuous learning–deliberately acquiring new skills and refining existing ones.*

A healthy passion for your work, drives you to continuously learn more and improve your skills. (elaborate)

The software development industry evolves constantly. It's not easy to keep up. Technologies change quickly–what you know today will be obsolete tomorrow. You must constantly refine your skills and learn new ones or risk being left behind. If you do not enjoy reading, listening to others, and learning from others, you will be at a huge disadvantage to those who do.

So how can you keep up with all the changes and new technologies in this fast changing industry? Here are some ideas:

Read Books

Reading books is one of the oldest ways to learn, and it is applicable now more than ever. Books are a great way to get a full understanding of a particular area or technology. They are usually structured in such that they cover all aspects of a topic. If you are new to a technology, grabbing a good book on the subject is one of the best ways to get up to speed. Books often make an assumption that the reader has a certain level of knowledge, so the book fills all the gaps that are necessary and tries to present the material in a certain order that will make sense to the reader and build upon concepts progressively. Books are great to establish an initial foundation of knowledge that you can build upon.

Select Books Carefully

How do you select which books to read? Time is precious. If you are going to take the time to read a book, make sure it's a good one. Research what others have said about books in the area you are seeking to learn. Find out what books are the "must reads" in that area and pick them up and read them. It is said that the most successful people read and average of ??? books per year (Need citation).

Take Notes or Highlight Your Book

To make the most of reading books, take notes, or highlight the text. Not only does this require you to think a little more (about what to write) but it also involves more motor skills which improves your retention of that information (citation). Furthermore, after reading the book, you can go back and skim your notes to refresh what you may have forgotten.

Read a Book Repeatedly

Don't be afraid to read a book more than once. You will likely get much more out of the book on your second pass through. You see, you usually can't assimilate all the information from a book the first time through. The concepts it presents often depend upon one another. You can't fully understand *Concept C* until you first understand *Concept A* and *Concept B*. Because your first time through a book, you are just learning these concepts for the first time, you may fully understand *Concept A*, but only partially understand *Concept B*. Consequently, you cannot fully understand *Concept C* or any concepts that depend upon it.

After reading the book the first time, and especially after putting the knowledge you gained into practice, you will have a better understanding of the basic concepts. Then, when you read the book a second time, you will have more experience and knowledge to relate the information to. This will

allow you to understand deeper concepts expressed in the book that you may have missed the first time.

Really good books can be read multiple times and still provide a new depth of information each time. (A great example of this is Domain Driven Design by Eric Evans. I have read this book multiple times and have learned a great deal each time I did.)

As a final note…when you do read a really good book, if you see someone who could benefit from it, please recommend it to them. Be a catalyst to help others learn.

Read Blogs

While books are great to get a full rounded understanding of a topic, blogs are a good way to continually gain deeper, more specific knowledge of topics. Each blog entry is usually written about a specific, focused subject. Sometimes you may find a series of blog posts that serve as an introduction to some topic…but these are not as common. Following blogs over time will help you to keep up with changes to technologies and keep yourself on the bleeding edge. Unlike books, which take time to write and publish, blog entries can be written very quickly and therefore contain knowledge that may not have had time to be published. Blogs are the bleeding edge of thought in the software world (and most others as well).

However, because blog entries are so small, focused, and released quickly, they usually assume the reader knows a great deal about the topic, so readers new to the technology may not know enough for the article to be useful. This is why books are better for learning a new technology. You can do it with blogs, but the order in which you read blog entries may not be beneficial to you.

Once you have gained an understanding of the fundamentals via books (or other means), blogs will likely serve as the means by which you understand more advanced topics. I read blogs every day. I have learned more from them than any other source. Do not neglect this very valuable resource.

(Note, make sure I use the verb “blog” rather than “write blog entries.)

Twitter

“Now wait a second!”, I hear you saying, “Twitter is for chatting, how am I supposed to learn from a ‘social network’?”. To which I would respond, “If you aren’t learning from the people you follow on Twitter, you are following the wrong people.”

Many of the same people who are blogging such valuable information are also on Twitter. And they are not only blogging, but also tweeting about it, and tweeting about what others have blogged or new technologies, techniques or tools they have seen. If you plug yourself into this constant discussion about the practice of software development, there is much you can learn.

Many of the tweets contain links to information that are useful. These links are usually to blog entries or other articles. Many of the blogs that I follow were blogs that I saw someone mention on twitter.

Who should I follow?

Start with any blogs you have read. If the blogger is on twitter, follow them. Another way to find good people to follow is to start paying attention to the "re-tweets" you see. If a person is retweeted, it means someone thought their content was good enough to be repeated. If the content was good, follow the person who was retweeted. You can also search twitter for particular topics. See who is tweeting good content there and follow them. After a while you will have a good number of people you are following.

It can get overwhelming if you try to read everything ever tweeted. Don't fall into this trap. Don't feel like you have to read it all, you can't and you never will. Simply join the stream and read a bit and get off. If you feel like you have to start where you last left off, you will never catch up and you will stop reading.

Sometimes tools can help a lot to distill the massive number of tweets down to something you can digest. For instance, one tool I have used is called paper.li (link). It aggregates all the links that were tweeted by the people you are following and displays them in a newspaper-like format by category/topic. You can scan all the links each day and read only the things you are interested in.

There are also tools that let you queue up items to read later. Some examples of these of ReadItLater, Instapaper or Spool. (links). They allow you to queue up items to read, them read them later on any of your devices (desktop, mobile, tablet, etc).

Twitter is a great way to expand your learning. It is also a great way to learn of events going on in the developer community that you may want to take advantage of.

Attend Community Groups and Events

Every major city (and many not-so-major cities) have user groups around many different technologies. I live in Austin, and there user groups and such for almost every kind of technology imaginable. There are multiple events every week. I could never attend them all. The ones I do attend have really broadened my horizons…not only in what I have learned there, but the people I have met there. Forging relationships with others in the developer community is a great way to learn. The most common means of learning is simply asking questions. If you find people that are experienced in the area you are learning, there is no better resource than that. Having a mentor or friend that you can bounce ideas off of and ask questions from is invaluable.

Another great place to learn is at any kind of brown-bag/lunch-n-learn groups you might have at work or in your area. Some groups are book clubs that read through a book and each week get together and discuss the book. Others have developers present some interesting technology or tools they have been using. If you don't have anything like this in your company, or in the area, start one! I started weekly "Dev Shares" and a "Continuous Learning" group at my company. They take little work to keep them going, and everyone benefits.

Blog and Present

It has been said that the best way to truly learn something is to teach it! I have found this to be absolutely true. When I prepare a presentation, I have to research the topic well and ensure not only that I understand the topic, but also that I am able to explain it clearly to others. This requires a you to understand the topic far more than if you just wanted to know it for yourself.

A great way to deepen your knowledge and increase your ability to communicate (which we will discuss later) is to start blogging. Pick topics that you are learning and begin blogging about what you are learning. It is very rewarding and you will not only help yourself learn, but others. There are several times that someone has had a problem and I have been able to direct them to a blog entry I wrote that clearly explains the problem and how to solve it.

Presenting on a topic takes it to the next level. Not only do you need to be able to explain the topic, but you are also opening up the possibility of others asking your questions, that you may not have rehearsed the answer for, or worse yet, you may not know the answer to. Always remember that saying "I don't know" is perfectly acceptable. Pretending you know something when you don't is not only dishonest, its a recipe for disaster.

So as you learn, reinforce your learning by teaching others what you have learned. You will benefit, and so will those you teach.

Attend your local user group(s) and learn from those around you.

Humility

Humility is one of the most underrated, yet essential attributes of an effective developer. It's easy to be misled into thinking that technical skills are the most important thing to look for when hiring a developer. It's a lesson I have learned on more than one occasion…I have hired people that were very knowledgable and technically skilled, but then I had to work with them. It was not a pleasant experience.

> The effective developer is humble.

Humility is the opposite of pride. A prideful person thinks they already know everything. The humble person listens to others because he realizes that they may have an idea that is better than his own. A prideful person is argumentative and doesn't take suggestions or criticism well. The humble person considers the position of others in design discussions and rarely gets into arguments. He also views criticism as constructive and an opportunity to improve himself. A prideful person seeks to place blame when something goes wrong and refuses to admit when they have made a mistake and accept responsibility for it. The humble person seeks the solution to a problem rather than where to place blame, and easily accepts responsibility for they mistakes they have made and openly admits to them.

Pride also stifles the ability to learn. Consider this analogy. A sick person cannot be helped by a doctor if they don't think they are sick. They will either never go to the doctor, or won't follow his advice. Likewise, the biggest barrier to learning the truth is the assumption that you already know it.

One of the primary traits of pride is self-preservation. That is the downfall of many developers–they seek to promote and preserve themselves at all costs. That is why it is so hard for them to admit fault–it is seen as a mark against them, and they don't want anyone to think negatively of them. In their minds, that's not the way to "get ahead".

A great definition of humility is "placing the needs of others before my own."...it is "other-centeredness" as opposed to self-centeredness. The humble person places the interests of the business and the needs of others before their own. This is why a humble person can easily admit when they have made a mistake. Doing so is the quickest way to solve the problem at hand rather than wasting time playing the blame game. Not only this, but a humble person is quick to point out their own mistakes to others so they they won't make the same mistakes. Not only can you learn from your mistakes, but so can others.

In design discussions, it is vital that all members of the team are free to discuss and debate alternatives without getting into an argument. It is equally important that all team members are able to speak freely. When an argumentative person is placed in the team, they can turn these design discussions into arguments, or they can dominate the conversations and not let others speak up. Either way, it stifles productivity–team members stop contributing their ideas, either for fear of rejection, or being drug into a lengthy conversation that they cannot escape from.

The damage this kind of personality inflicts on the team can easily go unnoticed. Unless you know what to look for, you may not realize why things are going so poorly on the team. I have been amazed (on more than one occasion) how productive our team became when one key person was removed from the team. Things just seemed to smooth out after their exit.

Don't be the bad egg that poisons the team. Make sure you are sufficiently humble. Be a blessing to your team, not a curse.

Communication Skills

Another important, yet often overlooked, trait of a great developer is communication skills. If you cannot clearly communicate your ideas, how can you expect them to be accepted? The importance of clear communication amongst the team in a software development project cannot be emphasized enough. Just think of the kinds of issues caused by miscommunication: Bugs are introduced due to misunderstood requirements. Arguments ensue because developers misunderstand one another. Rework must be done because standards or policies were not clearly communicated....and the list can go on and on...

Poor communication can cause problems that you don't even realize are a result of a communication failure. When I have a design disagreement with a developer that I know is intelligent and has the

same project goals in mind as I do, I immediately assume there is a misunderstanding somewhere and work with the other developer to verify the assumptions we are basing our positions on. The majority of the time we find a point or two where we misunderstood one another. Once we correct those misunderstandings, an obvious solution that we both agree upon becomes apparent. Had I not sought out the misunderstanding, we would not have been able to agree and it could have damaged our relationship.

Someone has said (reference?) "Communication is something the listener does" (verify quote with Manager Tools). You can speak about some concept to another person, but if they don't understand what you said, no communication has occurred. Communication is the act of transferring a concept from your understanding to their understanding. In the end, if they understand the same thing you do, communication has occurred (should I delete this sentence minus the "In the end," part?). If they do not understand the concept, communication has not occurred. If what they understand is different from what you understand, miscommunication has occurred. Miscommunication is far more dangerous than a lack of communication because it goes undetected. The listener/reader thinks they understand correctly and may take action based on that misinformation with unknown repercussions on the project. If no communication occurred, the listener at least knows that they did not understand and won't act on false information, but may ask more questions to learn (reword this).

This book is written assuming you are a developer. If this is true, then you probably spend more time talking to a computer than to people. While we might wish that people are as easy to talk to as computers–just say what you mean and be done–they aren't. People require a lot more care and attention than computers. You often need to consider things outside the context of your communication when dealing with people. (example of how you may want to communicate differently based on external contexts).

Types of communication.

Face to Face

Highest bandwidth communication (quick feedback with body language, + ability to question) This is why colocation is a big deal in agile. Mention reading people….expressions and body language, etc. Face 2 Face is two way communication even when only one person is talking

Voice communication (phone, etc)

Chat

Presentations

Spoken

Written (Email, SMS)

So, how can you learn to be a better communicator?

In Written Text (Email)

In Design Discussions

In Presentations

Chapter 2 : Foundations

This chapter introduces some very fundamental concepts for those who may not be familiar with them. If you are familiar with these concepts, please feel free to skip this chapter.

Object Oriented Programming

Encapsulation

Encapsulation comes from the verb *encapsulate*, which means "to enclose or be enclosed in or as if in a capsule" [1]. This properly describes the concept of encapsulation in object oriented programming. The principle of encapsulation motivates us to keep together data and the logic operates on it.

By consolidating (I mean, encapsulating) the logic and data, we can hide some of the data or logic from the outside world since it is only needed by the logic inside the encapsulation. By hiding these details, the system as a whole becomes easier to understand and maintain. This is why encapsulation is sometimes referred to as "information hiding".

In OO programming, you encapsulate logic and data into objects. An object consists of data (properties/fields/etc) and logic (functions/methods). Some of the properties and fields are public, meaning they can be seen and accessed from code outside the object. However, some are private–only accessible by code running inside the object.

The principle of encapsulation suggests that we make private any methods or data that are only used within the object. Only make public anything that needs to be accessible by logic outside the object.

Let's look at an example: <example here>

(This paragraph probably doesn't belong here) Wikipedia defines information hiding as "the hiding of design decisions in a computer program that are most likely to change" [2]. This definition gives some insight into good object design. Hide things that are likely to change often. We'll dive into why this is a good idea in a later chapter.

Polymorphism

Polymorphism refers to something taking many forms. In object oriented programming, it refers to one type of object that can take many forms (subtypes). Replace an object of a class with an object of its subclasses Abstraction Code against an interface, not implementation Depend on abstractions, not concretions Class is not know at compile time, but behaves at run time based on that actual object's class.

[1] Collins English Dictionary - Complete & Unabridged 10th Edition; 2009 © William Collins Sons & Co. Ltd. 1979, 1986 © HarperCollins Publishers 1998, 2000, 2003, 2005, 2006, 2007, 2009

[2] http://en.wikipedia.org/wiki/Information_hiding

Low Coupling / High Cohesion

Low Coupling Unit tests promote low coupling. Highly coupled code is difficult to test. Links Coupling And Cohesion on c2.com http://c2.com/cgi/wiki?CouplingAndCohesion "The degree to which each program module relies on each one of the other modules" – Wikipedia http://en.wikipedia.org/wiki/Coupling(*computer_science) High Cohesion http://codebetter.com/blogs/jeremy.miller/pages/129542.aspx "A measure of how strongly-related and focused the various responsibilities of a software module are" - Wikipedia http://en.wikipedia.org/wiki/Cohesion*(computer_science)

UML

Class Diagrams

Activity Diagrams

Sequence Diagrams

Chapter 3: Design Principles

KISS: *Keep It Super Simple*

YAGNI: *You Aren't Gonna Need It*

Last Responsible Moment

Avoid Premature Optimization

DRY: *Don't Repeat Yourself*

Separation of Concerns

Composition over Inheritance

Explicit over Implicit

Fail Fast

Law of Demeter

No Broken Windows

Boy Scout Principle

The SOLID Principles

SRP: *Single Responsibility Principle*

OCP: *Open Closed Principle*

LSP: *Liskov Substitution Principle*

ISP: *Interface Segregation Principle*

DIP: *Dependency Inversion Principle*