What I learned, notes - Week 2 Reset Phase

The Myth of the Genius Programmer

Everyone wants to look like a smart developer and hate to be seen when they fail. Find that guy at your group that if he is gone, he will leave you in hell. Even if you are a genius working in a team will be more successful. In order to not be a bad programmer or this type of "myth" here are some tips:

- Drop your ego, don't be territorial on your code
- Criticism is not evil, do code review
- Embrace failure and document about it
- Fail quickly, iterate as fast you can as this will make you learn faster
- Be a small fish, surround yourself with pros
- Be influenced, listen to people and they will listen to you

When working in a project don't categorize codes or start ruling who can or cannot do something somewhere, as you never know what everybody can add to any problem. Anyone should be able to report a bug.

Collaboration is important, but when you should do it?

- If you are done it is too late, if it is too long you won't be able to review it entirely, if there is no opportunity for someone to make their mark it won't be worth it for them.
- It is too early when there is nothing to look at or there is no concrete idea of the project.
- Sweet spot! Write goals and not goals, you need some running code but just to prove a concept or show something.

If you fail at something don't just erase it, save it as in the future it can help you or someone, or maybe someone can take over it.

When you get it started people will be more interested into joining your project.

Don't hide your projects, codes, ideas to people as it is just selfish, and you don't get feedback.

Don't try to be a genius. Collaborate early and often. Pay attention to your tools and to timing.

#perfmatters

What you can measure you can improve. Gather -> insight -> action. Validate everything you are seeing, so you can optimize the performance of your website, now there are multiple tools you can use for this.

Html5rocks

Variable Length Codes (Ep 1, Compressor Head)

Depending on the quantity of symbols per code and their repetition in it, you can start giving the percentage of probability of appearance. The symbol that has more probability of appearing you will give a less size bits representation than the one that is les likely to appear. This way you can reduce the size of a code.

Entropy represents the best estimate of the minimum number of bits required to represent the stream. Or the minimum number of bits needed per symbol on average to represent your data stream.

Principle: Assign shortest codes to most frequent symbols

Because of this sometimes it is difficult to determine how you read a stream as symbols have variable lengths, for this you use a:

- Prefix property: once a code has been aseigned to a symbol. No other code can start w/ that pattern. An example of this is the Unary Code

You can create your own variable length code (VLC), with the help of a Huffman Tree. For this you need symbols and their probability and create a binary tree.

The LZ77 Compression Family (Ep 2, Compressor Head)

Entropy is a theory not a law. Because you could get an inconstancy if in your data stream has the "same" probability i.e. 1,2,3,4,5,6,7. But there is a catch.

You shouldn't look at symbols individually, but you should look at them in groups or pair, this will change entropy once again. You should group dynamically with longest and smaller groups. An algorithm to find this groups is called LZ77 or LZ78, these use Search Buffer (symbols we have seen before) and Look Ahead Buffer (symbols we haven't seen before).

A compressing program would be Winrar which uses LZSS that is a variant of LZ77,

Compression (Ep 3, Compressor Head)

Markov chain work is the one that is reining right now. Encodes a string by looking at a letter and noting the transition to the following letter. ZPack, 7-zip and Lpack uses this compressing algorithm. You can reduce transfer sizes for different world places.

The Art of Organizational Manipulation

Companies are made out of people not code, and they are who prevent things from happening. For this you need to know how to navigate them, social engineering. Some good leadership is when you ask what people need for them to work better or make it done. pursue responsibility look for improvement, don't wait for them to tell you, try to grow, question things, don't put boundaries to people, take risks and be comfortable to fail, trust people on their work. Don't destroy, replace.

Let's try this…. This are words you can use for invoking ideas in the company, is better to say sorry than ask for permission, but do it for the right and worthy things. Do favors and never burn bridges.

You can manufacture luck for yourself. Make influential friends as they are influencers.

Learn how to ask busy people: Make three bullet point and a single request.

"Every day I go to work and do what I think is the right thing and then sit back and wait to get fired, and if I don't, I am in the right company.

Programming Well with Others: Social Skills for Geeks

Social stuff is hard but is necessary. Document things as this way you will shut discussion making it "official". Motivate people in their own way. Express yourself, don't be just a yes man. Sometimes addressing people is not the way, you have to address their behavior. Ego is good but when there is a team ego not individual.

Developing Expertise: Hearding Racehorses, Racing Sheep.

Process improvement requires people improvement. As software is written by people not tools. Tools have improved, but bugs have stayed the same, because the people hasn't got better. Dreyfus Model.

If you are not a master, Dreyfus Model:

1. Novice: you just want to know how to do it. You don't have knowledge about the topic, history context, etc.
2. Advance beginner: don't have experience, you try things on your own.
3. Competent: Seek out experts, develop conceptual models, you solve problems in your own.
4. Proficient: You understand and apply, self-correction, you see the whole picture, oversimplified information buggs you, learn from experience of others.
5. Expert: You follow rules, work from intuition, continuously look for better methods, you are primary sources of information.

Never allow an expert to choose how to do your project, because they are experimenting on you. Don't try to race sheeps, don't try to herd racehorses. Don't treat people equally as everyone is different. Don't put people of certain level on tasks they are too pro or too noob to do them.

Show yourself you are a novice, don't pretend proficiency.

For financial good tips or anything in life:

Have a plan

Diversify:

- Don't put all your eggs in one technology basket (languages, techniques, industries nontechnical)
- Mix high risk high return, low risk low return

Look for value: how can I do my job better with this.

Review: keep evaluating yourself, maybe things have change.

Invest regularly: invest a minimum time each month (2 hours a month), have a ritual, plan time in advance.

Measure progress, for your team to improve you have to improve first as well.

Perfection is an unrealistic goal

Sometimes managers pressure people just to finish things, but this will cause things to turn out not right. We are all hardwired to lie, because we will never get estimations right (estimate time to do a project). We are told to do agile, patterns, cycles, iterations all the time, and in fact we live in cycles. Agile doesn't need to be working longer periods of time without breaks or interruptions, we should be agile not do agile. Sometimes it is just an illusion of productivity the agile mode.

Everyone works differently, take a break, don't burn yourself out, don't try to change the whole world at once, do little things, do something different, you never know if you can take a break, how do you know if you take a break you won't be more productive.

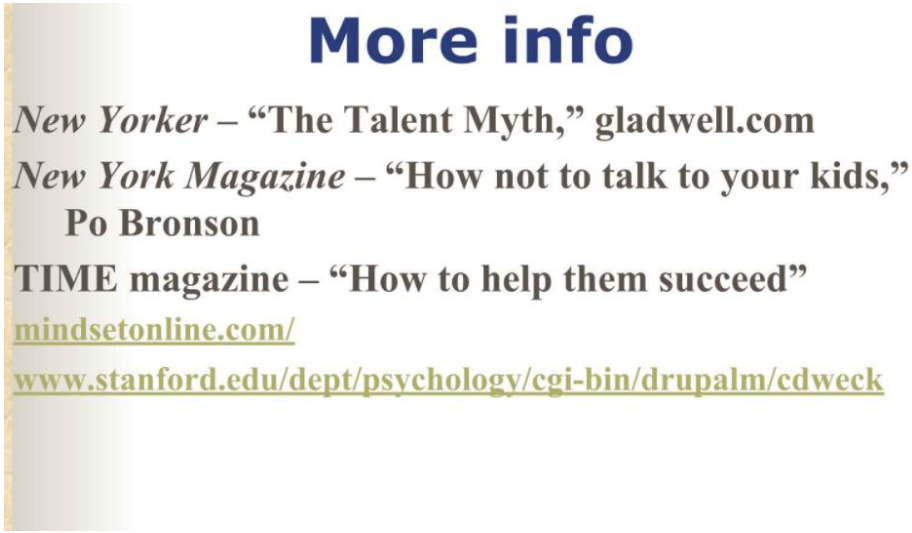
The Agile mindset

If you put your mind into it you can do it is a matter of mindset, regardless your age, situation, area, etc. Is it the best you can do, can you work harder? Intelligence is something you work for, you can be anyone you want, you don't have labels, don't have a fixed mindset. An agile mindset: you can always get better, grow, learn over time, and you challenge yourself constantly, as well don't say it is too hard or you can't do it. If you don't know something ask about it, of course sometimes things will be hard. So learn, fail and learn, grow, embrace challenge, be resilient. Don't keep doing the same things you always have done, change things up, try new things. Believe in yourself. Nobody is perfect.

If you keep telling kids they are great, so bright, etc. they will have fixed mindsets, as they will try to stay where they are comfortable, and people keep telling them they are smart. If you tell them hard stuff, they will get used to it, and when something hard comes they will be used to that kind of stuff.

No one is born knowing everything, It requires hard work. Praise effort, strategies, process and don't label kids. Failure fast, learn often, fail is an option.

Everything is a work of process, even how you work



Linda Rising on Collaboration, Bonobos and The Brain

Most of the time people don't want to know about new ideas or techniques. When you can have a goodnight sleep it's when you have gathered enough data about something, you can sleep, and your unconscious will help you make a decision later. We work better in small groups.

While working everyone needs to communicate and work with everyone and share (agile method), this type of environment will make people improve and have a better attitude, because they will enjoy it, as they will share something as well.

DAY 18: Queues and Stacks

Queue:

- First in, first out
- You can check the size, who is first and last

Stacks

- Last in, first out

Prejudices can alter teamwork

We stereotype people unconsciously son from that point you favor that person or not. Memory isn't accurate so you shouldn't rely on it.

Sometimes being part of a group you develop stereotypes of another and start judging them without even knowing them, but a shared goal, something they cared abotu can bring everyone together.

In Agile we become aware of everyone else, how can we help them, what can we do, because our success is tied with his.

Two types of monkeys example, one aggressive and other pacific, when one tried to attack the other one ignored them, and over time the aggressive ones started to be less aggressive and share more, after this they stayed the same.

A mangers rule should be "I want to catch them doing something good". My initial thought of someone shouldn't change what I expect of them, as this will not only change your behavior but also theirs.

Everything is a remix

We copy, transform, and combine. We remix everything now these days. Creation requires influence, we stan in the shoulders of giants. We learn through copying, there can't be other way because we need a foundation of knowledge and understanding, then que can transform to create variations, but the most important breakthroughs come from combining ideas.

The basics of creativity are copy, transform, and combine.

We are building with the same materials, sometimes by coincidence we create the same things, but as well innovation sometimes is inevitable. New ideas evolve from older ones.

When we copy, we justify it, but when we get copied, we get angry. Software patents tend to be abstract and written in very vague language

Shell tools and scripting

Foo -> is for creating variables, writing variables in string you use "hola soy $foo"

You can create for loops, create conditionals, or change files types (jpg -> png), crop videos, extract a frame from a video.

Create multiple files (touch), difference between folders. Sys.argv in python can use shell commands

Locate and find files, import them, show history

Classes

Community Building

Operations in a day

Hay que tener actitud de dueño, no de empleado. No solamente se quejen, pero resuelvan el problema. Levanten la mano cuando exista un problema, no solo hagas tu parte y te preocupes por solo tus problemas. Si te toca ser líder adáptate.

Intro to Labs

Donde se realizan productos innovadores. The objective is to create solutions that are unique, pain reliever, software related and doable (short term).

Safe spaces

It is a safe space to fail and share ideas. The same way the company gives you freedom it algo gives you more responsibilities. Diversity is a main thing in the company, as you can be whoever you want, share yourself. Don't assume everyone's knowledge.

Empathy:

- Perspective taking, staying out of judgement
- Recognizing emotion in other people, feeling with people. It is a vulnerable choice as you have to show something.
- It is okay to don't know what to say them, is already good you are listening.

Exercise

Una vez un amigo se emocionó por que gano en algo, y pues yo ya lo había echo antes y me mostré indiferente ante ello y como que mate su emoción y pues ya no dijo nada más, yo creo que debí mínimo compartir su emoción o haber dicho como "muy bien por el ya por fin pudo lograrlo".

Lectures

Test Driven Development

Is a software engineer technique, that combines two other ones: Test First Development (testing before coding) and Refactor

Objectives

- Reduce # of bugs
- Implement what the client asks for and no more,
- Create modular software, reusable and easy to adapt to change

This will help to keep yourself within the time you agreed with your client and not over-code as most of the time is time invested in functionalities that the client will never use.

Why doing this?

Guides the design and states the problem and the solution.

TDD is designing

First:

Focus on the problem and explore it. Write example cases, think about the what before how?

TDD is not about testing but more about designing: what I mean is that testing is just a secondary outcome, because what you really focus on, is to first think and write how is your program going to function, then you continue to code it. This helps us, because sometimes we just launch into coding and we start creating functions, methods and classes, before we even know how we are going to use them, or we don't even know if they are going to be useful at all. So TDD is a nice approach to avoid this.