Metacognition for Software Engineers

Introduction

This material is trying to shed some light on the importance of metacognition for the life of software engineers, and how metacognitive techniques can help make engineers perform at their best. It also tries to come up with some metacognitive techniques for software engineers to use in their different daily activities.
It worth mentioning that the techniques devised here is not a one-size fits all. This is mainly because we are different, our cognitive abilities are different and our cognitive limitations are different, and one technique will not work for all cases. This is an attempt to raise awareness of some of the techniques that can work in many cases, and maybe point the reader to the direction to find the technique that works best for the reader's cognitive abilities.
This content has been developed as the project for Educational Technology course in Georgia Institute of Technology, Fall 2017 class. This class lives on the web on http://www.metacognitive.life/

Introduction to metacognition

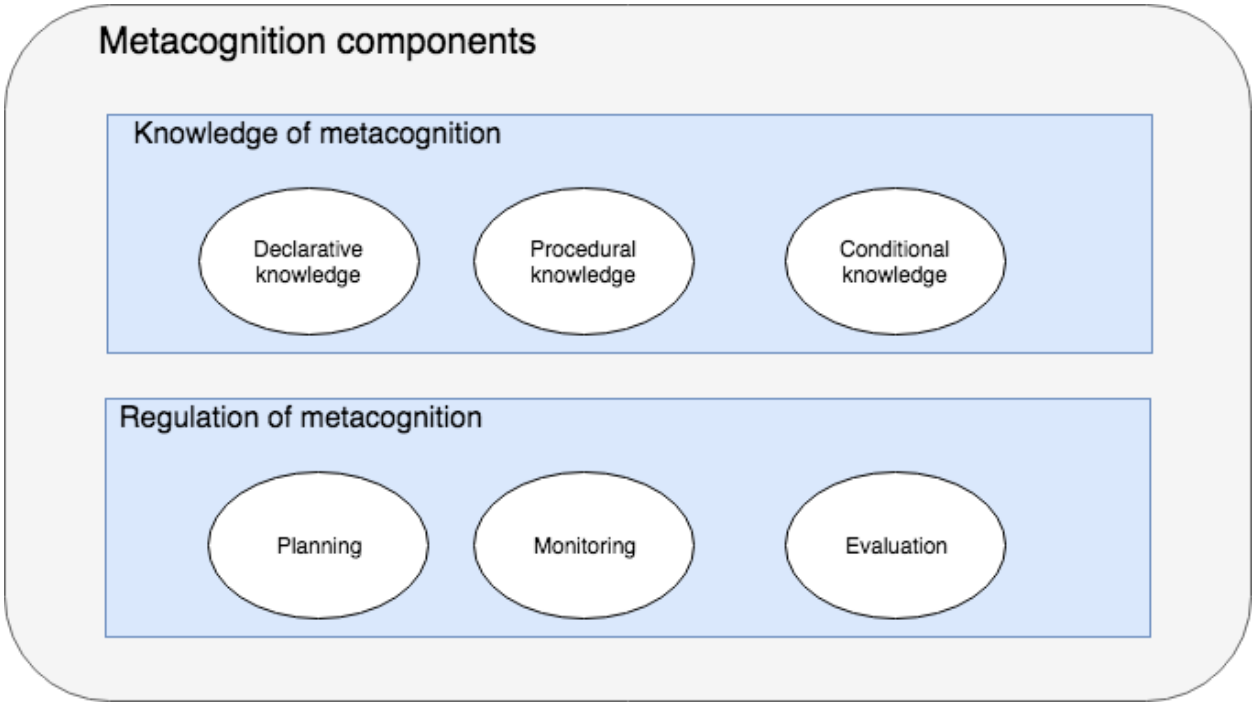
What is metacognition?

The simplest definition of Metacognition is that it is “thinking about thinking”. It is the knowledge you have of your own cognitive processes (your thinking) and It is your ability to control your cognitive processes using various techniques, such as organizing, monitoring, and adapting. Additionally, it is the ability to reflect upon the tasks or processes you perform and to select and utilize the appropriate techniques necessary in your intercultural interactions. So if a software engineer's cognitive skills are the skills that help him perform a task (code, test, etc), metacognitive skills are those that help the engineer monitor and regulative his cognitive performance.

Metacognition is considered a key component for successful learning. It involves self-regulation and self-reflection of strengths, weaknesses, and the types of strategies you create. It is a necessary foundation in culturally intelligent leadership because it underlines how you think through a problem or situation and the strategies you create to address the situation or problem.

Components of metacognition

Most theories of metacognition divide it into two major components, they are:



Knowledge of metacognition

This tries to answer the question: What do I know about my own cognitive system? How does it work? What helps it perform better and what makes it perform worse? Generally the knowledge of metacognition is split into three sub components:

1. Declarative knowledge: Knowing myself as a learner and knowing my limits, for example many engineers knows the limits of their own memory system, how much they can remember and how much will slip away.
2. Procedural knowledge: Knowing about the procedures and strategies that help me overcome my shortcomings. These shortcomings that I probably knew from the declarative knowledge of my cognitive abilities, like for example using note taking with a pen and paper if I know that my memory does not recall important points.
3. Conditional knowledge: Knowing which procedure or which strategy works when, and selecting the appropriate strategy for each situation to achieve the best results.

Regulation of metacognition

So now I know about my own cognition and how it behaves from the knowledge of metacognition component, it is time to try to use this knowledge to improve my performance and my cognitive techniques, this usually happens on three major activities:

1. Planning: The process of selecting the appropriate strategies and allocating the resources needed to execute these strategies. It also involves setting goals and deciding what you want to achieve in this phase.
2. Monitoring: After you have your plan set, you start executing it, and during execution you monitor how the selected strategies affected your cognitive performance.
3. Evaluation: After executing the plan and collecting observations from monitoring, you start reflecting on the collected data, and decide which processes and strategies was a good match to the situation, and which was not.

Why is metacognitive learning important for Software Engineers?

The Software Industry is changing very frequently, new technologies emerge everyday to replace older technologies and making them obsolete, which also causes software engineers' skills to become obsolete. Dr. Ernest T. Smerdon the previous head of National Academy of Engineering thinks that the half-life of software engineers skills --how long it takes for half of an engineer's skills to become obsolete-- is 2.5 years [1], the lowest in the Engineering industry in general. This shows the necessity for software engineers to become lifelong learners if they want to stay marketable and in demand.

The frequent learning activities by software engineers can really use the help of metacognition and self regulated learning strategies. As we seen metacognitive knowledge is a key attribute that defines an expert.

Defining expertise

If you examine job ads asking for software developers, they are always searching for an expert in technology X or framework Y, and while the technology itself might not be older than a year or two, but the ads are almost always looking for experts, and this is the common expectations for software engineers. It is not just to learn the basics, but to develop deep expertise in emerging technologies, but what is expertise? How can it be defined?

There has been multiple attempts at defining expertise, the naive definition would be that an expert is someone who has many years of experience in a topic. This definition is rather weak because the years of experience in a topic or a task has no direct relation with the person's proficiency in this topic.

Another definition possibly relates more to software engineers is that a person can be called an expert if he has:

1. Declarative knowledge: They know the theory and can explain it in clear terms.

2. Procedural knowledge: They know the how to do it and they can do it.

Let's give a concrete example on this definition, an engineer can be called an expert of Object-oriented programming (OOP) if he posses the declarative knowledge of it (encapsulation, inheritance, polymorphism, etc), and he posses the procedural knowledge that allows him to implement in a programming language (can create a class, extend it, etc)

One final definition published recently by Gobet F. [3] is that an expert in a specific domain is "somebody who obtains results that are vastly superior to those obtained by the majority of the population."

Measuring effectiveness of software engineer's learning

So in the route to become expert software engineers, how can the effectiveness of software engineer's learning can be measured? There are multiple models to measure the effectiveness of learning activities, maybe the most adopted one is the Kirkpatrick model. Donald Kirkpatrick came up with this model in his PhD dissertation, and later developed it further to become widely used in learning activities. The model employs 4 activities:

Reaction: This measures your initial reaction to the training, how do you feel about it? Do you feel you learned something new or was it not worth the time. For example, where you happy taking this Java 8 course? Or was it boring and did not add much to you?

Learning: This activity tries to measure how much did you learn from the training, this ideally can be done by giving yourself a quiz on the subject before and after the learning and see how did the learning affect your knowledge. For example, try to take a quiz in Java 8 before you take the class, and then take it again after and see if the grades are different.

Behavior: This activity measures how did your behavior change because of the learning, are you more productive, producing more quality work or did not it make a difference? For example, did this Java 8 course make your code better and makes use of all of the new features of Java 8? Or are you still writing the same old way since the Java 1.4 days?

Results: At this activity the final outcome of the training is analyzed, how did this training impact business targets? Did this Java 8 course help you deliver the project to the business in less time? Or in the same time but with higher quality and reliability? This activity is usually the hardest to measure, as it needs clear KPIs that are frequently absent.

Metacognitive strategies toolkit

There are many metacognitive strategies or tools that has been developed throughout the years, these strategies can be effective in a multitude of situations, some of these strategies are:

Cognitive wrappers

The cognitive wrapper is a tool for teaching yourself to monitor your learning. The wrapper is a small activity that wraps the main learning activity (happens before and after it). It can be in the form of a couple of questions that you write for yourself and answer before the learning and after the learning. Thinking about these answers will force the learner to monitor the learning process and hence being able to evaluate it.
Cognitive wrappers usually have the below 4 activities:

- Rationale: This is a statement of purpose, in this statement you write down why are you doing the learning activity you are doing now. This helps your mind stay focused on the goal. This is written before the learning activity.
- Reflection: This an assessment of your state before the learning, how much did you know about the topic? How much did you struggle with the problem that this topic is going to solve for you? Did you do any research before doing this learning?
- Comparison: This is an assessment of your state after the learning, how much did you learn? Do you now possess the declarative and procedural knowledge of the topic? How did your state before the learning help you during the learning?
- Adjustment: Now that you finished the learning and finished the assessment, how would you do this differently the next time? What went right and what went wrong?

Think-aloud protocol

This strategy depends on the learner speaking their thoughts out loud throughout the learning experience, it is developed by Ericsson and Simon [6], and then the teacher can use this feedback to evaluate the thought process of the learner and potentially correct any possible flaws.
This approach has many drawbacks, it obviously can’t work in on self paced e-learning environments because of the lake of feedback, and without enough guidance, the learner will not be able to know which thoughts should be spoken out loud, and finally it is mostly not realistic to expect someone to speak everything going through their head while they are reading a course text or watching a course video.

Questioning

In this strategy the learner finishes the learning activity, and then tries to develop a set of questions on the topic learned, and then goes back to the material again to make sure that these questions were not answered already. This exercise will encourage the mind to evaluate the learning to come up with questions on the material, and will encourage the learner to find the potential gaps in his/her learning technique.

Reciprocal teaching

This strategy reinforces the learning by making the learner teach the topic to someone else. It is widely said that “teaching is the best way to learn”, and this sentence is backed up by some modern research. A study [8] done in 2009 on a group of school students showed that they improved significantly when they started to take turns with the teachers in teaching the topics.

Types of learnings for a software engineer

Overview of the typical learning types

Many learning activities happen in the daily life of a software engineer, some of these types are listed below

Academic learning

Software engineering is an engineering practice that is taught in colleges and universities all over the world. Academic learning can have many variations, it can happen in a campus or it can happen completely online. It can also have different levels, it can be for a bachelor's degree or it can be for a postgraduate degree (masters or PHDs).

Learning new technologies

This happens very frequently throughout the career of a software engineers, as new technologies emerge everyday and become suddenly the hot trend in the market. The demand for it becomes higher, and engineers are chasing the learning of these technologies to stay relevant and in demand.
Although some engineers might have perfected their learning using cognitive skills (without even knowing the scientific terms for it), some others often struggle when learning a new technology. Also it is confusing to select which learning medium will work best for you, should I enroll in a face to face classroom course? Or should I settle with an online self paced course? Or maybe just ignore all kind of learning courses and jump directly into the new technology’s manual to read from the source? That’s often a confusing debate.

Project handover

Another activity that happens frequently to software engineers is project handover. This happens when a new person joins a project, or when the project members are rotated to other projects, or maybe when a project is relocated from one place in the organization to another. Project handover is a critical learning activity that involves the learning of a multitude of code, design decisions, architectures and systems. If it does not happen right, it can cause outages the loss of business opportunities.

Using metacognition for academic learning

Cognitive wrappers

Cognitive wrappers can be a very effective tools in preparation for exams, as exams are something that happens very often in the academic learning, so reflection at the end of every exam and adjusting for the next exam can result in exponential improvement in the exam performance. Although cognitive wrappers are usually considered the responsibility of the teacher, but this does not prevent it from also being a tool for the student to improve and enhance the learning performance. Below is a sample cognitive wrapper that can be used in academic exams:

Rationale (Before exam):

- Why are you taking this course? What are you looking to learn
- Why am I doing this cognitive wrapper? How will it help me improve my learning performance?

Reflection (Before exam):

- How much time did I spend on my preparation for the exam?
- How was this time split across the different topics of the subject?
- How much time and effort did I spend creating my own examples and analogies from the subject material?
- How much time did I spend in any of these activities (Thinking, finding online content, preparing, researching, finding inspiration, analyzing body posture, resting, etc)
- Was I focused on doing one thing at a time, or was I multitasking?

Comparison (After exam and getting the exam feedback):

- What kind of mistakes did I make in the exam?
- Why did I make this mistake? Was it something I covered in my study but still did not grasp it well, or was it something I completely skipped? Or was it something was not in the material at all?
- Estimate the points you lost because of any of these reasons:
    - Anxiety

    - Not studying enough
    - Studying the wrong material
    - Syntax errors (arithmetic or grammar)

Adjustment (After exam):

- Did the studying strategy I employed work well?
- What are the top 5 things I would do differently the next time I take an exam?

Mental readiness

It is important that you prepare your mind to what is going to happen, for example you should know the test format beforehand: Do not just study the material you are examining in, study the test format itself, is it multiple choice, is it open book or closed book, etc. A study on 56 students has shown that students who knew about the test format beforehand has performed better in the exam [4]

Using metacognition for learning new technologies

How can metacognition assist when a software engineer is learning a new technology (A new framework, a new programming language, etc)? The most widely used technique is the hands on approach, that is to actually use the technology and create a “Hello world” application, and while this approach is effective in evaluating and practicing the procedural knowledge of the technology, It does not directly help with the evaluating the declarative knowledge on the technology. Below are a couple of strategies that can help with this:

Questioning strategy

After finishing new technology training, try to come up with a set of 10 questions about the technology. Then go back to the source material and see if the material already covers these questions. Then for the questions covered already, try to think about the reasons why you missed the answers in the source material. This will help the mind to rethink the learning strategies in similar situations, and improve on the next iteration.

Reciprocal teaching

One of the great ways to reflect and reinforce your own learning is to teach it to someone else. For example you have been lucky to be sent to that fancy conference where you learned lots of modern architecture techniques, one of the best ways to make sure you learned the topic is to organize knowledge transfer session to the less lucky people who were not there. And then teach the same material to them, this will be beneficial for multiple reasons

1. You will need to build material that suits the audience, which will make you look at the source material and tweak it.
2. You will have to express the material in your own language and your own words, this will help your mind absorb and deeply understand the topic.
3. You will receive questions from the audience that probably have not crossed your mind, these questions will make your mind exercise to try to come with the answer on the spot, or maybe it will be too hard that it will send you back to the learner shoes to find the answers. It will be like you are using the questioning strategy but collectively from multiple people in the audience.

Using metacognition to onboard a new project (project handover)

It is quite often for software engineers to join an existing project and try to learn the project and its existing design and code base, how can metacognition assist with this learning activity?

Cognitive wrappers

Design a cognitive wrapper to force your mind to monitor and evaluate the effectiveness of the handover process. For example, the below cognitive wrapper can be used for handovers of software project that involves both code handover and environment configuration handover (deployments and such). This wrapper is based on the research done by Marcha C. Lovett [3]

Rationale (Before handover):

- Why is this project important? What is the value for the business?
- Why am I taking this hand over? Just getting an idea on the project or am I going to work on it full time?

Reflection (Before handover):

- What do I already know about the business of this project?
- What do I already know about the technologies used in this project?
- What do I already know about the software engineering process implemented in this project?

Comparison (After handover):

- How did my knowledge of the business of the project differ now?
- How did my background on the business of the project help me during the handover?
- Do I know what I need to do to do a change in the project's code base and to launch it live? Do I know the whole process (whatever the process is, peer review, testing, UATs, this differs from one project to another)

Adjustment (After handover):

- Going into the next handover, what would I change in my background before the handover?
- Did the communication medium (face to face, online meetings, etc) help during the learning activity? Would I change it later for something better?

Summary

This course is an attempt to shed more light on the topic of metacognition among the software engineering community, to show that it can help with the lifelong journey that typical software engineers have. This course tried to produce easy practical steps that can be used in different learning situations to enforce and use metacognition. But because our cognitive abilities are different and our cognitive limitations are different, it is not going to be a one size fits all guide, but merely an attempt to increase the awareness of the topic, and possibly point the reader to the right direction.

References

[1] ERNEST T. SMERDON - Lifelong Learning for Engineers: Riding the Whirlwind
https://www.nae.edu/19582/Bridge/LearningforEngineers/LifelongLearningforEngineersRidingtheWhirlwind.aspx
[2] A. Brown, Jacobs & Paris - Metacognitive theories
http://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1040&context=edpsychpapers
[3] Gobet, F. (2015). Understanding expertise: A multidisciplinary approach. London: Palgrave.
[4] Stephan Dutke, Jonathan Barenberg and Claudia Leopold : Learning from text: knowing the test format enhanced metacognitive monitoring
https://link.springer.com/article/10.1007%2Fs11409-010-9057-1
[5] Marsha C. Lovett, (2013), “Make exams worth more than grades: Using exam wrappers to promote metacognition” www.learningwrappers.org
[6] Ericsson and Simon (1984) - Protocol Analysis and Expert Thought: Concurrent Verbalizations of Thinking during Experts’ Performance on Representative Tasks
http://www.ida.liu.se/~nilda08/Anders_Ericsson/Ericsson_protocol.pdf
[7] Deborah Cotton and Karen - Reflecting on the think-aloud method for evaluating e-learning
https://goo.gl/wd9gLf
[8] Aannemarie Sullivan Palinscar & Ann L. Brown - Reciprocal Teaching of Comprehension-Fostering and Comprehension-Monitoring Activities
http://www.tandfonline.com/doi/abs/10.1207/s1532690xci0102_1