Coaching for Software Developers

Enhance your soft skills and performance as a programmer

MC Sport+

Abstract

As software development becomes more intricate and demanding, also owing to the emergence of new generative AI, soft skills are increasingly regarded as an integral component of a programmer's toolkit for enhancing productivity and well-being in the workplace. In this book, you will be guided through a Mental Training program with the objective of refining your soft skills and improving your performance as a programmer.

Contents

Coaching for Software Developers
Enhance your soft skills and performance as a programmer
v1.0.0

Introduction

Welcome reader, this book is a coaching course for Software Developers. Going down the content you will be guided to improve your soft skills in a Mental Training program and level-up as a programmer. You will discover what Mental Training is, how it is tailored to the developer, what are the soft skills involved in work performance and you will try hands on a selection of the training techniques utilized in a live course.

About the book

The book is divided into 4 sections. The introduction, the chapter you are reading, covers the structure, topics, and other information useful to navigate the contents. The next section contains some theory about mental skills, explaining what they are and how they influence performance of developers. Following the theory we will focus on practice. Here you will be provided with some techniques and tools for analysis and training of mental skills. In the last module we will recap the content, and since as developers we are used to learning and continuous improvement, we will provide some guidelines and references to further delve into the book topics.

It is important to observe that a real coaching requires tight interaction. The ultimate purpose of coaching includes getting to a goal which is well defined and significant to the coachee, but this entails a relationship to be established with the coach preferably in live interaction. In the context of a book it is not possible to establish such a relationship but it is still viable to get results to some extent, although with the given constraints.

Who is the Coach

My name is Massimiliano Cianci, I am a programmer and licensed Psychologist. I have been working in software engineering since 1996 starting my career in Hewlett-Packard Italy. I later joined Ericsson Telecommunications in which I have worked for 15 years in the technical department. Since 2014 I develop applications for Android and iOS devices working as a freelancer for several companies.

At the same time I carry on some personal projects yet in the ICT industry and the activity of sport psychologist, in which I deal with coaching of sport coaches and mental preparation of athletes. In this role I have worked with the Italian Football Federation and the Italian Tennis Federation, as well as with a number of sport schools and centres.

The coaching course in this book bears from my academic study and from many years of experiences as software developer and sports psychologist. This is where my knowledge and know-how as both programmer and coach come together. If you are a software developer and want to improve performance and also wellbeing at work, you will hardly find content more specific than this.

I will come along with you in this journey, which I hope you will find interesting and productive.

So I welcome you again and let's get started!

Who is the Coachee

This coaching is aimed at experienced software developers. Specifically, it is best suited for mid-level and senior developers. It is also important to note that the main daily work carried out by these professionals is in teams, whether in person or remotely. This is because among the mental skills we will examine, some relate to teamwork, which is the most common workmode in software development.

Although still possible for junior devs to be a coachee they will only get limited results due to the fact that beginners are not yet fully involved in all the typical work situations and therefore cannot apply the information to their daily work, which is crucial for an effective learning.

Learning Objectives

The overall goal of this coaching is to improve developer's performance by means of boosting soft skills in a Mental Training program. The breakdown of such goal yields 5 learning objectives.

1. Know what is Mental Training for devs
As a first objective, we will understand the meaning of Mental Training for developers. We will see how Mental Training originated in the sports field and is applied to the work environment of software development. Clearly, the application to a different field requires adaptation of various aspects such as structure, content, tools for analysis and intervention methods.

2. Discover mental skills
The second learning objective concerns the mental skills involved in job performance. We will explore these skills and the theoretical framework that underlies them.

3. Observe mental skills in action
Then, we will examine the practical implications, that is, the impact

of these skills on the developer's work activity.

4. Explore training tools and techniques
For each skill, we will look at the intervention tools utilized for improvement.

5. Design you improvement plan
In the final objective of the course, we will complete a practical activity by engaging with some of the training techniques. We will choose this technique after evaluating the level of our mental skills. This assessment will be done using a diagnostic tool designed specifically for developers.

Mental Training for Developers

Let's now step into the heart of the book and see what Mental Training is. Mental training is a practice that aims at developing cognitive, emotional, and behavioral abilities in individuals. It is a widely used form of training in sports, and in short, we can say that it is the mental aspect of sports training.

Figure 1: 4 pillars in sport training

Looking at the pillars of sport training, historically we find athletic, technical and tactical training. Later the number of pillars increased from 3 to 4 with the addition of mental preparation. This was a fairly recent and by no means obvious conquest that became necessary after the mental aspect had been neglected for a long time. Overlooking mental skills while preparing for a competition jeopardizes all

the work done by the athlete. In fact, if we regard athletic, technical, and tactical training as the foundation of competitive potential, the mental aspect focuses on the expression of that potential. Hindering one's potential negatively impacts the competition as a whole, negating the efforts of months or even years of training.

Trying to better specify Mental Training we can say that it is the training of psycho-physiological skills useful for improving performance, both in training and competition. It focuses on enhancing the athlete's mental abilities and the intervention during the course of a program, consists of a series of techniques and methodologies implemented by a qualified operator of psychology applied to sports.

Moreover a Mental Training program aims at developing cognitive, emotional and behavioral skills in a way to improve the performance of athletes, but the same techniques can also be used in software development yet to increase performance and also psychological well-being of programmers. The overall procedure is similar to that of sport but the contents are changed given the different application context. Such adaptation becomes particularly necessary for the software developer that is a highly specialized profession.

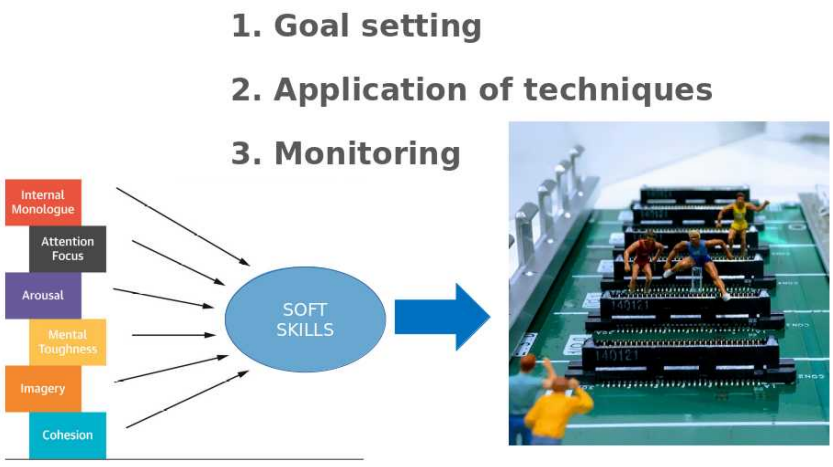

Figure 2: From sport to software development

First, it is important to identify the **goals** that the developer wants to achieve through mental training. These goals strictly concern the mental abilities we need to tune up. For example, you may want to increase concentration and focus while working, improve capability to handle stress that unfortunatley is never lacking in work, de-

velop greater creativity or improve communication. To identify one or more objectives, diagnostic tools are used such as self-report questionnaires or structured interviews carried out by experts to assess developer's behavior in daily work. The initial diagnostic phase provides an analysis of the current level of mental skills along with data to identify the objectives on which the training program will focus.

Once the objectives have been identified, various mental training **techniques** can be used to achieve them. For example, mindfulness meditation can be used to improve focus and stress management, while biofeedback can help boost attention.

During the application of the techniques, **monitoring** is also carried out to verify gains and make adjustments when necessary.

In this process we must emphasize the importance of working with an experienced professional. This is necessary to guarantee on one side that the training techniques are suitable and used in the correct way to maximize effectiveness of the training and, on the other hand, to guarantee and protect the psychic balance of the coachee.

We can better understand Mental Training by comparing it to pure counsulting and coaching. Consulting is that activity in which the expert of a certain subject advises and assists his client by providing or implementing information, opinions or solutions through his know-how and problem solving skills. On the other hand, coaching is the process of developing a person's potential to maximize their performance. It is implemented by helping them learn and become independent, rather than simply transferring knowledge. This definition was proposed by John Whitmore, a renowned coaching theorist.

Therefore we can say that consultancy is a directive type activity, while in coaching there is a supportive attitude. These attitudes are not to be considered clear-cut but more like trends because both coaching and consultancy can present directive and supportive moments. So if consulting tends to be directive and coaching tends to be supportive, we can place Mental Training more or less halfway.

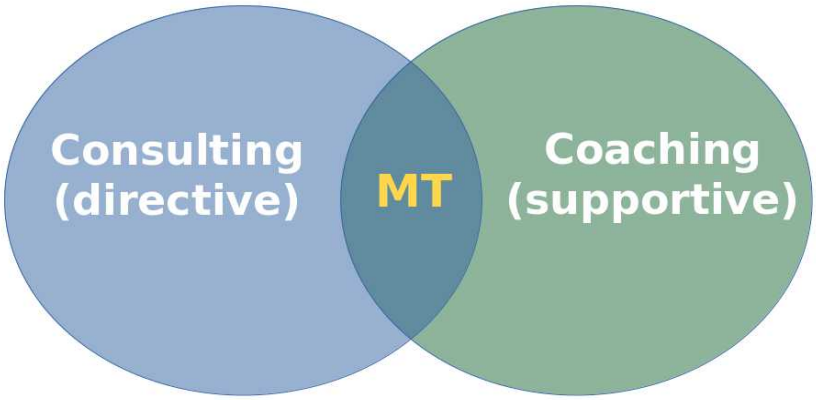

Figure 3: Mental Training comparison

In fact, in Mental Training we have a directive attitude that is necessary in the initial stages to train on the techniques, both to produce a change during the MT program itself and to put the athlete, in our case the developer, in the condition to become autonomous and train on his own. But this last attitude is supportive because it fosters independence of the developer by providing with tools to maintain a high quality work. So as a conclusion we can say that Mental Training is halfway pure consultancy and coaching.

Let's explore some of the contributions provided by mental training to developers' work.

- The first tangible effect of an effective training is the increase in performance, that is productivity in general and better code in particular.
- As a result, well-being and job satisfaction are also increased.
- Training mental skills improves code quality by reducing the need for refactoring and helping to contain technical debt.
- This work adds the so-called soft skills to the continuous training every developer is used to. This aspect of soft skills is always quite underestimated but in reality, with other conditions being equal, it is what distinguishes one developer from another, especially in this moment of transformation due to the adoption of artificial intelligence in daily work processes.
- As a matter of fact, the enhancement of mental skills better prepares the developer for the opportunities and challenges posed by the new tools based on generative AI.

- Finally, the training of mental skills develops cognitive, emotional and social spheres of the programmer fostering well-being that extends beyond the workplace.

It is important to note that a Mental Training program is real training. It is not a matter of employing a certain technique in an episodic or sporadic manner. In order to have tangible results, both numerically from instrumental evidence but, above all, in everyday work, it is necessary to train and refine oneself in the use of the techniques. Just like athletes do in sport.

Among expected benefits is the solution to situations such as the dilemma between code quality and milestones, as shown in this humorous flowchart.

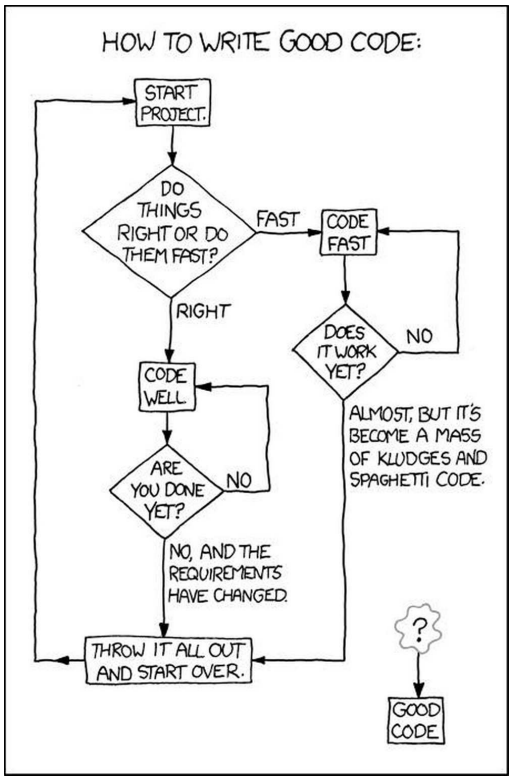

Figure 4: Writing good code

If we waste time taking care of the quality of the code, the development requirements will change forcing us to start over. If, on the other hand, we choose the path of speed, the resulting code will be so ugly and inefficient that we always have to start over. Bottom line is, that you can't write beautiful code in a real project. But humor aside,

and effective training of mental skills helps the developer meeting deadlines while maintaining a good quality of the code base.

Chapter Summary

In this initial section we looked into the structure of the book, the recipients and people to whom the contents are less suitable, the learning objectives, then we have described Mental Training, we saw how it is adapted from the world of sport to software development, compared it with coaching and consultancy and we talked about the effects on the developer's daily work.

Before we proceed, you can opt to complete a brief assignment if you feel like. Try to remember two situations you experienced at work. You can later use these situations as a reference to apply the coaching to your job experience.

- **WIN situation**: Take note of a situation in your career in which you got a positive result and you are particularly proud of.
- **LOSE situation**: Take note of a situation in your career you're not satisfied with and in which you feel you could have performed better.

In the first situation, try to recall a result that is significant for you or a acknowledgment to your skills and competences. It can be the implementation of a new feature or the fix of a difficult bug or anything you feel satisfied with. Take note of this situation by focusing not on the external conditions but on your qualities as a professional. For example, if you got an appreciation for the creativity with which you solved a bug, write creativity as your quality.

In the second situation try to remember a moment in your career when you failed a goal or you are not particularly satisfied with. Here think about what you should have done more, or even less, to achieve that result and try to identify one or more of your personal qualities that you think could have worked in that case. For example, if you made a mistake causing a malfunction in production, as a personal quality you can mark concentration.

While this activity is not mandatory, engaging in it can enhance your knowledge and understanding during the coaching.

Mental Skills

Welcome to the part of the book regarding mental skills. Here we are going to delve into the skills subject to analysis and tuning in Mental Training for developers.

For each of them we will provide a description, theoretical framework, the practical applications in the developer's work and we will list tools and techniques available for training.

What are Mental Skills

In general terms, we can consider a mental skill as person's ability to perform tasks. These tasks usually require learning, processing and applying knowledge, making decisions, reasoning and understanding complex ideas. Together with emotions and experience they contribute to the output of the person's behavior and are based on cognitive processes in order to function. For example, critical thinking, planning and imagination are all mental skills. As you can easily guess, mental skills are fundamental in many areas of life, including work, education and interpersonal relationships.

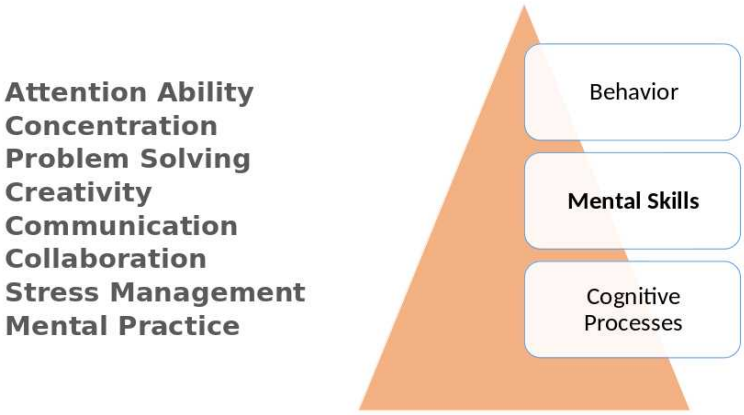

Figure 5: Mental Skills

In the Mental Training for developers we find the following mental skills:

- **Attention Ability** allows the person to direct and maintain

awareness towards a certain stimulus;
- **Concentration** is the activity of eliminating distractions;
- **Problem Solving** is the ability to analyze problems, develop solutions and choose the most suitable one among these;
- **Creativity** can be seen as the ability to connect elements and have insights from different fields to create new solutions;
- **Communication** is the exchange of information, ideas and thoughts between individuals or groups by various means of transmission;
- **Collaboration** can be seen as a mental skill that requires the ability to cooperate with others and follow collective decision-making processes;
- **Stress Management** is based on the stress concept borrowed from Biology. Stress is intended as the body response to every request made by the external environment to maintain internal balance;
- **Mental Practice** is the mental repetition of an action, thought or situation in order to improve performance, increase awareness or manage emotions.

One impotant aspect to keep in mind about mental skills is that they can be developed through exercises and training.

As mentioned earlier, mental abilities are based on cognitive processes. By “cognitive process” we refer to a series of mental activities involving information processing, organization, interpretation, storage and retrieval. Mental skills and cognitive processes are closely related. In fact, cognitive processes can be considered as building blocks that skills use to solve problems and make decisions. For example, to solve a complex mathematical problem, it is necessary to sequentially use attention, retrieve previous information (that is, memory), perform logical reasoning, and process information.

Cognitive Processes

Cognitive Processes are at the basis of mental skills. We find 5 cognitive processes underlying mental skills namely attention, perception, memory, reasoning and language.

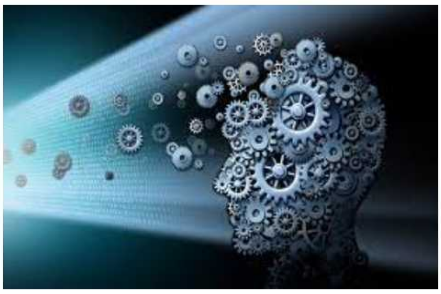

Figure 6: Cognitive Processes

Attention refers to the ability to focus on a particular stimulus while filtering out irrelevant ones.
Perception concerns the ability to process information that comes from senses such as sound, sight, smell and taste.
Memory encompasses the storage and retrieval of information, both long-term and short-term.
Reasoning involves analyzing and evaluating information to come to conclusions.
Language is the ability to process phonemes, produce words and formulate intelligible sentences.

Although to different degrees depending on the case, these processes are all always operational and active while mental skills are put in action.

Behavior

Behavior is another component tightly related to mental skills.

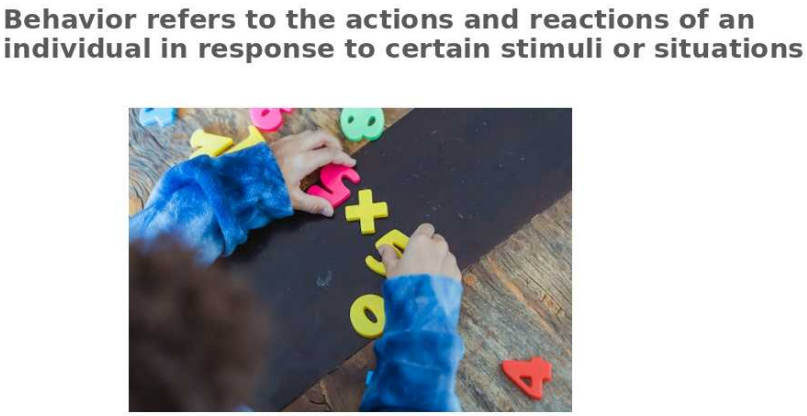

Figure 7: Behavior

Generally speaking, behavior is the visible and observable expression of mental, emotional and physical activity of individuals. As we have seen, mental skills are an important component underlying human behavior, significantly influencing it.

People who have superior mental abilities are often able to make better decisions and solve problems more efficiently. However, mental skills are not the only component of human behavior, factors such as experience, emotions and motivation also play a significant role.

Attention Skill

In this part of the book we will see what the attention skill is, we will describe the Attentional Style theorized by psychologist Robert Nideffer, and we will examine the practical applications for the developer providing a list of possible techniques for training.

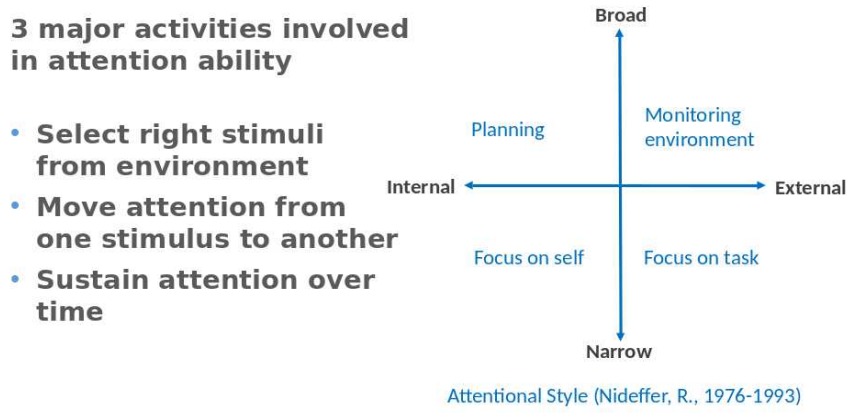

Figure 8: Attentional Style

Attentiveness is the ability that allows us to maintain awareness and cognitive resources on a specific activity for a certain period of time. This ability is essential for the software developer's job, in which it is often required to stay focused on one or more problems for many hours a day.

Three major activities take place when we use attention:

- selection of the right stimuli from the environment
- shift of attention from one set of stimuli to another
- keeping the attentional focus over time

In 1976, psychologist Robert Nideffer developed a model describing attentional style. In this theory Nideffer describes four styles: broad, narrow, internal, and external. Each style represents a way of using attention that can be interesting to analyze and apply to the developer's case.

Focusing on the axes, the broad attention style is characterized by widespread attention to a variety of stimuli.

On the other side, the narrow attention style focuses on a single stimulus.

The internal attention style is characterized by a focus of attention on the self, for example on one's thoughts.

Finally, the external attention style is about paying attention to details of the environment.

Moving from the axes to the squares we notice that an external and broad attentional style is activated, for instance, when we monitor the environment. An internal-broad one while adopting a general point of view or evaluating a range of possible solutions. The external-narrow style is used when attention is focused on single elements from the environment, while the internal-narrow style when one is "listening", so to speak, to a signal coming from ourselves such as an emotion, a thought or a sensation of the body.

The ability to maintain attention is quite important in the developer's job.
A certain skill in managing attention allows to focus on the quality of the code, solve problems faster and develop more efficient algorithms. It allows to stay focused on programming for extended periods of time, ignore distractions keeping them *sub-limen*, that is below the threshold of awareness, and therefore to maintain focus on the business logic. Furthermore, good attentiveness can help us maintain productivity even during the most boring phases of the programmer's work, such as writing documentation.

However, the developer must be able to modulate attention and change style efficiently based on the task at hand.

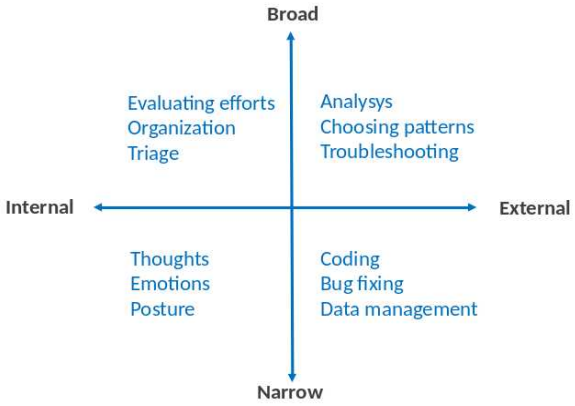

Figure 9: Attention Applications

For example, a broad-internal style allows to organize own and others' work, when coordinating other team members, evaluating effort for a certain implementation, identifying and classifying elements and so on. In using a broad-external style, the developer will perform analysis, choosing design or architectural patterns, troubleshooting, and so on. With a narrow-external style, s/he will do activities such as writing code, addressing bugs and handling various types of data, while a narrow-internal focus will allow to observe own thoughts, monitor emotional state and detect in time pain due to incorrect postures, helping to prevent possible musculoskeletal issues.

Therefore, a developer in their everyday work faces numerous situations that require the use of multiple attention styles. A flexible attentiveness is essential to perform the tasks of software development and meet the constant challenges of this profession. It is possible to make an analogy between switching from one attentional style to another and CPU task switching. In both cases there is a cost in terms of resources which decreases as the switching efficiency increases, clearly this cost in the case of the developer, who is a person, is about cognitive resources. Therefore, the greater the efficiency in passing from one task to another, or in changing the attentional style, the lower the cognitive cost of the operation, and a lower consumption of resources, as we know, frees them up for other activities. It is evident how crucial it is for developers to hone their attention skill.

Various tools and techniques can be employed for attention training.

For example **mindfulness meditation**. Among the various benefits of this type of meditation, we find an increase of emotional centering, decrease of anxiety and reduction of needless consumption of energy due to rumination and muscle tension.

Other tools for training attention include **biofeedback**, in particular one of the measures present in biofeedback which is HRV, Heart Rate Variablity. This measure represents the degree of coherence between heartbeats which relates directly and positively with cognitive performance, also benefiting attention skill as a whole.

Even **imagery**, consisting of visualizing and mentally repeating the program flow, the structure of the code or the solution to a specific problem, trains narrow-external attention and helps maintaining focus on the task at hand.

Finally, the famous **Pomodoro technique** is also worth mentioning to train attention. It involves alternating work with both short and longer breaks in order to maintain focus on a specific task and avoid distractions. The advantage of the Pomodoro technique is that it can be carried out easily and independently by the developer, without external support and supervision.

If you have completed the activity described at the end of the Introduction chapter, try now to compare what you have just learned about the attention skill with the WIN and LOSE situations. What role did attention play in the positive outcome? Was it a central or less important factor? And about the LOSE situation, do you believe it was due to a lack of attention?

Concentration Skill

Concentration is the ability to keep attention on the task at hand without being distracted by unrelated elements, that is the distractors. This mental skill is closely related to that of attention as it helps focus on the activity of interest instead of a distracting factor. In all jobs, including that of the developer, concentration is a crucial factor to work efficiently and effectively.

In 1975, Hungarian psychologist Mihali Siksentmihali formulated the Theory of Flow, also known as Theory of Optimal Experience. According to this theory, the Flow is a state of mind in which a person is completely immersed in a task, fully focused, and with a sense of control and satisfaction. It is the moment when you lose track of time because you are so focused on the task, and enjoying the experience so much, that you are unable to think of anything else. We can consider it the maximum degree of concentration. The name of the Flow Theory comes from the fact that patients described these experiences as feeling like being carried along by a current of water. In sports, Flow is a condition that preludes *peak performance*, which is that performance level going beyond the average level of the athlete's best performance. Flow is therefore a status in which one is so immersed in the task that completely ignores distractions.

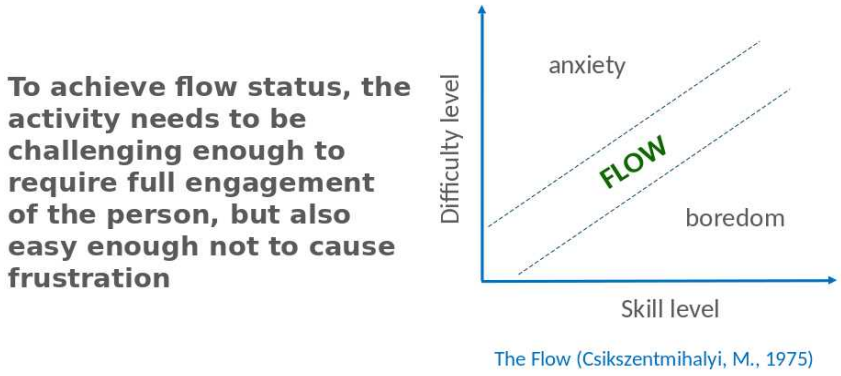

Figure 10: The Flow

To reach such status it is necessary that the activity is challenging enough to require commitment, but not too much so as not to be frustrating. The right level of difficulty is subjective and should be

proportionate to the skill level, as shown in the graph. If the task is too difficult there will be anxiety due to a sense of inadequacy, vice versa if the person is too skilled the activity will be boring.

Motivation is essential for good concentration and especially for achieving a state of flow. In fact, one of the major contributions to the ability to concentrate comes from the level of interest one has in the task. Usually in the developer this interest is always quite high given the passion for programming of these professionals, but obviously this cannot be taken for granted.

Let's see other aspects of concentration and distractions in the work of the developer. Distractions can be divided into two main categories: internal and external.

Internal distractors are those distractions coming from within the person. For example personal issues such as errands to be carried out, problems and worries of various kinds or a negative self-talk, that is a dysfunctional internal dialogue regarding the task. Even the idea of close project milestones are internal distracting factors, tasks in queue or already done, thoughts about possibible regressions and so on.

External distractors include incoming phone calls, conversations between colleagues, notifications from phones and computers, social media notifications, emails, traffic noise, construction works in the surroundings, and so on.

These and other distractions are always present in the work environment but a good level of the ability to concentrate will help the developer maintain or quickly return attention to the task, therefore protecting performance and productivity. Additionally, enhanced focus efficiency diminishes the cognitive resources required for the brain to actively suppress distractions.

Let's see some strategies, which can be used to train the ability to concentrate. Here too we find **mindfulness meditation**, which increases emotional and cognitive centering by reducing the possibility that incoming distractions interfere with the task.

Other tools to train concentration include **HRV measure of biofeedback**. This measure represents the coherence between heartbeats and improves cognitive functioning, helping to keep distractions below the threshold of awareness.

Imagery, or visualization, is the mental repetition of an action, thought or situation and produces an effect on a cognitive level similar to that of biofeedback.

Similar to the routines that athletes use, developers can also use **rou-**

tines to help manage distractions. The routine involves performing a short sequence of actions, like a mini-ritual, to help regain concentration when distractions become too overwhelming.

Lastly, also here we propose the **Pomodoro technique**, which involves alternating work and both short and long breaks. This tool helps keeping the focus on a specific task and avoid distractions with the advantage, compared to other techniques, that it can be used by the developer without external support and supervision.

If you have carried out the activity proposed at the end of the Introduction module, try now to compare what's above about concentration with the WIN and LOSE situations you took note of. What role did concentration play in the positive result? Was it a central feature? And also in the LOSE situation try to understand if concentration was not enough.

Stress Management

Let's now talk about the stress management skill of the developer. We will see this ability by referring to the work of the Austrian doctor Hans Seiie, famous for his research on stress and for the theory called General Adaptation Syndrome, abbreviated with the acronym GAS. This theory describes the way in which the human body reacts to stress through a homeostatic response. Homeostasis is a concept borrowed from biology and is described as the ability of the living organisms to maintain their characteristics in response to external conditions change, through a self-regulating mechanism. It is the organism's attempt to maintain an internal stability despite changes in the external environment.

In GAS, Seiie's describes three main phases of body response to stress.

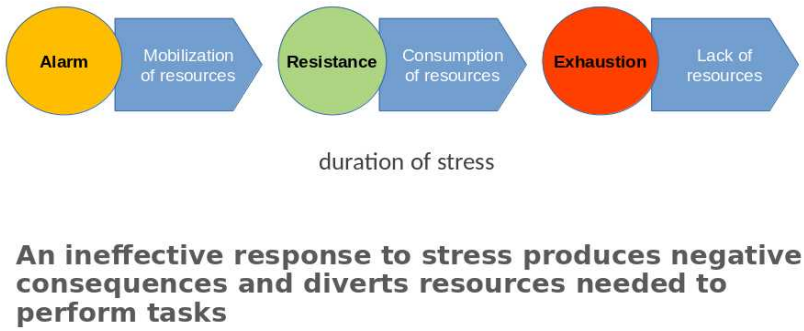

Figure 11: General Adaption Syndrome

An initial phase of *alarm* in which the threat is detected and the sympathetic nervous system is activated, which prepares the body for the response to the stressors. Here hormones such as adrenaline and cortisol are released, the heart beats faster, breathing quickens, blood pressure rises and the blood flow to limbs increases. That is, the organism prepares itself for action by mobilizing resources.

If the stress persists, the body switches to the *resistance* phase. In this phase, the body acts against stress and tries to restore its internal balance with respect to the changed environment, that is, it tries

to adapt to stress and manage it the best possible way, using the resources available. If this phase persists for too long, the body can consume all the resources and find itself unable to further cope with stress, which is the *exhaustion* stage. In this phase the resources are exhausted, the body remains defenseless and, in severe cases, with a compromised immune system which exposes the body to possible physical and mental illness. All this happens when the stressful stimulus, that is the source of stress, is chronic and lasts for long time.

Aside from clinical implications, which is nonetheless an important aspect for developers' health, another concept that interests us is that an inadequate response to stress involves the use of resources that could otherwise be utilized for work. In fact, intrusive thoughts, also known as ruminations, as well as the persistent muscular tensions that we often experience in our bodies, rob us of energy that we could instead allocate to bodily functions (e.g. the immune system), work tasks, and activities that enhance well-being and performance. Therefore, it is crucial to minimize such a complete waste of energy.

Let's see some unwanted effects of stress on the developer's job and life.

First of all, it is easy to imagine that a mismanagement of stress leads to a **limited productivity**: stress and anxiety can hinder the ability to process information, slowing down the pace of reasoning and preventing the developer from completing work efficiently. Stress also causes **performance degradation** with an increased error rate. When you're stressed it's easy to overlook important details and make mistakes that often reveal themselfes only in production. Stress can create **communication and collaboration issues**. In fact, it can interfere with the ability to communicate effectively with colleagues leading to misunderstandings or even conflicts that hinder individual and team work. There can also be a **deterioration in mental health**: chronic stress can lead to a number of mental health issues, including anxiety, depression, burnout and, in extreme cases, even drop-out. Due to the fact that private and professional life are closely linked, these problems can have a significant impact not only on work but also on other areas of the developer's life, such as friends and family. Finally, a **lowered quality of life** can manifest itself through physical problems such as migraines, gastrointestinal disorders and other somatizations like skin diseases, sleep disturbances and exposure of the body to a number of issues.

In order to train the developer's ability to manage stress, a Mental Training program starts by identifying and classifying the stressors, a phase similar to the triage activity of support tickets. The stressors are then ordered starting from the most anxiety-provoking, that is a descending sorting is applied, and finally the most suitable technique

is chosen and used against the most urgent stressor. The same way as if we were applying a fix.

Tools and strategies are many, above all we find a healthy, sustainable and possibly non-toxic **work environment**, but this is more a hygienic aspect and mainly out of the direct control of the developer. To personally train the ability to manage stress, the developer can use the **mindfulness meditation** we have already talked about, **autogenic training**, which is a fairly well-known practice capable of inducing physical and mental relaxation, some body relaxation techniques such as the **muscle relaxation by Jacobson**, breathing techniques such as the **diaphragmatic breathing**, **mental breaks** which consist in interrupting work and doing something that does not involve the use of technology and, last but not least, **physical activity** done regularly and possibly of aerobic type, which here we suggest for the benefits going even beyond the work context and for ease of execution. Twenty minutes of light jogging or even brisk walking is enough to be done every day, preferably outdoors and amidst the greenery.

If you have carried out the activity proposed at the end of the Introduction chapter, as usual I invite you to compare the information about stress management with the WIN and LOSE situations you took note of. Were you able to keep stress under control in the positive result? And also in the LOSE situation try to figure out if stress was amongts top reasons of the setback.

Creativity

In this section we talk about Creativity as mental skill of the developer. As usual we place the skill in a theoretical framework, observe the implications in everyday work and provide some ideas to plan workout.

According to the Hungarian psychologist Mihaly Siksentmihali, a key element of creativity is the ability to combine ideas and concepts from different fields to create new and original solutions. Creativity is more likely to occur when we are in a state of flow, also an important concept of the concentration skill, or when an activity excites us to the point that we lose track of everything else, including time. For Siksentmihali, creativity requires the ability to produce new and original ideas, which are useful, meaningful, integrated, and elaborated in a coherent and structured way.

Figure 12: Ideas in creativity

Original in the sense that they are new and have objective advantages over solutions already known. *Useful* because they have to solve a problem or satisfy a need. *Significant* to the person who produces them, the group in which the person works or to the society in which the person lives. *Integrated* because the new idea must integrate different concepts in a coherent and harmonious way.

This skill is clearly fundamental for developers because it enables them to identify and implement innovative solutions to problems that emerge daily in the dynamic field of information technology, where new technologies are constantly emerging.

So, creativity is a highly valuable skill for programmers and in particular it plays a pivotal role in productivity. It enables devs to craft innovative features, write more efficient code that enhances application performance, foster better implementation of business logic, elevate user experience, contribute to the development of knowledge within the IT industry, and distinguish clients from their competitors through the creation of more appealing products and services.

Developers can utilize various strategies to train creativity, from generic ones that are valid for anyone such as brainstorming to those more specific for the software developer. First, let's examine the strategies that are more specific to software development since they can be practiced more independently.

For example, **expand your horizons**. Reading books, watching movies, listening to music, and having experiences that expose to new ideas and concepts may seem trivial to some people but, especially when you're stuck on a programming problem, doing any of these activities helps to acquire new perspectives and see things from a different standpoint, unlocking the creative process. **Experimenting with new means** such as new development environments, languages or frameworks allows the acquisition of other skills, and new hard skills always offer new solutions. Engaging in **lateral thinking** is a technique that allows to explore different and unconventional solutions to problems. When a problem arises, try asking different questions to find answers that are not obvious or predictable. We include also the well-tested group techniques for producing ideas in the workplace such as **brainstorming, six hats, team work itself**, etc. And finally, to encourage creativity and facilitate insight, other individual techniques that we've already discussed such as **mindfulness meditation** and **biofeedback** still fit the bill.

If you have completed the activity proposed at the end of the Introduction chapter, also here I invite you to evaluate creativity in the WIN and LOSE situations. What role did creativity play in the positive result? Was it a central or fringe factor? And in the LOSE situation try to figure out if being more creative would have come to help and how.

Problem Solving

Welcome to this section about the problem solving skill. Our reference for this skill is the theory of psychologists and computer scientists Alan Newell and Herbert Simon, who in 1975 won the prestigious Turing award. In their theory, Problem Solving is regarded as the ability to analyze problems, find possible solutions and choose the most suitable for the problem. The two scientists are considered among the fathers of artificial intelligence, which as you know is a very hot topic at the moment. Probably you're probably already familiar with their work, especially if you're into artificial intelligence, but it may be worth a quick refresher. Newell and Simon's theory, known as the *problem-space model* is a theory of problem solving that explains how people approach problems and find solutions. When people are confronted with a problem, they create a mental representation of the problem and possible solutions, such a representation is called "problem-space". The person's goal is to move within this space to find the best solution.

Figure 13: Solving problems

The problem-space model describes three main stages of the problem solving process:

1. Analyze the problem
   The first step is analysis, where the problem solver analyzes the problem, identifies the relevant information and organizes it in order to understand the problem as a whole;
2. Generate solutions

The next step is the generation of solutions. In this stage, the person generates several possible solutions to the problem using knowledge, experience, and creativity;

3. Evaluate solutions
   Lastly, the person moves on to the evaluation of the identified solutions. In this phase, an evaluation is made and the best solution is chosen based on criteria such as effectiveness, feasibility and convenience.

It's easy to understand how this mental ability is closely linked to that of creativity, which is evident in the second phase of the problem solving process. In fact, good creativity will make it possible to generate new and original solutions, with all the advantages we talked about in the section dedicated to the creative process.

Problem solving is essential for computer programming.

It is necessary to implement the development requirements, that is, to move from user requirements to technical ones, and therefore to a working code that does what it is expected to. In fact we can see the development requirements, as well as other activities such as troubleshooting and bugfixing, as problems for which the developer must find and apply solutions. These are all basic activities in software development.

A good problem solver can analyze problems systematically, identify root causes, and implement lasting solutions. Here, it is necessary to eliminate, or at least minimize, the introduction of new bugs arising from side effects of the solution. Regrettably, dealing with this type of issue is a common part of a developer's life and strong problem solving is mandatory.

The problem solving skill is also needed to manage complex projects. From the very beginning, complexity has been an integral part of software development. Today projects are even more complex because applications do more and more stuff and this trend seems to be destined to remain stable for the future as well.

Finally, developers who are able to solve problems effectively also work better with other team members, helping to address and solve problems faster. This is also important because teamwork is prevalent workmode in software development projects.

Let's move on to the possible strategies to train the problem solving skill. Here too **biofeedback** comes to help, since it intervenes on mental processes such as reasoning, memory and attention, increasing cognitive performance. The **code review** is the activity according to which a colleague with equal or greater experience examines the code along with the developer, looking for optimizations in terms

of functioning but also for good programming practices. This mode offers the developer different perspectives which, as we know, is a key element when searching for solutions. The same applies to **pair programming**. Here too, working with others trains the ability to solve problems, in this case, however, not only by examining code that has already been produced but by writing it together with a colleague. **Exploration** involves trying alone or in group in solving new and never-before-managed problems, possibly also in a competitive form such as leetcodes, coding dojos and hackathons. In addition, also the study of new patterns-technologies-methodologies are all effective workouts. Finally, **mental experiments** consist of trying to imagine solutions to an already known programming problem, evaluating the pros and cons of each solution. This is also good for training the problem solving skill. Here I would suggest this latter technique yet because it is more suitable to practice alone than the others.

If you have carried out the activity proposed at the end of the Introduction section, I invite you here too to compare what you have just seen about problem solving with the WIN and LOSE situations. How much did your ability to solve problems influence the positive result? And in the LOSE situation, try to understand if better problem solving would have prevented failure and how.

Communication Skill

We are going to explore the communication skill as defined by Shannon and Weaver, whose work has had a profound impact on the development of numerous communication theories. In their framework, communication entails the exchange of information, ideas, and thoughts between individuals or groups through diverse transmission channels (Shannon & Weaver, 1948).

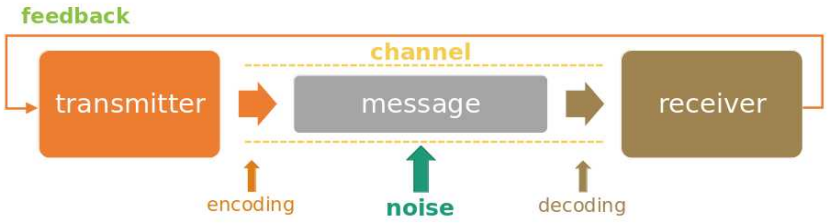

Figure 14: Communication

This theory stands as one of the earliest mathematical models of communication and centers on the transference of data from a sender to a receiver via a channel.

The Shannon-Weaver model comprises five key components: the transmitter, the receiver, the message, the communication channel, and the noise. According to this theory, the transmitter encodes the message, which is then relayed through a channel to the receiver, who decodes it. Noise refers to any disruption that can hinder the transmission. For instance, the encoding and decoding processes that occur on the transmitter and receiver sides, respectively, are sources of noise. The communication channel can encompass spoken language, writing, radio, television, and so forth.

This theory emphasizes the importance of feedback, which refers to the recipient's response to the message. This response is important because it affects how the sender sends future messages. The Shannon-Weaver model focuses on the amount of information trans-

mitted across the channel, rather than the quality or interpretation of the message. We can draw an analogy to the Signal-to-Noise Ratio (SNR) we encounter in signal theory. Due to the noise introduced by various sources and the resulting distortion in communication, transmitting a message in its original form is not a straightforward task, which is particularly true in human communication. Just consider how challenging it can be to fully convey your thoughts and deliver a concept with precision, as it was originally conceived in your mind. This is one of the most demanding aspects of everyday communication, and unfortunately, it's almost inevitable that misunderstandings will arise.

Programmers must be able to communicate not only with computers through programming languages but also with all the people involved in a software development project. Good communication enables *effective collaboration*, where developers can interact effectively with team members, including other developers, systems engineers, designers, project managers, quality assurance personnel, and customers. Such communication occurs on a daily basis in any software development project, particularly in agile environments with daily stand-ups and ongoing interaction.

Moreover, the developer's job often involves identifying and solving complex problems, and good communication skills are important in this process. They allow team members to work together to isolate the cause of the problem and find the most suitable solution.

Communication is also essential for *project management*. A developer must be able to inform the PM about the status of assigned tasks, the critical issues, the difficulties encountered and the expected delivery dates. This communication helps ensure that the project is on schedule and within budget.

Finally, communication is often required with *users, customers, and clients*. Technicians must frequently interact with these individuals to understand their needs, answer questions, provide project updates, and figure out how to improve the software. Effective communication with these stakeholders helps ensure customer satisfaction and delivery of a quality product.

Let's move onto the techniques that developers can use to train their ability to communicate effectively. Given the nature of communication these techniques are to be used mostly in groups, although we will see a way to practice individually as well.

Code review is a practice in which two or more team members examine previously written code to identify and correct errors, bugs, or design flaws. This practice requires communication and promotes collaboration between team members who are working to improve the

code quality.

Pair Programming is a software development technique in which two programmers work together at one workstation or through remote desktop technology. One programmer, the driver, writes code, while the other, the observer, reviews and provides feedback. This technique promotes strong communication between programmers, requiring sharing of ideas and concepts, and can lead to higher-quality code.

The **daily standup** is a daily meeting where team members come together, face-to-face or online, to discuss the previous day's progress, current day's goals, and any issues or roadblocks that need to be addressed. This practice promotes constant communication between team members and helps keep everyone informed of project progress.

User stories are a practice that describes the desired behavior of the software from the user's point of view. This technique promotes effective communication between developers and project stakeholders to define the functionality required by the software.

Prototyping is a development activity where you create a prototype to test functionality and usability. This technique promotes open communication between developers and project stakeholders to receive feedback on software features.

Cross-functional collaboration is an important practice that promotes communication between team members belonging to different functions, such as for example developers in other projects, testers, project managers, etc. This also allows you to interact with non-technical people towards whom the effort to communicate clearly is greater, making this activity even more effective in honing the communication skill.

Knowledge sharing consists of contributing to knowledge sharing systems such as wikis, blogs, forums, etc. This not only benefits the growth of the work group in terms of skills, but writing down your own knowledge greatly helps train the communication skill.

For ease of use, we recommend Knowledge sharing here since it is the only one, among all these activities, that it is possible to practice alone. However, within possible constraints the other options are also perfectly viable.

If you have completed the activity suggested at the conclusion of the Introduction, also here attempt to compare the information on communication with the WIN and LOSE situations. What role did your communication skills play in the positive outcome? Was it a significant

factor or a minor one? Also, in the LOSE situation, try to determine if being more proficient in communication would have been beneficial and how it could have helped.

Collaboration Skill

According to American psychologist Lillian Gilbreth, collaboration is a skill that requires the ability to work with others in a cooperative way. In particular, collaboration requires sharing ideas, solving problems together, having an open-mindedness towards other team members and being able to coordinate effectively. Gilbreth stressed out the importance of developing collaboration as an individual skill, because she believed that an individual capable of working well in a team could successfully perform any job that interaction of multiple people. According to this author's vision, collaboration as an individual skill is composed of a technical and social part.

Figure 15: Collaboration

These two aspects are distinct yet complementary for effective collaboration. ***Technical proficiency*** refers to the skills and knowledge required to perform a particular activity or task. In a collaborative setting, technical proficiency involves having a comprehensive grasp of the field of work, operating procedures, and the technical skills necessary to significantly contribute to teamwork. For instance, in a software development project, technical proficiency might encompass knowledge of programming languages, proficiency in development tools, and the ability to resolve technical issues. Thus, technical proficiency can be encapsulated as hard skills, a widely recognized concept nowadays.

Social competence refers to the interpersonal skills and relationship-building abilities needed to work effectively with others. This com-

petence encompasses open-mindedness, active listening, empathy, the capacity for clear and effective communication, conflict resolution expertise, and the ability to foster trusting and collaborative relationships with colleagues. Social competence is crucial for establishing a harmonious and productive work environment where team members feel heard, valued, and encouraged to share their ideas and perspectives. This ability falls under the umbrella of so-called soft skills.

In summary, collaboration is an individual mental ability that demands a set of technical and social skills, the acquisition of which is crucial for the success of teamwork and for the attainment of shared objectives.

The benefits of collaboration for both the developer and the organization are widely recognized in software development.

Collaboration empowers developers to work with others on a project, sharing technical knowledge and expertise. This translates to a refinement of the code's quality and discovery of more efficient solutions.

It facilitates the exchange of ideas and the adoption of best practices. This can lead to innovation, a deeper grasp of problems, creative solutions, and a reciprocal chance for professional growth.

Collaboration between developers can foster better project management. Through the allocation of tasks, the delegation of responsibilities, and unwavering communication, it is feasible to optimize the workflow and ensure superior efficiency in the delivery phase.

Collaboration also presents a chance for ongoing learning and professional development. By interacting with fellow developers, one can glean new knowledge, discover novel technologies and methodologies, and refine communication and teamwork skills.

Finally, collaboration enables the establishment of connections and relationships within the developer community. This can nurture career opportunities, business partnerships, and the sharing of resources and information that will aid in career advancement.

Now let's explore the tools that the developer can use to improve the ability to collaborate. They're the same we've seen for the ability to communicate, which is hardly surprising given that collaboration and communication are interdependent skills.

So also here we have **code review** which is a practice where two or more team members review already written code for errors, bugs or design flaws.

Pair programming is a software development mode where two programmers work together on a single computer or desktop shared re-

motely. This technique involves close cooperation between the two programmers and therefore is a great workout.

The **daily stand-up** is a daily meeting where team members get together to discuss the previous day's progress, current day's goals, and any issues or blockages that need to be addressed. This practice is collaboration-intensive.

User Stories are a technique that describes the desired behavior of software from the user's point of view. This technique requires collaboration between developers and other project stakeholders.

Prototyping is a development activity where a prototype is created to test the functionality and usability of the software. Developers and other project stakeholders collaborate on the creation of the prototype.

Cross-functional collaboration is a practice in which interaction between team members belonging to different company functions is used. This practice encourages developers to step out of their purely technical comfort zone and collaborate with people with different backgrounds.

Finally, **knowledge sharing** consists of contributing to wikis or other systems for sharing knowledge. This practice trains collaboration well, given the collaborative nature of these systems.

For collaboration as well we suggest practicing with knowledge sharing, as it's the only one of these activities that can be practiced alone. However, if compatible with daily operations, additional tools can be added to enhance training.

If you have carried out the activity proposed at the end of the Introduction, also consider comparing the information about collaboration with the WIN and LOSE situations. Did you achieve success independently or through collaboration? And in the LOSE situation, consider how collaboration could have contributed to a better outcome.

Mental Practice

In the book “Psycho-Cybernetics, A New Way to Get More Living Out of Life”, the American scientist Maxwell Maltz discusses different mental abilities. Among these we focus on visualization as a skill to be honed in mental training for the developer. The author defines visualization as the ability to create vivid mental representations, detailing desired goals or desired situations. Maltz asserts that this ability serves as a powerful tool for shaping thoughts and actions, empowering individuals to achieve their objectives, enhance self-confidence, and conquer mental constraints.

According to Maltz, practicing visualization yields several benefits.

Figure 16: Benefits of Mental Practice

First of all, it creates a clear mental framework of our goals, allowing us to focus our attention and energies on achieving them. In addition, mental visualization boosts self-confidence and self-esteem by allowing us to mentally visualize the success we desire, which can increase motivation and determination in pursuing it.

Maltz emphasizes the importance of mentally rehearsing processes as well as final outcomes. By visualizing yourself taking specific actions and overcoming obstacles along the way, you can develop skills, strategies, and creative solutions to tackle challenges.

It’s crucial to note that while visualization can be a powerful tool, Maltz stresses out that it’s no substitute for taking action. Therefore, this mental practice should be seen as complementary and empowering

our actions to transform goals into results. Just as with other mental abilities, a lack of visualization proficiency can hinder desired behavior.

Let's move on to the practical implications of a good mental practice. Based on my firsthand experience, I believe this mental ability is ubiquitous in software development. Consider how often we, as programmers, envision the potential pathways in a program's flow as we craft algorithms, or visualize the project architecture while mentally mapping the code's structure. This is also known as *computational thinking* and the better this skill the better our programming ability.

For instance, it empowers developers to clearly visualize the problem at hand and envision potential solutions. Indeed, creating mental representations of the steps required to resolve the issue leads to better comprehension and a more streamlined decision-making process.

This mental ability also serves as a complement to creativity during development, aiding in the conceptualization of fresh ideas, innovative features or approaches, and igniting original solutions.

Moreover, mental practice facilitates the visualization of what's under the hood of complex systems or algorithms. This enhances understanding of the interactions between the various system components, enabling better design and optimization.

This ability proves valuable as well when mentally preparing for a presentation, interview, or crucial project. By visualizing yourself achieving the desired success, confidence grows and performance anxiety diminishes, contributing to a positive outcome.

It is important to emphasize that the influence of this ability reaches into observable behavior. As a result, effective mental practice should consistently be followed by tasks directed at achieving outcomes; otherwise, it risks becoming a mere mental exercise.

Let's explore some tools that developers can employ to refine their mental visualization practice.

Imagery consists of recalling or creating vivid images mentally. When visualizing, you should try to make mental images as detailed as possible by paying attention to elements such as colors, shapes, movements and sounds and adding sensory details as well in order to make the experience more realistic. The goal is to imagine seeing, hearing, touching, smelling and tasting the elements of the mental image to make it as much engaging as possible.

The **routine** is the execution of a small sequence of actions, a sort of mini-ritual, which includes the mental recall of images and sensations.

This practice helps to train awareness even on body aspects such as breathing.

Mind mapping is a technique involving visualization of an idea and its connections so that it helps organize and structure information in a logical and intuitive way. This practice especially trains the ability to convert thoughts into graphic objects, the maps indeed.

Mental experiments consist of trying to imagine solutions to an already known programming problem, evaluating the pros and cons of each solution. This is also a good training in mental practice that we suggest here for the possibility to practice alone and immediate application to real work issues.

All these tools are for constant training and practice. It is similar to how we develop other mental abilities, and here too it is crucial to emphasize that the more you practice, the more you improve in forming clear and vivid mental images, which are not just images but a real inner experience.

If you have completed the activity suggested at the end of the Introduction chapter, I also encourage you to compare what you have just learned about mental practice with the WIN and LOSE situations. What role did your ability to visualize situations and outcomes play in achieving a positive outcome? Was it a crucial aspect? And in the LOSE scenario, try to determine if possessing greater proficiency in manipulating objects mentally would have altered the outcome and how.

Assess your Mental Skills

In this section of the book, you will assess your mental skills as a developer. You will evaluate the current level of your abilities using a questionnaire designed for this purpose. In the assessment process, we will avoid paper and instead fill out the questionnaire electronically to automate scoring and get results with minimal effort.

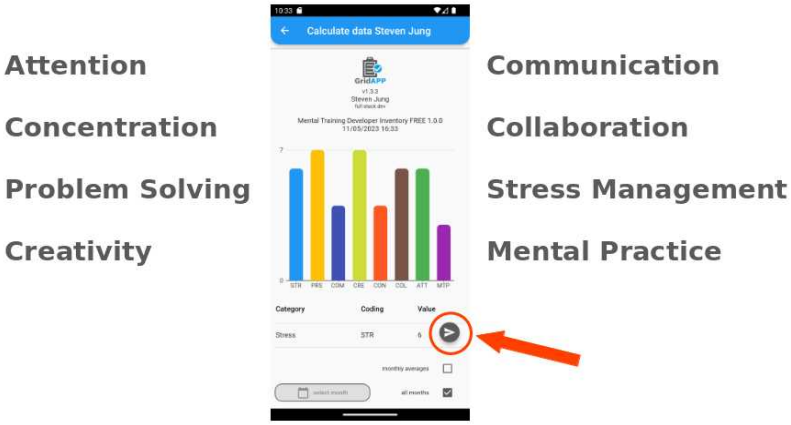

Figure 17: Assessment of Mental Skills

The analysis will encompass the mental skills we have sween so far namely Attention, Concentration, Problem-solving, Creativity, Communication, Collaboration, Stress Management, and Mental Practice.

The evaluation results will be displayed in graphical form by the application. You can forward the results to yourself via email to create an archive for future comparisons if you like.

In order to proceed with the assessment follow **this link** to open the application named **GridAPP** in your web browser, then follow the next steps.

1. Create user
   Once in the main screen create a new user. You can use any name, also a simple nickname of your choice is fine.

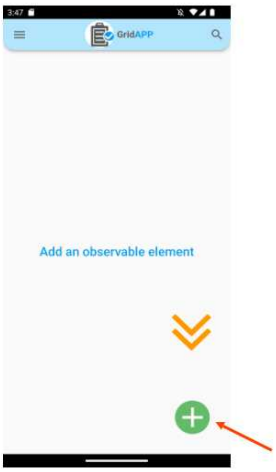

Figure 18: Create user

After saving the new user it will appear on the main screen. Click on it to get to the detail screen.

2. Find the questionnaire
   In the section below the header, expand the accordions to find the item named “Mental Training Developer Inventory FREE”.

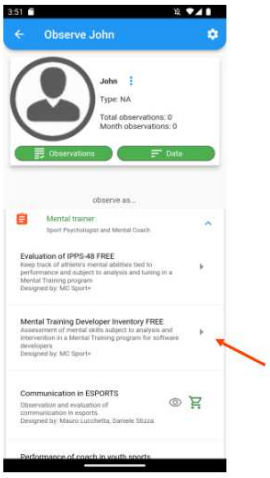

Figure 19: Find and open questionnaire

Click on this item to open the questionnaire.

3. Fill out the questionnaire

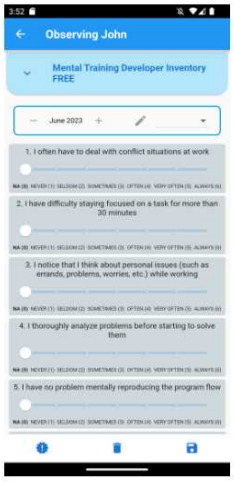

Figure 20: Fill out the questionnaire

Read each statement carefully and use the slider to answer. Try to remember exactly how often you have experienced the same situations and with the slider select the frequency. Don’t spend too much time thinking about the questions and rather answer straight away. If in doubt about the meaning of the question, choose the first thing that pops into your head and remember that there is no right or wrong answer. You have 3 minutes to complete the questionnaire.

4. Get results
   Upon completing the questionnaire, close it and return to the previous screen.

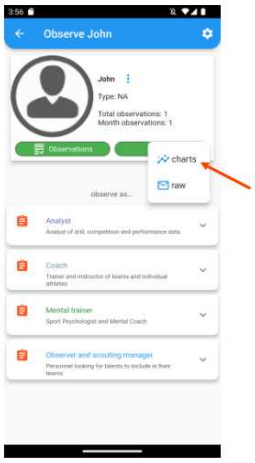

Figure 21: Get results

Here, tap the “Data” button and select “charts” from the drop-down menu to automatically perform the scoring and generate results.

Figure 22: Show data

This outcome can be shared as a PDF attachment via email by tapping the button indicated in the image above.

You can share it with yourself to create your own personal archive and also with me, your coach, for personal feedback. Simply find my contact information *here* and drop me a message.

Interpreting the results is simple stuff.

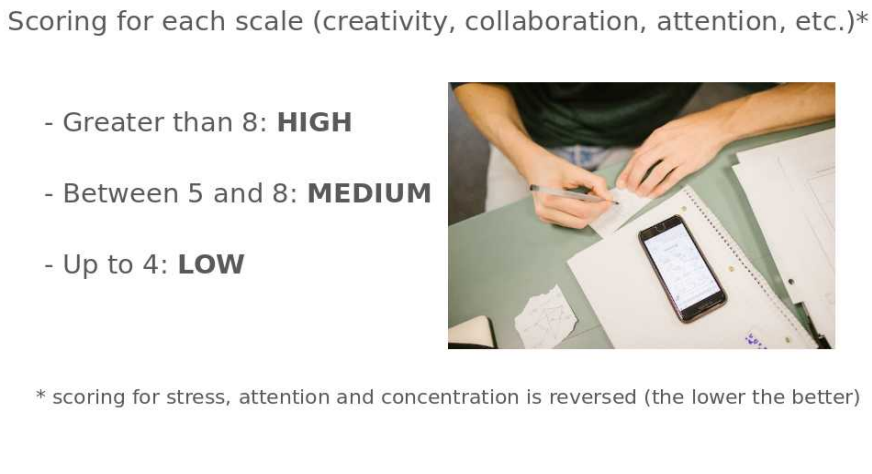

Figure 23: Scoring

Just look at the score for each scale, that is of each mental skill and consider a score of 4 or less to be low, medium from 5 to 8, and high starting from 9 onwards. Remember to invert the interpretation for the scales of stress, attention and concentration as here a low value is preferable.

These results are important because they will be used to select the tools and techniques for training in the next chapter.

Chapter Summary

In this section of the book, we have explored the mental abilities that contribute to developers' performance by drawing upon theories that explain their role.

For each of these skills, you have observed the practical applications, that is the work situations to which they contribute the most.

You have also looked at some of the possible tools available for training and improvement.

In the hands-on part, you employed an assessment tool to evaluate your current skill level.

The next module will focus on how to train these mental skills.

The assessment results and the WIN-LOSE situation exercise, if you completed it, will aid you in selecting the skill to concentrate on. In fact, simply cross-referencing this information will reveal which skills have the most room for improvement. For instance, if the assessment indicates that the first three skills to improve are concentration, attention, and communication, and attention appears more often in the various LOSE situations, this final skill may be a good starting point for your training. If you haven't completed the WIN-LOSE situation exercise, no worry; just refer to the assessment alone.

Tools and Techniques

This part of the book deals with the tools, techniques and the strategies you can use to train mental skills.

- Tools
  We will explore the available techniques and we will focus on those that can be used independently, that is without the help of an expert.
- Procedures
  We will describe the procedure to practice and carry out the workout.
- Skill-tool mapping
  For convenience, at the end of the module we will provide a mapping between skills and tools in order to facilitate planning of the training.

Overview

In the previous module we dealt with the skills involved in the performance of the software developer and for each of them we have proposed some strategies for improvement. Each time we have analyzed the techniques commonly used in a structured mental training program, but also those that can be used directly by the developer without support from professionals, both in groups and alone.

We talked about **mindfulness meditation**, which is a type of meditation increasing emotional and cognitive centering, decreasing anxiety and reducing consumption of resources wasted to rumination and muscle tension.

Knowledge sharing which consists of active contribution to wikis and other systems for sharing knowledge.

Imagery, or visualization, which is the process of remembering or mentally building up detailed images.

The **routine** that is the execution of a small sequence of actions in the form of a mini-ritual.

The **Pomodoro technique** which involves alternating work and both short and longer breaks.

Mental experiments consist of trying to imagine additional solutions to an already known programming problem.

To **expand horizons** in a way of reading books, watching movies, listening to music and having experiences that provide you with new concepts.

The **code review** in which the developer and one or more colleagues examine the code in search of optimizations.

Cross-functional collaboration is a practice that stimulates communication between team members belonging to different company functions.

User Stories are an agile development technique that describes the desired behavior of software from the user's point of view.

Pair programming is when you write code with a colleague on the same computer.

Regular **physical activity**, possibly of aerobic type, such as twenty minutes of light running or power walking.

Biofeedback, in particular one of the measures of biofeedback which is HRV to measure the heart rate variability.

Autogenic training which is a body relaxation technique as well as Jacobson's **muscle relaxation**.

Breathing techniques such as diaphragmatic breathing.

Mental breaks which consist of interrupting work and doing something that does not involve the use of technology.

Exploration involves trying alone or in group in solving new problems that have never been managed before with the use of new design/architectural patterns-technologies-methodologies.

Prototyping is a technique in which a prototype is created to test functionality and usability of software.

The **daily stand-up** is a daily meeting where team members get together to discuss the previous day's progress and today's goals.

Figure 24: Overview of Tools and Techniques

Among these options we will choose the most immediate and available, meaning those that can be directly used by the developer independently without any external assistance or supervision. We will delve into the **Pomodoro Technique**, **Mental Experiments**, **Physical Activity** and **Knowledge Sharing** as these strategies address the entire spectrum of mental abilities. With data from the assessment and with the WIN-LOSE situations at hand, if you have completed them, you will be able to engage in a comprehensive “full-skill” training, akin to the “full-body” approach in sports.

Knowledge Sharing

Knowledge sharing refers to the process of transferring information, skills and experiences from one person to another. We are going to see in greater detail what knowledge sharing is, the systems that facilitate it, and why documenting one’s own knowledge can help developers improve.

In the context of software development, knowledge sharing involves the transmission of technical expertise, best practices, and solutions to known programming challenges.

Systems used to share knowledge

- **Wiki**
- **Forum**
- **Blog**
- **Platform**

Mental skills impacted: communication, collaboration

Figure 25: Knowledge Sharing

Numerous systems enable knowledge sharing. For instance, **wikis** allow users to create, modify, and organize pages containing technical information, documentation, and useful guides.

Discussion **forums** are a virtual space where you can ask questions, answer inquiries, exchange ideas and seek solutions.

Technical **blogs** serve as valuable tools for sharing experiences, novel discoveries, and tailored solutions through articles and tutorials.

Specific **collaboration platforms** such as GitHub, GitLab, or Bitbucket enable developers to collaborate on projects, share source code, and track changes.

Because knowledge sharing is both informative and cooperative, engaging in it also hones the mental skills of communication and collaboration.

Writing down own's knowledge requires organization of ideas in a coherent and structured manner. This process fosters the ability to communicate clearly and present information in a way that is accessible to others. When writing articles, tutorials or documentation, developers are encouraged to critically evaluate their thought processes, the challenges they face, and the solutions they implement. Sharing personal knowledge also requires collaboration where colleagues provide feedback, suggestions, alternatives and create an environment for mutual learning and exchange. Additionally, committing knowledge to writing allows developers to reflect on their own growth trajectory

and document their progress. This enables tracking gains, identifying areas for improvement, and testing oneself in unfamiliar settings.

In conclusion, knowledge sharing proves to be a powerful tool for developers as it promotes collaboration and communication at both individual and team levels.

Pomodoro Technique

The Pomodoro Technique is a method that can be used for training the mental skills of attention and concentration. In this section we will describe the technique, how to use it and why it helps the developer to stay focused on the task increasing productivity.

The Pomodoro Technique is a time management method developed in the ’80s by the Italian programmer Francesco Cirillo. The name of this technique comes from the fact that Cirillo used a kitchen timer with tomato shape in his days at the college. The main objective of the technique is to maximize productivity by focusing on accomplishing specific tasks in defined time intervals.

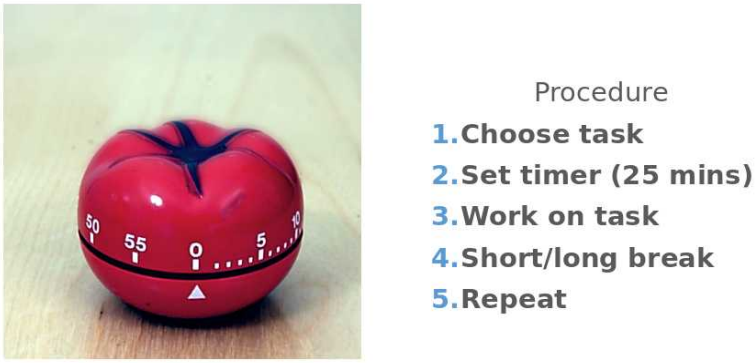

Figure 26: Pomodoro Technique

The Pomodoro Technique procedure is simple to carry on. You can use a simple kitchen timer, as Cirillo did, but also the one you have on the smartphone or from websites is fine.

1. You start by choosing a task you want to focus on. It is important to identify a clear and precise task so that it can be completed

in a defined period of time, which in the technique is called a "pomodoro".
2. Then you set the timer. Usually 25 minutes are used.
3. You start the timer and work on the activity. Here you need to fully devote yourself to the task at hand and minimize distractions such as phone calls, notifications, messages, emails, etc. It's time to focus all the attention on the task.
4. Once the 25-minute tomato has elapsed, mark off one pomodoro and take a break. This break will be 5 minutes if less than 4 checkmarks have been already written otherwise it will be longer, usually 15 to 30 minutes. After a short break you go back to point n. 2, while after a long break you proceed to the next step. The breaks are for stepping away from the screen, taking a short walk, or just relaxing. A short break here is intended to regenerate the mind to face the next tomato with greater concentration, while the long break marks the completion of the whole cycle.
5. After the long break you decide wether to start the procedure over by choosing another task or continue the previous one if not completed, then you set the timer again and so on. You can repeat this tomato-break sequence until your work or daily goal is complete.

The Pomodoro Technique is an effective tool for training the mental skills of attention and concentration independently. Choosing specific tasks, working concentrated for defined time intervals and taking breaks, helps to develop greater attention and concentration directly in daily work. In fact, the task to be chosen should be directly a work task, this way the results are maximized.

Mental Experiments

Mental experiments are a training tool for developing problem solving, creativity, and mental practice skills. They can be thought of as cognitive simulations we perform in our minds to solve problems or envision potential solutions to a given situation. In programming, mental experiments allow us to work on a specific problem by imagining additional solutions to a known problem without having to actually write code. This activity is essentially the bread and butter of programming since it is done regularly on the job. However, just as in sports, intentionally training a specific skill is more beneficial than only practicing it when necessary. In fact, "active" training is far more productive than "passive" learning done on the job. Of course the same principle applies to all the mental skills of a developer.

One possible procedure for practicing mental experiments is to try to imagine additional solutions to a known programming problem and

evaluate the pros and cons of each solution.

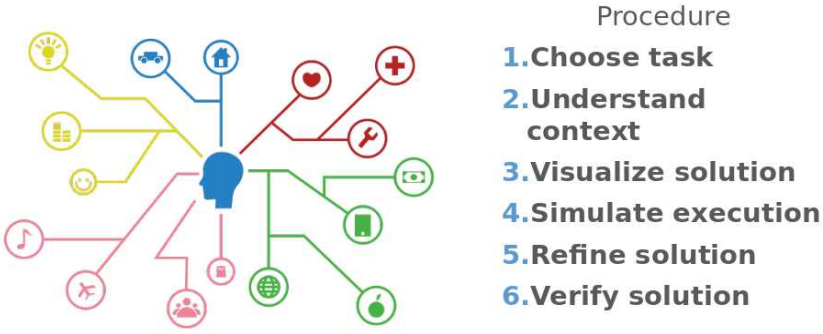

Figure 27: Mental Experiments

1. Start by choosing a coding problem you already know in an area you want to practice, e.g. sorting algorithms. Choose a small problem because the exercise is purely mental and you will have to retain all the information in your working memory.
2. Analyze the problem carefully, consider the specific constraints and requirements.
3. Imagine a possible approach to the solution. Visualize the program flow, the data involved, and the algorithms used.
4. Think through the program step-by-step, considering different situations and inputs. Try to predict potential outputs and errors, the same as if you were writing a unit test.
5. If you come across problems or errors in the mental simulation, change the imagined solution and repeat the visualization process
6. Once satisfied with the experiment, compare your imagined solution to an existing one. Analyze the differences and learn from mistakes.

This exercise allows you to explore solutions quickly and without writing real code or, at most, writing just pseudocode. It's a mental workout that improves your problem-analysis skills and makes you more capable of finding new solutions, which also benefits your creativity. Additionally, mental experiments are a good training tool for developing the ability of mental practice as they engage your mind in visual-

izing alternative scenarios, new paths, and possible outcomes.

Physical Activity

We are going to talk about physical activity as an aid to improve the developer's ability to cope with stress. Physical exercise has many beneficial effects on the body. Among these we are particularly interested in the production of endorphins, that are substances naturally produced by our body having the effect to improve mood and sooth anxiety. As a matter of fact endorphins are also known as *hormones of happiness*.

The World Health Organization recommends at least 60 minutes of physical activity per week for individuals aged 5 to 17, and at least 150 minutes of aerobic activity per week for those 18 and older. This recommendation is for moderate activity, which is any form of movement that increases your heart rate and breathing without reaching an intensity in which it becomes difficult to speak. Activities like walking, light jogging, swimming, yoga, or engaging in fitness class are good examples. It's important to distinguish physical exercise from actual sports, which typically involve high-intensity activity, organized competitions, and adherence to a set of rules.

Choose an activity you like

Allocate time for the activity

Use also short breaks from work

Mental skills impacted: stress management

Figure 28: Physical Activity

- Find an enjoyable activity: choose a form of exercise that genuinely piques your interest.

- Establish a suitable routine: Identify a time that fits comfortably into your schedule. Ending the workday can be an ideal time to release accumulated stress.
- Seize opportunities for movement: Even during work, incorporate brief breaks to engage in light activities like stretching. These breaks not only promote physical activity but also help prevent musculoskeletal issues associated with prolonged computer use.

Engaging in regular physical activity can help maintain stress levels under control. However, before starting physical excercise it is important to **consult your healthcare provider**, especially if you have a sedentary lifestyle or underlying health conditions. Once you get the green light you can start slowly. It is advisable to start with short sessions of physical activity, for example 10-15 minutes a day, and then gradually increase the duration and intensity over the following weeks.

The benefits of physical activity extend far beyond stress management. Mind and body are closely linked and send signals to each other via complex feedback mechanisms. It is the famous Latin proverb *mens sana in corpore sano*. For instance, aerobic exercise boosts blood flow rate delivering essential nutrients like oxygen, glucose, and minerals to the brain. This efficient blood supply provides neurons with the resources needed for neurotransmission, synaptogenesis, and cellular reconstruction. Consequently, physical activity helps cognitive functions and promotes the production of endorphins, further contributing to mood elevation and stress reduction.

Physical activity also offers purely physical benefits, such as better sleep, increased energy levels, and improved stamina. In summary, physical exercise is an invaluable practice for anyone, especially software developers, who often face cognitive-intensive work environments.

Skills-tools Mapping

After having reviewed some of the valuable methods for practice, we present a mapping between mental skills and training tools to simplify planning of daily workout.

The image below shows a list of the techniques explained in this chapter along with the skills they train better.

Mental skills are arranged across rows while training tools are found in columns. In this mapping the tool of **Physical Activity** helps with stress management. The **Pomodoro Technique** is a suitable approach to train attention and concentration. **Knowledge Sharing** is

	Physical Activity	Pomodoro Technique	Knowledge Sharing	Mental Experiments
Creativity				✓
Problem Solving				✓
Communication			✓	
Collaboration			✓	
Stress Management	✓			
Attention		✓		
Concentration		✓		
Mental Practice				✓

Figure 29: Skills-tools Mapping

beneficial for enhancing communication and collaboration. **Mental Experiments** are valuable for creativity, problem-solving, and mental practice. As we observed in the first part of this chapter, there are numerous tools for training mental skills, but the ones shown above are among the more practical as they can be employed entirely independently and without the guidance of an expert.

You can utilize this mapping as a reference to plan your training by considering the assessment results and the WIN-LOSE situation. If you haven't completed the latter, you can rely solely on the assessment. It is rational to focus on the mental ability that is most deficient in the assessment and that arises most frequently in LOSE situations.

Select only one skill to train at a time to avoid overloading yourself and to concentrate your efforts effectively. Once the skill has been identified, choose the tool designated for that skill from the mapping above and utilize it as long as you deem necessary. There is no predetermined minimum or maximum workout time; continue as long as you find it beneficial. Usually, one or more tangible changes should be observed during daily work activities, either directly by you or through feedback from colleagues.

Chapter Summary

In this module, we've explored various tools, techniques, and strategies for independently training mental abilities without external supervision.

For each tool, we've delved into its description, execution procedure, and the mechanism through which it impacts the mental skill chosen for training. In the module's concluding section, we examined a mapping between mental skills and tools to facilitate the planning of focused and personalized training.

The next chapter provides resources to deepen understanding of Men-

tal Training for Developers and continue training for those seeking to further elevate their work performance and raise the bar.

Wrap-up

In this book, you have been guided to enhance your performance as a Software Developer by refining your soft skills through a Mental Training program. We have delved into the concept of Mental Training for developers and how it has been tailored from traditional Mental Training in sports. Within the same section of the book, we also explored the effects of this practice on the developer's daily work and on the organization.

Next, we shifted our focus to the Mental Skills that are the subject of analysis and intervention. The skills we have examined are attention, concentration, problem-solving, creativity, communication, collaboration, stress management, and mental practice. Upon completing this review, we conducted an assessment of the current level of these mental skills, providing guidance on areas for improvement.

Finally, we reviewed various tools and techniques for independent training of mental skills without the need for external supervision. Here, we covered knowledge sharing, the Pomodoro Technique, mental experiments, and physical activity. At the end of this section, we also provided a mapping between mental skills and tools to support planning of individual workout.

The book ends up here. I hope you have found the content engaging and inspiring to pursue professional growth as software developer. As your coach I am always available for further information and clarifications. You can find my contacts *here* and drop me a message if you like, I will be happy to assist you in your efforts to improve.

I wish you a successful training and stay safe.

What's Next?

In this final part I want to provide you with some ideas on how to proceed and some resources to deepen your knowledge of both theoretical and practical aspects. Here are some suggestions.

Well, first of all, just keep training! As your coach I could not suggest otherwise. You have already started your training by going through this book... go on and stay on target! During the course you have identified the mental skill with the greatest room for improvement along with training techniques, so work on that by following the instructions provided. Focus on one skill at a time to maximize your results. As you know it's possible that a training technique impacts multiple areas simultaneously, as these are often interrelated, but in any case try to focus on one mental skill at a time.

This book provides all the information you need to achieve measurable results and enhance your performance as a software developer but remember that without consistent training there is no improvement. Unfortunately there is no quick fix or magic pill for changing your behavior or achieving results. As it happens in sports there is no shortcut; you need to dedicate yourself to training.

In addition you can explore further material. Unfortunately, at the moment of writing there is no specific literature on mental training for software developers available online. This is a relatively new and highly specialized field that combines classic mental training techniques used in sports with software engineering and computer programming. Therefore you may not find much information beyond this book but here is a short list of resources with interesting information, which you can take a look if you like.

- Coaching
  https://en.wikipedia.org/wiki/Coaching
- Soft-skills
  https://www.investopedia.com/terms/s/soft-skills.asp
- Mental Training
  https://mc-sportplus-blog.web.app/blog/mentaltraining/
- Mental Training for Developers
  https://mc-sportplus-blog.web.app/blog/mt4devs/
- Mental Skills
  https://www.flowperformancepsych.com/post/what-are-mental-skills-exactly
- Psychological Skills
  https://humanperformance.ie/psychological-skills/
- Computer programming
  https://en.wikipedia.org/wiki/Computer_programming

And then if you want to take action in a more impactuful way and maximize results, the suggestion I give you is to activate coaching as a live course. A live course is a path structured by an expert in which there is an in-depth analysis of the needs and a better selection of training techniques tailored to the individual's needs. Furthermore, live interaction with a coach makes it is possible to carry out continuous monitoring, stay on track and correct the course when necessary significantly saving time and energy.

But bear in mind, whatever you go for... enjoy! If while training you don't get satisfaction from what you do and the workout is just pain, simply change method. It's not a sign of defeat or disgrace, it simply means that the chosen strategy or tool is not for you. You can change the training technique, modality, planning, make smaller goals to make them more achievable, etc. But still, go for activities you like. Enjoying the process is essential for achieving the goal.

Bonus

In this additional section, I'd like to offer you a bonus as a coachee.

If you're ready to elevate your mental skills as a developer and further enhance your performance at work, then consider enrolling in a live coaching course. As mentioned, the results obtained in a live coaching are greater than in a book because more powerful tools and advanced techniques are used both in the diagnostic and in the training phase.

With the guidance of a coach with a strong background in both psychology and software development, it is possible to establish an expert-level interpersonal relationship that fosters rapid and lasting growth. On average, developer coaching involves 10 sessions during which the coach closely mentors and supports the individual throughout their journey of professional development.

To take advantage of this offer drop an email and show proof of purchase of this book to receive a special discount on your live coaching for software developers:

- MC Sport+ https://mc-sportplus.web.app/.

References

- Attention, Emotions, and Behavior. Nideffer, R., 2017
- Flow: The Psychology of Optimal Experience. Csikszentmihalyi, M., 1990
- The Stress of Life. Seiie, H, 1950
- The Evolving Self: A Psychology for the Third Millennium. Csikszentmihalyi, M., 1994
- The Problem Space Model. Simon, H. and Newell, A., 1974
- The Mathematical Theory of Communication. Shannon, C., E., 1948
- The Psychology of Management. Gilbreth, E., L., 1946
- Psycho-Cybernetics, A New Way to Get More Living Out of Life. Maltz, M., 1989