Towards a sustainable remote-first team environment: one pairing session at a time[1]

IGOR FORMIGA ALVES, YURI DO NASCIMENTO FARIAS DA SILVA, MARCO MONDINI, OLEKSII FEDOROV, FRANCISCO BRICENO, YIGIT TANRIVERDI, RIDE Capital

Everyone wants to work in a healthy and safe environment, where you're free and comfortable to share your ideas and feel valued and heard. We found pair programming as the glue to build a culture of trust within a remote environment. With a little initial investment, this XP practice can be exploited for successful onboarding and establishing sustainable collaboration among team members.

1. INTRODUCTION

Not just "working on the same code". Pair programming (Beck 2000) is the core activity in the engineering team at RIDE, though talking about the single pairing sessions, while the world experienced hands-on the need for remote collaboration, could sound a bit diminishing. Developers jumping on and off calls, missed meeting notifications, and virtual colleagues like GitHub Copilot (2021) providing suggestions on code implementation, testing, and even comments.

We will set the virtual assistant aside for this time. This is going to be a different story. We want to focus on how we are embracing the core values of XP (Beck 2000) and applying them in a remote-first environment for intra-/inter-team collaboration. We will go through the main challenges we faced and lessons learned along the way of developing in a fast-paced environment, under business pressure with a focus on scaling the teams without sacrificing values, quality, and happiness. Furthermore, we will focus on pair programming, since it represents for us the key practice to build a sustainable team environment.

2. TIMELINE AND TEAM PROCESSES

RIDE is an early stage startup navigating in the agitated waters of the Berlin FinTech sea. The tasks of our tech team are mostly directed into building a multi-user platform with different automation services, from bureaucratic processes for the foundation and administration of companies, to trades bookkeeping and taxation filing, on top of all internal toolings.

Our journey starts around the end of 2020, from Berlin, during the second wave of Covid-19. Alex is leading the team and Marco just joined, we are a single small engineering team of about 5 people. A colleague of ours was already working remotely from a different timezone, while it took only a few weeks before everyone started working from home.

By the beginning of Spring 2021, we had grown to two remote-first teams, and half of the developers were working from abroad. The time zone count reached 4 from IST to BRT, spanning more than 8 hours. Igor joined from São Paulo, while Francisco and Yigit from Berlin. The team continued to grow and by the end of the year, we established a third team.

During the current winter of 2022, a batch of new hires joined, including Yuri and most of the employees working from abroad. At this point, the developers doing some hybrid work are officially in the minority. With such structure, depending on each iteration, the pairing sessions cover roughly 50

[1]Author's address: Igor Formiga Alves, Marilia de Dirceu 300, 04632-030 São Paulo - SP, Brazil; email: igorformigaa@gmail.com
Second author's address: Yuri do Nascimento Farias da Silva, Rua do Cupim 132, Graças, 52011-070, P.O. Box: 543, Recife - PE, Brazil; email: yurinascimentofarias@gmail.com
Third author's address: Marco Mondini, Kopenhagener Str. 66, 10437 Berlin, Germany; email: mmondini@mondspace.com
Fourth author's address: Oleksii Fedorov, Friedrichstr. 123, 10117 Berlin, Germany; email: hello@fedorovalex.com
Fifth author's address: Francisco Briceno, Pauline-staegemann-str 4, 10249 Berlin, Germany; email: fbricenho@gmail.com
Sixth author's address: Yigit Tanriverdi, Aronsstrasse 73, 12057 Berlin, Germany; email: yigittanriverdi@yandex.com

to 90% of the overall working time. The rest would be dedicated to individual work or to the usual rituals, like weekly iteration planning meetings, retrospectives, 1:1s, and learning activities such as book clubs, coding dojos. This process goes hand in hand with a Test-driven development (Beck 2003) and trunk-based-development approach applied by each team member.

3. BENEFITS OF PAIR PROGRAMMING

As we know, pairing comes with a lot of great outcomes. It works as an instrument for knowledge sharing that can empower you to better support your peers. One example is when doing onboarding, applying the training wheels approach, which gives you the tools to embrace the new joiner into your codebase and practices, since you can contribute and co-create, that way sharing the responsibility on quality patterns and main implementation decisions. In our case, we relied upon pair-programming to avoid verbose documentation and keep a flat hierarchy, caring about processes and tools while valuing more individuals and interactions (Beck et al. 2001).

The team had linear and constant growth since the beginning, without big jumps. This was both a choice and effect of the processes in place. An onboarding based on pairing and interactions can be expensive in the beginning, especially for developers not used to applying such practices on a daily basis. As a result, it becomes a constraint to the capacity of the onboarding itself. Within the boundaries of our experience and in a context where such a steady growth is allowed by the business, pair programming enables us to build constructive interactions among team members, leading to trust and reciprocal support. Thus, fostering the autonomy of the team and mastering a psychologically safe environment (Buvik & Tkalich 2021). Finally, another motivation for such practices is to foster collective code ownership from day one and maintain the energized work (Beck 2000). We decided that the benefits were worth the challenge, especially when applied in a remote environment.

4. WHY PAIRING WITH REMOTE-FIRST TEAMS

Remote and hybrid work came into the scene particularly in the last years with huge efforts for businesses to quickly shift to remote-friendly processes. Most of the professions related to software development started from already a solid ground of autonomous work and could benefit from the pre-existing transparency of the processes, control versioning systems, and daily meetings to mention a couple. Despite that, this new scenario deeply affected the collaborations between individuals. The environment where we used to go every day disappeared and grew into a virtual world of calls and instant messaging.

At RIDE, we were starting to build the ground for growing from an engineering team to a department with multiple autonomous teams. How could we provide the best experience to new joiners? How to avoid your teammates from becoming just a nickname next to a commit message? Where did we lose all those blackboard discussions? Our team already adopted XP practices before moving away from the office. Furthermore, many of these practices provide a safe environment where you learn to navigate in extreme conditions and develop your personal tool-set for every situation. Such an environment allowed us to easily shift into this purely digital way of collaboration. In particular, the habit to spend most of our coding time pairing with a colleague developed into the current habit of quickly jumping on and off calls and screen-sharing thanks to an already established “virtual personal space”.

Looking back at such transitions, we identify four key benefits which represent for us the highlights of pair programming with regard to the effect on each individual and the overall team dynamics.

4.1 Fostering information flow while balancing out the lack of “corridor talks”

Back when we started to push the transition from hybrid to the remote-first team, we faced the classic work from home challenges and started missing out on the discussions rising from coffee or fresh air breaks. Initially, we felt the pressure to rush into tools like “virtual rooms”, “digital” coffee breaks, and open spaces. Though, we could soon realize that it was not a good fit with the daily activity, and was actually mostly getting in the way. While on one side many tools lacked features for smooth collaboration, on the other we still needed to get our own coffee breaks, and we would rarely do them in front of a screen. Such a situation pushed us away from adopting new tools, and we decided to better exploit the processes in place instead of coming up with new ones.

Nowadays, we use Microsoft Teams for the main communication between our tech teams, a general channel for our engineers, as well as other departments' channels. All these channels are visible and accessible to people from different departments. With such a setup, we strive to make it easy to ask and answer questions, as well as have transparency between projects as to what is going on in each sector of the company. Biweekly, we hold architecture meetings with an open agenda and defined by anyone in the engineering team who would like to propose a topic, as well as more business-oriented All Hands meetings, with occasional demos of the latest features delivered. Sure enough, this only works if everyone is attentive to new posts so that if we can't solve a request ourselves, we can always tag someone to help us find a solution asynchronously or to jump on a call.

While all the main communication can go through such channels and other project management tools, the daily talks, and the usual morning chit-chat are not easily shared and would get lost in the way. In the engineering department, we are following company habits, though on top of that we exploit pairing sessions to overcome such informal communication loss. How do we do that? We keep the pairing sessions on-point. This means we would be focusing, coding, and completing the tasks, but as with any efficient session, we need to keep an eye on breaks. Since everyone has different levels of attention and, thus, would keep a coding session running longer or shorter than others, the challenge would be to find the best balance. Usually, as in any relationship, each pair would find that with time.

What happens during such breaks? Often in our experience, especially in the beginning, the pair would turn off the call and come back after a few minutes. However, with time, we noticed the important difference between breaks from the tasks against breaks from the actual 1:1 interactions. We experienced that most of the personal bonding among teammates starts from conversations during such breaks, where the pair would not go away from the laptop but keep talking about different topics, from food to travel, to personal life gossip.

While breaks can foster opportunities for such informal conversations, when we actually need quick feedback or help because of any bottleneck, we can rely on calls. Within a team, used to spending most of the working day pairing and being on call collaborating with one or more teammates, it does not cost much extra energy. Finally, it comes natural, even for developers, to briefly jump on and off pairing sessions, when someone requires help in a different context. This happens without sacrificing the ongoing task, since you would be leaving that in the good hands of your pair.

4.2 Incepting feeling of belonging

Our target is to have you join RIDE on Monday, and by Tuesday you'll have already pushed some code up to production. We are not there yet. What we achieved in the latest months is to have new joiners being able to contribute to a task and also participate in a release within the first week. That's because if on one side any team member would welcome you and begin to code together in the first hour, every person is different and someone might prefer a slower start. In order to achieve this, we give space to set up your environment, or we try to facilitate with some ready-to-use presets. Having everything working could take some time, especially if devs are not familiar with the stack. That's when pairing comes in handy, since we can always share IDEs and work alongside our pair. In the first weeks, usually, the developer will be navigated most of the time by more experienced teammates, so that can have an early hands-on experience with the code while not being stressed by glitchy setups. On another note, this slow onboarding process is less intrusive for the existing team, whose velocity would not be extremely affected given the opportunity to rotate the pair of the new joiner. Other than this, we have more resources in place to effectively support the pairing time. At RIDE, new joiners are assigned an "onboarding buddy". Someone who will be the new joiner's closest point of contact during the first month or so, to help them solve any questions concerning processes, our ways of working, the attitude we seek to foster in our development team, and finally to help them find who to reach for any business-related question. Naturally, some onboarding tasks, readings or administrative chores can be done alone. But we strive to always make ourselves available and check on our new team members, so they do not feel disregarded or left out. We also find this creates a strong rapport among team members.

By having someone to navigate them through their first weeks at RIDE, new members feel taken into account, and they want to contribute as soon as possible. Pairing gives them the possibility to work closely with all the team early in the process, and to start participating actively in the usual agile rituals like retrospectives. Building a connection with the new joiner at a personal level that is not based on

the work done, it's extremely important for us. A high level of collaboration requires trust and a healthy environment where the team lies on the same page.

4.3 Career growth in the organisation: transparency of results and improvements

Even in XP teams, software developers are supposed to master autonomous work. Nevertheless, the pairing environment creates a situation where feedback on the work quality would be given naturally on a daily basis. Such a time frame could differ a lot in organisations where a colleague would jump hands-on on your code only after the task is ready for delivery, from multiple days to multiple weeks. Once shifting to a remote-first environment, such little discussions and insights on your colleague's work would decrease drastically, pushing even more against the current of a continuous feedback culture.

In our experience, the discussions rising or non-rising from your pair feedback allow the developers to understand their mates better by giving on one hand a baseline where to draw expectations while collaborating on the same project and tasks and on the other hand the possibility of more constructive and personal feedback in the occasion of 1:1s and performance reviews. Pair-programming allowed us to incept and nurture relationships with colleagues which we never had the chance to meet in person. So, now imagine your manager asking you what your colleague could improve, and the only thing you could start thinking about is that nice database migration, that annoying bug or a pipeline failure you had to fix, or any other improvement related to code, processes and artefacts in use. Clearly, it could not be constructive feedback for your colleague, since it would be missing out on all those personal traits that you could have drawn from close collaboration and empathic behaviours.

In our team, we strive to embrace a culture of collective growth, where each individual contributes to the growth of her team and mates. We found this attitude extremely helpful in order to naturally track and show personal career growth, while it also allows us to quickly spot outliers and behaviours that are not in line with the team culture and could endanger the relationships between colleagues.

4.4 Flexible time with limited constraints

In this remote work world, we started having the opportunity to work with colleagues in different time zones. This can bring problems, but it can also bring benefits. The greatest benefit of having different time zones is that not only you can have someone that can spot any mistakes while the other part of the engineering team is offline, but they can also fix it and ship it. This caused us a great sensation that we don't have to put in extra hours to work on urgent tickets or bugs. Because of the way we work, and pair, we can share the context with our peers, working a few hours behind the local time, and they can continue the work you were doing without having to sacrifice the team's health.

The challenge we faced was to combine the onboarding by pairing with a time zone difference. So, we decided to limit the time difference so that no team member would be left alone for a whole day, and each one would have at least 4 to 5 hours of pairing opportunities. This would imply flexibility and collaboration in deciding the working time. For example, Igor loves working in the morning and would start at 6 am/7 am in BRT while currently in the Berlin office it would be 11 am/12 am, establishing a 6 hours slot for pairing, and meetings.

5. REMOTE PAIR-PAIRING PATTERNS

In a methodology with no silver bullets, pair programming ain't the exception. When applied by people that would not even meet in person, the amount of possible pitfalls can only rise. Thus, to balance its benefits, we want to share a series of challenges we faced during the last years while building remote engineering teams. In an agile team, these challenges translate to patterns that each member should look closely while engaging in pairing sessions.

5.1 Take a damn break!

Forcefully working from home has eliminated the possibility to grab a coffee with a colleague and discuss ideas in the hallways, or even know how the other person is doing outside of work, which offered us a very much needed mental break. With that, we tend to get so caught up in work that we skip breaks altogether. But the breaks are still necessary. To avoid getting tired, which can influence our mood and decision-making abilities, it is always important to be mindful about taking breaks. And breaks are not

the same for everybody. For some people, simply switching tabs and watching a funny video can do the trick. Others need to completely get off the chair and walk around a little. It's up to each person to know what works best for them and make that clear, so the pair can adjust accordingly.

When the pair is not familiar with any time-boxing technique, we propose iterations of 25 minutes focus time, alternated with breaks as per the Pomodoro Technique (Cirillo 2006). However, it would be up to the first sessions of each pair to provide feedback to each other to reach the desired balance. To allow smooth progress of the sessions, we keep a close on the total amount of focus time available. That is, we periodically rearrange weekly and daily meetings' schedules to minimise the amount of planned interrupts. Currently, each team member has one or more 4-hours slots of possible uninterrupted work daily. When such a slot reduces to 3-2 hours because of unplanned meetings rising, we take it as a red flag and collectively find a better schedule.

5.2 That exciting never-ending refactoring

While refactoring is at the core of XP, it can be tempting to spend a whole sprint refactoring that cringy component, and then fixing all the places that use it, and "oh, look! Another refactoring opportunity", and before we realise we've burned all the time allotted for the card, going down the rabbit hole of the never-ending refactoring. As much as we like to refactor code, there is a fine line between healthy refactoring and damaging our pace to fix all the problems in the codebase. We also rely on our pairs to keep us on track so that we agree on the scope of the refactoring. This is not to say that long refactorings don't happen sporadically, but bigger undertakings are always brought to the table in our biweekly architecture meetings so that we can collectively plan and execute the needed changes in the system.

5.3 Slow Onboarding requires being more careful with new joiners' feedback

When adopting a hands-on first approach for the onboarding process, developers would absorb knowledge of the system and of the business facets more slowly. Despite indigestion is not something too pleasant, choosing a longer road can raise some hard conversations from a management perspective. It is important for us to define in the beginning what are the expectations for new joiners and follow them up closely. A possible paradox would consist of a new joiner becoming productive late in the perspective of the other employees and managers. That is why it is required to have a base culture of transparency and continuous feedback.

Good and open communication channels must be established from day one, in order to avoid a situation where the progress of new joiners is not praised as it deserves. On the other hand, the same relevance goes on looking for non-fit of the new joiners. Since, when pushing such practice to the extreme, a bad fit or bad employee behaviour could wrongly be interpreted as justified slow onboarding. Communication during pairing sessions and early feedback by other pairs during 1:1s play a critical role in this scenario.

5.4 When the camera is off

A huge deal of communication is non-verbal. Not sharing the same physical space takes away some of that, but having our cameras open during pairing sessions and meetings can make up for at least some of those lost cues, as well as improve the feeling that we are communicating with another person, not just a square on a piece of software. Having the pair invested in the activity is essential to the success of pair programming (just-in-time code review, refinements, attention to TDD principles, refactoring, taking breaks). When my pair still needs some clarification on that last block of code I wrote, a confused face might be all I need to dive a little more behind my reasoning. Or a squint can let me know my font size might be too small, making it harder for my partner to read what's on my screen; in which case I should ask them if they would like me to increase my font size or zoom in. There might be days when a person with a sleepy face would rather keep their camera off for the first hours in the morning - and that is fine - but we try the most we can to have our cameras open during pairing sessions.

A big part of communication is lost on remote teams that never open their cameras. Body language is a crucial part of it. If I can't see the way you're reacting to what I say and vice versa, that can lead to wrong assumptions based on the existing feedback (That can be just an "OK" or just a silent crowd). So, you can either create tools to go around this problem and be okay not being able to see your co-workers

when working together, or you can open your camera and make sure that you have a presence on the calls even when you don't necessarily have much to say.

When we have meetings, especially the ones that have more than 4 people, we start to realise how much visual contact is necessary. Not always you can spot if you're going to cut off someone when you can't see them. This affects how everybody there will interact with each other. When you can see everyone, you have instant feedback on how much the participants are involved in what's being discussed. This is not something that needs to be shared verbally but can help a lot when deciding on where to steer the agenda to keep engagement.

6. PAIRING BUDDY RED FLAGS

A good pair with not enough pairing experience could fall for some of the challenges we explained. However, going deeper to the roots of what builds an effective and efficient pair, we identified a series of patterns that would put at risk a healthy pairing session. In our experience, these behaviours when persevered are the first symptoms of a possible rotten apple and a disequilibrium in the team. Thus, these topics are common in our retrospective agenda and always watched closely.

6.1 Bad communication

Pairing is communicating, and communication has protocols. There is a message, there is a tone to convey that message, there is a receiver, a sender, there is a time to talk and a time to listen. When we are coding, we are putting our thought process out there, and that needs to be understood and refined by our pair. It is not nice when one person just spits out some code they think is good enough without developing a thought process and an action plan with their partner, either to get validation or to be challenged in order to reach something better. Thinking out loud is great to let your partner know what you are thinking and where you are going. It also makes them feel heard and considered, and gives them the chance to pinch in, building on top of your initial ideas.

It's important to realise when the pair is getting frustrated for not moving forward, or when one starts to talk on top of the other and disagreement spawns. This might indicate that it's time for a break.

Apart from that aspect of communication, there is also a bad internet connection, different time zones, a messaging software that does not always do what we want or need, and those other couple of meetings. Communicating that to your pair to come up with a plan, so nobody is blocked waiting on the other is extremely important for the good pace of work, and it also shows respect and consideration for your pair. If you need to leave the pairing session, let your pair know for how long it will be, how they can move forward without you and what to do should they get stuck for some reason.

6.2 Large commit batches

Sometimes we can consider that if we're not able to see small commits on a pair's ticket, that can be a red flag for what's going on in the session. When we're pairing with someone, we usually expect to see small commits, because it shows that we're rotating the driver navigator role in a way that doesn't burn out one or another for staying too long in one of the roles. In our experience, this was always a sign of things going wrong with most sessions. Because we thought that since we were in a hurry it would be best for someone to stay hours driving, sharing their screen, while you try to quickly navigate. We were always more prone to error than if we went for a more role-dynamic way of pairing.

6.3 Passive pairing

There can be two layers to passive pairing. One is when the partner does not really pinch in with suggestions or take the first option offered. Whether it is a variable or method to name or a solution to adopt, it can be discouraging to realise you are pairing alone. Even worse is when the pair is only physically there, but silent.

While there can be many reasons for this to happen, communication is key to improving the pairing session. "I would like to hear you more" can be a good encouragement for a silent or lenient partner to maybe change their attitude towards the session. Perhaps it is that the partner is tired, or having a tough day. "I noticed you were more silent today. Is everything alright?" can open the channel for the other to speak about it (if they are comfortable doing so, of course).

Keeping the cameras open can be a good action item to tackle this issue. A smile or a frowning face can say a lot about our partner's engagement and help us act in order to take more out of the pairing session, so it can be enjoyable for everyone.

6.4 Close-minded: staying on the defensive and diminishing other people's ideas

There is no way to pair with someone that doesn't trust your abilities and opinions. A close-minded developer can significantly hurt a team when you're trying to build a collaborative environment. One of the biggest advantages of choosing to use pairing in your team is the fact that you are always allowing the developers to share knowledge, not only about the codebase but also technically. But when developers start to feel afraid to share their ideas and to validate that because of someone's behaviour in the sessions, this becomes a huge problem.

Some signs that can show you that a developer had this experience in the team and is hurt by it, is when usually you had a very active partner in the sessions before, but now when you're pairing with this person it almost seems like the person out of nowhere started to do a lot of passive pairing. Which again we consider as a huge red flag for you to watch out for the health of the pairing sessions and the team.

6.5 Teacher paradox: a driver that loves too much to navigate

When you have a driver that loves too much to navigate, the feeling of your peer is that none of his ideas has any worth, and that also he's not needed there. When we had that, some engineers just stopped wanting to argue anything with a coworker that acts this way, and shield themselves behind a wall of non-communication.

The worst is that this jeopardises the trust of the team, the knowledge sharing and especially the feeling of shared ownership of the code, since it leads to developers not wanting to touch the code that this colleague has written, not willing to give constructive feedback, and in the worst scenario it might be the reason why someone would look to leave your company and team.

The way we handled it was by trying to catch this kind of persona in our pairing interviews. Trying to give opportunities to candidates to showcase as they deal with different opinions in the session that affects the next steps of the exercise.

6.6 Stalling: not switching driver/navigator for too long

The most used pairing style at RIDE is driver/navigator, where the former inputs the code and the latter leads the session. A common smell we find during pairing sessions is when the driver role would not change for too long. There can be many reasons for this to happen, and they most likely are not what we want for a healthy collaboration. If both developers are eager to work "hands-on", one can feel frustrated for not having the opportunity to more freely navigate the code and express their ideas. In a remote pairing session, the screen sharing software also plays an important part in how easily it allows developers who are navigating to point and highlight certain areas of the screen (if at all).

On the other hand, being the navigator for too long can be boring, causing the navigator to disengage from the activity. Driving and navigating require developers to take different stands concerning the code and the things they are aware of. While the driver is more focused on what's being typed at the moment and the code immediately around where they are looking, the navigator has a broader, more holistic view of the open tabs, things left to do, and the next test to write, etc. By switching driver-navigator, we exercise two different positions with different cognitive loads, perks, and limitations. It is less tiresome for the pair when we switch driver-navigator at a healthy frequency, maybe after every 25 minutes, which also forces us to make smaller commits to get the code out of our machines and available for the whole team.

It also takes some self-policing. "I've been driving for too long, let's commit this after this test is passing" or "can I drive for the next test?" are common to hear in our day-to-day work at RIDE.

7. CONCLUSIONS

Remember to have fun while delivering high quality. Our statement affirms that the values and benefits of pairing are worth the ride. The intensity and the extent where such practices can be applied will possibly evolve and improve in the future. Going beyond the boundaries of the software development

life-cycle, we apply Test-Driven development and Pair-Programming in our hiring process too. Allowing us to have a nice balance between technical interviews and team-cultural fit.

Pairing does not come as a free lunch, it requires the team to keep a high level of professionality since pitfalls and distractions are all around the corner. Though, it does come with great benefits with regard to team communication, engagement, trust, and flexibility. It becomes a great tool to balance out the reduced interactions for hybrid and remote-first teams who strive for continuous value delivery. It helps to establish autonomous and self-managed teams aligned with the business goals. With our experience, we aim to also highlight the challenges we had to tackle while adopting pair-programming. Finally, we identify a series of patterns that in our opinion should be watched closely when building a team within this context, since timing is important. Practice and incremental improvements can be good allies for teams who would like to give a chance to such processes.

8. ACKNOWLEDGEMENTS

We'd like to thank our colleagues for their advice and support along this journey. We'd especially wanted to thank our employer RIDE Capital for giving us the chance to build such a healthy and constructive environment and to share our own experience. Last, but not least, *thank you Ademar!* for guiding us to shape this report and the precious feedback.

REFERENCES

Beck K. Extreme programming explained: embrace change. addison-wesley professional; 2000.
K. Beck., M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries and J. Kern. The agile manifesto; 2001.
Beck K. Test-driven development: by example. Addison-Wesley Professional; 2003.
Buvik MP, Tkalich A. Psychological safety in agile software development teams: Work design antecedents and performance consequences. arXiv preprint arXiv:2109.15034 Sep 2021.
Cirillo F. The pomodoro technique (the pomodoro). Agile Processes in Software Engineering and. 2006;54(2):35.
GitHub Inc. © 2021. GitHub Copilot, visited 14 February 2022. <https://copilot.github.com/>