MAKERS

7 WAYS TO ACCELERATE JUNIORS TO MID-LEVEL DEVELOPERS IN 12 MONTHS

makers.tech

Introduction

How do you get the most out of your tech talent?

Here at Makers we help leading organisations to answer this question because we know that taking on junior developers is not easy - especially if you haven't done it before.

Even if you do have experience of recruiting computer science graduates fresh out of university, the impact they'll be able to make on your organisation will broadly depend on the learning and development plan you design for them.

By asking yourself and your team the right questions, you can get your junior developers to progress to mid-level developer ('mids') within 12 months of starting.

To clarify, these are the kinds of things we mean when we talk about what a 'mid' would be capable of doing:

- Working on large features without direction;
- Need very little hand-holding from their manager;
- Be able to navigate the organization effectively;
- Sort out problems without needing a manager's help.

Every organisation has a different definition of what a 'mid' is. You should make yours clear to any juniors you take on so they have a tangible goal to aim for.

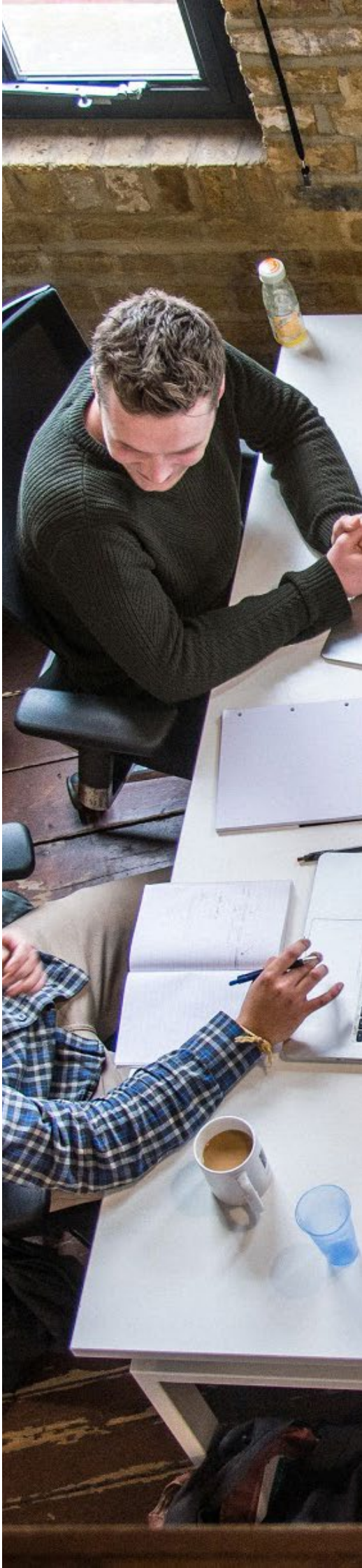
Contents

In the following chapters, we'll cover:

If you'd like to learn more about how Makers can help you to make the most of your tech talent, please feel free to contact us on **hire@makers.tech**

Thanks,

The Makers team



Decoding the ever-moving finish line

At Makers we talk to many of our hiring partners about the notion of "mid-level developer". This is a purposefully vague goal for where we expect our developers to be after a year, simply because what "mid-level" means is different for every organisation. But it's helpful for everyone involved to have some kind of goal to work towards.

To take an example, mid-level may mean that the developer can be independently successful. They can plan and manage their own work, they are a valued team member, and they can pick up relatively large tasks.

For any junior developer being brought on, a useful exercise is to break down what mid-level means to them. At a practical level, this boils down to setting some learning goals for the coming year.

Once the goals are in place, the next challenge is being able to accurately self-assess. That's a skill in itself, and we coach our students in how to do that.

Makers developers use a tool called SkillsMap that helps them evaluate at regular intervals, usually every 2 months. Both the Makers coaching team and our hiring partners have access to this system to help set monthly objectives and evaluate what new skills have progressed.

Accurate self-assessment relies heavily not just on self-awareness but also on getting timely feedback from managers and teammates. Makers itself has a strong feedback culture and our graduates will have experienced this from day one.

This is one area we see clear difference from computer science graduates. Our developers will be looking for feedback on a regular basis so that they can course-correct.

Hiring partners can take advantage of that by upping the amount of feedback they would normally give. A weekly check-in where you can cover what your Maker did well and what they didn't do well will go a long way.

Action:

Set some learning goals for the coming year. Discuss with your developer:

- Where do they want to be in a year's time?
- What technical skills do they want to master?
- Do they want to start specialising in one area?
- What skills are important for them to learn to be successful in their role?

Optimising Onboarding

It goes without saying that the initial onboarding experience should be as smooth as possible, for everyone involved. Your team may worry that the new hires will be a drain on their time, but did you know that new devs dread being a burden just as much?

The feedback from new developers at one of our hiring partners was that:

It felt that time was ”wasted” in the first month (e.g. observing seniors coding or in meetings) instead of starting to work on the stack.

The first month was generally super overwhelming mostly because it wasn’t clear what developers needed to learn and they often were “feeling useless”, “seating being paid” and had “depressing days” questioning whether they will be able to learn all the tech that they’d need to work with.

The solution is to set the right context and be super clear with expectations from the get-go, even before their first day in the office.

There’s a lot you can do before your team member even joins. Set the right culture for them to fit into and you’ll start your new hires on the right foot.

Action:
Create a great onboarding experience for your developer:

- During the interviews, give clear information about the the role, the stack and the team they’ll be joining.
- Set work to prepare before starting, in the relevant stack from the very start, preferably through a project.
- Make people feel useful from the get-go. An ideal onboarding would include lots of coding and more solo work on a project from the very start.
- Make expectations crystal clear. What do you expect your new hires to achieve in the first month, 3 months, 6 months?

Overcoming Imposter Syndrome

Many new developers find themselves feeling out-of-place when they first enter a real working environment.

They will likely have the least amount of technical training of everyone on their teams, and may find themselves working with people who are biased against junior developers.

This kind of attitude can often dent the confidence of new developers, leaving them feeling isolated and fearful.

If that wasn't bad enough, new developers can quickly become overwhelmed by the sheer complexity of organizational IT systems. When they complete their training, they will leave feeling like they know a lot. But that feeling won't last long.

Joining their new organization, they will be bombarded with dozens, if not hundreds, of technologies, concepts, buzzwords, acronyms, architectural patterns, and so on. For the new starter, it can feel like they know less than they did before began!

Relatively speaking, this is true: they are suddenly aware of how vast the scope of learning is. For those completing an intensive training like Makers, their 3 months has barely scratched the surface. However, they will benefit from having the right habits for continuing to accelerate their learning.

What's more, degree graduates are more than likely in the same position: most computer science theory is irrelevant to much of the standard daily work that programmers must do.

Degree graduates, however, often have a support network and a sense of entitlement that comes from simply traversing the expected, typical route into any organization.

Action:
Recognise these issues and discuss them with your new hires to alleviate their concerns.

- Imposter syndrome can be averted by letting your developers know that you understand their position and reassuring them when they are feeling overwhelmed.
- As an employer you can help them overcome this by cultivating an environment where it's okay to ask questions - make this clear to them.
- Have one person on your team that juniors know they can always ask - think about assigning that role to someone.

Compassionate social coding

There are many aspects of programming that require a decent amount of etiquette if the team wants to survive in the long term. Without some level of human decency, team dynamics can often fall apart, which results in unproductive programmers.

Pull requests (PRs) are one example of this. They invite unfiltered personal commentary on the work of other people.

It's human nature to believe "this isn't the way I'd do it; it'd be better to do it this other way instead". This is undoubtedly true for some definition of better, but to give this suggestion to a fellow programmer is to trash their creative output, which is a hurtful experience.

So, sometimes it's better to just leave things as they are. It's fascinating how many senior developers fail to grasp this simple concept.

Of course, some practices, like pairing, mobbing and test-driven development, remove much of the need for pull requests, either partially or entirely.

Action:
When it comes to pull requests, make your developer aware of traps like nit-picking, micro-managing and negative language which can be demoralising to the original author. This is particularly true when suggestions are around style. Help them to recognise the potential influence of their behaviour.

Understanding architecture

Nowadays, serverless architecture is fashionable and microservices are a given, but the journey of how we got here is not well understood. A historical perspective on how we got here can be very beneficial to all developers.

Unfortunately, this is a blind spot for many new developers, who can really only scratch the surface of software architecture, by understanding what it is they are building.

At Makers this means web applications with a simple front and back end split, and some knowledge of the HTTP protocol. Diving into other communication protocols and microservice architecture will surely be one of the first things they do when they land their first job.

Employers can help them along. One option is to encourage your graduates to create their own hand-drawn diagrams of the organisation's architecture. This can be an illuminating (and fun!) group exercise that will help them ask the right questions of each other, so that they can fill in each other's knowledge gaps.

Action:
Ask a knowledgeable team member to run a 30-minute lunchtime session to describe how and why the system is built the way it is. (Just bear in mind that a fresh graduate is unlikely to pick up everything in this session, and it will be worth revisiting the same topic once or twice more throughout their first year.)

Navigating the technical landscape

Speaking of diagramming, it really is a wonderful technique for understanding all sorts of aspects of our work. In particular, codebases lend themselves very well to diagrams: mind maps, flow charts and UML.

New developers are unlikely to have worked with UML because the systems they built during their training will have been comparatively simple. But after a time at your organization, it's worth educating them how to draw class diagrams and interactions diagrams.

They are also unlikely to have fully understood the power of the IDE. Many of our hiring partners use IntelliJ or Visual Studio, but Makers developers won't have used these IDEs during the course.

Debugging support, for example, is something they will have had little practice with. That's partly down to use of TDD which mitigates the need for debugging. About the only debugging they'll have done is print-line debugging. But they will benefit from understanding breakpoints, watch variables, and so on.

Action:
Give your developer the language of UML and suddenly they will be able to articulate their own designs.

Coding better

At Makers we give our developers a broad understanding of "good" programming principles: DRY, SOLID, patterns, and so on. But these are topics which really benefit from revisiting many times throughout their career.

The single greatest thing you can do to help your graduates write better code is to give them a lot of code to write. Practice makes perfect.

Unfortunately, very often we see hiring partners limit their graduates to small pieces of work. While this is only natural, there's a danger that these pieces of work trickle in, leaving the graduate to write only a few lines of code each day.

There's no reason why this should be the case. For all the bigger tasks that you are unwilling to give fresh graduates, there are undoubtedly smaller tasks that you can carve out of these tasks.

But that's easier said than done:
Interrupting a developer's flow in order to ask them to re-plan work is likely to be met with a lukewarm response from the developer.

In essence you are asking your team to rewire itself to prioritise the growth of the newcomer. This will undoubtedly cause a small amount of slowdown as the process rewiring embeds itself, but you will reap the rewards when your new developer is up to speed in months rather than years.

Action:
Do more planning so that you have smaller tasks that gets your developer coding more. Perhaps you'll need to ask your senior developers to break out their stories into smaller tickets as they make progress on them, but this gives your developer more practice which pays off in the long term.

Next steps

This material is just the start of learning more about how to accelerate your technical talent.

The material is based on ideas by Daniel Irvine, a coach here at Makers, and a software consultant based in London. He is a member of the software craft community, and a founder and organizer of Queer Code London.

We work with other coaches as well and we run a number of events at Makers to help employers learn more about how to get the most out of their tech team.

If you'd like to learn more, please feel free to contact us on **hire@makers.tech.**

We hope to see you at Makers soon!

The Makers team