How to effectively lead an inexperienced team of junior developers

Written by: Joseph Gefroh, VP of Engineering at HealthSherpa

Leading a team of inexperienced juniors is challenging but rewarding.

Many leaders struggle and eventually throw in the towel when being placed in charge of a team full of inexperienced members. They often become overwhelmed by the lack of skill available to their team. Like pushing against a rope, they make no progress towards goals and may even may even make negative progress.

Leaders stuck in this situation may blame their team, forgetting the mantra that "there are no bad teams, only bad leaders." Though often well-intentioned and otherwise talented, these leaders end up flailing in their role.

Eventually they either burn out or are replaced due to their poor performance.

It doesn't have to be this way.

Inexperienced teams can perform incredibly well when guided by the right leadership. They can even perform better than a team full of experienced members under poor leadership.

Good leadership requires good planning

A lot of leads simply go with the flow, performing ad-hoc interventions on an as-needed basis to lead their team. They call out mistakes as they see them, give pats on the back here and there, and otherwise wonder why their teams fail.

Their freewheeling attempts to tackle problems as they come results in a lack of focus, reducing the effectiveness of the team as they work on efforts that don't synergistically build on each other.

A failure to plan is a plan to fail. Good leadership requires planning and follow-through, while understanding that no plan survives contact with reality. As a leader it's your responsibility to create a solid plan and be prepared for any contingencies, adapting accordingly to changing conditions.

The following plan should help you.

Figure out what kind of team you have

The terms "inexperienced" and "junior" can be incredibly broads labels.

A person can be great at one thing but junior in another. Identifying where your team's individual strengths and weaknesses lie is therefore the key first step in leading them.

Make a skills matrix, listing the various skills an individual would need to succeed in your endeavor. Be sure to list soft-skills like communication, the ability to work with others, and whether they encompass the values and traits of your organization. These are just as important as hard, technical skills.

After you rate them, you'll notice that the juniors can fall into certain categories.

Some juniors encompass all of the categories, while others may only need work in one or two areas. These categories aren't mutually exclusive, and it's important to understand what kind of junior you are dealing with so you can appropriately alter your approach.

The technical junior

The technical junior is a person that lacks the hard skillsets of the industry or role. It's a fresh college graduate starting their first job, or someone making a mid-life career change.

Technical juniors do not have familiarity with the tools, techniques, and skills needed to perform competently. They require very conscious effort to do things that may otherwise be simple or subconsciously perform by more technically experienced members. They're unable to make tradeoffs and good decisions because they don't know what they don't know.

The process junior

The process junior lacks the experience and skills working within a team. They may not understand the team culture, structure, dynamics. They lack context into the history of the team and how the team collaborates, coordinates, and communicates. Working with them is an act of friction.

The end result is chaos for the team.

The behavioral junior

The behavioral junior has personal character or behavioral deficits that keep them junior, despite how good they may otherwise be.

They may lack the initiative to take on work after they are done with their current work. They may not care to learn, only doing the barest minimum needed to complete their task. They may lack follow-through or have issues communicating. They may not have attention-to-detail. They may be unable to take constructive criticism. They may lack drive, initiative, or good judgment.

Whatever the issue is, it prevents them from operating at the level they need to operate at.

Develop your team

Once you've conducted an honest assessment of your team's individual strengths and weaknesses, you can make plans that take their strengths and weaknesses into account as individuals and as a team.

Developing technical juniors

Developing technical juniors is a matter of focused, targeted training. The goal is to build up their knowledge and experience of the fundamentals until it becomes unconsciously known.

Develop a training program that conducts focused, targeted instruction on the various technical aspects of the role. Provide real-life examples when possible. Conduct drills, leverage the power of repetition, and ensure the fundamentals are mastered.

Developing process juniors

Developing process juniors is a matter of ensuring two things are understood:

- The process itself
- The reason why the process exists

Some process juniors simply don't know a better way, and once exposed to it are eager to follow the process.

Other process juniors may have a difficult time following any process, seeing them as unnecessary friction to accomplishing their goals. So-called "cowboy coders", they likely haven't experienced the pain the process prevents.

Describing why the process exists and the consequences of not following the process in detail can help align their behaviors. If they still fail to follow the process, establishing checkpoints, boundaries, and penalties can help align behavior.

Process juniors are often a source of tremendous initiative. With a fresh pair of eyes, they can bring in new efficiencies that challenge the status quo. However, their idealism must be balanced by the reality of the situation. Ensure they can follow the existing process first and appreciate why it exists before allowing them to introduce new processes and refinements.

Developing behavioral juniors

Developing behavioral juniors is the most challenging. Change comes from within, and asking someone to change their behavior can be a fool's errand.

The behavioral junior can lack the self-awareness to recognize their flaws, exhibits defensiveness when criticized, or allows pride or ego to prevent them from making progress.

Pointing out behavioral flaws is also one of the hardest subjects for most leads to broach. People have a natural aversion to directly criticizing others, and a lack of control over emotions can turn a potential teaching moment into a heated argument.

Developing junior developers with behavioral traits you'd like to change comes down:

- Clarifying which specific behavioral traits you'd like to see and why
- Demonstrating those traits yourself
- Explicitly pointing out when those traits were or were not demonstrated
- Monitoring and holding juniors accountable for their progress

Changing behavior is a long-term game even in the best scenarios. Sometimes you won't have the runway to wait for that change to occur. In these cases, it's best to cut your losses.

Execution

While the best executing teams have clarity, competency, and control, providing a team of junior developers full autonomy is a recipe for disaster. It leads to poor decisions, which leads to broken systems and negative effectiveness as you work to undo the damage and pay down the tremendous amount of technical debt they generate.

As a leader it is your responsibility to give junior developers room to grow and potentially fail, but not room to sink the ship. It requires careful calibration between freedom and restriction. While learning can't happen without mistakes and failure, it's your job to ensure the mistakes junior developers make are survivable.

Front-load decision-making

Junior developers by definition don't have the clarity or competency to utilize good judgement when they make decisions. It's important to ensure that major decisions, such as architecture, technologies, and patterns, are not initially in their purview.

This means that when starting a new feature or module, make the major decisions ahead of time. They should be working within a framework of established patterns, practices, architectures, technologies, and procedures. If this doctrine doesn't exist, create it.

Ensure this framework exists before they reach the point where they have to make these decisions themselves. Train them on the patterns and practices. Create component libraries. Demonstrate the way you want the system built.

Front-loading major decisions has the added benefit of helping junior developers avoid decision fatigue. They avoid making decisions they don't have the competency or clarity to make, and focus their decision-making at the micro-level that they are gaining mastery over: naming, variables, etc.

Making the major decisions doesn't mean completely shutting off a junior developer's viewpoint or voice. Junior developers can introduce new ideas and help keep your organization from becoming an antiquated dinosaur. However, balance their new ideas with the risks and realities of your organization.

Give them visibility into the thought processes behind your decisions so they can build up their own ability to judge trade-offs. Ask for their input, but ensure that the final decision itself remains with you.

By front-loading the decisions before they reach the decision-point, it helps focus junior developers during execution.

Break it down, barney style

Junior developers may have knowledge of individual pieces, but lack the intuition and experience that allows them to apply their pattern-recognition skills to other situations. As a result, they're unable to rapidly move outside their areas of comfort and knowledge, even if the work is very similar.

Junior developers won't recognize a rose by any other name, even if it looks as pretty or smells just as sweet.

Turn knowledge work into mechanical work

For junior developers who haven't mastered the fundamentals and basics, it's imperative to break down their work and provide step-by-step instructions. They need to learn to crawl before they learn to run.

Be as explicit as possible. Perform drills, going step by step over setting up a class or calling a method. Ensure they achieve mastery over the building blocks and fundamentals before having them attempt anything else.

Once they begin achieving competency with their tools, you can move on to working with them on integrating their fundamentals into the larger effort.

Teach them principles and rules of thumb

As junior developers learn the basics, you can start explaining the "why" behind the techniques.

Explain the principles that form the foundation and reasoning behind what they are doing. Give them rules of thumb they can generally reliably follow. As they execute their day-to-day work, provide course-correction on where the principles are violated and how they can change their work to remediate the defects.

Over time, they will learn to apply these principles themselves without being told to. They'll come to develop their own heuristics, which are critical to being able to apply what they are doing to other similar situations.

Teach them when to violate the principles and rules of thumb

Software engineering is a context-based profession. Decisions that may be terrible in one circumstance can be the right thing to do in another. It's important that you teach junior developers the various decision-making tradeoffs and how to make the tradeoffs themselves.

This will help avoid turning them into dogmatic engineers who are incapable of adapting when needed.

Provide clarity on direction and where they fit

If you isolate junior developers and treat them as cogs in a machine, you prevent them from understanding their ultimate role in the project.

Without understanding their place in the bigger picture, they will become perpetual juniors, merely carrying out tasks and incapable of executing anything beyond simple, explicit instructions.

This may have been the end goal of Taylorist management, but the world has evolved in complexity since the philosophy's heyday. Today, the best teams have the ability to adapt to a constantly changing environment.

The only way a team can adapt is if its members have a holistic view of the situation.

You don't want junior developers staying junior forever.

Teach them their role

Explain to them the other moving parts in terms they can understand. Paint a picture of how their contribution (or lack thereof) impacts the overall goals and initiatives of the project and organization.

As junior developers improve, their understanding of their role will help guide them as they take on increasing amounts of responsibility. These guide-rails will focus their efforts on the things that matter and help ensure they execute within the constraints that makes sense for the team and organization.

Set goals

Focus and effective execution requires a goal or objective to work towards.

Work with your team to set attainable goals at regular intervals. These goals should be larger goals your team works towards, as well as individual goals that contribute to the main effort. By having your team work with you on deciding these goals, it improves their buy-in and motivation, increasing engagement.

Ensure you also establish milestones and interim goals that can serve as progress markers and checkpoints to determine whether the entire effort is on track and where attention may be needed.

These act as yellow flags and early warning signs of potential issues that need to be addressed.

Set boundaries

It's not enough to set goals — goals can be achieved in many ways, some highly negative and damaging, especially if the junior developer hasn't yet developed proper judgement. It's important to ensure that boundaries are explicitly clarified and understood by the entire team.

Boundaries can be situation-specific things like "never use single-letter variables" to generalized value-based boundaries like "never make a customer feel bad".

These boundaries act as anti-goals, things that should not be done, and are important in establishing what is accepted and not accepted behavior. Norm-setting is important in creating alignment and clarity, and without it you can end up with a team that achieves its goals, but in a chaotic and destructive way.

As your junior developers improve their judgement and prove credibility, you can start lifting boundaries and widening their area of operation.

Supervise

It's not enough to set the pieces and press "play". Good execution requires supervision. Course-correction and pointing out just-in-time lessons are valuable learning opportunities that leaders need to provide to junior developers.

Frequently check-in

Monitor progress and check in frequently with your junior developers on their growth. Constantly assess where they are and ensure they are making progress. Check progress against goals and milestones.

Provide them resources they need to learn — whether that be training materials, larger challenges, or your own time.

Ensure they know where they stand and where they are expected to be, and have the roadmap to get there.

Ensure there's a process

Junior developers don't do well when given an infinite amount of possibilities, and a tremendous amount of damage can be done over time if you leave them to their own devices.

Limit potential damage by ensuring that there's a gated process where you have final authority. Ensure that there's a code review process in place. Only allow deployments that you approve.

If your developers have trouble following the process, enforce it with technical solutions or management penalties.

These processes can be lifted as your junior developers grow and improve, but until then they act as safety nets that prevent harm to themselves and the business.

Give feedback often

Junior developers thrive when given feedback. They need this constant course correction, both to ensure they don't sink the ship and to ensure they are learning the right things. Over time, they can begin giving themselves this feedback, providing them the ability to self-regulate once they are capable of doing so. This will ultimately ease the burden on you as a manager or lead.

Use 1:1 meetings with them effectively, ensuring that they know exactly where they stand in their professional development and performance. Point out areas in conversations where they can improve and where they are making good progress.

Be patient

No learning can be done without the possibility of failure. Accept that your junior developers will fail, or seemingly backslide on progress.

Act as necessary, but understand that punishing failures doesn't stop them, it simply hides them. Sometimes you'll have to let certain areas fail while you focus growth and attention on a more important area.

Failure is a part of the process of growing and learning. Be patient.

Know when to loosen the reins

Over time, junior developers will get better. The sort of managing that helped them initially will start to feel more constraining, dampening effectiveness and morale. It's important as a lead to know when to loosen the reins and provide more autonomy and less supervision.

Keep a pulse of how junior developers grow and make progress. You can't track what you don't measure, so maintain a checklist of the traits, skills, knowledge you want them to demonstrate. Reward progress with more autonomy and trust.

Leading a team of junior developers is difficult but rewarding. As they learn and grow, the team will gel over the shared challenge and become more and more effective over time.

Use 1:1 meetings with them effectively, ensuring that they know exactly where they stand in their professional development and performance. Point out areas in conversations where they can improve and where they are making good progress.