Sourcegraph

Continuous Developer Onboarding:

A guide to cultivating a culture of professional growth in your engineering organization

Table of Contents

I wonder if part of what we do as engineers is kind of this continuous onboarding because no one likes to work in the same part of the code for too long. We're always looking to acquire new skills, learn about new things and build things in other areas of the product. So I guess, are there lessons that we can take from what makes a good onboarding experience to what makes a productive software team in general in kind of the steady state?[1]

Beyang Liu
Co-founder & CTO at Sourcegraph

At most companies, onboarding processes assume an endpoint. When an engineer finishes their checklist, whether that's after a week, a month, or a year, they're onboarded. At that point, if onboarding went well, they're a successful employee. They've *learned* what they needed about workflows and infrastructure, the codebase they're working on, and the culture in which they work.

But, as Beyang Liu, co-founder and CTO at Sourcegraph, notes, engineers want to explore new parts of a codebase, learn new skills, and build new things. Engineers will switch teams, and teams will themselves take on new features and own different parts of a codebase over time. Successful engineers are those who can make these transitions in a largely self-guided way. They're engineers who continuously onboard.[2]

Given how time-consuming and resource-intensive onboarding can be, how do companies set developers up to continuously onboard themselves through the transitions of their careers? In this guide, we describe one approach that focuses on onboarding processes and an engineering culture that nourishes engineers and provides them with the flexible, supported room to develop their own learning capacities.

We frame this approach with the metaphor of a garden and gardening, as developed by Dr. Alison Gopnik in her work on developmental psychology and parenting.[2] While companies aren't parents, and professional engineers aren't developing children, Gopnik's metaphor helps to recontextualize traditional onboarding practices and organizational approaches to knowledge management, tool use, and culture.[3]

[1]Liu, "Developer Onboarding with Jean du Plessis, Director of Engineering, Sourcegraph."

[2]For an explanation of the importance of metaphors for how we think, see George Lakoff and Mark Johnson, *Metaphors We Live By*, ed. With a new Afterword (Chicago, IL: University of Chicago Press, 2003).

[3]Alison Gopnik, *The Gardener and the Carpenter: What the New Science of Child Development Tells Us About the Relationship Between Parents and Children* (Farrar, Straus and Giroux, 2016); Annie Murphy Paul, *The Extended Mind: The Power of Thinking Outside the Brain* (Mariner Books, 2021). Gopnik's work, while focused on children, also coheres well with contemporary research on effective cognition, particularly in regard to the role of people, relationships, and environments. Paul's, *The Extended Mind*, is a very effective overview of *embodied cognition, situational cognition, and distributed cognition,* that is, how we think with and through our bodies, our environments, and other people.

Sourcegraph

Continuous Developer Onboarding:

Gardening and onboarding

Gopnik contrasts two approaches to parenting. The first sees being a parent like being a carpenter. She writes, “Essentially your job is to shape that material into a final product that will fit the scheme you had in mind to begin with. And you can assess how good a job you’ve done by looking at the finished product. Are the doors true? Are the chairs steady? Messiness and variability are a carpenter’s enemies; precision and control are her allies.”[4] Carpenters have a plan, and clear and concrete measures for the success of a project. They’re also constrained by the raw materials with which they work and the designs of their plan.

Traditional onboarding assumes this approach, to one degree or another. It starts with a new employee, with an existing set of skills and experiences known through their application materials and interviews, and shapes them into an employee with a certain set of knowledge and skills appropriate to their new position. Onboarding can be measured by self-review and completion of a set of tasks that demonstrate these skills and knowledge, confirmed by later employee reviews. This approach is necessary for new employee onboarding. Business needs require some ability to measure whether onboarding processes are successful.

Being a gardener is rather different than being a carpenter though. As Gopnik notes, “When we garden… we create a protected and nurturing space for plants to flourish.”[5] She continues: The good gardener works to create a fertile soil that can sustain a whole ecosystem of different plants with different strengths and beauties–and with different weaknesses and difficulties too. Unlike a good chair, a good garden is constantly changing, as it adapts to the changing circumstances of the weather and the seasons. And in the long run, that kind of varied, flexible, complex, dynamic system will be more robust and adaptable than the most carefully tended hothouse bloom.[6]

While a carpenter might produce a chair, gardeners don’t just produce plants.[7] They build and sustain gardens. They create an environment, and shape it, knowing that they might need to regularly change and adjust depending on internal and external factors. Unpredictability shapes the garden, yet the garden focuses on the healthy growth and maturation of its inhabitants. Crucially, the care of the whole garden isn’t determined by the needs of any single plant but instead by the balanced needs of the whole.

This approach to parenting, of being a gardener instead of a carpenter, isn’t without a goal. Its goal just isn’t about defining a single, definable shape for a child. Gopnik explains that being a good parent, “won’t transform children into smart or happy or successful adults. But it can help create a new generation that is robust and adaptable and resilient, better able to deal with the inevitable, unpredictable changes that face them in the future.”[8]

Continuous onboarding isn’t just learning in an open field. It’s learning and adapting within the determined, though potentially shifting, context of an existing organization, set of working practices, and network of people and codebases.

Codebases, maybe the most important context for developers, change regularly and rapidly. Ryan Djurovich, Senior Engineering Manager at Xero, mentions that “when it comes to onboarding onto codebases and infrastructure and that kind of thing, you should never be done with learning there because those systems aren’t in a consistent state or hopefully aren’t in a consistent state, they’re evolving over time as the business evolves.”[9]

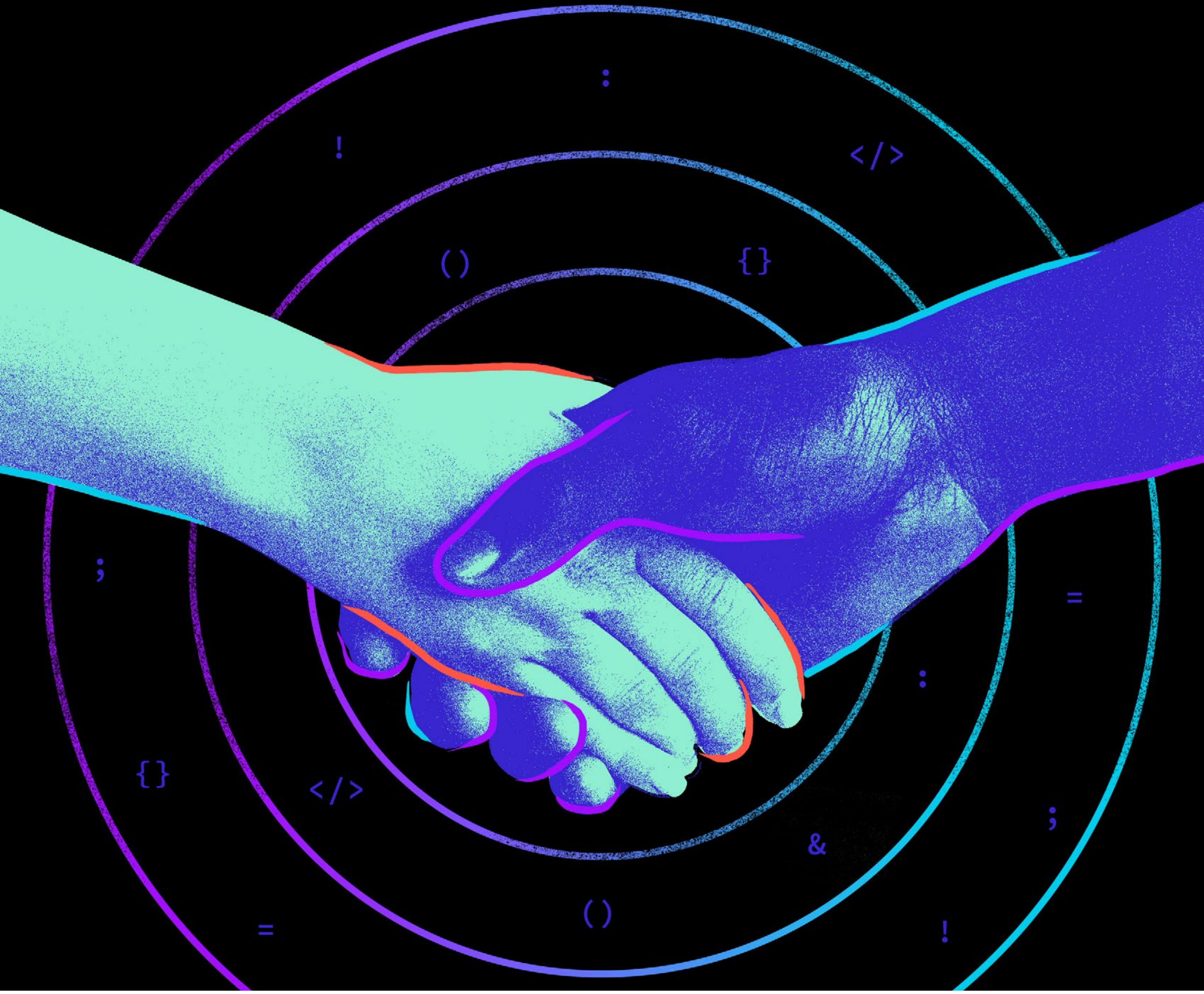
Engineers who are adaptable, resilient, and able to deal with unpredictable changes are more likely to be successful in switching teams and exploring new parts of a codebase, especially when provided with tools that facilitate their self-serve exploration of a codebase.

As an approach, continuous onboarding encourages companies to create and sustain an environment that flexibly supports and develops an ecosystem of adaptable and resilient engineers. These engineers are more likely to persist in the company, gradually accumulating knowledge across the breadth of the product.

This approach won’t replace traditional company onboarding, which is still necessary and important. It does help recontextualize and situate it within a holistic approach to employee happiness and company success.[10]

[4]Gopnik, *The Gardener and the Carpenter*, 18.
[5]Gopnik, 18.
[6]Gopnik, 19.
[7]Gopnik knows that there are different types of gardeners and gardens, some of which are more constrained and directed. See the introduction of Gopnik, *The Gardener and the Carpenter,* for her discussion on this.
[8]Gopnik, 19.
[9]Beyang Liu, “Developer Onboarding with Ryan Djurovich, DevOps Manager, Xero,” 2021, https://about.sourcegraph.com/podcast/onboarding-ryan-djurovich/.
[10] In the interest of holding the difference between children and engineers, we’ll leave behind the specific details of Gopnik’s approach to being a parent unless directly relevant in favor of examples from existing tech companies, such as Sourcegraph.

Sourcegraph



Continuous Developer Onboarding:

People and relationships

The most compelling feature of many “guides to onboarding” is the recommendation to help new employees build relationships with existing employees, in and outside of their department, right away. This can take several forms.

Companies might, for example:

- Give new employees a list of individuals to set up coffee chats with in the first month or two.
- Use Donut, a Slack app that automatically matches employees for short chats at preset intervals.
- Offer new employees an onboarding buddy from outside their department, who they can meet weekly for the first month or two.

All of these are great approaches since they give clear direction to a new employee who might not know who to reach out to or be anxious about initiating meetings. They have the added effect, though, of setting the standard for full team buy-in in new employees. This inclusive concern shapes company culture to support employees from the start. Adam Harvey, a Sourcegraph engineer, said, “Culture becomes important. You want the space to make mistakes when you’re coming into a new environment. You want a combination of psychological safety (knowing that your team won’t throw you under the bus) and technical safety (guardrails to know that you won’t make a huge mistake for a customer).”[11]

How do you create that space for psychological safety? Harvey suggests that, in part, it’s full-team buy-in across all the levels of the company. He notes how Sourcegraph’s CEO, Quinn Slack, lets every cohort of onboarding employees know that every person at Sourcegraph will drop what they’re doing to help, which “sets a strong signal across the company that it’s not just platitudes, but a priority.”[12]

Balancing top-down with bottom-up

This top-down signal has to be matched by follow-through. Bottom-up approaches such as Donuts and onboarding buddies build relationships with individuals across the company that make it easier to ask someone for help or see examples of someone pausing their work to help.

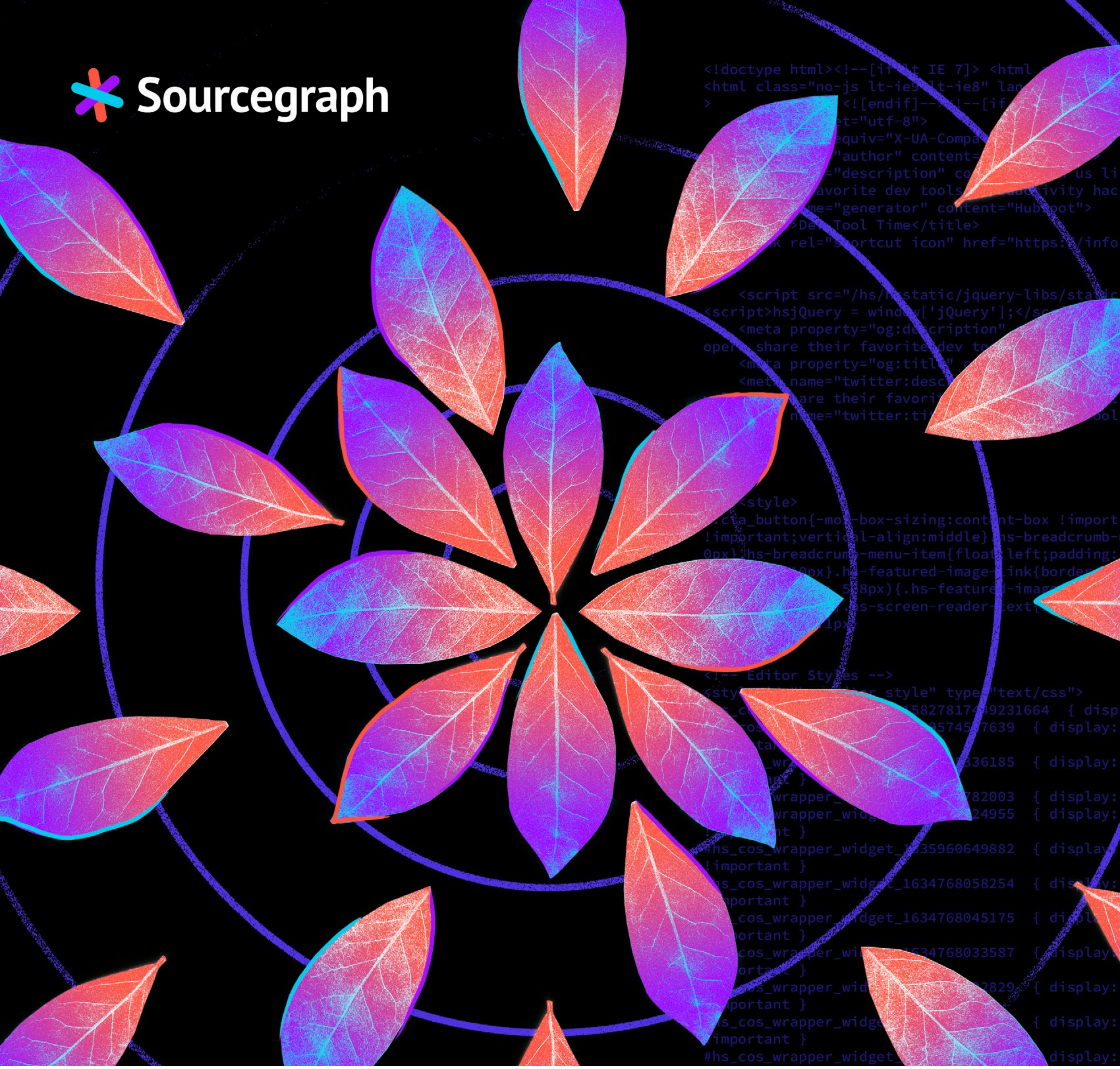
In the best cases, these relationships persist and help smooth the way when developers begin to work on code owned by other engineers, or when they switch teams. Commenting on effectively learning about new parts of a codebase, Stephen Gutekanst, one of Sourcegraph’s longest-tenured engineers, said that even when centralized knowledge bases and documentation exist, it’s often best simply to talk to the engineers who’ve worked most on a section of code, or who feel ownership over it. This isn’t just about handling egos, but setting a pattern of collaboration built on respectful relationships. It’s being attentive to people, especially when you’re proposing a change or have hit the limits of self-guided learning.

As Patricia No and Beyang Liu note in a piece on onboarding: “Sometimes being lost isn’t just a discovery problem, but a social one.”[14] There are genuine technical problems in onboarding, continuous or not, but most are socio-technical. What marks engineers with the capacity to continuously onboard is care for the people that constitute the social network within which codebases and their changes exist.

Focus on people is marked by two sides in both traditional and continuous onboarding: the onboarding individual and the existing community. Onboarding is successful when individuals seek out connection and actively cultivate and maintain relationships, understanding the importance of other people in the networks in which we work.

12 No and Liu.
13 No and Liu.

Sourcegraph



Continuous Developer Onboarding:

Centralizing knowledge: current and historical

Even when everyone in a company is happy to answer questions at any time, most organizations provide some form of centralized knowledge base with which new employees thoroughly acquaint themselves.

For some organizations—particularly those with handbooks—this knowledge base persists as a main source of truth for all employees, is *mostly* kept up to date, and is regularly shared as the answer to many questions.[14] Such knowledge bases often crystallize the current state of knowledge or the truth as the company understands it at the time of writing. Depending on the technical platform or the approach to the knowledge base, it may also contain historical knowledge or explain how something changed over time.[15]

When it comes to codebases, though, such knowledge bases typically document knowledge about the codebase and its relation to teams and processes rather than technical knowledge. You might go to the knowledge base to learn which engineering team owns part of a codebase, but the dependencies of that part of the codebase and its architecture are likely to be documented elsewhere, whether in a set of docs or the code itself.

How do we bridge these two locations of knowledge and their use to facilitate an environment of continuous onboarding?

Document context for decisions, not just decisions themselves

Providing relevant and necessary context is vital for continuous onboarding. According to Jean du Plessis, Director of Engineering at Sourcegraph, companies are "deliberate about making sure that they [new hires] have the right information."[16] But when teams and employees are established and moving into new product features or areas, du Plessis says, "we tend to overlook that upfront work that you always need to do to get yourself into that ready state."[17] This upfront work often involves understanding "the context of the area of work" rather than rushing in.

"If it's proven that context and being deliberate in preparing for an engineer while they're onboarding is effective, and we have this similar pattern that happens in different stages of teams … it might be worthwhile exploring those again," du Plessis said.[18]

What counts as context can vary depending on the situation. With any codebase, the technical architecture of code relevant to a feature, its connection to other features, and its dependencies are a minimal context. Potentially as important are the decisions that resulted in that code, and the tradeoffs already considered. In some cases, a developer can make significant changes, enhancements, or bug fixes without knowing why the code is the way it is. But in many other cases, particularly where there is a longer history and a larger number of people who've touched the code, understanding the history of the code's formation amidst its alternative possibilities is crucial.

Documenting this history in a centralized, accessible fashion is difficult. Depending on a team's coding practices, some of this information may be in code comments or written up in technical documentation. It might be in GitHub pull requests or issues, or in Google Docs where a team has weighed multiple approaches.[19]

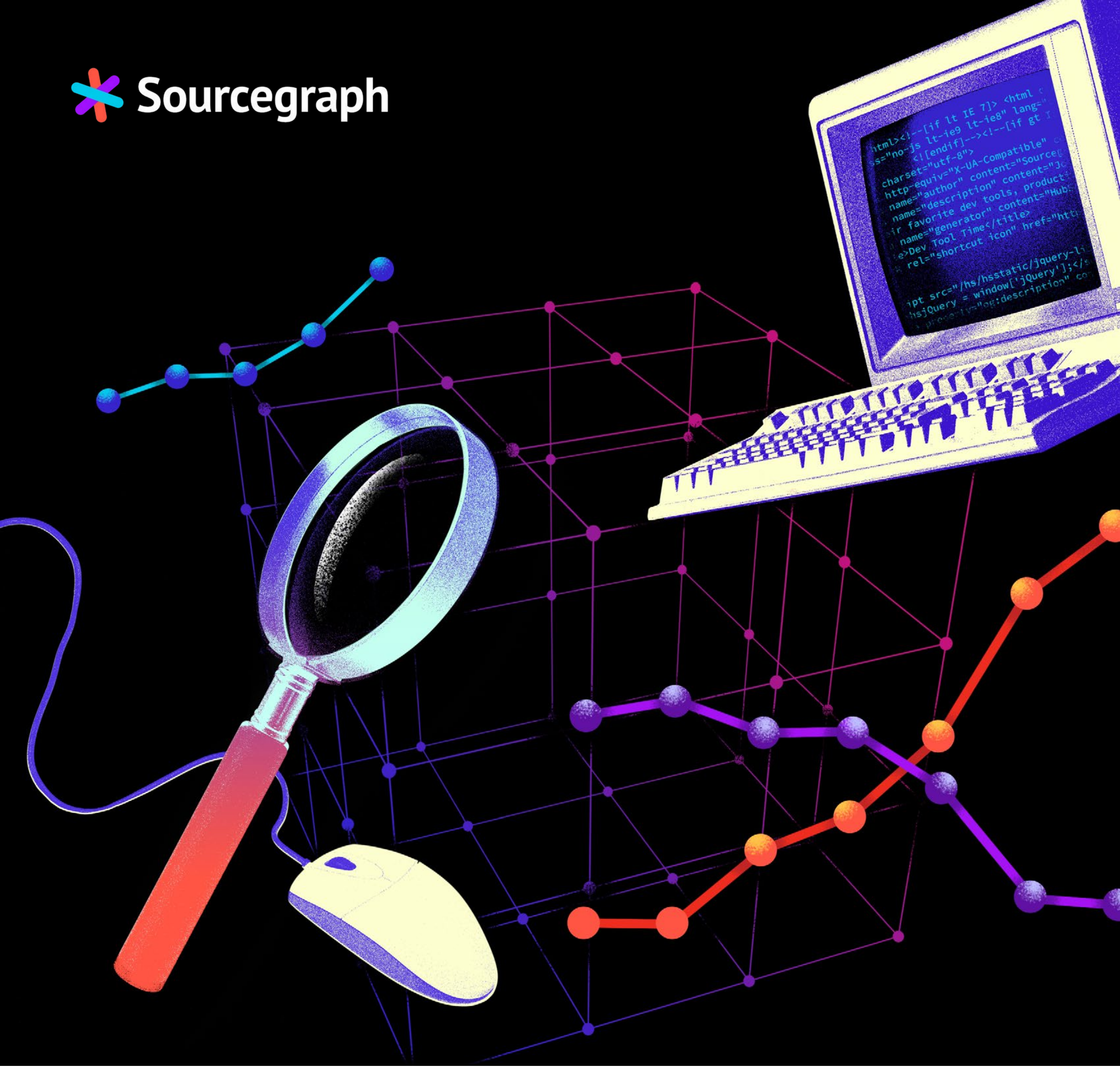
Regardless of place, engineers shifting teams or sections of a codebase need to find as much context as possible to make decisions that acknowledge work already done. If both current and historical knowledge can be kept in a single place, all the better, but if not, an organizational knowledge base should have an ordered index of relevant places to look for context.

It's never possible to completely document all of the context for any code or technical decision. There will always be some piece of information that exists in a single person's head that never made it into docs. When there are synchronous conversations, there will always be some relevant comment that isn't recorded.[20] Some knowledge may only be surfaced through collaborative work.

Even so, centralized knowledge provides a foundation for engineers to exercise their problem-solving skills and enables them to ask better questions when they need help or additional context from others.

[14] Examples of handbook-focused companies are GitLab and Sourcegraph.
[15] For example, a knowledge base under version control would contain its previous versions, unless its history is deliberately obscured.
[16] Liu, "Developer Onboarding with Jean Du Plessis, Director of Engineering, Sourcegraph."
[17] Liu.
[18] Liu.
[19] Some organizations use formal Requests for Comment (RFCs) for this, while others use less formal structures.
[20] Or never listened to, even if it is recorded.

Sourcegraph



Continuous Developer Onboarding:

Tools for discovery

I think having those tools and methods of finding the information you need as you need it, is a good thing. It's a good skill to develop, it's a good tool belt to develop. So not just for onboarding, but also for your time at a company.[21]

-Ryan Djurovich
Senior Engineering Manager at Xero

Knowledge is most useful when it is findable. Centralization is helpful, but given that most companies store knowledge in at least two locations, how do you make knowledge accessible to engineers?[22]

As Ryan Djurovich notes in the quote above, providing engineers with tools particularly suited to finding and sharing information in the most relevant locations is one of the biggest facilitators of self-service continuous onboarding. Even more importantly, the best search tools help developers build approaches to answering questions that are reusable and applicable in other situations, and potentially transferable even if a tool were to no longer be available.

As a code intelligence platform, Sourcegraph is one example of a tool that fits the nature of the knowledge searched (code) and empowers developers to drive their own learning over time and in different situations. It's built for one main type of content, and surfaces relevant information.

There is a learning curve, and it isn't automatic. The best results come from learning to use the different types of search (literal, regular expression, and structural) in conjunction with the particular questions asked. This skill in narrowing contexts and learning to operationalize questions according to the limits of a tool is itself valuable, and transferable to other tools.

As a tool, Sourcegraph also facilitates developer autonomy, allowing individual developers to search in and across repositories, find the up-to-date results regardless of underlying changes, and craft multi-repository change sets. According to Bjørn Sørensen of Lunar, "Sourcegraph makes it possible for us to enable every team to develop autonomous practices, [which are]vital to ensure developers and their teams can accomplish their day-to-day work in isolation without being blocked." Tools that enable developers to work on their own and with their small teams, especially tools that enable them to efficiently access the shared knowledge of their peers, current and historical, are particularly powerful in unblocking developers.

Keeping a consistent toolset

It's important that these tools remain relatively consistent. As Djurovich notes, with "familiarity with what tools you're using, how things tend to be set up, what the conventions are for that company, it becomes easier to onboard onto other teams because you don't have that initial hurdle to get over of adjusting to the new environment, new tools, new conventions."[23] Initial company onboarding, or trainings over time, should cover the fundamental set of tools that engineers are expected to use. This knowledge should be transferable across teams, even if there is some variation in workflows and how tools are used.

The developer tools ecosystem changes rapidly. When adopting new tools to accelerate velocity or make developers happier, it's worth evaluating how long you expect to remain committed to that tool given downstream impacts to developers' continuous onboarding. While some engineers love exploring new tools, time spent learning a series of regularly changing, required tools is time taken from building relationships and knowledge while writing code.

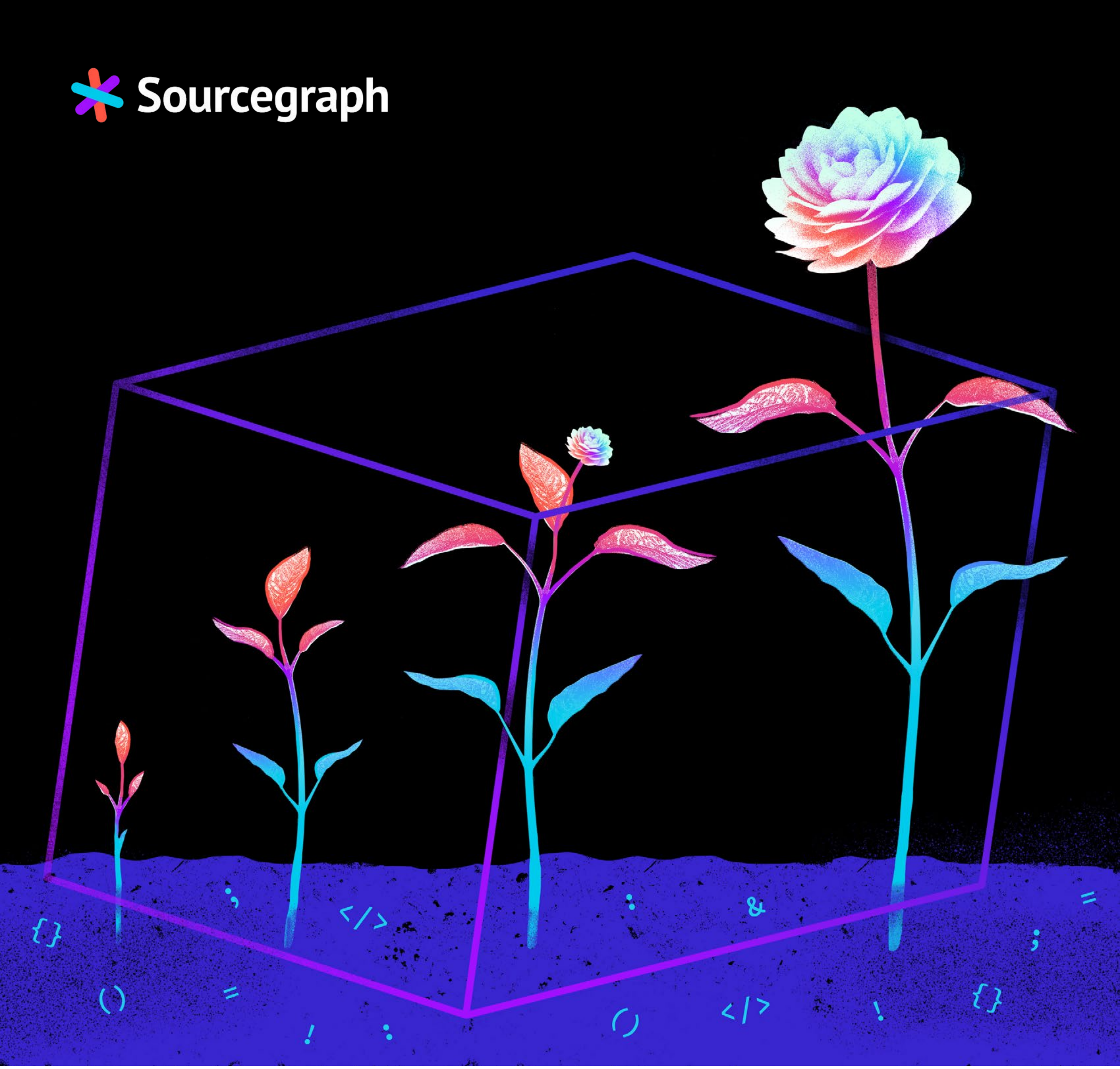
Autonomy is crucial for continuous onboarding. Depending on company size and infrastructure, it isn't possible to run formal onboarding processes every time someone switches teams. When developers simply move on to new features or parts of a codebase, it doesn't require a formal process. It's better for companies to focus on enabling developer self-serve learning through effective and consistent tools for information access.

At some point, even the best tools combined with the most centralized knowledge will hit their limits. Successful engineers will develop their own sense of these limits as they apply tools in different contexts to answer myriad questions. Organizations can facilitate this skill-building by understanding and documenting understood limits themselves, not to discourage developers from experimenting and testing the limits, but to signal where developers might expect more friction and a challenging path forward.

When even experiments that stretch the limits of tools fall short, the best resource left will often be people, whether internal to the organization or in the broader developer ecosystem. Sustained and cared-for relationships open up depths of knowledge that often exceed the limits of our tools.

[21] Liu, "Developer Onboarding with Ryan Djurovich, DevOps Manager, Xero."
[22] Minimally, in code and in some documentation external to code.
[23] Liu.

Sourcegraph



Continuous Developer Onboarding:

Building and holding space for growth

Gardens are cultivated spaces. They're spaces that a gardener actively holds together, shapes, and maintains to provide an environment for growth. In any company, engineers exist within a network of people, tools, documentation, decisions, and cultural practices that both constrain and facilitate their capacities for growth.

Approaching this network with the aim of continuous onboarding requires attending to each factor that impacts an employee's capacity to learn and grow, and balancing their effect and cost. It also means balancing organizational structure and flexibility.

Few gardens are shapeless sprawls. They have boundaries and some amount of structure, but not so much that plants have no room to grow. Rigid organizational structure can suffocate growth and make change painful. Yet, too much flexibility and too few consistent standards across departments and teams, particularly in tools and workflows, makes it difficult for engineers to work across and even change teams.

Instead of long rulebooks and minutely defined procedures, successful companies rely on shared goals and shared context to hold engineers and teams together and balance their variations. These shared goals, especially in the context of a network of people and knowledge, enable managers and decision-makers to trust engineers to make the right decisions as they find their own ways forward, whether into new parts of the codebase or new parts of the company.

Conclusion

Much of what we've explored here as continuous onboarding is familiar to anyone who has ever been through or run an onboarding process. But the goal of facilitating continuous onboarding isn't the same as building out a finite, measurable onboarding process for new employees.

As an idea, continuous onboarding recontextualizes traditional onboarding practices and organizational approaches to knowledge management and tool use. Especially framed with the garden metaphor, continuous onboarding emphasizes cultivating an environment—particularly its culture and practices—that enables developers to build themselves as flexible and resilient engineers.

It's a practice for the long run, and for capacitating developers through a culture of care and empowerment that helps them do what many love best: write code and solve problems.

Sourcegraph

about.sourcegraph.com

Sourcegraph is a code intelligence platform that unlocks developer velocity by helping engineering teams understand, fix, and automate across their entire codebase.