# **CH\_3 : Knowledge Sharing**

Challenges to learning

Sharing expertise across an organization is not an easy task.

Without a strong culture of learning, challenges can emerge.

Google has experienced a number of these challenges,

especially as the company has scaled:

Lack of psychological safety

Being afraid to take risks or make mistakes => fear of punishment.

Because of tendency to avoid transparency.

Information islands

Knowledge fragmentation in different parts don’t communicate with one another or shared resources.

each group develops its way of doing things.

fragmentation: Each island has an incomplete picture of the bigger whole.

duplication: Each island has reinvented its own way of doing something.

skew: Each island has its own ways of doing the same thing, and these might or might not conflict.

Single point of failure (SPOF)

A bottleneck when info is only from a single person. => bus factor

SPOFs can arise out of good intentions: => “Let me take care of that for you.” / “It’s faster for me to do it”.

This mindset also tends to lead to all-or-nothing expertise.

All-or-nothing expertise

people who know “everything” and novices, with little middle ground.

don’t take the time to develop new experts (mentoring / documentation)

knowledge & responsibilities are on experts.

novices are left to fend for themselves and ramp up more slowly.

Parroting

Mimicry without understanding.

copying patterns or code without understanding purpose

code is needed for unknown reasons.

Haunted graveyards (*Unlike the parroting*)

avoid touching or changing code because they are afraid of being wrong.

fear & superstition.

Philosophy

SWE is a multiperson development of multiversion programs.

People are at the core of software engineering:

code is an important small part of building a product.

Nothing emerges spontaneously neither code nor expertise.

growing and investing in its people is crucial.

one-to-one advice is invaluable.

if the expert goes on vacation or switches teams, the team can be left in the lurch.

Although one may provide help for many, this doesn’t scale and is limited to small numbers of “many.”

Documented knowledge can better scale to the entire organization.

Mechanisms such as a team wiki to share expertise.

However, documentation is scalable but, that scalability comes with trade-offs:

it might be more generalized and less applicable to individual learners’ situations, and it comes with the added maintenance cost required to keep information relevant and up to date over time.

Written knowledge has scaling advantages, but so does human-targeted help.

A human expert can assess what information is applicable to the individual’s use case.

Tribal and written knowledge complement each other.

good mix (Tribal and written knowledge) will vary based on your organization.

Train, focus on learning and growth, and build your own stable of experts: there is no such thing as too much engineering expertise.

Setting the stage: psychological safety

Psychological safety is critical to promoting a learning environment.

To learn, you must first acknowledge that there are things you don’t understand.

learning is to try things and feeling safe to fail.

people feel comfortable asking questions, being wrong, and learning new things.

psychological safety is the most important part of an effective team.

*Mentorship*

an important way of building psychological safety is to assign Nooglers a mentor (manager, or tech lead)

for answering questions & helping the Noogler ramp up.

no worry about taking up too much of co-workers’ time.

This mentor isn’t on the same team => comfortable.

the mentor is there to be a safety net to talk to => doesn’t know who to ask for advice.

Mentorship formalizes and facilitates learning.

but learning itself is an ongoing process.

In healthy team co-workers can learn from one another (new employee OR expert).

Growing your knowledge

Knowledge sharing starts with yourself. It is important to recognize that you always have something to learn. The following guidelines allow you to augment your own personal knowledge.

Ask Questions

always be learning; always be asking questions.

We tell Nooglers that ramping up can take around six months. learning is an ongoing, iterative process.

One of the biggest mistakes that beginners make is not to ask for help when they’re stuck.

You might be tempted to struggle through it alone or feel fearful that your questions are “too simple.”

Your co-workers are often the best source of information: leverage this valuable resource.

There is no magical day - there’s always more to learn.

Even Engineers who have been at Google for years (they can ask without fear).

Senior or lead ? you should always be in an environment in which there’s something to learn. If not, you stagnate (and should find a new environment).

it’s important not to mistakenly equate “seniority” with “knowing everything.”

In fact, the more you know, [the more you know you don’t know](https://oreil.ly/VWusg).

Openly asking questions or expressing gaps in knowledge reinforces that it’s OK for others to do the same.

This make people feel safe looking for help. Making it easier to even “trivial” questions to get an answer.

Targeted help allows engineers to be productive faster, which in turn makes their entire team more productive.

Understand Context

Learning is not just about understanding new things; it also includes developing an understanding of the decisions behind the design and implementation of existing things.

Suppose that your team inherits a legacy codebase for a critical piece of infrastructure that has existed for many years. The original authors are long gone, and the code is difficult to understand. It can be tempting to rewrite from scratch rather than spend time learning the existing code. But instead of thinking “I don’t get it” and ending your thoughts there, dive deeper: what questions should you be asking?

Consider the principle of “[Chesterson’s fence](https://oreil.ly/Ijv5x)”: *, a fence or gate erected across a road*.

Seek out and understand context, especially for decisions that seem unusual. After you’ve understood the context and purpose of the code, consider whether your change still makes sense. If it does, go ahead and make it; if it doesn’t, document your reasoning for future readers.

understanding the rationale behind a given guideline allows authors to make informed decisions about when the guideline shouldn’t apply or whether the guideline needs updating.

Scaling Your Questions: Ask the community.

Getting one-to-one help is high bandwidth but necessarily limited in scale.

And as a learner, it can be difficult to remember every detail. Do your future self a favour: when you learn something from a one-to-one discussion, write it down.

share what you write down with future newcomers.

Although sharing the answers you receive can be useful, it’s also beneficial to seek help not from individuals but from the greater community.

Group Chats

When you have a question, it can sometimes be difficult to get help from the right person. Maybe you’re not sure who knows the answer, or the person you want to ask is busy.

In these situations, group chats are great, because you can ask your question to many people at once and have a quick **back-and-forth conversation with whoever is available**.

As a bonus, other members of the group chat can learn from the question and answer, and many forms of group chat can be automatically archived and searched later.

Group chats tend to be dedicated either to topics or to teams.

Topic-driven group chats are typically open so that anyone can drop in to ask a question.

Team-oriented chats, on the other hand, tend to be smaller and restrict membership.

their smaller size can feel safer to a newcomer.

It’s difficult to extract meaningful information from a conversation.

As soon as you need to share information outside of the group, or make it available to refer back to later, you should write a document or email a mailing list.

Mailing Lists

Most topics at Google have a topic-users@ or topic-discuss@ Google Groups mailing list that anyone at the company can join or email.

the question reaches a lot of people who could potentially answer it and anyone following the list can learn from the answer. public mailing lists are easy to share with a wider audience: they are packaged into searchable archives.

Don’t get on with your work when you find an answer. You never know when someone will need the same information [in the future](https://xkcd.com/979), so it’s a best practice to post the answer back to the list.

Mailing lists are not without their trade-offs. They’re well suited for complicated questions that require a lot of context, but they’re clumsy for the quick back-and-forth exchanges at which group chats excel.

Email archives are immutable, and it can be hard to determine whether an answer discovered in an old discussion thread is still relevant to a present-day situation.

EMAIL AT GOOGLE

Google engineers receive hundreds of emails each day, with varying degrees of actionability.

Some groups CC large mailing lists onto every discussion by default, without trying to target information to those who are likely to be specifically interested in it; as a result, the signal-to-noise ratio can be a real problem.

Keep this in mind as your organization considers what forms of communication to encourage or invest in.

YAQS: Question-and-Answer Platform

YAQS (“Yet Another Question System”) is a Google-internal version of a [Stack Overflow](https://oreil.ly/iTtbm)–like website.

Like Stack Overflow, YAQS shares many of the same advantages of mailing lists and adds refinements: answers marked as helpful are promoted in the user interface, and users can edit questions and answers so that they remain accurate and useful as code and facts change.

As a result, some mailing lists have been superseded by YAQS, whereas others have evolved into more general discussion lists that are less focused on problem solving.

Scaling Your knowledge: You always have something to teach.

Teaching is not limited to experts, nor is expertise a binary state in which you are either a novice or an expert.

Expertise is a multidimensional vector of what you know: everyone has varying levels of expertise across different areas. This is one of the reasons why diversity is critical to organizational success: different people bring different perspectives and expertise to the table.

Google engineers teach others in a variety of ways, such as office hours, giving tech talks, teaching classes, writing documentation, and reviewing code.

Office Hours

Sometimes it’s important to have a human to talk to, and in those instances, office hours can be a good solution. Office hours are a regularly scheduled event during which one or more people make themselves available to answer questions about a particular topic.

Office hours are almost never the first choice for knowledge sharing: if you have an urgent question, it can be painful to wait for the next session for an answer; and if you’re hosting office hours, they take up time and need to be regularly promoted. That said, they do provide a way for people to talk to an expert in person.

This is particularly useful if the problem is still ambiguous enough that the engineer doesn’t yet know what questions to ask (such as when they’re just starting to design a new service) or whether the problem is about something so specialized that there just isn’t documentation on it.

Tech Talks and Classes

Google has a robust culture of both internal and external tech talks and classes.

Our engEDU (Engineering Education) team focuses on providing Computer Science education to many audiences, ranging from Google engineers to students around the world.

g2g (Googler2Googler) program lets Googlers sign up to give or attend talks and classes from fellow Googlers.

**Tech talks** typically consist of a speaker presenting directly to an audience.

**Classes** can have a lecture component but often centre on in-class exercises => more active participation from attendees.

As a result, instructor-led classes are more demanding and expensive to create and maintain than tech talks and are reserved for the **most** **important** or **difficult** topics.

after a class can be scaled as many instructors can teach a class from the same course materials.

classes work best => following circumstances exist:

**The topic is complicated enough**,

**The topic is relatively stable**. Updating materials - subject is rapidly evolving

**The topic benefits from having teachers available** to answer questions and provide personalized help.

Documentation

Documentation is written to help learn something.

goal of the original question on the thread was to seek answers, & only secondarily to document it for others.

NOTE

Updating documentation

The first time you learn something is the best time to see ways that the existing documentation and training materials can be improved.

if you find a mistake or omission in the documentation, fix it! Leave the campground cleaner than you found it **&** try to update the documents yourself.

It also leaves a trail of change history no different than that for code.

Creating documentation

As your proficiency grows, write your own documentation, and update existing docs (easier for others).

Finally, make sure there’s a mechanism for **feedback**. If there’s no easy and direct way for readers to indicate that documentation is outdated or inaccurate, they are likely not to bother telling anyone, and the next newcomer will come across the same problem.

Promoting documentation

Traditionally, encouraging engineers to document their work can be difficult.

Writing documentation takes time and effort that could be spent on coding, and the benefits that result from that work are not immediate and are mostly reaped by others.

Asymmetrical trade-offs as many people can benefit from the time investment of a few.

However, a document author can often directly benefit from writing documentation. Suppose that team members always ask you for help debugging certain kinds of production failures.

Documenting your procedures requires an upfront **investment of time**, but after that work is done**, you can save time in the future** by pointing team members to the documentation and providing hands-on help only when needed.

Writing documentation also helps your team and organization scale.

Code

code is **knowledge**, so the very act of writing code can be considered a form of knowledge transcription.

Although knowledge sharing might not be a direct intent of production code, it is often an emergent side effect, which can be facilitated by code readability and clarity.

Code documentation is one way to share knowledge. clear documentation not only benefits consumers of the library, but also future maintainers.

Code reviews are often a learning opportunity for both author(s) and reviewer(s).

Scaling Your Organization’s knowledge

Ensuring that expertise is shared as the organization grows.

Cultivating a Knowledge-Sharing Culture

Organizational culture is the human thing that many companies treat as an afterthought.

focusing on the culture and environment first results in better outcomes than focusing on only the output (i.e code).

We don’t pretend to have all the answers, but we can share specific steps Google has taken to create a culture that promotes learning.

**Respect**

The bad behaviour of just a few individuals [can make an entire team or community unwelcoming](https://oreil.ly/R_Y7N).

the group reduces to its most toxic members. It can be difficult to recover from this state.

Knowledge sharing can and should be done with kindness and respect.

*Leaders improve the quality of the people around them.*

Incentives and recognition

Good culture actively nurtured and encouraging a culture of knowledge sharing requires a commitment to recognizing and rewarding it at a systemic level.

At more senior levels, the ladder explicitly calls out the importance of wider influence, and this expectation increases as seniority increases. At the highest levels, examples of leadership include the following:

Growing future leaders by serving as mentors to junior staff, helping them develop both technically and in their Google role.

NOTE

Peer bonuses are a **monetary** award and formal recognition that any Googler can bestow on any other Googler for above-and-beyond work. Because peer bonuses are employee driven, not management driven, they can have an important and powerful grassroots effect.

When a Googler gives another Googler a peer bonus or kudos, they can choose to copy additional groups or individuals on the award email, boosting recognition of the peer’s work.

A system in which people can formally and easily recognize their peers is a powerful tool for encouraging peers to keep doing the awesome things they do. It’s not the bonus that matters it’s the peer acknowledgement.

Establishing Canonical Sources of Information

Canonical sources of information are centralized, company-wide corpuses of information that provide a way to standardize and propagate expert knowledge.

They work best for information that is relevant to all engineers within the organization, which is otherwise prone to information islands.

Establishing canonical sources of information requires higher investment than maintaining more localized information such as team documentation, but it also has broader benefits.

Providing centralized references for the entire organization makes broadly required information easier and more predictable to find and counters problems with information fragmentation that can arise when multiple teams grappling with similar problems produce their own—often conflicting—guides.

Because canonical information is highly visible and intended to provide a shared understanding at the organizational level, it’s important that the content is actively maintained and vetted by subject matter experts.

The more complex a topic, the more critical it is that canonical content has explicit owners. Well-meaning readers might see that something is out of date but lack the expertise to make the significant structural changes needed to fix it, even if tooling makes it easy to suggest updates.

canonical sources of information are expensive and time consuming, and not all content needs to be shared at an organizational level.

**Developer guides**

Instead, a human expert already familiar with a guideline can send a link to a fellow engineer, who then can read the reference and learn more.

The expert saves time by not needing to personally explain a company-wide practice, and the learner now knows that there is a canonical source of trustworthy information that they can access whenever necessary.

Such a process scales knowledge because it enables human experts to recognize and solve a specific information need by leveraging common, scalable resources.

go/ links

go/ links (sometimes referred to as goto/ links) are Google’s internal URL shortener.

go/ links provide a permalink to the content, even if the underlying URL changes

**Codelabs**

Google codelabs are guided, hands-on tutorials that teach engineers new concepts or processes by combining explanations, working best-practice example code, and code exercises.

Codelabs are an interesting halfway point between static documentation and instructor-led classes, and they share the best and worst features of each.

Their hands-on nature makes them more engaging than traditional documentation.

**Static analysis** Static analysis tools are a powerful way to share best practices that can be checked programmatically.

Every programming language has its own particular static analysis tools, but they have the same general purpose: to alert code authors and reviewers to ways in which code can be improved to follow style and best practices.

Some tools go one step further and offer to automatically apply those improvements to the code.

NOTE

Setting up static analysis tools requires an investment, but as soon as they are in place, they scale efficiently. When a check for a best practice is added to a tool, every engineer using that tool becomes aware of that best practice.

This also frees up engineers to teach other things: the time and effort that would have gone into manually teaching the (now automated) best practice can instead be used to teach something else. Static analysis tools augment engineers’ knowledge. They enable an organization to apply more best practices and apply them more consistently than would otherwise be possible.

Staying in the Loop

Some information is critical to do one’s job, such as knowing how to do a typical development workflow. Other information, such as updates on popular productivity tools, is less critical but still useful.

For this type of knowledge, the formality of the information sharing medium depends on the importance of the information being delivered. For example, users expect official documentation to be kept up to date, but typically have no such expectation for newsletter content, which therefore requires less maintenance and upkeep from the owner.

**Newsletters**

These are a good way to communicate information that is of interest to engineers but isn’t mission critical.

Even though most Google newsletters are sent via email, some are more creative in their distribution.

NOTE

Communities

Googlers like to form cross-organizational communities around various topics to share knowledge.

These open channels make it easier to learn from others outside your immediate circle and avoid information islands and duplication.

Google Groups are especially popular: Google has thousands of internal groups with varying levels of formality. Some are dedicated to troubleshooting; others, like the Code Health group, are more for discussion and guidance.

Internal Google+ is also popular among Googlers as a source of informal information because people will post interesting technical breakdowns or details about projects they are working on.

Readability: Standardized Mentorship through code review

**Readability** is a standardized, Google-wide mentorship process for disseminating programming language best practices.

Readability covers a wide expertise, including language idioms, code structure, API design, appropriate use of common libraries, documentation, and test coverage.

So many engineers found the process valuable that they volunteered their own time to scale the program.

What Is the Readability Process?

Code review is mandatory at Google. Every changelist (CL) requires readability approval, which indicates that someone who has readability certification for that language has approved the CL.

Certified authors implicitly provide readability approval of their own CLs; otherwise, one or more qualified reviewers must explicitly give readability approval for the CL.

NOTE

Within Google, having readability they consistently write clear, idiomatic, and maintainable code that exemplifies Google’s best practices and coding style for a given language.

They do this by submitting CLs through the readability process, during which a centralized group of readability reviewers review the CLs and give feedback on how much it demonstrates the various areas of mastery.

As authors internalize the readability guidelines, they receive fewer and fewer comments on their CLs until they eventually graduate from the process and formally receive readability.

engineers with readability are trusted . and to act as reviewers for other engineers’ code.

Readability reviewers are held to the highest standards because they are expected not just to have deep language expertise, but also an aptitude for teaching through code review.

They are expected to treat readability as first and foremost a mentoring and cooperative process.

Readability reviewers and CL authors alike are encouraged to have discussions during the review process. Reviewers provide relevant citations for their comments so that authors can learn about the rationales that went into the style guidelines.

If the rationale for any given guideline is unclear, authors should ask for clarification (“ask questions”).

Readability is deliberately a human-driven process that aims to scale knowledge in a standardized yet personalized way. As a complementary blend of written and tribal knowledge, readability combines the advantages of written documentation, which can be accessed with citable references, with the advantages of expert human reviewers, who know which guidelines to cite.

Canonical guidelines and language recommendations are comprehensively documented. corpus of information is so large that it can be overwhelming, especially to newcomers.

Why Have This Process?

Code is read far more than it is written, and this effect is magnified at Google’s scale and in our (very large)

An important feature of documented best practices is that they provide consistent standards for all Google code to follow. Readability is both an enforcement and propagation mechanism for these standards.

One of the primary advantages of the readability program is that it exposes engineers to more than just their own team’s tribal knowledge.

To earn readability in a given language, engineers must send CLs through a centralized set of readability reviewers who review code across the entire company.

readability is a heavyweight process compared to other mediums like documentation and classes because it is mandatory and enforced by Google tooling. These costs are nontrivial and include the following:

As we continue to invest in static analysis, readability reviewers can increasingly focus on higher-order areas, like whether a particular block of code is understandable by outside readers who are not intimately familiar with the codebase instead of automatable detections like whether a line has trailing whitespace.

But aspirations aren’t enough. Readability is a controversial program: some engineers complain that it’s an unnecessary bureaucratic hurdle and a poor use of engineer time. Are readability’s trade-offs worthwhile? For the answer, we turned to our trusty Engineering Productivity Research (EPR) team.

The EPR team performed in-depth studies of readability, including but not limited to whether people were hindered by the process, learned anything, or changed their behaviour after graduating. These studies showed that readability has a net positive impact on engineering velocity.

CLs by authors with readability take statistically significantly less time to review and submit than CLs by authors who do not have readability. Self-reported engineer satisfaction with their code quality—lacking more objective measures for code quality—is higher among engineers who have readability versus those who do not. A significant majority of engineers who complete the program report satisfaction with the process and find it worthwhile.

They report learning from reviewers and changing their own behaviour to avoid readability issues when writing and reviewing code.

NOTE

Google has a very strong culture of code review, and readability is a natural extension of that culture. Readability grew from the passion of a single engineer to a formal program of human experts mentoring all Google engineers. It evolved and changed with Google’s growth, and it will continue to evolve as Google’s needs change.