CHAPTER 3

Knowledge Sharing

第三章 知識共享

Written by Nina Chen and Mark Barolak

Edited by Riona MacNamara

Your organization understands your problem domain better than some random person on the internet; your organization should be able to answer most of its own questions. To achieve that, you need both experts who know the answers to those questions and mechanisms to distribute their knowledge, which is what we'll explore in this chapter. These mechanisms range from the utterly simple (Ask questions; Write down what you know) to the much more structured, such as tutorials and classes. Most importantly, however, your organization needs a culture of learning, and that requires creating the psychological safety that permits people to admit to a lack of knowledge.

你的組織對你問題領域的理解比網際網路上的一些隨機的人要好；你的組織應該能解答你的大部分問題。要做到這一點，你需要知道解決問題答案的專家在哪裡和傳播知識的機制，這就是我們將在本章中探討的。這些機制的範圍很廣，從完全簡單的（提問；寫下你所知道的）到系統化，如課程和課程。然而，最重要的是，你的組織需要一種學習文化，這需要創造一種心理上的安全感，允許人們承認自己缺乏知識。

Challenges to Learning 學習的挑戰

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*
  An environment in which people are afraid to take risks or make mistakes in front of others because they fear being punished for it. This often manifests as a culture of fear or a tendency to avoid transparency.
- *Information islands*
  Knowledge fragmentation that occurs in different parts of an organization that don't communicate with one another or use shared resources. In such anenvironment, each group develops its own way of doing things.[1] This often leads to the following:
  - **Information fragmentation**
    Each island has an incomplete picture of the bigger whole.
  - **Information duplication**
    Each island has reinvented its own way of doing something.

    - **Information 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 that occurs when critical information is available from only a single person. This is related to bus factor, which is discussed in more detail in Chapter 2.
  SPOFs can arise out of good intentions: it can be easy to fall into a habit of “Let me take care of that for you.” But this approach optimizes for short-term efficiency (“It’s faster for me to do it”) at the cost of poor long-term scalability (the team never learns how to do whatever it is that needs to be done). This mindset also tends to lead to all-or-nothing expertise.
- *All-or-nothing expertise*
  A group of people that is split between people who know “everything” and novices, with little middle ground. This problem often reinforces itself if experts always do everything themselves and don’t take the time to develop new experts through mentoring or documentation. In this scenario, knowledge and responsibilities continue to accumulate on those who already have expertise, and new team members or novices are left to fend for themselves and ramp up more slowly.
- *Parroting*
  Mimicry without understanding. This is typically characterized by mindlessly copying patterns or code without understanding their purpose, often under the assumption that said code is needed for unknown reasons.
- *Haunted graveyards*
  Places, often in code, that people avoid touching or changing because they are afraid that something might go wrong. Unlike the aforementioned parroting, haunted graveyards are characterized by people avoiding action because of fear and superstition.

在一個組織內共享專業知識並非易事。沒有強大的學習文化，挑戰隨時出現。Google 經歷了許多這樣的挑戰，尤其是隨著公司規模的擴大：

- *缺乏安全感*
  一個環境中，人們不敢在別人面前冒險或犯錯，因為他們害怕因此受到懲罰。這通常表現為一種恐懼文化或避免透明的傾向。
- *資訊孤島*
  在一個組織的不同部分發生的知識碎片，這些部分沒有相互溝通或使用共享資源。在這樣的環境中，每個小組都形成了自己的做事方式。這往往導致以下情況：
    - **資訊碎片化**
      每個孤島對整體都有一個不完整的描述。
    - **資訊重複**
      每個孤島都重新發明了自己的做事方式。
    - **資訊偏移**
      每個孤島都有自己做同一件事的方法，這些方法在一起協作可能會或可能不會發生衝突。
- *單點故障（SPOF）*
  當關鍵資訊只能從一個人那裡獲得時，就會出現瓶頸。這與巴士因子有關，在第二章有詳細討論。
  SPOF 可能是出於良好的意圖：我們很容易陷入 "讓我來幫你解決 "的習慣。但這種方法提高了短期效率（"我做起來更快"），但代價是長期可擴充性差（團隊從未學會如何做需要做的事）。這種心態也往往導致失敗，組員要麼全會或要麼都

不會某方面的知識。

- *要麼全會要麼都不會*
  一群人被分成了 "什麼都懂 "的老人和什麼都不會的新手，幾乎沒有中間地帶。如果專家總是自己做所有的事情，而不花時間透過指導或編寫文件來培養新的專家，這個問題往往會加劇。在這種情況下，知識和責任繼續在那些已經擁有專業知識的人身上積累，而新的團隊成員或新手則只能自生自滅，提升速度更慢。
- *鸚鵡學舌*
  模仿而不理解。這典型的特徵是在不瞭解其目的的情況下無意識地複製模式或程式碼，通常是在假設上述程式碼是出於未知原因而需要的情況下。
- *鬧鬼墓地*
  人們避免接觸或改變的地方，通常在程式碼中，因為他們擔心會出問題。與前面提到的鸚鵡學舌不同，鬧鬼墓地的特點是人們因為恐懼和迷信而避免行動。

In the rest of this chapter, we dive into strategies that Google's engineering organizations have found to be successful in addressing these challenges.

在本章的其餘部分，我們將深入探討 Google 的工程組織在應對這些挑戰方面成功的策略。

> 1 換句話說，我們沒有形成一個單一的全球最大值，而是有一堆的區域性最大值。

Philosophy 理念

Software engineering can be defined as the multiperson development of multiversion programs.[2] People are at the core of software engineering: code is an important output but only a small part of building a product. Crucially, code does not emerge spontaneously out of nothing, and neither does expertise. Every expert was once a novice: an organization's success depends on growing and investing in its people.

軟體工程可以定義為多人協作開發多版本程式。人是軟體工程的核心：程式碼是重要的產出，但只是建構產品的一小部分。至關重要的是，程式碼不是憑空出現的，專業知識也不會憑空出現。每個專家都曾經是菜鳥：一個組織的成功取決於其員工的成長和投入。

Personalized, one-to-one advice from an expert is always invaluable. Different team members have different areas of expertise, and so the best teammate to ask for any given question will vary. But if the expert goes on vacation or switches teams, the team can be left in the lurch. And although one person might be able to provide personalized help for one-to-many, this doesn't scale and is limited to small numbers of "many."

來自專家的個性化、一對一的建議總是寶貴的。不同的團隊的成員有不同的專業領域，因此對於任何給定的問題，最佳的隊友都會有所不同解法。但如果專家休假或調換團隊，原團隊可能會陷入困境。儘管一個人可以為一對多人提供個性化的幫助，但這並不具有規模，只限於少量的 "多"。

Documented knowledge, on the other hand, can better scale not just to the team but to the entire organization. Mechanisms such as a team wiki enable many authors to share their expertise with a larger group. But even though written documentation is more scalable than one-to-one conversations, that scalability comes with some 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.

另一方面，文件化的知識不僅可以更好地擴充到團隊，還可以擴充到整個組織。團隊 wiki 等機制使許多作者能夠與更大的團隊分享他們的專業知識。但是，儘管書面文件比一對一的對話更具可擴充性，但這種可擴充性也帶來了一些代價：它可能更具普遍性，不太適用於個別學習者的情況，而且隨著時間的推移，還需要額外的維護成本來，保持資訊的相關性和即時性。

Tribal knowledge exists in the gap between what individual team members know and what is documented. Human experts know these things that aren't written down. If we document that knowledge and maintain it, it is now available not only to somebody with direct one-to-one access to the expert today, but to anybody who can find and view the documentation.

內部知識存在於單個團隊成員所知道的和被記錄下來的東西之間的差距。人類專家知道這些沒有寫下來的東西。如果我們把這些知識記錄下來並加以維護，那麼現在不僅可以讓今天的專家一對一地直接接觸到這些知識，而且可以讓任何能夠找到並檢視這些檔案的人獲得這些知識。

> `tribal knowledge`：內部知識；是指一種僅存在於某個部落中的資訊或知識， 這些知識不為外界所知，沒有正式記 錄， 只能口口相傳。

So in a magical world in which everything is always perfectly and immediately documented, we wouldn't need to consult a person any more, right? Not quite. Written knowledge has scaling advantages, but so does targeted human help. A human expert can synthesize their expanse of knowledge. They can assess what information is applicable to the individual's use case, determine whether the documentation is still relevant, and know where to find it. Or, if they don't know where to find the answers, they might know who does.

因此，在一個所有事情總是完美地、立即地被記錄下來的神奇世界中，我們就不需要再諮詢一個人了，對吧？並非如此。書面知識具有擴充優勢，但有針對性的人力投入也具有擴充優勢。人類專家可以利用他們廣博的知識。他們可以評估哪些資訊適用於個人的使用案例，確定檔案是否仍然相關，並知道在哪裡可以找到它。或者，如果他們不知道在哪裡可以找到解答，他們知道誰可以解決。

> 2 David Lorge Parnas, 軟體工程。多人開發多版本程式 (Heidelberg: Springer-Verlag Berlin, 2011).

Tribal and written knowledge complement each other. Even a perfectly expert team with perfect documentation needs to communicate with one another, coordinate with other teams, and adapt their strategies over time. No single knowledge-sharing approach is the correct solution for all types of learning, and the particulars of a good mix will likely vary based on your organization. Institutional knowledge evolves over time, and the knowledge-sharing methods that work best for your organization will likely change as it grows. Train, focus on learning and growth, and build your own stable of experts: there is no such thing as too much engineering expertise.

內部知識和書面知識相互補充。即使是一個擁有完美文件的專家團隊也需要相互溝通，與其他團隊協調，並隨著時間的推移不斷調整他們的策略。沒有任何一個單一的知識共享方法對於所有型別學習而言都是正確的解決方案，最佳組合的具體內容會根據你的組織而有所不同。團隊知識隨著時間的推移而演變，對你的組織最有效的知識共享方法可能會隨著組織的發展而改變。培訓，專注於學習和成長，並建立自己穩定的專家隊伍：沒有太多的工程專業知識。

Setting the Stage: Psychological Safety 搭建舞臺：心理安全

Psychological safety is critical to promoting a learning environment.

心理安全是促進學習環境的關鍵。

To learn, you must first acknowledge that there are things you don't understand. We should welcome such honesty rather than punish it. (Google does this pretty well, but sometimes engineers are reluctant to admit they don't understand something.)

要學習，你必須首先承認有些事情你不明白。我們應該歡迎這種誠實，而不是懲罰它。（Google 在這方面做得很好，但有時工程師不願意承認他們不懂一些東西。）

An enormous part of learning is being able to try things and feeling safe to fail. In a healthy environment, people feel comfortable asking questions, being wrong, and learning new things. This is a baseline expectation for all Google teams; indeed, our research has shown that psychological safety is the most important part of an effective team.

學習的一個重要部分是能夠嘗試事情，並感覺到失敗的無責。在一個健康的環境中，人們對提出問題、犯錯和學習新事物感到自在。這是所有 Google 團隊的基本期望；事實上，我們的研究表明，心理安全是有效團隊最重要的組成部分。

Mentorship 導師制

At Google, we try to set the tone as soon as a "Noogler" (new Googler) engineer joins the company. One important way of building psychological safety is to assign Nooglers a mentor—someone who is not their team member, manager, or tech lead—whose responsibilities explicitly include answering questions and helping the Noogler ramp up. Having an officially assigned mentor to ask for help makes it easier for the newcomer and means that they don't need to worry about taking up too much of their coworkers' time.

在 Google，我們嘗試在 "Noogler"（新的 Googler）工程師加入公司時就確定基調。建立心理安全的一個重要方法是為 Noogler 分配一個導師——一個不是他們的團隊成員、經理或技術負責人的人——其職責明確包括回答問題和幫助 Noogler 成長。有一個官方指定的導師可以尋求幫助，這對新人來說更容易，也意味著他們不需要擔心會佔用同事太多的時間。

A mentor is a volunteer who has been at Google for more than a year and who is available to advise on anything from using Google infrastructure to navigating Google culture. Crucially, the mentor is there to be a safety net to talk to if the mentee doesn't know whom else to ask for advice. This mentor is not on the same team as the mentee, which can make the mentee feel more comfortable about asking for help in tricky situations.

導師是在 Google 工作了一年以上的志願者，他可以就使用 Google 基礎設施和了解 Google 文化等方面提供建議。最重要的是，如果被指導者不知道該向誰尋求建議，指導者就會成為一個安全網。這位導師與被指導者不在同一個團隊，這可以使被指導者在棘手的情況下更放心地尋求幫助。

Mentorship formalizes and facilitates learning, but learning itself is an ongoing process. There will always be opportunities for coworkers to learn from one another, whether it's a new employee joining the organization or an experienced engineer learning a new technology. With a healthy team, teammates will be open not just to answering but also to asking questions: showing that they don't know something and learning from one another.

導師制使學習正規化並促進學習，但學習本身是一個持續的過程。無論是加入組織的新員工還是學習新技術的有經驗的工程師，同事之間總是有機會互相學習。在一個健康的團隊中，隊友們不僅願意回答問題，也願意提出問題：表明他們不知道的東西，並相互學習。

Psychological Safety in Large Groups 大團體的心理安全

Asking a nearby teammate for help is easier than approaching a large group of mostly strangers. But as we've seen, one-to-one solutions don't scale well. Group solutions are more scalable, but they are also scarier. It can be intimidating for novices to form a question and ask it of a large group, knowing that their question might be archived for many years. The need for psychological safety is amplified in large groups. Every member of the group has a role to play in creating and maintaining a safe environment that ensures that newcomers are confident asking questions and up-and- coming experts feel empowered to help those newcomers without the fear of having their answers attacked by established experts.

向附近的隊友尋求幫助比接近一大群大多是陌生的人容易得多。但正如我們所看到的，一對一的解決方案並不能很好地擴充。對於新手來說，出現一個問題並向一大團隊人提問是一種威脅，因為他們知道自己的問題可能會存在多年。對心理安全的需求在大團隊中被放大了。小組的每個成員都應在創造和維持一個安全的環境中發揮作用，以確保新人自信提出問題，而新晉專家則感到有能力幫助這些新人，而不必擔心他們的答案會受到老專家的攻擊。

The most important way to achieve this safe and welcoming environment is for group interactions to be cooperative, not adversarial. Table 3-1 lists some examples of recommended patterns (and their corresponding antipatterns) of group interactions.

實現這種安全和受歡迎的環境的最重要的方法是團體互動是合作性的，而不是對抗性的。表 3-1 列出了一些推薦的團體互動模式（以及相應的反模式）的例子。

Table 3-1. Group interaction patterns

Recommended patterns (cooperative)	Antipatterns (adversarial)
Basic questions or mistakes are guided in the proper direction	Basic questions or mistakes are picked on, and the person asking the question is chastised
Explanations are given with the intent of helping the person asking the question learn	Explanations are given with the intent of showing off one's own knowledge
Responses are kind, patient, and helpful	Responses are condescending, snarky, and unconstructive
Interactions are shared discussions for finding solutions	Interactions are arguments with "winners" and "losers"

Table 3-1. 團隊互動模式

推薦的模式（合作型）	反模式(對抗型)
基本的問題或錯誤被引導到正確的方向	基本的問題或錯誤被挑剔，提出問題的人被責備
解釋的目的是為了幫助提問的人學習	解釋的目的是為了炫耀自己的知識
回應親切、耐心、樂於助人	回應是居高臨下、尖酸刻薄、毫無建設性的
互動是為尋找解決方案而進行的共同討論	"互動是有 "贏家 "和 "輸家 "的爭論

These antipatterns can emerge unintentionally: someone might be trying to be helpful but is accidentally condescending and unwelcoming. We find the Recurse Center's social rules to be helpful here:

- *No feigned surprise ("What?! I can't believe you don't know what the stack is!")*
  Feigned surprise is a barrier to psychological safety and makes members of the group afraid of admitting to a lack of knowledge.
- *No "well-actuallys"*
  Pedantic corrections that tend to be about grandstanding rather than precision.
- *No back-seat driving*
  Interrupting an existing discussion to offer opinions without committing to the conversation.
- *No subtle "-isms" ("It's so easy my grandmother could do it!")*
  Small expressions of bias (racism, ageism, homophobia) that can make individuals feel unwelcome, disrespected, or unsafe.

這些反模式可能是無意中出現的：有人可能是想幫忙，但卻意外地居高臨下，不受歡迎。我們發現 Recurse 中心的社交規則在這裡很有幫助：

- *不要假裝驚訝（"什麼？ 我不相信你不知道堆疊是什麼！"）*
  假裝驚訝是心理安全的障礙，使團體成員害怕承認自己缺乏知識。
- *不根據事實*
  迂腐的糾正，往往是為了譁眾取寵而非糾正。
- *不開小會*
  打斷現有的討論，提供意見，而不投入到對話中。
- *不說微妙的謊言（"這太容易了，我奶奶都能做！"）*
  小小的偏見表達（種族主義、年齡歧視、恐同症），會使個人感到不受歡迎、不被尊重或不安全。

Growing Your Knowledge 增長你的知識

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.

知識共享從自己開始。重要的是要認識到，你總是有東西要學。下面的準則可以讓你增加自己的個人知識。

Ask Questions 提問

If you take away only a single thing from this chapter, it is this: always be learning; always be asking questions.

如果你從這一章中只帶走一件事，那就是：永遠學習；保持好奇。

We tell Nooglers that ramping up can take around six months. This extended period is necessary to ramp up on Google's large, complex infrastructure, but it also reinforces the idea that 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." "I just need to try harder before I ask anyone for help," you think. Don't fall into this trap! Your coworkers are often the best source of information: leverage this valuable resource.

我們告訴 Nooglers，提升可能需要 6 個月左右。這個時間的延長對於在 Google 龐大而複雜的基礎設施上的提升是必要的，但它也強化了學習是一個持續、迭代的過程的理念。初學者犯的最大錯誤之一是在遇到困難時不尋求幫助。你可能會想獨自掙扎一下，或者感到害怕你的問題 "太簡單了"。"你想："我只是需要在向別人尋求幫助之前更努力地一下。不要落入這個陷阱! 你的同事往往是最好的資訊來源：利用這一寶貴資源。

There is no magical day when you suddenly always know exactly what to do in every situation—there's always more to learn. Engineers who have been at Google for years still have areas in which they don't feel like they know what they are doing, and that's OK! Don't be afraid to say "I don't know what that is; could you explain it?" Embrace not knowing things as an area of opportunity rather than one to fear.[3]

不會有神奇的一天，你突然總是確切地知道在任何情況下該怎麼做——總是有更多的東西需要學。在 Google 工作多年的工程師們仍然有一些領域他們覺得自己不知道自己該怎麼做，這沒關係！不要害怕說 "我不知道那是什麼，你能解釋一下嗎？"。把不知道事情當作瞭解新領域的機會，而不是一個恐懼這個未知領域。

It doesn't matter whether you're new to a team or a senior leader: you should always be in an environment in which there's something to learn. If not, you stagnate (and should find a new environment).

不管你是新加入的團隊還是高階領導者：你應該始終處在一個有東西可學的環境中。如果不是這樣，你就會停滯不前（應該找一個新的環境）。

It's especially critical for those in leadership roles to model this behavior: 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. Openly asking questions[4] or expressing gaps in knowledge reinforces that it's OK for others to do the same.

對於那些擔任領導角色的人來說，塑造這種行為尤為重要：重要的是不要錯誤地將 "資歷 "等同於 "無所不知"。事實上，你知道的越多，你知道你不知道的就越多。公開提問或表達知識差距，強化了其他人也可以這樣做。

On the receiving end, patience and kindness when answering questions fosters an environment in which people feel safe looking for help. Making it easier to overcome the initial hesitation to ask a question sets the tone early: reach out to solicit questions, and make it easy for even “trivial” questions to get an answer. Although engineers could probably figure out tribal knowledge on their own, they’re not here to work in a vacuum. Targeted help allows engineers to be productive faster, which in turn makes their entire team more productive.

在接受端，回答時的耐心和善意培養人們安心地尋求幫助的環境。讓人們更容易克服最初對提問的猶豫不決，儘早定下基調：主動徵求問題，讓即使是“瑣碎”的問題也能輕鬆得到答案。雖然工程師們可能會自己摸索出內部知識，但他們不是獨自在工作。有針對性的幫助可以讓工程師更快地提高工作效率，從而使整個團隊的工作效率更高。

> 3 冒名頂替綜合症在成功人士中並不少見，Google 也不例外。事實上，本書的大多數作者都患有冒名頂替綜合症。我們承認，對於冒名頂替綜合徵患者來說，對失敗的恐懼可能很難，並且會強化他們避免分道揚鑣的傾向。
>
> 4 見 "如何提出好問題"。

Understand Context 瞭解背景

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?

學習不僅僅是瞭解新事物；它還包括對現有事物的設計和實施背後的決策的理解。假設你的團隊繼承了一個已經存在多年的關鍵基礎設施的遺留程式碼庫。原作者早就不在了，程式碼也很難理解。與其花時間學習現有的程式碼，不如從頭開始重寫，這很有誘惑力。但是，不要想著“我不明白”並在那裡結束你的想法，而是深入思考：你應該問什麼問題？

Consider the principle of “Chesterson’s fence”: before removing or changing something, first understand why it’s there.
In the matter of reforming things, as distinct from deforming them, there is one plain and simple principle; a principle which will probably be called a paradox. There exists in such a case a certain institution or law; let us say, for the sake of simplicity, a fence or gate erected across a road. The more modern type of reformer goes gaily up to it and says, “I don’t see the use of this; let us clear it away.” To which the more intelligent type of reformer will do well to answer: “If you don’t see the use of it, I certainly won’t let you clear it away. Go away and think. Then, when you can come back and tell me that you do see the use of it, I may allow you to destroy it.”

考慮一下 "Chesterson's fence "的原則：在移除或改變某些東西之前，首先要了解它為什麼存在。
在改造事物的問題上，不同於使事物變形，有一個簡單明瞭的原則；這個原則可能會被稱為悖論。在這種情況下，存在著某種制度或法律；為了簡單起見，讓我們說，在一條道路上豎起了柵欄或大門。更現代的改革者興高采烈地走到它面前，說："我看不出來這有什麼用；讓我們把它清除掉吧。" 對此，更聰明的改革者會很好地回答。"如果你看不到它的用途，我當然不會讓你清除它。走吧，好好想想。然後，當你能回來告訴我你確實看到了它的用途時，我才會允許你銷燬它。"

This doesn't mean that code can't lack clarity or that existing design patterns can't be wrong, but engineers have a tendency to reach for "this is bad!" far more quickly than is often warranted, especially for unfamiliar code, languages, or paradigms. Google is not immune to this. 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.

這並不意味著程式碼不可能缺乏清晰，也不意味著現有的設計模式不可能是錯誤的，但工程師們有一種傾向，比起程式碼存在的正當理由，反而容易想到這段程式碼 "很糟糕！"，特別是對於不熟悉的程式碼、語言或範例。Google 也不能倖免。尋找和了解背景，特別是對於那些看起來不尋常的決定。在你瞭解了程式碼的背景和目的之後，考慮你的改變是否仍然有意義。如果有意義，就繼續做；如果沒有意義，就為未來的繼任者記錄下你的理由。

Many Google style guides explicitly include context to help readers understand the rationale behind the style guidelines instead of just memorizing a list of arbitrary rules. More subtly, 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. See Chapter 8.

許多 Google 風格指南明確地包括背景，以幫助讀者理解風格指南背後的理由，而不是僅僅記住一串武斷的規則。更微妙的是，瞭解某條準則背後的理由，可以讓作者做出明智的決定，知道該準則何時不適用，或者該準則是否需要更新。見第 8 章。

Scaling Your Questions: Ask the Community 擴充你的問題：向社群提問

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 favor: when you learn something from a one-to-one discussion, write it down.

獲得一對一的幫助是高頻寬的，但規模必然有限。而作為一個學習者，要記住每一個細節很困難。幫你未來的自己一個忙：當你從一對一的討論中學到一些東西時，把它寫下來。

Chances are that future newcomers will have the same questions you had. Do them a favor, too, and share what you write down.

未來的新來者可能會有和你一樣的問題。也幫他們一個忙，分享你寫下的東西。

Although sharing the answers you receive can be useful, it's also beneficial to seek help not from individuals but from the greater community. In this section, we examine different forms of community-based learning. Each of these approaches—group chats, mailing lists, and question-and-answer systems—have different trade-offs and complement one another. But each of them enables the knowledge seeker to get help from a broader community of peers and experts and also ensures that answers are broadly available to current and future members of that community.

儘管分享你得到的答案可能是有用的，但尋求更多社群而非個人的幫助也是有益的。在本節中，我們將研究不同形式的社群學習。這些方法中的每一種——群聊、郵件列表和問答系統——都有不同的權衡，並相互補充。但它們中的每一種都能使知識尋求者從更廣泛的同行和專家社群獲得幫助，同時也能確保該社群的當前和未來成員都能廣泛獲得答案。

Group Chats 群聊

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.

當你有一個問題時，有時很難從適合的人那裡得到幫助。也許你不確定誰知道答案，或者你想問的人很忙。在這種情況下，群聊是很好的方式，因為你可以同時向許多人提出你的問題，並與任何有空的人進行快速的對話。另外，群聊中的其他成員可以從問題和答案中學習，而且許多形式的群聊可以自動存檔並在以後進行搜尋。

Group chats tend to be devoted either to topics or to teams. Topic-driven group chats are typically open so that anyone can drop in to ask a question. They tend to attract experts and can grow quite large, so questions are usually answered quickly. Team- oriented chats, on the other hand, tend to be smaller and restrict membership. As a result, they might not have the same reach as a topic-driven chat, but their smaller size can feel safer to a newcomer.

群聊往往是專門針對主題或團隊。以主題為導向的群聊通常是開放的，因此任何人都可以進來問問題。他們傾向於吸引專家，並且可以發展得相當大，所以問題通常會很快得到回答。另一方面，以團隊為導向的聊天，往往規模較小，並限制成員。因此，他們可能沒有話題驅動型聊天的影響力，但其較小的規模會讓新人感到更安心。

Although group chats are great for quick questions, they don't provide much structure, which can make it difficult to extract meaningful information from a conversation in which you're not actively involved. 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.

雖然小組聊天對快速提問很有幫助，但它們沒有提供太多的結構化，這會使你很難從一個你沒有積極參與的對話中提取有意義的資訊。一旦你需要在群組之外分享資訊，或使其可在以後參考，你應該寫一份文件或給郵件列表發郵件。

Mailing Lists 郵件列表

Most topics at Google have a topic-users@ or topic-discuss@ Google Groups mailing list that anyone at the company can join or email. Asking a question on a public mailing list is a lot like asking a group chat: the question reaches a lot of people who could potentially answer it and anyone following the list can learn from the answer. Unlike group chats, though, public mailing lists are easy to share with a wider audience: they are packaged into searchable archives, and email threads provide more structure than group chats. At Google, mailing lists are also indexed and can be discovered by Moma, Google's intranet search engine.

Google 的大多數主題都有一個 topic-users@或 topic-discuss@的 Google 群組郵件列表，公司的任何人都可以加入或傳送電子郵件。在公共郵件列表上提出問題，很像在群組聊天中提出問題：這個問題可以傳到很多有可能回答它的人那兒，而且任何關注這個列表的人都可以從答案中學習。但與群聊不同的是，公共郵件列表很容易與更多人分享：它們被打包成可搜尋的檔案，而且電子郵件比群聊提供更多的結構化。在 Google，郵件列表也被編入索引，可以被 Google 的內部網搜尋引擎 Moma 發現。

When you find an answer to a question you asked on a mailing list, it can be tempting to get on with your work. Don't do it! You never know when someone will need the same information in the future, so it's a best practice to post the answer back to the list.

當你找到了你在郵件列表中提出的問題的答案時，你可能會很想繼續你的工作。請不要這樣做! 你永遠不知道將來什麼時候會有人需要同樣的資訊，所以最好將答案發回郵件列表。

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. A thread about a particular problem is generally most useful while it is active. 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. Additionally, the signal-to-noise ratio can be lower than other mediums like formal documentation because the problem that someone is having with their specific workflow might not be applicable to you.

郵件列表也有其不足之處。它們很適合處理需要大量背景資料的複雜問題，但對於小組聊天所擅長的快速來回交流來說，它們就顯得很笨拙。一個關於特定問題的線索通常在它活躍的時候是最有用的。電子郵件檔案是不可改變的，而且很難確定在一箇舊的討論主題中發現的答案是否仍然對今天的情況有效。此外，信噪比可能比其他媒介（如正式檔案）低，因為某人在其特定工作流程中遇到的問題可能並不適用於你。

Email at Google Google 的電子郵件

Google culture is infamously email-centric and email-heavy. Google engineers receive hundreds of emails (if not more) each day, with varying degrees of actionability. Nooglers can spend days just setting up email filters to deal with the volume of notifications coming from groups that they've been autosubscribed to; some people just give up and don't try to keep up with the flow. 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.

Google 的文化是臭名昭著的以電子郵件為中心和重度使用電子郵件。Google 的工程師們每天都會收到數以百計的電子郵件（如果不是更多的話），其中有不同程度的可操作性。新手們需要花好幾天時間來設定電子郵件過濾器，以處理來自他們自動訂閱的群組的大量通知；有些人乾脆放棄了，放棄追隨最新的郵件。一些群組將大型郵件列表預設為每一個討論，而不試圖將資訊定向傳送給那些可能對其特別感興趣的人；結果，信噪比成為了一個真正的問題。

Google tends toward email-based workflows by default. This isn't necessarily because email is a better medium than other communications options—it often isn't—rather, it's because that's what our culture is accustomed to. Keep this in mind as your organization considers what forms of communication to encourage or invest in.

Google 預設傾向於基於電子郵件的工作流程。這並不一定是因為電子郵件是一個比其他通訊選項更好的媒介——它往往不是——而是因為這是我們的文化所習慣的。當你的組織考慮要鼓勵或投入什麼形式的溝通時，請記住這一點。

YAQS: Question-and-Answer Platform YAQS：問答平台

YAQS (“Yet Another Question System”) is a Google-internal version of a Stack Overflow–like website, making it easy for Googlers to link to existing or work-in-progress code as well as discuss confidential information.

YAQS（"另一個問題系統"）是 Google 內部版本的Stack Overflow——類似網站，使 Googlers 能夠輕鬆地連結到現有或正在進行的程式碼，以及討論機密資訊。

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.

像 Stack Overflow 一樣，YAQS 與郵件列表有許多相同的優點，並增加了完善的功能：標記為有用的答案在使用者介面上被推廣，使用者可以編輯問題和答案，以便隨著程式碼和事實的變化保持準確和有用。因此，一些郵件列表已經被 YAQS 所取代，而其他的郵件列表已經演變成更一般的討論列表，不再專注於解決問題。

Scaling Your Knowledge:You Always Have Something to Teach 擴充你的知識：你總是有東西可教的

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 (see Chapter 4). Google engineers teach others in a variety of ways, such as office hours, giving tech talks, teaching classes, writing documentation, and reviewing code.

教學不侷限專家，專業知識也不是一種二元狀態，你要麼是新手，要麼是專家。專業知識是你所知道的一個多維向量：每個人在不同領域都有不同水平的專業知識。這就是為什麼多樣性是組織的成功至關重要的原因之一：不同的人帶來不同的觀點和專業知識（見第四章）。Google 工程師以各種方式教授他人，如辦公時間、舉辦技術講座、教授課程、編寫文件和審查程式碼。

Office Hours 固定時間

Sometimes it’s really 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 (typically weekly) 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.

有時與人交談非常重要，在這些情況下，固定時間是一個很好的解決方案。固定時間是一個定期安排的活動（通常是每週一次），在此期間，一個或多個人可以回答關於某個特定主題的問題。固定時間幾乎從來不是知識共享的首選：如果你有一個緊急的問題，等待下一次會議的答案可能會很痛苦；如果你主持固定時間，它們會佔用時間，需要定期宣傳。也就是說，它們確實為人們提供了一種與專家當面交談的方式。如果問題還很模糊，工程師還不知道該問什麼問題（比如他們剛開始設計一個新的服務），或者問題是關於一個非常專業的東西，以至於沒有相關的文件，那麼這就特別有用。

Tech Talks and Classes 技術講座和課程

Google has a robust culture of both internal and external[^5] 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. At a more grassroots level, our g2g (Googler2Googler) program lets Googlers sign up to give or attend talks and classes from fellow Googlers. [5] The program is wildly successful, with thousands of participating Googlers teaching topics from the technical (e.g., “Understanding Vectorization in Modern CPUs”) to the just-for-fun (e.g., “Beginner Swing Dance”).

Google 擁有強大的內部和外部技術講座和課程的文化。我們的 engEDU（工程教育）團隊專注於為許多受眾提供計算機科學教育，包括 Google 工程師和世界各地的學生。在更底層的層面上，我們的 g2g（Googler2Googler）計劃讓 Googlers 報名參加，以舉辦或參加 Googlers 同伴的講座和課程。該計劃非常成功，有數千名 Googlers 參與，教授的主題從技術（如 "瞭解現代 CPU 的向量化"）到只是為了好玩（如 "初級搖擺舞"）。

Tech talks typically consist of a speaker presenting directly to an audience. Classes, on the other hand, can have a lecture component but often center on in-class exercises and therefore require more active participation from attendees. As a result, instructor-led classes are typically more demanding and expensive to create and maintain than tech talks and are reserved for the most important or difficult topics. That said, after a class has been created, it can be scaled relatively easily because many instructors can teach a class from the same course materials. We’ve found that classes tend to work best when the following circumstances exist:

- The topic is complicated enough that it’s a frequent source of misunderstanding. Classes take a lot of work to create, so they should be developed only when they’re addressing specific needs.
- The topic is relatively stable. Updating class materials is a lot of work, so if the subject is rapidly evolving, other forms of sharing knowledge will have a better bang for the buck.
- The topic benefits from having teachers available to answer questions and provide personalized help. If students can easily learn without directed help, self- serve mediums like documentation or recordings are more efficient. A number of introductory classes at Google also have self-study versions.
- There is enough demand to offer the class regularly. Otherwise, potential learners will get the information they need in other ways rather than waiting for the class. At Google, this is particularly a problem for small, geographically remote offices.

技術講座通常由演講者直接向聽眾介紹。另一方面，課堂可以有講座的部分，但往往以課堂練習為中心，因此需要與會者更積極地參與。因此，與技術講座相比，教師授課的課程在建立和維護方面通常要求更高、成本也更高，而且只保留給最重要或最難的主題。也就是說，在一個課程建立之後，它的規模可以相對容易地擴大，因為許多教員可以用同樣的課程材料來教課。我們發現，當存在以下情況時，課程往往效果最好：

- 主題足夠複雜，以至於經常出現誤解。課程的建立需要大量的工作，因此只有在滿足特定需求時才能開發。

- 該主題相對穩定。更新課堂材料是一項繁重的工作，所以如果該主題快速發展，其他形式的知識共享將有更好的回報。
- 該主題得益於有教師回答問題和提供個性化的幫助。如果學生可以在沒有指導幫助的情況下輕鬆學習，那麼像文件或錄音這樣的自我服務媒介就會更有效率。Google 的一些介紹性課程也有自學版本。
- 有足夠的需求定期提供課程。否則，潛在的學習者會透過其他方式獲得他們需要的資訊，而不是等待課程。在 Google，這對於地理位置偏遠的小型辦公室來說尤其是一個問題。

> [5]: https://talksat.withgoogle.com and https://www.youtube.com/GoogleTechTalks, to name a few.
>
> 5 https://talksat.withgoogle.com 和 https://www.youtube.com/GoogleTechTalks，僅舉幾例。
>
> 6 g2g 程式詳見。Laszlo Bock, Work Rules! 來自 Google 內部的洞察力，將改變你的生活和領導方式（紐約：十二書局，2015 年）。該書包括對該計劃不同方面的描述，以及如何評估影響，並就設立類似計劃時應關注的內容提出建議。

Documentation 文件

Documentation is written knowledge whose primary goal is to help its readers learn something. Not all written knowledge is necessarily documentation, although it can be useful as a paper trail. For example, it's possible to find an answer to a problem in a mailing list thread, but the primary goal of the original question on the thread was to seek answers, and only secondarily to document the discussion for others.

文件是書面知識，其主要目的是幫助讀者學習一些東西。並非所有的書面知識都一定是文件，儘管它可以用作書面記錄。例如，有可能在一個郵件列表線索中找到一個問題的答案，但線索上的原始問題的主要目標是尋求答案，其次才是為其他人記錄討論情況。

In this section, we focus on spotting opportunities for contributing to and creating formal documentation, from small things like fixing a typo to larger efforts such as documenting tribal knowledge.

在這一節中，我們著重於發現為正式檔案做出貢獻的機會，小到修正一個錯別字，大到記錄內部知識等。

Updating documentation 更新文件

The first time you learn something is the best time to see ways that the existing documentation and training materials can be improved. By the time you've absorbed and understood a new process or system, you might have forgotten what was difficult or what simple steps were missing from the "Getting Started" documentation. At this stage, if you find a mistake or omission in the documentation, fix it! Leave the campground cleaner than you found it, [6] and try to update the documents yourself, even when that documentation is owned by a different part of the organization.

你第一次學習某樣東西的時候，最好是看看如何改進現有的文件和培訓材料。當你吸收並理解了一個新的流程或系統時，你可能已經忘記了 "入門 "文件中的難點或缺少哪些簡單的步驟文件。在這個階段，如果你發現檔案中的錯誤或遺漏，就把它改正過來! 離開營地時要比你發現時更乾淨 [6]，並嘗試自己更新檔案，即使文件屬於組織的其他部門。

At Google, engineers feel empowered to update documentation regardless of who owns it—and we often do—even if the fix is as small as correcting a typo. This level of community upkeep increased notably with the introduction of g3doc,[7] which made it much easier for Googlers to find a documentation owner to review their suggestion. It also leaves an auditable trail of change history no different than that for code.

在 Google，工程師們覺得無論文件的所有者是誰，都有權更新文件——我們經常這樣做——即使修復的範圍很小，比如糾正一個拼寫錯誤。這種社群維護的水平隨著 g3doc[7] 的引入而明顯提高，這使得 Googlers 更容易找到一個文件的所有者來審查他們的建議。這也讓可稽核的變更歷史記錄與程式碼的變更歷史記錄相同。

Creating documentation 建立文件

As your proficiency grows, write your own documentation and update existing docs. For example, if you set up a new development flow, document the steps. You can then make it easier for others to follow in your path by pointing them to your document. Even better, make it easier for people to find the document themselves. Any sufficiently undiscoverable or unsearchable documentation might as well not exist. This is another area in which g3doc shines because the documentation is predictably located right next to the source code, as opposed to off in an (unfindable) document or webpage somewhere.

隨著你的熟練程度的提高，編寫你自己的文件並更新現有的文件。例如，如果你建立了一個新的開發流程，就把這些步驟記錄下來。然後，你可以讓別人更容易沿著你的道路走下去，把他們引導到你的文件。甚至更好的是，讓人們自己更容易找到這個檔案。任何足以讓人無法發現或無法搜尋的檔案都可能不存在。這是 g3doc 大放異彩的另一個領域，因為文件可預測地位於原始碼旁邊，而不是在某個（無法找到的）文件或網頁上。

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. People are more willing to contribute changes if they feel that someone will actually notice and consider their suggestions. At Google, you can file a documentation bug directly from the document itself.

最後，確保有一個反饋的機制。如果沒有簡單直接的方法讓讀者指出文件過時或不準確，他們很可能懶得告訴別人，而下一個新來的人也會遇到同樣的問題。如果人們覺得有人會真正注意到並考慮他們的建議，他們就會更願意做出改變。在 Google，你可以直接從文件本身提交一個文件錯誤。

In addition, Googlers can easily leave comments on g3doc pages. Other Googlers can see and respond to these comments and, because leaving a comment automatically files a bug for the documentation owner, the reader doesn't need to figure out who to contact.

此外，Googlers 可以輕鬆地在 g3doc 頁面上留下評論。其他 Googlers 可以看到並回復這些評論，而且，由於留下評論會自動為文件擁有者歸檔一個錯誤，讀者不需要弄清楚該與誰聯絡。

Promoting documentation 推廣文件

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 like these are good for the organization as a whole given that many people can benefit from the time investment of a few, but without good incentives, it can be challenging to encourage such behavior. We discuss some of these

structural incentives in the section “Incentives and recognition” on page 57.

傳統上，鼓勵工程師記錄他們的工作可能是困難的。編寫文件需要消耗編碼的時間和精力，而且這些工作所帶來的好處並不直接，大部分是由其他人獲益的。鑑於許多人可以從少數人的時間中獲益，像這樣的不對稱權衡對整個組織來說是好的，但如果沒有好的激勵措施，鼓勵這樣的行為是很有挑戰性的。我們在第 57 頁的 "激勵和認可 "一節中討論了其中的一些結構性激勵。

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.

然而，文件作者往往可以直接從編寫文件中受益。假設團隊成員總是向你尋求幫助，以除錯某些種類的生產故障。記錄你的程式需要前期的時間投入，但在這項工作完成後，你可以在未來節省時間，指點團隊成員去看文件，只在需要時親自提供幫助。

Writing documentation also helps your team and organization scale. First, the information in the documentation becomes canonicalized as a reference: team members can refer to the shared document and even update it themselves. Second, the canonicalization may spread outside the team. Perhaps some parts of the documentation are not unique to the team’s configuration and become useful for other teams looking to resolve similar problems.

編寫文件也有助於你的團隊和組織的擴充。首先，文件中的資訊成為規範化的參考：團隊成員可以參考共享的文件，甚至自己更新它。其次，規範化可能擴散到團隊之外。也許文件中的某些部分對團隊的配置來說並不獨特，對其他想要解決類似問題的團隊來說變得有用。

> 7 見 "童子軍規則 "和 Kevlin Henney，《每個程式設計師應該知道的 97 件事》（波士頓：O'Reilly，2010）。
>
> 8 g3doc 是 "google3 文件 "的縮寫。google3 是 Google 單儲存庫原始碼庫的當前化身的名稱。

Code 程式碼

At a meta level, 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.

在元層面上，程式碼就是知識，所以寫程式碼的行為本身可以被認為是一種知識的轉錄。雖然知識共享可能不是生產程式碼的直接目的，但它往往是一個副產品，它可以透過程式碼的可讀性和清晰性來促進。

Code documentation is one way to share knowledge; clear documentation not only benefits consumers of the library, but also future maintainers. Similarly, implementation comments transmit knowledge across time: you’re writing these comments expressly for the sake of future readers (including Future You!). In terms of trade- offs, code comments are subject to the same downsides as general documentation: they need to be actively maintained or they can quickly become out of date, as anyone who has ever read a comment that directly contradicts the code can attest.

程式碼文件是分享知識的一種方式；清晰的文件不僅有利於庫的使用者，而且也有利於後繼的維護者。同樣地，實現註釋也能跨時空傳播知識：你寫這些註釋是為了未來的讀者（包括未來的你！）。就權衡利弊而言，程式碼註釋和一般的文件一樣有缺點：它們需要積極維護，否則很快就會過時，任何讀過與程式碼直接矛盾的註釋的人都可以證明這一點。

Code reviews (see Chapter 9) are often a learning opportunity for both author(s) and reviewer(s). For example, a reviewer's suggestion might introduce the author to a new testing pattern, or a reviewer might learn of a new library by seeing the author use it in their code. Google standardizes mentoring through code review with the readability process, as detailed in the case study at the end of this chapter.

程式碼審查（見第 9 章）對作者和審查者來說都是一個學習機會。例如，審查者的建議可能會給作者帶來新的測試模式，或者審查者可能透過看到作者在他們的程式碼中使用一個新的函式庫來了解它。Google 透過程式碼審查的可讀性過程來規範指導，在本章末尾的案例研究中詳細介紹了這一點。

Scaling Your Organization's Knowledge 擴充組織的知識

Ensuring that expertise is appropriately shared across the organization becomes more difficult as the organization grows. Some things, like culture, are important at every stage of growth, whereas others, like establishing canonical sources of information, might be more beneficial for more mature organizations.

隨著組織的發展，確保專業知識在整個組織內得到適當的分享變得更加困難。有些事情，比如文化，在每一個成長階段都很重要，而其他事情，比如建立規範的資訊源，可能對更成熟的組織更有利。

Cultivating a Knowledge-Sharing Culture 培養知識共享文化

Organizational culture is the squishy human thing that many companies treat as an afterthought. But at Google, we believe that focusing on the culture and environment first [8] results in better outcomes than focusing on only the output—such as the code—of that environment.

組織文化是許多公司視為事後諸葛亮的東西。但在 Google，我們相信首先關注文化和環境[8]會比只關注該環境的產出（如程式碼）帶來更好的結果。

Making major organizational shifts is difficult, and countless books have been written on the topic. 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.

進行重大的組織轉變是很難的，關於這個主題的書已經不計其數。我們並不假設擁有所有的答案，但我們可以分享 Google 為創造一種促進學習的文化而採取的具體步驟。

See the book Work Rules [9] for a more in-depth examination of Google's culture.

請參閱《工作規則》[9]一書，對 Google 的文化進行更深入的研究。

> 9 Laszlo Bock, Work Rules! 來自 Google 內部的洞察力，將改變你的生活和領導方式（紐約：十二書局，2015）。

> 10 同上。

Respect 尊重

The bad behavior of just a few individuals can make an entire team or community unwelcoming. In such an environment, novices learn to take their questions elsewhere, and potential new experts stop trying and don't have room to grow. In the worst cases, the group reduces to its most toxic members. It can be difficult to recover from this state.

僅僅幾個人的不良行為就可以使整個團隊或社群不受歡迎。在這樣的環境中，新手將會把問題轉移到其他地方，而潛在的新專家則停止嘗試，沒有成長的空間。在最糟糕的情況下，這個團體會只剩下有有毒的成員。要從這種狀態中恢復過來很困難。

Knowledge sharing can and should be done with kindness and respect. In tech, tolerance—or worse, reverence—of the "brilliant jerk" is both pervasive and harmful, but being an expert and being kind are not mutually exclusive. The Leadership section of Google's software engineering job ladder outlines this clearly:
Although a measure of technical leadership is expected at higher levels, not all leadership is directed at technical problems. Leaders improve the quality of the people around them, improve the team's psychological safety, create a culture of teamwork and collaboration, defuse tensions within the team, set an example of Google's culture and values, and make Google a more vibrant and exciting place to work. Jerks are not good leaders.

知識分享可以而且應該以善意和尊重的方式進行。在科技界，對 "聰明的混蛋 "的容忍——還有更糟糕的是，崇尚 "聰明的混蛋"，即是普遍又是危害的，但作為一個專家和善良並不互斥。Google 軟體工程職位階梯的領導力部分清楚地概述了這一點：
雖然在更高的層次上需要衡量技術領導力，但並非所有的領導力都針對技術問題。領導者可以提高周圍人的素質，改善團隊的心理安全感，創造團隊合作文化，化解團隊內部的緊張情緒，樹立 Google 文化和價值觀的榜樣，讓 Google 成為一個更具活力和激情的工作場所。混蛋不是好領導。

This expectation is modeled by senior leadership: Urs Hölzle (Senior Vice President of Technical Infrastructure) and Ben Treynor Sloss (Vice President, Founder of Google SRE) wrote a regularly cited internal document ("No Jerks") about why Googlers should care about respectful behavior at work and what to do about it.

這種期望是由高階領導層示範的:Urs Hölzle（技術基礎設施高階副總裁）和 Ben Treynor Sloss（副總裁，Google SRE 的創始人）寫了一份經常被參考的內部檔案（"No Jerks"），說明為什麼 Google 人應該關心工作中的尊重行為以及如何做。

ntives and recognition 獎勵和認可

Good culture must be actively nurtured, and encouraging a culture of knowledge sharing requires a commitment to recognizing and rewarding it at a systemic level. It's a common mistake for organizations to pay lip service to a set of values while actively rewarding behavior that does not enforce those values. People react to incentives over platitudes, and so it's important to put your money where your mouth is by putting in place a system of compensation and awards.

良好的文化必須積極培育，而鼓勵知識共享的文化需要獲得在系統層面上認可和獎勵。一個常見的錯誤是，組織在口頭上支援一套價值觀的同時，積極獎勵那些不執行這些價值觀的行為。人們對語言的表揚很難有感覺，因此，透過建立薪酬和獎勵制度，把錢放在嘴邊就很重要的。

Google uses a variety of recognition mechanisms, from company-wide standards such as performance review and promotion criteria to peer-to-peer awards between Googlers.

Google 使用了各種認可機制，從全公司的標準，如績效審查和晉升標準到 Google 員工同行獎勵。

Our software engineering ladder, which we use to calibrate rewards like compensation and promotion across the company, encourages engineers to share knowledge by noting these expectations explicitly. 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
- Sustaining and developing the software community at Google via code and design reviews, engineering education and development, and expert guidance to others in the field

我們的軟體工程師級別用於校準整個公司的薪酬和晉升等獎勵，透過明確記錄這些期望，鼓勵工程師分享知識。在更高的層次上，級別明確指出了更廣泛影響力的重要性，這種期望隨著資歷的增加而增加。在最高級別，領導力的例子包括以下內容：

- 透過擔任初級員工的導師，幫助他們在技術和 Google 角色上發展，培養未來的領導者。
- 透過程式碼和設計審查、工程教育和開發以及對該領域其他人的專家指導，維持和發展 Google 的軟體社群。

Job ladder expectations are a top-down way to direct a culture, but culture is also formed from the bottom up. At Google, the peer bonus program is one way we embrace the bottom-up culture. Peer bonuses are a monetary award and formal recognition that any Googler can bestow on any other Googler for above-and-beyond work.[10] For example, when Ravi sends a peer bonus to Julia for being a top contributor to a mailing list—regularly answering questions that benefit many readers—he is publicly recognizing her knowledge-sharing work and its impact beyond her team. Because peer bonuses are employee driven, not management driven, they can have an important and powerful grassroots effect.

工作階梯的期望是一種自上而下引導文化的方式，但文化也是自下而上形成的。在 Google，同行獎金計劃是我們擁抱自下而上文化的一種方式。同行獎金是一種貨幣獎勵和正式認可，任何 Google 員工都可以將其授予任何其他 Google 員工，以表彰他們的超越性工作。例如，當 Ravi 將同行獎金發給 Julia，因為她是一個郵件列表的最上層貢獻者——定期回答問題，使許多讀者受益，他公開承認她的知識共享工作及其對團隊以外的影響。由於同行獎金是由員工驅動的，而不是由管理層驅動的，因此它們可以產生重要而強大的基層效應。

Similar to peer bonuses are kudos: public acknowledgement of contributions (typically smaller in impact or effort than those meriting a peer bonus) that boost the visibility of peer-to-peer contributions.

與同行獎金相似的是嘉獎：對貢獻的公開承認（通常比那些值得同行獎金的影響或努力要小），提高同行貢獻的知名度。

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. It's also common for the recipient's manager to forward the award email to the team to celebrate one another's achievements.

當一個 Googler 給另一個 Googler 頒發同行獎金或嘉獎時，他們可以選擇在獎勵郵件上抄送其他組或個人，提高對同行工作的認可。收件人的經理將獎勵郵件轉發給團隊以慶祝彼此的成就也很常見。

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.

一個人們可以正式和容易地認可他們的同行系統是一個強大的工具，可以鼓勵同行繼續做他們所做的了不起的事情。重要的不是獎金：而是同行的認可。

> 11 同行獎金包括現金獎勵和證書，以及在一個名為 gThanks 的內部工具中成為 Googler 獎勵記錄的永久組成部分。

Establishing Canonical Sources of Information 建立規範的資訊源

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. For example, a guide to setting up a basic developer workflow should be made canonical, whereas a guide for running a local Frobber instance is more relevant just to the engineers working on Frobber.

規範的資訊源是集中的、公司範圍的資訊庫，提供了一種標準化和傳播專家知識的方法。它們最適用於與組織內所有工程師相關的資訊，否則容易出現資訊孤島。例如，建立一個基本的開發者工作流程的指南應該成為規範，而執行一個本地 Frobber 例項的指南則只與從事 Frobber 的工程師有關。

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.

建立規範的資訊源需要比主要獲取更本地化的資訊（如團隊文件）更高的投資，但也有更多的好處。為整個組織提供集中的參考資料，使廣泛需要的資訊更容易找到，也更可預測，並解決了資訊碎片化的問題，因為這些問題可能會在多個處理類似問題的團隊制定自己的指南時出現，這些指南往往相互衝突。

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.

因為規範資訊是高度可見的，並且旨在提供組織層面的共同理解，所以內容由主題專家積極維護和稽核是很重要的。主題越複雜，規範內容的所有者就越明確。善意的讀者可能會看到某些東西已經過時，但缺乏進行修復所需的重大結構更改的專業知識，即使工具可以很容易地提出更新建議。

Creating and maintaining centralized, canonical sources of information is expensive and time consuming, and not all content needs to be shared at an organizational level. When considering how much effort to invest in this resource, consider your audience. Who benefits from this information? You? Your team? Your product area? All engineers?

建立和維護集中的、規範的資訊來源是昂貴和耗時的，而且不是所有的內容都需要在組織層面上共享。當考慮在這個資源上投入多少精力時，要考慮你的受眾。誰會從這些資訊中受益？你嗎？你的團隊？你的產品領域？所有的工程師？

Developer guides 開發者指南

Google has a broad and deep set of official guidance for engineers, including style guides, official software engineering best practices,[11] guides for code review[12] and testing,[13] and Tips of the Week (TotW).[14]

Google 為工程師提供了一套廣泛而深入的官方指導，包括風格指南、官方軟體工程最佳實踐[11]、程式碼審查[12]和測試指南[13]以及每週提示（TotW）[14]。

The corpus of information is so large that it's impractical to expect engineers to read it all end to end, much less be able to absorb so much information at once. 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.

資訊函式庫是如此之大，以至於期望工程師從頭到尾讀完它是不切實際的，更不用說能夠一次吸收這麼多資訊了。相反，已經熟悉某項準則的專家可以將連結傳送給工程師同事，他們可以閱讀參考資料並瞭解更多資訊。專家不需要親自解釋公司範圍內的做法，從而節省了時間，而學習者現在知道有一個值得信賴的資訊的典型來源，他們可以在需要時存取。這樣過程可以擴充知識，因為它使專家能夠透過利用共同的、可擴充的資源來重新認識和解決特定的資訊需求。

> 12 如 Google 公司有關軟體工程的書籍。
>
> 13 見第 9 章。
>
> 14 見第 11 章。
>
> 15 可用於多種語言。對外可用於 C++，在https://abseil.io/tips。

go/links 連結

go/links (sometimes referred to as goto/ links) are Google's internal URL shortener.[15] Most Google-internal references have at least one internal go/ link. For example, "go/ spanner" provides information about Spanner, "go/python" is Google's Python developer guide. The content can live in any repository (g3doc, Google Drive, Google Sites, etc.), but having a go/ link that points to it provides a predictable, memorable way to access it. This yields some nice benefits:

- go/links are so short that it's easy to share them in conversation ("You should check out go/frobber!"). This is much easier than having to go find a link and then send a message to all interested parties. Having a low-friction way to share references makes it more likely that that knowledge will be shared in the first place.

- go/links provide a permalink to the content, even if the underlying URL changes. When an owner moves content to a different repository (for example, moving content from a Google doc to g3doc), they can simply update the go/link's target URL. The go/link itself remains unchanged.

go/links（有時被稱為 goto/連結）是 Google 的內部 URL 縮短器。[15] 大多數 Google 內部的參考資料至少有一個內部 go/links。例如，"go/spanner "提供關於 Spanner 的資訊，"go/python "是 Google 的 Python 開發者指南。這些內容可以存在於任何資源庫中（g3doc、Google Drive、Google Sites 等），但有一個指向它的 go/links 提供了一種可預測的、可記憶的存取方式。這產生了一些很好的好處：

- go/links 非常短，很容易在談話中分享它們（"你應該看看 go/frobber！"）。這比去找一個連結，然後給所有感興趣的人發一個訊息要容易得多。有一個低成本的方式來分享參考資料，使得這些知識更有可能在第一時間被分享。
- go/links 提供內容的固定連結，即使底層的 URL 發生變化。當所有者將內容移到一個不同的資源庫時（例如，將內容從 Google doc 移到 g3doc），他們可以簡單地更新 go/link 的目標 URL。go/link 本身保持不變。

go/links are so ingrained into Google culture that a virtuous cycle has emerged: a Googler looking for information about Frobber will likely first check go/frobber. If the go/ link doesn't point to the Frobber Developer Guide (as expected), the Googler will generally configure the link themselves. As a result, Googlers can usually guess the correct go/link on the first try.

go/links 在 Google 文化中根深蒂固，以至於出現了一個良性迴圈：一個尋找 Frobber 資訊的 Googler 可能會首先檢視 go/frobber。如果 go/links 沒有指向 Frobber 開發者指南（如預期），Googler 一般會自己配置連結。因此，Googler 通常可以在第一次嘗試時猜出正確的 go/links。

Codelabs 程式碼實驗室

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.[16] A canonical collection of codelabs for technologies broadly used across Google is available at go/codelab. These codelabs go through several rounds of formal review and testing before publication. 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, but engineers can still access them on demand and complete them on their own; but they are expensive to maintain and are not tailored to the learner's specific needs.

Google codelabs 是有指導的實踐課程，透過結合解釋、工作中的最佳實踐範例程式碼和程式碼練習，向工程師傳授新概念或流程[16]。go/codelab 上提供了一個規範的 codelabs 集合，用於 Google 廣泛使用的技術。這些程式碼集在釋出前經過了幾輪正式的審查和測試。Codelabs 是介於靜態文件和講師指導課程之間的一個有趣的中間點，它們分享了兩者的最佳和最差的特點。它們的實踐性使它們比傳統的文件更有吸引力，但工程師仍然可以按需存取它們，並自行完成；但它們的維護成本很高，而且不適合學習者的特定需求。

> 16 go/link 與 go 語言無關。
>
> 17 外部程式碼實驗室可在 `https://codelabs.developers.google.com`。

Static analysis 靜態分析

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.

靜態分析工具是分享程式設計檢查最佳實踐的強大方式。每種程式語言都有其特定的靜態分析工具，它們有相同的共同目的：提醒程式碼作者和審查者注意可以改進程式碼的方式，以遵循規範和最佳實踐。有些工具更進一步，提供自動將這些改進應用到程式碼中。

Setting up static analysis tools requires an upfront 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.

設定靜態分析工具需要前期投入，但一旦這些工具到位，它們就會有效地擴充。當最佳實踐的檢查被新增到一個工具中時，每個使用該工具的工程師都會意識到這是最佳實踐。這也使工程師們可以騰出時間來教其他東西：原本用於手動教授（現在是自動的）最佳實踐的時間和精力，可以用來教授其他東西。靜態分析工具增強了工程師的知識。它們使一個組織能夠應用更多的最佳實踐，並比其他方式更一致地應用它們。

Staying in the Loop 保持互動

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.

有些資訊對於完成工作至關重要，例如知道如何執行典型的開發工作流。其他的資訊，比如流行的生產力工具的更新，雖然不那麼關鍵，但仍然有用。對於這種型別的知識，資訊共享媒介的正式性取決於所傳遞資訊的重要性。例如，使用者希望官方文件保持最新，但通常對新聞稿內容沒有這樣的期待，因此新聞稿內容需要所有者進行較少的維護和保養。

Newsletters 時事通訊

Google has a number of company-wide newsletters that are sent to all engineers, including EngNews (engineering news), Ownd (Privacy/Security news), and Google's Greatest Hits (report of the most interesting outages of the quarter). These are a good way to communicate information that is of interest to engineers but isn't mission critical. For this type of update, we've found that newsletters get better engagement when they are sent less frequently and contain more useful, interesting content. Otherwise, newsletters can be perceived as spam.

Google 有一些發給所有工程師的公司範圍內的新聞簡報，包括 EngNews（工程新聞），Ownd（隱私/安全新聞），以及 Google 的 Greatest Hits（本季度最有趣的故障報告）。這些都是傳達工程師感興趣但並非關鍵任務的資訊的好方法。對於這種型別的更新，我們發現，如果通訊傳送的頻率較低，並且包含更多有用的、有趣的內容，就會得到更好的參與度。否則，新聞簡報會被認為是垃圾郵件。

Even though most Google newsletters are sent via email, some are more creative in their distribution. Testing on the Toilet (testing tips) and Learning on the Loo (productivity tips) are single-page newsletters posted inside toilet stalls. This unique delivery medium helps the Testing on the Toilet and Learning on the Loo stand out from other newsletters, and all issues are archived online.

儘管大多數 Google 新聞通訊都是透過電子郵件傳送的，但有些新聞通訊的傳送方式更有創意。廁所測試（測試提示）和廁所學習（產品活動提示）是張貼在廁所裡的單頁新聞通訊。這種獨特的傳送媒介幫助廁所測試和廁所學習從其他新聞通訊中脫穎而出，而且所有期刊都在線上存檔。

Communities 社群

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.

Google 人喜歡圍繞各種主題建立跨組織的社群和分享知識。這些開放的渠道可以讓你更容易地向周圍的人學習，避免資訊孤島和重複。Google 群組尤其受歡迎：Google 有數千個內部團體，形式各異。有些專門用於故障排除；其他人，如程式碼健康小組，更多的是討論和指導。內部 Google+作為非正式資訊來源在 Google 使用者中也很受歡迎，因為人們會發布有趣的技術分類或他們正在從事的專案的詳細資訊。

Readability: Standardized Mentorship Through Code Review 可讀性：透過程式碼審查實現標準化指導

At Google, “readability” refers to more than just code readability; it is a standardized, Google-wide mentorship process for disseminating programming language best practices. Readability covers a wide breadth of expertise, including but not limited to language idioms, code structure, API design, appropriate use of common libraries, documentation, and test coverage.

在 Google，"可讀性 "指的不僅僅是程式碼的可讀性；這是一個標準化的、Google 範圍內的指導過程，用於傳播程式語言最佳實踐。可讀性涵蓋了廣泛的專業知識，包括但不限於語言語義、程式碼結構、API 設計、通用庫的正確使用、文件和測試覆蓋率。

Readability started as a one-person effort. In Google’s early days, Craig Silverstein (employee ID #3) would sit down in person with every new hire and do a line-by-line “readability review” of their first major code commit. It was a nitpicky review that covered everything from ways the code could be improved to whitespace conventions. This gave Google’s codebase a uniform appearance but, more important, it taught best practices, highlighted what shared infrastructure was available, and showed new hires what it’s like to write code at Google.

可讀性最初是一個人的努力。在 Google 早期，Craig Silverstein（員工 ID#3）會親自與每一位新員工坐下來，逐行對他們的第一個主要程式碼提交進行“可讀性審查”。這是一次挑剔的審查，涵蓋了從程式碼改進到空白約定的所有方面。這讓 Google 的程式碼庫有了統一的模式，但更重要的是，它教授了最佳實踐，強調了什麼是可用的共享基礎設施，並向新員工展示了在 Google 編寫程式碼的感覺。

Inevitably, Google's hiring rate grew beyond what one person could keep up with. So many engineers found the process valuable that they volunteered their own time to scale the program. Today, around 20% of Google engineers are participating in the readability process at any given time, as either reviewers or code authors.

不可避免地，Google 的招聘速度越來越快，超出了一個人的能力範圍。如此多的工程師發現這個過程很有價值，於是他們自願拿出自己的時間來擴充這個專案。今天，大約有 20%的 Google 工程師在任何時候都在參與可讀性處理序，他們要麼是審查員，要麼是程式碼作者。

What Is the Readability Process? 什麼是可讀性過程？

Code review is mandatory at Google. Every changelist (CL)[17] 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. This requirement was added after Google grew to a point where it was no longer possible to enforce that every engineer received code reviews that taught best practices to the desired rigor.

在 Google，程式碼審查是強制性的。每個變更列表（CL）[17]都需要可讀性批准，這表明擁有該語言的可讀性認證的人已經批准了該 CL。經過認證的作者隱含地對他們自己的 CL 提供可讀性批准；否則，一個或多個合格的審查員必須明確地對 CL 提供可讀性批准。這項要求是在 Google 發展到無法強制要求每個工程師接受程式碼審查，從而將最佳實踐傳授到所需的嚴格程度之後新增的。

Within Google, having readability certification is commonly referred to as “having readability” for a language. Engineers with readability have demonstrated that 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. Readability brings increased responsibility: engineers with readability are trusted to continue to apply their knowledge to their own code and to act as reviewers for other engineers' code.

在 Google 內部，擁有可讀性認證通常被稱為一門語言的 "可讀性"。擁有可讀性認證的工程師已經證明，他們始終如一地寫出清晰、習慣和可維護的程式碼，體現了 Google 對特定語言的最佳實踐和編碼風格。他們透過可讀性程式提交 CL，在此過程中，一個集中的可讀性審查員小組審查 CL，並就它在多大程度上展示了各個領域的掌握程度給出反饋。隨著作者對可讀性準則的內化，他們收到的關於他們的 CL 的評論越來越少，直到他們最終從這個過程中畢業並正式獲得可讀性。可讀性帶來了更多的責任：擁有可讀性的工程師被信任，可以繼續將他們的知識應用於他們自己的程式碼，並作為其他工程師的程式碼的審查者。

Around 1 to 2% of Google engineers are readability reviewers. All reviewers are volunteers, and anyone with readability is welcome to self-nominate to become a readability reviewer. 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, not a gatekeeping or adversarial one. 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 ("Chesterson's fence"). If the rationale for any given guideline is unclear, authors should ask for clarification ("ask questions").

大約有 1%到 2%的 Google 工程師是可讀性審查員。所有的審查員都是志願者，任何有可讀性的人都歡迎自我提名成為可讀性審查員。可讀性審查員被要求達到最高標準，因為他們不僅要有深厚的語言專業知識，還要有透過程式碼審查進行教學的能力。他們被期望把可讀性首先作為一個指導和合作的過程，而不是一個把關或對抗的過程。我們鼓勵可讀性審查員和 CL 作者在審查過程中進行討論。審查人為他們的評論提供相關的引文，這樣作者就可以瞭解制定文體指南的理由（"切斯特森的籬笆"）。如果任何特定準則的理由不清楚，作者應該要求澄清（"提問"）。

> 18 變更列表是構成版本控制系統中的一個變更的檔案列表。變更列表與變更集是同義的。

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—which is good!—but the corpus of information is so large[18] that it can be overwhelming, especially to newcomers.

可讀性是一個人為驅動的過程，旨在以標準化但個性化的方式擴充知識。作為書面知識和內部知識的互補混合體，可讀性結合了書面檔案的優勢，可以透過可參考的參考文獻來獲取，也結合了專家審查員的優勢，他們知道應該參考哪些指南。典範指南和語言建議被全面地記錄下來——這很好！——但資訊的語料庫非常大[18]，可能會讓人不知所措，特別是對新人來說。

> 19 截至 2019，Google C++風格指南只有 40 頁長。構成完整的最佳實踐語法庫的次要材料要長很多倍。

Why Have This Process? 為什麼有這個過程？

Code is read far more than it is written, and this effect is magnified at Google's scale and in our (very large) monorepo.[19] Any engineer can look at and learn from the wealth of knowledge that is the code of other teams, and powerful tools like Kythe make it easy to find references throughout the entire codebase (see Chapter 17). An important feature of documented best practices (see Chapter 8) is that they provide consistent standards for all Google code to follow. Readability is both an enforcement and propagation mechanism for these standards.

程式碼的閱讀量遠遠大於編寫量，這種影響在 Google 的規模和我們（非常大的）monorepo 中被放大。[19] 任何工程師都可以檢視並學習其他團隊的程式碼的豐富知識，而像 Kythe 這樣強大的工具使得在整個程式碼庫中尋找參考資料變得很容易（見第 17 章）。文件化最佳實踐的一個重要特徵（見第 8 章）是，它們為所有 Google 程式碼提供了一致的標準。可讀性是這些標準的可強制執行和傳播的基礎。

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. Centralizing the process makes a significant trade-off: the program is limited to scaling linearly rather than sublinearly with organization growth, but it makes it easier to enforce consistency, avoid islands, and avoid (often unintentional) drifting from established norms.

可讀性專案的主要優勢之一是，它讓工程師接觸到的不僅僅是他們自己團隊的內部知識。為了獲得特定語言的可讀性，工程師們必須將 CLs 傳送給一組集中的可讀性審查員，他們審查整個公司的程式碼。將流程集中化會帶來顯著的折衷：該計劃僅限於隨著組織的發展而線性擴充，而不是亞線性擴充，但它更容易實現一致性，避免孤島，並避免（通常是無意的）偏離既定規範。

The value of codebase-wide consistency cannot be overstated: even with tens of thousands of engineers writing code over decades, it ensures that code in a given language will look similar across the corpus. This enables readers to focus on what the code does rather than being distracted by why it looks different than code that they're used to. Large-scale change authors (see Chapter 22) can more easily make changes across the entire monorepo, crossing the boundaries of thousands of teams. People can change teams and be confident that the way that the new team uses a given language is not drastically different than their previous team.

整個程式碼庫的一致性的價值怎麼強調都不為過：即使數十年來有數萬名工程師編寫程式碼，它也確保了一門語言中的程式碼在整個語法庫中看起來都是相似的。這使讀者能夠專注於程式碼的作用，而不是被為什麼它看起來與他們習慣的程式碼不同而分散注意力。大規模的變更作者（見第 22 章）可以更容易地在整個語法庫中進行變更，跨越成千上萬個團隊的界限。人們可以更換團隊，並確信新的團隊使用特定語言的方式不會與他們以前的團隊有很大的不同。

> 20 有關 Google 使用 monorepo 的原因，請參閱https://cacm.acm.org/magazines/2016/7/204032-why-google-stores-billions-of-lines-of-code-in-a-single-repository/fulltext. 還要注意的是，並非 Google 的所有程式碼都存在於 monorepo 中；此處描述的可讀性僅適用於 monorepo，因為它是儲存庫內一致性的概念。

These benefits come with some costs: readability is a heavyweight process compared to other mediums like documentation and classes because it is mandatory and enforced by Google tooling (see Chapter 19). These costs are nontrivial and include the following:

- Increased friction for teams that do not have any team members with readability, because they need to find reviewers from outside their team to give readability approval on CLs.
- Potential for additional rounds of code review for authors who need readability review.
- Scaling disadvantages of being a human-driven process. Limited to scaling linearly to organization growth because it depends on human reviewers doing specialized code reviews.

這些好處伴隨著一些成本：與文件和類等其他媒介相比，可讀性是一個重量級的過程，因為它是強制性的，並由 Google 工具化強制執行（見第 19 章）。這些成本是不小的，包括以下幾點：

- 對於那些沒有任何團隊成員具備可讀性的團隊來說，增加了衝突，因為他們需要從團隊之外尋找審查員來對 CL 進行可讀性審批。
- 對於需要可讀性審查的作者來說，有可能需要額外的幾輪程式碼審查。

- 作為一個由人驅動的過程，其擴充性成為瓶頸。由於它依賴於人類審查員進行專門的程式碼審查，所以對組織的增長具有線性擴充的限制。

The question, then, is whether the benefits outweigh the costs. There's also the factor of time: the full effect of the benefits versus the costs are not on the same timescale. The program makes a deliberate trade-off of increased short-term code-review latency and upfront costs for the long-term payoffs of higher-quality code, repository-wide code consistency, and increased engineer expertise. The longer timescale of the benefits comes with the expectation that code is written with a potential lifetime of years, if not decades.[20]

那麼，問題是收益是否大於成本。還有一個時間因素：收益與成本的全部效果並不在同一時間維度上。該計劃對增加的短期程式碼審查延遲和前期成本進行了慎重的權衡，以獲得更高品質程式碼、儲存庫範圍內的程式碼一致性和增加的工程師專業知識的長期回報。效益的時間尺度較長，期望編寫的程式碼有幾年甚至幾十年的潛在壽命[20]。

As with most—or perhaps all—engineering processes, there's always room for improvement. Some of the costs can be mitigated with tooling. A number of readability comments address issues that could be detected statically and commented on automatically by static analysis tooling. 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.

與大多數——或許是所有的工程過程一樣，總是有改進的餘地。一些成本可以透過工具來降低。許多可讀性註釋解決了靜態檢測和靜態分析工具自動註釋的問題。隨著我們對靜態分析的不斷投資，可讀性審查員可以越來越多地關注更高層次的領域，比如某個特定的程式碼塊是否可以被不熟悉程式碼庫的外部讀者所理解，而外部讀者不熟悉程式碼庫，而不是自動檢測，例如行是否有尾隨空白。

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.

但光有願望是不夠的。可讀性是一個有爭議的專案：一些工程師抱怨說這是一個不必要的官僚主義，是對工程師時間的浪費。可讀性的權衡是值得的嗎？為了找到答案，我們求助於我們可信賴的工程生產力研究（EPR）團隊。

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 behavior 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.[21] 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 behavior to avoid readability issues when writing and reviewing code.

EPR 團隊對可讀性進行了深入的研究，包括但不限於人們是否受到這個過程的阻礙，是否學到了什麼，或者畢業後是否改變了他們的行為。這些研究表明，可讀性對工程速度有正向的積極影響。具有可讀性的作者的 CL 比不具有可讀性的作者的 CL 在統計上要少花時間。具有可讀性的工程師與不具有可讀性的工程師相比，自我報告的對其程式碼品質的滿意度——缺乏對程式碼品質更客觀的衡量標準——更高。絕大多數完成該計劃的工程師對這一過程表示滿意，並認為這是值得的。他們報告說從審查員那裡

學到了東西，並改變了自己的行為，以避免在編寫和評審程式碼時出現可讀性問題。

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.

Google 有著非常濃厚的程式碼審查文化，可讀性是這種文化的延伸。可讀性從一個工程師的熱情發展到一個由專家組指導所有 Google 工程師的正式專案。它隨著 Google 的成長而不斷髮展變化，並將隨著 Google 需求的變化而繼續發展。

> 21 因此，已知時間跨度較短的程式碼不受可讀性要求的約束。考試範例包括實驗/目錄（明確指定為實驗程式碼，不能推動生產）和 Area 120 計劃，這是 Google 實驗產品的研討會。
>
> 22 這包括控制各種因素，包括在 Google 的任職期限，以及與已經具備可讀性的作者相比，沒有可讀性的作者的 CLs 通常需要額外的審查。

Conclusion 結論

Knowledge is in some ways the most important (though intangible) capital of a software engineering organization, and sharing of that knowledge is crucial for making an organization resilient and redundant in the face of change. A culture that promotes open and honest knowledge sharing distributes that knowledge efficiently across the organization and allows that organization to scale over time. In most cases, investments into easier knowledge sharing reap manyfold dividends over the life of a company.

在某些方面，知識是軟體工程組織最重要的（儘管是無形的）資產，而知識的共享對於使組織在面對變化時具有彈性和冗餘至關重要。一種促進開放和誠實的知識共享的文化可以在整個組織內有效地分配這些知識，並使該組織能夠隨著時間的推移而擴充。在大多數情況下，對更容易的知識共享的投入會在一個公司的生命週期中獲得許多倍的回報。

TL;DRs 內容提要

- Psychological safety is the foundation for fostering a knowledge-sharing environment.
- Start small: ask questions and write things down.
- Make it easy for people to get the help they need from both human experts and documented references.
- At a systemic level, encourage and reward those who take time to teach and broaden their expertise beyond just themselves, their team, or their organization.
- There is no silver bullet: empowering a knowledge-sharing culture requires a combination of multiple strategies, and the exact mix that works best for your organization will likely change over time.
- 心理安全是培養知識共享環境的基礎。
- 從小事做起：問問題，把事情寫下來。
- 讓人們可以很容易地從專家和有記錄的參考資料中獲得他們需要的幫助。

- 在系統的層面上，鼓勵和獎勵那些花時間去教授和擴大他們的專業知識，而不僅僅是他們自己、他們的團隊或他們的組織。
- 沒有什麼靈丹妙藥：增強知識共享文化需要多種策略的結合，而最適合你的組織的確切組合可能會隨著時間的推移而改變。

---

1. In other words, rather than developing a single global maximum, we have a bunch of local maxima./ ↩

2. David Lorge Parnas, Software Engineering: Multi-person Development of Multi-version Programs (Heidelberg: Springer-Verlag Berlin, 2011). ↩

3. Impostor syndrome is not uncommon among high achievers, and Googlers are no exception—in fact, a majority of this book's authors have impostor syndrome. We acknowledge that fear of failure can be difficult for those with impostor syndrome and can reinforce an inclination to avoid branching out. ↩

4. See "How to ask good questions.. ↩

5. The g2g program is detailed in: Laszlo Bock, Work Rules!: Insights from Inside Google That Will Transform How You Live and Lead (New York: Twelve Books, 2015). It includes descriptions of different aspects of the program as well as how to evaluate impact and recommendations for what to focus on when setting up similar programs. ↩

6. See "The Boy Scout Rule" and Kevlin Henney, 97 Things Every Programmer Should Know (Boston: O'Reilly, 2010). ↩ ↩

7. g3doc stands for "google3 documentation." google3 is the name of the current incarnation of Google's monolithic source repository. ↩ ↩

8. Laszlo Bock, Work Rules!: Insights from Inside Google That Will Transform How You Live and Lead (New York: Twelve Books, 2015). ↩ ↩

9. Ibid. ↩ ↩

10. Peer bonuses include a cash award and a certificate as well as being a permanent part of a Googler's award record in an internal tool called gThanks. ↩

11. Such as books about software engineering at Google. ↩ ↩

12. See Chapter 9. ↩ ↩

13. See Chapter 11 ↩ ↩

14. Available for multiple languages. Externally available for C++ at https://abseil.io/tips. ↩ ↩

15. go/ links are unrelated to the Go language. ↩ ↩

16. External codelabs are available at `https://codelabs.developers.google.com`. ↩ ↩

17. A changelist is a list of files that make up a change in a version control system. A changelist is synonymous with a changeset. ↩ ↩

18. As of 2019, just the Google C++ style guide is 40 pages long. The secondary material making up the complete corpus of best practices is many times longer. ↩ ↩

19. For why Google uses a monorepo, see `https://cacm.acm.org/magazines/2016/7/204032-why-google-stores-billions-of-lines-of-code-in-a-single-repository/fulltext`. Note also that not all of Google's code lives within the monorepo; readability as described here applies only to the monorepo because it is a notion of within- repository consistency. ↩ ↩

20. For this reason, code that is known to have a short time span is exempt from readability requirements. Examples include the experimental/ directory (explicitly designated for experimental code and cannot push to production) and the Area 120 program, a workshop for Google's experimental products. ↩ ↩

21. This includes controlling for a variety of factors, including tenure at Google and the fact that CLs for authors who do not have readability typically need additional rounds of review compared to authors who already have readability. ↩