CHAPTER 7

Measuring Engineering Productivity

第七章 度量工程效率

Written by Ciera Jaspan

Edited by Riona Macnamara

Google is a data-driven company. We back up most of our products and design decisions with hard data. The culture of data-driven decision making, using appropriate metrics, has some drawbacks, but overall, relying on data tends to make most decisions objective rather than subjective, which is often a good thing. Collecting and analyzing data on the human side of things, however, has its own challenges. Specifically, within software engineering, Google has found that having a team of specialists focus on engineering productivity itself to be very valuable and important as the company scales and can leverage insights from such a team.

Google是一家資料驅動型公司。我們的大部分產品和設計決策都有可靠的資料支援。資料驅動的決策文化，使用適當的指標，有一些不足，但總的來說，依靠資料往往使大多數決策變得客觀而不是主觀，這往往是一件好事。然而，收集和分析資料是人性的弱點，有其自身的挑戰。具體來說，在軟體工程領域，Google發現，隨著公司規模的擴大，擁有一支專注於工程生產效率的專家團隊本身是非常有價值和重要的，可以利用這樣一支團隊的洞察力。

Why Should We Measure Engineering Productivity? 我們為什麼要度量工程效率

Let’s presume that you have a thriving business (e.g., you run an online search engine), and you want to increase your business’s scope (enter into the enterprise application market, or the cloud market, or the mobile market). Presumably, to increase the scope of your business, you’ll need to also increase the size of your engineering organization. However, as organizations grow in size linearly, communication costs grow quadratically.[1] Adding more people will be necessary to increase the scope of your business, but the communication overhead costs will not scale linearly as you add additional personnel. As a result, you won’t be able to scale the scope of your business linearly to the size of your engineering organization.

讓我們假設你有一個蓬勃發展的業務（例如，你經營一個線上搜尋引擎），並且你想擴大你的業務範圍（進入企業應用市場，或雲市場，或移動市場）。據推測，為了增加你的業務範圍，你也需要增加你的工程組織的規模。然而，隨著組織規模的線性增長，溝通成本也呈二次曲線增長。增加人員對擴大業務範圍是必要的，但溝通成本不會隨著你增加人員而線性擴充。因此，你將無法根據你的工程組織的規模線性地擴大你的業務範圍。

There is another way to address our scaling problem, though: *we could make each individual more productive*. If we can increase the productivity of individual engineers in the organization, we can increase the scope of our business without the commensurate increase in communication overhead.

不過，還有一種方法可以解決我們的規模問題：*我們可以使每個人的生產效率提高*。如果我們能提高組織中單個工程師的生產效率，我們就能增加我們的業務範圍，而不會相應地增加溝通成本。

Google has had to grow quickly into new businesses, which has meant learning how to make our engineers more productive. To do this, we needed to understand what makes them productive, identify inefficiencies in our engineering processes, and fix the identified problems. Then, we would repeat the cycle as needed in a continuous improvement loop. By doing this, we would be able to scale our engineering organization with the increased demand on it.

Google不得不迅速發展新業務，這意味著要學習如何讓我們的工程師更有效率。要做到這一點，我們需要了解是什麼讓他們富有成效，找出我們工程流程中的低效之處，並解決所發現的問題。然後，我們將根據需要在一個持續改進的迴圈中重複這個迴圈。透過這樣做，我們將能夠在需求增加的情況下擴充我們的工程組織。

However, this improvement cycle *also* takes human resources. It would not be worthwhile to improve the productivity of your engineering organization by the equivalent of 10 engineers per year if it took 50 engineers per year to understand and fix productivity blockers. *Therefore, our goal is to not only improve software engineering productivity, but to do so efficiently.*

然而，這個增量改進過程*同樣*需要人力資源。如果每年需要 50 個工程師來了解和解決生產力的障礙，那麼以每年 10 名工程師的數量來提高工程組織的生產力是不值當的。*因此，我們不僅要在目標上提高軟體工程的生產力，而且這一改進過程也要同樣高效。*

At Google, we addressed these trade-offs by creating a team of researchers dedicated to understanding engineering productivity. Our research team includes people from the software engineering research field and generalist software engineers, but we also include social scientists from a variety of fields, including cognitive psychology and behavioral economics. The addition of people from the social sciences allows us to not only study the software artifacts that engineers produce, but to also understand the human side of software development, including personal motivations, incentive structures, and strategies for managing complex tasks. The goal of the team is to take a data-driven approach to measuring and improving engineering productivity.

在Google，我們透過建立一個致力於瞭解工程生產效率的研究團隊來解決這些權衡問題。我們的研究團隊包括來自軟體工程研究人員和普通的軟體工程師，但我們也包括來自不同領域的社會學家，包括認知心理學和行為經濟學。來自社會科學的人員的加入使我們不僅可以研究工程師生產的軟體構件，還可以瞭解軟體開發過程中人的一面，包括個人動機、激勵結構和管理複雜任務的策略。該團隊的目標是採用資料驅動的方法來度量和提高工程生產效率。

In this chapter, we walk through how our research team achieves this goal. This begins with the triage process: there are many parts of software development that we *can* measure, but what *should* we measure? After a project is selected, we walk through how the research team identifies meaningful metrics that will identify the problematic parts of the process. Finally, we look at how Google uses these metrics to track improvements to productivity.

在本章中，我們將介紹我們的研究團隊是如何實現這一目標的。這從分類過程開始：我們對軟體開發的許多部分*可以*進行計量，但是我們到底應該計量什麼呢？在一個專案被選中後，我們將介紹研究團隊如何確定有意義的指標，以確定該過程中存在問題的部分。最後，我們看一下Google是如何使用這些指標來追蹤生產效率的改進。

For this chapter, we follow one concrete example posed by the C++ and Java language teams at Google: readability. For most of Google's existence, these teams have managed the readability process at Google. (For more on readability, see Chapter 3.) The readability process was put in place in the early days of Google, before automatic formatters (Chapter 8 and linters that block submission were commonplace (Chapter 9). The process itself is expensive to run because it requires hundreds of engineers performing readability reviews for other engineers in order to grant readability to them. Some engineers viewed it as an archaic hazing process that no longer held utility, and it was a favorite topic to argue about around the lunch table. The concrete question from the language teams was this: is the time spent on the readability process worthwhile?

在本章中，我們遵循Google的 C++和 Java 語言團隊提出的一個具體例子：可讀性。在Google存在的大部分時間裡，這些團隊一直在管理Google的可讀性過程。(關於可讀性的更多資訊，請參見第三章）。可讀性過程是在Google的早期建立的，當時自動格式化工具（第 8 章）和阻止提交的鎖定還沒有普及（第 9 章）。這個過程本身執行成本很高，因為它需要數以百計的工程師為其他工程師進行可讀性審查，以便授予他們可讀性。一些工程師認為這是一個古老的自欺欺人過程，不再具有實用性，這也是午餐桌上最喜歡爭論的話題。來自語言團隊的具體問題是：花在可讀性過程上的時間是值得的嗎？

> 1 Frederick P.Brooks，《人月神話：軟體工程隨筆》（紐約：Addison Wesley，1995）。

Triage: Is It Even Worth Measuring? 分類：是否值得度量？

Before we decide how to measure the productivity of engineers, we need to know when a metric is even worth measuring. The measurement itself is expensive: it takes people to measure the process, analyze the results, and disseminate them to the rest of the company. Furthermore, the measurement process itself might be onerous and slow down the rest of the engineering organization. Even if it is not slow, tracking progress might change engineers' behavior, possibly in ways that mask the underlying issues. We need to measure and estimate smartly; although we don't want to guess, we shouldn't waste time and resources measuring unnecessarily.

在我們決定如何度量工程師的生產效率之前，我們需要知道某個指標是否值得度量。度量本身是昂貴的：它需要人去度量過程，分析結果，並將其傳播給公司的其他部門。此外，度量過程本身可能是繁瑣的，會拖累工程組織的其他部門。即使它不慢，追蹤進度也可能改變工程師的行為，可能會掩蓋潛在的問題。我們需要聰明地度量和估計；雖然我們不想猜測，但我們不應該浪費時間和資源進行不必要的度量。

At Google, we've come up with a series of questions to help teams determine whether it's even worth measuring productivity in the first place. We first ask people to describe what they want to measure in the form of a concrete question; we find that the more concrete people can make this question, the more likely they are to derive benefit from the process. When the readability team approached us, its question was simple: are the costs of an engineer going through the readability process worth the benefits they might be deriving for the company?

在Google，我們想出了一系列的問題來幫助團隊確定是否值得優先度量生產效率。我們首先要求人們以具體問題的形式描述他們想要度量的東西；我們發現，人們提出這個問題越具體，他們就越有可能從這個過程中獲益。當可讀性團隊與我們接觸時，其問題很簡單：工程師在提高可讀性過程中的成本增加是否匹配他們為公司帶來的好處？

We then ask them to consider the following aspects of their question:

What result are you expecting, and why?
Even though we might like to pretend that we are neutral investigators, we are not. We do have preconceived notions about what ought to happen. By acknowledging this at the outset, we can try to address these biases and prevent post hoc explanations of the results.
When this question was posed to the readability team, it noted that it was not sure. People were certain the costs had been worth the benefits at one point in time, but with the advent of autoformatters and static analysis tools, no one was entirely certain. There was a growing belief that the process now served as a hazing ritual. Although it might still provide engineers with benefits (and they had survey data showing that people did claim these benefits), it was not clear whether it was worth the time commitment of the authors or the reviewers of the code.

If the data supports your expected result, what action will be taken?
We ask this because if no action will be taken, there is no point in measuring. Notice that an action might in fact be “maintain the status quo” if there is a planned change that will occur if we didn’t have this result.
When asked about this, the answer from the readability team was straightforward: if the benefit was enough to justify the costs of the process, they would link to the research and the data on the FAQ about readability and advertise it to set expectations.

If we get a negative result, will appropriate action be taken?
We ask this question because in many cases, we find that a negative result will not change a decision. There might be other inputs into a decision that would override any negative result. If that is the case, it might not be worth measuring in the first place. This is the question that stops most of the projects that our research team takes on; we learn that the decision makers were interested in knowing the results, but for other reasons, they will not choose to change course.
In the case of readability, however, we had a strong statement of action from the team. It committed that, if our analysis showed that the costs either outweighed the benefit or the benefits were negligible, the team would kill the process. As different programming languages have different levels of maturity in formatters and static analyses, this evaluation would happen on a per-language basis.

Who is going to decide to take action on the result, and when would they do it?
We ask this to ensure that the person requesting the measurement is the one who is empowered to take action (or is doing so directly on their behalf). Ultimately, the goal of measuring our software process is to help people make business decisions. It’s important to understand who that individual is, including what form of data convinces them. Although the best research includes a variety of approaches (everything from structured interviews to statistical analyses of logs), there might be limited time in which to provide decision makers with the data they need. In those cases, it might be best to cater to the decision maker. Do they tend to make decisions by empathizing through the stories that can be retrieved from interviews?[2] Do they trust survey results or logs data? Do they feel comfortable with complex statistical analyses? If the decider doesn’t believe the form of the result in principle, there is again no point in measuring the process.
In the case of readability, we had a clear decision maker for each programming language. Two language teams, Java and C++,

actively reached out to us for assistance, and the others were waiting to see what happened with those languages first.[3] The decision makers trusted engineers' self-reported experiences for understanding happiness and learning, but the decision makers wanted to see "hard numbers" based on logs data for velocity and code quality. This meant that we needed to include both qualitative and quantitative analysis for these metrics. There was not a hard deadline for this work, but there was an internal conference that would make for a useful time for an announcement if there was going to be a change. That deadline gave us several months in which to complete the work.

然後我們要求他們考慮以下問題：

你期望的結果是什麼？為什麼？
儘管我們可能想假裝我們是中立的調查人員，但事實並非如此。我們確實對一些事有先入為主的觀念。透過一開始就承認這一點，我們可以嘗試解決這些偏見，防止對結果進行事後解釋。
當這個問題被提給可讀性小組時，該小組指出，它並不確定。人們確信在某個時間點上，成本是值得的，但是隨著自動格式化和靜態分析工具的出現，沒有人完全確定。越來越多的人認為，這個過程現在成了一種自欺欺人的儀式。雖然它可能仍然為工程師提供了好處（他們有調查資料顯示人們確實聲稱有這些好處），但不清楚它是否值得作者或程式碼審查員投入時間。

如果資料支援你的預期結果，將採取什麼行動
我們這樣問是因為如果不採取任何行動，那麼度量就沒有意義了。請注意，如果沒有這一結果，就會發生計劃變更，那麼行動實際上可能是"維持現狀"。
當被問及這個問題時，可讀性團隊的回答很直截了當：如果好處足以證明這個過程的成本是合理的，他們會連結到關於可讀性的 FAQ 上的研究和資料，並進行宣傳以設定期望。

如果我們得到一個負面的結果，是否會採取適當的行動？
我們問這個問題是因為在許多情況下，我們發現負面結果不會改變決策。決策中可能會有其他的投入，而這些投入將取代任何負面的結果。如果是這樣的話，可能一開始就不值得度量。這也是阻止我們研究團隊所做的大多數專案的問題；我們瞭解到決策者對了解結果感興趣，但由於其他原因，他們不會選擇改變方向。
然而，在可讀性的案例中，我們有一個來自團隊的強有力的行動宣告。它承諾，如果我們的分析顯示成本大於收益，或者收益可以忽略不計，團隊將放棄這個專案。由於不同的程式語言在格式化和靜態分析方面有不同的成熟度，因此該評估將基於每種語言進行。

誰將決定對結果採取行動，以及他們何時採取行動？
我們這樣問是為了確保要求度量的人是被授權採取行動的人（或直接代表他們採取行動）。歸根結底，度量我們的軟體流程的目的是幫助人們做出業務決策。瞭解這個人是誰很重要，包括什麼形式的資料能說服他們。儘管最好的研究包括各種方法（從結構化訪談到日誌的統計分析等各種方法），但為決策者提供他們需要的資料的時間可能有限。在這些情況下，最好的辦法是迎合決策者的要求。他們是否傾向於透過訪談中可以獲取到的故事來做出決策？他們是否信任調查結果或日誌資料？他們對複雜的統計分析感到滿意嗎？如果決策者壓根就不相信結果的形式，那麼度量過程又沒有意義。
在可讀性方面，我們對每種程式語言都有一個明確的決策者。有兩個語言團隊，即 Java 和 C++，積極向我們尋求幫助，而其他團隊則在等待，看這些語言先發生什麼。決策者相信工程師自我報告的經驗，以瞭解快樂和學習，但決策者希望看到基於日誌資料的速度和程式碼品質的 "硬數字"。這意味著，我們需要對這些指標進行定性和定量分析。這項工作沒有一個硬性的截止日期，但有一個內部會議，如果有變化的話，這個會議將宣佈一個新的時間點。這個期限給了我們幾個月的時間來完成這項工作。

By asking these questions, we find that in many cases, measurement is simply not worthwhile…and that's OK! There are many good reasons to not measure the impact of a tool or process on productivity. Here are some examples that we've seen:

You can’t afford to change the process/tools right now
 There might be time constraints or financial constraints that prevent this. For example, you might determine that if only you switched to a faster build tool, it would save hours of time every week. However, the switchover will mean pausing development while everyone converts over, and there’s a major funding deadline approaching such that you cannot afford the interruption. Engineering trade-offs are not evaluated in a vacuum—in a case like this, it’s important to realize that the broader context completely justifies delaying action on a result.

Any results will soon be invalidated by other factors
 Examples here might include measuring the software process of an organization just before a planned reorganization. Or measuring the amount of technical debt for a deprecated system.
 The decision maker has strong opinions, and you are unlikely to be able to provide a large enough body of evidence, of the right type, to change their beliefs.
 This comes down to knowing your audience. Even at Google, we sometimes find people who have unwavering beliefs on a topic due to their past experiences. We have found stakeholders who never trust survey data because they do not believe self-reports. We’ve also found stakeholders who are swayed best by a compelling narrative that was informed by a small number of interviews. And, of course, there are stakeholders who are swayed only by logs analysis. In all cases, we attempt to triangulate on the truth using mixed methods, but if a stakeholder is limited to believing only in methods that are not appropriate for the problem, there is no point in doing the work.

The results will be used only as vanity metrics to support something you were going to do anyway
 This is perhaps the most common reason we tell people at Google not to measure a software process. Many times, people have planned a decision for multiple reasons, and improving the software development process is only one benefit of several. For example, the release tool team at Google once requested a measurement to a planned change to the release workflow system. Due to the nature of the change, it was obvious that the change would not be worse than the current state, but they didn’t know if it was a minor improvement or a large one. We asked the team: if it turns out to only be a minor improvement, would you spend the resources to implement the feature anyway, even if it didn’t look to be worth the investment? The answer was yes! The feature happened to improve productivity, but this was a side effect: it was also more performant and lowered the release tool team’s maintenance burden.

The only metrics available are not precise enough to measure the problem and can be confounded by other factors
 In some cases, the metrics needed (see the upcoming section on how to identify metrics) are simply unavailable. In these cases, it can be tempting to measure using other metrics that are less precise (lines of code written, for example). However, any results from these metrics will be uninterpretable. If the metric confirms the stakeholders’ preexisting beliefs, they might end up proceeding with their plan without consideration that the metric is not an accurate measure. If it does not confirm their beliefs, the imprecision of the metric itself provides an easy explanation, and the stakeholder might, again, proceed with their plan.

透過問這些問題，我們發現在許多情況下，度量根本不值得……這沒有關係！有許多很好的理由不度量一個工具或過程對生產效率的影響。以下是我們看到的一些例子：

至少在現階段，你承擔不了改變這個過程/工具的成本
 可能有時間上的限制或資金上的制約，使之無法進行。例如，你可能確定，只要你切換到一個更快的建構工具，每週就能節省幾個小時的時間。然而，轉換意味著在每個人都轉換的時候暫停開發，而且有一個重要的資金期限即將到來，這樣你就無法承受這種中斷。工程權衡不是在真空中評估的——在這樣的情況下，重要的是要意識到，更廣泛的背景完全可以說明推遲對結果採取行

動是合理的。

任何結果很快就會因其他因素而失效
 這裡的例子可能包括在計劃重組之前度量一個組織的軟體流程。或者度量一個被廢棄的系統的技術債務的數量。
 決策者有強烈的意見，而你不太可能提供足夠多的正確型別的證據，來改變他們的信念。
 這就需要了解你的受眾。即使在Google，我們有時也會發現一些人由於他們過去的經驗而對某一主題有堅定的信念。我們曾發現一些利益相關者從不相信調查資料，因為他們不相信自我觀念。我們也發現一些利益相關者，他們最容易被由少量訪談得出的令人信服的敘述所動搖。當然，也有一些利益相關者只被日誌分析所動搖。在所有情況下，我們都試圖用混合方法對真相進行三角分析，但如果利益相關者只限於相信不適合問題的方法，那麼做這項工作就沒有意義。

結果只能作為浮華的指標，以來支援你一定要做的事情
 這也許是我們在Google告訴人們不要度量軟體過程的最常見的原因。很多時候，人們已經為多個原因規劃了一個決策，而改進軟體開發過程只是這些決策的一個好處。例如，Google的釋出工具團隊曾經要求對釋出工作流程系統的計劃變更進行度量。由於變化的性質，很明顯，這個變化不會比目前的狀態差，但他們不知道這是一個小的改進還是一個大的改進。我們問團隊：如果結果只是一個小的改進，無論如何你會花資源來實現這個功能，即使它看起來不值得投資？答案是肯定的! 這個功能碰巧提高了生產效率，但這是一個副作用：它也更具有效能，降低了釋出工具團隊的維護負擔。

唯一可用的指標不夠精確，無法度量問題，而且會被其他因素所幹擾
 在某些情況下，所需的指標（見即將到來的關於如何識別指標的章節）根本無法獲得。在這些情況下，使用其他不那麼精確的指標（例如，編寫的程式碼行）進行度量是很誘人的。然而，這些指標的任何結果都是無法解釋的。如果這個指標證實了利益相關者預先存在的觀念，他們最終可能會繼續執行他們的計劃，而不考慮這個指標不是一個準確的度量標準。如果它沒有證實他們的觀念，那麼指標本身的不精確性就提供了一個簡單的解釋，利益相關者可能再次繼續他們的計劃。

When you are successful at measuring your software process, you aren't setting out to prove a hypothesis correct or incorrect; *success means giving a stakeholder the data they need to make a decision*. If that stakeholder won't use the data, the project is always a failure. We should only measure a software process when a concrete decision will be made based on the outcome. For the readability team, there was a clear decision to be made. If the metrics showed the process to be beneficial, they would publicize the result. If not, the process would be abolished. Most important, the readability team had the authority to make this decision.

當你成功地度量你的軟體過程時，你並不是為了證明一個假設的正確與否；*成功意味著給利益相關者提供他們做出決定所需的資料*。如果這個利益相關者不使用這些資料，那麼這個專案就是失敗的。我們只應該在根據結果做出具體決定的時候才去度量一個軟體過程。對於可讀性團隊來說，有一個明確的決定要做。如果度量標準顯示這個過程是有益的，他們將公佈這個結果。如果沒有，這個過程就會被廢除。最重要的是，可讀性小組有權力做出這個決定。

2 在此值得指出的是，我們的行業目前貶低 "軼事資料"，而每個人都有一個 "資料驅動 "的目標。然而，軼事仍然存在，因為它們是強大的。軼事可以提供原始數字無法提供的背景和敘述；它可以提供一個深刻的解釋，因為它反映了個人的經驗，所以能引起別人的共鳴。雖然我們的研究人員不會根據軼事做出決定，但我們確實使用並鼓勵結構化訪談和案例研究等技術，以深入理解現象，併為定量資料提供背景。

3 Java 和 C++擁有最大量的工具支援。兩者都有成熟的格式化工具和靜態分析工具，可以捕捉常見的錯誤。兩者也都有大量的內部資金。即使其他語言團隊，如 Python，對結果感興趣，但顯然，如果我們連 Java 或 C++都不能顯示出同樣的好處，那麼 Python 就不會有刪除可讀性的好處。

Selecting Meaningful Metrics with Goals and Signals 用目標和訊號來選擇有意義的度量標準

After we decide to measure a software process, we need to determine what metrics to use. Clearly, lines of code (LOC) won't do,[4] but how do we actually measure engineering productivity?

在我們決定度量一個軟體過程之後，我們需要確定使用什麼指標。顯然，程式碼行（LOC）是不行的，但我們究竟該如何度量工程生產效率呢？

At Google, we use the Goals/Signals/Metrics (GSM) framework to guide metrics creation.

在Google，我們使用目標/訊號/指標（GSM）框架來指導指標建立。

- A *goal* is a desired end result. It's phrased in terms of what you want to understand at a high level and should not contain references to specific ways to measure it.
- A signal is how you might know that you've achieved the end result. Signals are things we would *like* to measure, but they might not be measurable themselves.
- A *metric* is proxy for a signal. It is the thing we actually can measure. It might not be the ideal measurement, but it is something that we believe is close enough.
- *目標* 是一個期望的最終結果。它是根據你希望在高層次上理解的內容來表述的，不應包含對具體度量方法的參考。。
- *訊號* 是你如何知道你已經實現了最終結果。訊號是我們*想要*度量的東西，但它們本身可能是不可度量的。
- *指標* 是訊號的代表。它是我們實際上可以度量的東西。它可能不是理想的度量，但它是我們認為足夠接近的東西。

The GSM framework encourages several desirable properties when creating metrics. First, by creating goals first, then signals, and finally metrics, it prevents the *streetlight effect*. The term comes from the full phrase "looking for your keys under the streetlight": if you look only where you can see, you might not be looking in the right place. With metrics, this occurs when we use the metrics that we have easily accessible and that are easy to measure, regardless of whether those metrics suit our needs. Instead, GSM forces us to think about which metrics will actually help us achieve our goals, rather than simply what we have readily available.

GSM 框架在建立指標時鼓勵幾個理想的屬性。首先，透過首先建立目標，然後是訊號，最後是指標，它可以防止*路燈*效應。這個詞來自於 "在路燈下找你的鑰匙 "這個完整的短語：如果你只看你能看到的地方，你可能沒有找對地方。對於指標來說，當我們使用我們容易獲得的、容易度量的指標時，就會出現這種情況，不管這些指標是否適合我們的需求。相反，GSM 迫使我們思考哪些指標能真正幫助我們實現目標，而不是簡單地考慮我們有哪些現成的指標。

Second, GSM helps prevent both metrics creep and metrics bias by encouraging us to come up with the appropriate set of metrics, using a principled approach, *in advance* of actually measuring the result. Consider the case in which we select metrics without a principled approach and then the results do not meet our stakeholders' expectations. At that point, we run the risk that stakeholders will propose that we use different metrics that they believe will produce the desired result. And because we didn't select based on a principled approach at the start, there's no reason to say that they're wrong! Instead, GSM encourages us to select metrics based on their ability to measure the original goals. Stakeholders can easily see that these metrics map to their original goals and agree, in advance, that this is the best set of metrics for measuring the outcomes.

第二，GSM 透過鼓勵我們使用原則性的方法提出適當的指標集，從而有助於防止指標蔓延和指標偏差，從而有助於實際度量結果。考慮這樣一種情況，我們在沒有原則性方法的情況下選擇指標，然後結果不符合我們的利益相關者的期望。在這一點上，我們面臨著利益相關者建議我們使用他們認為會產生預期結果的不同指標的風險。而且因為我們一開始並沒有基於原則性的方法進行選擇，所以沒有理由說他們錯了！相反，GSM 鼓勵我們根據度量原始目標的能力選擇指標。利益相關者可以很容易地看到這些指標對映到他們的最初的目標，並提前同意這是度量結果的最佳指標集。

Finally, GSM can show us where we have measurement coverage and where we do not. When we run through the GSM process, we list all our goals and create signals for each one. As we will see in the examples, not all signals are going to be measurable and that's OK! With GSM, at least we have identified what is not measurable. By identifying these missing metrics, we can assess whether it is worth creating new metrics or even worth measuring at all.

最後，GSM 可以告訴我們哪裡有度量覆蓋，哪裡沒有。當我們執行 GSM 流程時，我們列出所有的目標，併為每個目標建立訊號。正如我們在例子中所看到的，並不是所有的訊號都是可度量的，這沒關係！透過 GSM，至少我們已經確定了什麼是可度量的。透過 GSM，至少我們已經確定了哪些是不可度量的。透過識別這些缺失的指標，我們可以評估是否值得建立新的指標，甚至是否值得度量。

The important thing is to maintain *traceability*. For each metric, we should be able to trace back to the signal that it is meant to be a proxy for and to the goal it is trying to measure. This ensures that we know which metrics we are measuring and why we are measuring them.

重要的是要保持*可追溯性*。對於每個指標，我們應該能夠追溯到它所要代表的訊號，以及它所要度量的目標。這可以確保我們知道我們正在度量哪些指標，以及為什麼我們要度量它們。

> 4 “從那時起，用‘每月產生的程式碼行數’來度量‘程式設計師生產效率’只需一小步。這是一個非常昂貴的度量單位，因為它鼓勵編寫平淡的程式碼，但今天我對這個單位的愚蠢程度不感興趣，甚至從純商業的角度來看也是如此。我今天的觀點是，如果我們希望計算程式碼的行數，我們不應該將它們視為“生產的行數”，而應該視為“花費的行數”：當前的傳統智慧愚蠢到將這些行數記在賬本的錯誤一側。”Edsger Dijkstra，關於真正教給計算機科學的殘酷性，EWD 手稿 1036。

Goals 目標

A goal should be written in terms of a desired property, without reference to any metric. By themselves, these goals are not measurable, but a good set of goals is something that everyone can agree on before proceeding onto signals and then metrics.

目標應該根據所需的屬性來編寫，而不需要參考任何指標。就其本身而言，這些目標是無法度量的，但一組好的目標是每個人都可以在繼續進行訊號和度量指標之前達成一致的。

To make this work, we need to have identified the correct set of goals to measure in the first place. This would seem straightforward: surely the team knows the goals of their work! However, our research team has found that in many cases, people forget to include all the possible *trade-offs within productivity*, which could lead to mismeasurement.

為了使其發揮作用，我們首先需要確定一套正確的目標來度量。這看起來很簡單：團隊肯定知道他們工作的目標！但是，我們的研究團隊發現，在許多情況下，人們忘記了將所有可能的*權衡因素包括在生產效率中*。然而，我們的研究團隊發現，在許多情況下，人們忘記了將所有可能的*生產力內的權衡因素包括在內*，這可能導致錯誤的度量。

Taking the readability example, let's assume that the team was so focused on making the readability process fast and easy that it had forgotten the goal about code quality. The team set up tracking measurements for how long it takes to get through the review process and how happy engineers are with the process. One of our teammates proposes the following:

> I can make your review velocity very fast: just remove code reviews entirely.

以可讀性為例，我們假設團隊太專注於使可讀性過程快速和簡單，以至於忘記了關於程式碼品質的目標。團隊設定了追蹤度量，以瞭解透過審查過程需要多長時間，以及工程師對該過程的滿意程度。我們的一個隊友提出以下建議：

> 我可以讓你的審查速度變得非常快：只要完全取消程式碼審查。

Although this is obviously an extreme example, teams forget core trade-offs all the time when measuring: they become so focused on improving velocity that they forget to measure quality (or vice versa). To combat this, our research team divides productivity into five core components. These five components are in trade-off with one another, and we encourage teams to consider goals in each of these components to ensure that they are not inadvertently improving one while driving others downward. To help people remember all five components, we use the mnemonic "QUANTS":

__Quality__ of the code
 What is the quality of the code produced? Are the test cases good enough to prevent regressions? How good is an architecture at mitigating risk and changes?

__Attention__ from engineers
 How frequently do engineers reach a state of flow? How much are they distracted by notifications? Does a tool encourage engineers to context switch?

Intellectual complexity
 How much cognitive load is required to complete a task? What is the inherent complexity of the problem being solved? Do engineers need to deal with unnecessary complexity?

Tempo and velocity
 How quickly can engineers accomplish their tasks? How fast can they push their releases out? How many tasks do they complete in a given timeframe?

Satisfaction
 How happy are engineers with their tools? How well does a tool meet engineers' needs? How satisfied are they with their work and their end product? Are engineers feeling burned out?

雖然這顯然是一個極端的例子，但團隊在度量時總是忘記了核心的權衡：他們太專注於提高速度而忘記了度量品質（或者反過來）。為了解決這個問題，我們的研究團隊將生產效率分為五個核心部分。這五個部分是相互權衡的，我們鼓勵團隊考慮每一個部分的目標，以確保他們不會在無意中提高一個部分而使其他部分下降。為了幫助人們記住所有五個組成部分，我們使用了 "QUANTS "的記憶法：

程式碼的***品質***
 產生的程式碼的品質如何？測試使用案例是否足以預防迴歸？架構在減輕風險和變化方面的能力如何？

工程師的***關注度***
 工程師達到流動狀態的頻率如何？他們在多大程度上被通知分散了注意力？工具是否鼓勵工程師進行狀態切換？

知識的複雜性
 完成一項任務需要多大的認知負荷？正在解決的問題的內在複雜性是什麼？工程師是否需要處理不必要的複雜性？

節奏和速度
 工程師能多快地完成他們的任務？他們能以多快的速度把他們的版本推出去？他們在給定的時間範圍內能完成多少任務？

滿意程度
 工程師對他們的工具有多滿意？工具能在多大程度上滿足工程師的需求？他們對自己的工作和最終產品的滿意度如何？工程師是否感到筋疲力盡？

Going back to the readability example, our research team worked with the readability team to identify several productivity goals of the readability process:

Quality of the code
 Engineers write higher-quality code as a result of the readability process; they write more consistent code as a result of the readability process; and they contribute to a culture of code health as a result of the readability process.

Attention from engineers
 We did not have any attention goal for readability. This is OK! Not all questions about engineering productivity involve trade-offs in all five areas.

Intellectual complexity
 Engineers learn about the Google codebase and best coding practices as a result of the readability process, and they receive mentoring during the readability process.

Tempo and velocity
 Engineers complete work tasks faster and more efficiently as a result of the readability process.

Satisfaction
 Engineers see the benefit of the readability process and have positive feelings about participating in it.

回到可讀性的例子，我們的研究團隊與可讀性團隊合作，確定了可讀性過程中的幾個生產力目標：

程式碼的品質
 由於可讀性過程，工程師們寫出了更高品質的程式碼；由於可讀性過程，他們寫出了更一致的程式碼；由於可讀性過程，他們為程式碼的健康文化做出了貢獻。

來自工程師的關注
 我們沒有為可讀性制定任何關注目標。這是可以的! 並非所有關於工程生產力的問題都涉及所有五個領域的權衡。

知識複雜性
 工程師們透過可讀性過程瞭解Google程式碼庫和最佳編碼實踐，他們在可讀性過程中接受指導。

節奏和速度
 由於可讀性過程，工程師更快、更有效地完成工作任務。

滿意度
 工程師們看到了可讀性過程的好處，對參與該過程有積極的感受。

Signals 訊號

A signal is the way in which we will know we've achieved our goal. Not all signals are measurable, but that's acceptable at this stage. There is not a 1:1 relationship between signals and goals. Every goal should have at least one signal, but they might have more. Some goals might also share a signal. Table 7-1 shows some example signals for the goals of the readability process measurement.

透過約定的訊號，我們可以知曉某個目標已被實現。並非所有的訊號都是可度量的，但這在現階段是可以接受的。訊號和目標之間不是 1:1 的關係。每個目標應該至少有一個訊號，但它們可能有更多的訊號。有些目標也可能共享一個訊號。表 7-1 顯示了可讀性過程度量的目標的一些訊號範例。

Table 7-1. Signals and goals _表 7-1. 訊號和目標 _

Goals	Signals
Engineers write higher-quality code as a result of the readability process.	Engineers who have been granted readability judge their code to be of higher quality than engineers who have not been granted readability. The readability process has a positive impact on code quality.
Engineers learn about the Google codebase and best coding practices as a result of the readability process.	Engineers report learning from the readability process.
Engineers receive mentoring during the readability process.	Engineers report positive interactions with experienced Google engineers who serve as reviewers during the readability process.
Engineers receive mentoring during the readability process. Engineers complete work tasks faster and more efficiently as a result of the readability process. Engineers see the benefit of the readability process and have positive feelings about participating in it.	Engineers who have been granted readability judge themselves to be more productive than engineers who have not been granted readability. Changes written by engineers who have been granted readability are faster to review than changes written by engineers who have not been granted readability. Engineers view the readability process as being worthwhile.

目標	訊號
由於可讀性過程，工程師們會寫出更高品質的程式碼。	被授予可讀性的工程師判斷他們的程式碼比沒有被授予可讀性的工程師的品質更高。可讀性過程對程式碼品質有積極影響。
工程師們透過可讀性過程瞭解Google的程式碼庫和最佳編碼實踐。	工程師們報告了從可讀性過程中的學習情況。
工程師在可讀性過程中接受指導。	工程師們報告了與經驗豐富的Google工程師的積極互動，他們在可讀性過程中擔任審查員。
工程師在可讀性過程中得到指導。 由於可讀性過程，工程師們更快、更有效地完成工作任務。 工程師們看到了可讀性過程的好處，並對參與這一過程有積極的感受。	被授予可讀性的工程師判斷自己比沒有被授予可讀性的工程師更有生產效率。被授予可讀性的工程師所寫的修改比未被授予可讀性的工程師所寫的修改審查得更快。 工程師認為可讀性過程是值得的。

Metrics 指標

Metrics are where we finally determine how we will measure the signal. Metrics are not the signal themselves; they are the measurable proxy of the signal. Because they are a proxy, they might not be a perfect measurement. For this reason, some signals might have multiple metrics as we try to triangulate on the underlying signal.

指標是我們最終確定如何計量、評判訊號的標準。指標不是訊號本身；它們是訊號的可度量的代表。因為它們是一個代表，所以它們可能不是一個完美的度量。出於這個原因，我們試圖對基本訊號進行三角度量分析，一些訊號可能有多個指標。

For example, to measure whether engineers' code is reviewed faster after readability, we might use a combination of both survey data and logs data. Neither of these metrics really provide the underlying truth. (Human perceptions are fallible, and logs metrics might not be measuring the entire picture of the time an engineer spends reviewing a piece of code or can be confounded by factors unknown at the time, like the size or difficulty of a code change.) However, if these metrics show different results, it signals that possibly one of them is incorrect and we need to explore further. If they are the same, we have more confidence that we have reached some kind of truth.

例如，為了度量工程師的程式碼在可讀性之後是否審查得更快，我們可能會同時使用調查資料和日誌資料。這兩個指標都沒有真正提供基本的事實。(人類別的感知是易變的，而日誌指標可能沒有度量出工程師審查一段程式碼所花時間的全貌，或者可能被當時未知的因素所混淆，比如程式碼修改的大小或難度)。然而，如果這些指標顯示出不同的結果，就表明可能其中一個指標是不正確的，我們需要進一步探索。如果它們是一樣的，我們就更有信心，我們已經達到了某種真相。

Additionally, some signals might not have any associated metric because the signal might simply be unmeasurable at this time. Consider, for example, measuring code quality. Although academic literature has proposed many proxies for code quality, none of them have truly captured it. For readability, we had a decision of either using a poor proxy and possibly making a decision based on it, or simply acknowledging that this is a point that cannot currently be measured. Ultimately, we decided not to capture this as a quantitative measure, though we did ask engineers to self-rate their code quality.

此外，一些訊號可能沒有任何相關的指標，因為訊號可能在這個時候根本無法度量。例如，考慮度量程式碼品質。儘管學術文獻已經提出了許多程式碼品質的代用指標，但沒有一個能真正抓住它。對於可讀性，我們必須做出決定，要麼使用一個糟糕的代表，並可能根據它做出決定，要麼乾脆承認這是一個目前無法度量的點。最終，我們決定不把它作為一個量化的指標，儘管我們確實要求工程師對他們的程式碼品質進行自我評價。

Following the GSM framework is a great way to clarify the goals for why you are measuring your software process and how it will actually be measured. However, it's still possible that the metrics selected are not telling the complete story because they are not capturing the desired signal. At Google, we use qualitative data to validate our metrics and ensure that they are capturing the intended signal.

遵循 GSM 框架是一個很好的方法，可以明確你為什麼要度量你的軟體過程的目標，以及它將如何被實際度量。然而，仍然有可能選擇的指標沒有說明全部情況，因為它們沒有捕獲所需的訊號。在Google，我們使用定性資料來驗證我們的指標，並確保它們捕捉到了預期的訊號。

Using Data to Validate Metrics 使用資料驗證指標

As an example, we once created a metric for measuring each engineer's median build latency; the goal was to capture the "typical experience" of engineers' build latencies. We then ran an *experience sampling study*. In this style of study, engineers are interrupted in context of doing a task of interest to answer a few questions. After an engineer started a build, we automatically sent them a small survey about their experiences and expectations of build latency. However, in a few cases, the engineers responded that they had not started a build! It turned out that automated tools were starting up builds, but the engineers were not blocked on these results and so it didn't "count" toward their "typical experience." We then adjusted the metric to exclude such builds.[5]

舉個例子，我們曾經建立了一個度量每個工程師平均建構延遲中位數的指標；目的是為了捕捉工程師建構延遲的 "典型經驗"。然後我們進行了一個*經驗抽樣研究*。在這種研究方式中，工程師在做一項感興趣的任務時被打斷，以回答一些問題。在工程師開始建構後，我們自動向他們傳送了一份小型調查，瞭解他們對建構延遲的經驗和期望。然而，在少數情況下，工程師們回答說他們沒有開始建構！結果發現，自動化工具正在啟動。事實證明，自動化工具正在啟動建構，但工程師們並沒有被這些結果所阻礙，因此它並沒有"計入"他們的"典型經驗"。然後我們調整了指標，排除了這種建構。

Quantitative metrics are useful because they give you power and scale. You can measure the experience of engineers across the entire company over a large period of time and have confidence in the results. However, they don't provide any context or narrative. Quantitative metrics don't explain why an engineer chose to use an antiquated tool to accomplish their task, or why they took an unusual workflow, or why they circumvented a standard process. Only qualitative studies can provide this information, and only qualitative studies can then provide insight on the next steps to improve a process.

定量指標是有用的，因為它們給你能力和規模。你可以在很長一段時間內度量整個公司的工程師的經驗，並對結果有信心。然而，它們並不提供任何背景或敘述。定量指標不能解釋為什麼一個工程師選擇使用一個過時的工具來完成他們的任務，或者為什麼他們採取了一個不尋常的工作流程，或者為什麼他們繞過了一個標準流程。只有定性研究才能提供這些資訊，也只有定性研究才能為改進流程的下一步提供洞察力。

> 5 我們在Google的常規經驗是，當定量指標和定性指標不一致時，是因為定量指標沒有捕捉到預期的結果。

Consider now the signals presented in Table 7-2. What metrics might you create to measure each of those? Some of these signals might be measurable by analyzing tool and code logs. Others are measurable only by directly asking engineers. Still others might not be perfectly measurable—how do we truly measure code quality, for example?

現在考慮一下表 7-2 中提出的訊號。你可以建立什麼指標來度量其中的每一個？其中一些訊號可能是可以透過分析工具和程式碼日誌來度量的。其他的只能透過直接詢問工程師來度量。還有一些可能不是完全可度量的——例如，我們如何真正度量程式碼品質？

Ultimately, when evaluating the impact of readability on productivity, we ended up with a combination of metrics from three sources. First, we had a survey that was specifically about the readability process. This survey was given to people after they completed the process; this allowed us to get their immediate feedback about the process. This hopefully avoids recall bias,[6] but it does introduce both recency bias[7] and sampling bias.[8] Second, we used a large-scale quarterly survey to track items that were not specifically about readability; instead, they were purely about metrics that we expected readability should affect. Finally, we used fine-grained logs metrics from our developer tools to determine how much time the logs claimed it took engineers to complete specific tasks.[9] Table 7-2 presents the complete list of metrics with their corresponding signals and goals.

最終，在評估可讀性對生產力的影響時，我們最終綜合了三個方面的指標。首先，我們有一個專門針對可讀性過程的調查。這個調查是在人們完成了這個過程之後進行的；這使我們能夠得到他們對這個過程的即時反饋。這有望避免回憶偏差，但它確實引入了近期偏差和抽樣偏差。其次，我們使用大規模的季度調查來追蹤那些不是專門關於可讀性的專案；相反，它們純粹是關於我們預期可讀性應該影響的指標。最後，我們使用了來自我們的開發者工具的細粒度的日誌指標來確定日誌中聲稱的工程師完成特定任務所需的時間。表 7-2 列出了完整的指標清單及其相應的訊號和目標。

Table 7-2. Goals, signals, and metrics 表 7-2. 目標、訊號和指標

QUANTS	Goal	Signal	Metric
Quality of the code	Engineers write higherquality code as a result of the readability process.	Engineers who have been granted readability judge their code to be of higher quality than engineers who have not been granted readability. The readability process has a positive	Quarterly Survey: Proportion ofengineers who report being satisfied with the quality of their own code Readability Survey: Proportion of engineers reporting that

		impact on code quality.	readability reviews have no impact or negative impact on code quality
			Readability Survey: Proportionof engineers reporting thatparticipating in the readability process has improved codequality for their team
	Engineers write more consistent code as a result of the readability process.	Engineers are given consistent feedback and direction in code reviews by readability reviewers as a part of the readability process.	Readability Survey: Proportion of engineers reporting inconsistency in readability reviewers' comments and readability criteria.
	Engineers contribute to a culture of code health as a result of the readability process.	Engineers who have been granted readability regularly comment on style and/or readability issues in code reviews.	Readability Survey: Proportion of engineers reporting that they regularly comment on style and/or readability issues in code reviews
Attention from engineers	n/a	n/a	n/a
I**n**tellectual	Engineers learn about the Google codebase and best coding practices as a result of the readability process.	Engineers report learning from the readability process.	Readability Survey: Proportion of engineers reporting that they learned about four relevant topics
			Readability Survey: Proportion of engineers reporting that learning or gaining expertise was a strength of the readability process
	Engineers receive mentoring during the readability process.	Engineers report positive interactions with experienced Google engineers who serve as reviewers during the readability process.	Readability Survey: Proportion of engineers reporting that working with readability reviewers was a strength of the readability process

Tempo/velocity	Engineers are more productive as a result of the readability process.	Engineers who have been granted readability judge themselves to be more productive than engineers who have not been granted readability.	Quarterly Survey: Proportion of engineers reporting that they're highly productive
		Engineers report that completing the readability process positively affects their engineering velocity.	Readability Survey: Proportion of engineers reporting that not having readability reduces team engineering velocity
		Changelists (CLs) written by engineers who have been granted readability are faster to review than CLs written by engineers who have not been granted readability.	Logs data: Median review time for CLs from authors with readability and without readability
		CLs written by engineers who have been granted readability are easier to shepherd through code review than CLs written by engineers who have not been granted readability.	Logs data: Median shepherding time for CLs from authors with readability and without readability
		CLs written by engineers who have been granted readability are faster to get through code review than CLs written by engineers who have not been granted readability.	Logs data: Median time to submit for CLs from authors with readability and without readability
		The readability process does not have a negative impact on engineering velocity.	Readability Survey: Proportion of engineers reporting that the readability process negatively impacts their velocity
			Readability Survey: Proportion of engineers reporting that readability reviewers responded promptly
			Readability Survey: Proportion of engineers reporting that timeliness of reviews was a strength of the readability process

Satisfaction	Engineers see the benefit of the readability process and have positive feelings about participating in it.	Engineers view the readability process as being an overall positive experience.	Readability Survey: Proportion of engineers reporting that their experience with the readability process was positive overall
		Engineers view the readability process as being worthwhile	Readability Survey: Proportion of engineers reporting that the readability process is worthwhile
			Readability Survey: Proportion of engineers reporting that the quality of readability reviews is a strength of the process
			Readability Survey: Proportion of engineers reporting that thoroughness is a strength of the process
		Engineers do not view the readability process as frustrating.	Readability Survey: Proportion of engineers reporting that the readability process is uncertain, unclear, slow, or frustrating
			Quarterly Survey: Proportion of engineers reporting that they’re satisfied with their own engineering velocity

QUANTS	目標	訊號	指標
程式碼 的品質	由於可讀性過程，工程師們會寫出更高品質的程式碼。	被授予可讀性的工程師判斷他們的程式碼比沒有被授予可讀性的工程師的品質更高。 可讀性過程對程式碼品質有積極影響。	季度調查。對自己的程式碼品質表示滿意的工程師的比例 可讀性調查。報告可讀性審查對程式碼品質沒有影響或有負面影響的工程師的比例
			可讀性調查。報告說參與可讀性過程改善了他們團隊的程式碼品質的工程師的比例

	作為可讀性過程的結果，工程師們寫出的程式碼更加一致。	工程師在程式碼審查中由可讀性審查員提供一致的反饋和指導，這是可讀性過程的一部分。	可讀性調查。報告可讀性審查員的意見和可讀性標準不一致的工程師比例。
	工程師對程式碼健康文化的貢獻是可讀性過程的結果。	被授予可讀性的工程師經常在程式碼審查中評論風格和/或可讀性問題。	可讀性調查：報告他們經常在程式碼審查中評論風格和/或可讀性問題的工程師的比例
工程師的關注	不適用	不適用	不適用
知識	工程師們透過可讀性過程瞭解Google的程式碼庫和最佳編碼實踐。	工程師們報告了從可讀性過程中的學習情況。	可讀性調查。報告說他們瞭解了四個相關主題的工程師比例
			可讀性調查：報告學習或獲得專業知識是可讀性過程的優勢的工程師比例
	工程師在可讀性過程中接受指導。	工程師們報告說，在可讀性過程中，他們與作為審查員的經驗豐富的Google工程師進行了積極的互動。	可讀性調查：報告說與可讀性審查員一起工作是可讀性過程的優勢的工程師的比例
節奏/速度	由於可讀性過程，工程師們的工作效率更高。	被授予可讀性的工程師判斷自己比沒有被授予可讀性的工程師更有生產力。	季度調查：報告他們具有高生產力的工程師的比例
		工程師們報告說，完成可讀性過程對他們的工程速度有積極的影響。	可讀性調查：報告不具備可讀性會降低團隊工程速度的工程師比例
		由被授予可讀性的工程師編寫的變更列表（CLs）比由未被授予可讀性的工程師編寫的變更列表審查得更快。	日誌資料：有可讀性和無可讀性的作者所寫的 CL 的審查時間中位數
		由被授予可讀性的工程師編寫的 CL 比由未被授予可讀性的工程師編寫的 CL 更容易透過程式碼審查。	Logs 資料：具備可讀性和不具備可讀性的作者編寫的 CL 的指導時間的中位數
		由被授予可讀性的工程師編寫的 CL 比由未被授予可讀性的工程師編寫的 CL 更快地透過程式碼審查。	日誌資料：具有可讀性和不具有可讀性的作者的 CL 提交時間的中位數

		可讀性過程不會對工程速度產生負面影響。	可讀性調查：報告可讀性過程對其速度有負面影響的工程師比例
			可讀性調查：報告可讀性審查員及時回覆的工程師比例
			可讀性調查：可讀性調查：報告審查的及時性是可讀性過程的優勢的工程師比例
滿意度	工程師們看到了可讀性過程的好處，並對參與這一過程有積極的感受。	工程師們認為可讀性過程是一個總體上積極的經歷	可讀性調查：報告說他們在可讀性過程中的經驗總體上是積極的工程師的比例
		工程師認為可讀性過程是值得的	可讀性調查：報告說可讀性過程是值得的工程師比例
			可讀性調查：報告稱可讀性審查品質是流程優勢的工程師比例
			可讀性調查：報道稱徹底性是流程的優勢的工程師比例
		工程師不認為可讀性過程是令人沮喪的。	可讀性調查：報告說可讀性過程不確定、不清楚、緩慢或令人沮喪的工程師的比例
			季度調查：報告他們對自己的工程速度感到滿意的工程師的比例

6 回憶偏差是來自記憶的偏差。人們更願意回憶那些特別有趣或令人沮喪的事件。

7 近期偏見是另一種形式的來自記憶的偏見，即人們偏向於最近的經歷。在這種情況下，由於他們剛剛成功地完成了這個過程，他們可能會感覺特別好。

8 因為我們只問了那些完成過程的人，所以我們沒有捕捉到那些沒有完成過程的人的意見。

9 有一種誘惑，就是用這樣的指標來評價個別的工程師，甚至可能用來識別高績效和低績效的人。不過，這樣做會適得其反。如果生產效率指標被用於績效評估，工程師們就會很快學會操弄這些指標，它們將不再對度量和提高整個組織的生產效率有用。讓這些指標發揮作用的唯一方法是，不將其用於度量個體，而是接受度量總體效果。

Taking Action and Tracking Results 採取行動並追蹤結果

Recall our original goal in this chapter: we want to take action and improve productivity. After performing research on a topic, the team at Google always prepares a list of recommendations for how we can continue to improve. We might suggest new features to a tool, improving latency of a tool, improving documentation, removing obsolete processes, or even changing the incentive structures for the engineers. Ideally, these recommendations are “tool driven”: it does no good to tell engineers to change their process or way of thinking if the tools do not support them in doing so. We instead always assume that engineers will make the appropriate trade-offs if they have the proper data available and the suitable tools at their disposal.

回顧我們在本章中的最初目標：我們希望採取行動，提高生產效率。在對某個主題進行研究之後，Google的團隊總是準備一份建議清單，說明我們可以如何繼續改進。我們可能會建議給一個工具增加新的功能，改善工具的延遲，改善文件，刪除過時的流程，甚至改變工程師的激勵結構。理想情況下，這些建議是 "工具驅動"的：如果工具不支援工程師改變他們的流程或思維方式，那麼告訴他們這樣做是沒有用的。相反，我們總是假設，如果工程師有適當的資料和合適的工具可以使用，他們會做出適當的權衡。

For readability, our study showed that it was overall worthwhile: engineers who had achieved readability were satisfied with the process and felt they learned from it. Our logs showed that they also had their code reviewed faster and submitted it faster, even accounting for no longer needing as many reviewers. Our study also showed places for improvement with the process: engineers identified pain points that would have made the process faster or more pleasant. The language teams took these recommendations and improved the tooling and process to make it faster and to be more transparent so that engineers would have a more pleasant experience.

就可讀性而言，我們的研究表明，總體上是值得的：實現了可讀性的工程師對這一過程感到滿意，並認為他們從中學到了東西。我們的記錄顯示，他們的程式碼審查得更快，提交得也更快，甚至不再需要那麼多的審查員。我們的研究還顯示了過程中需要改進的地方：工程師們發現了可以使過程更快、更愉快的痛點。語言團隊採納了這些建議，並改進了工具和流程，使其更快、更透明，從而使工程師有一個更愉快的體驗。

Conclusion 總結

At Google, we’ve found that staffing a team of engineering productivity specialists has widespread benefits to software engineering; rather than relying on each team to chart its own course to increase productivity, a centralized team can focus on broad- based solutions to complex problems. Such “human-based” factors are notoriously difficult to measure, and it is important for experts to understand the data being analyzed given that many of the trade-offs involved in changing engineering processes are difficult to measure accurately and often have unintended consequences. Such a team must remain data driven and aim to eliminate subjective bias.

在Google，我們發現配備一個工程生產效率專家團隊對軟體工程有廣泛的好處；與其依靠每個團隊制定自己的路線來提高生產效率，一個集中的團隊可以專注於複雜問題的廣泛解決方案。這種 "以人為本"的因素是出了名的難以度量，而且鑑於改變工程流程所涉及的許多權衡都難以準確度量，而且往往會產生意想不到的後果，因此專家們必須瞭解正在分析的資料。這樣的團隊必須保持資料驅動，旨在消除主觀偏見。

TL;DRs 內容提要

- Before measuring productivity, ask whether the result is actionable, regardless of whether the result is positive or negative. If you can’t do anything with the result, it is likely not worth measuring.
- Select meaningful metrics using the GSM framework. A good metric is a reasonable proxy to the signal you’re trying to measure, and it is traceable back to your original goals.
- Select metrics that cover all parts of productivity (QUANTS). By doing this, you ensure that you aren’t improving one aspect of productivity (like developer velocity) at the cost of another (like code quality).
- Qualitative metrics are metrics, too! Consider having a survey mechanism for tracking longitudinal metrics about engineers’ beliefs. Qualitative metrics should also align with the quantitative metrics; if they do not, it is likely the quantitative metrics that are incorrect.
- Aim to create recommendations that are built into the developer workflow and incentive structures. Even though it is sometimes necessary to recommend additional training or documentation, change is more likely to occur if it is built into the developer’s daily habits.
- 在度量生產效率之前，要問結果是否可操作，無論結果是積極還是消極。如果你對這個結果無能為力，它很可能不值得度量。
- 使用 GSM 框架選擇有意義的度量標準。一個好的指標是你試圖度量的訊號的合理代理，而且它可以追溯到你的原始目標。
- 選擇涵蓋生產效率所有部分的度量標準（QUANTS）。透過這樣做，你可以確保你不會以犧牲另一個方面（如程式碼品質）為代價來改善生產力的一個方面（如開發人員的速度）。
- 定性指標也是指標。考慮有一個調查機制來追蹤關於工程師信念的縱向指標。定性指標也應該與定量指標一致；如果它們不一致，很可能是定量指標不正確。
- 爭取建立內置於開發人員工作流程和激勵結構的建議。即使有時有必要推薦額外的培訓或文件，但如果將其納入開發人員的日常習慣，則更有可能發生變化。

---

1. Frederick P. Brooks, The Mythical Man-Month: Essays on Software Engineering (New York: Addison-Wesley, 1995). ↩

2. It’s worth pointing out here that our industry currently disparages “anecdata,” and everyone has a goal of being “data driven.” Yet anecdotes continue to exist because they are powerful. An anecdote can provide context and narrative that raw numbers cannot; it can provide a deep explanation that resonates with others because it mirrors personal experience. Although our researchers do not make decisions on anecdotes, we do use and encourage techniques such as structured interviews and case studies to deeply understand phenomena and provide context to quantitative data. ↩

3. Java and C++ have the greatest amount of tooling support. Both have mature formatters and static analysis tools that catch common mistakes. Both are also heavily funded internally. Even though other language teams, like Python, were interested in the results, clearly there was not going to be a benefit for Python to remove readability if we couldn’t even show the same benefit for Java or C++. ↩

4. “From there it is only a small step to measuring ‘programmer productivity’ in terms of ‘number of lines of code produced per month.’ This is a very costly measuring unit because it encourages the writing of insipid code, but today I am less interested in how foolish a unit it is from even a pure business point of view. My point today is that, if we wish to count lines of code, we should not regard them as ‘lines produced’ but as ‘lines spent’: the current conventional wisdom is so foolish as to book that count on the wrong side of the ledger.” Edsger Dijkstra, on the cruelty of really teaching computing science, EWD Manuscript 1036. ↩

5. It has routinely been our experience at Google that when the quantitative and qualitative metrics disagree, it was because the quantitative metrics were not capturing the expected result. ↩

6. Recall bias is the bias from memory. People are more likely to recall events that are particularly interesting or frustrating. ↩

7. Recency bias is another form of bias from memory in which people are biased toward their most recent experience. In this case, as they just successfully completed the process, they might be feeling particularly good about it. ↩

8. Because we asked only those people who completed the process, we aren't capturing the opinions of those who did not complete the process. ↩

9. There is a temptation to use such metrics to evaluate individual engineers, or perhaps even to identify high and low performers. Doing so would be counterproductive, though. If productivity metrics are used for performance reviews, engineers will be quick to game the metrics, and they will no longer be useful for measuring and improving productivity across the organization. The only way to make these measurements work is to let go of the idea of measuring individuals and embrace measuring the aggregate effect. ↩