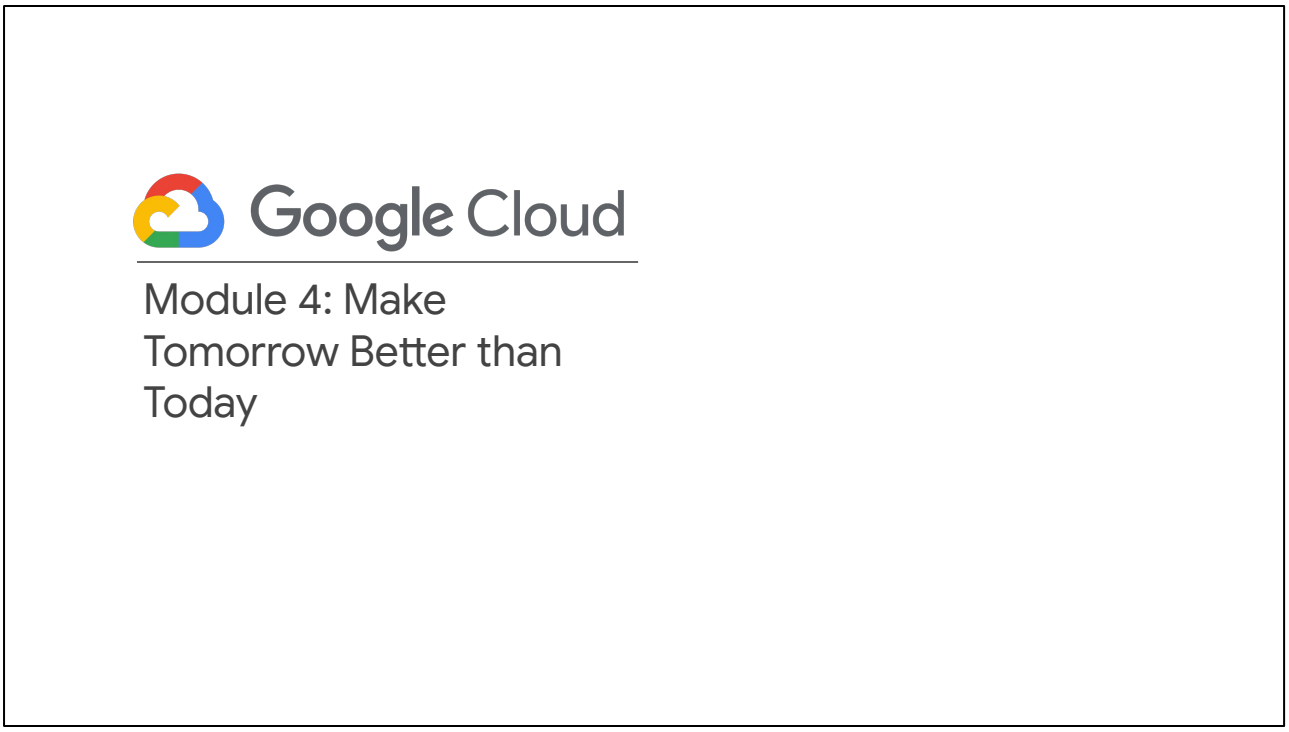


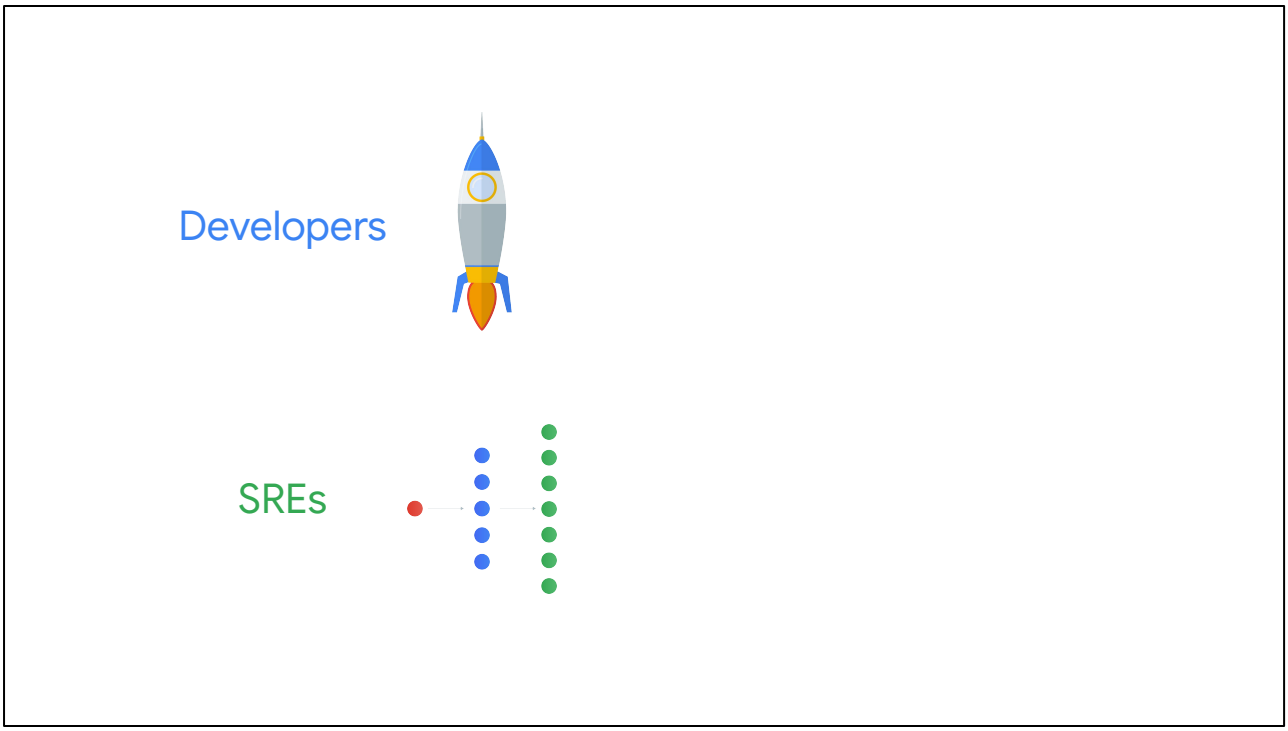
In the previous module, we discussed the first step on the journey to SRE: SLOs with Consequences. The next step on the journey is to Make Tomorrow Better than Today.

Learning topics

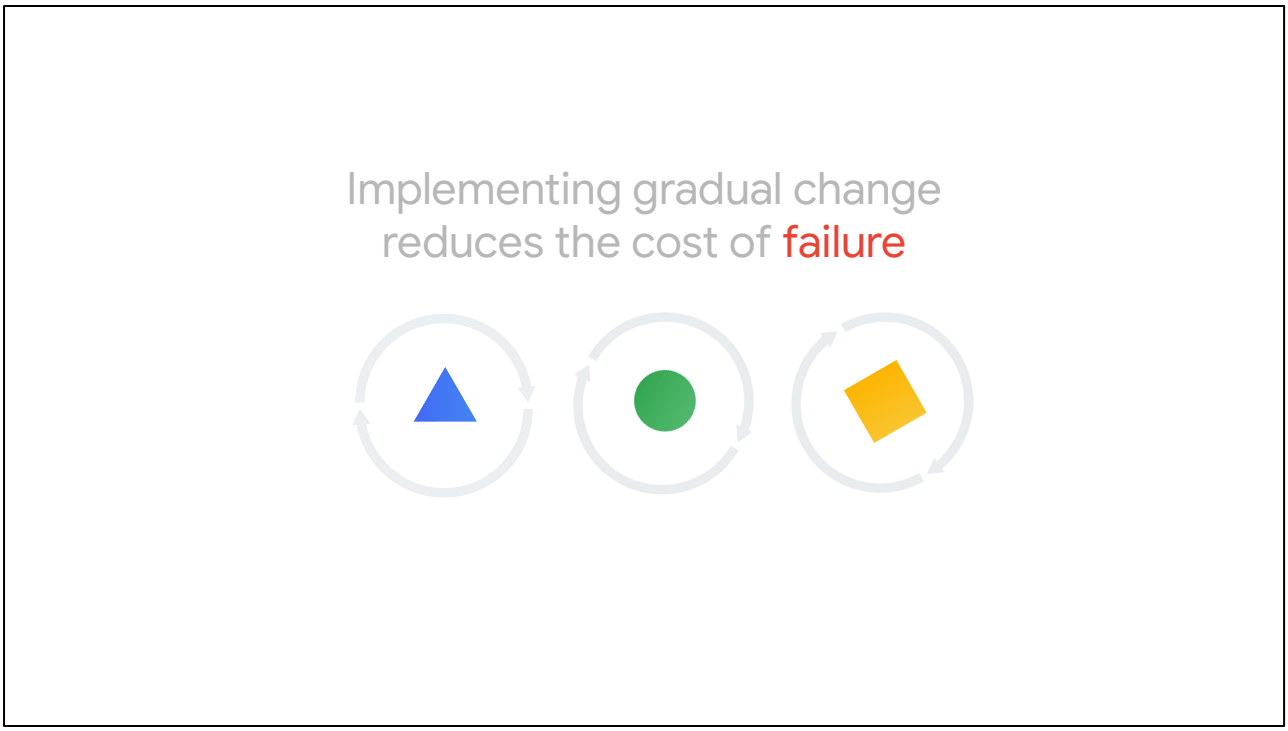
- Continuous integration/Continuous delivery (CI/CD)
- Canarying
- Toil
- Automation

In this module, you'll learn about the SRE concepts of *continuous integration/continuous delivery* (CI/CD) and *canarying* as they relate to the DevOps pillar of implementing gradual change. We'll also look at the concepts of *toil* and *automation* and the idea of automating this year's job away.

Along with these technical concepts, you'll also learn about cultural concepts of design thinking, prototyping, and how you can support your teams through change.



In the world of IT, there are different perspectives on how to approach software development. Culturally, developers tend to focus on “moonshot” thinking, where they can implement massive software changes that create breakthroughs and could fundamentally change society, but have a high likelihood of failing. In contrast are SREs focus on **gradual change**, where they can test smaller changes that will have less impact on users if they fail.

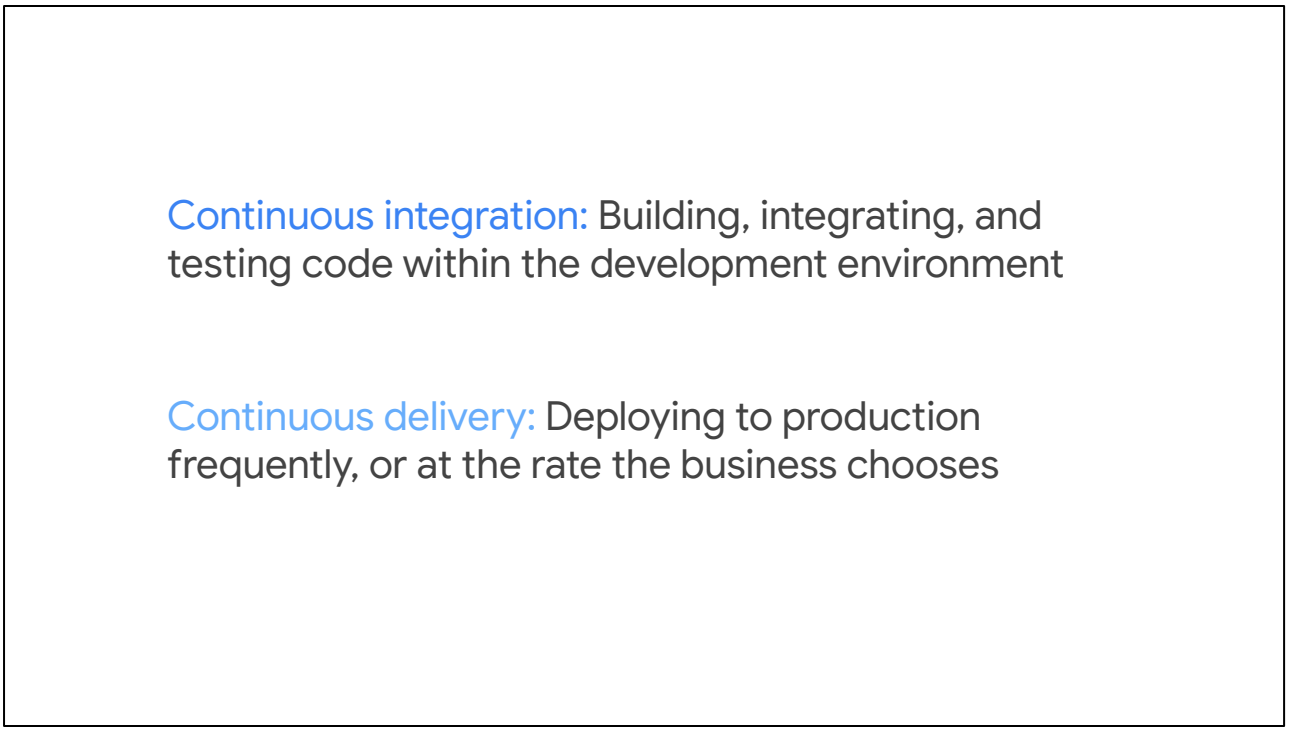


The DevOps pillar of implementing gradual change is practically implemented by SREs in a few different ways to **reduce the cost of failure**. SREs believe that change is best when it is small and frequent. And although change is risky, it's less disruptive to users when rolled out in smaller waves.

Continuous integration/Continuous delivery (CI/CD)

Canarying

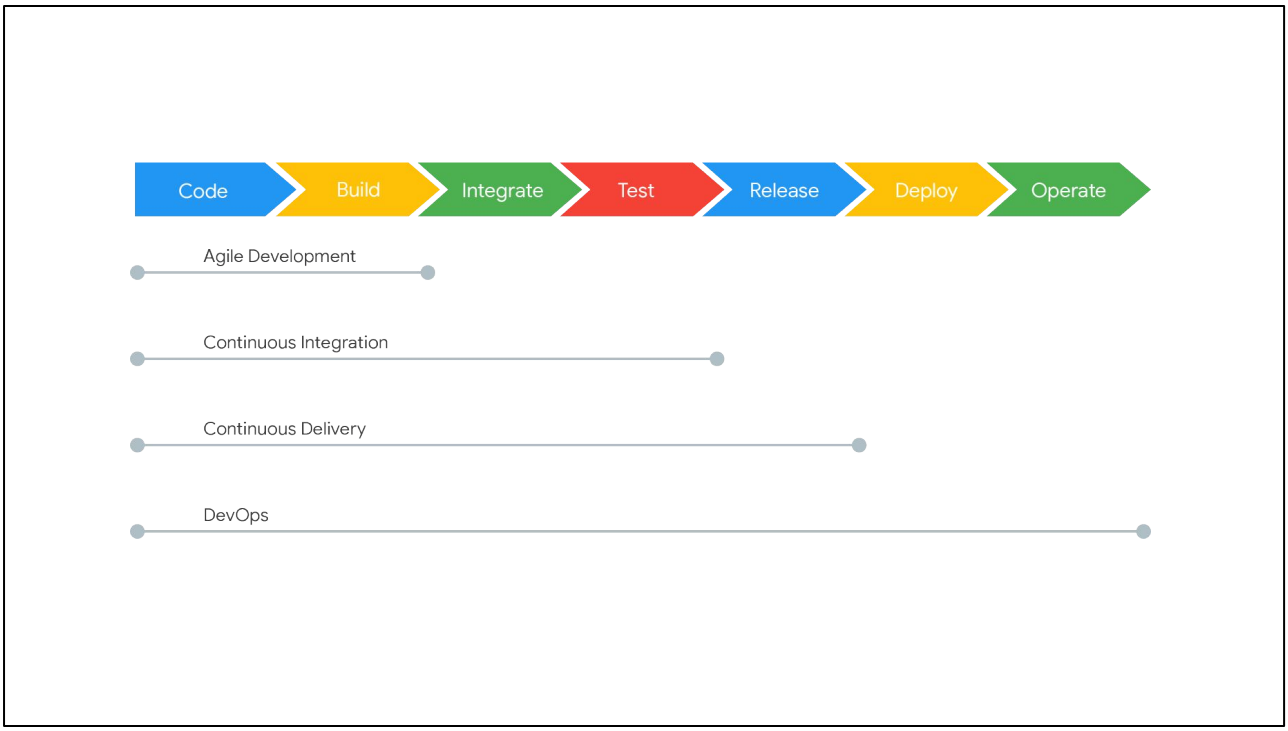
Google SRE culture focuses on practices of continuous integration/continuous delivery, or CI/CD, and canarying.



Let's first define CI/CD.

Continuous integration (CI) usually refers to, building, integrating, and testing code within the development environment. The main goal of this practice is to enable engineers to work on code and test more often. As a result, code quality increases, and critical issues can be avoided earlier.

Continuous delivery (CD) just means that you can deploy to production frequently but may choose not to, usually due to businesses preferring a slower rate of deployment. This stage involves continuous integration, testing automation, and deployment automation.



If you think of software development as a process, you can divide it into these categories: code, build, integrate, test, release, deploy, and operate. In agile development, the process covers code and build. DevOps philosophy spans from code to operate. Continuous integration and continuous delivery fill in the middle with code to test, to release, and to deploy.

CI/CD

- Helps to overcome agile transformation challenges.
- Can minimize code integration headaches.
- Reduces human error.
- Promotes higher code quality.

So how does the practice of CI/CD help reduce the cost failure when implementing gradual change?

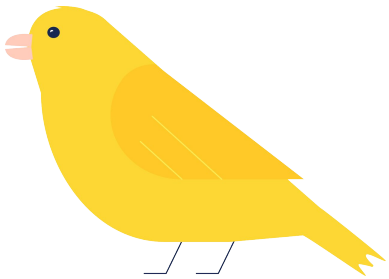
- It helps to overcome agile transformation challenges.
- It can minimize code integration headaches.
- It reduces human error.
- It promotes higher code quality.

CI/CD

- Easier to “recover” after something goes wrong.
- Can automate everything, which saves time and money.
- Provides visibility on project completion.
- Time to market is shorter.
- Provides you with more metrics to review and act on.

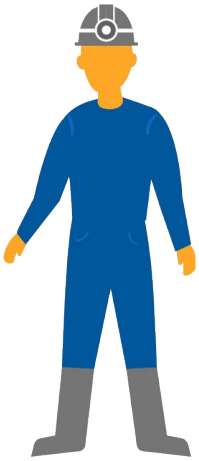
- It’s easier to “recover” after something goes wrong.
- You can automate everything, which saves time and money.
- It provides visibility on project completion.
- The time to market is shorter.
- It provides you with more metrics to review and act on.

Canarying



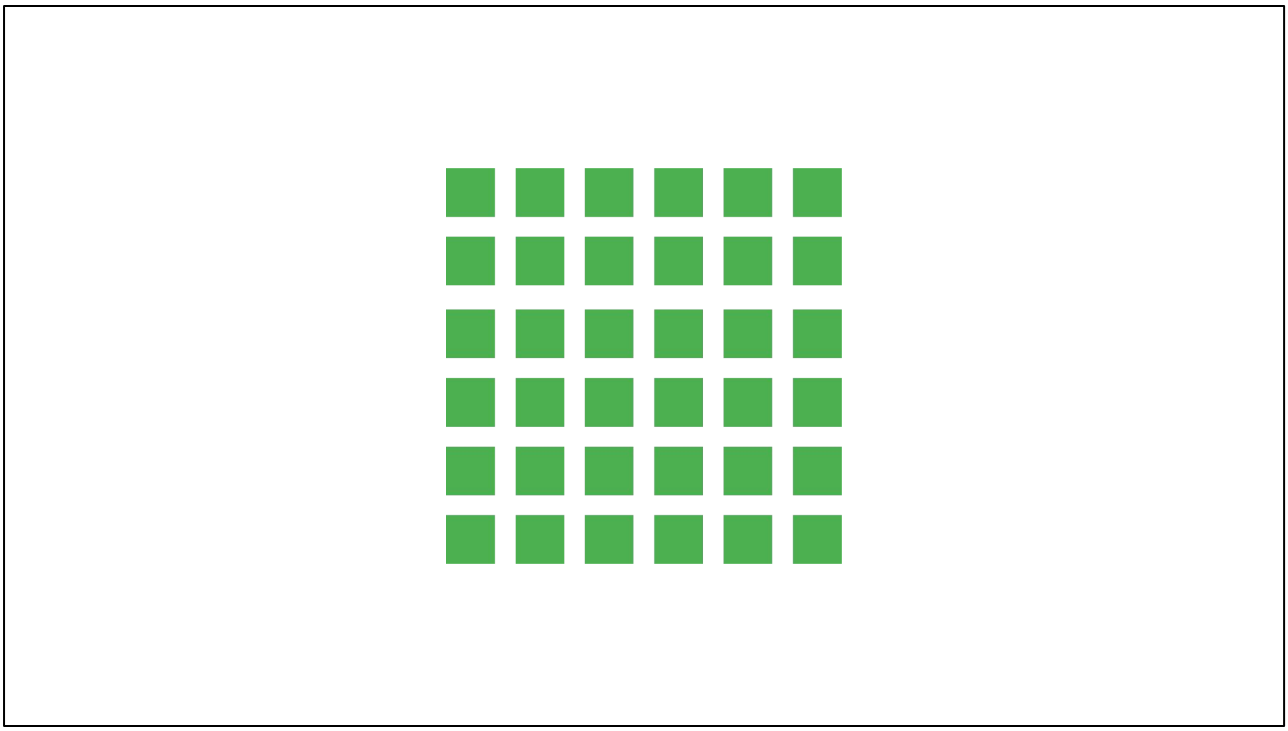
The other practice SREs use for implementing gradual change is **canarying**.

You may have heard of the phrase “canary in a coal mine,” which is a metaphor for advanced warning of danger. Coal miners would bring canaries into coal mines to detect dangerous gases. Canaries are smaller and breathe faster than humans, so if they died, the miners knew that there was danger.



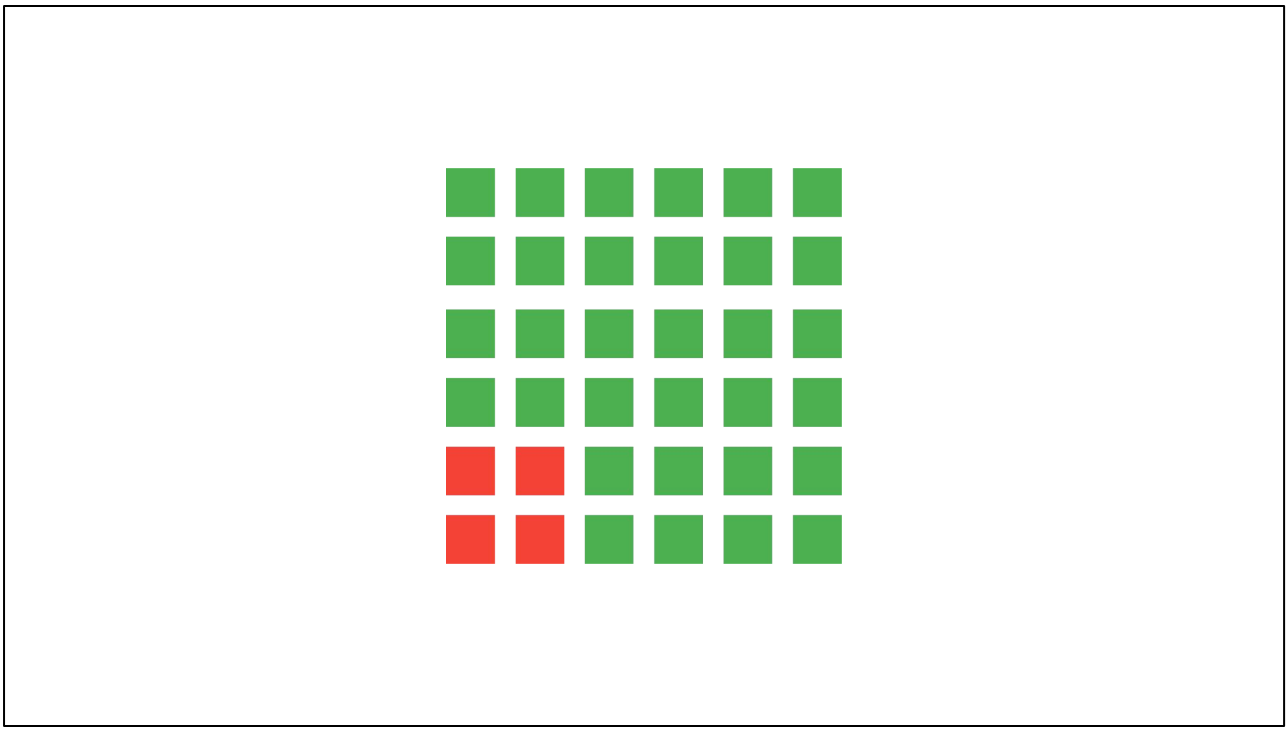
Let’s simplify this metaphor a bit.

- We have something large that we don’t want to harm.
- We have something small that we are okay losing.
- The small thing detects danger as we go into the unknown.

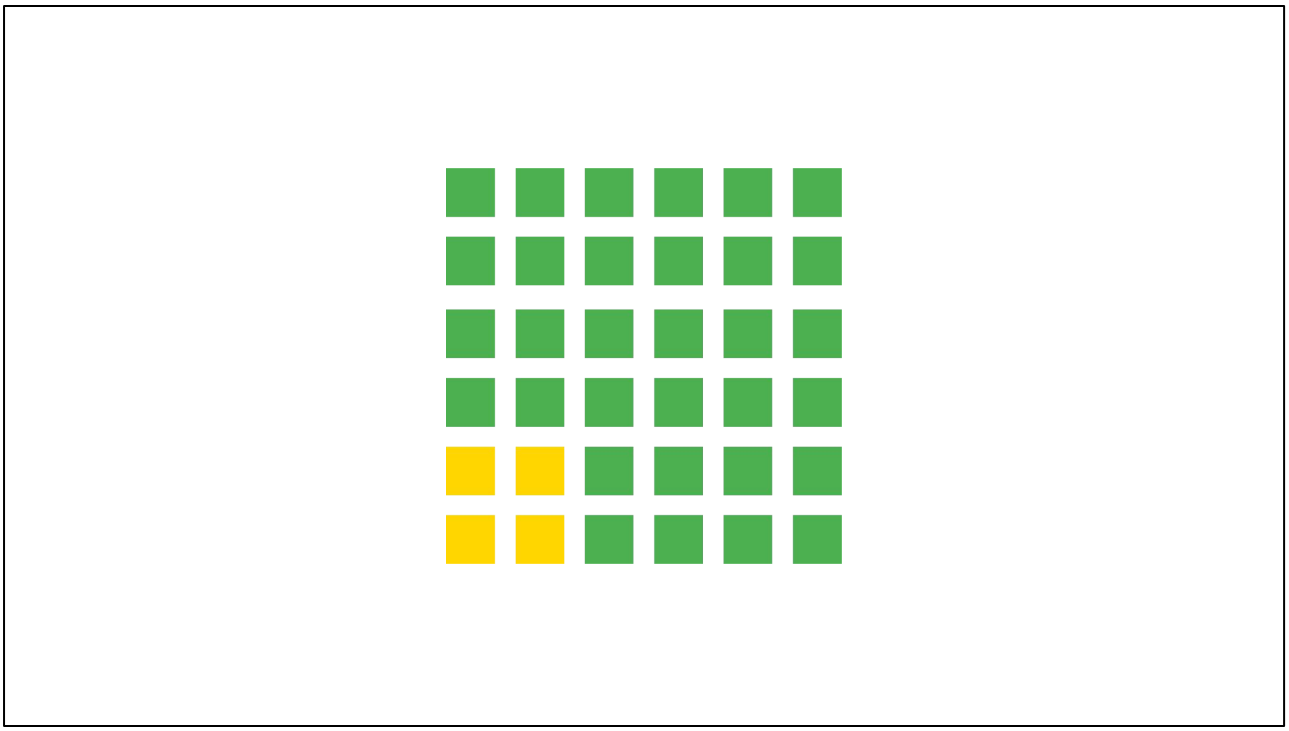


Now let’s see how this relates in terms of SRE practice in production systems.

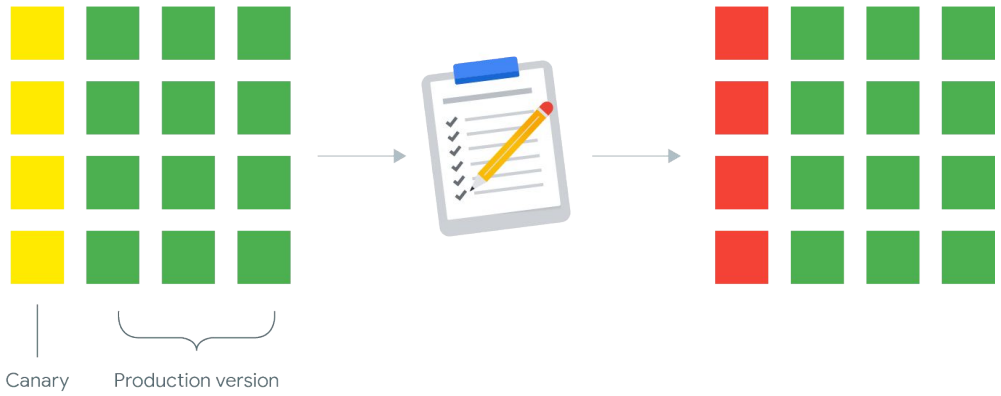
- We have a large service that we want to sustain.



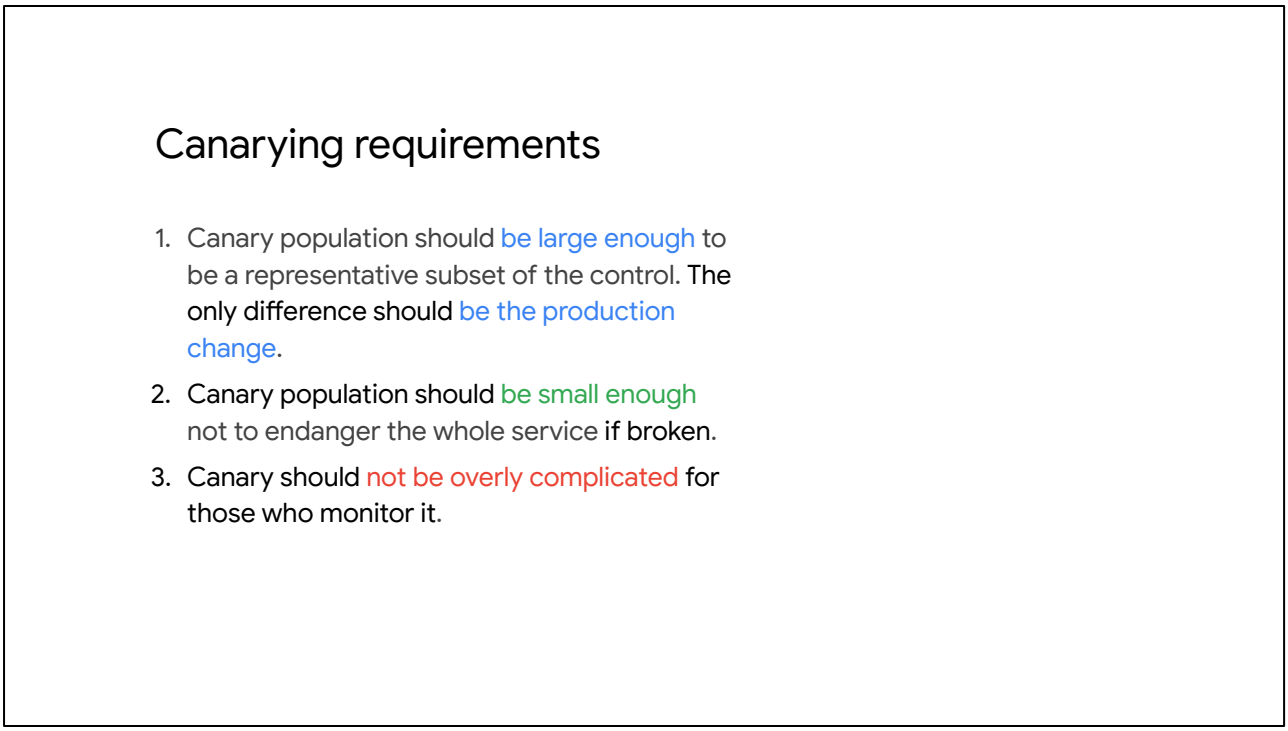
- We are okay losing a small portion of it.



- We employ a production change with unknown impact to the small portion to detect danger.



So what exactly does this mean? Canarying is deploying a change in service to a group of users who don’t know they are receiving the change, evaluating the impact to that group, then deciding how to proceed. If the change contains bugs, the cost is much less than if it was rolled out to the whole system and can be reversed quickly.



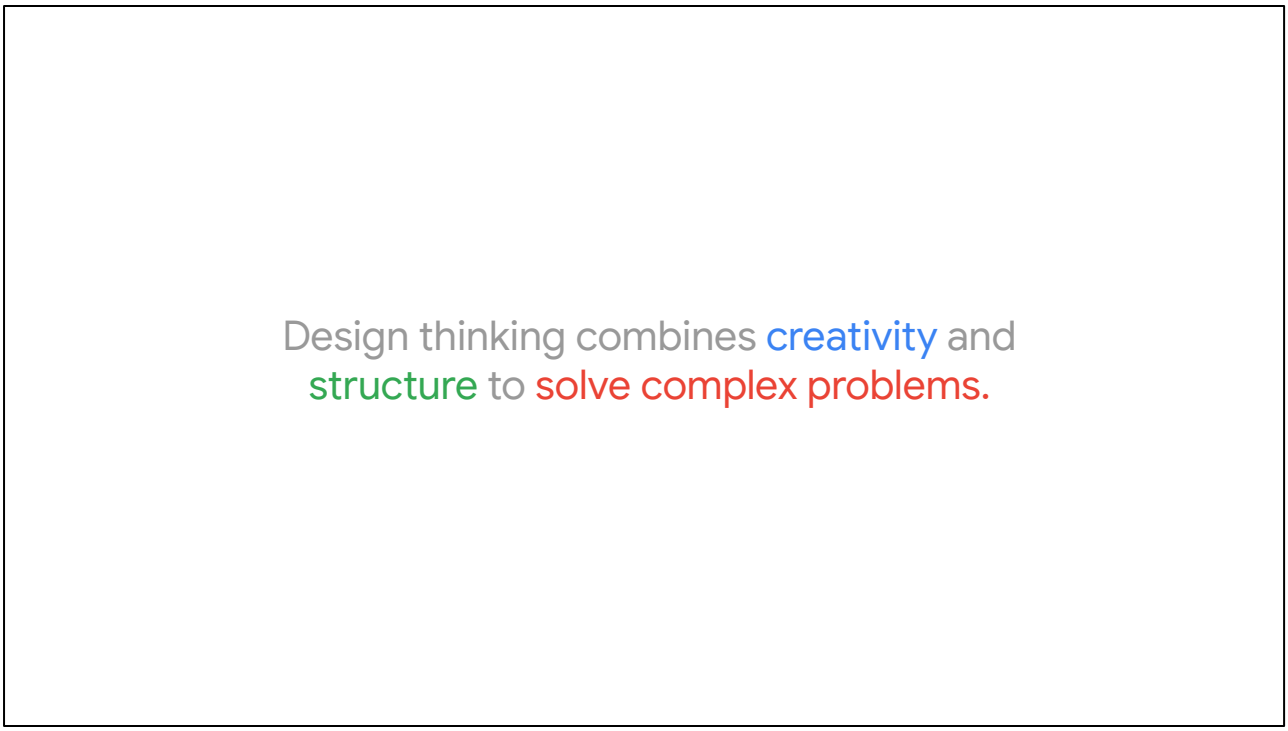
So **what** are requirements for canarying?

- The canary population should be large enough to be a representative subset when compared to the control population. The difference between a canary and the control population should, to the greatest extent practically possible, be **only the production change** that you are testing.
- The canary population should be **small enough** to not endanger the quality of service as a whole if the canary is broken.
- The canary deployment should **not be overly complicated** and impose significant cognitive load on the operator. In other words, it should be easy to reason about the canary process so that it's easy to understand how it can impact current service health overall and easy to cancel in case of problems.

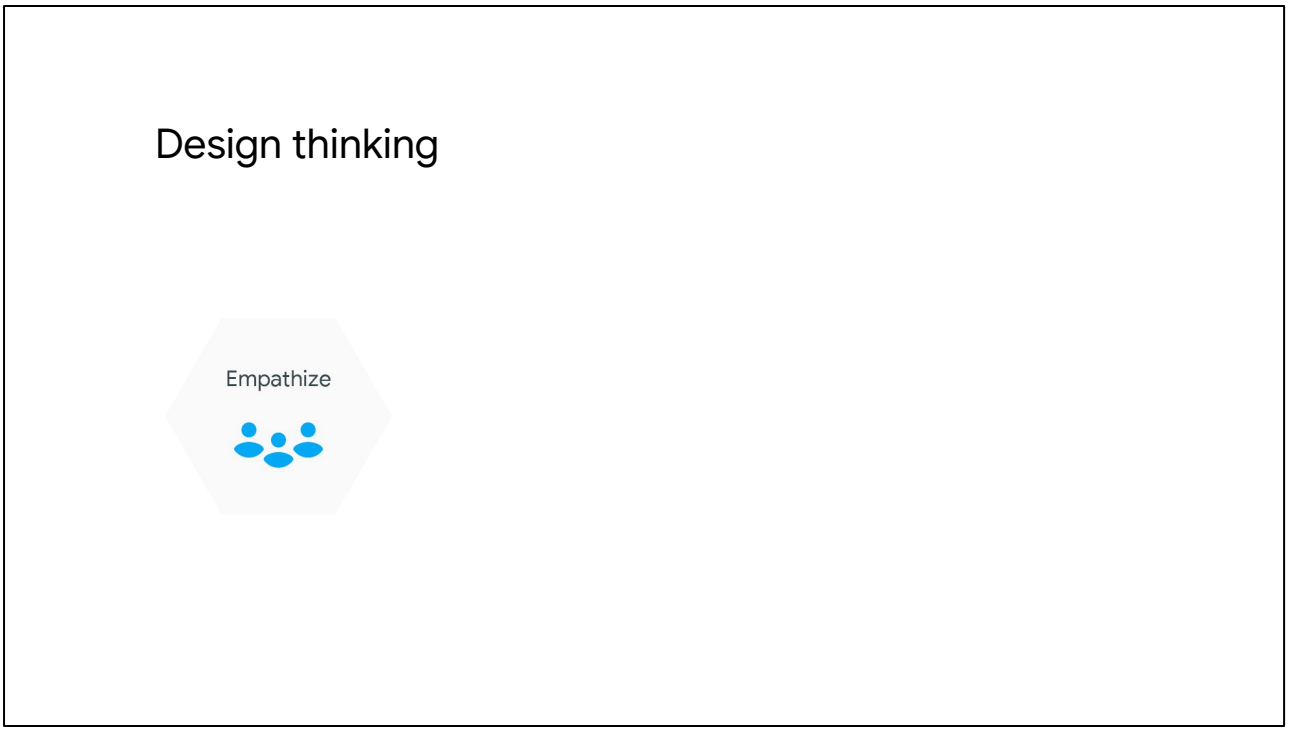
It's true that these three points are, to some degree, in conflict with each other. These generic requirements also do not include any additional requirements specific to a service.

In the next video, we'll talk about the SRE cultural concepts of *design thinking* and *prototyping* that relate to implementing gradual change.

When you reduce the cost of failure by implementing gradual change practices such as CI/CD and canarying, you open up your teams to be more innovative. Knowing that any change will be tested allows individuals to think big and not restrict their creativity or ideas. This is why **design thinking and prototyping** are key aspects of SRE organizational culture.



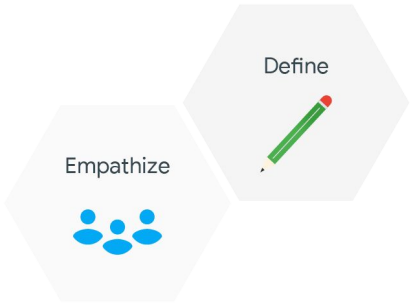
Design thinking is one approach that combines creativity and structure to solve complex problems. Google uses design thinking as one method to teach teams and individuals to think creatively, which is an important step in the process of innovation.



Design thinking methodology has five phases.

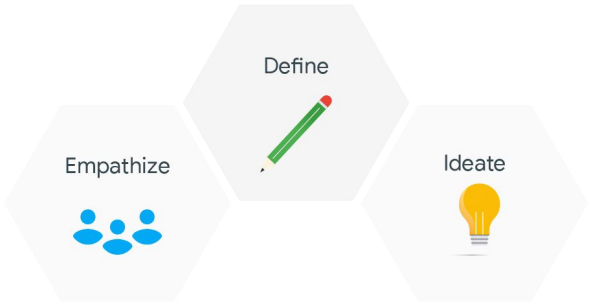
First, **empathize.** In this phase you want to observe and engage with your intended users to learn more about them and immerse yourself in their environments. Empathy helps you set aside your own assumptions in order to gain insight into your users and their needs.

Design thinking



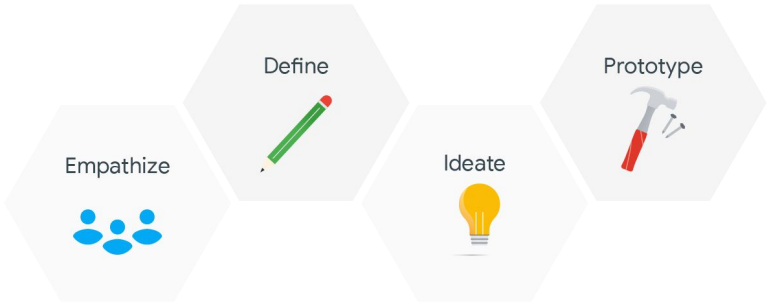
Second, **define** the problem you are attempting to solve. Express the problem in the form of a point of view of the user, versus what you want to accomplish.

Design thinking



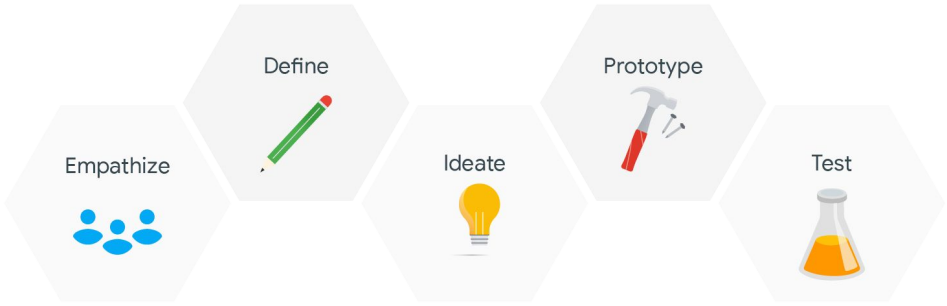
Third, **ideate**. Now that you've defined the problem, you can start generating ideas for solutions. This is a time to "think outside the box."

Design thinking

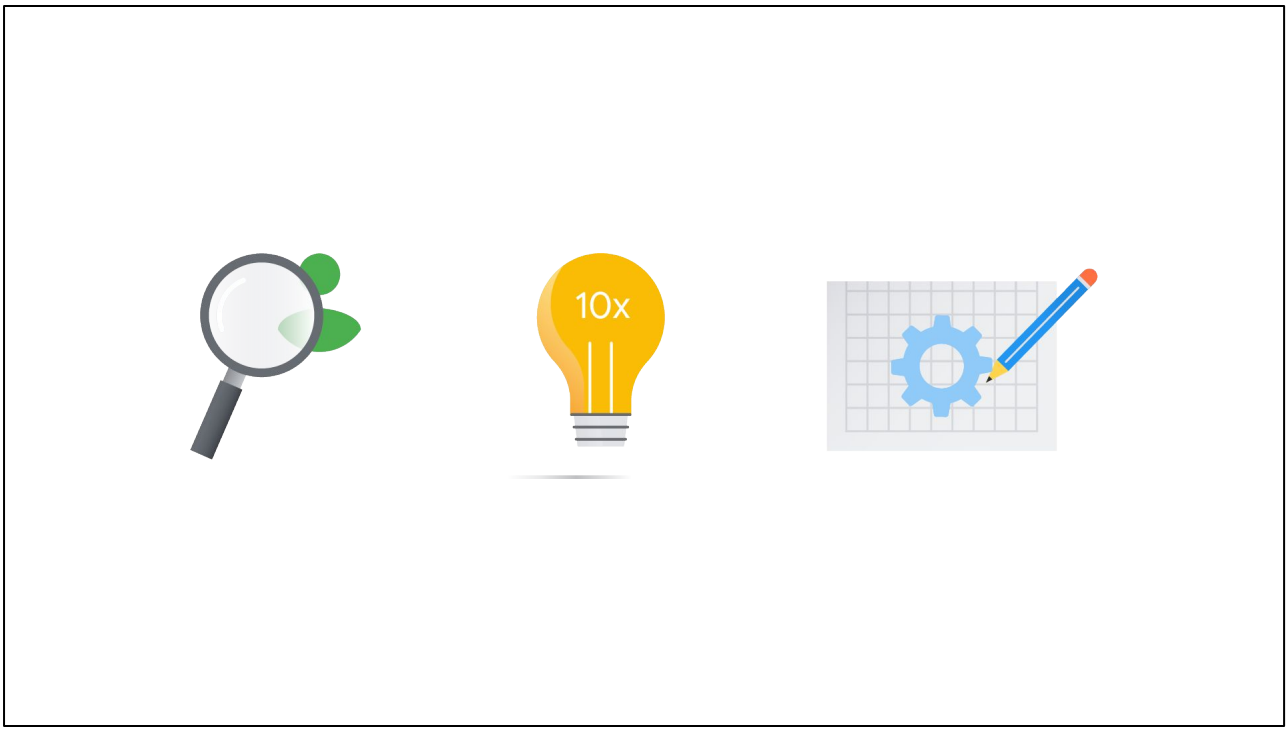


Fourth, it's time to **prototype.** In this phase, you can get the ideas out of your head and into the real world. It's meant to be experimental, so you can identify the best possible solution before committing.

Design thinking

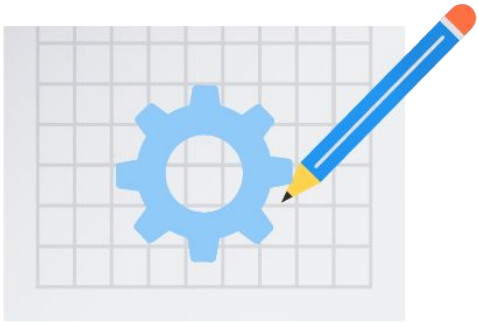


And finally, **test.** You'll want to test your prototype solutions in a real-world setting with your intended users.



If you analyze this even more simply with a software development mindset, you want to first focus on the user, then do some 10x thinking, and then prototype to test your solution. This approach encourages your teams to think about what they are solving for from the user's perspective. Then they can brainstorm expansively on the solution, and then prototype their solution. Finally, they can then test gradually using SRE practices such as CI/CD and canarying, which you learned about in the last video.

It's very important for organizations with SRE teams to promote being **prototype-driven**.



Without prototyping:

- Fewer ideas are tested
- Slower failures
- Fewer successes

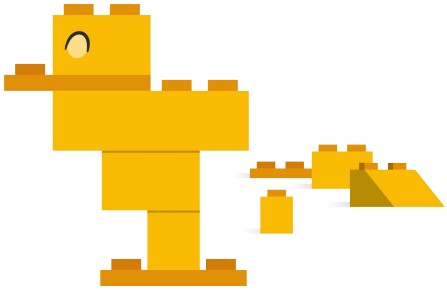
With prototyping:

- More ideas are tested
- Faster failures
- More successes

Without prototyping, fewer ideas are tested. This creates slower failures versus fast ones, which can lead to fewer successes. However with a prototyping culture, teams are encouraged to try more ideas. This leads to an increase in faster failures versus slow ones, and ultimately can lead to more successes than without prototyping.

Ways to prototype

1. **Physical prototyping**



Remember that there is no right way to prototype. Here are some examples of how teams can prototype fast to test their solutions.

1. **Physical prototyping:** Build a model with legos or other small building blocks.

Ways to prototype

1. Physical prototyping
2. **Paper and drawing**

2. **Paper and drawing**: Use pencil and paper to wireframe out ideas.

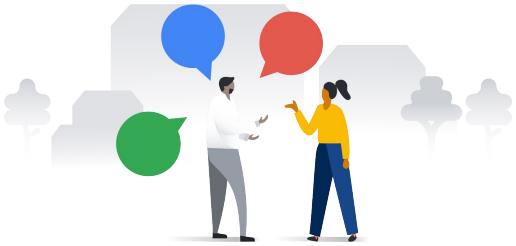
Ways to prototype

1. Physical prototyping
2. Paper and drawing
3. **Clickable**

3. **Clickable**: Create a clickable solution using software that simulates a solution.

Ways to prototype

1. Physical prototyping
2. Paper and drawing
3. Clickable
4. **Role play**



4. **Role play**: Find people to role-play testers and try out the prototype.

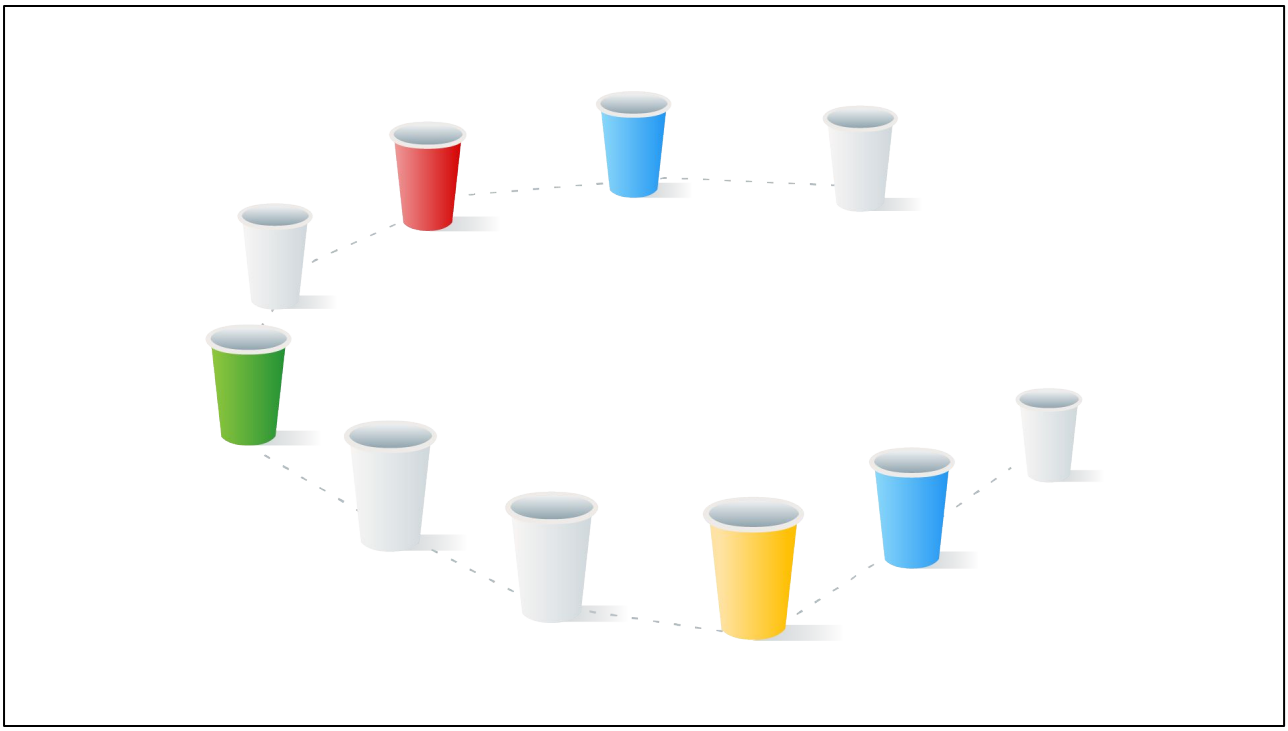
Ways to prototype

1. Physical prototyping
2. Paper and drawing
3. Clickable
4. Role play
5. **Video**

5. **Video**: Use video to record a solution or see how users interact with it.

No matter how you prototype, try to make it real enough for users to feel. More concrete prototypes increase the likelihood of actionable feedback.

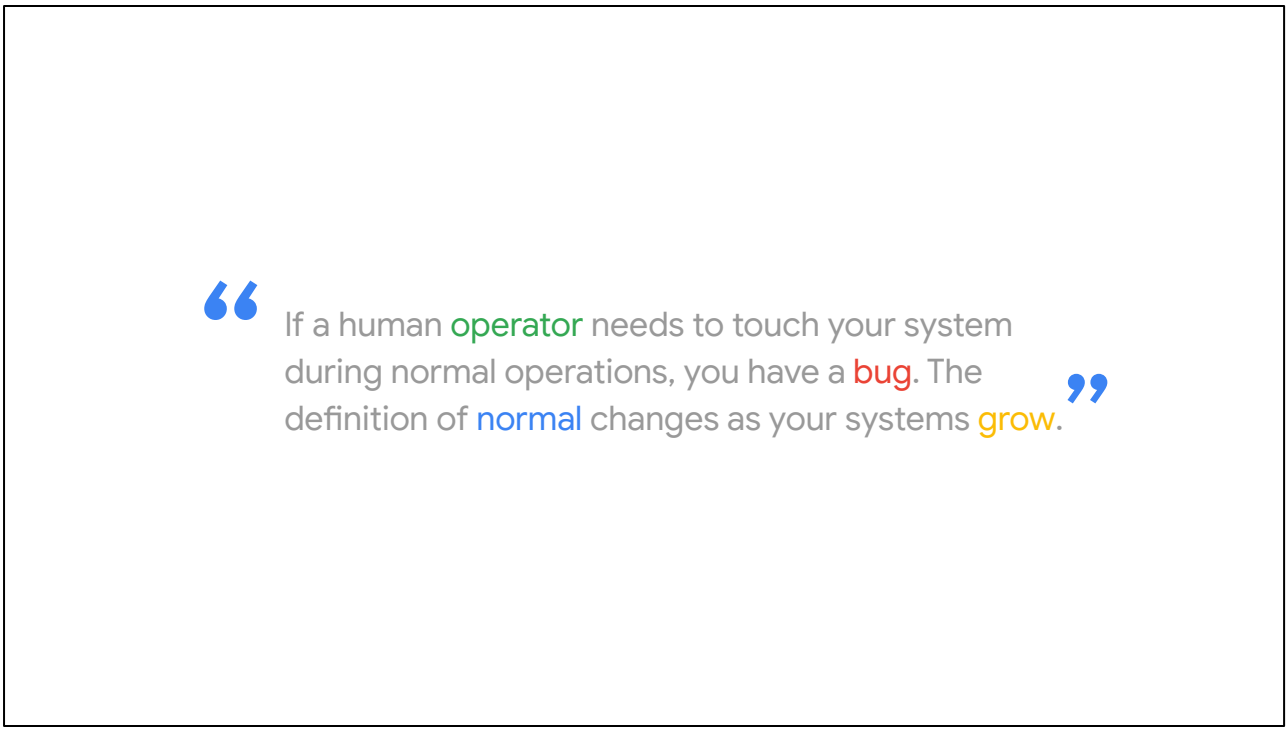
We at Google have learned, based on our customer interactions, that with the help of simple prototypes, customers are able to improve the most complex processes. By activating imaginative thinking, individuals feel motivated and encouraged to have audacious ideas that they might not have by sitting at their desks or during a regular meeting. Some of the examples of prototypes we've seen from customers include a video of a panel discussion, a heatmap, and a banner with post-it notes.



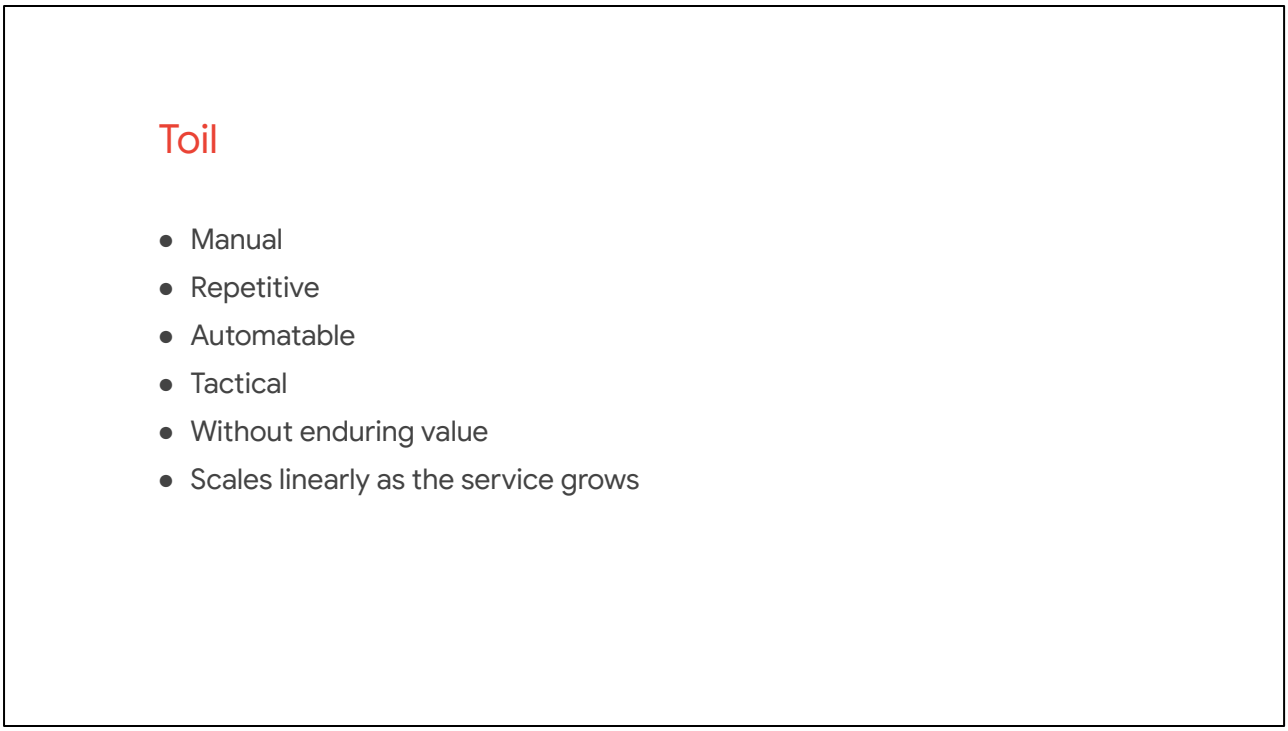
One of the leading online retailers in the Netherlands used the design thinking methodology to brainstorm about changes to their production process. The participants arranged paper cups to represent each step in the process. While prototyping, the team used different-colored cups to mark the steps needed to be improved or deprecated.

Hopefully now you can understand how your organization can benefit from a culture of design thinking and prototyping. Your teams will need your support to help promote and encourage that culture.

In the next video you will learn about ways SREs leverage automation in order to put more time and focus into software development versus operations, which is vital for any business's success.



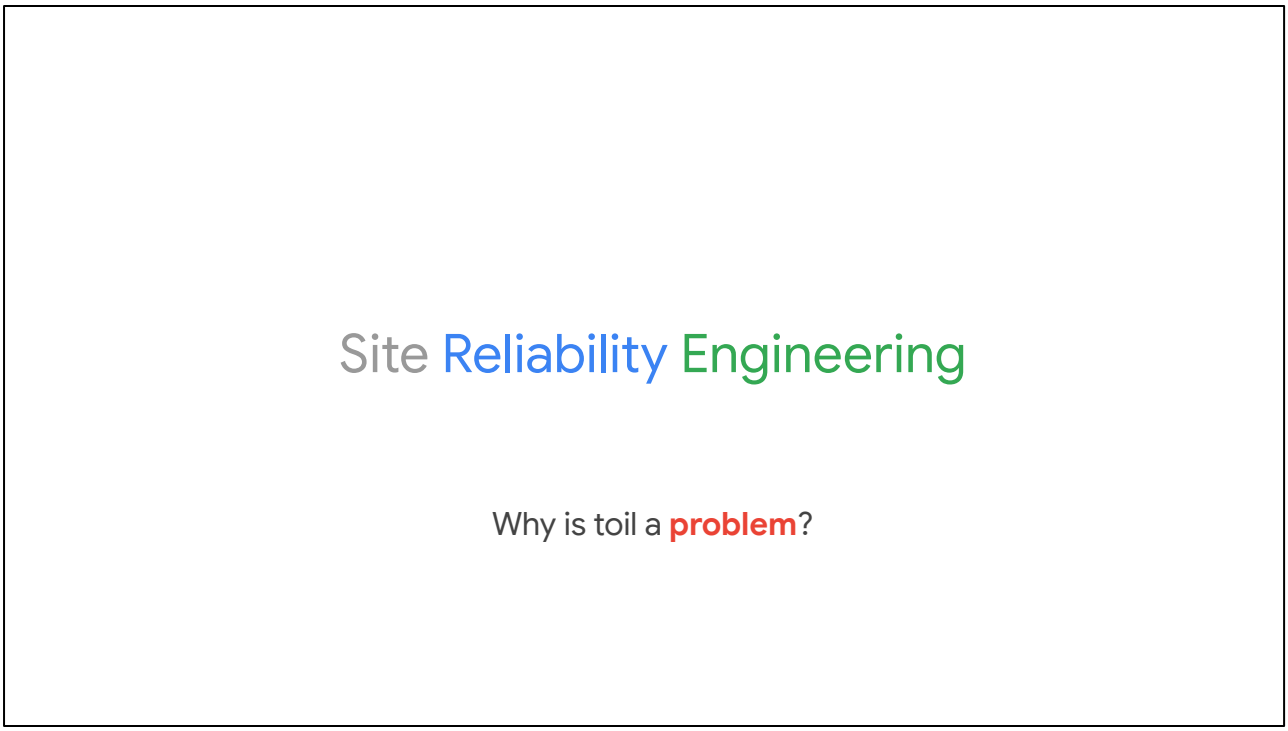
A Google SRE once said, “If a human operator needs to touch your system during normal operations, you have a bug. The definition of normal changes as your systems grow.” As discussed earlier in the course, a key pillar of DevOps philosophy for Google is leveraging tooling and automation. Focusing on this allows your engineering teams to focus on development work instead of operational work. SREs do this by eliminating that operational work, which we call *toil*.



So what exactly do we mean by toil? Toil is work directly tied to a service that is manual, repetitive, automatable, tactical, or without enduring value, or that scales linearly as the service grows.

Toil isn't just administrative work or work you don't want to do, because that kind of work can still be very important. Different people like different types of work, and administrative work can be necessary overhead, such as team meetings or HR paperwork. This type of work also isn't tied to running a production service.

By eliminating toil, SREs can focus the majority of their time on work that will either reduce future toil or add service features, which generally focuses on improving reliability, performance, or utilization.



Until now, you've learned a lot about the "reliability" part of site reliability engineering. Reducing toil and scaling up services is now the "engineering" part of site reliability engineering. Engineering work is what enables an SRE team to scale up and to manage services more efficiently than either a pure Dev team or a pure Ops team. By keeping your SREs working on toil less than 50% of the time, you're also distinguishing the SRE role as clearly different from a typical operations role.

So why is toil really a problem? Well, toil can create multiple issues in your organization.

Excessive toil

1. **Career stagnation**

Toil can lead to **career stagnation**. Individual team members' career progress will slow down or stop if they spend too little time on projects. While it's true that Google rewards undesirable work when it's inevitable and has a large positive impact, you can't make a career out of it.

Excessive toil

1. Career stagnation
2. **Low morale**

It promotes **low morale**. People have different levels of tolerance for how much toil they can do, but everyone has a limit. Too much toil leads to burnout, boredom, and discontent.

Excessive toil

1. Career stagnation
2. Low morale
3. **Confusion**

It **creates confusion**. At Google, we work hard to ensure that everyone who works in or with the SRE organization understands that we are an *engineering* organization. Individuals or teams within SRE that engage in too much toil undermine the clarity of that communication and confuse people about the SRE role.

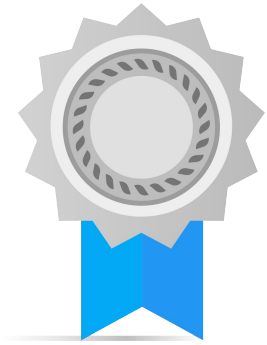
Excessive toil

1. Career stagnation
2. Low morale
3. Confusion
4. **Slower progress**

Toil **slows progress.** Excessive toil makes a team less productive. A product's feature velocity will slow if the SRE team is too busy with manual and reactionary work to roll out new features promptly.

Excessive toil

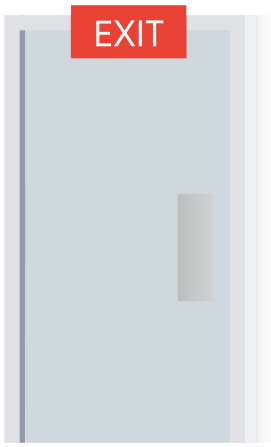
1. Career stagnation
2. Low morale
3. Confusion
4. Slower progress
5. **Precedence**



It **sets precedent**. If you're too willing to take on toil, your developer counterparts will have incentives to load you down with even more toil, sometimes shifting operational tasks that should rightfully be performed by developers to SRE. Other teams may also start expecting SREs to take on such work, further perpetuating the issue.

Excessive toil

1. Career stagnation
2. Low morale
3. Confusion
4. Slower progress
5. Precedence
6. **Attrition**



It **promotes attrition**. Even if you're not personally unhappy with toil, your current or future teammates might like it much less. If you build too much toil into your team's procedures, you motivate the team's best engineers to start looking elsewhere for a more rewarding job.

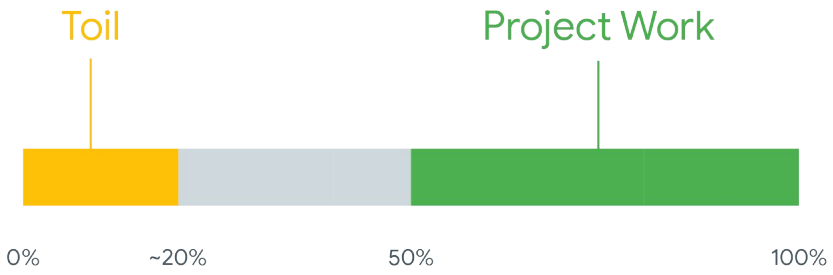
Excessive toil

1. Career stagnation
2. Low morale
3. Confusion
4. Slower progress
5. Precedence
6. Attrition
7. **Breach of faith**

Lastly, toil **causes breach of faith**. New hires or transfers who join SRE with the promise of project work will feel cheated, which is bad for morale.

Even though a lot of toil is unhealthy when running a service, there are some positives for having a little bit of toil. Toil doesn’t make everyone unhappy all the time, especially in small amounts. Predictable and repetitive tasks can be quite calming. They produce a sense of accomplishment and quick wins. They can be low-risk and low-stress activities. Some people gravitate toward tasks involving toil and may even enjoy that type of work. But it should never be primary work for an SRE.

Toil isn’t always and invariably bad, and everyone needs to be absolutely clear that some amount of toil is unavoidable in the SRE role, and in almost any engineering role. Toil becomes toxic when experienced in large quantities. You should be concerned if your teams complain about being burdened with too much toil.



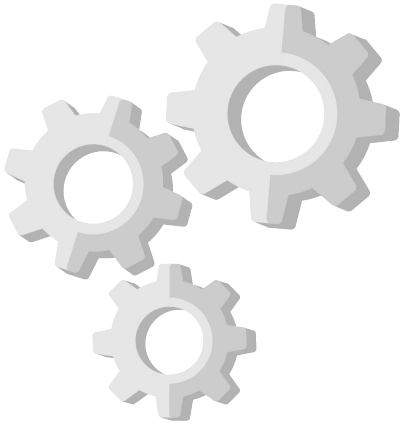
So now you may be wondering how you can balance toil with project work.

Toil must be a bounded part of an SRE role. If SREs don't have time for anything else, they are doing traditional sysadmin tasks that DevOps advocates against.

If you put a threshold for toil at 50% for SREs, they are freed to do project work that supports your engineering and reliability goals the rest of the time. Priority project work for SREs is work that impacts or might impact the team's SLOs. After that, their focus should be work that causes SREs toil.

Automation

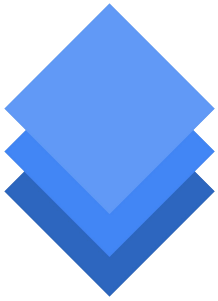
- What to automate
- How to automate it



A key aspect of eliminating toil is automation. SREs strive to automate this year's job away—that is, determining what to automate, under what conditions, and how to automate it.

Value of automation

1. **Consistency**

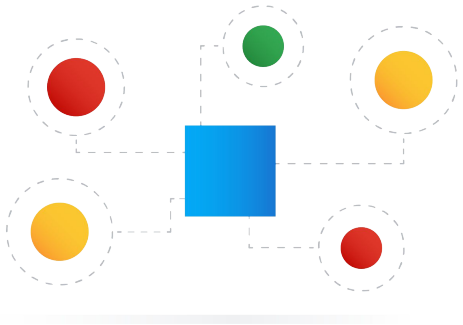


Automation in a production service provides several values.

First, it can provide **consistency**. Any action performed by a human is prone to error, especially the same action performed hundreds of times. A person isn't likely to be as consistent as a machine. Lack of consistency leads to mistakes, oversights, issues with data quality, and even reliability problems. Automation remedies this by creating consistency.

Value of automation

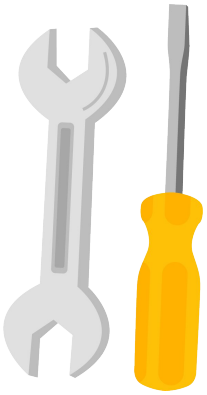
1. Consistency
2. **A platform**



Next, automated systems provide **a platform** that can be extended and applied to more systems. A platform also provides a way to centralize mistakes, so that a bug is fixed once in one place. With humans, you'd have to communicate that fix across multiple people, and there is more room for error and for the bug to be reintroduced. Additionally, a platform can execute additional tasks faster and with more accuracy than humans, and can also export performance metrics more easily than a manual system.

Value of automation

1. Consistency
2. A platform
3. **Quicker resolutions**



If automation runs regularly and successfully enough, any common faults can be **resolved more quickly**. You can then spend your time on other tasks instead, which promotes increased developer velocity since you don't have to spend time either preventing a problem or, more commonly, cleaning up after it. A problem discovered later in the product life cycle is more expensive to fix. Generally, problems that occur in actual production are most expensive to fix, both in terms of time and money. This means that an automated system looking for problems as soon as they arise has a good chance of lowering the total cost of the system.

Value of automation

1. Consistency
2. A platform
3. Quicker resolutions
4. **Faster action**

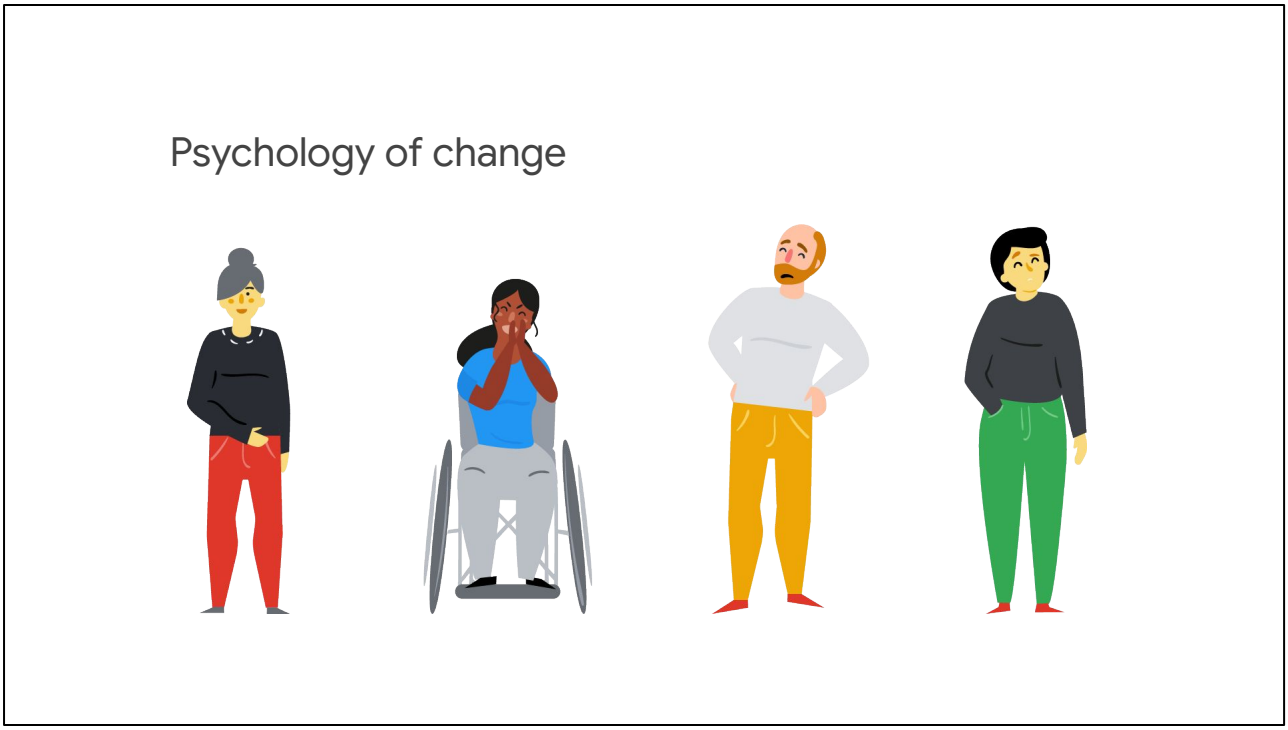
Another value of automation is **faster action**. Machines react faster than humans, so for large production services, automating is necessary for survival since the amount of work required is usually beyond a manageable manual threshold.

1. Consistency
2. A platform
3. Quicker resolutions
4. Faster action
5. **Time saved**

Finally, automation **saves time**. Even though it may be a significant time investment to code a particular automated process, once done there is no need for continual training of humans and maintenance of the process. Once a task is automated, anyone can execute it.

If automation is not common in your organization, you're likely to see some resistance to change from your teams as you start to introduce it or any SRE practices. In the next video, you'll learn about the psychology of and resistance to change, and how you can help support your teams through SRE adoption.

When you start to implement the SRE practice of automation to eliminate toil, some individuals will probably begin to resist. It's important to acknowledge that people react to a push for automation in different ways and that some may resist it more than others. Individuals may feel as though their jobs are in jeopardy, or they may disagree that certain tasks are toil and don't need to be automated. Because of this, it's important to understand how to address resistance to change in your organization.

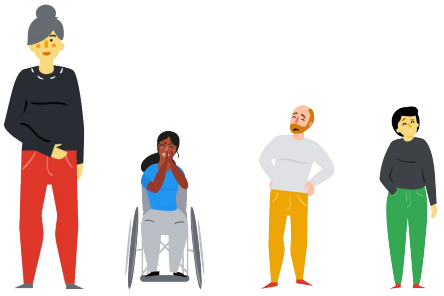


But let’s first start with the **psychology of change**.

Change elicits emotions. There are hundreds of different types of reactions and emotions. You should always expect to get positive and negative reactions, even if the change is for the good. Broadly, people and their emotions fall into four categories. Let’s talk about each group and how you can support them.

Navigators

- Help you succeed.



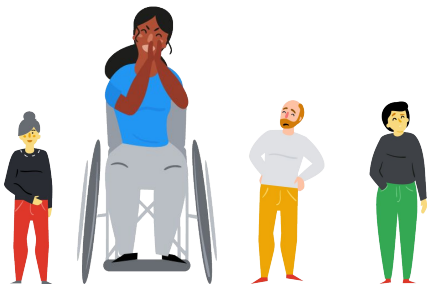
You should:

- Celebrate their behaviors.
- Use them as champions.

Navigators: These are the people who will make teams and businesses successful. As leaders, spot them and celebrate their behaviors. Use them as champions for the change.

Critics

- Have passion and energy.
- Have valid fears.



You should:

- Spend time with them.
- Persuade them.

Critics: These are the second type of individuals that you should care about. They have passion and energy. Critics care, and they have valid fears, so it's important not to ignore them. Spend some time with them, because they will be very powerful advocates if you can persuade them.

Victims

- Need to express emotions.
- Take change personally.

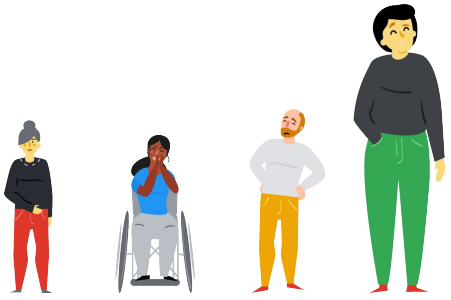
You should:

- Listen to and empathize with them.

Victims: Often, this type of individual just needs to get their emotions out. Victims tend to take organizational change very personally. Your role as a leader is to listen to them and empathize. Once they feel heard, then they can start to listen.

Bystanders

- Are difficult to understand.
- Do not know what's going on.
- Continue with normal routine.



You should:

- Communicate with them.
- Ascertain their feelings.

Bystanders: These people are tricky because you never know what they are thinking. Often, bystanders have no idea what's going on, and they will just continue as if nothing, or no change, is happening. You should try to communicate with them to ascertain their feelings.

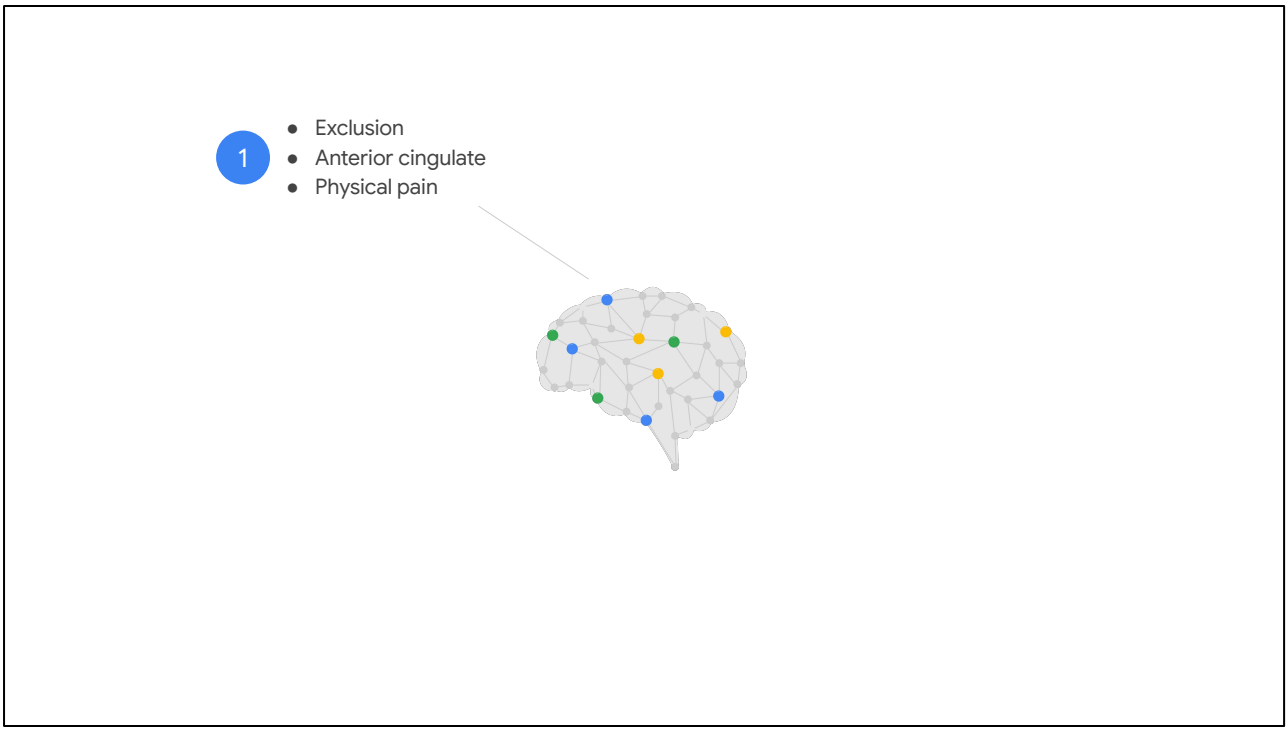
Sometimes one person can fall into several categories. Remember that it's likely you've experienced all of these faces of change at some point in your career.

As a leader, the way you navigate your own emotions to change will highly impact the teams you lead. Teams look to their leaders to get signals on how to react to change.

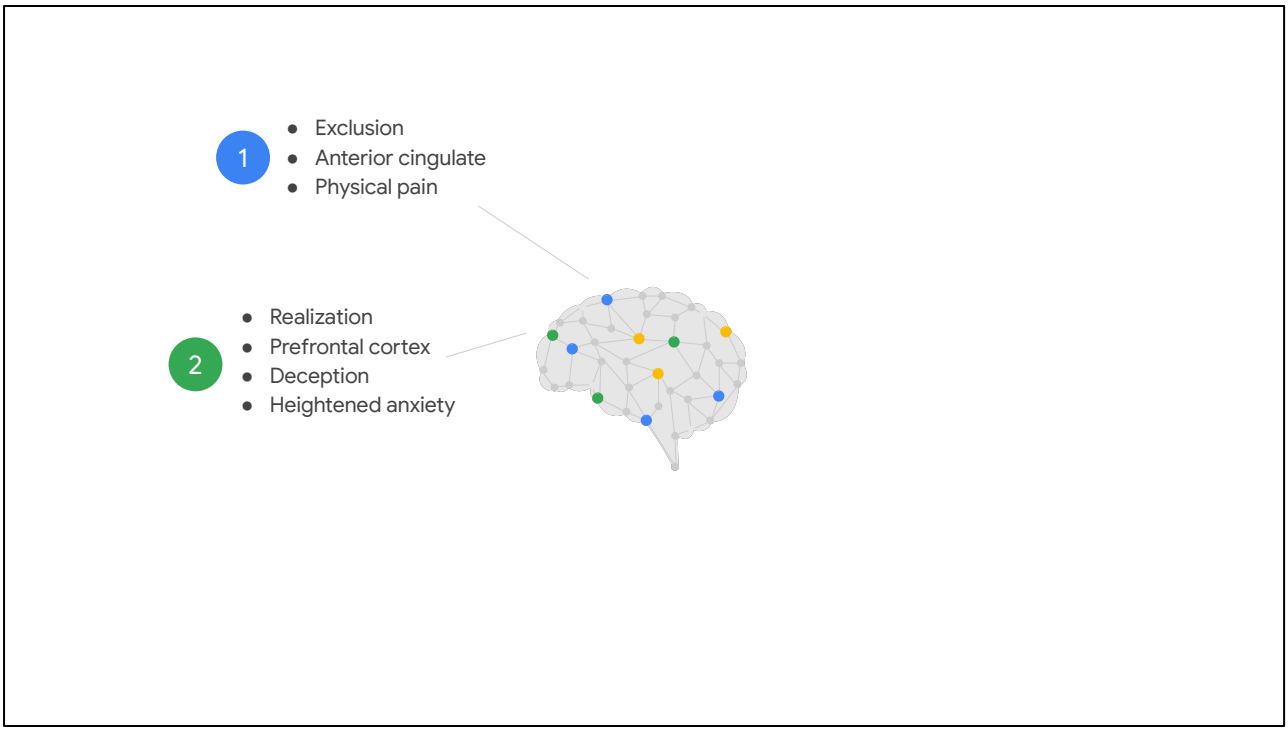
Brains are hard-wired
to reflect emotions.

Brains are hard-wired to reflect emotions. People experience reactions to change not because they are trying to be difficult, but because it’s natural.

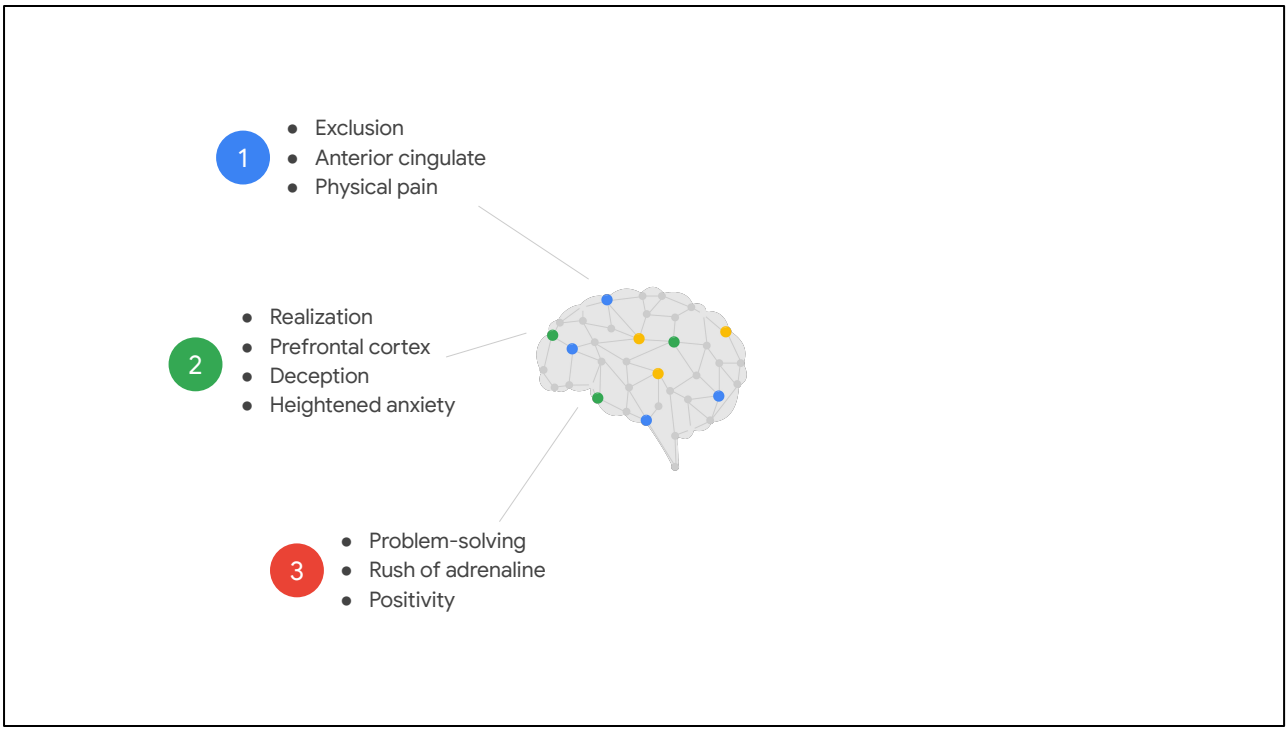
Take a look at how the brain responds to change biologically.



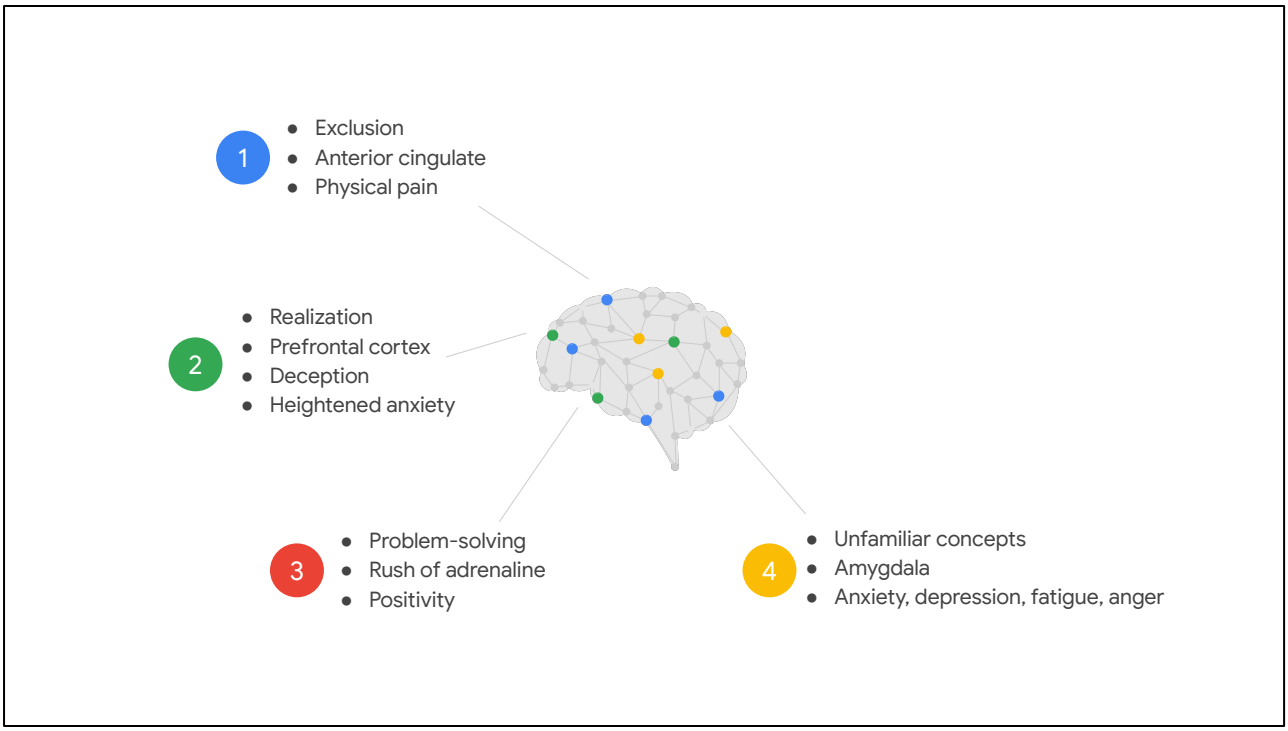
1. When you experience the feeling of being excluded from something, it triggers response in the anterior cingulate (dorsal portion)—which is the same part of the brain that deals with physical pain.



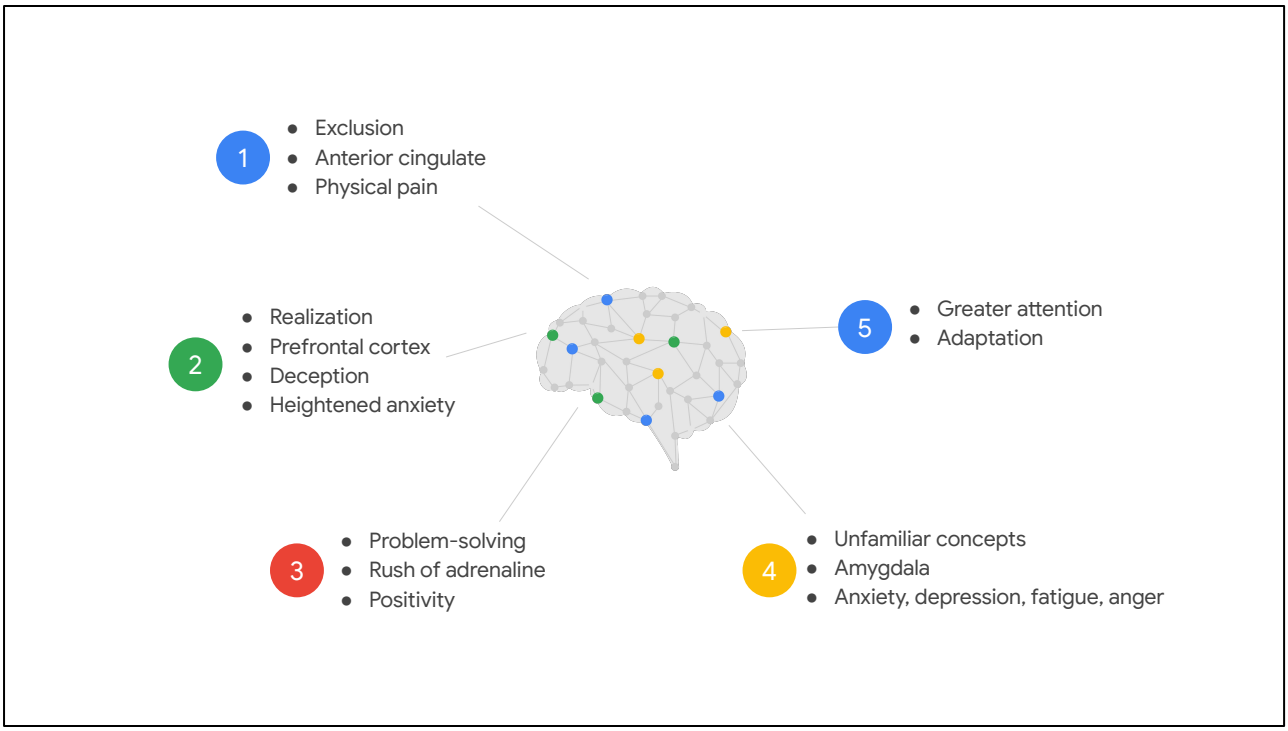
2. When you realize that something you were told in the past is unrealistic or untrue, the prefrontal cortex switches to high alert, looks for other signs of deception, and triggers feelings of heightened anxiety.



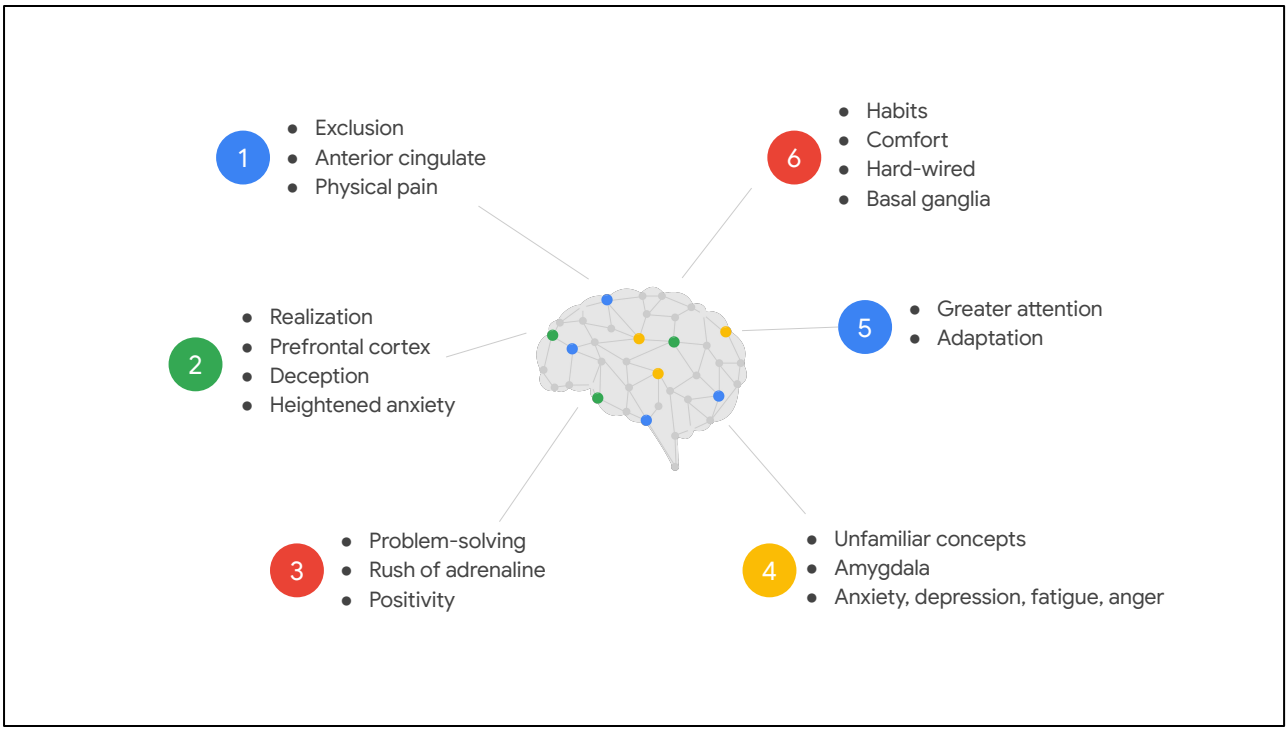
3. When you solve your own problems, you get a rush of adrenaline (positivity/natural high).



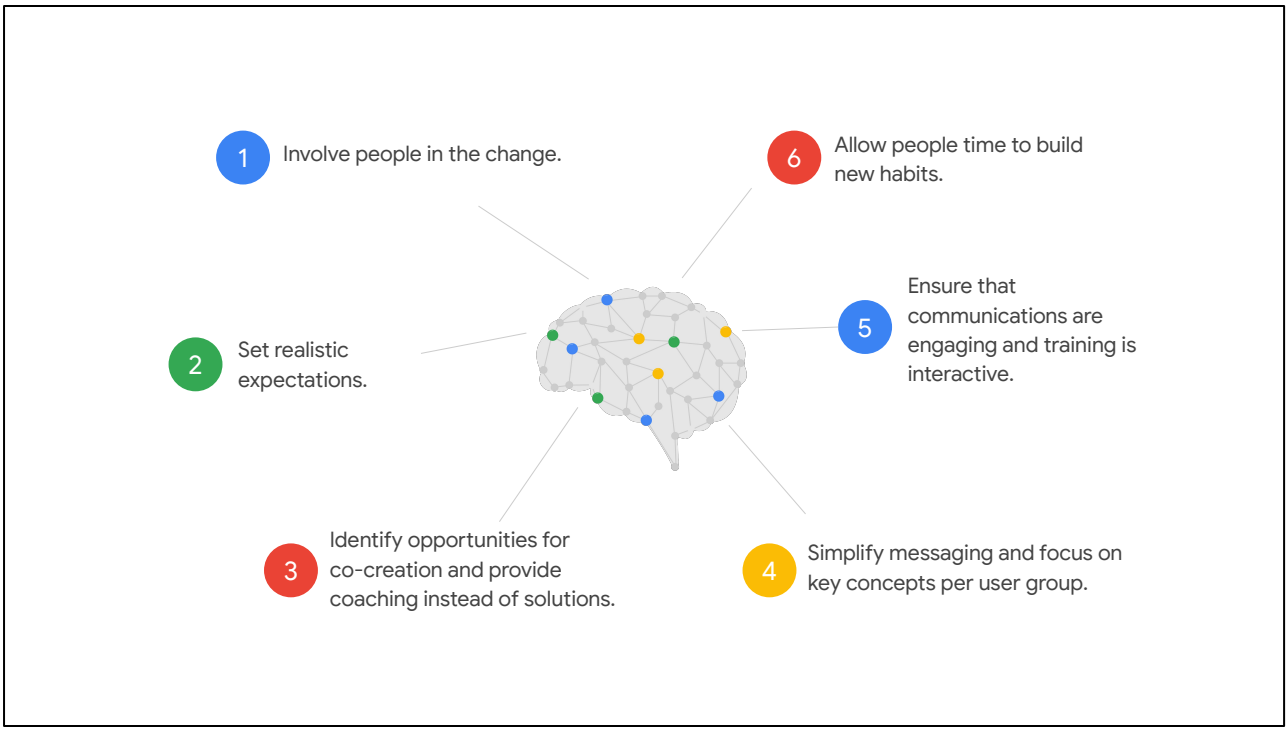
4. The prefrontal cortex can only deal with a few concepts at a time. When you are overwhelmed by unfamiliar concepts, your amygdala is triggered, making you feel anxious, afraid, depressed, tired, or angry.



5. When you pay a greater amount of attention (attention density) to something, you find it easier to adapt.

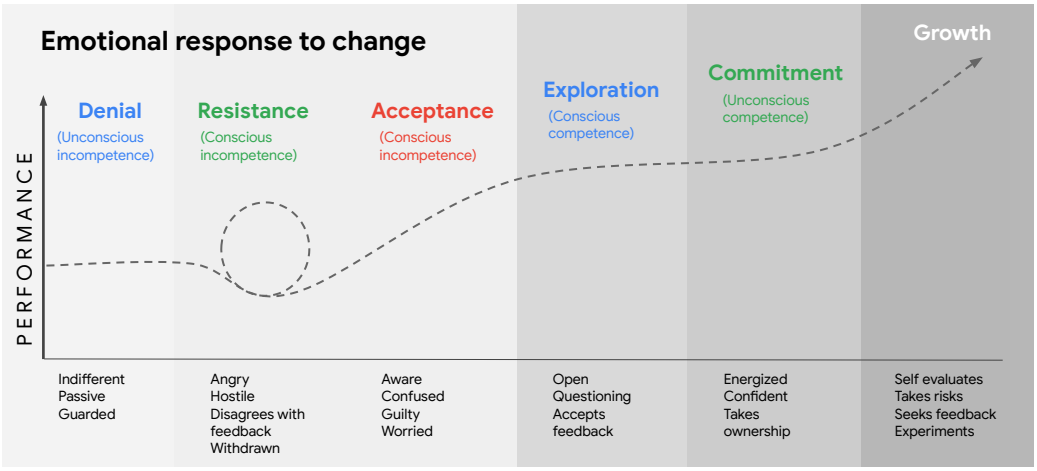


6. Habitual tasks feel easy and comfortable because they are hard wired and require little conscious thought (controlled by the basal ganglia).



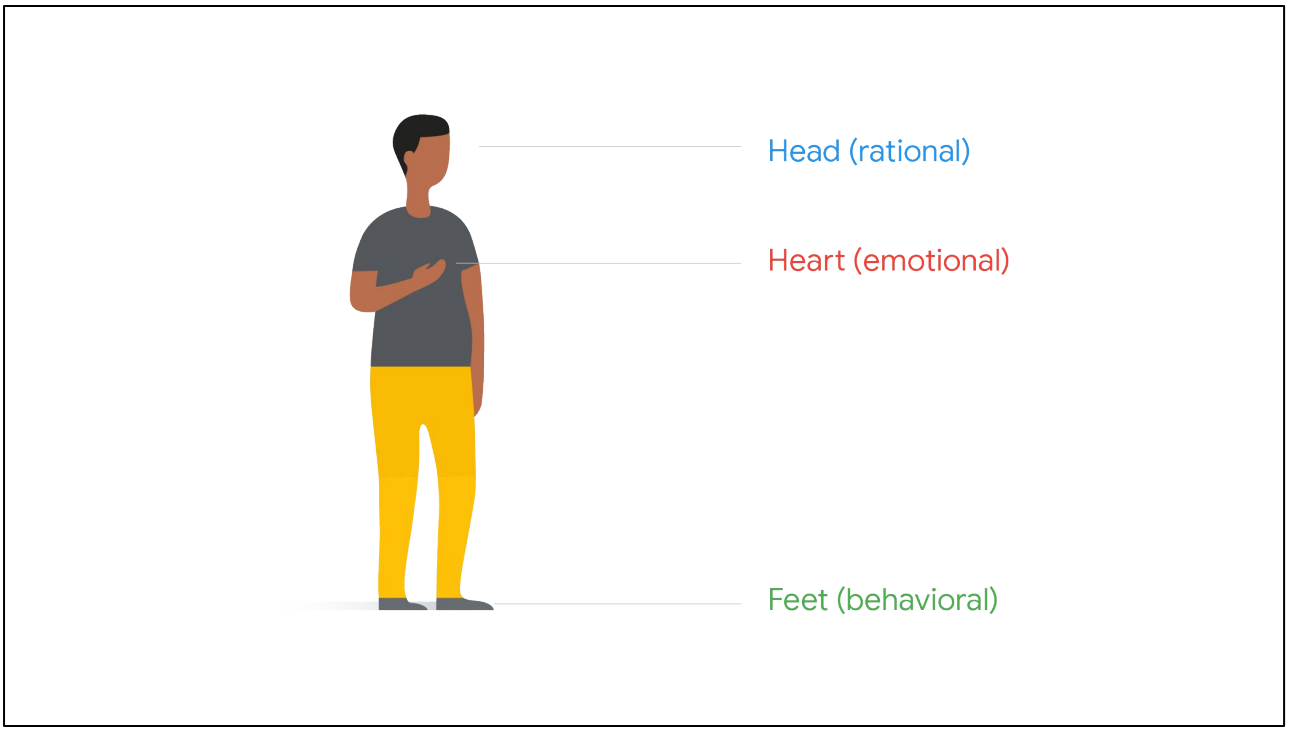
So how can you address these responses? Google has some recommended ways to manage and account for people's reactions to change in the teams you lead.

1. **Exclusion is painful:** Involve people in the change.
2. **Deception anxiety:** Set realistic expectations.
3. **Self-solve adrenaline:** Identify opportunities for co-creation and provide coaching rather than solutions.
4. **Amygdala hijack:** Simplify messaging and focus on key concepts per user group.
5. **Attention density:** Ensure that communications are engaging and training is interactive.
6. **Unconscious habit:** Allow people time to build new habits.



Keeping the neuroscience of change in mind, let's look at the stages of transition that individuals experience when going through change. There are different versions of the change curve, but this is the way we'll look at it today.

As you can see, there is a beginning, middle, and an end, yet it is completely normal for people to move backward and forward at different times.



Remember to present change as an **opportunity**, not a threat, to your teams. To do this, you'll want to connect with individuals on three levels:

1. **Head**, which is rational.
2. **Heart**, which is emotional.
3. **Feet**, which is behavioral.

For the Head, talk about why the change is happening and the strategic mission, vision, and rationale behind it.

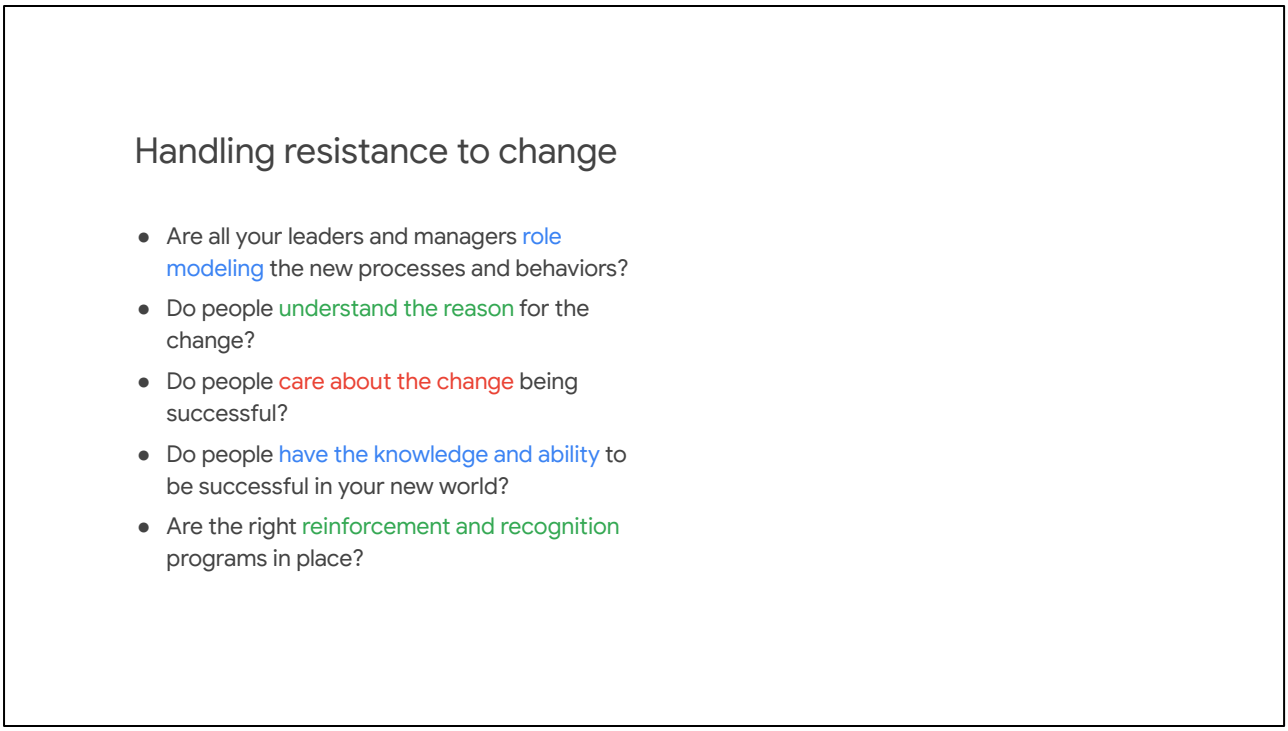
For the Heart, talk about why people should care. Remember that people can be egotistical and self-motivated. Address how the change will affect them personally in their day-to-day role and how it will impact them positively. Find a way to make them feel like they are a part of the change. People often just want to feel like they are a part of something.

And lastly, for the Feet, talk about the knowledge, skills, and resources you will provide to make sure they are successful in this change. Teams need support to make sure they are and feel competent when asked to change what they know.

Now that you've learned some ways to address change in your organization, let's look at how to handle **resistance to change.**

Research has shown that resistance is the primary reason changes fail in businesses. Resistance to change is usually a fear of loss. Specifically, people fear losing security or control, competence, relationships, or sense of direction.

Human reaction to loss is much stronger than human reaction to gain.



So how can you handle resistance to change? Keep this checklist in mind as you navigate your business's adoption of SRE practices.

- Are all your leaders and managers role modeling the new processes and behaviors?
- Do people understand the reason for the change?
- Do people care about the change being successful?
- Do people have the knowledge and ability to be successful in your new world?
- Are the right reinforcement and recognition programs in place?

If the answers to any of these are "no," we recommend that your leadership teams brainstorm how to address them, because all of these pieces need to happen for any successful organizational changes.

Let’s look at how one of our customers experienced resistance to change.

One of the leading commercial banks in Spain identified a number of processes that they wanted to automate. Their VP of Engineering, an executive sponsor for SRE implementation, was eager to showcase the benefits of the project to the business by investing in automation. The teams on the ground were very reluctant and skeptical about automating the earlier-identified processes. It turned out that they perceived automation as a direct threat to their jobs. Because they hadn’t been supported during the SRE implementation with a proper explanation of what the changes entailed and the impact changes would have on their projects, their resistance was prominent.

Now that we've covered SRE concepts that help organizations make their tomorrow better than today, the next module will cover the last step of the SRE journey, regulating workload.