Spotify Engineering Culture 1

One of the key success factors here in Spotify is our agile engineering culture.

Culture tends to be invisible. We do not notice it because it's there all the time. It’s kind of like the air we breathe.

But if everyone understands the culture, we are more likely to be able to keep it and even strengthen it as we grow. So that is the purpose of this video.

22 - When our first music player was released in 2008, we were pretty much a scrum company. Scrum is a well-established agile development approach. And it gave us a nice team-based culture.

However, a few years later, we had grown into a bunch of teams, and found that some of the standards scrum practices were actually getting in the way. So we decided to make all these optional. Rules are a good start, but then break them when needed.

44 - We decided that agile matters more than scrum, and agile principles matters more than any specific practices. So we renamed the "scrum master" to "agile coach", because we wanted servant leaders more than process masters.

We also started using the term "Squad" instead of "Scrum team." And our key driving force became autonomy.

64 - So what is an autonomous squad? A squad is a small cross-functional self-organizing team, usually less than 8 people. They sit together, and they have end-to-end responsibility for the stuff they build: design, commit, deploy, maintenance, operations ... the whole thing.

Each squad has a long-term mission, such as "make Spotify the best place to discover music," or internal stuff like infrastructure for A/B testing.

124 - Autonomy basically means that the squad decides what to build, how to build it, and how to work together while doing it.

There are of course some boundaries to this, such as the squad mission, the overall product strategy for whatever area they are working on, and short-term goals that are renegotiated every quarter.

143 - Our office is optimized for collaboration. Here is a typical squad area. The squad members work closely together, here, with adjustable desks, and easy access to each other's screens.

The gather over here in the lounge is for things like planning sessions and retrospectives. And back there is a huddle room, for smaller meetings, or just getting some quiet time. Almost all the walls are whiteboards.

202 - So ... why is autonomy so important? Well, because it is motivating, and motivated people build better stuff. Also autonomy makes us faster, by letting decisions happen locally in the squad, instead of via a bunch of managers,

committees and etc. It help us to minimize handoffs and waiting, so we can scale without getting bogged down with dependencies and coordination.

221 - Although each squad has its own mission, they need to be aligned with product strategy, company priorities, and other squads. Basically be a good citizen in the Spotify ecosystem.

The Spotify overall mission is more important than any individual squad. So the key principle is really "be autonomous, but don't suboptimize".

It is kinds of like a jazz band, although each musician is autonomous and plays his own instrument, they listen to each other and focus on whole song together. That's how great music is created.

249 - So our goal is loosely coupled, but tightly aligned squads. We're not quite there yet, but we have experienced a lot of with different ways getting closer. In fact that applies to most things in this video, this cultural description is really a mixture of what we are today and we are trying to become in the future.

305 - Alignment and autonomy may seem like different ends of a scale, as in more autonomy equals to less alignment. However, we think it more like two different dimensions.

- Down here is low alignment and low autonomy ... a micro-management culture. No high-level purposes, just shut up and follow orders.
- Up here is high alignment, but still low autonomy.

  So leaders are good at communicating what problem need to be solved, but they are also telling people how to solve it.
- High alignment and high autonomy means that leaders focus on what problem to solve, but let the teams figure out how to solve it.
- What about down here then ... Low alignment and high autonomy means: teams do whatever they want, and basically they are all running in different directions. Leaders are helpless, so our product becomes a Frankenstein.

We are trying hard to be up here, aligned autonomy. And we keep experimenting with different ways to achieve this.

358 - So alignment enables autonomy. The stronger alignment we have, the more autonomy we can afford to grant. That means the leader's job is to communicate what problem needs to be solved and why. And the squads collaborate with each other to find the best solution.

413 - One consequence of autonomy is that we have very little standardization. When people ask things like "What code editor do you use?" or "how do you plan?" the answer is mostly "depends on which squad." Some use scrum sprints, other Kanban; some estimate stories and measure velocity. Others do not. It's really up to each squad.

Instead of formal standards, we have a strong culture of cross-pollination. When enough squads use a specific practice or tool, such as Git, that becomes the path of least resistance, and other squads tend to pick the same tool.

Squads start supporting that tool and helping each other. And it becomes like a defacto standard. This informal approach gives us a healthy balance between consistency and flexibility.

455 - Our architecture is based on over a hundred separate systems, coded and deployed independently. There is plenty of interaction, but each system focuses on one specific need, such as playlist management, search or monitoring. We try to keep them small and decoupled, with clear interfaces and protocols.

Technically, each system is owned by one squad. In fact, most squads own several. But we have an internal-open source model. And our culture is more about sharing than owning.

521 - Suppose that squad one here needs something done in the system B, and squad two knows that code best, they will typically ask squad two to do it. However, if the squad two doesn't have enough time, or they have other priorities, then squad one doesn't necessarily need to wait. We hate waiting. Instead, they are welcome to go ahead and edit the code themself, and then ask the squad 2 to review the changes.

So anyone can edit any code. But we have a culture of peer code review. This improves quality and, more importantly, spreads knowledge.

548 - Over time, we evolved design guidelines, code standards and other things to reduce engineering fiction, but only when badly needed. So, on a scale from authoritarian to liberal, we definitely more on the liberal side.

600 - Now, none of this would work if it wasn't not for the people. We have a really strong culture of mutual respect. I keep hearing comments like "My colleagues are awesome!" People often give credit to each other for great work, and seldom take credit for themselves.

Considering how much talent we have here, there is surprisingly little ego.

618 One big thing for new hires is that autonomy is kind of scary at first. You and your squad mates are expected to find your own solutions. No one will tell you what to do. But it turns out, if you ask for help, you'll get lots of it, and fast.

There is a genuine respect for the fact that we are all in this boat together, and need to help each other to succeed.

639 We focus a lot on motivation. Here is an example: an actual email from the head of people operation.

Hi everyone,

Our employee satisfaction survey says 91% enjoy working here, and 4% do not.

Now that may seems like a pretty high satisfaction rate, especially considering our growing pain from 2006 to 2013, we have

doubled every year, and now have over 1200 people.

But then the e-mail continues...

This is of course not satisfactory, and we want to fix it.

If you are one of those unhappy 4%, please contact us. We are here for your sake, and nothing else.

712 - So good enough is not good enough. Half a year later things had improved, and satisfaction rate was up to 94%. With this strong focus on motivation, it is no coincidence that we have such this awesome reputation as workplace. Nevertheless, we do have plenty of problems to deal with. So we need to keep improving.

730 - OK. So we have over 50 squads spread across four cities. Some kind of structure is needed. Currently squads are grouped into "tribes". A tribe is a lightweight matrix. Each person is a member of a squad as well as a "chapter".

- The squad is the primary dimension, focusing on product release and quality.
- While the chapter is a competency area, such as quality assistance, agile coaching or web development.

As a squad member, my chapter leader is my formal manager, a servant leader, focusing on coaching, and mentoring me as an engineer. So I can switch squads without getting a new manager.

It's a pretty picture, eh? Except that it's not really true.

808 - In reality, the lines aren't nice and straight, and things keep changing. Here's a real-life example, from one moment in time, from one tribe.

And of course, it is all different by now. And that's OK! The most valuable communication happens in informal and unpredictable ways.

822 - To support this, we also have "guilds". A guild is a lightweight community of interest, where people across all the company gather and share knowledge within a specific area. For example, leadership, web development, or continuous delivery.

Anyone can join or leave a guild at any time. Guilds typically have mailing lists, bi-annual conferences and other informal communication methods.

843 - Most organizational charts are illusion. So our main focus is community, rather than hierarchical structures.

We've found that a strong enough community can get away with an informal volatile structure. If you always need to know exactly who is making decisions, you are in the wrong place.

859 - One thing that matters a lot for autonomy is: how easily can we get our stuff into production?

- If releasing is hard, we will be tempting to release seldom to avoid the pain. That means that each release is bigger, and therefore even harder. It's a vicious cycle.

- But if releasing is easy, we can release often. That means each release is smaller and therefore easier.

922 - To stay in this loop and avoid that that one, we encourage small, frequent releases. And invest heavy in test automation, and continuous delivery infrastructure.

Release should be routine, not a drama. Sometimes we make big investments to make releasing easier.

938 - For example, the original Spotify desktop client was a single monolithic application. In the early days, with just a handful of developers, that was fine.

But as we grew, this became a huge problem. Dozens of squads had to synchronize with each other for each release, and it could take months to get a stable version.

955 - Instead of creating lots of processes, rules stuff to manage this, we changed the architecture to enable decoupled releases. Using Chromium Embedded Framework, the client is now basically a web browser in disguise. Each section is like a frame in a website, and squads can release their own thing directly.

1012 - As part of this architectural change, we started seeing each client platform as a client app. It involves three different types of squads:

1. Client app squads,
2. Features squads, and
3. Infrastructure squads.

1. A feature squad focuses on one feature area, such as search. This squad will build, ship and maintain search related features on all platforms.
2. A client app squad focuses on making release easier on one specific client platform, such as desktop, IOS or Android.
3. Infrastructure squads focus on making other more effective. They provide tools and routines for things like continuous delivery, A/B testing, monitoring and operations.

1051 - Regardless of the current structure, we always strive for a self-service model, kind of like a buffet. The restaurant staff don't serves you directly. They enable you to serve yourself. So we avoid handoffs like The Plague.

For example, an operational squad, or client app squad does not put code into production for people. Instead, their job is to make it easy for feature squad to put their code in production.

1115 - Despite the self-service model, we sometimes need a bit of sync between squads making releases. We manage this using "release trains" and "feature toggles."

Each client app has a release train that departs on a regular schedule, typically every one to three weeks, depending on which client. Just like in the physical world, if trains depart frequently and reliably, you do not need much upfront planning. Just show up and take the next train.

1141 - Suppose these three squads are building stuff, and when the next release train arrives, the feature A, B and C are ready, while D is still in progress. The release train will include all four features, but the unfinished one is hidden with a feature toggle.

1156 - It may sound weird to release unfinished features and hide them. But it’s nice, because it exposes integration problems early and minimizes the need for code branches. Unmerged code hide problems, and is a form of technical debt.

1208 - Feature toggles lets us dynamically show and hide stuff in test as well as production. In addition to hiding unfinished work, we use this to A/B testing gradually roll out finished features.

1219 - All in all, our release process is better than it used to be. But we still see plenty of improve areas. So we’ll keep experimenting.

1227 - This may seem like a scary model. Letting each squad put their own stuff into production without any form of centralized control.

And we do screw up sometimes. But we have learned that trust is more important than control. Why would we hire someone we don't trust?

1242 - Agile at scale requires trust at scale, and that means "no politics", it also means "no fear". Fear not only kills trust, it kills innovation. Because if failure get punished, people won’t dare try new things.

1248 - So let's talk about failure. Actually no... Let's take a break... Get on your feet, get some coffee. Let's let these stuff thinking for a bit. And then come back when you're ready for part 2.

Spotify Engineering Culture 2

Hi, you are back. Great!

Now you've probably forgotten all about part one. So let’s do a quick recap.

1. Our culture is based on agile principles. All engineering happens in squads. And we try to keep them loosely coupled and tightly aligned.
2. We like cross-pollination, and have an internal open source model for code.
3. Squad do small and frequent releases, which is enabled by decoupling.
4. Our self-service model minimizes the need for handoffs. And we use "release train" and "feature toggles" to get stuff into production early and often.
5. And since culture is all about the people, we focused on motivation, community and trust, rather than structure and control. That was part one.

50 - And now, I' like to talk about "failure". Our founder Daniel put it nicely: "We aim to make mistakes faster than anyone else."

Yeah, I know, it sounds a bit crazy. But here is the idea. "To build something really cool. We will inevitably make mistakes along the way. Right?

108 - But each failure is also learning. So when we do fail, we want it to happen fast, so we can learn fast, and therefore improve fast. It's a strategy for long-term success.

It's like with kids, you can keep the toddler in the crib, and she will be safe, but she won't learn much, and won't be very happy. If instead you let she run around and explore the world. She'll fail and fall sometimes, but she'll be happier, and develop faster, and the wounds, well, they usually heal. So Spotify is a fail-friendly environment. We are more interested in fast failure recovery, than failure avoidance.

140 - Our internal blog has articles like: "Celebrate failure", and stories like "How we shot ourselves in the foot". Some squads even have a "fail wall", where people show off their latest failures and learnings.

150 - Failing without learning is, well, just failing. So when something goes wrong, we usually follow up with a postmortem. This is never about "Whose fault was it!" It's about:

- What happened?
- What did we learn?
- What will we change?"

205 - Postmortem is actually part of our incident management workflow. So an incident "ticket is not closed when the problem is solved" is closed when we capture the learnings to avoid the same problem in the future.

Fix the process, not just the product.

In addition, all squads do retrospect every few weeks to talk about what's working well, and what to improve next.

All in all, Spotify has strong culture of continuous improvement, driven from below and supported from above.

232 - However, failure must be non-lethal, or we don't live to fail again. So we promote the concept of "Limited Blast Radius".

240 - The architect is quite decoupled. So if a squad makes a mistake. It will usually impact a small part of the system. And not bring everything down.

And since the squad has end-to-end responsibility for their stuff without handoffs. They can usually fix the problem fast.

Also most new features are rolled out gradually, start with just a tiny percent of our users, and closely monitored. Once the feature proves to be stable, we gradually roll it out to the rest of the world.

303 - So if something goes wrong, it normally only affect a small part of the system, for a small number of users, over a short period of time.

This limited blaster radius give squad courage to do lots of small experiments, and learn really fast, instead of wasting time trying to predicting and control all risky events.

Mario Andretti put it nicely. "If everything is under control, you're going to slow!"

326 - All right, let's talk about product development. Our product development approach is based on lean startup principles. And it summarized by the mantra:

- Think it.
- Build it.
- Ship it.
- Tweak it.

338 - The biggest risk is always building the wrong thing. So before deciding to build a new product or major feature. We try to inform ourselves with research, to people actually want this, if this solves the real problem for them.

Then we define a narrative, kind of like a press release, or elevator pitch, showing up the benefits. For example, radio you can save, or follow your favorite artist.

358 - We also define hypothesis. How will this feature impact user behavior and our core metrics.

- Will they share more music?
- Will they login more often?

And we build various prototypes, and have people try them out to get a sense of what the feature might feel like, and help people react.

412 - Once we feel confident this thing is worth building. We go ahead and build a MVP -- minimal viable product, just enough to fulfil the narrative, but far from feature complete. You might call it the minimum lovable product.

425 - The next stage of learning happens once we put something into production. So we want to get there as quickly as possible. We release the MVP to just a few percent of users. And use techniques like A/B testing to measure the impact and test the hypothesis.

438 - The squad monitors the data, and continues tweaking and redeploying, until they see the designed impact. Then they gradually roll out to the rest of the world, while take a time needed to sort out practical stuff like operation issues, and scaling.

By the time the product and features are fully rolled out, we already know it's a success, because if it isn't we don't not roll it out.

459 - Impact is always more important than velocity. So a feature is not considered done until it has achieved the desired impact.

504 - Note that like most things in this video, this is how we try to work. But our actual track record, of course, varies.

511 - Now with all this experimentations going on, how do we actually plan? How do we know what is going to be released by which date. Well, the short answer is we mostly don't. We care more about innovation than predictability. And 100% predictability means 0% innovation. On a

scale, we probably be somewhere around here.

531 - Of course, sometimes we do need to make delivery commitments, like for partner integration or marketing events. And that is sometimes involves standard agile planning techniques, like velocity and burn up charts.

542 - But if we have to promise a date, we generally defer that commitment, until the features are already close to ready.

549 - By minimize the need for predictability, Squad can focused on delivering value, instead of being a slave to someone's arbitrary plan.

555 - One product owner said, I think of my squad as a group of volunteers that are here to work on something they are super-passionate about.

603 - So where do ideas come from. An amazing new product always start with a person and a spark inspiration. But it will only become real if people are allowed to play around, and try things out.

So we encourage everyone to spend about 10% of their time engaged in hacking. That when people get experiment, and build whatever they want.

631 - Like this dial a song product. Just pickup and dial the number of the song you want to listen to. Is it useful? It doesn't matter!

The point is: if we try enough ideas, we are bound to strike gold from time to time, and quite often, the knowledge gained is worth more than the actual hack itself. Plus it's fun.

641 - As a part of this, we do a Spotify wild hack week twice a year. Hundreds of people hacking oil for a whole week. The mantra is

1. Make cool things real!
2. Build whatever you want!
3. With whoever you want!
4. In whatever way you want!

And then we have a big demo and party on Friday.

656 - We are always surprised by how much cool stuff can be built in just a week with this kind of creative freedom. Whether it is a helicopter made of lollipop sticks, or a whole new way of discovering music.

It turns out that innovation is not really that hard. People are natural innovators. So just get out of their way, and let them try things out.

714 - In general, our culture is very experiment friendly. For example, should we use tool A or tool B? I don't know. Let's try both and compare.

Or, do we really need sprint planning meetings? I don't know, let's skip a few, and see if we miss them.

728 - Or should we show five or ten top songs on the singer's page. I don't know. Let's test both, and measure the impact. Even the Spotify wild hack week started as

an experiment, and now as a part of the culture.

740 - So instead of arguing initially to death, we are try to talk about things like

1. What's the hypothesis?
2. What did we learn?
3. What will we try next?

747 - This give us more data-driven decisions, and less opinion driven, ego driven, or authority driven decisions.

754 - Although we are happy to experiment, and try different ways of doing things. Our culture is very waste-repellent. People will quick stop doing anything that does not have any value. If it works, keep it. Otherwise dump it.

806 - For example, something that work for us so far, are:

1. Retrospectives
2. Daily standups
3. Google docs
4. Git
5. and Guild Unconferences

Something that don't work for us are:

1. Time reports
2. Handoffs
3. Separate test teams, or test phases
4. Task estimates

822 - We mostly just don't do these things. We also strongly allergic to

1. useless meetings
2. and anything remotely near corporate BS

828 - One common source of waste is what we called Big Projects. Basically anything that requires a lots of squads to work tightly coordinated for many month.

840 - Big project means big risk. So we are organized to minimize the need. And instead try hard to break project into a series of smaller efforts.

848 - Sometime however, there is a good reason to do a big project. And its potential benefits outweigh the risks. In those cases, we found some practices to be essential.

1. Visualized progress using various combinations of physical or electronical boards.
2. Do a daily sync meeting, where all squads involved meet up to resolve dependences.
3. Do a demo every week or two, for all the pieces come together. So we can evaluate the integrated produce together with stake holders.

914 - These practices we do reduce risk and waste because of improved collaboration and shorter feedback loop.

920 - We've also find that a project needs a small tight leadership group, to keep an eye on the big picture. Typically we have a tech lead, a product lead, and sometimes a design lead working tightly together.

930 - On a whole, we still experiment a lot with how to do big project, and we are not so good at it yet.

936 - One thing we keep wrestling with is growth pain. As we grow, we risk falling into chaos. But if we overcompensate, and have too much structure and process, we risk getting stuck in bureaucracy instead, and that is even worse.

951- So the key question is really: What is the minimum viable bureaucracy - the least amount of structure or process we can get away with, to avoid total chaos.

1000 - Both sides cost waste, but in different ways. So the waste repelling culture in agile mindset help us stay balanced.

1007 - The key thing about reducing waste is to visualize it, and talk about it often. So in addition to retrospectives and postmortems, many squads and tribes have big visible improvement boards that show things like what's block us? What are we doing about it?

1021 - We also like to talk about definition of awesome. For example, awesome for this squad means things like

1. Really finishing stuff.
2. Easily ramp up new team members.
3. And no recurring tasks and bugs.

1032- And our definition of Awesome Architecture includes:

1. I can build, test, and ship my feature within a week.
2. I use data to learn from it, and my improved version is live in week two.

1043 - Keep in mind, though, awesome is a direction, not a place. So it doesn't even have to be realistic. But if we can agree on what awesome would look like, it helps focus on our improvement efforts, and track progress.

1054 - Here is an example of an improvement tracking board inspired by a lead technique called Toyota Improvement Kata.

1. Top left shows what is the current situation? In this case, this squad was having quality problems.
2. Bottom left shows definition of awesome. In a perfect world, we have no quality problems at all.
3. Top right is a realistic target condition. If we were one step closer to awesome, what will that look like?
4. And finally, the bottom right shows the next three concrete actions. That will move us to the target condition. As things get done, the squad fills it up with new actions.

1126 - Boards like this live on the wall in the squad room. And are typically followed up at the next retrospective.

1132 - All right, I realize that making this video make it seems like everything in Spotify is just great.

Well, truth is we have plenty of problems to deal with. And I can give you a long list

of pain points. But I won't, because it will go out of date quickly.

1147 - We grow fast and change fast. And quite often, a seemly brilliant solution today will cause a nasty new problem tomorrow, just because we have grown, and everything is different.

However, most problems are short-lived, because we are actually do something about it.

1200 - This company is pretty good at changing the architecture, processes, organization, or whatever is needed to solve the problem. And that is really the key point. Healthy culture heals broken process.

1212 - Since culture is so important, we put a lot of effort into strengthen it. This video is just one small example. No one actually owns culture. But we do have quite a lot of people focusing on it. Group such as people operations had about 30 or so agile coaches spread across all squads.

1229 - And we do boot camps where new hires form a temporarily squad. They get to solve a real problem, while also learning about our tech stack and processes, and learning to work together as a team, all in one week.

It's intense, but along the way, they really get the culture. They often manage to put code in production in that time, which is impressive.

1246 - But again, failing is perfectly OK, as long as they learn. The main culture still spread through storytelling, whether it happens in blogs, at a postmortem, a demo or a lunch.

As long we keep sharing our successes or failures and learning from each other, I think the culture will stay healthy.

At the end of the day, culture, in any organization, is really just a sum of everyone's attitude and actions. You are the culture, so model the behavior you want to see.

That is it, we are done. I hope you enjoy this story, thanks for listening.