ChatGPT

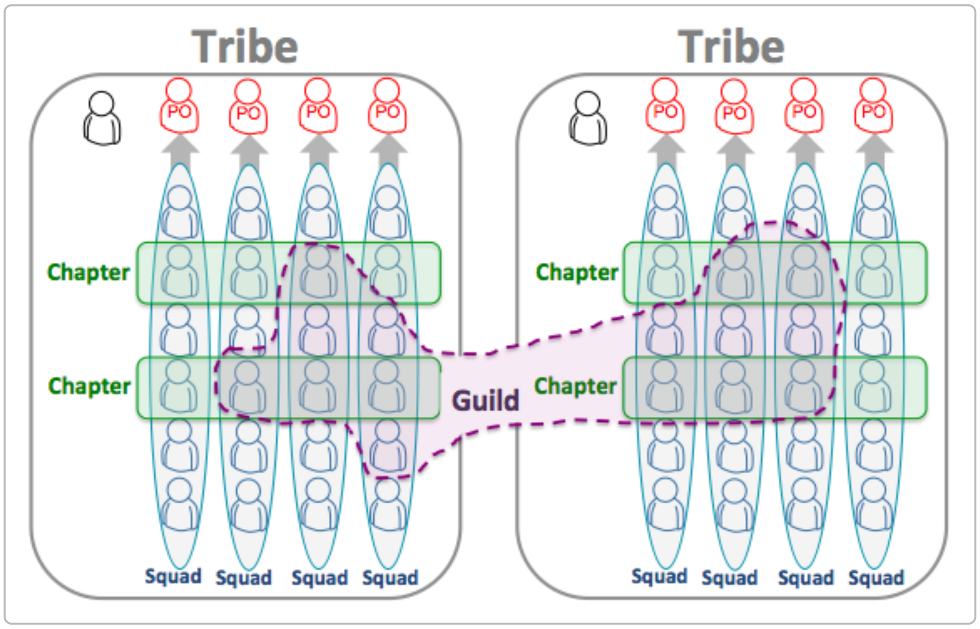
Implementing the Spotify Scrum Model at the Department Level

Overview of the Spotify Model and Its Origins

The **Spotify model** (often called the *"Spotify Engineering Culture"* or *"Spotify Agile Model"*) is a framework for scaling agile practices that was pioneered at Spotify around 2012 [1]. It was first described in a whitepaper by Henrik Kniberg and Anders Ivarsson, which outlined how Spotify maintained a strong agile culture as it grew to dozens of teams [1]. Instead of prescribing strict processes, the Spotify model emphasizes *organizational structure* and *culture* to enable agility at scale. As Kniberg himself noted, it was never meant as a one-size-fits-all method – it's "just an example of how one company works," rather than an official framework [2]. Over time, the model gained popularity as many organizations looked to Spotify's success as inspiration for their own agile transformations.

Key idea: *Organize around empowered teams and networks (rather than top-down hierarchy), to achieve agility with autonomy.* In traditional scaling frameworks, specific roles and ceremonies are mandated for coordination, whereas the Spotify model focuses on how to structure teams to be agile **without** heavy centralized processes [3]. This approach originated from Spotify's need to scale up rapidly (to hundreds of engineers across multiple offices) while preserving speed and innovation. The result was an informal *people-driven* model that has influenced agile thinking globally. Below, we break down the core components of this model and how they map to (or differ from) traditional Scrum roles.

Key Components: Squads, Tribes, Chapters, and Guilds



Structure of squads, tribes, chapters, and guilds in the Spotify model. Squads (blue) operate as autonomous teams inside a Tribe (gray boxes); Chapters (green) connect specialists across squads within a tribe, and Guilds (purple) span multiple tribes to share knowledge.

The Spotify model defines a **matrix of teams and groups** that replaces or augments traditional Scrum team structures. Each concept has a specific role in balancing **cross-functional autonomy** with **cross-team alignment**:

- **Squads:** A Squad is a small, cross-functional, self-organizing team – very similar to a Scrum team. Squads (typically 6–12 members) have all the skills needed to design, develop, test, and release product features, and they're meant to feel like mini-startups [4] [5]. Each squad has a defined **mission** (e.g. "improve the Android client" or "build the payment service") and a high degree of autonomy in how they work. In place of a dedicated Scrum Master, squads usually have an **Agile Coach** who helps them improve their process (this role is analogous to a Scrum Master, but one coach may support multiple squads) [6]. Each squad also includes a **Product Owner (PO)** responsible for prioritizing work and clarifying the product vision for the team [6]. The squad itself chooses its way of working – some squads use Scrum sprints, others Kanban or a mix – whatever best fits their context [7].

- **Tribes:** A Tribe is a collection of multiple squads that work in related product areas or share an overarching mission. For example, several squads working on different features of the same product will form a tribe. Tribes typically consist of up to 100 people or so (Spotify sized tribes based on **Dunbar's number** to ensure effective collaboration) [8] [9]. Each tribe has a **Tribe Lead** – a senior leader responsible for providing a conducive environment for squads in that tribe and helping coordinate across squads [10]. The tribe can be thought of as a *"mini organization"* or department focused on a product domain. Tribes periodically hold gatherings or demos where squads share what they're working on with each other, ensuring **alignment** and visibility across the tribe [11]. (In traditional Scrum terms, a tribe is roughly akin to a scaled agile *"team of teams"*, though Scrum itself doesn't define this layer.)

- **Chapters:** While squads are autonomous, Spotify recognized the need for specialists to share knowledge and standards across squads. A Chapter is a small group of people within a tribe who have similar skills or roles (for example, a "frontend developers chapter" or a "QA chapter"). All members of a chapter might belong to different squads, but they form a **chapter** to discuss best practices, align on technical standards, and mentor each other [12]. Chapters meet regularly (e.g. all testers across squads might meet to sync on testing practices). Each chapter has a **Chapter Lead**, who is usually a senior expert in that skill area and also serves as the **line manager** for the members of that chapter [13]. (This is a key difference from standard Scrum: rather than each team member reporting to a separate functional manager, their chapter lead is their manager, providing career development and ensuring technical excellence in that discipline. The chapter lead role has no direct equivalent in Scrum's framework.) Chapters help maintain **consistency** in how work is done across different squads – for instance, ensuring consistent engineering practices or design guidelines within a tribe.

- **Guilds:** A Guild is an informal, voluntary community of interest that cuts across the whole organization [14]. **Guilds** form around shared passions or topics – for example, a *DevOps guild*, *Machine Learning guild*, or *Agile Coaching guild*. Anyone from any squad or tribe can join a guild if they're interested in that topic. Guilds typically organize knowledge-sharing sessions, workshops, or hackathons, but unlike chapters, they are not limited to one tribe and do not have a formal hierarchy [14]. There is usually no formal leader of a guild; instead, one person may act as a coordinator to schedule meetings or curate information [15]. Guilds are essentially **communities of practice** (very similar to what many agile organizations do to share knowledge broadly). In Scrum terms, guilds don't exist formally, but you can think of them as extended networks for learning – ensuring that, say, all DevOps enthusiasts across the company can learn from each other even if they're on different squads in different tribes.

How these relate to Scrum: In summary, a **Squad** is analogous to a Scrum Team (with a Product Owner role present, and an Agile Coach playing the Scrum Master's role to support the team). **Tribes** and **Tribe Leads** introduce a lightweight scaling layer (comparable to a program or Agile Release Train in other frameworks) to coordinate multiple teams – something outside the scope of base Scrum. **Chapters** and **Guilds** have no direct Scrum equivalent; they are Spotify's solution to maintaining functional excellence and knowledge sharing in a scaled environment (other companies might use Communities of Practice similarly). The Spotify model thus extends the Scrum concept of a single team into a **network of teams**, connected by chapters and guilds rather than a strict hierarchy.

Core Principles: Autonomy, Alignment, and Trust

At the heart of the Spotify model is a culture that balances **team autonomy** with **organizational alignment**, all built on a foundation of **trust**. Spotify's mantra became achieving *"aligned autonomy"* – giving teams freedom to innovate while ensuring they're all moving in the same strategic direction. This balance is crucial: too much autonomy without alignment leads to chaos, while too much control stifles innovation. The model explicitly values trust in people to make decisions, rather than relying on heavy processes or micromanagement [16].

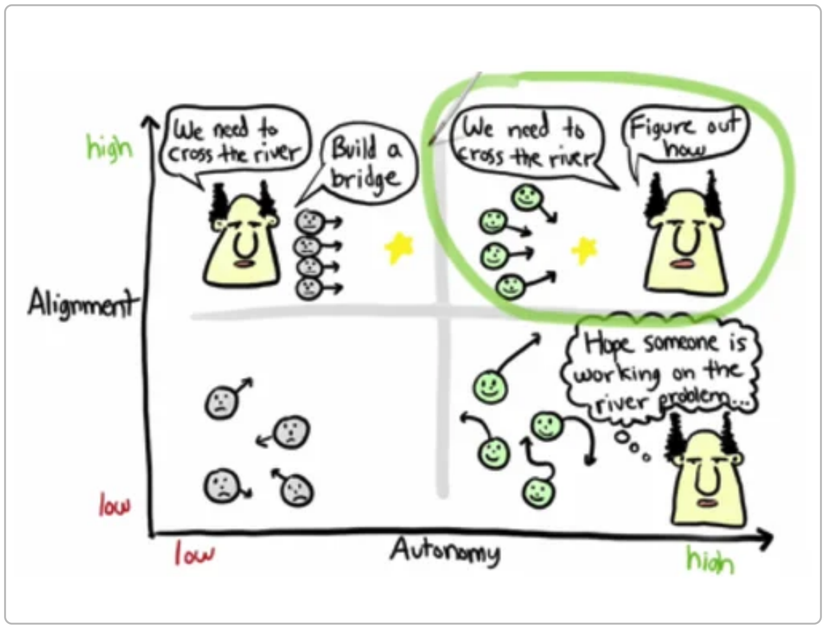

high
We need to cross the river
Build a bridge
We need to cross the river
Figure out how
Alignment
Hope someone is working on the river problem...
low
low
Autonomy
high

Illustration of balancing alignment and autonomy. In the Spotify model, leaders communicate **high-level objectives** ("We need to cross the river") and trust squads to figure out **how** to achieve them ("let's design a boat or a bridge") instead of prescribing solutions [17]. This high-autonomy, high-alignment approach is supported by a culture of mutual trust – leadership trusts teams to solve problems creatively, and teams trust that leadership will support them (and tolerate smart failures) in pursuit of innovation [18].

The **core principles** can be summarized as follows:

- **Autonomy:** Teams (squads) have a high degree of autonomy in how they work and make decisions. The Spotify model pushes decision-making down to the teams doing the work. Each squad is empowered to choose its development process, tools, and tactics to achieve its mission [16]. This autonomy encourages ownership and creativity. As one motto put it, *"control leads to compliance; autonomy leads to engagement."* Spotify found that giving teams autonomy resulted in more motivation and faster problem-solving than a top-down approach [16]. In practice, autonomy means a squad doesn't need to ask permission for most decisions – they are self-organizing, much like Scrum teams, but with even greater leeway to define "how" they work.

- **Alignment:** Autonomy only works well if everyone is aligned on the broader goals. **Alignment** means that all squads and tribes understand the product vision, company objectives, and how their work contributes to them. Spotify ensured strong alignment by communicating clear missions for squads and setting few but firm priorities at the top level [17]. For example, each squad has a *"long-term mission"* that guides its work, and tribes plan quarterly outcomes based on high-level priorities set by senior leadership [17]. Alignment mechanisms in the model include tribe gatherings (to share progress), transparent objectives, and the Product Owner role ensuring each squad's work fits the overall strategy. In essence, while squads decide **how** to work, leadership defines **what** problem to focus on – this keeps everyone moving in the same direction. A key insight from real implementations (e.g. ING Bank's adoption of the model) is that *"alignment enables autonomy"* – clear strategic direction actually **frees** teams to innovate responsibly [17].

- **Trust:** Underpinning autonomy and alignment is a **culture of trust**. Spotify's leadership deliberately trusted teams to do the right thing, which meant tolerating some risk and mistakes in exchange for speed and innovation [18]. Teams, in turn, trusted that they could experiment, voice concerns, and learn from failures without blame. This created a high-trust environment where information is shared openly (no hidden agendas) and people are not afraid to take initiative. Trust is evident in practices like allowing any developer to modify another team's code when necessary (with goodwill), or openly sharing lessons learned in guilds. The model assumes that people, when given freedom and context, will act in the company's best interest. This requires **management to let go of constant control**, which can be challenging in traditional cultures. However, with trust, Spotify saw increased transparency, more experimentation, and ultimately better products and happier teams [19]. In summary, trust is both a prerequisite and an outcome of the Spotify model – without trust, autonomy would collapse into disorder, and alignment would turn into top-down commands.

These principles are interdependent: **Autonomy** drives engagement and innovation, **alignment** ensures coherence and focus, and **trust** is the glue that makes the other two possible. A department adopting the Spotify model should nurture all three values, not just mimic the terminology. Next, we'll compare how this model differs from traditional Scrum or other Agile frameworks to further clarify its unique approach.

Comparison to Traditional Scrum and Agile Frameworks

The Spotify model emerged from real-world practice rather than theory, so it has some clear differences from textbook Scrum or other scaling frameworks (like SAFe, LeSS, Scrum@Scale, etc.). Understanding these differences will help in applying the model appropriately:

- **Framework vs. Model:** *Scrum* is a formal framework with a fixed set of roles, ceremonies, and artifacts defined in the Scrum Guide. The Spotify approach is not a formalized framework but a *model (example)* of how one company scaled agile – it's a **structure and culture template**, not a prescribed process [20] [2]. Unlike SAFe or Scrum@Scale, there is no official "Spotify Model Guide" or certification; it's intended to be adapted. In other words, Scrum tells you *"what to do each sprint"*, while the Spotify model shows *"how you might organize many teams and roles"*. This makes the Spotify model more flexible but also less explicitly defined than frameworks.

- **Organizational Structure vs. Process:** Traditional Scrum (and many agile scaling frameworks) put a lot of emphasis on specific processes (sprint planning, daily stand-ups, PI Planning, etc.) as the way to coordinate work. The Spotify model instead emphasizes **organizational structure** – how you arrange people into squads, tribes, chapters, guilds – and leaves the process details to each team [3]. For example, Scrum prescribes a Daily Scrum meeting for every team; in the Spotify model, a squad might choose to do daily stand-ups if they practice Scrum, or they might not if they use Kanban – that choice is up to them. The model focuses on setting up communication channels (like chapter meetings and tribe syncs) rather than mandating uniform rituals. This means less standardized process across teams, but more customization. As Atlassian's agile coach noted, *"specific practices (e.g. daily standups) are how [traditional] frameworks are executed, whereas the Spotify model focuses on how you structure an organization to enable agility"* [3].

- **Roles and Leadership:** Scrum defines three core roles (Product Owner, Scrum Master, Development Team) and no formal managers within the Scrum team. In the Spotify model, the **Product Owner** role remains central at the squad level (same as in Scrum) [6], but the **Scrum Master** role is usually replaced by an **Agile Coach** (who might work with multiple squads to mentor them on agile practices) [6]. Additionally, Spotify introduces **new roles** at the *tribe* and *chapter* level that Scrum doesn't have: **Tribe Leads** (like a department or program leader focusing on coordination and removing impediments for squads) [10], and **Chapter Leads** (line managers for specialists, responsible for people's growth in their discipline) [13]. There's also no direct equivalent of a "Scrum of Scrums" event in Spotify's model; instead, Tribe Leads and chapters handle cross-team coordination organically. This represents a more **matrix-style leadership**: one axis is product-focused (POs, Tribe Leads) and another is competency-focused (Chapter Leads). Traditional Scrum avoids matrix structures, whereas Spotify embraces a lightweight matrix to get the best of both worlds (autonomous teams *and* functional mentorship) [21].

- **Scaling and Coordination:** Out of the box, Scrum is intended for single teams. To apply Scrum to large programs, additional coordination mechanisms are needed (like Scrum of Scrums meetings, or scaled frameworks such as SAFe's Agile Release Trains). The Spotify model inherently addresses scaling by **grouping teams (squads) into tribes** and using chapters and guilds for alignment. For example, whereas SAFe might introduce "Agile Release Train Engineers" or "Solution Trains", Spotify uses the tribe structure and Tribe Leads to fulfill a similar need in a more informal way.

Dependencies between squads are managed through open communication in tribes and guilds, rather than a centralized PMO. One trade-off is that Spotify's model relies on **informal networks** to resolve dependencies (with Tribe Leads facilitating), rather than formal program planning events. This can be more agile and people-centric, but it requires a culture willing to communicate frequently. In practice, companies adopting the Spotify model often implement quarterly or incremental planning meetings at the tribe level (much like Scrum's Sprint Planning but scaled up) to synchronize squads – but these are not rigidly defined by the model itself, they are adapted to the organization's needs [17]. The key point: *Scrum* by itself doesn't tell you how to coordinate 10 teams working on one product, whereas *Spotify's model* provides a structural blueprint (squads in tribes, etc.) for that scenario.

- **Ceremonies and Rigor:** Scrum and other frameworks enforce certain ceremonies: e.g. every sprint you *must* do Sprint Planning, Daily Scrum, Sprint Review, Retrospective. The Spotify model is **less prescriptive** about ceremonies. Squads can run on different cadences – some squads might even have no fixed-length sprints (perhaps using Kanban). One squad might do two-week sprints with all Scrum ceremonies, while another squad does continuous flow. What matters is that they deliver and align with others. *"Squads may have ceremonies like sprint planning and retrospectives, but it's up to each squad to figure out the best way to get the job done,"* explains one summary of the model [22]. This flexibility can be a double-edged sword: it grants teams freedom to optimize their way of working, but it also means that management can't easily expect uniform metrics or timelines across squads. By contrast, in a framework like SAFe, all teams synchronize on a common sprint schedule and Program Increment. In the Spotify model, synchronization is achieved through communication and alignment practices rather than enforced timeboxes.

- **Evolution and Adaptability:** The Spotify model is deliberately *evolutionary*. It was explicitly described as a "snapshot" of Spotify's organization in 2012, expected to keep changing [23]. In fact, Spotify continued to evolve its approach – introducing new concepts like **"Trio"** (a triad of product, design, and tribe leads working together) and **"Alliance"** (a grouping of multiple tribes working toward a large objective) as the company grew [24] [25]. Traditional Scrum has changed only slightly over the years (the roles and events remain essentially the same), and big frameworks like SAFe are updated periodically but are fairly static between versions. The lesson from Spotify is that *no model stays perfect forever* – you should continuously inspect & adapt your organizational design. Even today, Spotify doesn't rigidly follow the exact model as originally described; they adjust it to fit their context [26]. For example, they found certain roles or practices needed tweaking as they scaled to thousands of employees. So, a difference in philosophy is: Scrum (and scaled frameworks) present a defined method to implement; the Spotify model invites you to start with a set of patterns and then evolve your own way of working from there [26]. It's more **principle-based** than rule-based.

In summary, the Spotify model shares Agile values with Scrum (teamwork, iterative delivery, etc.) but applies them via a **dynamic organizational design** rather than a fixed framework. It trades some consistency for greater flexibility. It also expands the scope from one team to a **network of teams**, using creative constructs (tribes, chapters, guilds) to handle issues of scale that Scrum by itself doesn't address. Many agile at scale frameworks (like Scrum@Scale, LeSS) have since incorporated similar ideas – in fact, Scrum@Scale acknowledges drawing inspiration from the early Spotify approach [27]. For a department leader, the takeaway is: **implementing Spotify's model will change team structures and leadership roles more than day-to-day team process**. You'll move to a multi-team structure with new alignment mechanisms, rather than enforcing a single standardized process on all teams.

Benefits of the Spotify Model for Software Teams

Adopting the Spotify model at a department level can yield several **benefits**, particularly for software organizations looking to scale agile methods while keeping teams innovative and motivated. Here are some key advantages:

- **Faster Delivery and Innovation:** By empowering small squads to make decisions quickly, the model reduces bottlenecks and overhead. Spotify's goal was to let squads *"move fast [and] ship software quickly, with minimum pain and overhead"*, which they achieved by minimizing bureaucracy [28]. Less waiting for approvals means faster development cycles. Teams can try ideas, release, and iterate rapidly. In a departmental context, this can translate to shorter time-to-market for features, because each squad can deploy independently (continuous delivery practices are often embraced). The autonomy of squads often fosters **innovation**, as developers feel ownership and experiment with new solutions without needing permission from multiple layers of management.

- **Less Formal Process, More Flexibility:** The Spotify model deliberately avoids one-size-fits-all processes. There are fewer mandatory meetings and reports than in some scaled frameworks [29]. This **lowers the procedural overhead** on teams. Each squad can tailor its workflow (Scrum, Kanban, etc.), which means the department can accommodate different work styles under one structure. For instance, a maintenance squad might use Kanban for flow, while a new development squad uses Scrum sprints – and both can coexist and coordinate through tribes/guilds. This flexibility makes the organization more adaptable: processes can evolve as teams learn, without needing a top-down rollout of a new methodology. It also helps teams feel that *process serves them, not the other way around.* In practice, this often leads to higher efficiency, because teams aren't performing ceremonies that don't add value to their specific work.

- **High Team Autonomy = High Engagement: Empowering teams** tends to increase motivation and accountability. Spotify found that giving squads autonomy and trusting them boosted team morale and engagement [16]. Developers who can choose how to solve problems often take greater pride in the outcome. Dan Pink's research on motivation (cited by Spotify) notes that autonomy is a key driver of engagement [16]. In a department using this model, team members feel "this is *our* product to build," rather than just executing management orders. This can lead to a culture of **ownership**, where squads proactively improve products and processes. With autonomy comes a sense of entrepreneurship at the team level – which can be very energizing in a software environment, leading to creative solutions. Engaged teams are also less likely to churn, improving retention of talent.

- **Alignment without Bureaucracy:** Despite the freedom given to squads, the model's structure (tribes, POs, tribe leads) ensures that all teams stay aligned with the department's goals. There is built-in **transparency** – through tribe gatherings, demos, and informal communication – so everyone knows what others are working on. This reduces duplicate work and conflicts. Importantly, alignment is achieved through **clear vision and communication** rather than heavy documentation or command-and-control. Leaders set a few high-level priorities, and squads align their missions accordingly [17]. For a software department, this means you can coordinate multiple product teams towards a common roadmap **without** having to institute rigid approval committees or top-down schedules. The tribe lead role and PO network help synchronize efforts in a lightweight manner. As a result, the organization can scale (to dozens of teams) while still feeling *"joined up"* – everyone knows the direction and how their work contributes to the whole.

- **Knowledge Sharing and Quality Improvement:** The **Chapter** and **Guild** system is a boon for spreading best practices and technical standards across the department. Chapters create a natural forum for specialists (e.g. all UX designers, all database engineers) to share knowledge, do code reviews across teams, and set coding standards or testing practices that raise the bar for quality [13]. This addresses a common challenge in scaled agile where teams can become siloed and reinvent the wheel; in the Spotify model, chapters and guilds act as the connective tissue to prevent silos. For example, if one squad discovers a great approach to automated testing, the QA Chapter meeting is where they'll showcase it so other squads benefit. Similarly, guilds let passionate volunteers drive improvements (a "DevOps guild" might organize a session on new CI/CD tools for any interested engineer). All of this leads to a **continuous improvement culture** at a broader scale than just within a single team. The outcome is often higher consistency in engineering practices and a **learning organization** where ideas spread organically. The model thus leverages the **collective intelligence** of the department: problems solved in one squad can rapidly propagate as lessons to others through chapters/guilds.

- **Scalability with Minimal Chaos:** The Spotify model was explicitly designed to handle growth – Spotify scaled to over 30 teams across multiple locations while maintaining agility [4]. For a department, this model provides a **scalable unit**: you can form new squads as needed, group them into a new tribe if they exceed a certain number, and so on. Because squads are semi-independent, adding more squads doesn't linearly increase complexity – it's managed by the tribe structures and the standard ways squads operate. The model also caps tribe size (~100 people), which helps prevent the team-of-teams from becoming too large to manage [9] [30]. In essence, it offers a **modular way to grow** an engineering organization. As you scale, you don't have to invent new management layers; you already have the Tribe concept to accommodate more teams. This can be less disruptive than, say, a re-org to introduce a new hierarchy. Furthermore, because alignment mechanisms are informal and continuous (instead of big bang planning), scaling up tends to be **incremental and less chaotic** than in some traditional approaches. The benefit is being able to handle an increase in team count or project scope without a proportional drop in communication or code quality. Many companies also see that this model scales globally – squads can be distributed, as long as tribes maintain cohesion (Spotify had squads in different cities, using video meetings for tribe syncs, etc., and it still worked due to the clear structure).

- **Employee Satisfaction and Talent Attraction:** An often overlooked benefit is that engineers generally appreciate the autonomy and learning opportunities that the Spotify model provides. Working in a culture of trust and empowerment can make the department a more attractive place to work (engineers often cite Spotify's culture as desirable). The model encourages a **"high-trust, high-responsibility"** environment, which can lead to happier employees [19]. Happy, engaged developers are more productive and creative. Additionally, the emphasis on guilds and learning means people have avenues to pursue their interests and grow their skills beyond their day-to-day squad work. This can improve job satisfaction and career growth. From a management perspective, empowered teams might even reduce the burden on managers since teams handle many decisions autonomously (managers can focus on strategy and development of people rather than micromanaging tasks).

Of course, realizing these benefits depends on implementing the model well and fostering the right culture (as we'll discuss in the challenges section). But many organizations that successfully adapted the Spotify model have reported improved delivery speed, innovation, and employee engagement as key outcomes [31] [19]. In short, the model can create an environment where **small, empowered teams build great software quickly** and **share what they learn** across the larger organization, which is exactly what most agile departments aspire to.

Common Challenges and Pitfalls to Watch Out For

Implementing the Spotify model is not a magic bullet – there are several common challenges and potential pitfalls a department should be mindful of. Here are the major ones, along with tips on avoiding them:

- **Copying the Model without Cultural Change:** Perhaps the biggest pitfall is treating the Spotify model as just a structural template (renaming teams to squads, managers to chapter leads, etc.) without adopting the underlying culture of trust, autonomy, and continuous improvement. Simply overlaying the squad/tribe terminology on a traditional command-and-control culture will *not* yield positive results [21]. In fact, Henrik Kniberg warned that if you just rename things without changing behaviors, you're *"putting lipstick on a pig"* [32]. The model's success at Spotify was deeply tied to their agile mindset. **Pitfall:** Organizations sometimes do a superficial copy – e.g., calling teams "squads" but still micromanaging them, or appointing "chapter leads" but not actually empowering people. This leads to confusion and cynicism among staff. **Mitigation:** Ensure leadership and teams understand the *principles* (autonomy, alignment, trust) and are committed to changing how they work, not just what they're called. Invest in coaching and cultural training so that the new structure is accompanied by new ways of thinking.

- **Increased Complexity (Matrix Management):** The Spotify model introduces a matrix-like organization (people belong to a squad and a chapter, with potentially different leaders). This can be **complex to manage**, especially in the early stages [33]. There's inherently more coordination needed – e.g., chapter leads need to stay in sync with squad POs about people's performance and workload. Organizations new to agile scaling might struggle with this dual reporting line concept. **Pitfall:** Without clarity, team members might be unsure "who is my boss?" or "which meeting takes priority, my chapter meeting or my squad's sprint review?" Additionally, managing resources across squads (like balancing how many front-end developers each squad needs) can become a juggling act for chapter leads and tribe leads. **Mitigation:** Clearly define the responsibilities and decision boundaries for each role early on. For example, establish that the squad's day-to-day priorities come from the PO, but personal development and career matters come from the Chapter Lead. Communicate this to all staff. It's also wise to start with a **small scope** (maybe one tribe) to pilot the model, so you can iron out coordination issues before scaling up.

- **Resistance to Change:** Moving to the Spotify model is a significant organizational change. Team members and middle managers may resist for various reasons [34]. Developers might worry about the new expectations that come with autonomy; managers might fear loss of control or unclear roles in the new setup. People who are used to traditional hierarchies or siloed departments may feel uneasy with the fluid, networked structure. **Pitfall:** Lack of buy-in can lead to passive resistance – e.g., chapter leads not actually engaging with their chapters, or squads ignoring guild meetings. This can stall the transformation. **Mitigation:** Engage people early and provide ample **education and communication** (as noted in the implementation steps below). Highlight success stories and quick wins from the new model to show its value. Provide training for new roles like Tribe Lead or Agile Coach. It's also helpful to have leadership explicitly champion the change, addressing fears and clarifying that careers and evaluations will adapt fairly to the new system (so people don't cling to old titles for security).

- **Lack of Standardization & Fragmentation:** While flexibility is a benefit, it can also become a drawback if taken to an extreme. In the Spotify model, each squad can choose its way of working – which might lead to very different processes, tools, and definitions of "done" across teams. This **lack of standardization** can make it hard to measure performance uniformly or coordinate work across squads [35]. For example, one squad might estimate in story points and another doesn't estimate at all; or one uses Jira and another uses Trello. **Pitfall:** If every squad diverges too much, management finds it difficult to get a consistent view of project status, and squad-to-squad collaboration can suffer (due to incompatible workflows). It could also overwhelm supporting roles like Agile Coaches if each squad needs a completely different approach. **Mitigation:** Align on some common guidelines – identify a *minimum viable bureaucracy*. For instance, you might agree on company-wide definitions for priority/severity, or require all squads to produce a quarterly OKR. Encourage knowledge sharing between squads (via chapters/guilds) about their working agreements, which can organically lead to convergence on best practices. Essentially, allow freedom but within guardrails. If certain standards are critical (security, regulatory processes), make those explicit so all squads follow them even as they maintain autonomy in other areas.

- **Overlapping Roles and Role Confusion:** In the Spotify setup, roles like Tribe Lead, Chapter Lead, Product Owner, and Agile Coach have to dance together. If these roles aren't clearly defined, there can be **overlap or gaps**. For example, a Tribe Lead and a Product Owner might both think they are responsible for strategy, or both neglect it expecting the other to handle it. Chapter Leads might struggle to evaluate their chapter members since they don't work with them daily (they work in squads), leading to confusion in performance reviews [36] [37]. Also, if a squad's Agile Coach is not clearly positioned, the team might ignore that guidance thinking "we don't have Scrum Masters." **Pitfall:** Without careful role clarity, decision-making can slow down (multiple people need to consult) or things slip through cracks (everyone thought someone else was handling a certain issue). Team members could receive conflicting guidance (e.g., chapter lead says "improve code quality" while PO says "just deliver features"). **Mitigation:** Define a RACI (Responsible, Accountable, Consulted, Informed) for key activities. For instance, clarify that the PO is accountable for the product backlog, the Tribe Lead is accountable for cross-squad alignment and dependency resolution, Chapter Lead is accountable for skills growth and technical standards, etc. Regular syncs between Tribe Leads and Chapter Leads can help keep everyone aligned (some companies create a "Tribe leadership team" including the Tribe Lead, the POs, and chapter leads of that tribe to coordinate decisions). Additionally, train chapter leads on how to gather feedback on their members from the squads' perspective (since they don't see daily work, 360-feedback from squad mates or POs becomes important for evaluations).

- **New Silos or Disconnects:** Ironically, if not nurtured, the very structures meant to enhance communication can turn into silos of their own. For example, *tribes* could become isolated from each other – focusing inward and not sharing lessons with other tribes. Similarly, chapters could become cliques that push their own agenda (e.g., an "architects chapter" enforcing heavy rules on squads without considering product needs). There's also a risk that a strong focus on guilds/chapters might dilute loyalty to the squad – people might feel more attached to their functional group than the cross-functional team. **Pitfall:** Without care, you might end up with *tribalism* (tribes optimizing locally but sub-optimizing for the company) or tension between chapter “standards” and squad “get-it-done” attitudes. In some reports, organizations found that tribes themselves needed alignment mechanisms (hence Spotify later added *Alliances* to group related tribes) [25]. **Mitigation:** Maintain mechanisms for *cross-tribe* communication. Encourage guilds that span the whole organization to thrive – this allows information and people to flow across tribal boundaries. Rotate people between squads occasionally to keep ideas fresh and networks broad. Ensure that the executive leadership periodically brings tribe leads together to discuss company-wide direction (so tribes don’t drift apart). Essentially, keep an eye on the *big picture* beyond each tribe. Also, balance alignment and autonomy continuously: if one tribe’s practices are diverging in a way that hurts another tribe, address it through leadership intervention. Remember that the model is supposed to be a network, not isolated islands – that requires conscious effort in larger departments.

- **Misalignment of Incentives:** This is a more subtle challenge – if your HR performance evaluations, rewards, and promotions are still geared toward individual or siloed achievements, they might conflict with the collaborative spirit of the Spotify model. For instance, if chapter leads are rewarded only on their chapter’s skill improvements, they might hoard the best people or discourage sharing talent with other areas. If squad success is measured in isolation, squads might not help each other as much. **Pitfall:** The organizational structure might change, but if old incentive systems remain, they can undermine the new model’s effectiveness. **Mitigation:** Update appraisal and reward systems to reflect squad goals and tribe goals. Encourage evaluating not just “what” people deliver, but *how* they collaborate and contribute to guilds/chapters (e.g., contributions to a guild could be recognized in performance reviews as positive). Ensure that Chapter Leads and Tribe Leads have shared objectives (perhaps an objective that ties the success of squads to collaboration). Essentially, align the “people systems” (hiring, review, promotions) with the model so that you reinforce the desired behaviors (teamwork, knowledge sharing, agility).

In summary, **successful implementation requires more than just re-drawing the org chart**. Many pitfalls are avoided by remembering that the Spotify model’s power comes from its *culture*. If your department fosters openness, trust, and a learning mindset, the structures will blossom. If not, the structures can falter or even do harm. A wise approach is to **introduce the model gradually**, get feedback often (Spotify famously used squad “health checks” to gauge how things were working), and be ready to adjust course. Avoid dogma – if something in the model isn’t working for your context, tweak it. The next section provides a step-by-step guide to beginning the implementation in a thoughtful way.

Step-by-Step Guidance to Begin Applying the Model

Implementing the Spotify model in your department should be done systematically. Below is a step-by-step guide to help you get started. This sequence assumes you have decided to adopt the model (at least in part) for your department or product group. The steps will take you from initial planning through to iterative refinement:

Step 1: Define the Scope and Vision – Start by determining **where** and **why** you are implementing the Spotify model [38]. Identify the part of the organization (e.g., a particular department, product line, or project area) that will adopt this model. It’s important to have a clear **vision** of what you hope to achieve (faster delivery, better cross-team alignment, etc.). Also, decide on scale: will it involve 5 squads in one tribe initially? Is this a pilot in one department with plans to expand? Defining the scope sets boundaries so everyone knows who is involved. For example, you might decide: “We will reorganize our Mobile Apps group (30 developers) using the Spotify model structure, covering the iOS and Android teams, and related QA and UX roles.” Communicate the *why* – e.g., “to improve collaboration and autonomy so we can release app updates more frequently.” Having management consensus on scope and goals will guide the subsequent steps.

Step 2: Form Cross-Functional Squads – Restructure (or form) your teams into **squads**. A squad should be a **cross-functional team** responsible for an end-to-end slice of product functionality [39]. Determine the right squad missions based on your product/domain. For each squad, ensure it has the necessary mix of roles (developers, QA, ops, UX, etc.) to deliver value independently. Keep squad size reasonable (ideally 5–9 people is a good start, aligning with two-pizza team concept) [39]. Assign a **Product Owner** to each squad – someone who will own the backlog and requirements for that team. If you have existing Scrum teams, they might map one-to-one to squads (possibly with some reconfiguration if they were not truly cross-functional). If you are coming from a functional silo org, you’ll need to break apart those silos and assemble new teams. It’s often useful to involve team members in this design: discuss what product areas make sense for squads and who would like to work on which mission. Once squads are identified, **clarify their missions** in writing (e.g., “Squad A: improve search feature of the product”), and introduce squad members to each other. This step basically lays the foundation: your execution units will now be these squads.

Step 3: Group Squads into Tribes – Next, organize related squads into **tribes** to manage coordination at a higher level [40]. Look at the missions or components your squads cover; group those that belong to a larger product area or business domain. For instance, if you have squads for “Search Feature” and “Recommendations Engine”, they could form a tribe focused on “User Content Discovery”. Aim for each tribe to have a **cohesive purpose** – tribes might correspond to what traditionally was a department or a major subsystem. Make sure the tribe size stays in a sensible range (if you end up with 200 people, consider splitting into two tribes; remember the guideline of ~100 people max to maintain social cohesion [9]). Appoint a **Tribe Lead** for each tribe [40]. This should be someone with enough authority and respect to facilitate across squads – often an engineering manager or senior tech/product leader. The Tribe Lead’s role is to be a servant leader: they will help resolve cross-squad impediments, coordinate tribe-wide priorities, and support the squads in that tribe. Essentially, you are setting up a *mini-organization* for each product area, with the Tribe Lead as its anchor. At this step, also plan practicals like: will the tribe have a common area or regular tribe meetings? Set initial tribe meeting cadences (e.g., a bi-weekly tribe sync where squads demo progress to each other).

Step 4: Establish Chapters – Define the **chapters** within each tribe [41]. Identify the major disciplines or skill areas represented in your squads – common ones are Software Engineering (which could be broken further into frontend, backend, etc.), Quality Assurance, UX Design, DevOps, etc. Within each tribe, gather the individuals of each discipline into a chapter. For example, all frontend developers across all squads in Tribe X form the “Frontend Chapter” of Tribe X. Do this for each key role where sharing knowledge is beneficial. Now assign a **Chapter Lead** for each chapter [41]. This is typically the most senior or experienced person in that skill set within the tribe, or a person with leadership aptitude in that discipline. The Chapter Lead might double-hat as a squad member too (for instance, a senior developer could be Chapter Lead for all developers in the tribe, while also coding on a squad). Clarify the Chapter Lead’s managerial responsibilities – they will be the line manager for people in their chapter, responsible for things like performance reviews, skill development, and hiring in that expertise. With chapters set up, schedule the first **chapter meeting** for each chapter to introduce members to each other and discuss their goals (e.g., "ensure we all follow good API design practices" for a backend chapter). This step effectively creates the horizontal lines that connect your squads, addressing the mentorship and standards aspect.

Step 5: Encourage Guilds – Promote the formation of **guilds** for interests that cut across the whole department or even the company [42]. Guilds are not assigned top-down; they emerge from the ground up. However, as a leader, you can facilitate their creation by providing tools and time. Announce that anyone can start or join a guild on any topic that they're passionate about. You might kick off a few likely guilds: for example, an "Agile Guild" (for agile coaches and interested team members to share retrospectives and process improvements), a "DevOps Guild" (to share CI/CD practices across tribes), or a "UI/UX Guild". Encourage people in similar roles from different tribes to create a guild if they want broader knowledge sharing than the chapter (since chapters don't cross tribe boundaries). Also, consider starting a **Spotify Model Adoption Guild** – basically a working group of volunteers from each squad to monitor how the implementation is going (as suggested in best practices [43]). Guilds can use communication channels (like a Slack channel or Confluence space) and should have the freedom to organize workshops or hack days. In this step, the key is to seed the idea of communities of interest – you don't have to formalize them too much, just let them grow. Over time, active guilds will become a backbone for cross-department learning.

Step 6: Adopt Agile Practices Within Squads – Ensure each squad adopts a suitable **agile methodology** or way of working for itself [44]. The Spotify model doesn't tell squads *how* to do agile, but to get the benefits, squads should be practicing agile or lean techniques (Scrum, Kanban, Scrumban, etc.). Provide guidance and coaching to squads to help them choose a process that fits. For instance, if a squad's work is well-defined feature development, Scrum with sprints might work; if a squad handles interrupt-driven operational tasks, Kanban might be better. The main point is each squad should have a *cadence* of planning, delivering, and reflecting (e.g., do retrospectives to improve continuously). **Agile Coaches** or Scrum Masters (if you have them) can assist squads during this transition. It's also important at this stage to align on engineering practices: encourage squads to use modern DevOps practices – continuous integration, automated testing, frequent releases – because the Spotify model favors quick, independent deployments. You might establish some shared tools if not already (source repo, CI pipeline) so squads can be independent in deployments. Additionally, encourage a **continuous improvement mindset**: squads should regularly reflect (via retrospectives or health checks) on how to get better. This step is about making sure the *engine of delivery* (the squad) is running agile, not falling back into waterfall or siloed habits even though the structure changed. Offer training in Scrum/Kanban if needed to squad members so that everyone has the skill to operate in an agile way.

Step 7: Foster a Data-Driven, Learning Culture – To complement autonomy and continuous improvement, start instilling a **data-driven mindset** in the teams [45]. This means encouraging squads to define metrics or OKRs (Objectives and Key Results) for their missions, and to use data to guide decisions. For a software product, this could involve instrumentation of features to collect usage data, A/B testing new ideas, and analyzing user feedback. Spotify famously uses metrics and A/B tests to inform product decisions. Within your department, you might establish a practice that each squad monitors at least one or two key metrics (e.g., page load time, conversion rate, user engagement score) so they can measure the impact of their work. Being data-driven also means making empirical decisions: if a squad tries a new approach, they gather evidence on its success and share that. This culture of **validated learning** helps in scaling because teams make decisions based on facts and experiments rather than guesswork. It can also be motivating – squads see the real customer impact of their work directly. As a department leader, support this by providing analytics tools, dashboards, and training on how to interpret data. Celebrate experiments (even those that fail) so long as they yield learnings. Over time, this will make your implementation of the Spotify model more resilient, because teams will iterate based on feedback and outcomes, not just intuition.

Step 8: Communicate and Educate Continually – Throughout the implementation, invest heavily in **communication and training** for all involved [46]. This step is about change management to ensure everyone understands the new model and their role in it. Hold kickoff workshops to explain the Spotify model concepts (squads, tribes, etc.) to the department – perhaps bring in an agile coach or use Henrik Kniberg's videos to illustrate how it works. Provide specific training for new roles: e.g., a session for Tribe Leads on servant leadership, a session for Chapter Leads on people development and cross-team influence, a session for Product Owners on handling their expanded responsibilities across possibly more autonomous teams. It's also crucial to set up communication channels: an internal wiki or handbook describing "How we work in the Spotify model" can be a reference for everyone. Encourage questions and openly discuss concerns (maybe hold an AMA – Ask Me Anything – with leadership on the changes). Regularly communicate wins as you roll out each part (for example, after forming squads and tribes, share a new org chart and some positive feedback quotes from team members). Also, update any documentation or tools to reflect the new terminology (it helps if your ticket system, for instance, uses the word "Squad" instead of "Team" to reinforce the language). Make sure supporting departments like HR and Finance are also informed (they may need to adjust how they interact with this new structure). Education is not one-and-done: plan for ongoing coaching. Perhaps assign an experienced agile coach or someone who has seen the Spotify model before to mentor your organization through the first few months. The more people understand *why* and *how* you're implementing this, the smoother the adoption will go.

Step 9: Monitor, Iterate, and Adjust – Once the model is in motion, treat the implementation itself in an agile way. **Monitor** the health of squads, tribes, and chapters, and be prepared to adapt as necessary [47]. You can use retrospectives at multiple levels: squads do retros for their own issues, and you might do a higher-level retrospective after, say, the first quarter of operating in the Spotify model to discuss what's working or not at the tribe/department level. Gather feedback from team members: this could be through surveys (Spotify used a "Squad Health Check" survey method to get structured feedback [48]), or through the guild you formed for model adoption. Look at objective indicators too: has deployment frequency improved? Are there fewer project delays? Or conversely, did certain dependencies get worse? **Adjust** course based on these observations. For example, you might find that one tribe ended up too large – so you split it into two smaller tribes. Or you might realize Chapter Leads are overburdened with line management, so you assign a second chapter lead or give them training. It's possible you'll tweak the model – that's expected! You could decide to implement a "Trio" concept (like Spotify's trio of Tribe Lead, Product, Design) if you notice a gap between product and technical leadership. Don't be afraid to deviate from the textbook if it solves a problem for your context. The goal is to *continuously improve your organizational design*. Make incremental changes and measure again. Over time, the model you run in your department may not look *exactly* like Spotify's, and that's okay – it will be your own evolved version that suits your needs. The key is that by monitoring and iterating, you ensure the structure continues to serve its purpose (enhancing agility) rather than becoming stale.

By following these steps, you'll systematically transition your department into the Spotify model. Remember to pace the implementation in a way that your organization can absorb; some companies do this over a few months, others take a year to fully shift. It often helps to pilot with one tribe (a few squads) first, learn lessons, and then expand the approach to other parts of the department. Through it all, keep the lines of communication open and support your people through the change.

Real-World Tips and Case Examples

As you embark on implementing the Spotify model, it's valuable to learn from others who have tried it. Here are some **real-world tips and insights** to consider:

- **Adapt the Model to Your Context (Don't Copy-Paste):** One of the most repeated pieces of advice is *not to rigidly copy* Spotify, but rather use the model as inspiration [49]. Every organization has its own culture, size, and challenges. Spotify's model was what *worked for Spotify at that time*. In your case, you might need to modify certain aspects. For example, you might decide not to use the term "guild" if it confuses your staff, or you might merge the Tribe Lead and Product Manager roles if that fits your context better. The success stories around the Spotify model involve companies picking and choosing elements that make sense for them. As Atlassian's agile coach put it, *"tweak the aspects of the model to fit your own environment. Your goal is not to be Spotify, but to leverage their model to improve how your organization works."* [49]. So be flexible – it's better to implement the *spirit* (autonomous teams, knowledge sharing) than the exact *letter* of the model.

- **Secure Leadership Support and Foster the Right Culture:** Implementing this model requires a supportive culture, which starts with leadership behavior. Leaders should actively promote **trust, empowerment, and transparency** – otherwise the model will stall. Leadership must be willing to give teams autonomy (and not interfere constantly) and must also invest time in alignment (clearly communicating goals). In the case of **ING Bank**, which famously adopted a Spotify-like model, top executives bought into the model early and even visited Spotify to learn [50]. They set a clear vision and then empowered squads to achieve it. Make sure your organizational leaders (department heads, senior managers) understand the mindset change involved. They might need to shift from "managing tasks" to "coaching and enabling teams". Also, equip your new Tribe Leads and Chapter Leads to embody servant leadership. A **high-trust culture** is essential: encourage managers to trust their people (avoid micromanagement) and encourage teams to trust each other (through openness). One concrete tip is to create an **internal communications forum** (or guild, as mentioned) for discussing the model adoption [43]. This provides transparency – everyone can see progress and issues – which builds trust that concerns are heard. Remember, culture changes take time, so leaders should consistently reinforce positive behaviors (like thanking teams for taking initiative, or openly admitting mistakes to encourage learning).

- **Emphasize Alignment (Clear Goals) to Empower Autonomy:** A key real-world insight is that **alignment is the prerequisite for effective autonomy**. In practice, this means you should invest effort in setting and communicating clear objectives at the department and tribe levels. For example, **ING's agile transformation** found that squads initially needed more top-down clarity – teams "quickly asked for more direction from the Executive Board to be able to make the right squad decisions" [17]. In response, ING's leadership set 6-7 company-wide priorities each quarter, and tribes then determined what to deliver to support those priorities [17]. This ensured everyone was rowing in the same direction. You can emulate this by establishing a simple OKR (Objectives and Key Results) structure or similar goal-setting mechanism: company goals → tribe goals → squad missions. When teams know the **"north star"**, they can self-organize much more effectively to get there. So, as a tip: **don't under-do alignment** because you fear micromanagement. Provide a strong strategic framework – it *enables* autonomy rather than hindering it. Regular alignment check-ins (quarterly planning, all-hands briefings on vision, etc.) are healthy. Just avoid telling squads *how* to execute;

focus on *what* outcome is needed and *why* it matters. Strong alignment also helps in prioritizing between squads and avoiding conflicts.

- **Iterate and Evolve the Model Over Time:** Treat your implementation as a living experiment. Even Spotify didn't freeze their model – they continued to change it as they grew [26]. You should plan to do the same. As you run with squads and tribes, periodically ask: what's not working? For instance, you might discover that your Chapter Leads are too overloaded with HR duties; maybe you then introduce a "People Operations" support or adjust the chapter size. Or perhaps communication is lacking between two tribes, so you institute a monthly cross-tribe meetup or introduce the concept of an *Alliance* (group of tribes) if needed [25]. **Don't be afraid to innovate on the model.** Another company's case might help illustrate: **Spotify itself no longer uses the exact original model** – they added things like Trios (Product, Design, Tech leads together) to improve decision-making [24] [26]. This shows that the model was not a final destination, but a journey. Therefore, use retrospectives and feedback to refine your structure. If a certain guild isn't providing value, you might dissolve or reform it. If squads get too independent and don't communicate, you might add a synchronization ceremony. Keep what works, adjust what doesn't. This continuous improvement approach to organization design will ensure the model remains optimal for your department's evolving needs.

- **Leverage Communities of Practice during and after the Transition:** Setting up **guilds or communities** is not just a structural element, but also a change management tool. One tip is to create a guild specifically for those driving the agile transition (as noted earlier) – sometimes called an "Agile guild" or "Coaching guild". This group can share lessons learned as you go, and propagate improvements quickly. Spotify credited a lot of their success to focusing on building a **community** and **transparency** around work [43]. In your department, encourage people to share what's working (maybe via internal blogs or brownbag sessions). Also, use guilds to break down any lingering barriers between old silos. For example, if historically Dev and Ops were separate, a DevOps guild can accelerate the cultural merge. Moreover, guilds can keep morale and enthusiasm up – they remind everyone that they're part of a bigger movement to change how work gets done. **Tip:** In the early phase, perhaps have a *guild coordinator* (maybe an enthusiastic team member) organize informal meetups on various topics to get guilds rolling. Free form discussion can surface issues that might not come up in formal meetings. Over time, these communities will sustain the knowledge sharing ethos of the Spotify model.

- **Learn from Case Studies but Mind the Differences:** A number of companies have shared experiences with the Spotify model – **ING Bank**, **LEGO**, **Netflix**, **Alibaba** are among those who took inspiration (each with their own twist). For instance, ING reported improved time-to-market and employee engagement after reorganizing into squads and chapters, but also noted they had to upgrade their talent management and give significant training to managers for the new roles. One of their learnings was that not every Product Owner was equipped to suddenly handle very autonomous teams, so they put effort into coaching POs to become more visionary leaders rather than task managers. **LEGO** initially adopted a similar model but found that they needed a bit more structure in some areas, so they combined Spotify's approach with elements of SAFe for planning at scale – demonstrating a hybrid can work. **Jeremiah Lee**, a former Spotify engineer, wrote about how parts of the model at Spotify were aspirational; this doesn't mean the model "failed," but it emphasizes that you should keep verifying what's reality versus theory [51] [52]. The takeaway is: gather insights from others (what pitfalls they hit, what benefits they saw) but be ready to chart your own path. No two implementations will be identical, and that's fine.

- **Manage Dependencies Proactively:** A practical tip from Spotify's own journey is to keep an eye on **inter-squad dependencies**. The model strives to reduce dependencies by having autonomous squads, but in complex products some level of dependency is inevitable (one squad's work may impact another's). Spotify used a quarterly "Squad Health Check" where squads self-rated things including how they felt about dependencies – this helped identify problematic areas to address [53]. You might implement a similar health check or at least have Tribe Leads bring dependency issues to light in tribe meetings. The sooner you spot a blocking dependency, the faster you can resolve it (perhaps by reassigning work, improving APIs between teams, or merging squads if necessary). The Spotify model gives you structures (like tribe gatherings, and chapter discussions) that can surface these issues, but you need to actively listen. **Tip:** Encourage a culture where squads openly flag when they're waiting on another squad or when something is impeding them, rather than suffering in silence. Tribe Leads can maintain a visible dependency board if needed. This ensures the "autonomy" doesn't turn into isolated teams stepping on each other's toes.

Finally, remember that implementing the Spotify model is a **significant transformation**. It can take time for people to adjust, and performance might dip during the transition period as new teams form and new roles learn the ropes. Patience and persistence are key. Celebrate small successes – e.g., the first release delivered by a fully autonomous squad, or the first time a guild produces a useful new standard for the organization. These build momentum and buy-in. If something isn't working, don't abandon the whole model outright; analyze and tweak. Many organizations succeed with this model by treating it not as a strict recipe, but as a *set of guiding principles* implemented with continuous learning.

In conclusion, the Spotify model offers a compelling vision for a modern, agile organization – one where **small, empowered squads** innovate quickly, supported by a network (tribes, chapters, guilds) that ensures knowledge flows and everyone stays aligned to a common purpose. With the overview, structure, principles, comparisons, and guidance provided in this report, a software team leader or manager should be equipped to understand the model and take the first steps in applying it. By spending the time to get the culture right and heeding the lessons from those who've gone before, you can adapt the Spotify model to your department and potentially achieve a higher level of agility, scalability, and team happiness in your software development efforts. Good luck on your journey to "Spotify-ing" your organization – and remember to **inspect, adapt, and rock on!**

Sources:

1. Kniberg, H. & Ivarsson, A. (2012). *Scaling Agile @ Spotify with Tribes, Squads, Chapters & Guilds* – (Whitepaper) [1] [54] [55] [13] [15].
2. Atlassian Agile Coach (2020). *The Spotify Model for Scaling Agile* [3] [6] [22] [16] [19] [21] [49] [25].
3. Xebia (2016). *Spotify-ing Your Organization – Insights from ING's Agile Transformation* [17].
4. PM Tips (2023). *The Spotify Way of Project Management* (Step-by-step implementation guide) [56] [57] [44] [58] [59] [60] [46].
5. Functionly (2021). *How Spotify Organizes its Org Chart* (Analysis of Spotify's structure and evolution) [61] [52].

1 3 5 6 7 8 10 12 13 14 15 16 18 19 21 22 24 25 26 27 28 29 31 32 43 49 54 55 **The Spotify Model for Scaling Agile | Atlassian**
https://www.atlassian.com/agile/agile-at-scale/spotify

2 20 **Is the Spotify Model an Early Instance of Scrum@Scale?**
https://agileeducation.org/spotify-model-scrum-at-scale/

4 9 11 23 30 **blog.crisp.se**
https://blog.crisp.se/wp-content/uploads/2012/11/SpotifyScaling.pdf

17 50 **Spotify-ing your organization | Xebia**
https://articles.xebia.com/spotify-ing-your-organization

33 34 35 38 39 40 41 42 44 45 46 47 56 57 58 59 60 **The Spotify Way of Project Management | Project Mangement Tips**
https://pmtips.xyz/2023/03/02/the-spotify-way-of-project-management/

36 37 48 **Scaling Agile @ Spotify with Tribes, Squads, Chapters & Guilds - Crisp's Blog**
https://blog.crisp.se/2012/11/14/henrikkniberg/scaling-agile-at-spotify

51 52 53 61 **How Spotify Organizes its Org Chart**
https://www.functionly.com/orginometry/real-org-charts/spotify-org-structure