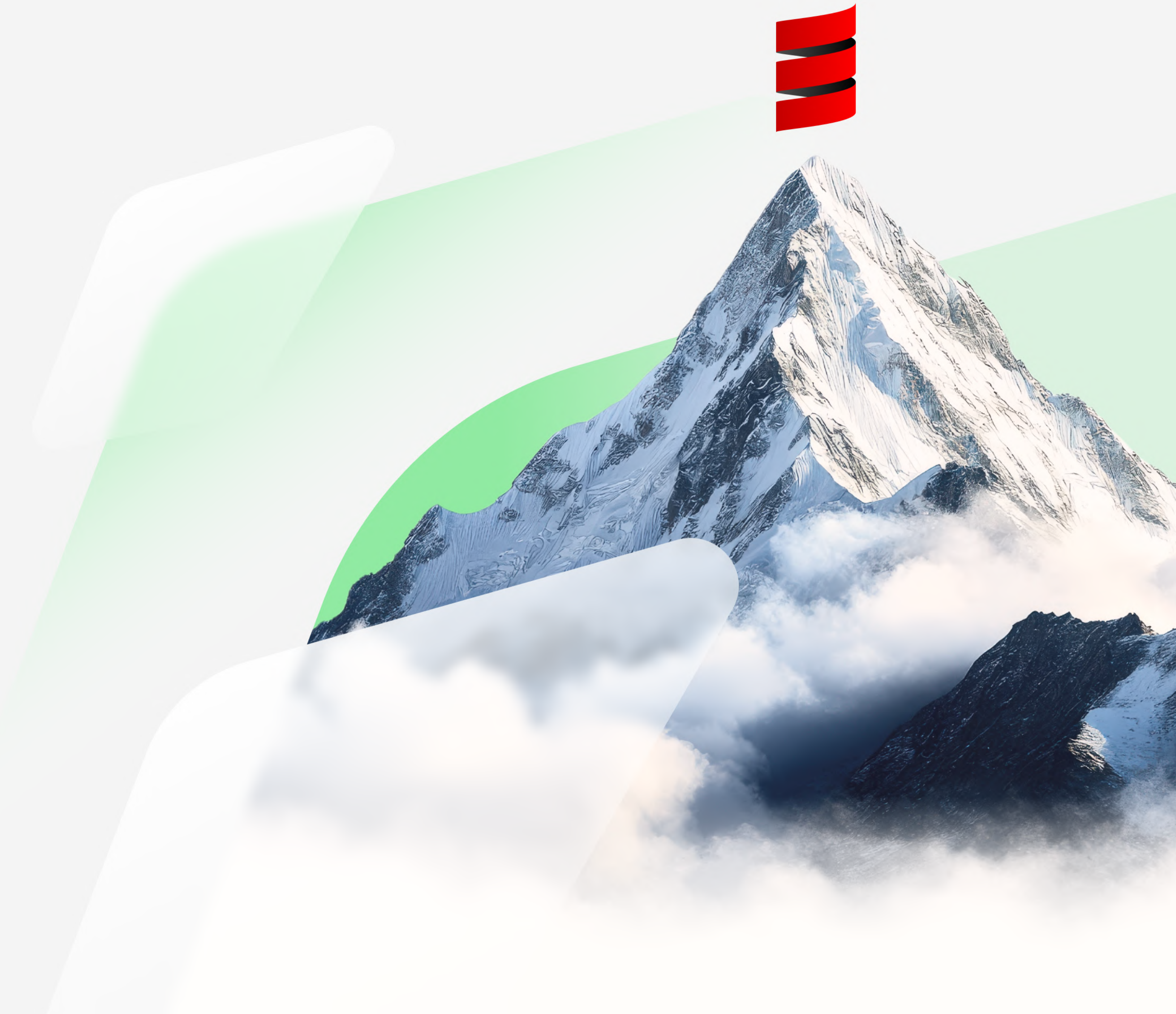
VIRTUSLAB

VIRTUSLAB.COM

Scala Project Maintenance Survey

REPORT



VirtusLab conducted a Scala Project Maintenance Survey in late autumn of 2024 and collected 232 complete responses.

The survey consisted of **37 questions,** some of which were asked conditionally based on the previous responses. Multiple questions allowed unstructured input from survey takers if none of the provided answers were a fit. There were **9 free-form questions** in total.

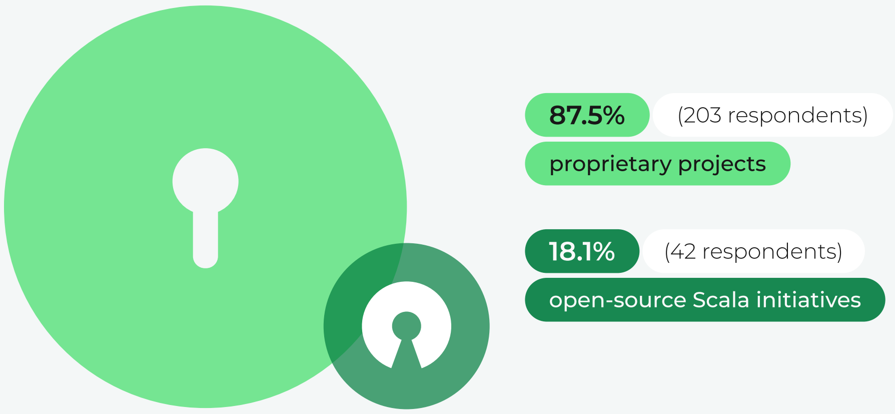
The respondents were reached through social media like X, LinkedIn and Mastodon, chats like community Discord servers, and on r/scala Reddit.

Table of contents

Respondents and their projects

PROJECT TYPE: PROPRIETARY VS OPEN SOURCE

> One of the foundational aspects of the survey focused on the nature of the respondents' projects. The vast majority, 87.5% (203 respondents), reported working on proprietary projects. Conversely, 18.1% (42 respondents) contributed to open-source Scala initiatives. This, in turn, means that 5.6% of projects are simultaneously proprietary and open-source as this was an available answer to this question.



PROJECT TYPE: PROPRIETARY VS OPEN SOURCE

PROFESSIONAL POSITIONS HELD BY RESPONDENTS

> Respondents represented a diverse range of roles within their organizations and the Scala community. Multiple options were allowed to be selected to reflect the reality of workplace role assignment. The breakdown of professional roles is as follows:

Software Engineers:

Constituting the majority, 75% (174 respondents) identified themselves as software engineers.

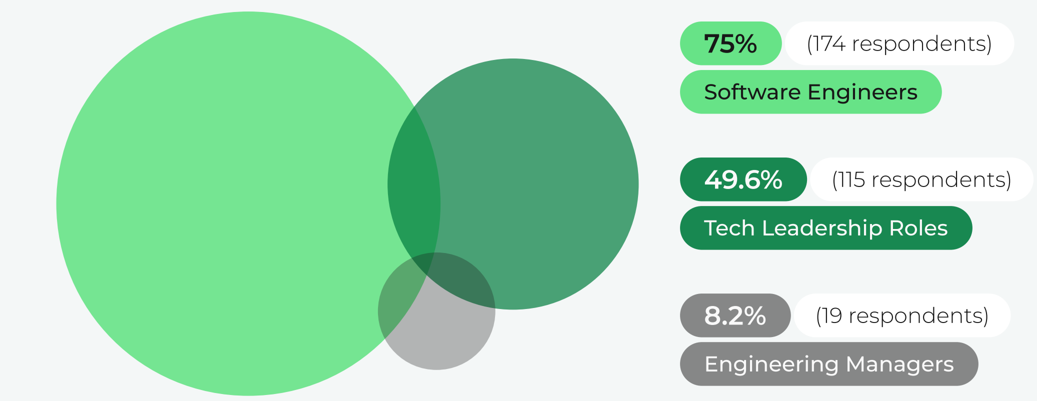
Tech Leadership Roles:

Approximately 49.6% (115 respondents) held senior technical positions such as tech leads, principal engineers, architects, or other leadership roles.

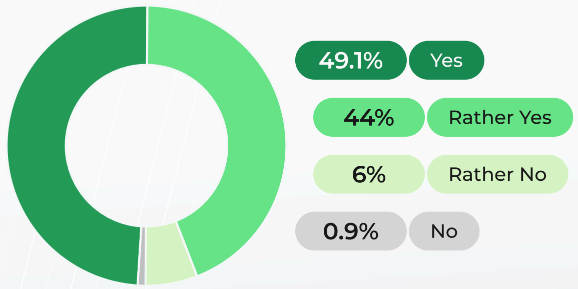
Engineering Managers:

A smaller segment, 8.2% (19 respondents), were engineering managers, reflecting a managerial perspective within the survey.

PROFESSIONAL POSITIONS HELD BY RESPONDENTS



GENERAL SATISFACTION WITH SCALA



Respondents provided feedback on their satisfaction with Scala:

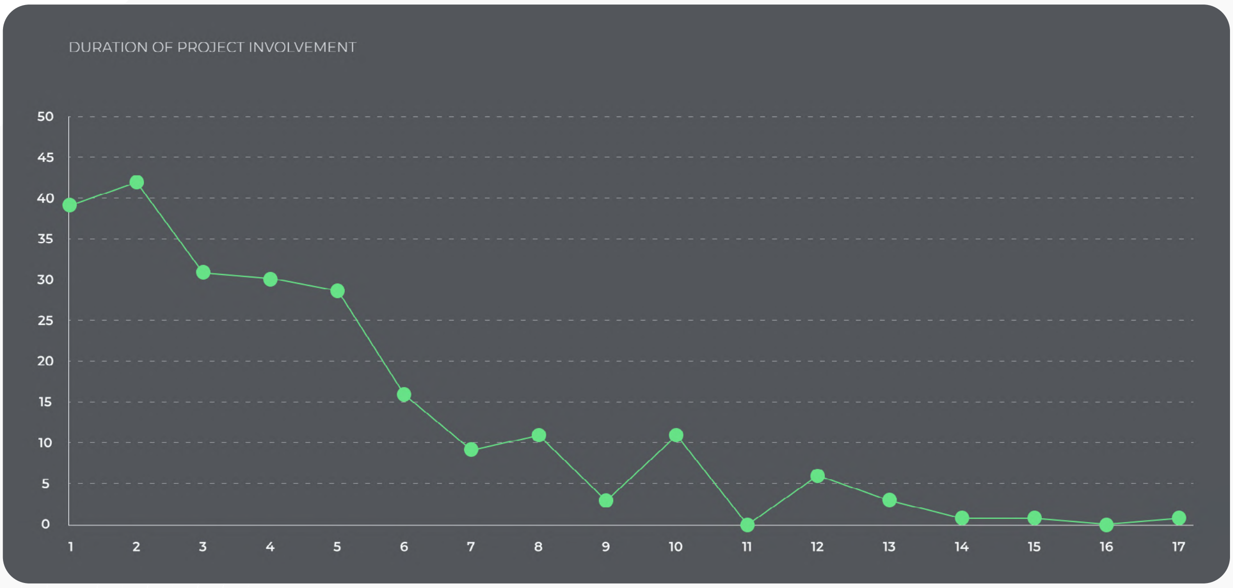
- Yes: 49.1%
- Rather Yes: 44%
- Rather No: 6%
- No: 0.9%

This result is significant as ***it highlights the generally positive sentiment toward Scala****. As the most popular functional programming language, its core value proposition - the ability to express complex ideas succinctly and robustly - remains a key factor in its appeal to software engineers.*

The survey identified challenges that need to be addressed, and ***this is where we (VirtusLab) focus our efforts*** *with collaborations with other organisations as a part of Scala Governance: EPFL, Akka, and Scala Center. We list concrete actions at the end of the document.*

DURATION OF PROJECT INVOLVEMENT

The survey explored respondents' tenure with their projects, capturing the varying levels of experience and engagement. As visualized in the accompanying chart, the majority of respondents reported between **1 and 5 years** of project involvement, accounting for nearly **70%** of responses. Beyond the 5-year mark, the number of contributors gradually declines, reflecting a smaller group of long-term participants who have been engaged for up to 17 years.



A NOTE ON THE GEOGRAPHICAL REACH OF THE SURVEY

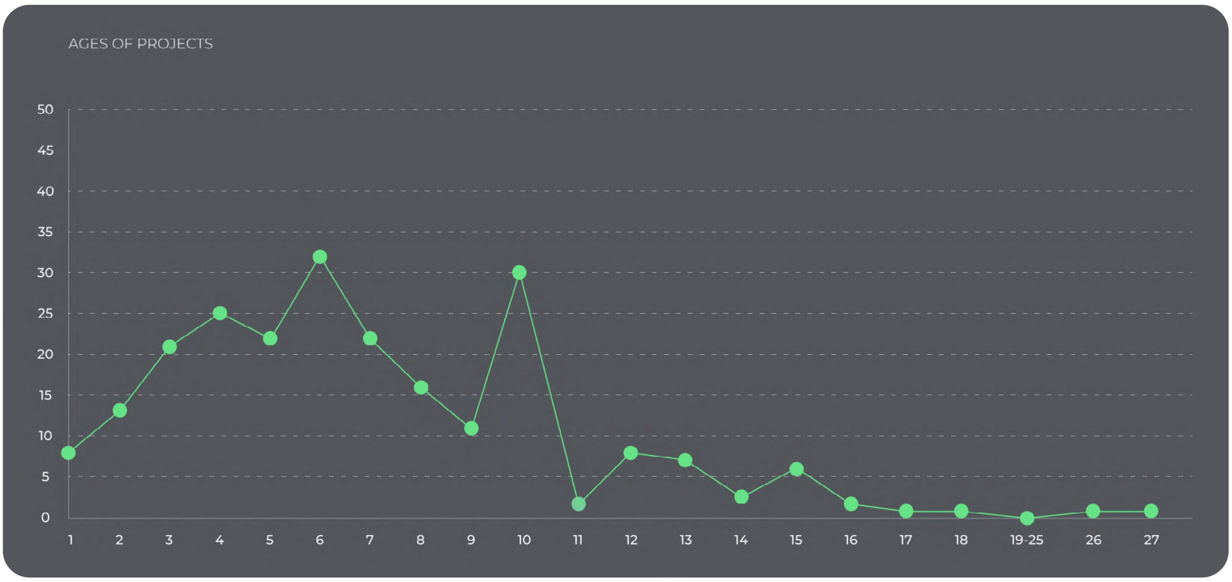
Based on the data gathered by the question on hiring markets, we can infer some insights regarding the survey's reach. ***Europe*** *dominated the participation, accounting for* ***69.84%*** *of respondents who shared their hiring market information. The only other significantly represented region was* ***North America*** *with* ***17.46%*** *of respondents hiring there. Unfortunately, other regions were represented by very small samples (below 3 hits).*

General ecosystem insights

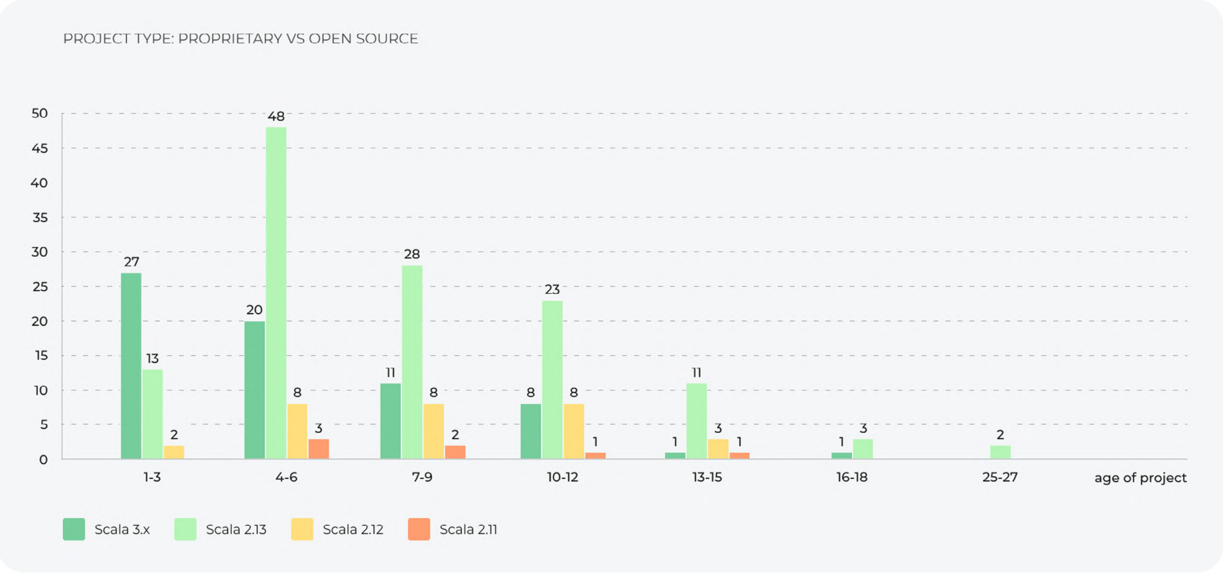
This section provides an overview of the broader Scala ecosystem based on the survey results. It examines project characteristics, technology usage, and the challenges faced by teams adopting Scala.

AGES OF PROJECTS

The survey revealed a wide range of project ages, with the average project being 7 years old and the median age at 6 years. The chart accompanying this section illustrates an observable decline in the number of new projects compared to six years ago. Specifically, one-year-old projects are only 25% as numerous as the peak value recorded six years ago, and two-year-old projects account for 41% of the same benchmark. This decline suggests a reduced rate of new projects adopting Scala, an issue that will be addressed further in the report.



Additional analysis between the age of projects and the primary version of Scala used showed that it's mostly the newer projects that are embracing Scala 3. Scala 2.13 is the most common version in all project age groups with the exception of the youngest projects between a year and three years old where Scala 3 is the clear winner.



TECHNOLOGICAL HOMOGENEITY VS. HETEROGENEITY

Respondents indicated whether their projects mix multiple Scala ecosystems or adhere to a single one:

Mix Ecosystems:

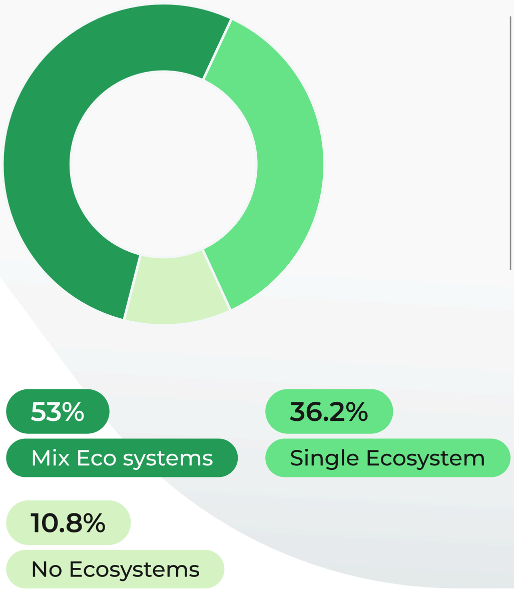
53% of respondents combine multiple ecosystems (e.g., using both Akka-related and Cats-related libraries).

Single Ecosystem:

36.2% of respondents stick to a single ecosystem (e.g., ZIO-based projects).

No Ecosystems:

10.8% of respondents rely solely on the standard library or proprietary frameworks.



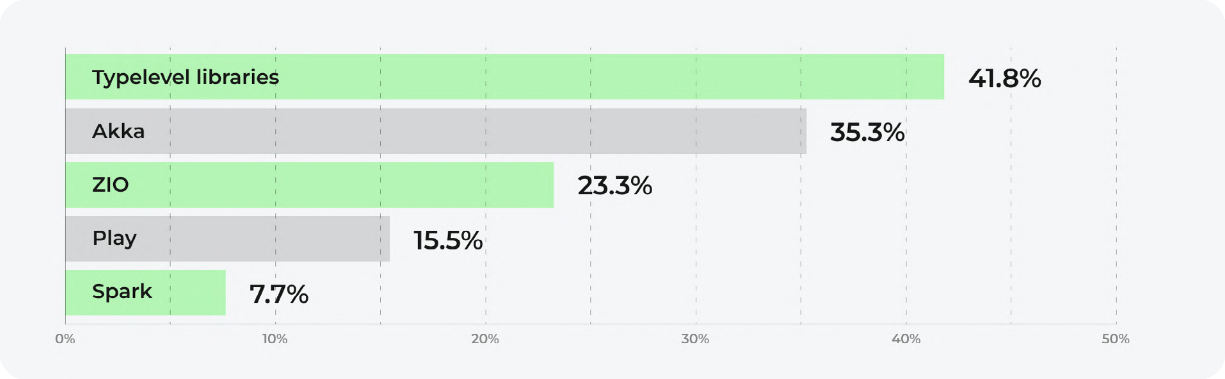
This data ***highlights the diversity of approaches*** *within the Scala community, with a significant proportion of projects integrating multiple ecosystems. A corollary of this metric is the significance of integration layer libraries and increased brittleness in the dependency layer introduced by these kinds of artifacts.*

DOMINATING TECHNOLOGIES

> Scala projects incorporate a wide palette of technologies, while Scala itself is used mostly on the backend.

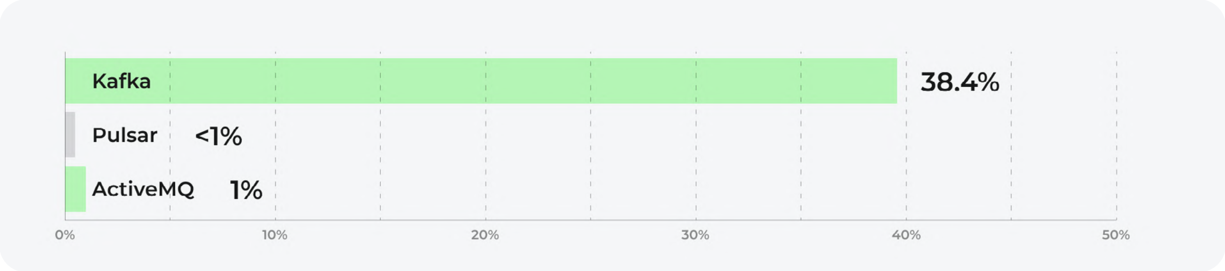
Ecosystems:

Typelevel libraries (41.8%) and Akka (35.3%) are among the most prominent, with ZIO (23.3%) and Play (15.5%) also frequently adopted. Spark was mentioned to be used in 7.7% of projects.



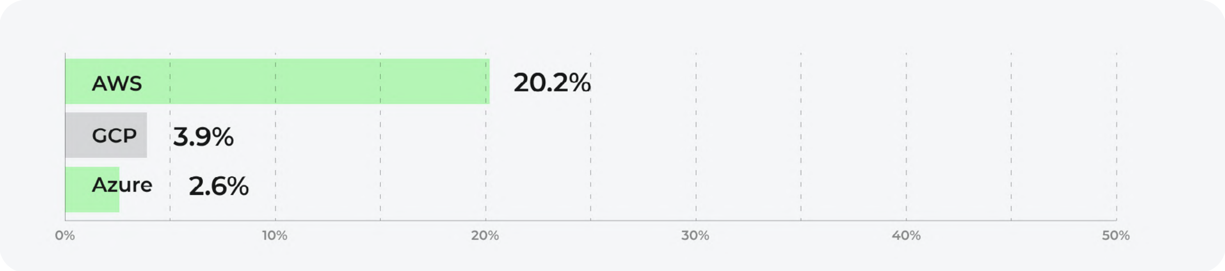
Queues:

Kafka is widely used (38.4%), with limited use of Pulsar (<1%) and ActiveMQ (1%).



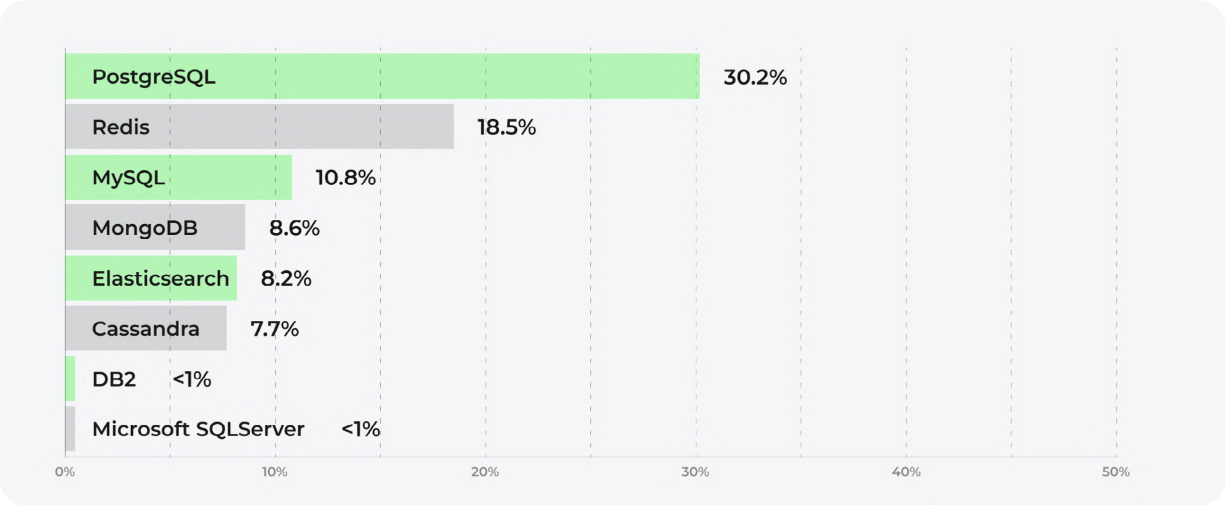
Cloud Providers:

AWS (20.2%) is the most commonly used, followed by GCP (3.9%) and Azure (2.6%).



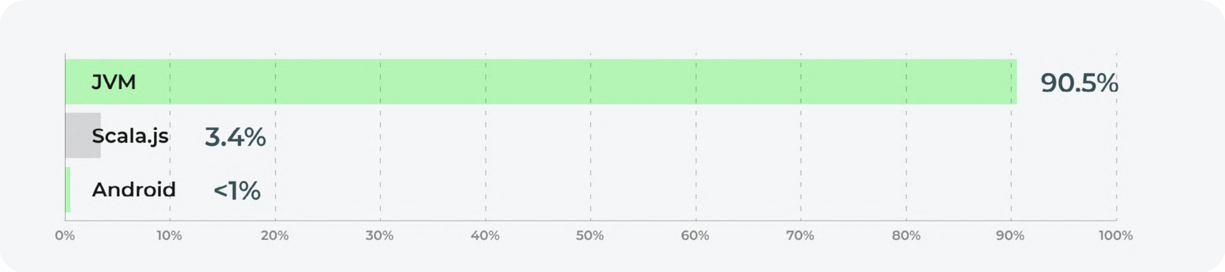
Databases:

PostgreSQL (30.2%) is the most popular, followed by Redis (18.5%), MySQL (10.8%), MongoDB (8.6%), Elasticsearch (8.2%) and Cassandra (7.7%). DB2 and Microsoft SQLServer were mentioned in less than one percent of projects.



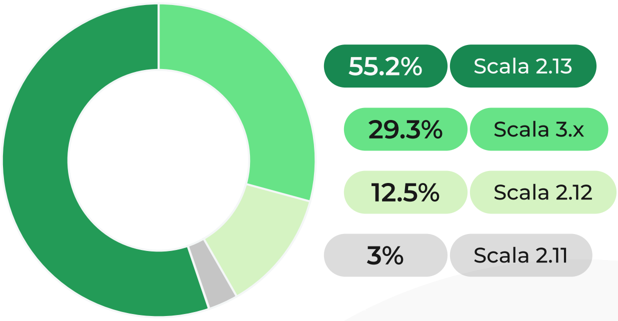
Platforms:

The JVM is overwhelmingly dominant (90.5% of projects), with a smaller subset using Scala.js (3.4%) and a single mention of Android. The remaining responses did not contain information about the target platform.



SCALA VERSIONS AND MIGRATION CHALLENGES

The survey highlighted the versions of Scala currently in use in commercial settings:



Scala 2.13:	55.2% of projects
Scala 3.x:	29.3% of projects
Scala 2.12:	12.5% of projects
Scala 2.11:	3% of projects

Scala 3 migration in commercial projects is making some progress, with **22.4%** of purely commercial projects (not marked as simultaneously being open source) using it as the most common version of the language.

On the other hand, **37% of respondents** reported no plans to migrate at all. Key reasons provided, sorted by prevalence, include:

- Tooling and ecosystem readiness concerns
- Lack of perceived business value of migration
- Resource constraints
- Preference for other languages
- Distrust or concerns about Scala 3 priorities
- Technical challenges

*This metric is quite grim as it reflects **the reality of breaking changes percolating throughout the ecosystem** and has to be addressed where possible by organisations as a part of Scala Governance: Akka, EPFL, Scala Center and VirtusLab. We, VirtusLab Scala team will propose solutions to alleviate these issues further in the report.*

Scala migration SERVICES

Efficiently navigate the migration to Scala 3 with VirtusLab's free support for your Scala projects and products.

Comprehensive workshop:

We help your team understand potential migration challenges and showcase Scala 3's capabilities in a 2–3 hour workshop.

In-depth codebase analysis:

Our experts assess your codebase to identify issues and provide a tailored migration plan.

Ongoing mentorship:

We guide you through code reviews, pair programming, and workshops to ensure a seamless migration.

Library migration support:

We assist in updating essential libraries to Scala 3 or offer effective workarounds when necessary.

Complete codebase migration:

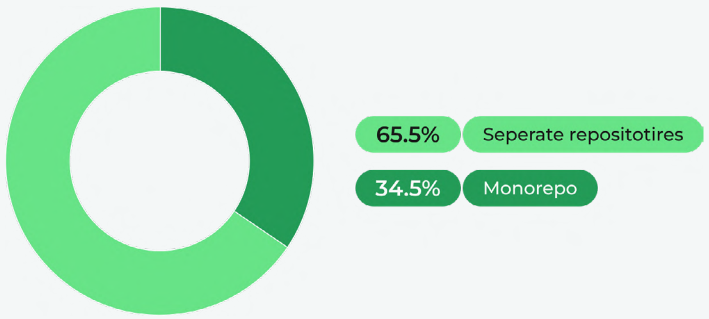
For smaller projects, we handle the entire migration; for larger ones, we tackle complex areas and mentor your team through the rest.

Visit our **Scala migration** page and learn more about how to elevate your projects.

Go to offer

REPOSITORY STRUCTURES, BUILD TOOLS AND IDEs

The survey explored **repository structures** and **build tools**:



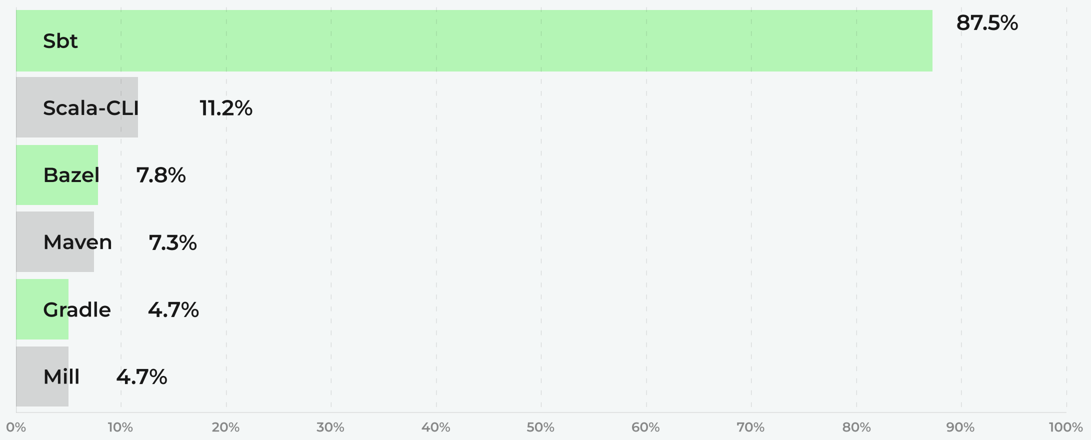
Repository structure:

65.5% of projects use separate repositories, while 34.5% adopt a monorepo approach.

Build tools:

sbt remains the most commonly used (87.5%), followed by: Scala-CLI (11.2%), Bazel (7.8%), Maven (7.3%), Gradle (4.7%) and Mill (4.7%).

BUILD TOOLS



Notable mentions include proprietary build tools, bleep, and npm for Scala.js projects.

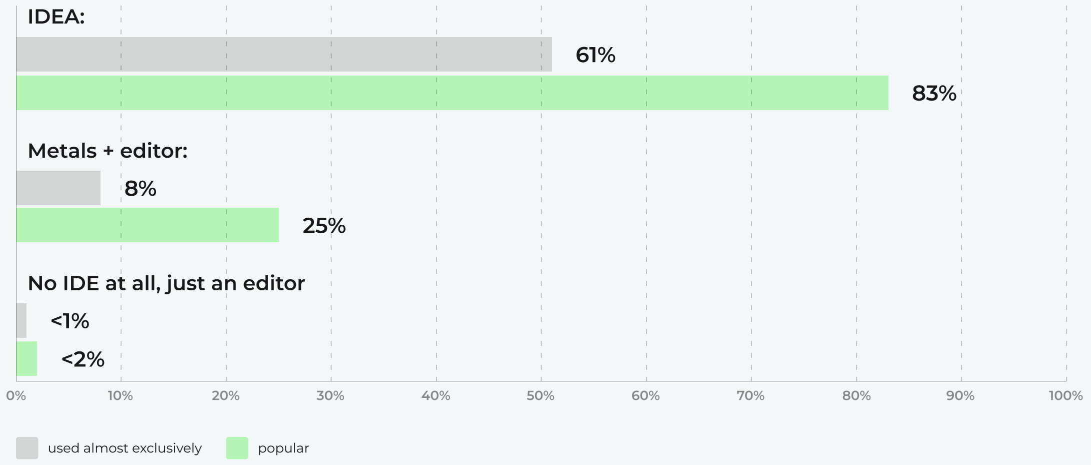
An interesting fact is that out of all monorepo-based projects 78.8% are using sbt and only 17.5% of them are using Bazel.

IDEs:

Intellij IDEA with Scala Plugin remains the top choice, with 61% of projects using it exclusively and 83% of projects having it as a popular developer choice.

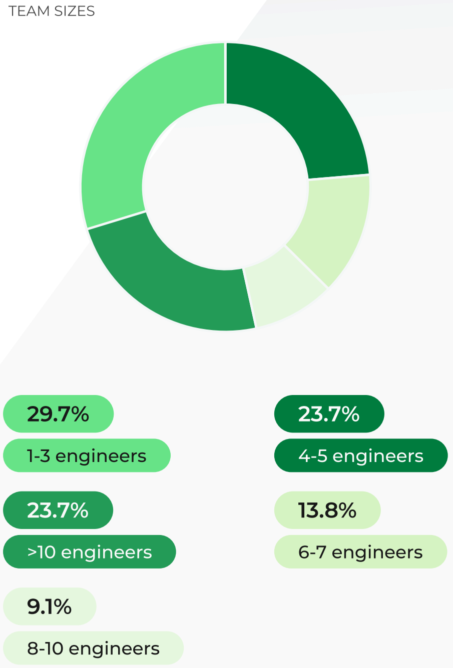
Metals are popular in 25% of projects, but only 8% of projects use them exclusively.

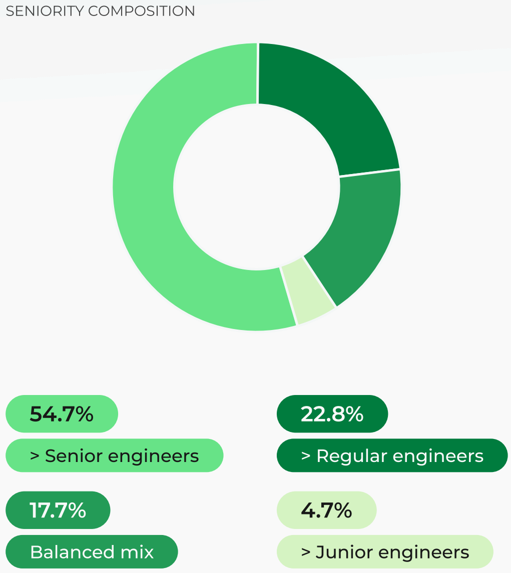
Raw text editors (without IDE features) are a niche option that's popular in less than 2% of projects and used exclusively in less than 1%.



TEAM SIZES AND SENIORITY COMPOSITION

The survey captured detailed insights into **team sizes** and their **seniority composition**:





Difficulties related to Scala and its ecosystem

COMPLEXITY OF SCALA IN THE WILD

The survey examined how different approaches to using Scala influence project complexity. Respondents provided the following breakdown of Scala usage styles:

Functional Scala:

The most popular approach, adopted by 38.8% of respondents, focuses on functional programming principles and libraries.

Regular Scala:

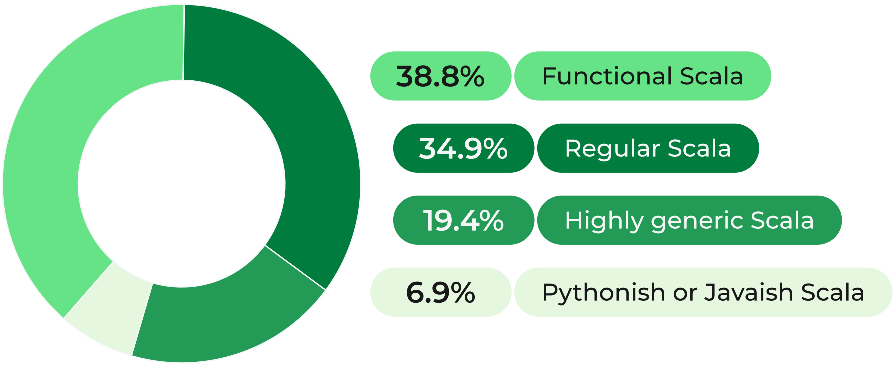
Used by 34.9% of respondents, this style retains a similarity to how the Scala compiler is written and to techniques pervasive in Play! / Akka ecosystems.

Highly generic Scala:

Employed by 19.4% of respondents, this style leverages advanced type system features, generic programming capabilities and/or metaprogramming.

Pythonish or Javaish Scala:

The least popular approach, adopted by only 6.9% of respondents, involves writing Scala in a manner akin to Python or Java, often emphasizing simplicity over idiomatic use.



IMPACT OF SCALA'S COMPLEXITY ON PRODUCTIVITY

Overall, **80.2%** of developers reported being satisfied with how Scala is used in their projects (specifically: that the selected Scala style does not impact their productivity negatively). However, several areas of concern emerged among the remaining **19.8%** of engineers:

Unnecessary complexity:

10.3% of developers indicated that Scala introduces unnecessary complexity in their projects, an issue that requires further attention.

Code fragility:

Only 1.7% of respondents believed that their Scala usage causes unnecessary breakage, suggesting that codebase fragility is not a widespread concern.

Negative productivity impact:

Approximately 7.8% of respondents highlighted specific reasons for Scala's negative impact on productivity. These reasons can be grouped into the following categories:

1. **Technical challenges:**
   - Long compilation times and high memory usage.
   - Hard-to-predict performance impacts when targeting Scala Native or Scala.js.
   - Insufficient IDE support for Scala 3.
   - Difficult tooling setup, especially in enterprise environments.

2. **Framework and ecosystem fragmentation issues:**
   - Interoperability challenges between frameworks like ZIO and Akka, lead to the need to choose between incomplete or overly complex solutions.
   - Unnecessary migrations between ecosystems (e.g., Akka → Monix → Cats → ZIO).

3. **Code complexity:**
   - Overengineering with type classes, complex abstractions, and unnecessary monadic transformations.
   - Legacy code is often simpler and more productive than newer, overly abstract patterns.
   - Some parts of "functional Scala" are viewed as unnecessarily complex and counterproductive.
   - Flexibility in Scala allows for novelty-driven over-complication by some developers.

4 **Developer Experience:**

- The “cake pattern” in early implementations slowed productivity significantly.
- Lack of cohesion in development styles, with preferences ranging from concrete IO to tagless final.

5 **Scala 3 migration:**

- Challenges in migrating to Scala 3, sometimes requiring full rewrites.

> *Some of these issues reflect high innovation rate in Scala’s ecosystem and are a testament to users’ search for best techniques, while others are strictly related to target platforms of the language. Unfortunately, none of these issues are easy to fix as they are deeply ingrained into the existing code written in Scala. On the other hand, most of these issues can be addressed going forward, and some propositions will be offered in subsequent sections of the report.*

BIGGEST TIME SINKS AND HINDRANCES IN SCALA DEVELOPMENT

> Respondents identified several major time sinks and productivity hindrances:

1 **State of the ecosystem:**

- Libraries becoming unmaintained and causing friction in dependency updates: 44% of responses.
- Custom or poorly maintained internal frameworks blocking updates of business components: 33.6% of responses.
- Libraries causing binary compatibility issues at runtime: 21.6% of responses.
- Lack of libraries to solve specific problems: 15.1% of responses.
- Poor integrations or lack of integrations between libraries: 11.6% of responses.
- Poor quality of libraries: 3.4% of responses.

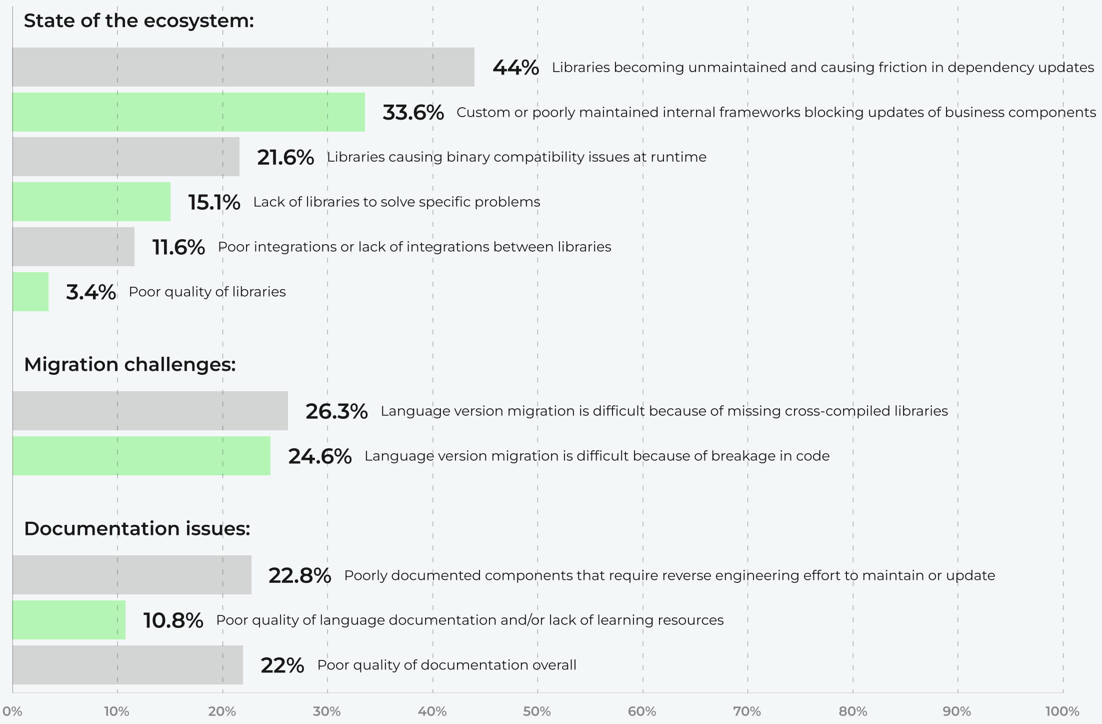
2 **Migration challenges:**

- Language version migration is difficult because of missing cross-compiled libraries: 26.3% of responses.
- Language version migration is difficult because of breakage in code: 24.6% of responses.

3 Documentation issues:

- Poorly documented components that require reverse engineering effort to maintain or update: 22.8% of responses.
- Poor quality of language documentation and/or lack of learning resources: 10.8% of responses.
- Poor quality of documentation overall: 22% of responses.

BIGGEST TIME SINKS AND HINDRANCES IN SCALA DEVELOPMENT



> *Custom responses (48 free-form inputs) further highlighted some other issues such as* ***tooling inefficiencies*** *(slow builds, poor IDE support, dependency management) and* ***language complexity*** *(learning curve, accidental complexity, migration struggles). These are exacerbated by* ***team and community challenges*** *(developer shortage, knowledge gaps, hostility between community groups) and* ***evolving requirements*** *in a business context.*

MAJOR BLOCKER LIBRARIES

Respondents reported a range of library-related issues, including lack of Scala 3 support, performance bottlenecks, and maintenance concerns. Key issues include:

1. **No Scala 3 support:**
   - spark, mongo-scala-driver, mockito-scala, chisel, scala-newtype, refined (partial support), scala-supertagged, better-monadic-for, twitter/scrooge, elastic4s (partial support), ficus*, monix, phantom, sangria-akka-http, play-swagger

   ** some of these libraries have been released for Scala 3 already*

2. **Missing/Unsupported macros:**
   - mockito-scala, refined, monocle, circe (annotations not supported), ZIO (no mockable annotations in Scala 3), tapir, scala-pact

3. **Library abandonment or maintenance issues:**
   - monix, phantom, sangria-akka-http, alpakka, slick, rediscala (or scala-redis), vertx-scala, cloudflow, scala-pact, awscala, unfiltered, kamon, deadbolt

4. **Performance issues:**
   - circe

5. **Breaking changes/Compatibility issues:**
   - tapir (frequent API changes, binary incompatibilities), cats-effect (breaking cross-version support), jackson-module-scala (binary compatibility problems, frequent CVEs), elastic4s (binary compatibility issues), scala-supertagged (runtime failures in Scala 2)

6. **Licensing concerns:**
   - akka (aggressive licensing changes, migration to Pekko required)

7. **Poor documentation or usability:**
   - ZIO (runtime breakages, insufficient documentation for some libraries), shapeless (lacking documentation), fs2-grpc (confusing design choices), ZIO Test (complex, difficult debugging for mocks) and ZIO Mock (hard to create mocks/expectations)

8. **Integration/Interoperability challenges:**
   - Swagger/Tapir integration with Play, ZIO interop with Cats (runtime breakages), Solr/OpenSearch/Elasticsearch clients (issues with ZIO-HTTP connections), scala-pact (poor Pact integration), fs2-grpc (integration challenges)

9 **Compilation and tooling issues:**

- Protocol Buffers libraries (issues with Alpine containers, incremental compilation), sbt (lack of BOM support, deprecated custom configuration scopes), scalafix (rules like `ExplicitResultTypes` and `UseNamedParameters` missing in Scala 3*), shapeless (no cross-compilation for Scala 3), avro4s (slow compilation, large class files**), quill (poor compile times)

** both those rules are now supported*
*** these avro4s issues were fixed recently*

10 **Ecosystem lag:**

- ZIO (slow maintenance for libraries like zio-schema, zio-kafka), cats-effect (delayed ecosystem updates for Scala 3), akka (delayed update to Scala 3 locked behind new license).

COMPILE TIMES

> Compile times remain a notable challenge for some teams:

Local development:

36.6% of respondents reported slow compile times impacting the developer iteration loop.

Continuous integration:

36.2% noted slow compile times in CI, affecting team productivity.

> Compile times are a subject of constant effort for the compiler team. New features like build pipelining are constantly improving this area while new library design patterns like sanely automatic derivation help to cut down inefficiencies introduced by metaprogramming-based libraries. Less complex builds can also leverage the bloop compile server which is capable of immense speed ups.

MAINTENANCE MODE PROJECTS

Around 43% of respondents indicated they have projects in maintenance-only mode. Common reasons include:

1. **Resource constraints:**
   - Not enough qualified developers or general resources to maintain or improve these projects.
   - Prioritization of other critical or high-impact projects.

2. **Stable and feature-complete:**
   - The projects reliably perform their intended functions and meet stable requirements.
   - They are considered “final software” or “feature-complete” with no pressing need for updates or new features.

3. **Lack of business value:**
   - No immediate business need for improvements or updates.
   - Business priorities have shifted, or the projects no longer align with the company’s strategic goals.
   - They are often low-visibility internal tools or services with limited usage.

4. **Changing contexts:**
   - Changes in business focus, such as a pivot to new approaches, have deprioritized older systems.
   - The projects may be tied to legacy technology that the company is phasing out.

5. **Low maintenance needs:**
   - Projects are stable, resilient, and scalable, requiring minimal upkeep.
   - They “just work” and meet their original objectives without ongoing intervention.

6. **Legacy considerations:**
   - Some projects are tied to older, deprecated technology stacks but must remain operational (e.g., for field devices or contractual obligations).
   - Efforts to replace them have been deprioritized due to resource or technical challenges.

7. **Developer turnover:**
   - Departure of key champions for these projects has left them without strong advocacy.
   - New developers may lack the skills or preference for maintaining legacy codebases.

> *A common theme for these projects seems to be that they are either considered legacy from a business perspective or that they "just work" and require minimal effort to maintain. The most pressing issues mentioned relate to the availability of developers capable of working on older code without a lengthy onboarding process. This aligns with previous insights regarding the fractured history of styles and approaches within the Scala ecosystem.*

Scala commercial support

> Rely on our expert support for the compiler and tooling to keep your Scala projects running smoothly. Our commercial support gives you direct access to industry hands-on experts who troubleshoot, optimize, and enhance your developer productivity.

Build tools:

We support and improve Bazel, SBT, and Scala CLI to ensure fast, efficient, and reliable builds.

IDE tools:

We develop and maintain VS Code Metals while also supporting IntelliJ IDEA with plugin development and integrations.

AI-powered tooling:

We have experience with Model Context Protocol-capable tools like Cursor and Cline, helping you integrate AI-driven development workflows.

DevOps tools:

We provide expert support for Pulumi's Scala SDK to ensure smooth infrastructure automation.

Scala compiler and core tools:

As Scala 3 core contributors, we offer direct expertise in core Scala tooling like Scaladoc, compiler enhancements, and other core tooling, giving you the stability and performance you need.

Address codebase complexity:

With our experience delivering solutions to a wide range of industry challenges, we maximize Scala's capabilities to deliver your project efficiently.

> Get in touch with us today, and **we will bring your projects back on track.**

Get in touch with us

Hiring and training issues

LACK OF ENGINEERS

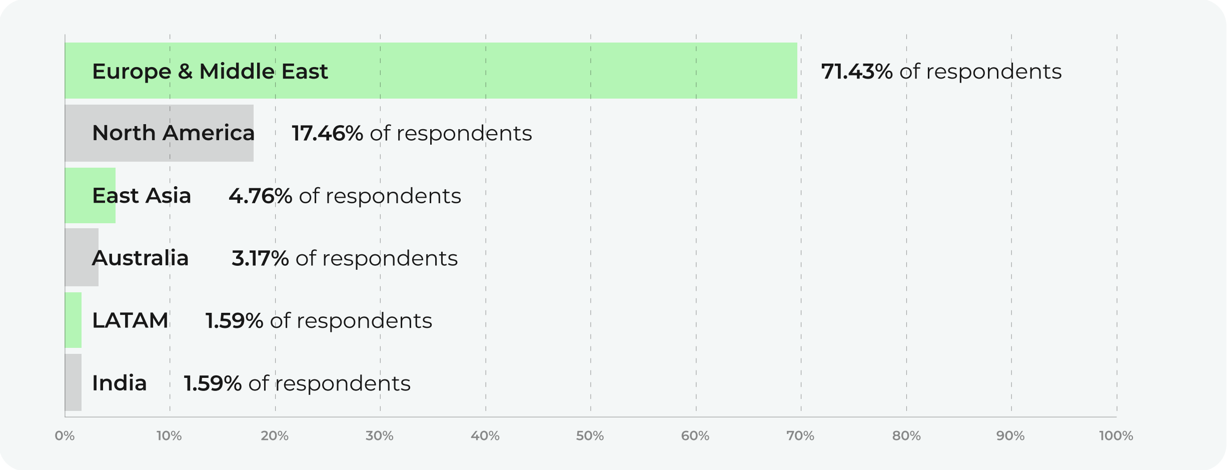
The lack of suitable engineers was identified as the primary challenge by 56.5% of respondents, with an additional 8.2% noting that engineers are too expensive. Further insights were provided by 13.8% of respondents:

- **High costs** of experienced Scala developers make hiring prohibitive for some organisations.
- **Difficulty** finding developers with Scala-specific expertise forces many organisations to rely on training Java developers or juniors.
- Scala's efficiency can reduce the need for larger teams, which might partially explain **reduced hiring** in some organisations.
- Some respondents perceive Scala's **focus on technology** over user value as a drawback.

This situation leads to another significant issue – losing Scala hires poses a significant risk for 60.3% of surveyed organizations. Together, these two factors are the largest issue highlighted by the survey when measured by the size of consensus among respondents.

HIRING MARKETS

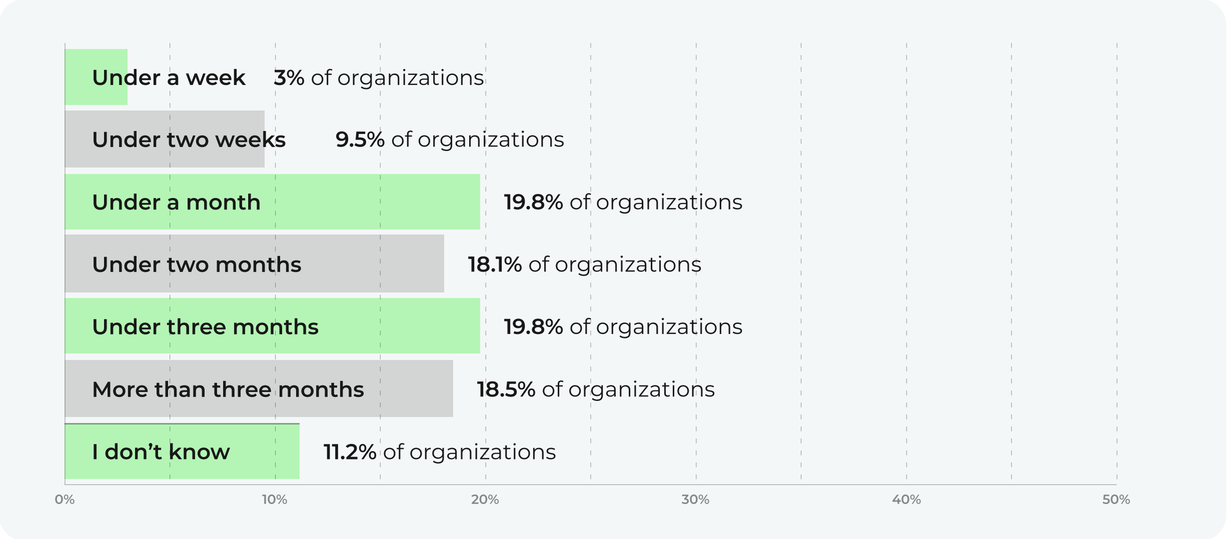
The geographic distribution of hiring markets shows a strong concentration in Europe, but this is likely an artifact of the survey's reach, as it appears to have limited penetration in regions such as the US, LATAM, or India. Additionally, it is worth noting that this question was optional, and only 30% of respondents provided an answer. These are the markets on which respondents' organizations hire:



Beside this geographic hiring distribution, a significant **68.6% of projects have the capability to hire for remote roles.**

TRAINING AND TIME TO PRODUCTIVITY

A majority (57.8%) of organizations train developers internally. Respondents reported varying times for developers to become proficient and productive with Scala:



Notably, ***40.5% of respondents believe that using Scala slows down the onboarding process*** *for new engineers. This highlights the need for initiatives to reduce the learning curve for most projects, enable newcomers to grasp advanced language features more quickly, and provide clear guidance on adopting patterns and understanding their trade-offs. Proposals of such initiatives can be found in the last section of the report.*

Scala consulting services

Use the expertise of core Scala maintainers:

Collaborate directly with the core team behind Scala 3 to ensure unparalleled insight and proficiency in your projects.

Optimize your delivery processes:

Our deep understanding of Scala enables us to refine your development processes, leading to faster and more reliable outcomes.

Address codebase complexity:

With our experience delivering solutions to a wide range of industry challenges, we maximize Scala's capabilities to deliver your project efficiently.

Tailored training and support:

We offer customized training programs and ongoing support to ensure your team is equipped to develop smoothly with Scala.

Visit our **Scala consulting** page and learn more about how to elevate your projects.

Go to offer

Future perspectives

CONSIDERATIONS OF MOVING AWAY FROM SCALA

A total of 23.2% of organizations reported considering moving away from Scala, within this group the reasons are distributed as follows:

1. **Hiring challenges:**
   11.6% of organizations cite difficulty finding skilled Scala engineers and the general scarcity of developers as a primary reason.

2. **Engineering issues:**
   5.6% of organizations are considering the move due to technical challenges such as high-cost burden from Scala's complexity, compiler, and library incompatibilities, lack of commercial focus and resulting tooling issues with IDEs and sbt.

3. **Specific reasons (6%):**
   - **Organizational alignment:** Many organizations prefer languages like Java, Python, or Go, driven by developer familiarity, existing infrastructure, and the desire for a unified tech stack.
   - **Ecosystem and perception:** Scala is sometimes viewed as a small and unstable community with waning relevance in fields like data engineering. Past misuses, such as creating overly complex codebases, have contributed to lingering biases against the language.
   - **Preferred alternatives:** organizations often turn to languages like Node.js for web development, Python for generative AI, or other languages offering superior tooling and broader community support.

SCALA'S POPULARITY FOR NEW PROJECTS

Despite these challenges, 88.4% of respondents indicated they would still consider Scala for new projects within their organizations. The remaining 11.6% of respondents provided the following reasons for not choosing Scala in the future:

1. **Hiring challenges:**
   Difficulty finding experienced Scala developers in certain regions or at competitive salaries. Developers are also migrating to other languages like Kotlin or Go.

2. **Complexity and ecosystem:**
Scala’s perceived complexity, compile times, JVM resource usage, and fragmented library ecosystem contribute to hesitation in adopting it for new projects.

3. **Management and organizational Policies:**
Scala has lost some trust among higher management due to issues like the Scala 3 rollout. Many organizations now favor other languages like Java, Kotlin, Go, or Rust.

4. **Tooling and stability:**
Concerns about IntelliJ IDEA’s support for Scala 3 and instability in the newer version make some hesitant to adopt it.

5. **Shift in language perception:**
Scala’s declining popularity and focus on niche features over commercial needs have reduced its appeal for broader adoption.

6. **Cost vs. benefit:**
Scala’s expressive type system is often seen as outweighed by higher maintenance costs and complexity when compared to alternatives like Kotlin, Rust or newer Java features.

7. **Industry-specific misfit:**
In certain domains, such as embedded systems, Scala is less suitable, leading organizations to prefer other languages.

SCALA AS THE NEW COBOL?

> In a lighter final segment of the survey, we asked respondents whether they saw Scala becoming the “new COBOL” in their organizations with interpretation of what that means left to the respondents. Encouragingly, 80.4% of respondents disagreed with this notion. However, the remaining 19.6% who expressed concerns signal a need to address long-term relevance and growth issues to avoid such perceptions.

Key challenges

A strong majority of respondents (over 90%) remain satisfied with Scala, praising its expressive power, solid JVM heritage, and the ability to write concise yet robust code. Many teams acknowledge that once a developer becomes proficient, Scala helps reduce the overall engineer count needed for complex work due to its higher-level abstractions. Over 88% of respondents would still choose Scala for new projects, reflecting a lasting confidence in its long-term viability and underscoring the language's ongoing appeal for both greenfield initiatives and critical production systems. Having said that - the survey highlighted key pressing issues listed below in order of importance:

1. **Engineer scarcity:**
   Scala adopters are looking almost invariably for experienced Scala developers to work in teams filled mostly with other experienced engineers. There's a distinct lack of junior positions available, most probably caused by the fact that many projects are leveraged on high-tech solutions (distributed systems, type-level programming, pure functional programming) and the time necessary for a new developer to be onboarded into the tech stack of a project.

2. **Ecosystem fragmentation and library maintenance problems:**
   Scala's overall ecosystem is suffering due to the unfortunate talent split between competing ecosystems. This situation exacerbates problems caused by the already relatively small size of the Scala community - some libraries targeting niche problems exist only in some of the ecosystems, some general libraries do not include an integration layer for some ecosystems. Library abandonment is even more painful in this situation as usually, Scala libraries have complex builds to allow cross-compilation and publication of integration sub-libraries.

3. **Style differences and complexity budgets:**
   Scala is a very powerful, elastic language and while this brings a lot of value to experienced teams, it also makes it easier to go over the complexity budget and therefore harder for newcomers to onboard into a project, even if they are already familiar with Scala.

4. **Scala 3 migration and tooling pain points:**
   The relatively slow adoption rate of Scala 3 in enterprise settings signals that technical and tooling-related issues related to migration (macros, library ecosystem issues, IDE support) are a significant burden for organizations that adopted Scala. These problems seem to outweigh the perceived benefits of using Scala 3. On the other hand, it should be noted that enterprises are often sluggish to adopt new versions of technologies, even when said technologies have met their official end-of-life deadlines and updates are not considered to require a lot of effort. The best example here would be the sluggish adoption of newer JVM versions as Java 8 is still used in almost 29% of all production applications in 2024.

Solution proposals

ADDRESSING ENGINEER SCARCITY

1. **Onboarding experience:**
VirtusLab will continue its work on improvement of the onboarding experience with Scala-CLI and the official Scala language command along with efforts related to Scala Toolkit. The goal of this initiative is to make Scala as approachable to the newcomers as possible.

2. **Leveraging the experience of established companies and tutors:**
Organizations such as VirtusLab have successfully implemented methods to train and grow Scala developers and established relationships with academic communities to start the Scala training early. The Scala community should leverage the expertise of Scala-centric companies to create scalable training frameworks, workshops, and onboarding methodologies to address the scarcity of experienced Scala engineers. VirtusLab offers opportunities to bolster companies' ranks with mixed experience levels but supervised teams, lowering the risk profile of employee loss while simultaneously providing the ability to quickly up-skill new junior hires. Scala already has solid materials for learning the basics both in the form of documentation and online courses by Martin Odersky and Scala Center. Simultaneously, tutors like Rock the JVM have created incredibly rich offerings of courses and learning materials that enable quick training and tackling of the learning curve.

ADDRESSING ECOSYSTEM FRAGMENTATION AND LIBRARY MAINTENANCE PROBLEMS

1. **Core libraries and library standards:**
To address ecosystem fragmentation, a standard for library development should be established. This standard will outline necessary capabilities for libraries, such as:

- multi-platform support (where feasible)
- support and/or integration with all major Scala ecosystems along with direct style
- searchable, runnable documentation with compile-verified snippets.

A curated list of libraries that meet this standard will be created, and an upgrade path will be prepared for best-candidate libraries that do not currently comply in domains where there's no library that would meet the standard. The goal here is to have at least one high-quality library for every common use case that can be used by any member of any community. Additionally, a continually maintained library template will simplify compliance, providing pre-configured CI setups, documentation scaffolding, and multi-platform publishing tools. These measures will ensure a consistent, high-quality developer experience across the Scala ecosystem. VirtusLab experts will also provide guidance regarding the design of libraries in a way that makes them truly usable with all Scala ecosystems.

2. **Coordinate and increase efforts around the ecosystem and core libraries:** The Scala ecosystem mostly follows a decentralized model, with some more prominent organizations like Typelevel or ZIO. On the other hand, users need a coherent and stable set of libraries and tools. Following the core libraries proposal, VirtusLab will establish a way for Scala developers to prioritize their needs across the entire ecosystem and allocate resources where they are most needed. VirtusLab teams are primarily considering extending Scaladex to facilitate voting, communicate priorities, and visualize issues across the ecosystem. The actual solution will be developed and communicated in the coming months.

ADDRESSING STYLE DIFFERENCES AND COMPLEXITY BUDGETS

An **official best-practice guide** for Scala will be created, emphasizing when to use advanced patterns and outlining the trade-offs associated with them. This guide will include recommendations for specific ecosystems and feature a dual configuration of `scalafmt` and `scalafix` to detect and manage deviations from declared complexity budgets.

By providing clear guidelines and tools for enforcement, this initiative will help teams maintain consistency and manage complexity effectively. Scala ecosystem maintainers are encouraged to collaborate on this effort and to produce best-practice guides for their respective ecosystems so that in-ecosystem consistency can be also achieved using this centralized know-how store.

ADDRESSING SCALA 3 MIGRATION AND TOOLING PAIN POINTS

1. **Improved IDE support:**
The Core Scala Team has already tightened collaboration with IDE providers, and we will keep up with changes to enhance the development experience, improve release synchronization and address new feature support positioning in roadmaps.

2. **Free Scala 3 migration consultation:**
VirtusLab offers a free Scala 3 migration consultation service that can help companies by leveraging the skills of Scala 3 experts and compiler team members to solve the most difficult problems introduced by migration. The compiler team has a very good track record of enabling migrations of Scala codebases for organizations of any size.

3. **Migration-related articles and guides:**
VirtusLab experts have consulted several migrations of large enterprise projects and gathered a significant amount of experience in regard to troublesome parts of the process. This experience will be shared with the larger community in the form of long-form articles that will help tackle the difficulties during the migration and make the process more effortless for companies using Scala.

About VirtusLab

VirtusLab, established in 2010 and headquartered in Poland, is a global software technology company specializing in Scala consulting, ML engineering, developer productivity, and custom software solutions. By actively contributing to open-source communities, including Scala, the company drives innovation, enhances developer experience, and collaborates with enterprises in logistics, insurance, manufacturing, and retail.

Let's connect

Contact Details

info@virtuslab.com

POLAND

Kraków Headquarters

Virtus Lab Sp. z o.o.
ul. Szlak 49
31-153 Kraków

GERMANY

Berlin Office

+49 30 52014256

VirtusLab GmbH
Potsdamer Platz 10
10785 Berlin

UNITED KINGDOM

London Office

+44 (0)20 4577 1051

Virtuslab Ltd.
40 Bank Street HQ3
London E14 5NR