Software Improvement Group

THROUGH THE SIG LOOKING GLASS

Benchmark Report | 2023

softwareimprovementgroup.com

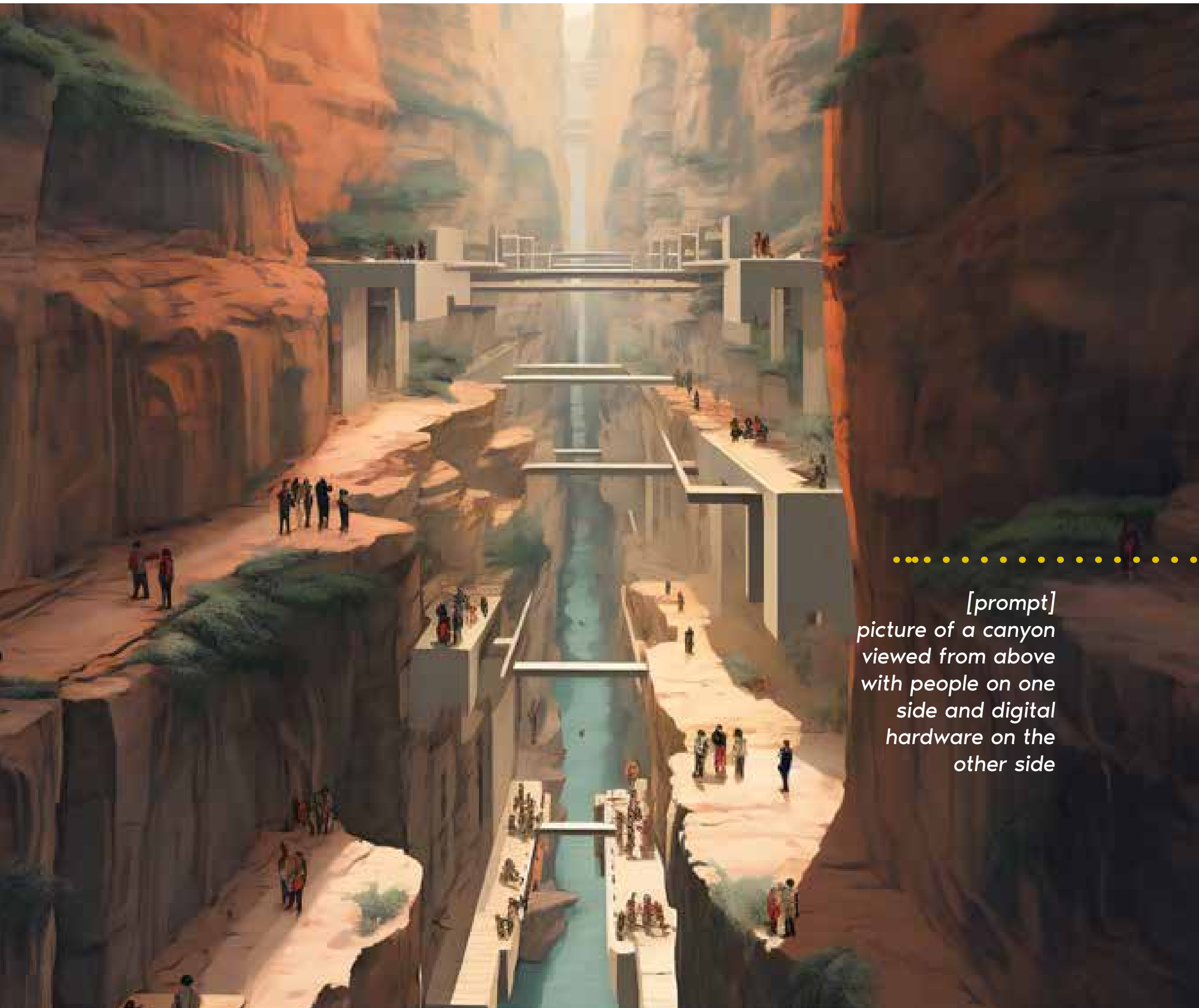
THROUGH THE SIG LOOKING GLASS

Benchmark Report | 2023

Luc Brandts

FOREWORD

Here it is, our fifth SIG Benchmark Report on the state of technology. We started this tradition five years ago, producing fascinating results for the digital community annually. This year is no different. We have great things to share with you, with thorough data analysis behind it to substantiate our findings. I would like to thank the entire team who worked on this great piece of work. I hope you enjoy reading it as much as we did creating it.



*[prompt]
picture of a canyon viewed from above with people on one side and digital hardware on the other side*

But first and foremost, digest the results, and put them into action. Some of our findings are quite positive and can be seen as a compliment to the digital world: we do see that, on average, the build quality is rising. However, in all honesty, most of our findings are less favorable and should require your immediate action. Read the section of your choice or review the entire report: every section contains actionable findings.

There are many things to list here, but some really stand out. In no particular order, here are a few things you can learn more about in the report:

- Please take care of your architecture when using low-code: this is getting messy, and action is required from the low-code world.
- Then, our new architecture model that integrates technical analysis with people-aspect will show you how to double your speed of innovation, how you can get rid of cloud impediments, and so on.
- Next, if you're in AI: you're doing wonderful work, and the world is in awe of you all. But please remember it is still software you're creating, and what is being built is significantly worse than we would expect. That's a pity, as AI is surely not a temporary hype. It's here to stay, and so is your software: in other words, you're creating tomorrow's legacy.
- Then, the use of open source. Our 2022 SIG Benchmark Report already rang the alarm bells, but we're adding a loud siren this year. There is a lot to be said on the subject, but, for one thing, the disconnect between business and IT priorities is showing very clearly here: business critical systems get the same (i.e. quite bad) attention as lesser important systems. Let's work together to close the open back door that open source is.
- To conclude, I would like to draw your attention to our first skill gap benchmark report. Having assessed over 5,500 people in more than 180 countries, we now have a very good view of where the digital world is with their skills. Our conclusion is: not so good. We observe a skill gap of more than 50-60% for all (!) of the 41 digital skills, demonstrating a need for upskilling.

As you can see, there is so much to share, so please read our report and let us know what you think.

Luc Brandts, Group CEO

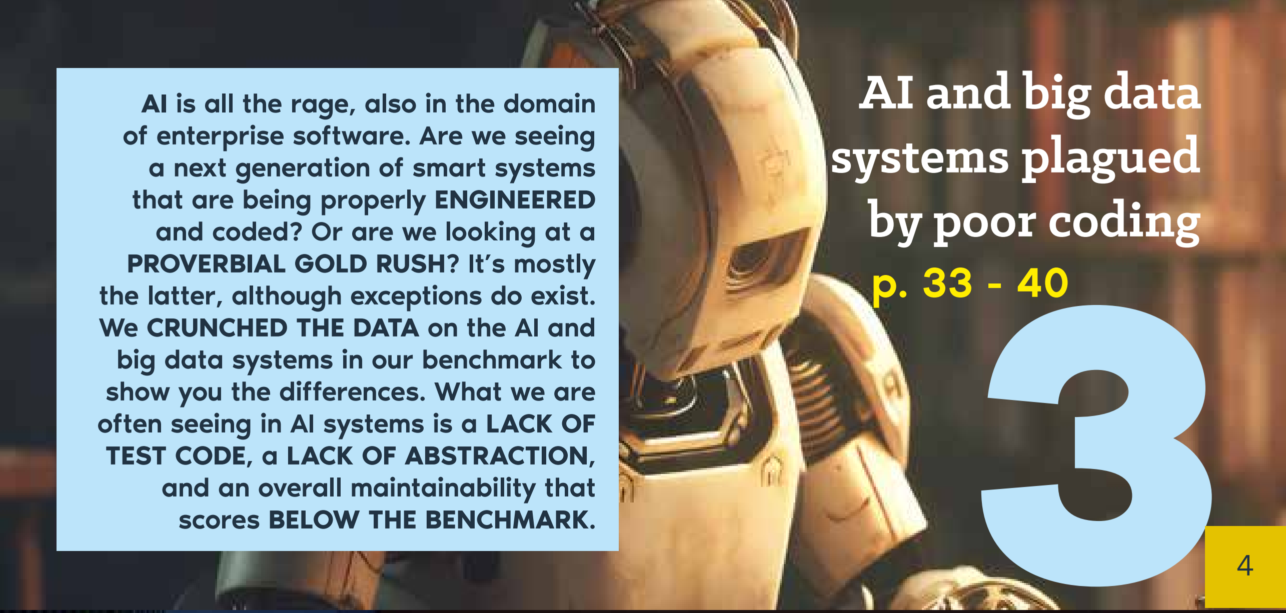
CONTENT

The annual SIG Benchmark Report looks at the enterprise software industry through the SIG looking glass. SIG measures software build quality on a global scale with an ever increasing range of analysis capabilities. The 2023 edition spans 14 years of measurements across more than 12,000 enterprise software systems.

As we celebrate the 5th anniversary of our report, we are proud to present 5 diverse chapters with our latest data insights and calls to action.

SOFTWARE BUILD QUALITY is an essential data point in enterprise software **DECISION-MAKING.** Without it, you are essentially flying blind. Are you aware of how well the quality of your software portfolio stacks up against **PEERS**? Or others using similar **TECHNOLOGY STACKS**? Are the **GROWTH AND CHANGE** patterns of your applications in line with their expected life cycles?

Did you know that great software **ARCHITECTURE** needs a great alignment with the **ORGANIZATIONAL** and **SOCIAL** aspects of your teams? As well as a solid design on top of the right technological choices? With **OUR NEW MODEL**, it is possible to **MEASURE AND CONTROL** these aspects. In terms of resolution speed, good architecture quality can mean a factor of **TWO TIMES FASTER** than poor quality.

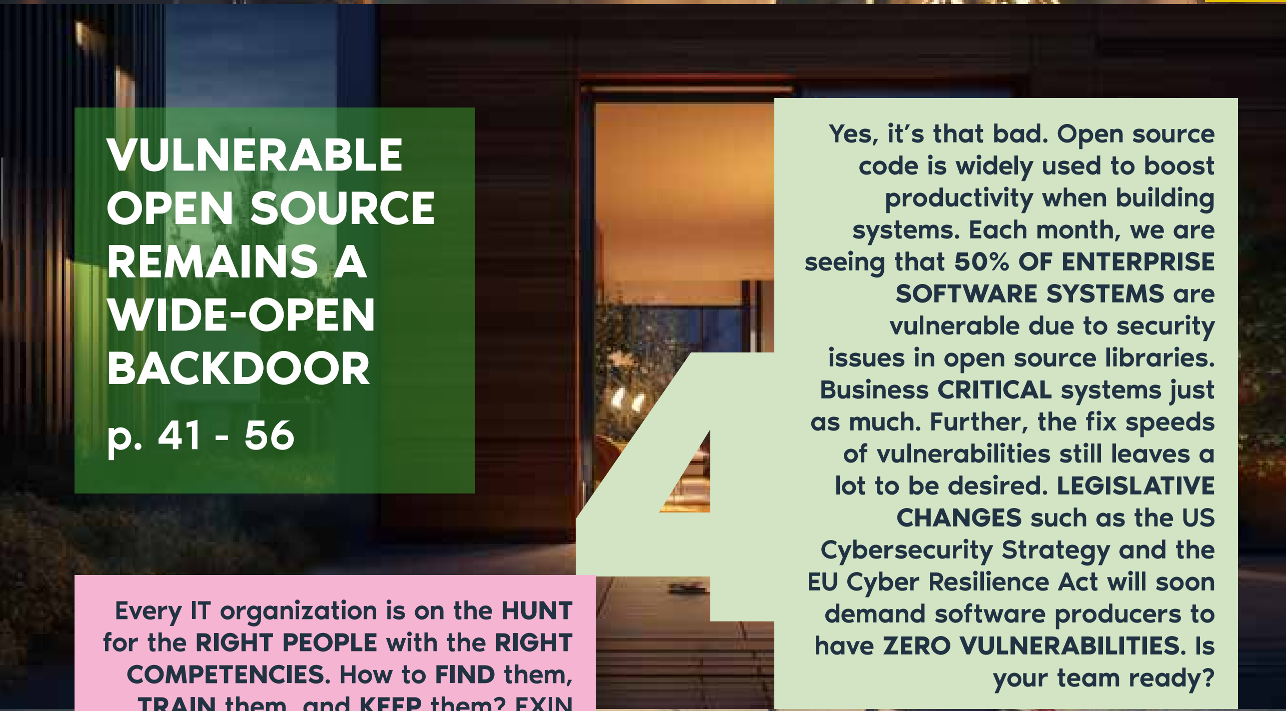


AI and big data systems plagued by poor coding

p. 33 - 40

3

AI is all the rage, also in the domain of enterprise software. Are we seeing a next generation of smart systems that are being properly **ENGINEERED** and coded? Or are we looking at a **PROVERBIAL GOLD RUSH**? It's mostly the latter, although exceptions do exist. We **CRUNCHED THE DATA** on the AI and big data systems in our benchmark to show you the differences. What we are often seeing in AI systems is a **LACK OF TEST CODE**, a **LACK OF ABSTRACTION**, and an overall maintainability that scores **BELOW THE BENCHMARK**.



VULNERABLE OPEN SOURCE REMAINS A WIDE-OPEN BACKDOOR

p. 41 - 56

4

Yes, it's that bad. Open source code is widely used to boost productivity when building systems. Each month, we are seeing that **50% OF ENTERPRISE SOFTWARE SYSTEMS** are vulnerable due to security issues in open source libraries. Business **CRITICAL** systems just as much. Further, the fix speeds of vulnerabilities still leaves a lot to be desired. **LEGISLATIVE CHANGES** such as the US Cybersecurity Strategy and the EU Cyber Resilience Act will soon demand software producers to have **ZERO VULNERABILITIES**. Is your team ready?

First digital skills benchmark shows poor job alignment

p. 57 - 64

5

Every IT organization is on the **HUNT** for the **RIGHT PEOPLE** with the **RIGHT COMPETENCIES**. How to **FIND** them, **TRAIN** them, and **KEEP** them? EXIN and SIG are **ASSESSING** thousands of IT professionals globally to find out what they are good at. Surprisingly, many are in positions where they need **UPSKILLING TO COMPETE** better in their current roles. At the same time, **DEMANDING POSITIONS** such as enterprise architecture and leadership roles are truly becoming **SKILL HUBS**. Those few sought-after professionals are in a great position to move to their **NEXT ROLES** and the challenge will be to keep them on.

1

Magiel Bruntink / Pepijn van de Kamp / Benedetta Lavarone

SOFTWARE BUILD QUALITY

Major differentiator between industries and technologies

In order to fully understand the risks of a software system, it is not enough to look at the software from the outside. You really need to look at all the code, only then a full understanding is possible. Seeing a demo, using the software, or trying to break in from the outside will show less than 10% of potential trouble.

[prompt] a photographic picture of a world in the future as a mix of happy people working in a modern office doing digital work and industrialization elements like heavy machines in light colors blue and white and green and yellow

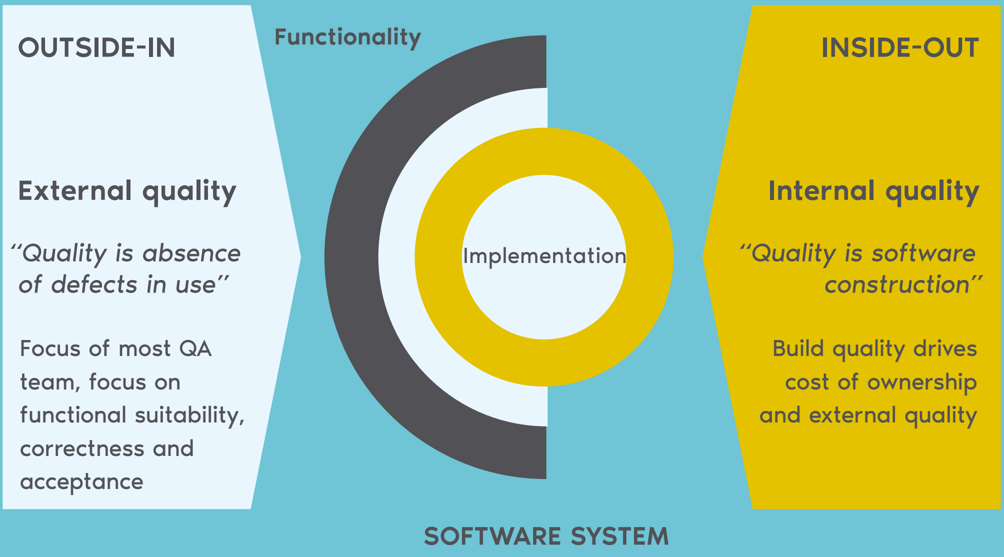
This chapter will start by explaining a few concepts and measurements that SIG employs in its software assurance business. You will find the following topics in the next sections:

1. Looking at enterprise software from both inside and outside: SIG's model of build quality and its global measurement across 12,000 enterprise software systems.

2. Software build quality across industries and technology stacks: The yearly recurring rankings of build quality in the top 10 industry sectors and top 10 software technology stacks.

3. Software life-cycle impacts growth and change estimation by an order of magnitude: Our novel analysis of the growth and change rates of software systems depends on their life cycle phase.

LOOKING AT ENTERPRISE SOFTWARE FROM BOTH INSIDE AND OUTSIDE



OUTSIDE-IN

External quality

"Quality is absence of defects in use"

Focus of most QA team, focus on functional suitability, correctness and acceptance

INSIDE-OUT

Internal quality

"Quality is software construction"

Build quality drives cost of ownership and external quality

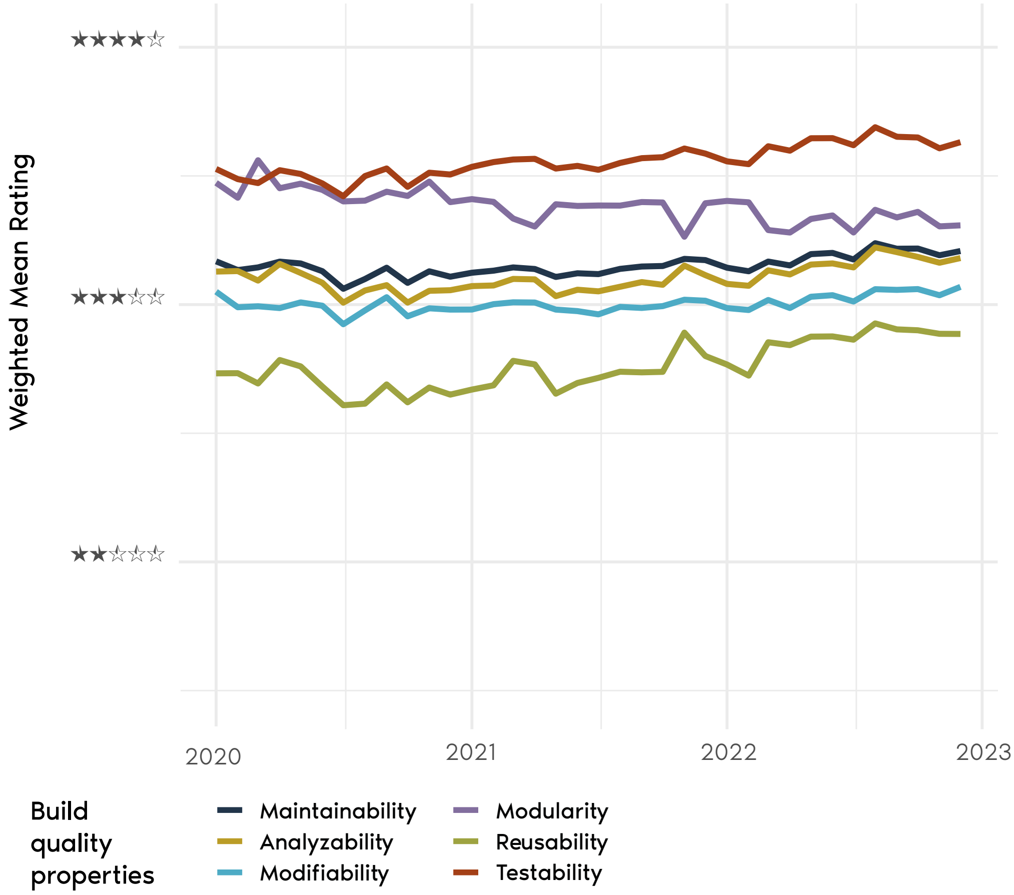
A core element of SIG's software assurance is the measurement of maintainability, an aspect of software quality as defined by ISO/IEC 25010:2011. Maintainability is a major factor in keeping software-related costs low, and business agility high. With Sigrid, SIG has measured maintainability and its underlying metrics for 12,000 of our client's software systems over the years.

Overall, we are seeing a continuing trend of *gradual improvement of the main build quality* properties. Each year we re-calibrate our maintainability measurement model to include the latest data and to conform to ISO/IEC 17025 standard for test and calibration laboratories. The latest model is then applied to our historical data set to have a normalized view of the past year's performance.

Globally speaking, a gradual upwards trend build quality across the enterprise software domain means that our measurement models become more strict over time as well. There is a need to continuously improve the software in order to compete with the market, both in functionality and maintainability.

GLOBAL BUILD QUALITY AS MEASURED BY SIG

Including 100K person-years worth of enterprise software



[prompt] a photographic picture of a world in the future as a mix of happy people working in a modern office doing digital work and industrialization elements like heavy machines in light colors blue and white and green and yellow

Compared to previous years' benchmark reports, this year we are reporting with updated architectural metrics to give a better reflection of architectural quality in modern software systems[1]. Underlying the Modularity property, the aging Component Balance metric was replaced with Component Entanglement. Component Entanglement measures to what extent components are adhering to good layering and code dependency patterns.

Looking back at the past three years, we observe that the average Modularity rating (purple line) is in gradual decline, in contrast to the other build quality properties. The implication is that the architecture aspect of build quality needs more attention to prevent further decline. For this reason, SIG created a new Architecture Quality model: the first outcomes of that are presented in the next chapter.

As our dataset at this point contains measurements from all kinds of enterprise software technology, from over 300 different technologies, ranging from Cobol to Java and C#, to Python and JavaScript, in the following we zoom in further to point out the areas of concern more specifically.

[1] https://www.softwareimprovementgroup.com/software-analysis

2. Software build quality across industries and technology stacks

Software industry sectors

Let's start by slicing the SIG Build Quality benchmark by software industry sectors. Most enterprise software producers have a clearly defined target industry like Banking or Retail, for instance. We further include Government as a broad category of different kinds of governmental and regulatory responsibilities. Furthermore, the category Software & Computer Services includes clients that are active across many different industries or are focussed on clientele in the software industry itself.

Our 2023 top 10 ranking of software industry sectors can be seen in the following table. The Delta column indicates ranking changes compared to 2022. Overall, the ranking remains rather stable, with a position swap between first and second place, where Energy, Oil & Gas companies are again leading the pack. Government systems gain a place at the expense of Insurance. This year's newcomer Health Care enters at position #8, slightly below the market average of 3.0 stars.

In 2020 we published our industry sector ranking for the first time. Back then, the margin between #1 and #10 was about .51 stars, while we are now looking at a difference of .64 stars. The top sectors are gaining, rather than the lower performance losing stars. The rates at which legacy technologies can be phased out play a major role in these trends. Trailing industries should therefore increase their actions toward modernization to avoid being disrupted by newcomers.

#	Delta	Top 10 Software industry sectors 2020-2022	Score
1	↑	Energy, Oil & Gas	3.40
2	↓	Industrial Transportation	3.34
3		Banking	3.33
4	↑	Government	3.25
5	↓	Insurance	3.22
6		Financial Services	3.05
7		Software & Computer Services	2.98
8	new	Health Care	2.92
9	↑	Telecommunications	2.92
10	↓	Retail	2.76

- Scores range between 0.5 and 5.5 stars in the SIG Maintainability Model, calculated as mean maintainability weighted by systems' volume, for each system's most recent measurement.
- Industry sectors or technology stacks have at least 50 systems across at least 10 clients.
- The table only shows the top 10 ranked industry sectors.
- For industry sectors, the deltas are rank position changes since the 2022 Benchmark Report.

Enterprise software technology stacks

Compared to previous editions, we thoroughly revised the technology stack categories to better align with modern developments like systems built-in web technology exclusively. In general, we classify a system by looking at its most dominant programming language. The major changes compared to last year are:

- Introduction of the Web/Templating category for systems built mostly in web technologies like JavaScript or TypeScript for instance.
- The mainstream languages Java and C# are now grouped together with other modern general-purpose languages like Python, Go, and Kotlin.
- System programming languages, like C and C++, were moved from Legacy 3GL/4GL to a new Embedded/System category.
- Database language, including SQL dialects, were moved from Legacy 3GL/4GL to a new Database category.

These changes make a comparison of the ranking with last year less meaningful. We observe that the Low Code category is no longer the leader of the top 10, which it has been since 2020. The new leaders are the front-end focused category Web/Templating and the existing BPM/Middleware categories.

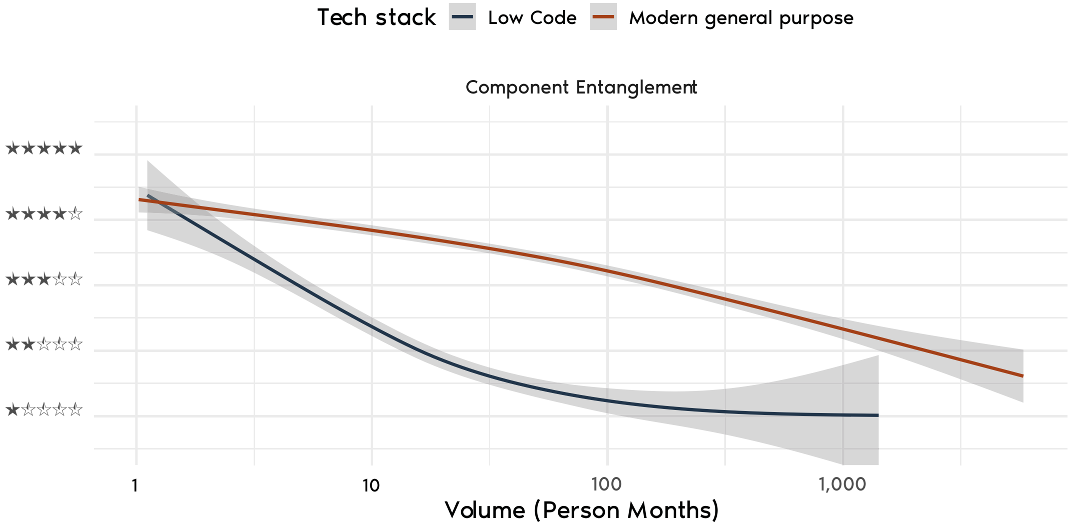
Systems built only in BPM/Middleware techs are typically just a few person-months in size. That fact makes the category adaptable and able to climb the ranking relatively quickly (up from #6 last year).

#	Top 10 Software technology stacks 2020–2022	Score
1	Web/Templating	3.40
2	BPM/Middleware	3.33
3	Low Code	3.22
4	Modern general purpose	3.18
5	Configuration	2.94
6	Scripting	2.84
7	Embedded/System	2.73
8	Packaged Solution Customization	2.57
9	Legacy 3GL/4GL	2.45
10	Database	2.45

Of course, these rankings are mostly indicative. An actual choice of technology stack depends on many factors depending on application type, organization, and other factors. Within each industry sector there can also be high- and low performers that may not be reflected by the averages shown here.

For the Low Code category, we want to share the following additional analysis. Looking at Component Entanglement, one of the underlying metrics for architectural quality, we see a stark difference with competing modern general-purpose languages. Component Entanglement is a measure of overall layering and adherence to good architectural principles.

As the graph shows, there is a growing difference between Low Code and Modern General Purpose languages. The vertical axis is the SIG rating scale from 1 to 5 stars (higher is better), while the horizontal axis shows the volume of systems. As system volume goes up, on average, Low Code systems quickly drop their Component Entanglement score. This indicates a stark deterioration of architectural quality, and thus future maintainability of Low Code systems as they grow larger.



Key Finding: Low code systems are more entangled, especially at larger system sizes. This makes them harder to maintain, compared to systems built in modern general-purpose languages.

Good architectural guidelines induce a clear component layering and allowable dependency structure. Specifically, cyclic or layer-skipping dependency patterns should be avoided. In Low Code systems, developers are not always aware of or equipped to adhere to such guidelines. Furthermore, tool support to visualize and enforce architectural best practices is not yet in place on every Low Code platform.

When adopting Low Code platforms, it is strongly recommended to follow up on the following points to prevent the creation of a new generation of legacy software:

- Utilize guidelines and tooling for architectural design also in Low Code, such as those available in the Sigrid platform.
- Make Low Code developers aware of the future growth of their applications and ask them to prepare their design accordingly.
- Ensure that Low Code developers have sufficient knowledge and expertise in architectural design.

3. Software life-cycle impacts growth and change estimation by an order of magnitude

Enterprise software needs to evolve in order to stay relevant in the context in which it is used. Life-cycle management of software systems is a core portfolio governance practice that aims at:

- budgeting,
- planning and tracking evolution,
- maintenance, including security,
- modernization activities.

Without performing such activities, the durability of the software is impacted, and code bases fall into the next life cycle phase (regardless of whether this is desirable or not).

Key Findings:

- **Software systems in different life-cycle phases have very different growth and change characteristics, impacting estimation by an order of magnitude.**
- **While systems are in Evolution, they typically grow at 10% per year and have a change rate of 47% per year.**
- **In Maintenance, the growth of a code base stagnates while existing code is still changed at a typical rate of 15% per year.**

Software lifecycle management within organizations with large software portfolios is hard to do efficiently. *We recommend tracking high-level KPIs to indicate whether the expected maintenance activities take place according to the expected life-cycle phase.*

Over the past years, we collected a dataset on about 500 enterprise software systems, tracking their growth and life cycle phases. The dataset spans the years 2020 to 2022, amounting to *approximately 1,000 years of software evolution and maintenance.*

Before diving in, we need to define the typical life-cycle phases of enterprise software systems:

- **Initial development**
  The Initial Development phase starts with the first code being written and typically ends when the software is considered both stable and feature-rich enough to be rolled out to the full target group of users. In this phase, the software is typically written by one or more dedicated development teams. In the initial development phase, the rapid growth of new code volume and a large amount of changes to the existing code is expected.

- **Evolution**
  After going into production, typically evolutionary activities take place: addressing feedback from users on existing features, adding more features to the software, and working on non-functional aspects of the software (e.g. increasing scalability) as the user base grows. The foundation of the software is now in place. In this phase, the software is typically under development by one or more dedicated teams.

- **Maintenance**
  In the Maintenance phase, the code base is typically brought under the responsibility of a development team that maintains multiple code bases (no dedicated team). In this phase, the ability of this team to make changes to the existing software depends on the degree to which knowledge of the code base is still available in the team, as well as the quality of the code base, documentation, and integrity of the architecture. Typical activities that are performed in this phase are handling small change requests, bug fixes, and keeping the underlying software libraries, frameworks, and other infrastructure components up to date.

- **Decommissioning**
  In this phase, it is time to execute change activities that are needed for the sun set of the software system. Functionality and users are migrated to other systems. No major changes are typically made to the code base at this stage, other than changes that are needed for phasing out specific functionality or to keep the software in a safe and secure state (e.g. patching security vulnerabilities).

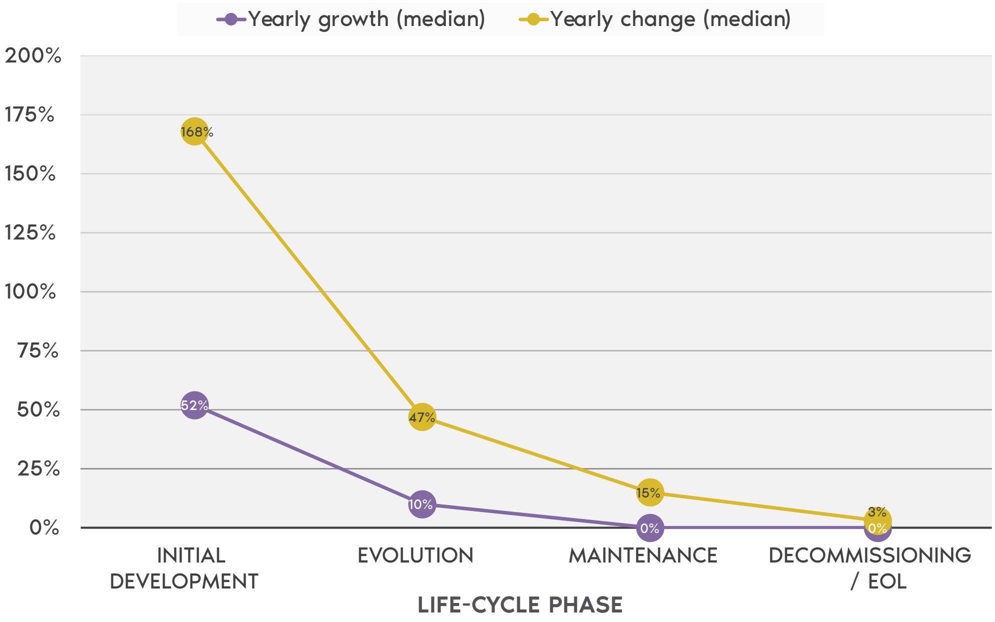
- **End of life**
  In this phase, the software system is switched off and no more changes are made to the software. The code base and related artifacts are safely archived.

Let's have a look at our data on 2 high-level KPIs that can help with tracking evolution and maintenance activities:

- **Yearly growth rate:** A percentage that indicates the yearly growth of code volume.
- **Yearly change rate:** A percentage that indicates the amount of changes done to existing code.

YEARLY GROWTH AND CHANGE RATE OF ENTERPRISE SOFTWARE SYSTEMS PER LIFE-CYCLE PHASE

Based on analysis of code changes to 500 enterprise software systems between 2020 and 2022



The graph above shows that software systems in different life cycle phases have very different growth and change characteristics.

- **Initial Development Phase:** As expected, significant growth (52%) and change (168%) occur during this phase, with a large variance in the data. Code bases can double, triple, or even quadruple in code volume on a yearly basis as new code is added and existing code is modified.
- **Evolution Phase:** Growth of systems in this phase now stabilizes to about 10% per year, while existing code still changes at a significant rate of 47% per year. This indicates that many modifications to existing functionality are made in this phase while still adding new features.
- **Maintenance Phase:** In this phase, typically 15% of the existing code is changed on a yearly basis, while the growth in terms of code volume stagnates (with a median of 0% yearly growth and a variance of -3% to 8% growth per year). This is an indication that only small change requests, bug fixes, and security patches are performed, while the addition of new functionality is very limited in this phase.
- **Decommissioning:** As expected, in this phase code bases have typically stopped growing (median of 0% with a very small variance). Also, the yearly change rate drops to 3%, indicating only changes are made that are needed to either keep the lights on or switch off functionality.

[prompt] picture of great software design showing from inside a computer

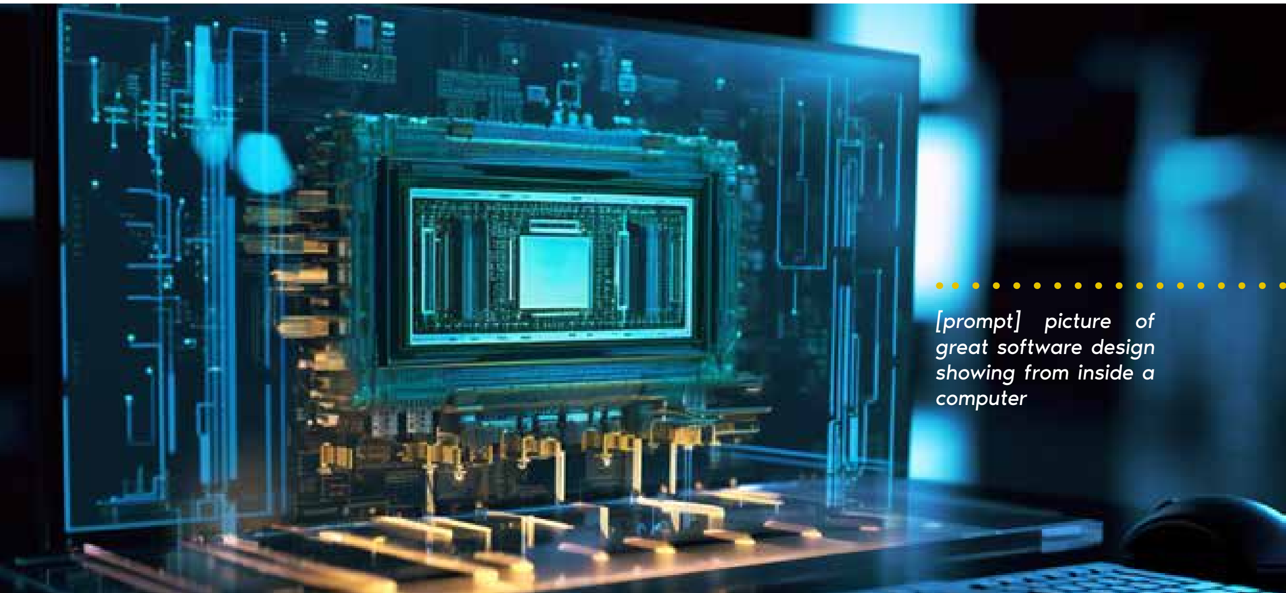
The data obviously shows variance. These numbers should be used as initial guidance, and a more detailed SIG analysis may be required to adequately forecast a specific system of a portfolio.

	Yearly Growth			Yearly Change		
	Low	Median	High	Low	Median	High
Initial Development	8%	**52%**	215%	29%	**168%**	562%
Evolution	0%	**10%**	33%	10%	**47%**	149%
Maintenance	-3%	**0%**	8%	3%	**15%**	53%
Decommissioning & End of life	-1%	**0%**	2%	0%	**3%**	16%

It's clear that the software lifecycle phases have a major impact on yearly growth and change rates. What are the implications? We see two major recommendations:

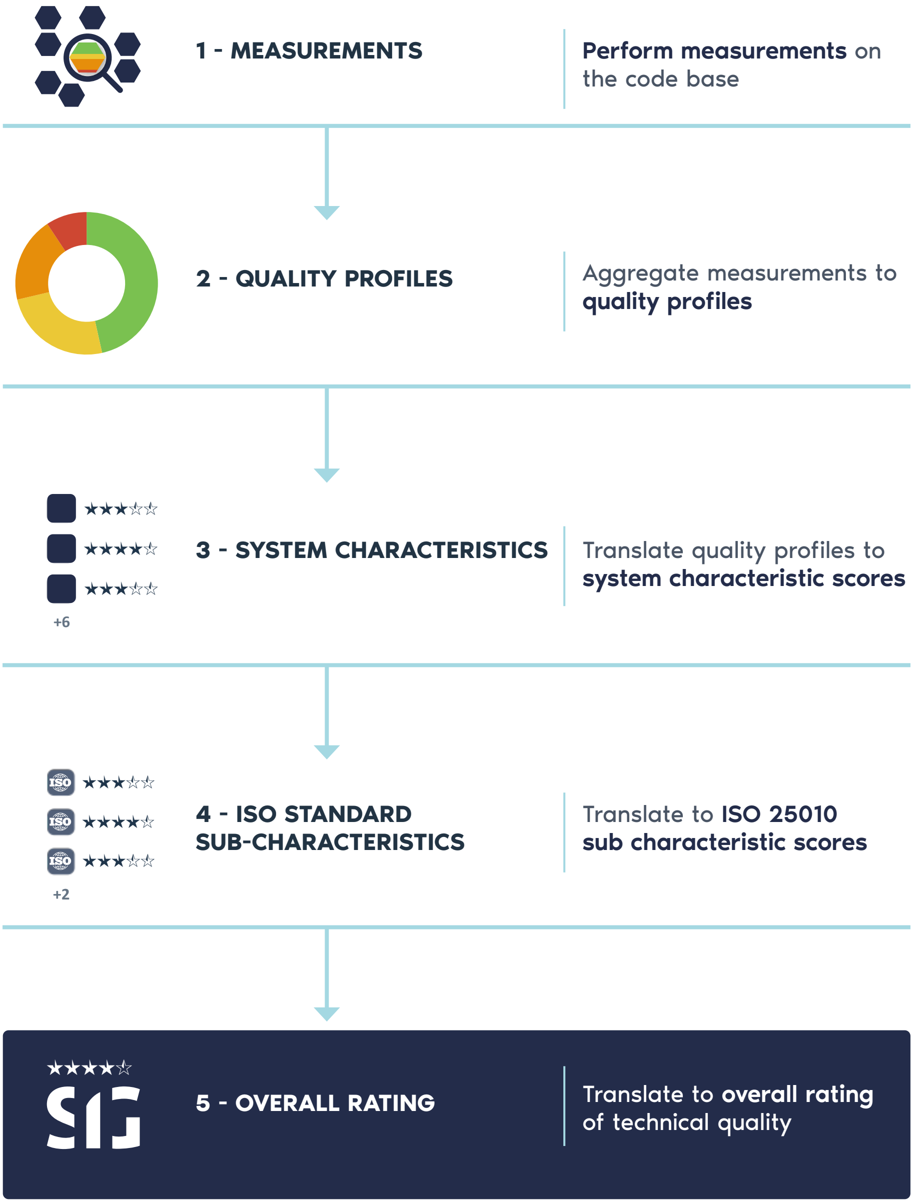
1. Organizations should further rationalize software portfolio management by making lifecycle phases explicit for each software system. Following that, year-over-year tracking of code growth and change rates should be evaluated against known benchmarks, such as the dataset presented in this report.

2. With the first part in place, software estimations should be enhanced following defined lifecycle phases. Code growth and change are major factors in the effort necessary to maintain and enhance the software. We advise fine-tuning and using these metrics in budgeting, decision-making, and portfolio roadmap design.



[prompt] picture of great software design showing from inside a computer

MAINTAINABILITY MEASUREMENT IS OUR TOOL TO DETERMINE SOFTWARE BUILD QUALITY



1 - MEASUREMENTS

Perform measurements on the code base

2 - QUALITY PROFILES

Aggregate measurements to **quality profiles**

★★★☆☆
★★★★☆
★★★☆☆
+6

3 - SYSTEM CHARACTERISTICS

Translate quality profiles to **system characteristic scores**

ISO ★★★☆☆
ISO ★★★★☆
ISO ★★★☆☆
+2

4 - ISO STANDARD SUB-CHARACTERISTICS

Translate to **ISO 25010 sub characteristic scores**

★★★★☆

5 - OVERALL RATING

Translate to **overall rating of technical quality**

2

Dennis Bijlsma / Lodewijk Bergmans

NEW SIG ARCHITECTURE QUALITY MODEL

pinpoints cost, risk, and slowdown factors

[prompt] an image from a technical point of view visual difference between a good building and a bad building

Software engineering is a socio-technical activity. Software is built by teams of people, working together to produce a joint product. High performers in socio-technical architecture quality show faster issue resolution times. To avoid the problems related to low architecture quality, it is recommended to avoid large system sizes, consistently pursue the reduction of coupling between components, and pay attention to the even distribution of developer activity across the system components.

The SIG Architecture Quality model measures the socio-technical architecture of software systems, providing insight into the ability of the architecture to evolve and scale.

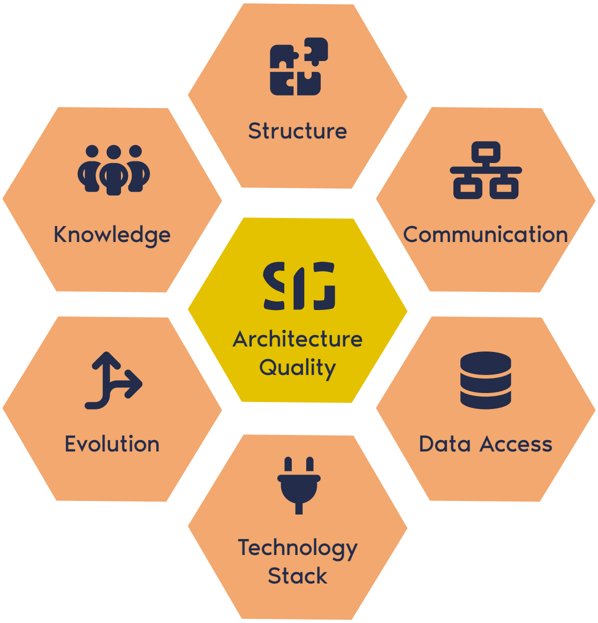
Organizations have long been under pressure to evolve their software landscapes in a way that aligns with ever-changing business demands. Systems no longer able to meet these changing demands are commonly referred to as *legacy systems*, which need to be modernized to address these challenges.

IT teams everywhere want to make the leap to modernization, but such an initiative presents enormous challenges and risks. In fact, 74% of organizations fail to complete legacy modernization projects, according to a recent report by Advanced[2].

"Legacy" means much more than just outdated technology. Rather, it refers to any pre-existing software solution that has become too fragile for changes to be timely, predictable, and reliable, usually due to poor architecture, or team knowledge loss.

SIG uses an Architecture Quality Model to quantify these aspects of socio-technical software architecture. The model captures six architecture aspects, covering both technical and social aspects, and evaluates the results against other systems in SIG's benchmark.

[2] https://www.oneadvanced.com/trends-report/2020-21/



Each architecture aspect influences the degree to which the system can be easily changed or extended. The six aspects are:

- **Structure:** The grouping and organization of functional and technical areas in a code base.
- **Communication:** The dependencies between functional or technical areas in a code base.
- **Data Access:** The way various components depend on databases and other data stores.
- **Technology Stack:** The combination of technologies used in the code base and their associated risks.
- **Evolution:** How the frequency and distribution of changes affect a code base over time.
- **Knowledge:** The degree to which activities on, and knowledge of, a code base is distributed among team members.

In this chapter, we will discuss how architecture quality is related to development speed, how architecture quality requires attention in addition to the maintainability of the source code, and what the key improvement areas are for establishing and maintaining good architecture quality.

1. Improving socio-technical architecture leads to faster issue resolution time

As the amount of software in organizations continues to grow, those same organizations need to keep innovating to address their customers' ever-increasing expectations. One innovation trend is the move towards microservice architectures. This architecture pattern avoids large, monolithic applications in favor of many small independent components. Those components, called microservices, each focus on one specific responsibility.

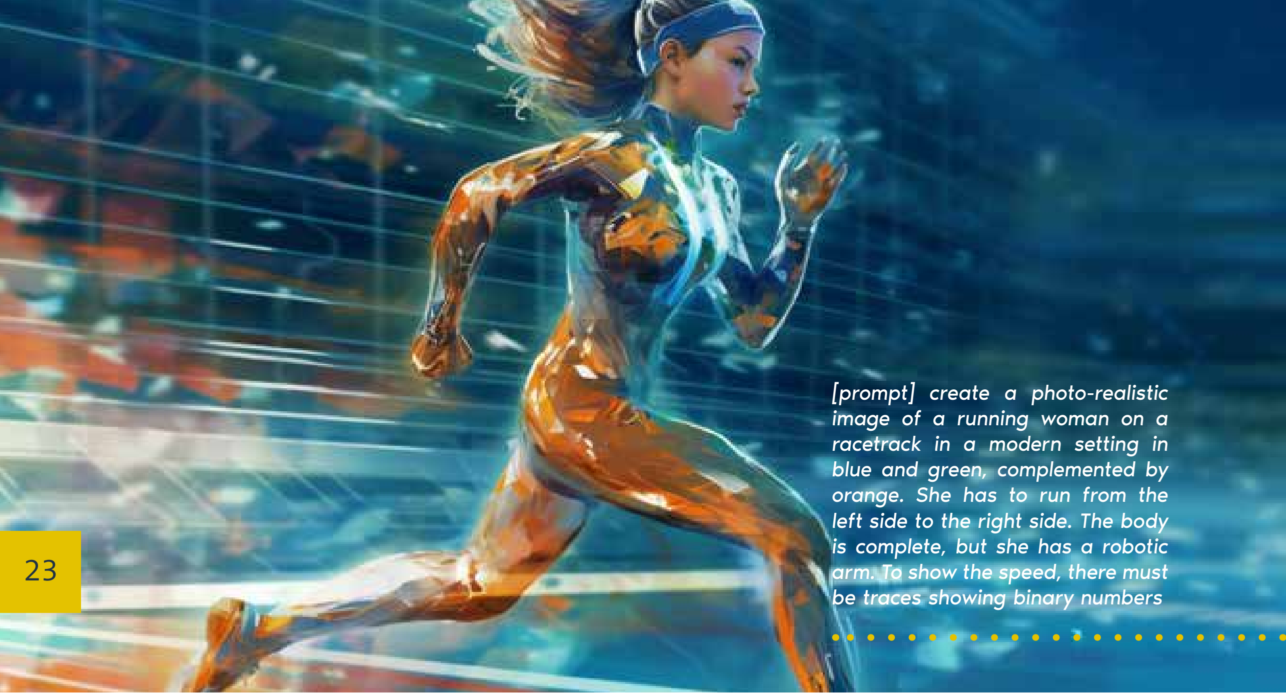
So are microservices just a fad that is already dying out, or are these principles here to stay? If we look at SIG's benchmark data, we see that microservice architectures have become mainstream around 2017, and this has caused a significant increase in the average number of components per system that is still visible to this day. This means the trend towards systems that consist of many small components is both widespread and showing no signs of slowing down.

However, microservices are not entirely independent in practice. Their code is independent, but they still communicate through APIs, interfaces, middleware, or databases. Though the components/services are no longer coupled on the code level, there are still other types of coupling that developers need to consider.

This doesn't mean microservice architectures are suddenly a bad idea or have no benefits. Instead, this simply means that designing a system architecture needs to carefully consider people, code structure, interfaces, and deployment.

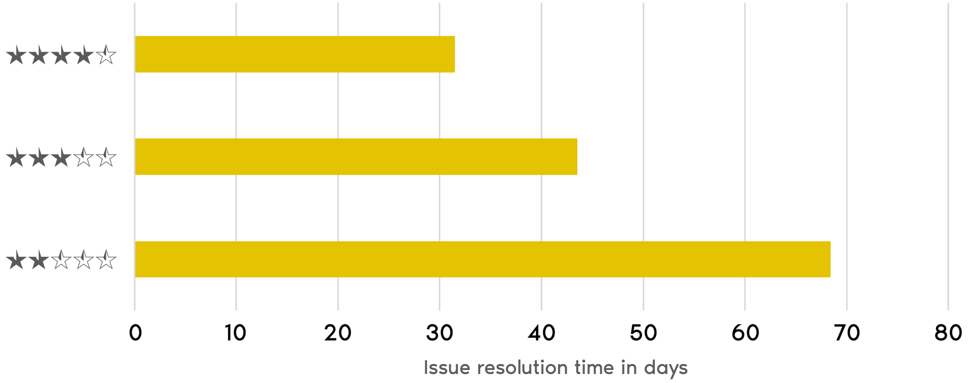
As explained in the introduction to this section, SIG has introduced an Architecture Quality Model that captures these challenges and trade-offs. When done well, a good software architecture will facilitate people to work independently by reducing coupling (both technical and social/organizational).

So can we quantify those benefits? For an initial analysis, we took 50 systems from SIG's benchmark, with an average size of 40 person-years of code. For those systems, we then compared their architecture quality (as measured by SIG's model) with their issue resolution time (i.e. time-to-market for changes and new features).



[prompt] create a photo-realistic image of a running woman on a racetrack in a modern setting in blue and green, complemented by orange. She has to run from the left side to the right side. The body is complete, but she has a robotic arm. To show the speed, there must be traces showing binary numbers

Relation between architecture quality and issue resolution time



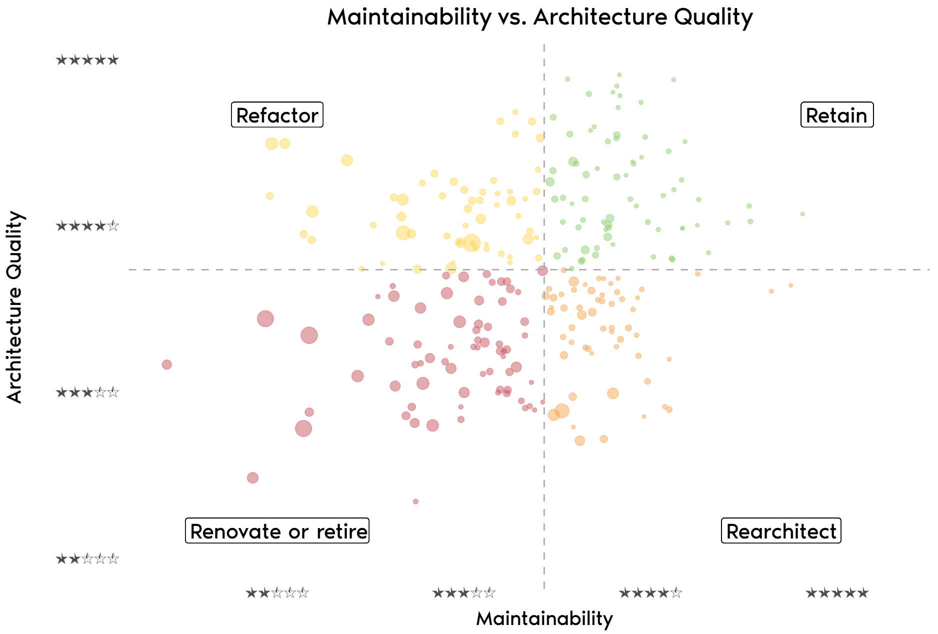
This diagram shows a correlation between architecture quality and issue resolution time. When it comes to architecture quality, it is, on average, 30% faster to make changes in a 4-star system than in a market-average system. Reversely, making changes to a 2-star system will be about 40% slower.

Key Finding: Systems with 4-star Architecture Quality resolve issues two times faster than 2-star systems.

2. Measure maintainability and socio-technical architecture to determine technical strategy

Historically, most attention on legacy systems has focused on the functional, operational, and technical challenges surrounding such systems. However, socio-technical architecture is increasingly a focus area, being named in a report by Ardoq[3] as the #1 trend for enterprise architects.

SIG considers maintainability to be the foundation for ensuring agility and flexibility. So how does the SIG maintainability rating relate to the socio-technical software architecture ratings?



This chart shows the maintainability and architecture ratings for all systems where SIG evaluated both aspects in 2022. The x-axis depicts a system's maintainability rating, while the y-axis depicts the socio-technical architecture rating. The size of each dot represents the size of each system. The colors are used to easily identify each quadrant.

This chart shows that maintainability and socio-technical architecture are mostly independent challenges: there are many systems that have poor

[3] https://content.ardoq.com/enterprise-architecture-trends-infographic

maintainability but acceptable socio-technical architecture, but there are also many highly maintainable systems that have poor architecture. This means that, depending on which quadrant your system is located in, you can decide to focus on specific quality aspects first.

Each of the quadrants can be characterized by the most likely action to consider:

- **Renovate or retire:** in the bottom-left corner are systems with both low maintainability and low architecture quality; these systems are candidates for serious quality improvement, or to be retired completely.
- **Refactor:** these are the systems with a sound architecture quality, but low maintainability, hence refactoring the code to improve maintainability is a worthwhile consideration.
- **Rearchitect:** this quadrant contains the systems that have a high maintainability, but suffer from low architecture quality. Especially for systems that are expected to evolve substantially in the future, it may be a worthwhile investment to re-architect the structure and organization of the system.
- **Retain:** The top-right quadrant contains systems with both good architecture and high maintainability, so these systems are in a good situation for further enhancement and evolution.

The size of the dots also reveals that most of the systems in the 'retain' category are smaller systems, which is generally also a trend for high maintainable systems. For architecture quality, the size of the systems matters much less: there are many mid- to large-sized systems with high architecture quality, even though we see that the largest systems are in the renovate/retire category.

Key Finding: Our architecture benchmark confirms that larger systems often suffer in quality - but not always, so it is indeed possible to create big systems at high architectural quality. High architecture quality allows for systems to be refactored more easily and kept maintainable.

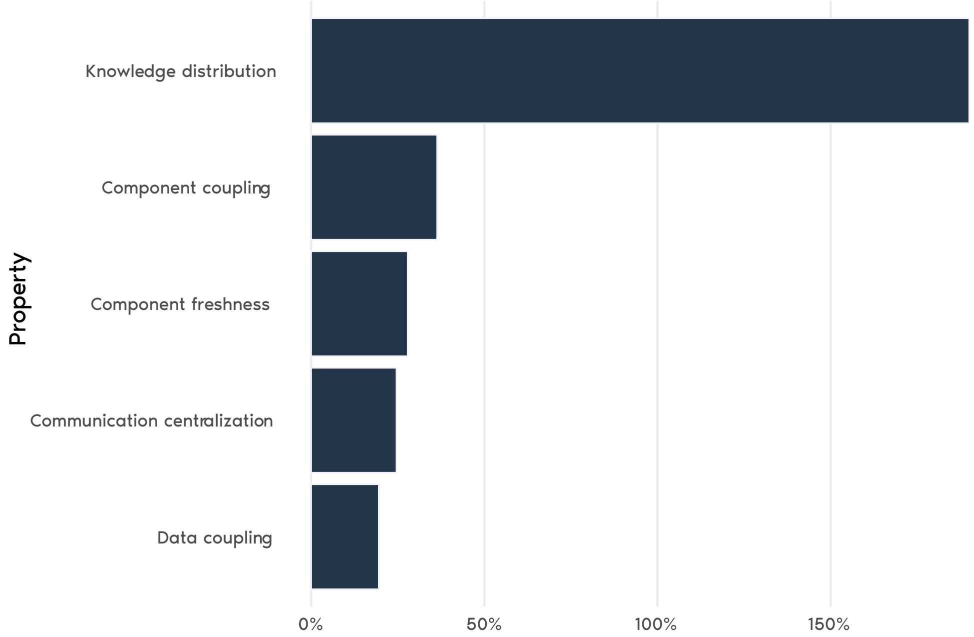
In the next section, we investigate which factors most strongly influence architecture quality and which are the most relevant areas to focus on to ensure good architecture quality and an evolvable system.

3. Coupling and knowledge are the main challenges for socio-technical architecture

In the previous section, we explained the challenges that organizations face in their socio-technical architecture, especially when it comes to modernizing legacy systems. But what factors contribute most to these challenges?

SIG's Architecture Quality Model produces an overall rating from 1 to 5 stars, but this rating is actually composed of 10 underlying system properties. Exploring the benchmark data for these system properties will help to determine which properties are the biggest challenge on the road to good architecture.

WHAT ASPECTS DISTINGUISH THE HIGH PERFORMERS ON ARCHITECTURE QUALITY?



The above diagram shows the top five properties where high architecture quality systems outperform the medium quality systems: it shows how the average high performer compares to the average system in the benchmark for each of the properties. More precisely, each bar shows the delta between the median rating of the high performers and the median rating of the mid-performers in the benchmark. A significant delta for a given property indicates that, for most systems, there is significant room for improvement, especially for that property.

Key Finding: *Knowledge Distribution, Component Coupling,* and *Communication Centralization* are the big factors of high-quality architecture in the SIG architecture quality benchmark: get these right to increase architecture quality.

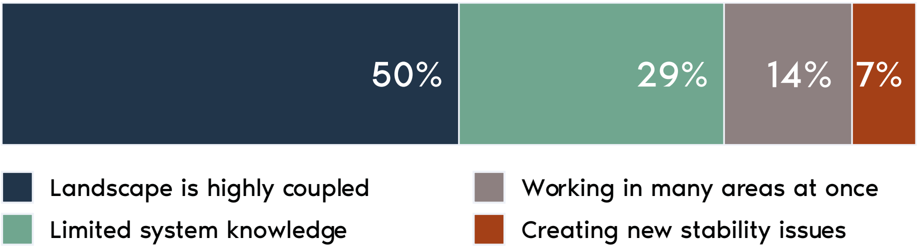
The system properties with the most significant deltas are *Knowledge Distribution, Component Coupling*, and *Communication Centralization*. These properties cover very different aspects of the architecture: Communication Centralization and Component Coupling cover *technical* coupling; the dependencies between different parts of the code. More dependencies, spread more widely across the code, makes the code harder to change, and causes changes to one component to ripple through to other components (often impacting additional development teams).

Knowledge Distribution indicates the degree to which developers can effectively work in parallel, with a joint knowledge that covers the entire code base: it measures to what extent there is a balanced distribution of development activity across all software components. This means that there are developers with recent knowledge about all parts of the system and that developers do not work too much on the same parts of the code, which is a known source of inefficiencies and bugs[4].

The above conclusions about the key areas influencing architecture quality, as derived from applying the architecture quality model on a large set of systems analyzed by SIG, are very well complemented by the results of a poll for ± 150 software architects that SIG organized on LinkedIn. The following chart summarizes the answers to one of the questions.

[4] Chuanqi Wang, Yanhui Li, Lin Chen, Wenchin Huang, Yuming Zhou, Baowen Xu, Examining the effects of developer familiarity on bug fixing, Journal of Systems and Software, Volume 169, 2020

SURVEY: WHAT DO YOU SEE AS THE MAIN CHALLENGE IN MODERNIZING YOUR CURRENT SOFTWARE LANDSCAPE?



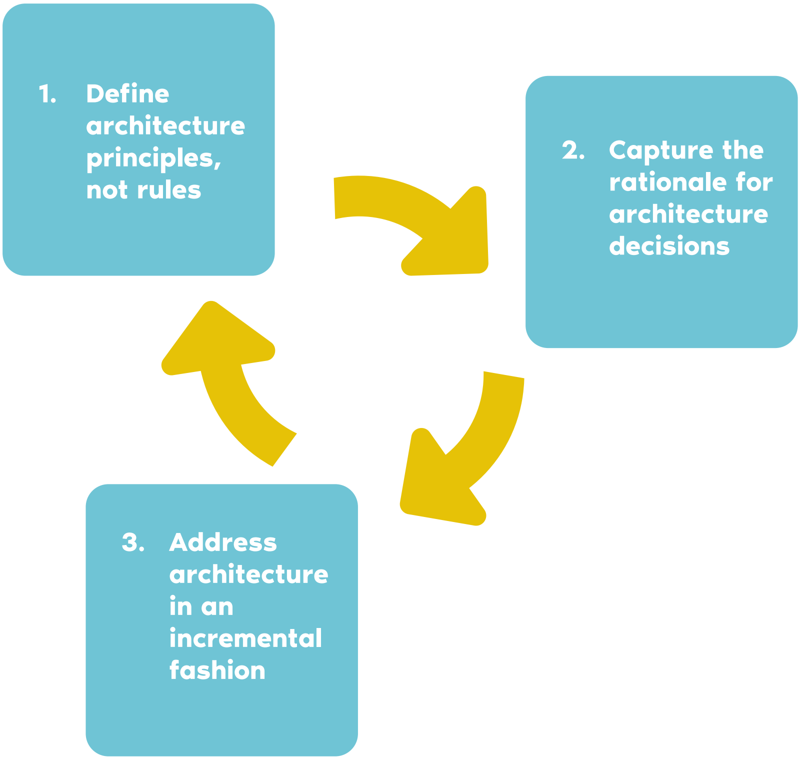
Of the responding software- and enterprise architects, 50% considered the coupling of the software landscape as the main challenge for modernization, and in the second place, 29% considered limited system knowledge as the main challenge. So for this aspect, the Architecture Quality model and architects are well aligned about the key challenges.

4. Improve Architecture Quality with incremental modernization principles

In the previous sections, we have seen that socio-technical architecture quality is a separate quality aspect of software systems, which is becoming increasingly important as the complexity of software architectures is increasing.

The SIG architecture quality model aims to bring better insights into the ability of an architecture to evolve swiftly. The underlying metrics of the model offer suggestions on what properties of the system can (or need to be) improved.

Based on SIG's experience in helping organizations to modernize and improve their architecture, we recommend the following best practices for ensuring a future-proof architecture:


1. Define architecture principles, not rules
2. Capture the rationale for architecture decisions
3. Address architecture in an incremental fashion

SIG recommended practice:

DEFINE ARCHITECTURE PRINCIPLES, NOT RULES.

An evolving architecture means it is not possible to fully define the architecture up front. Nor is this desirable. Having independent systems and components should facilitate the teams working on them to adapt those systems and components, without the need for a central decision authority to approve every single change. Teams should be able to guide their own architecture as long as they follow the general direction laid out by the organization's architecture principles.

These architecture principles should help guide teams in decision-making without dictating or micro-managing every single aspect. For example, defining a list of which services are allowed to communicate with each other easily becomes unworkable. In a landscape with hundreds or even thousands of services, this list will quickly become extremely long. Moreover, the list will keep changing as the landscape is continuously evolving. Having to approve every modification to the list quickly becomes a bottleneck, especially when the person needing to approve the change is not a member of the team.

It is, therefore, better to define general guidelines and principles and then let the teams decide how to achieve them. As a simple example: communication between services could be allowed assuming they only access each other via REST APIs.

Obviously, some sort of feedback loop is still needed to make sure the principles are actually applied in practice. This is where SIG's architecture quality model can help: low ratings can indicate that a principle is not followed, or that a new principle needs to be defined.

SIG recommended practice:
CAPTURE THE RATIONALE FOR ARCHITECTURE DECISIONS.

Software architecture is often a series of trade-offs. However, for people not involved in the original decision, it is often no longer clear what the trade-off was or how that trade-off led to the decision.

One approach to capturing these decisions is Architecture Decision Records[5] (ADRs). Note that ADRs are not some kind of technical rule, they are essentially a document. But having such a document can be a useful communication device, as it creates a history that allows people to keep track of trade-offs made in the past. The code and architecture themselves can only communicate the current state, but not how that current state came to be.

Note that recording decisions doesn't make them permanent or immutable. It is still perfectly fine to revisit architecture decisions, especially during changing circumstances. In fact, having an Architecture Decision Record makes it easier to revisit decisions, since there is a clear overview of which decisions were made and why.

[5] https://www.softwareimprovementgroup.com/using-architecture-decision-records-to-guide-your-architecture-choices/

[prompt] a picture of digital growth in fancy colors and binary code

SIG recommended practice:
ADDRESS ARCHITECTURE IN AN INCREMENTAL FASHION.

Addressing technical debt at code level is often done using small, incremental refactorings. Such an approach leads to lower risk, as the scope of changes is smaller. In many organizations, architecture changes do not follow this agile approach and tend to be structured into "projects" where large changes are made over a period of time.

Incremental architecture modernization removes some of the risks associated with architecture changes: small, incremental changes have a smaller scope and, therefore, lead to less stability risk. Moreover, it avoids a situation where architecture modernization is directly competing with functional changes.

Changing the architecture in an incremental way often seems unfeasible. And indeed, you will not be able to solve the problem of thousands of unwanted dependencies between two systems in a single sprint or iteration. But you can divide the overall goal into smaller parts: for example: first investigate dependencies between subcomponents and strive to change those within a sprint.

The system properties in the SIG Architecture Quality Model map directly to architecture modernization techniques that can be applied in this way:

- **Address the coupling between architecture components**
  There is a large body of (architecture) design patterns that can be applied to reduce various forms of coupling.

- **Improve communication centralization**
  This requires especially a disciplined approach to group or reduce outgoing calls, as well as a focus on developing APIs that are cleanly separated from the implementation of the component.

- **Reduce the size of the system**
  By cleaning up 'dead' or unused code, reducing the scope, improving code reuse within the system or by adopting more standard (library) solutions.

To ensure these architecture improvements remain a point of attention, these aspects also need to be incorporated into the definition of done for each sprint. This allows for architecture to remain a topic of continuous consideration, to avoid architectural decay over time. Defining explicit goals, for example, using Sigrid's objectives dashboard helps to track incremental progress while keeping the eventual long-term goal in view.

3

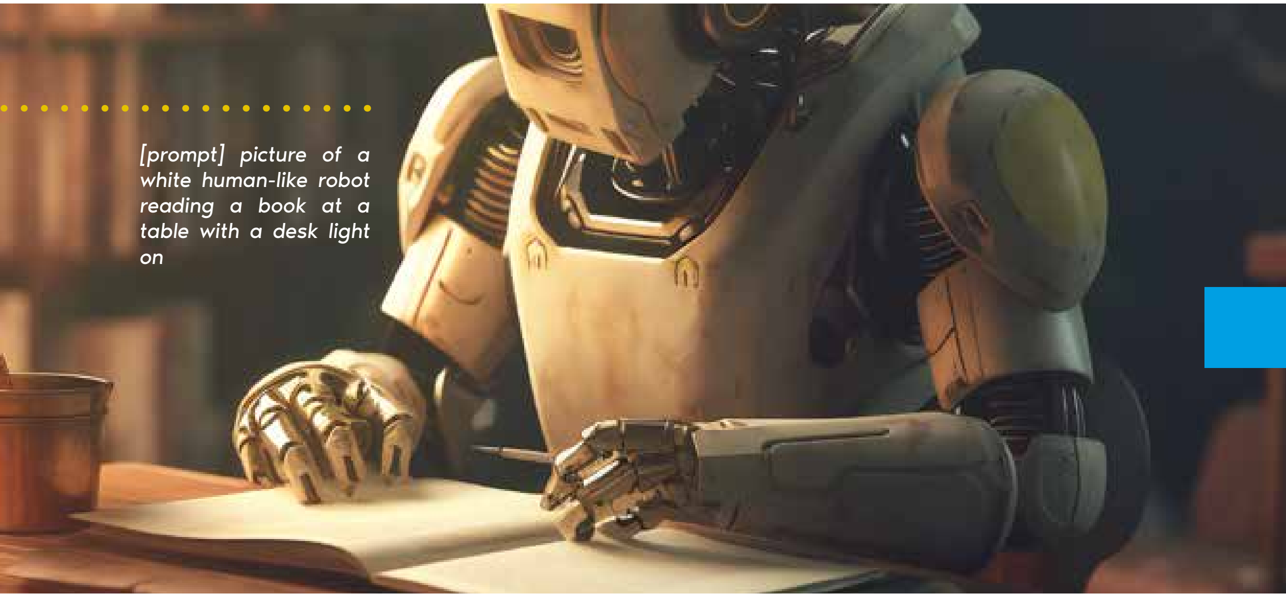
Rob van der Veer / Asma Oualmakran

AI AND BIG DATA SYSTEMS PLAGUED BY POOR CODING

Artificial Intelligence is on the rise, and therefore an increasing part of the systems we see at SIG involve AI and big data. We observe, in most cases, these systems suffer severe quality issues, predominantly in maintainability and testability. What does the SIG benchmark have to say about this? What are typical quality issues of AI/big data systems, what can organizations do to control them, and how can they make AI a sustainable success?

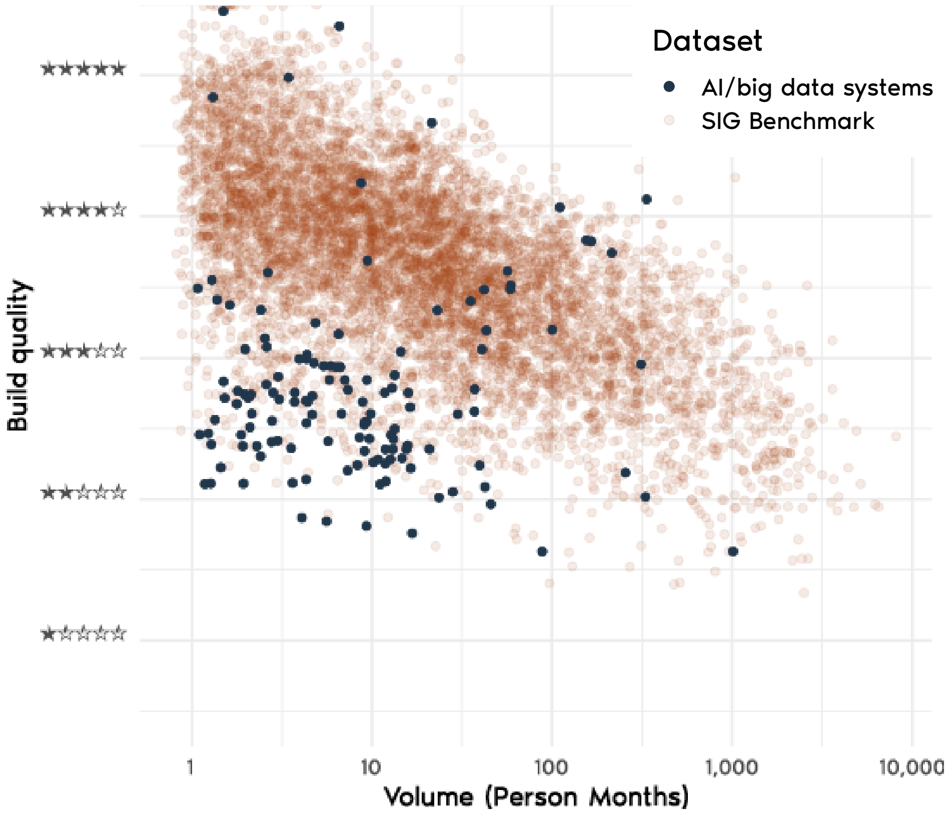
Data analysis of our benchmark shows that AI/big data systems are significantly less maintainable than other systems. 73% of AI/big data systems score below the benchmark average. Their average maintainability rating of 2.7 stars is significantly lower than the average of other systems[6]. This is mostly caused by the quality properties Unit Size and Unit Complexity. AI/big data systems are, on average, in the 5% bottom of the industry regarding Unit Size (long blocks of code), and for Unit Complexity in the bottom 25%.

Fortunately, there are also AI/big data systems with high maintainability, as our benchmark clearly shows. This demonstrates that it is, thankfully, not impossible to build maintainable AI/big data systems.



[prompt] picture of a white human-like robot reading a book at a table with a desk light on

[6] T-test rejected the null hypothesis that there is no significant difference between the maintainability of AI systems and general SIG benchmark.



Our dataset of AI/big data systems was compiled by selecting systems that revolve around statistical analysis or machine learning, based on the technologies used (e.g. R and Tensorflow) and documentation.

According to the SIG maintainability model, long and complex blocks of code are hard to analyze, hard to modify, hard to reuse and hard to test. The longer code blocks are, the more responsibilities they tend to cover, and the more complex, the more decision paths there are. This explains why it is harder, if not impossible to create tests that cover everything. The testability problem is demonstrated by the dramatically small amount of test code. In the typical AI/big data system, 1.5% of the code is test code, whereas for the benchmark this is typically 43%.

Key Finding: In the typical AI/big data system, 1.5% of the code is test code, whereas the benchmark is 43% test code.

Low maintainability makes it increasingly difficult to perform changes, and the risk of introducing errors grows without the proper ability to detect these errors. In other words: typical AI/big data code tends to become a liability the longer it needs to be maintained. Over time, data changes and requirements change, and because of lacking maintainability, adaptations are bolted on - which reduces the probability of error but adds to the maintainability problem.

Key Finding: On average, AI/big data systems have significantly lower maintainability, mostly caused by long and complex blocks of code accompanied by a very low amount of test code. The result is that AI/big data systems tend to be difficult and costly to change, extend, and integrate, with a high risk of making mistakes. Furthermore, this can severely hinder transferring AI/big data systems to another team.

What could be causing these long and complex code units? Typically, such issues are the result of unfocused code (having more than one responsibility) and the lack of abstraction: useful pieces of code are not isolated into separated units. Instead, they are copy-pasted in code. Without exception, it is our experience that AI/big data code suffers from these problems.

The lack of abstraction can be illustrated by the example below. The purpose of this code is to select the most recent date plus to make the dates valid. When looking at the code, it looks similar to SQL. We often see this in the AI/big data field, as data scientists are accustomed to SQL for processing data. It is typical for SQL code to lack abstractions because developers are often unaware of the available language abstraction mechanisms, making it very hard to read and understand the purpose.

```
GREATEST(IIF(ISNULL(i_RS_VLD_FM_DT),TO_DATE(v_
LOGC_RSVD_VAL_UNKNOWN, ‘YYYY-MM-DD HH24:MI:’),i_
RS_VLD_FM_DT),IIF(ISNULL(i_RS_VLD_FM_DT_fauit),
TO_DATE(v_LOGC_RSVD_VAL_UNKNOWN, ‘YYYY-MM-DD
HH24:MI:SS’), i_RS_VLD_FM_DT_fauit),IIF(ISNULL(i_
RS_VLD_FM_DT_xref_sol),TO_DATE(v_LOGC_RSVD_VAL_
UNKNOWN,‘YYYY-MM-DD HH24:MI:SS’),i_RS_VLD_FM_DT_
xref_sol))
```

The above code snippet can be refactored in the below more abstracted and readable code. Conditions of each parameter are abstracted into the function MakeValidDate. This single improvement has a significant impact on maintainability as repeated functionality is simplified, centralized, and made testable.

```
Greatest        ( MakeValidDate(i_RS_VLD_FM_DT),
                MakeValidDate(i_RS_VLD_FM_DT_fauit),
                MakeValidDate(i_RS_VLD_FM_DT_xref_sol))
```

Key Finding: Important causes for maintainability issues in AI/big data systems are code having multiple responsibilities and the lack of abstractions.

What could be the root cause for the AI/big data maintainability issues?

1. **Lab programming:** most of data science work is aimed at a single experiment, to try things in one-shot, or solve an analytical problem ad hoc - not with the intention to deliver something to go into production for a long time per se. The problem is that once things work, there is no real incentive for the data scientist to refactor and improve code quality. After all, there are no tests, so changing code is risking breaking it without noticing it.
2. Data science **education programs** typically focus more on data science and less on software engineering best practices.
3. Traditionally, **data science development tools** lack the support for software engineering best practices.
    a. R and Jupyter notebooks for example are based on the paradigm of a step by step one-shot approach, which is suitable for experiments but not for maintainable software.
    b. Some data science languages lack powerful abstraction and testing mechanisms.
4. The **SQL pattern** is often the standard paradigm for data preparation. This pattern comes down to working with datasets that are joined and contain many consecutive operations on many fields at the same time. In AI/big data this represents a large part of the work (75-90%[7]), and it has its maintainability challenges - for which the solutions are often unknown to data scientists. Data scientists find this the least enjoyable part of the work[8] and the most difficult[9].

For AI/big data systems, we typically encounter teams with predominantly data scientists. When working with them, we observe that they are focused on creating working analytics and models while often lacking a number of software engineering best practices, typically leading to the issues that have been discussed.

[7] (Microsoft 2019)

[8] Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says", Forbes, Gil press, March 2016

[9] Biggest difficulty: "Software Engineering for Machine Learning: A Case Study", 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE- SEIP), Amershi et al. Microsoft

Part of the reason why unit testing is lacking in AI/big data systems is that the engineers rely on integration tests, which can be done elegantly for this type of system by measuring the correctness of the AI model. If the model performs badly, this can be caused by some issue in the pipeline. The problem with this approach is twofold:

1. Due to the lack of unit tests, it is not clear where an issue is located.
2. The model can perform okay, but there may be an issue preventing the model from performing much better.

For example, a model to predict sales of drinks uses the weather as input, and let's say it scores 80% correct. Suppose there is an issue causing the temperature to always say zero - preventing the model to score 95%. Without unit tests, this may never be found.

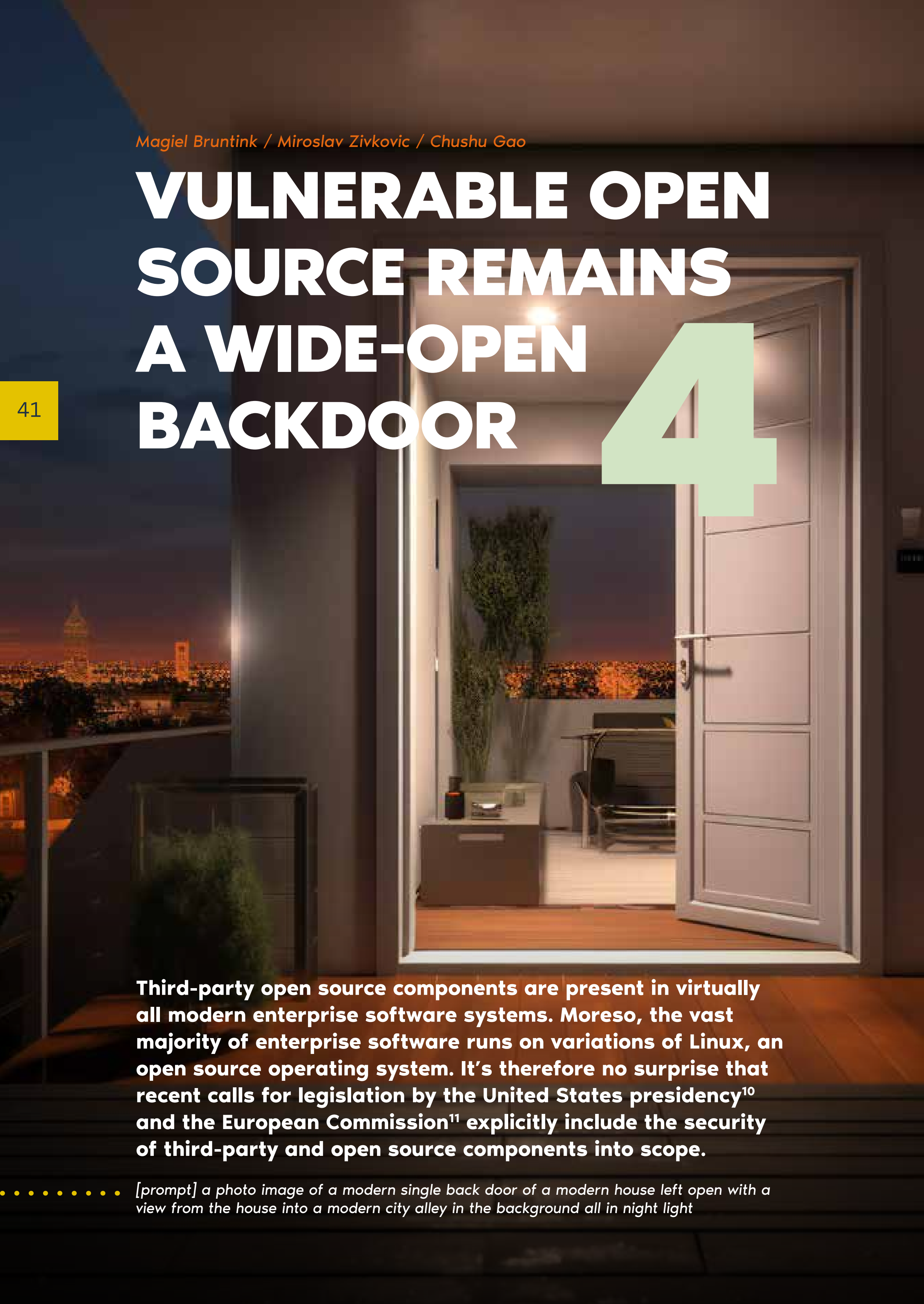
What are the SIG recommended practices to deal with AI/big data systems?

First of all, by measuring and improving the maintainability of AI/big data systems. Data science teams can then get immediate feedback on the quality of their work. What also helps is mixing people coming from data science with people from software engineering. This helps in two ways:

- Data scientists can be coached to write code in a more future-proof and robust fashion, and they will happily embrace this once they see how they benefit from it in their daily work.
- On the other hand, it is beneficial to have engineers that are new to data science learn from the powerful paradigms and tools available for big data and AI.

Key Finding: Maintainability issues in AI/big data systems have a root cause in the way data scientists tend to work, given their focus on experiments, their education, their tools, and the fact that data preparation is the dominant part of their work.

[prompt] a visual of the problems for engineers with AI from the view of maintainability



Magiel Bruntink / Miroslav Zivkovic / Chushu Gao

VULNERABLE OPEN SOURCE REMAINS A WIDE-OPEN BACKDOOR

4

Third-party open source components are present in virtually all modern enterprise software systems. Moreso, the vast majority of enterprise software runs on variations of Linux, an open source operating system. It's therefore no surprise that recent calls for legislation by the United States presidency[10] and the European Commission[11] explicitly include the security of third-party and open source components into scope.

[prompt] a photo image of a modern single back door of a modern house left open with a view from the house into a modern city alley in the background all in night light

One of the most significant aspects of that legislation is that producers of software on either side of the Atlantic will become accountable for cybersecurity issues in their products. A burden previously borne mostly by users. Producers will have to perform adequate due diligence for vulnerabilities in open source components, among many other defects that potentially compromise security. They will need to either comply with accepted security standards or be held liable for damages, fines, or both.

Nowadays, SIG analyzes more than a hundred thousand dependencies on open source components in enterprise software systems, every month. In light of the upcoming legislative revolution, we can therefore provide an urgently needed status update.

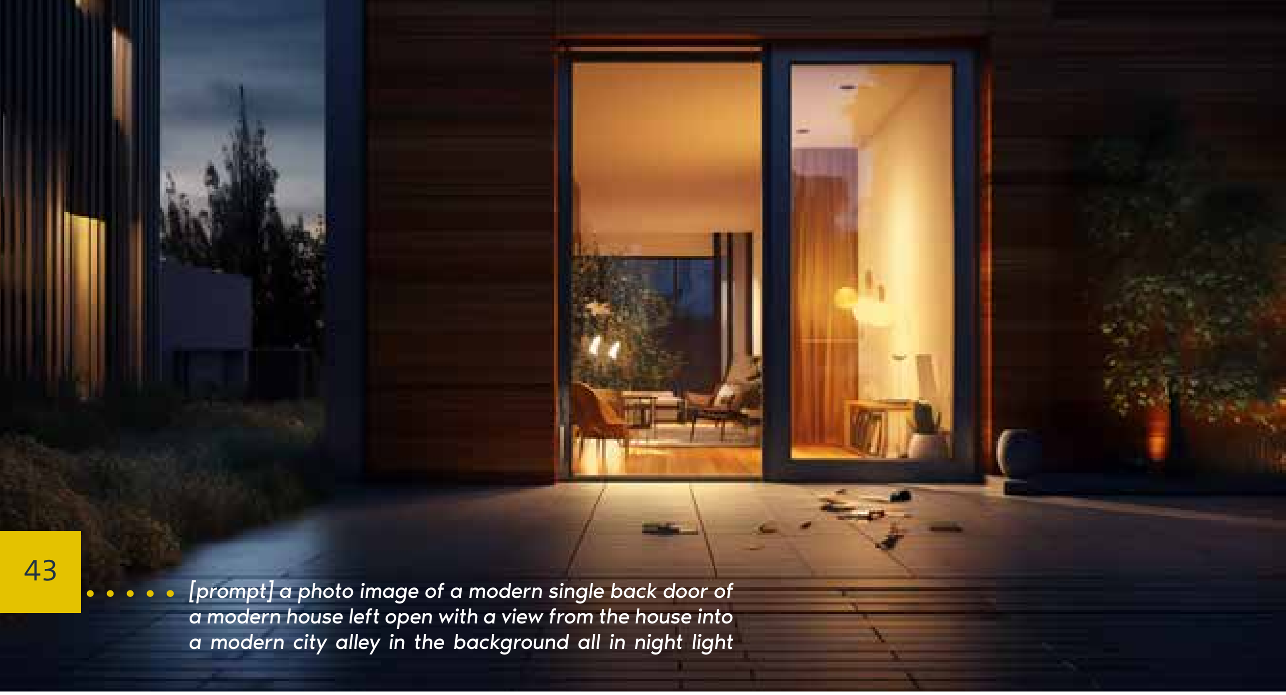
In last year's Benchmark Report 2022[12], we already shared some insights that were cause for alarm.

- Overall, enterprise software producers update their open source dependencies after years, rather than weeks, of updates being available.
- Even in the presence of critical security vulnerabilities, average patch times reflected no urgency.
- We revealed correlations between the build quality of open source code and the risk of vulnerabilities: worse code quality is linked to higher risk.

One year on we revisited our core findings to provide a fresh view of the state of affairs. Spoiler: we're not yet where we need to be, not by a longshot. There may be some recent hints of gradual improvements, but it's more accurate to say that we are still in the middle of a vulnerability pandemic.

Key Finding: 50 to 60% of enterprise software systems have a vulnerable open source dependency, each month. Around 30% have a critically vulnerable dependency. Business critical systems are only marginally less exposed than less critical systems.

[10] National Cybersecurity Strategy https://www.whitehouse.gov/wp-content/uploads/2023/03/National-Cybersecurity-Strategy-2023.pdf
[11] Cyber Resilience Act https://digital-strategy.ec.europa.eu/en/policies/cyber-resilience-act
[12] SIG Benchmark Report 2022 https://www.softwareimprovementgroup.com/publications/2022-sig-benchmark-report/



[prompt] a photo image of a modern single back door of a modern house left open with a view from the house into a modern city alley in the background all in night light

Let's have a good look at our extended data and distill answers to the following questions:

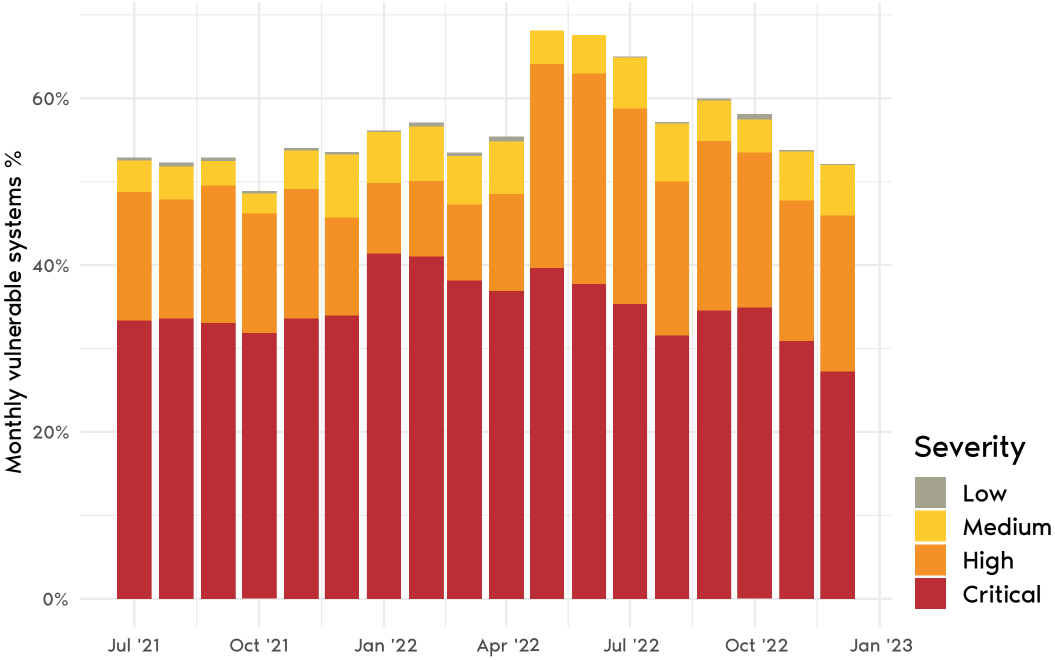
1. How many enterprise software systems have vulnerabilities in their open source components? What sectors of the software industry do a comparatively good job managing vulnerabilities?
2. Are there aspects of open source components that make some relatively safer than others? How to use open source components responsibly?
3. What should software-producing organizations do to prepare for future legislative changes? How to turn the tide and manage the list of vulnerabilities to zero?

1. More than 50% of enterprise software systems use vulnerable open source

Using *Sigrid® | Open Source Health*, SIG helps clients monitor risks in the open source usage of their enterprise software systems. Each month, the dependencies of 1,000 or more systems are scanned for risks related to security vulnerabilities, licensing, and out-of-datedness.

The following graph shows the overall percentage of systems that are found to include a vulnerable open source dependency. As the vulnerabilities are colored by severity, low (sand) to critical (red), it's easy to note that, on average, 30% of systems include a critical vulnerability among their dependencies. We see a vulnerable dependency of any severity for 50% to 60% of the systems monitored each month.

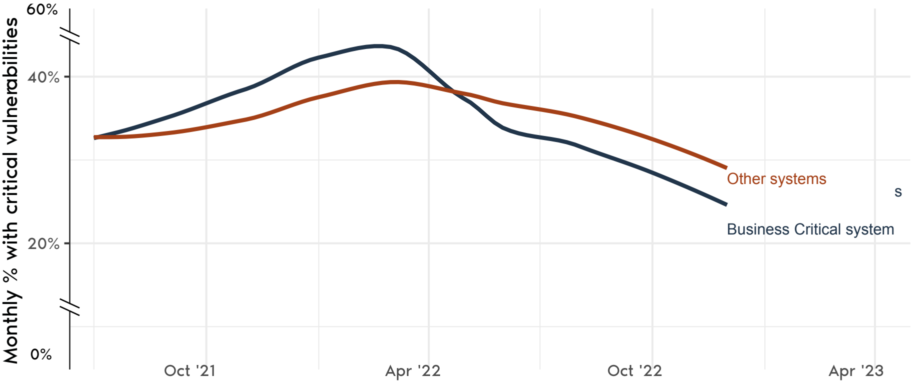
OVERALL TREND OF VULNERABLE ENTERPRISE SOFTWARE SYSTEMS



In short, the majority of enterprise software systems we monitor are affected by these -potentially exploitable- security issues. Looking at the data from the second half of 2022 slightly optimistically, a modest downward trend of critical vulnerabilities can be seen.

Carving our data to highlight the class of systems that are deemed to be *Business Critical* by their owners reveals the following trends:

BUSINESS CRITICAL SYSTEMS ARE ONLY MARGINALLY LESS EXPOSED TO CRITICAL VULNERABILITIES



It appears that marking systems as business critical, for example, enterprise architects or higher management has no significant impact on the exposure to critical vulnerabilities. In our data, we observe much the same rates of critical vulnerabilities across the years 2021 and 2022 for the business critical and the less critical systems. It seems there is a significant disconnect between the business owners and the technical owners: a gap that is urgently asking to be bridged.

The usage of open source libraries is most common for modern programming technologies. In most cases, a modern language is embedded in sprawling ecosystems of libraries and tools that provide substantial benefits to productivity and developer well-being. At the same time, libraries from these ecosystems can pose risks, as build quality is not up to standards, new ways to exploit old software are continuously discovered, or human errors remain undetected during development.

Below we list the top 10 most seen dependencies that had a critical vulnerability in 2022. These are popular libraries that enjoy a lot of attention among developers and hackers alike. If vulnerabilities become known in these libraries, teams do well to update to safer versions on short notice.

#	Critically vulnerable dependencies	Language ecosystems	% systems that used
1.	FasterXML Jackson	Java Maven	9.9%
2.	Spring Framework (SpringShell)	Java Maven	9.8%
3.	OWASP HTML Sanitizer	Java Maven	7.8%
4.	Spring Framework	Java Maven	7.8%
5.	Log4net	.NET NuGet	7.4%
6.	Log4j 1.2	Java Maven	7.0%
7.	.NET Core	.NET NuGet	6.7%
8.	Commons Text	Java Jar	5.6%
9.	PDFBox	Java Jar	5.1%
10.	PostgreSQL	Java Maven	4.6%

The top 10 list for 2022 indeed features exclusively the common modern programming technologies in enterprise software, Java and .NET. The XML handling library Jackson is one of the most used libraries in Java and at the same time the one for which the most vulnerabilities have been reported in previous years. Close to 10% of systems we analyzed in 2022 used a critically vulnerable version of Jackson. A very close second is the popular Spring Framework, with an exploit known as SpringShell (following the naming of the late 2021 Log4Shell incident).

Why should you care? Check Point Research reported a staggering weekly average of 1,200 cybersecurity attacks per organization worldwide in 2022, an increase of 38% from 2021[13]. The majority of supply chain security attacks (66% as reported by ENISA, the European Union Agency for Cybersecurity[14]) are vectored in through third-party software. So, plenty of reasons to tighten up the handling of vulnerable open source dependencies.

Our top 10 list consists of vulnerable libraries that are commonly used directly, or sometimes in an indirect and less visible way. In those cases, a vulnerable library is only used through another library that is directly and visibly used with a code base. In particular, the logging libraries are commonly pulled in without much fanfare and with a risk of causing unknown vulnerability to exploits.

Another finding is that some commonly vulnerable Java libraries are used as an unmanaged Jar; a bad practice from several angles. In those cases, the library code itself is distributed inside the code base of a software system, without the use of a package management tool. Such libraries are generally updated slower and hence vulnerable for longer than properly managed libraries[15].

The ***SIG recommended practice*** is to urgently check for vulnerable versions of our top 10 critically vulnerable libraries. This requires looking at both the directly used open source packages and those used indirectly. Further, implement our recommended practices to get in control of vulnerable open source. Those are discussed in detail in the third section of this chapter.

[14] https://blog.checkpoint.com/2023/01/05/38-increase-in-2022-global-cyberattacks/

[15] https://www.enisa.europa.eu/news/enisa-news/understanding-the-increase-in-supply-chain-security-attacks

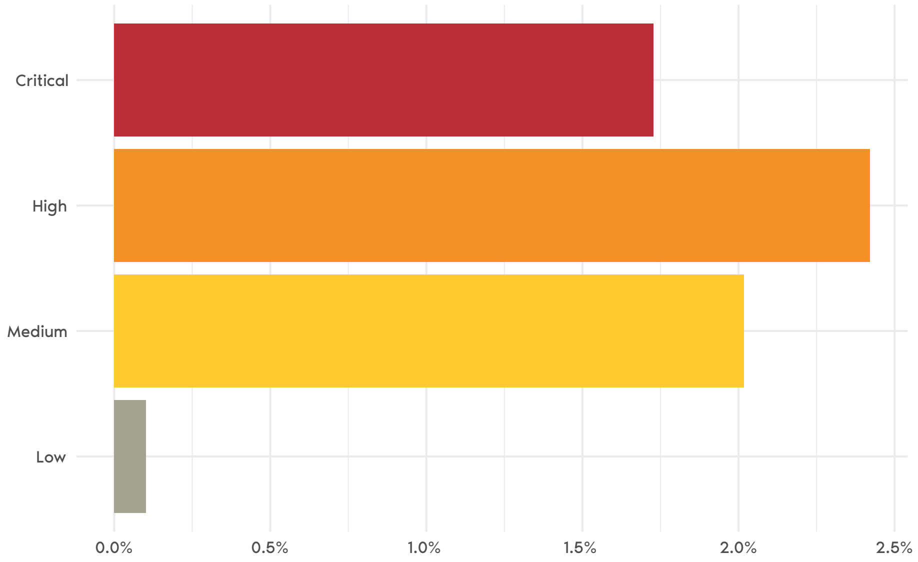
[16] As shown in our Benchmark Report 2022, page 23

2. Reduce risk in open source usage by addressing build quality

Open source ecosystem hosting vendors such as SonaType provide yearly updates on the state of their managed ecosystems. Over 2022, they reported that 14% of downloaded libraries were vulnerable. Often that's because the downloaded version was outdated and, while known to be vulnerable, was still available for download. Also, they report that 6 out of 7 vulnerable libraries are downloaded as indirect dependencies.

Turning to our data, we observe an overall more positive picture. The following graph shows the percentage of vulnerable libraries used in enterprise software observed by SIG. The good news is that the overall rate is well below the SonaType-reported 14%, in particular for the critically severe ones (at just 1-2%). So, the enterprise software systems we observe with Sigrid are using fewer vulnerable versions than the general software public.

VULNERABLE LIBRARIES ACROSS THE ENTERPRISE SOFTWARE INDUSTRY
Based on 106K dependencies seen in 2022

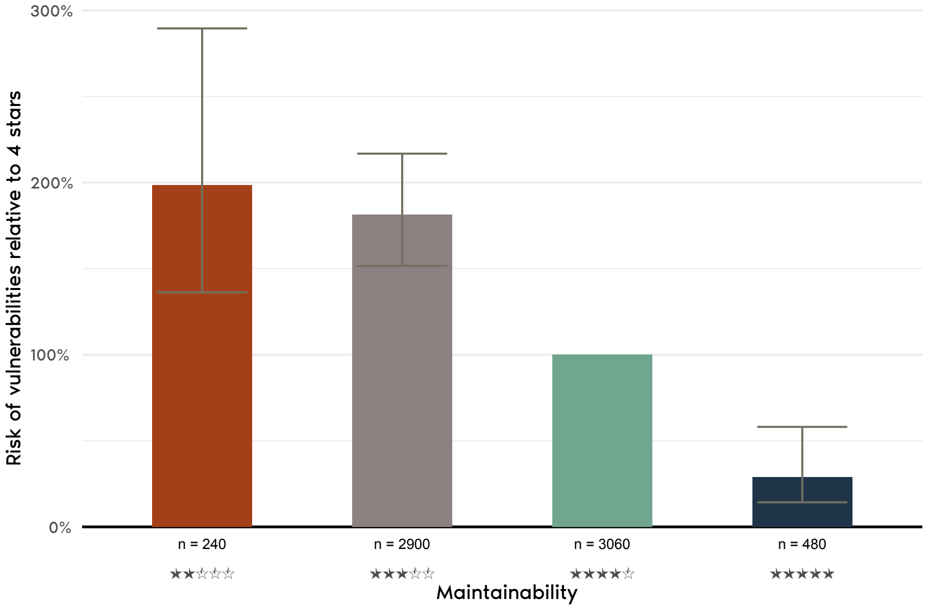


However, the bad news is that typical enterprise software systems still use hundreds of libraries, either directly or indirectly. So even a small percentage of vulnerable libraries increases the risk of security issues in enterprise software systems.

In last year's Benchmark Report, we showed a clear correlation between the risk that a library has vulnerable versions and its build quality. To further corroborate that result, we extended our dataset with a factor of 5 by including both more Java Maven libraries and also Python libraries from the PyPI ecosystem. In total we are now reporting on approximately 10,000 libraries: again we see a clear higher risk related to having lower maintainable code. High code maintainability is a major underpinning factor to overall build quality, facilitating the code to be understood, changed, tested, and reused.

In the graph below we show the risk of a library being vulnerable at each build quality level, relative to the recommended level of 4-star SIG maintainability[16]. Clearly, libraries of higher, 5-star quality show less than half of the risk, while 3- and 2-star maintainability have up to two times higher risk levels[17].

RELATIVE VULNERABILITY RISK COMPARED TO RECOMMENDED BUILD QUALITY (4 STARS)



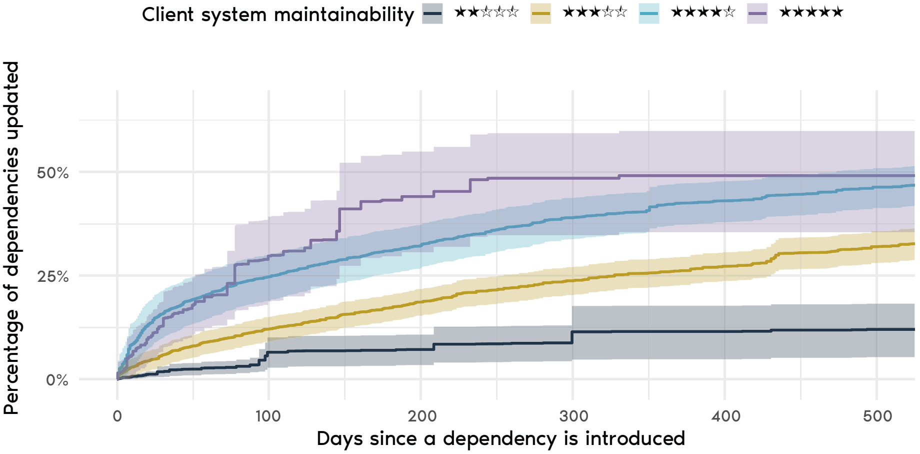
[16] https://www.softwareimprovementgroup.com/software-analysis/
[17] Libraries of 1-star Maintainability are not yet available in the dataset

Key Finding: Compared to recommended 4-star Maintainability, 2-star Java and Python libraries have 2x more risk of having vulnerable versions. At the same time, exceptional quality 5-star libraries show only a quarter of the risk.

The SIG Maintainability measurement is often correlated with other best software development practices. To name just a few core practices: automated unit testing, modern (security) code review, and the use of automated tools for continuous integration and deployment. One reason for the correlation is that such best practices are hard or impossible to implement in code bases of low quality. Such code bases resist being understood, tested, and changed.

Teams that manage to maintain high levels of maintainability implement such best practices and therefore reap other benefits of quality as well, including lower defect rates and shorter fix times. Among such benefits is a reduced time to update open source libraries. Let's have a look at our maintainability measurement data for 3,500 enterprise software systems and their 326K open source dependencies.

TIME-TO-UPDATE VERSUS MAINTAINABILITY
Tracking 326K dependencies from 18 ecosystems in 3500 client systems



[prompt] future fit software environment sport setting outside blue colored racetrack with a feeling of a modern office environment colors with people running

In the plot on the page before, we show longitudinal data for the time-to-update in days since a dependency was introduced. In short, the higher the graph, the quicker dependencies were updated to newer versions. It's clear to see that 5- and 4-star code wins the race, while 3-star code lags behind and never really catches up. 2-star code is often older and generally uses a stagnant set of libraries. Given that the libraries themselves tend to evolve quickly, and vulnerabilities are discovered at an increasing pace, stagnation of update speed is a genuine concern.

3. Closing the door on vulnerable open source dependencies

Let's review our thoughts and general recommendations to move toward secure and accountable development. The governing bodies of the United States and the European Union are preparing to introduce far-reaching legislation regarding cybersecurity. The software-producing industry will need to respond by adhering to unprecedented standards, with matching fines or liability claims for failure.

There are three main action areas to address:

1. *Establish a duty of care based on current security standards.* Increase transparency on software products by providing SBOMs, plus threat and mitigation analysis.

2. *Implement practices to get in control of vulnerable open source.* Review our list of 9 open source usage practices and implement the missing ones in daily development practice.

3. *Review SIG R&D on a novel requirements-driven approach in secure development.* Get in touch with us on the SCRAMBLE project, where SIG is creating secure code review innovations.

[prompt] picture of great software design showing from inside a computer

Establish a duty of care based on current security standards

In order for the software industry to prevent countless lawsuits and fines, it is time to start taking responsibility by building security in from the start. Many organizations are working on this but struggle for different reasons, including that there is no clear and shared duty of care in the industry. When is software secure enough?

One challenge we face in answering this is the complex and fragmented landscape of security standards. Standard makers such as the ISO see this and seek ways to harmonize. SIG is determined to help make this happen, for a large part by building on SIG's proven ISO/IEC 25010 software security model and supporting harmonization efforts. For example, we donated our model to the OpenCRE[18] open source initiative to link security standards together in a uniform framework.

While legislation is being drafted and discussed by lawmakers and industry representatives, the question arises of what software producers should be doing today. Should every software product be subjected to certification under the existing Common Criteria[19] or the forthcoming EUCC[20] model? For critical products, there are good arguments to warrant the cost of such certification processes.

However, for the vast majority of software products that burden of certification may be prohibitive. What are more pragmatic approaches that the regular software producer could adopt? Providing transparency would be a key practice that could perhaps alleviate the need for innovation-stifling legal requirements. Indeed, software-bills-of-material (SBOM) are becoming commonplace in the software industry, but those are not enough.

SIG is making efforts to urge the software industry to action. For this, the right incentives could be provided by the right legislation. However, requiring software makers to adhere to strict security standards could reduce the freedom to innovate. Instead, we would like to see that the industry can showcase how secure their product is so that buyers can decide. The responsibility of ensuring secure software is then moved to the buyers provided that the software producers have done their job regarding assurance.

If software producers would complete their products with a transparent security analysis, buyers would be able to better inform their decisions to acquire and implement. In our view, next to an SBOM, there should be a clear analysis of the threat model and applied mitigations, in addition to the secure development practices employed. With that information, a producer clearly defines its scope of accountability, while a software buyer would be better positioned to take responsibility for secure implementation.

[18] https://www.opencre.org/
[19] https://www.commoncriteriaportal.org/
[20] https://www.enisa.europa.eu/publications/cybersecurity-certification-eucc-candidate-scheme

As an example of an effective security prescription, we would like to point out The Update Framework (TUF)'s threat analysis[21]. By providing transparency on the perceived threats and implemented mitigations, together with external audit reports, the users of TUF can implement the product more securely within their own scope of responsibility.

Implement practices to get in control of software security issues in general

In addition to legislative compliance and implementing duty of care, there are several smaller steps to take that will help stem the tide. The following are some practical and recommended directions to get in control of open source vulnerabilities:

1. Implement continuously up-to-date Software Bills of Material (SBOM) across the entire software portfolio. This can be achieved by automated software composition analysis tooling.

2. Define or refresh dependency version update policies to prevent unmitigated use of stale and vulnerable dependencies. Such policies should have a clear timeframe for addressing vulnerable dependencies, make clear what mitigation steps are acceptable, and allocate responsibility. Embed these policies into the software development lifecycle with automated tools.

3. Review actual in-use dependencies by considering whether they are essential for operations or largely redundant, reducing attack surface as much as possible. In-use vulnerable dependencies should be brought in line with policies as soon as possible.

4. Identify any dependencies with low build quality and reconsider if they are worth the higher risk of future vulnerabilities. Prioritize the replacement of dependencies that are no longer maintained actively or have a very small group of users.

5. Ensure that the build quality of your own software portfolio is up to standards (4-stars) to facilitate development best practices, including testing, and to enable fast dependency update policies.

6. Provide standard building blocks and frameworks to teams in order to take care of many of the security requirements.

[21] https://theupdateframework.io/security/

7. Instruct teams on security requirements with a combination of training, coding guidelines, and continuous knowledge exchange. To deal with a large number of requirements, it is essential to implement a process to attach the relevant instructions and verifications to individual tasks (e.g. stories[22]).

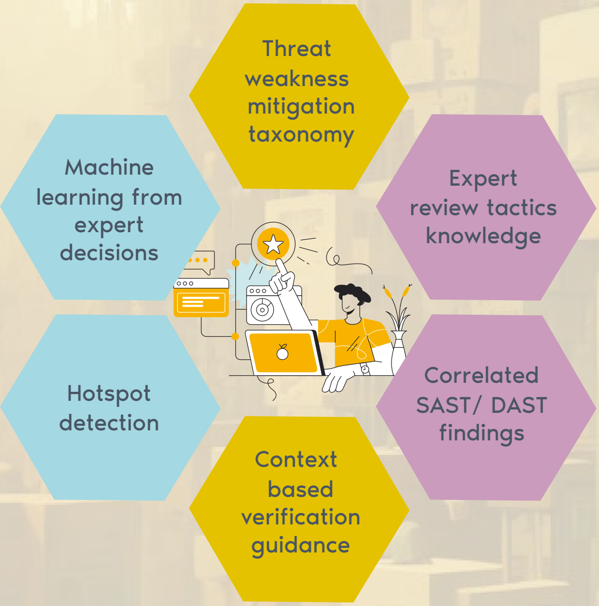
8. Perform automated static and dynamic security verification throughout the software development life cycle on all produced code to find potential weaknesses before they become exploitable. In best practice, tools of different classes (SAST, DAST, SCA, and IAST) are combined into an automated security analysis suite.

9. Because automated tools have inherent blind spots. Introduce tool-supported security code review for critical systems that address security-by-design, common weaknesses in code, mitigation strategies, and unsafe usage of APIs or dependencies.

[prompt] a photographic image of a poorly constructed building showing locks at doors

[22] https://owaspsamm.org/guidance/agile/

PROJECT HIGHLIGHT


Threat weakness mitigation taxonomy

Machine learning from expert decisions

Expert review tactics knowledge

Hotspot detection

Correlated SAST/ DAST findings

Context based verification guidance

SCRAMBLE

Smart Code Review Assistance Module
Blending Leading Expertise

Project SCRAMBLE takes place at SIG under a one-year government-funded program until July 2023, in collaboration with Radboud University and Netherlands Organisation for Applied Scientific Research (TNO).

[prompt] visual of the concept of scramble in the programming of digital solutions in the style of an old painting

Apply a requirement-driven approach to security

Regulations and the duty of care require organizations to build in security, based on standards. In order to do this, the functional and non-functional requirements in these standards need to be the basis of the work of developers and testers. Applying security requirements effectively is currently an unsolved problem - especially in agile software development. How can the large and complex set of requirements be provided in such a way that developers can apply them effectively, and how can the verification work for these requirements be done efficiently, given that only part of the testing work can be automated?

To help apply a requirement-driven approach, SIG is developing a unique intelligent platform under the working title SCRAMBLE. This platform provides instructions to teams for writing secure code and performing the right verifications. This verification step includes manual code review since many types of security flaws cannot be found by tools alone. The problem is that expertise in secure code review is scarce, while the work is very time-consuming.

SCRAMBLE addresses these issues by using AI to harness methodology and world-leading review expertise at SIG. An expert system guides reviewers in performing verifications, while a machine learning model provides recommendations regarding places to look in the code, and how to make decisions. At SIG, we believe that AI-assisted developers and testers are the future.

The current prototype is already being used in client assignments and has led to a substantial increase in efficiency, quality of work, consistency, and availability to a larger group of experts - users at SIG, but we are also taking steps to involve users at clients and partners.

SCRAMBLE represents the top-down approach to software security and addresses the unsolved problem of effectively managing security requirements in the software lifecycle. SCRAMBLE manages these requirements as instructions to engineering teams and assures their verification:

- by SAST, DAST, SCA, and IAST tools,
- by fuzzing tools,
- and by manual review.

This builds on the idea that we also promote through our Sigrid platform: allowing organizations to apply an approach in the business context: a risk-driven and cost-based way of dealing with security, instead of the bottom-up approach of having to deal with thousands of tool findings that need to be fixed.

Wouter Knigge / Edward Song / Magiel Bruntink / Xander Schrijen

FIRST DIGITAL SKILLS BENCHMARK SHOWS POOR JOB ALIGNMENT

5

At the end of 2022, Astride began, and since then, over 5,500 people in over 180 countries have taken their Digital Skills Assessment which has provided professionals with insight into how they relate to their current and potential future job roles and has provided us with a unique view on global digital skills across industries. Our major finding is that we clearly see that digital skills are very poorly met. The average skill gap for the vast majority of roles is more than 50%. There is significant room for upskilling, to put it mildly.

[prompt] a photo-realistic picture of a happy casual man with eight arms and hands with digital devices in it

1. Job market alert: a desperate need for digital skills

We are in an era of constant disruption, and the IT and Software industry is leading the charge. Over the past three decades, technological advancements have significantly accelerated, reshaping the ways we communicate and connect. Consider the smartphone revolution, which transformed and mobilized our means of communication. This phenomenon is not an isolated incident. Disruptive leaps in technology, such as the robotization of the manufacturing sector, have consistently replaced earlier methods at an ever-increasing pace.

However, is the workforce keeping up? Over the past decade, discussions surrounding the "Talent Shortage" in IT have persisted. The rate of change does not correspond with the number of skilled professionals entering the job market. For example, the software engineering industry's shortage of technical personnel is growing at an alarming rate, with the security niche leading the pack[23]. Moreover, the existing workforce requires continuous upskilling to remain competitive and relevant.

Demographic data indicate that waiting for new graduates (university/college/tech programs) will not solve this problem[24]. Instead, there is a pressing need (and opportunity) to repurpose the existing global workforce to address these demands. The crucial question remains: how do we initiate this process?

This chapter explores the dataset we have accumulated with Astride over six months and with more than 5,500 participants. Our benchmark analysis will cover the following key points:

- *Employer's perspective:* are people equipped for the jobs they are currently in? No, on average, there is a high skill gap for practitioners in their current jobs, with only a few exceptions.

- *Candidate's perspective:* what jobs are easy to get into, and which have good follow-up opportunities? The data show plenty of job entry opportunities for the Astride participants, in particular for those with specialized and demanding jobs.

[23] https://www.forrester.com/report/the-security-skills-shortage-takes-its-toll-on-organizations/RES178724
[24] https://gaper.io/tech-talent-shortage/

2. Astride – The digital skills compass for tomorrow's learning journey

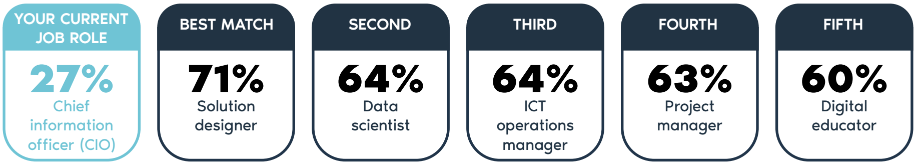
Achieving equilibrium in the digital skills marketplace necessitates aligning supply and demand. Both candidates and employers require insight into their respective positions and potential actions:

- *Candidates* – need to understand their skill levels, alignment with specific job profiles, and the areas requiring further development.
- *Employers* - must evaluate their current workforce's skill coverage, team/departmental balance, and the external skills they must shop the job market for.

With nearly 40 years of experience creating competency and skills-based certifications, EXIN is ideally positioned to provide such insights. EXIN has developed Astride, a self-assessment portal based on an internationally recognized scientific competency framework that connects 121 competencies to 31 job profiles in the digital skills market (the e-CF)[25].

Astride[26] enables candidates to self-assess their competencies in six areas. Each participant provides their country of origin, current job role, and industry. Candidates can opt to skip competency areas if they lack relevant experience. The assessment generates 250+ weighted data points applied to specific competencies.

After completing the assessment, Astride generates a Custom Insights Report for each candidate, including the summary below. A candidate's compatibility with a job role is represented by the scores on up to 10 selected competencies, which are averaged to produce a single percentage.



YOUR CURRENT JOB ROLE	BEST MATCH	SECOND	THIRD	FOURTH	FIFTH
27%	71%	64%	64%	63%	60%
Chief information officer (CIO)	Solution designer	Data scientist	ICT operations manager	Project manager	Digital educator

[25] https://esco.ec.europa.eu/en/about-esco/escopedia/escopedia/european-e-competence-framework-e-cf
[26] https://www.exin.com/astride-by-exin/

3. Insights from the Astride Benchmark on the current job market

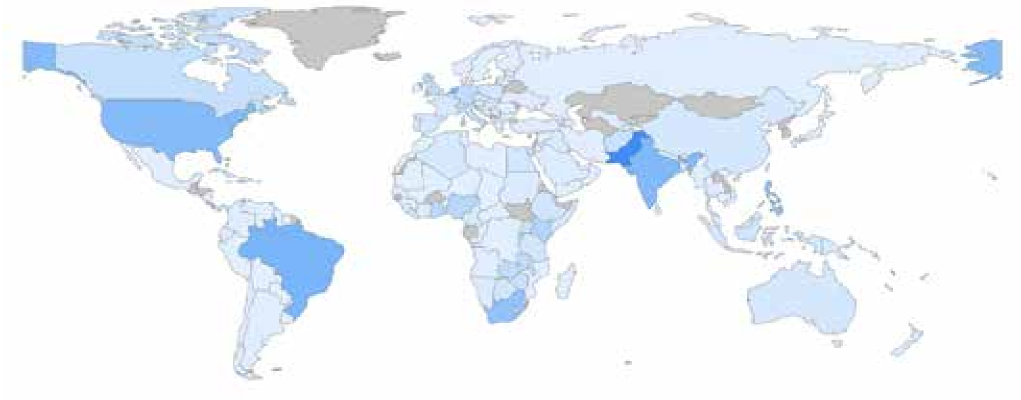
Drawing from the anonymized self-assessment data, we can examine the current trends in the global job market. In addition to the skill assessments, EXIN collects information on participants' country of residence and industry, allowing for contextual analysis. Hence, our dataset can be utilized to combine findings into insights, as represented below.

This Benchmark Report provides a few key findings out of a deeper analysis that will soon become available as an EXIN whitepaper. Stay tuned on EXIN's LinkedIn[27] for an update on that.

Key Finding: 5,500 IT practitioners from across the globe have used Astride so far.

- **180 out of 195 countries are represented,**
- **Most common industry is IT / Software, but many more specific industries are represented,**
- **Most respondents indicated "Project Manager" as their job role.**

[27] https://www.linkedin.com/school/exin/



Employer’s perspective: poor alignment between jobs and digital skills

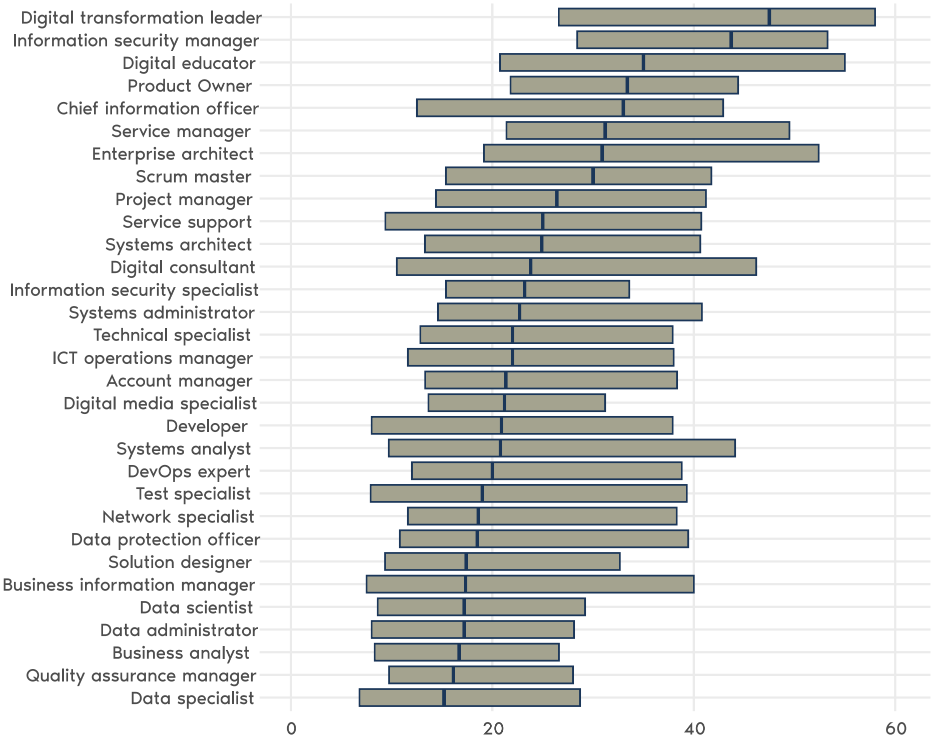
Astride assesses to what extent participants have the skill they need for their current and future jobs. With the data collected on 5,500 assessments, let’s see how well job roles are currently filled with people that have the right skills. As the next analysis shows, the overall picture is not so rosy, indeed.

Key Finding: Except for people in the jobs of *Digital transformation leader* and *Information security manager*, the average Astride score for job roles shows a skill gap of at least 50%. The average *Developer* has a 75% gap with having all skills for the job.

For each job role assessed, the graph shows the average Astride score (dark blue line) inside the middle 50% scores (gray box). The maximum Astride score a participant can get for their current job is 80 points. While there are no doubt skilled individuals in all of the assessed jobs, this analysis focuses on the average cases to allow organizations to reflect on recruitment, training, and retainment policy.

Looking at the graph, it's obvious that people in most job roles score far below 40 points on average, implying a skill gap of higher than 50%. The only exceptions are people in the jobs of Digital *transformation leader* and *Information security manager*. The average *Developer* even has a 75% gap with having all skills for the job, with an average Astride score of about 20.

AVERAGE ASTRIDE JOB-SKILL MATCH SCORE



The Astride data reveal that the top-3 missing skills for *Developers* are the following:

1. Component Integration (level 2),
2. Testing (level 2),
3. Documentation Production (level 3).

These findings should be of concern to IT organizations. Apparently, many people are only marginally skilled for the jobs they are currently active in. Either they expect to be trained on the job, or they expect to move on quickly to other jobs that are more suitable to their skill sets.

For organizations, the priority should be to identify what skills are most often lacking, given the organization's specific profile of job roles. Then, revisit training, recruitment, and retainment policies, to gradually decrease skill gaps.

Job candidate's perspective: job entry and follow-up opportunities

The days of any one individual staying in the same role for many years are gone. From individuals to corporate employers and government agencies, job market mobility is essential[28].

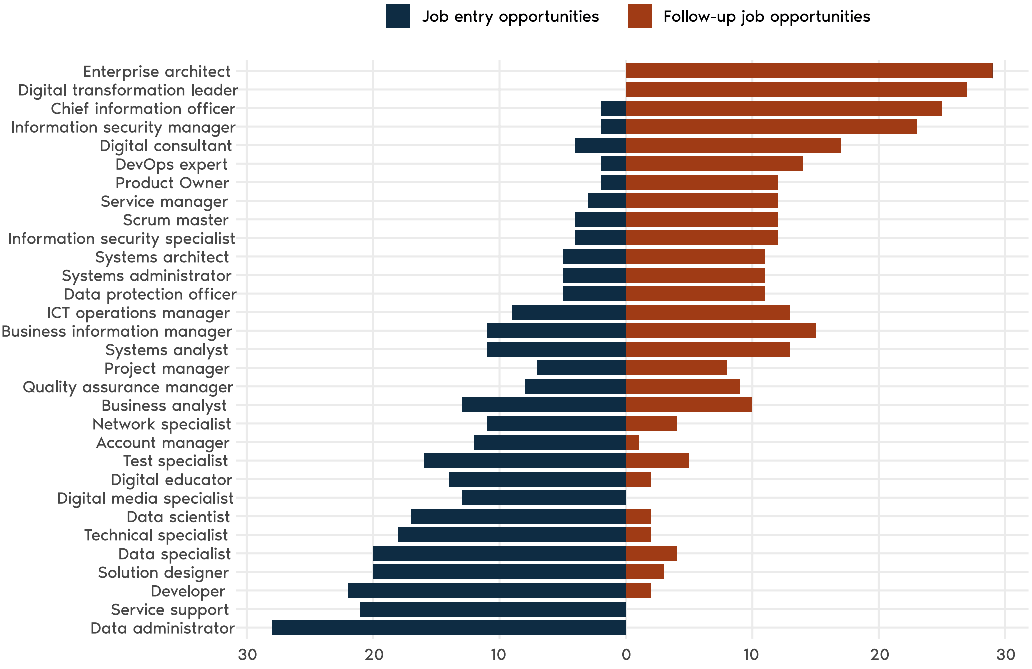
- From an individual standpoint, growth prospects are a significant factor in job mobility. As employees develop skills over time, career opportunities and salary growth become available, allowing for variation in the different activities throughout their careers.
- For employers, mobility enables organic employee growth within the company, utilizing skills and knowledge acquired over time. High job mobility in the market also ensures the availability of attractive candidates for new positions.

Astride assesses what skills people currently possess and what the requirements for each job are. The data can also tell whether the people in each job have skills that also apply to other jobs. For instance, if you are in a data science role but have more development skills than the average developer, then a developer role could be a good follow-up opportunity. The next graph reflects the overall job mobility picture that follows from the Astride data:

- People in *Enterprise Architect* and *Digital Transformation leader* jobs are the most densely skilled: on average, they are out-skilling people in most other jobs. At the same time, this makes their jobs harder to enter for others and provides them with many options for the next job. *Enterprise Architects* in general did not show a great skill match in the previous analysis due to the job demanding a greater skill set than others.
- At the other end of the spectrum, there are plenty of jobs that are relatively easy to get into for most respondents. For instance, the average skill sets for *Service support* and *Data administrato*r jobs are exceeded quite easily, allowing candidates a starting position.

[28] https://www.gartner.com/en/newsroom/press-releases/2020-02-27-gartner-says-hr-leaders-must-build-a-robust-strategy-

JOB MOBILITY OPPORTUNITIES BASED ON ASTRIDE SKILL DATA



Of course, the growth path of an individual candidate can be different. The Astride benchmark data reflects the average skills in each job, giving professionals options for further growth. Candidates should ensure that they find jobs that grow their skill sets and find organizations that provide them with the necessary training.

Key Finding: *Enterprise Architect* and *Digital Transformation leader* jobs are the most densely skilled: on average, they are out-skilling people in most other jobs.

[prompt] a copyright sign in the sky in a cloud of binary code in Vermeer style

Luc Brandts

FINAL THOUGHTS

Now that we are at the end of our fifth SIG Benchmark Report, it is clear that the digital world needs to get its act into gear, everybody. The issues we revealed are major issues requiring a concerted effort to resolve.

There is a lot more underlying data we can share to build your own specific case, and we are more than happy to assist.

In our report, we shared the most important findings. There is something for everybody to act upon. Whether it is to upskill you or your team's digital skills, make the promise of low-code work, or ensure that your great AI project is not becoming a legacy nightmare. Or, how you can make sure open source is not the open door it seems to be: in fact, today it’s a whole warehouse of open doors, so big that it's scary.

A lot is happening in the digital world, not in the least by generative AI solutions that are getting impressively strong. Of course, this will be of great help, but it will also create new challenges. For one, is the code that is being generated not an (unintentional) copyright infringement of a previously created piece of software? And you still need to tell the generative AI what you want to have. The specification in natural language is not necessarily without its challenges, as everybody knows who has ever been in a conversation and was not immediately understood (if you don’t recognize this, that is even scarier).

So, the digital world is progressing fast, a lot is happening, and there is still a lot to be fixed.

In this report, we gave you the guidelines to focus your short-term efforts and drive your long-term plans. Let us know where we can help with our data, insights, and technology.

We’re at the end of the report with nothing more to read but all the more to do!

Thanks for your attention and focus. We loved writing this and hope you enjoyed reading it, and most importantly, we hope it is of value to you.

On behalf of the entire team of authors,

Luc Brandts and Magiel Bruntink

GETTING SOFTWARE RIGHT FOR A HEALTHIER DIGITAL WORLD

[prompt] a photo-realistic picture in a happy setting of a smiling little girl dressed as a hip programmer working on a laptop in fresh colors

About Software Improvement Group

Software Improvement Group (SIG) helps organizations trust the technology they depend on. We've made it our mission to get software right for a healthier digital world by combining our intelligent technology with our human expertise to dig deep into the build quality of enterprise software and architecture - measuring, monitoring, and benchmarking it against the world's largest software analysis database.

With SIG software assurance, organizations can surface the factors driving software total cost of ownership and make fact-based decisions to cut costs, reduce risk, speed time to market, and accelerate digital transformation.

Software Improvement Group is the first fully certified laboratory in the world to measure against the ISO 25010 standard. We make this lab accessible to our clients through our SaaS software assurance platform – Sigrid® – which enables them to take a risk-based approach to improving the health of their IT landscapes.

We serve clients spanning the globe in every industry, including DHL, Philips, ING, KLM, BTPN, Weltbild, KPN, as well as leading European governmental organizations.

SIG was founded in 2000 as an independent technology company with embedded consulting services. SIG is headquartered in Amsterdam, with offices in New York, Copenhagen, Antwerp and Frankfurt.

Learn more at www.softwareimprovementgroup.com.

Software
Improvement
Group

Fred. Roeskestraat 115
1076 EE Amsterdam
The Netherlands

www.softwareimprovementgroup.com
marketing@softwareimprovementgroup.com

COLOFON

SIG Benchmark Report | 2023
Enterprise software through the SIG looking glass

Authors

- Asma Oualmakran, SIG
- Benedetta Lavarone, SIG
- Chushu Gao, SIG
- Dennis Bijlsma, SIG
- Edward Song, EXIN
- Lodewijk Bergmans, SIG
- Luc Brandts, SIG
- Magiel Bruntink, SIG
- Miroslav Zivkovic, SIG
- Pepijn van de Kamp, SIG
- Rob van der Veer, SIG
- Xander Schrijen, SIG
- Wouter Knigge, EXIN

Legal Notice

This document may be part of a written agreement between Software Improvement Group (SIG) and its customer, in which case the terms and conditions of that agreement apply hereto. In the event that this document was provided by SIG without any reference to a written agreement with SIG, to the maximum extent permitted by applicable law this document and its contents are provided as general information 'as-is' only, which may not be accurate, correct and/or complete and SIG shall not be responsible for any damage or loss of any nature related thereto.

Design: Plushommes.com - Martijn Meerman
All illustrations are made with the help of AI (Midjourney)

Software
Improvement
Group

Fred. Roeskestraat 115
1076 EE Amsterdam
The Netherlands