Grid Dynamics Whitepaper

Evaluating the impact of GenAI on enterprise software development

www.griddynamics.com

Grid Dynamics

Contents

Executive summary **4**

What aspects of software development does GenAI impact? 4
How should software development organizations approach GenAI adoption? 6
How does GenAI impact developer productivity at the task level? 7
How does GenAI impact productivity at the project level? 8
Can GenAI help to release resources from ongoing projects? 9
What should the released resources be used for? 10
How should GenAI tool selection be approached? 11
What are the notable limitations of GenAI-powered development? 13

Accelerating individual development tasks using GenAI **14**

Code translation 14
Code writing 16
Unit test writing 19
API schema generation 21
Data access layer creation 22
Deployment script writing 23
Performance analysis and optimization 25
API documentation writing 27
Additional notes on tool usage techniques 28

Transforming the development process using GenAI **29**

Legacy migration 29
New application development 32
Existing application modification 34

Conclusions **36**

Appendix: Evaluation methodology **37**

Evaluation process 37
Task design 38
Task evaluation 39
Tools 39

About Grid Dynamics **40**

Generative AI (GenAI) is widely perceived as a technology that can drastically transform the software development industry–from optimizing day-to-day coding techniques to improving market positions of major IT companies. At Grid Dynamics, we conducted a study to quantitatively evaluate the performance gains delivered by GenAI tools for various types of software development tasks and project types. Concurrently, we assessed the organizational and process implications of these tools, and formulated recommendations for GenAI adoption by engineering teams. The study involved about 50 engineers performing extensive hands-on evaluation, culminating in the broad adoption of GenAI tools across the company.

We present the main findings in the Executive summary, focusing on the strategic potential and implications of GenAI adoption. The subsequent sections delve into the precise details of how GenAI can be applied to individual development tasks and various project types.

Executive summary

Most GenAI tools and usage techniques aim to improve developer productivity within specific tasks. Moreover, these improvements eventually accumulate and drive broader changes in development processes, infrastructure, and talent management too. In this section, we assess the impact of GenAI at each of these levels.

What aspects of software development does GenAI impact?

GenAI impacts software development organizations at multiple levels:

- **Task execution:** GenAI changes the way developers, QA engineers, tech writers, and other specialists perform individual tasks such as business logic modification or unit test writing.
- **Team composition:** GenAI tools can be used by junior, middle, and senior engineers to different extents and with different productivity gains. This makes changes in team composition and redistribution of talent across the teams and projects possible.
- **Development process and scope:** GenAI tools facilitate onboarding through accelerated comprehension of unfamiliar codebases for developers. This generally reduces the onboarding time and enables individual developers to work across a broader range of tasks, components, and projects.
- **Project roadmaps:** The impact of GenAI significantly varies across different project types such as backend and frontend development. This generally requires organizations to revisit their roadmaps, identify opportunities to do more in a shorter time and at lower costs, and reallocate the resources accordingly.
- **Tools:** Applicability of GenAI and productivity gains depend significantly on GenAI tools and how these new tools are integrated with Integrated Development Environments (IDEs) and other traditional tools. Research of tools and corresponding usage techniques stand as pivotal aspects of GenAI adoption.

It is also important that GenAI impacts virtually all software development specializations, so each specialization (practice) needs to master GenAI-assisted development and establish new operational standards. This can be illustrated with the following examples:

Application development

- As-you-type code completion
- Method generation based on natural language description
- Automatic error fixing
- Automatic performance fixing
- Question answering about the codebase and APIs

No-code and low-code development

- No-code application creation
- No-code legacy and cloud-to-cloud migration
- No/low-code deployment flow management
- No/low-code UI layout creation
- More powerful building blocks (chatbots, search, etc.)

Data engineering

- Data model creation
- Legacy data pipelines migration
- Assisted data discovery and explanation
- Assisted SQL writing
- SQL generation based on natural language
- SQL explanation and optimization
- Question answering about cloud data services and APIs

DevOps and CI/CD

- Question answering about cloud services
- Deployment scripts creation
- Advanced pre-commit checks (code readability, performance, etc.)
- Automatic defect triage, assignment, and fixing

Quality assurance

- Unit test writing
- Functional tests writing
- Integration test writing
- Test data creation

Security

- Codebase analysis for security and compliance breaches
- Automatic alerts on security breaches, news monitoring
- Guidance on threat remediation
- Automatic dependency upgrades

Documentation

- API reference documentation writing
- User guides writing
- Code examples creation

Infrastructure and operations

- Recommendations on cloud cost optimization
- Recommendations on architecture optimization
- Automatic incident triage and assignment
- Guidance on issues resolution

Some of these use cases are widely recognized and supported by a broad range of tools, but many of them have limited support in the existing tooling, or are limited by the current capabilities of the backing large language models (LLMs). This creates an opportunity to achieve competitive advantage through early adoption of advanced or non-trivial GenAI tools and techniques.

How should software development organizations approach GenAI adoption?

Your GenAI adoption strategy should address two crucial strategic aspects: what tools should be used and what organizational changes should be made; and tactical steps covering who will mentor the developers and how this process can be scaled. We recommend the following adoption strategy that reflects these considerations as well as the multifaceted impact of GenAI:

1. **Project profile identification:** Identify task categories and project profiles that are most common or important in your organization. For example, it can be mainframe legacy migration, data pipeline development, or backend development.
2. **Create working groups:** For each profile, create a working group to research the applicability of GenAI for a particular profile in the context of a particular organization and develop the adoption plan. The members of the working groups can later become GenAI evangelists and adoption facilitators.
3. **Define tooling and security policies:** Define what tools can be used for what purposes, and what the security constraints are. Plan how tools and policies can evolve over time. For example, you can start with 3rd party services and later switch to internal GenAI models.
4. **Start pilot projects:** Use the results produced by the working groups to start short-term pilot projects. These projects should aim to validate the hypothesis created by the working groups, and train a broader range of engineers in GenAI usage techniques.
5. **Plan strategic changes:** Use the results produced by the working groups to plan strategic changes–what tools should be leveraged for what tasks? How should the processes be changed? What new projects should be started? What resources can be released? What should these resources be used for?
6. **Create a GenAI CoE:** Transform the working groups into one or several centers of excellence (CoE). The GenAI CoE should develop the infrastructure (tools), help individual teams to adopt GenAI, and establish best practices.
7. **Scale up:** Create training materials for engineers and guidelines for engineering managers. Monitor the progress and outcomes.

How does GenAI impact developer productivity at the task level?

The impact of GenAI is not uniform across various development tasks. To create a quantitative picture, we defined 8 standard development tasks and measured the following metrics for each of them through our crowdsourcing study:

Productivity

How the total level of effort (man-hours) reduces with GenAI tools.

Applicability

The typical percentage of tasks in a real-world project that can benefit from GenAI tools.

Usefulness

How the survey participants ranked the usefulness of GenAI for solving each task, on a scale from 1 to 100.

The measurement results are summarized in the following table:

Task *	Productivity	Applicability	Usefulness
Code translation	**15–90% ****	**60-70%**	**50-85%**
Code writing	**6–40% *****	**75-85%**	**40-100%**
Unit test and test data creation	**35–70%**	**80-90%**	**40-90%**
API schema creation and maintenance	**35–60%**	**80-90%**	**75%**
Data access layer creation	**40–60%**	**80-90%**	**90%**
Deployment scripts creation/migration	**20-25%**	**20%**	**70%**
Performance analysis and optimization	**10-20%**	**20%**	**60%**
API documentation writing	**70-90%**	**90%**	**90%**

(*) See the *Accelerating individual development tasks using GenAI* section for task definitions and detailed results.

(**) The gain heavily depends on the tool usage techniques. Basic methods tend to produce poor results, while advanced techniques discussed later in the report deliver major gains.

(***) The gains greatly vary depending on task complexity. See the *Accelerating individual development tasks using GenAI* section for more details.

We found that GenAI tools are particularly efficient for test automation and routine API documentation writing tasks, where the gains can reach **75-90%, which is 4-10x acceleration**. Some other use cases such as code translation can be more challenging, but the baseline results presented on the previous page can be improved using advanced techniques.

In real-world projects, multiple tasks need to be performed, and these tasks consume different percentages of time and effort. Figure 1 visualizes how the development effort within a typical project is redistributed after GenAI tools adoption. On the left-hand side, the typical distribution of effort across the tasks is shown. On the right-hand side, the productivity gains are shown. This takes into account how often different tasks are being faced, how applicable AI tools could be for those tasks, and how impactful the usage of AI tools is on the said task. We estimate the average gain for a typical project to be around **28%**.

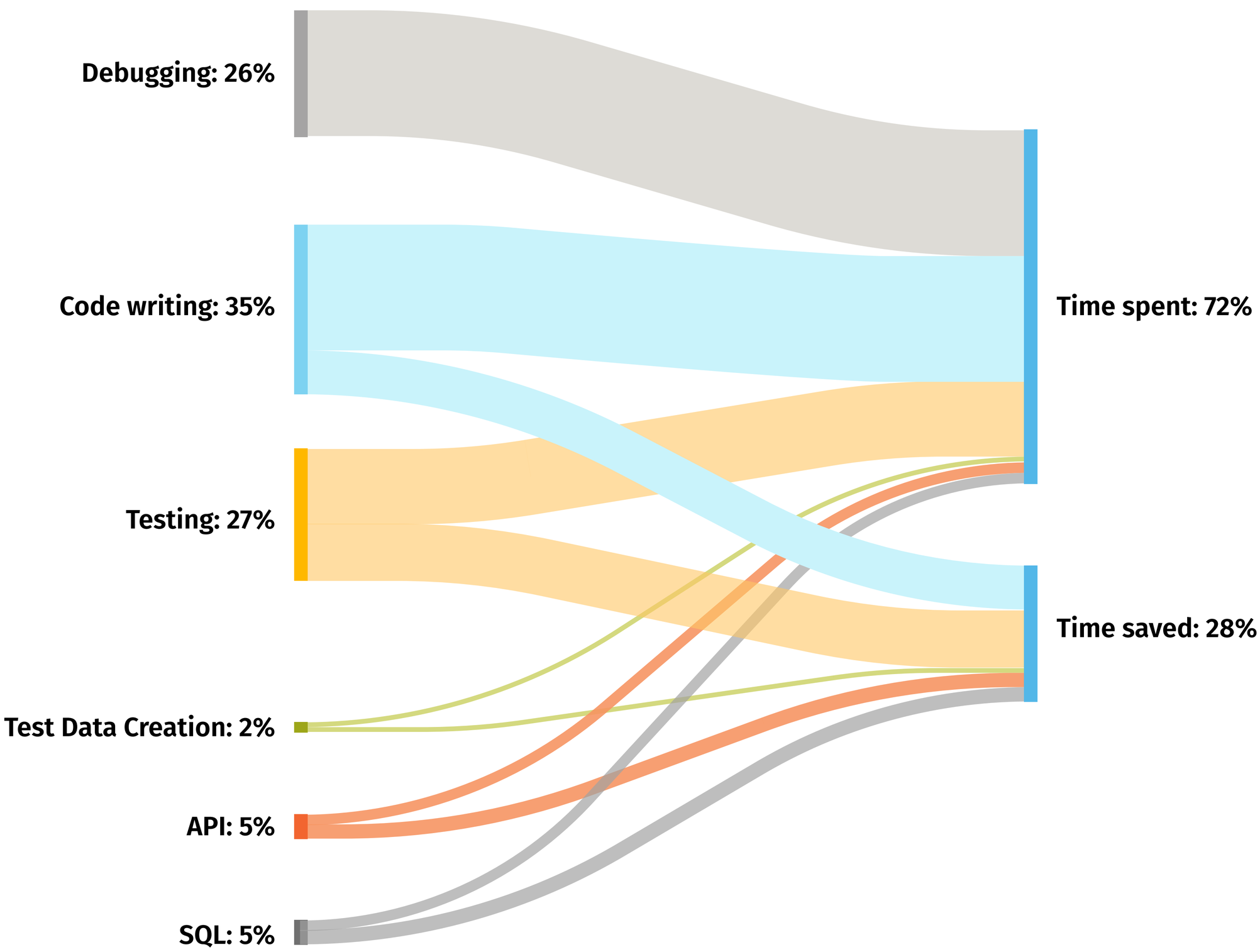


Figure 1. Productivity improvements at the level of tasks. API stands for "API schema generation", SQL for "SQL Schema generation".

How does GenAI impact productivity at the project level?

The productivity gains vary across different project types such as migration and new development. We defined four major project profiles and estimated the productivity gains for them as follows:

Project type *	Productivity
Legacy migration	**15–80%**
New backend application development	**20–30%**
New frontend application development	**20–30%**
Existing application modification	**25–35%**

(*) See the *Transforming the development process using GenAI* section and *Appendix* for detailed project type definitions.

Can GenAI help to release resources from ongoing projects?

We believe that it is easier to improve the development velocity (reduce project timelines) than it is to change the team composition (reduce the number of resources on a project) using GenAI. Velocity improvements will be more immediate, while structural changes will take effect in the longer term.

From the team composition perspective, we have deduced that the currently available GenAI tools are most suitable for Middle and Senior level developers. We do not recommend introducing such tools to Junior developers yet. We also do not expect any major changes for Lead level developers, because even if some teams might shrink, they will still need Leads to oversee the processes.

Figure 2 shows the redistribution of time/effort by seniority level and what percentage of developers could be reallocated to other activities. The left-hand side shows the typical distribution of resources by seniority level, and the right-hand side shows the productivity gains.

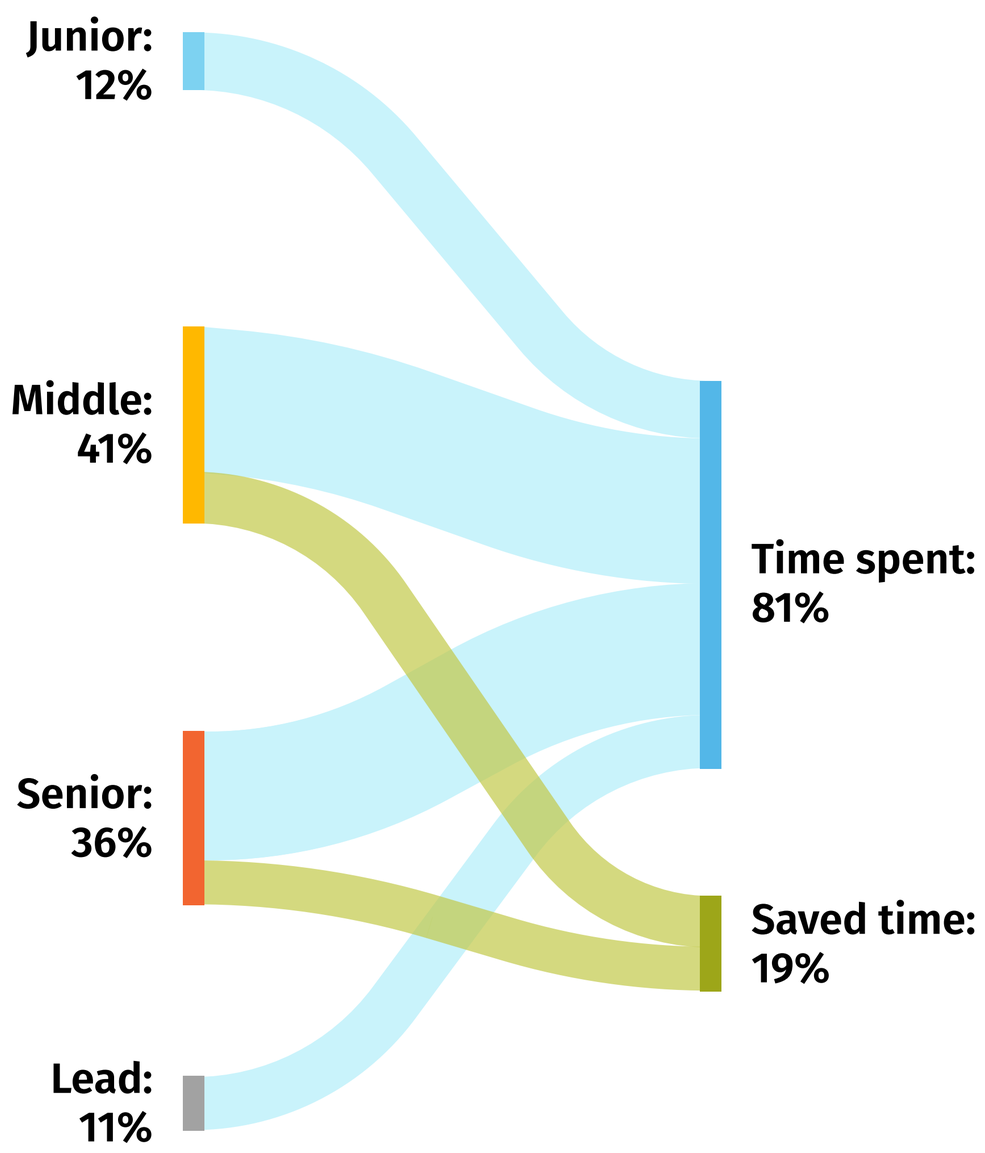


Figure 2. Productivity improvements by the seniority level.

What should the released resources be used for?

With more productive developers, some resources can be reallocated to other projects and activities. We recommend considering the following options:

- **Legacy migration:** GenAI enables efficient legacy migration provided that advanced techniques are applied. This creates the opportunity to perform migrations at much lower cost and with shorter timelines than is possible using conventional methods.
- **Prototypes and innovation:** The released resources can be leveraged for innovation. One aspect is that GenAI tools are particularly efficient for prototyping, which can be leveraged for a broad range of innovations including new customer-facing features, analytics, and internal tools. Another aspect is prototyping non-development GenAI use cases such as customer-facing chatbots and internal knowledge management systems.
- **Tech debt improvements:** GenAI tools improve the productivity of refactorings, performance optimizations, and dependency version upgrades. This creates the opportunity to close technical debt more efficiently than before.
- **Infrastructure improvements:** GenAI provides fundamentally new ways of solving some Cloud/DevOps problems such as cost optimization (FinOps) and architecture optimization. The available resources can be leveraged to improve these areas.

Finally, the available resources can be used to create GenAI CoEs and drive broader GenAI adoption in software development processes specifically. These considerations are summarized in Figure 3.

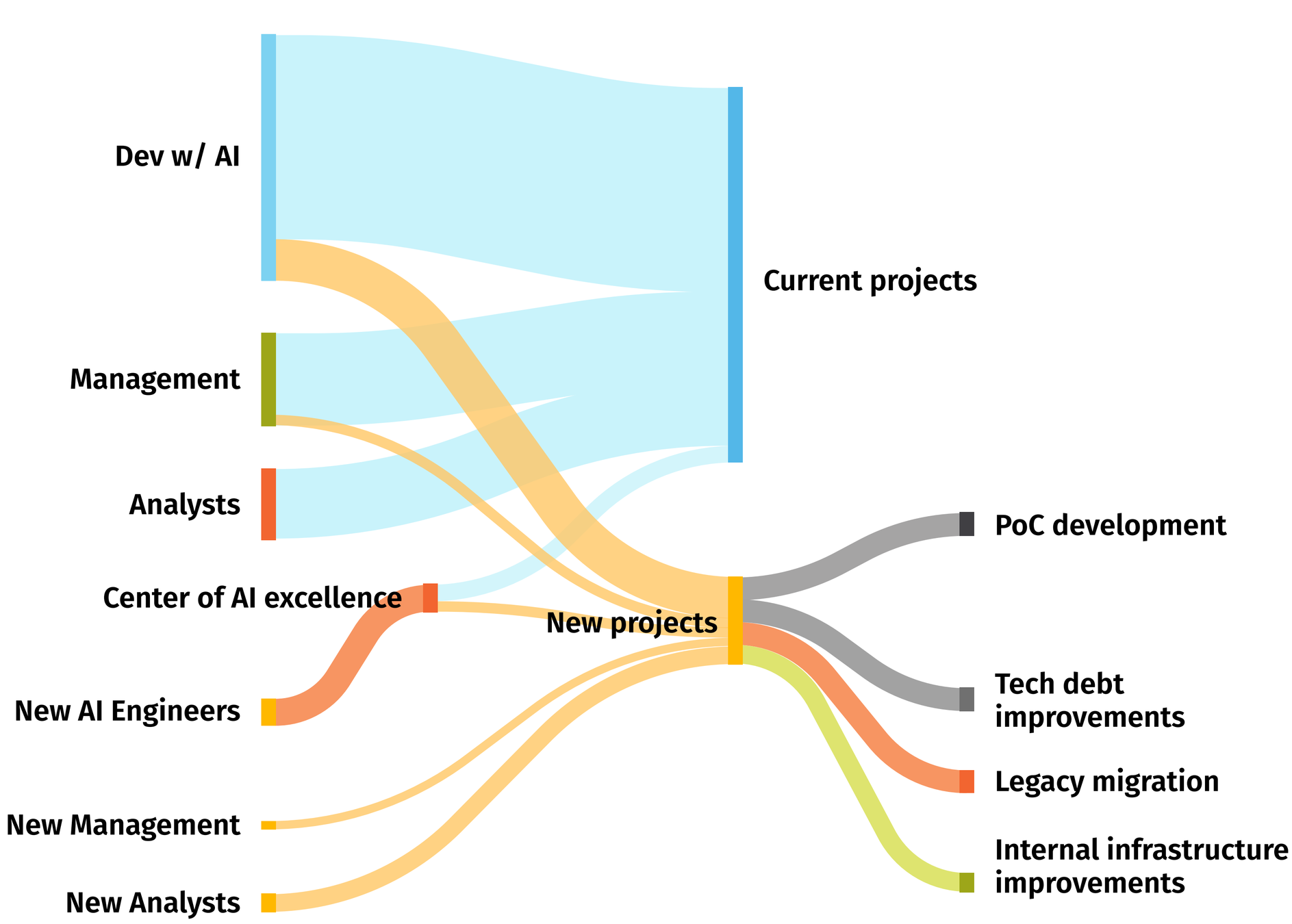


Figure 3. Example resource reallocation strategy.

How should GenAI tool selection be approached?

Most GenAI tools consist of two components:

IDE plugin, Editor, or API

An interface that is used to prepare and send the contextual information to the LLM. Examples include GitHub Copilot, Machinet, ChatGPT Reborn, and ChatGPT web interface. The role of the IDE/Editor is paramount, as complex tasks require a comprehensive understanding of the projects's context. For example, editing a specific method in some class generally requires knowing the semantics of other classes, methods, and libraries that are used in it, or those that could potentially be used to solve the problem at hand. Tools that do not provide contextual capabilities (e.g. ChatGPT web interface) are good for providing quick answers to simple questions without a lot of context (E.g. "How to build a side-by-side bar graph in python?"), but not for modifying large codebases.

Core LLM

A model that receives context about the project and task and generates code. Examples include OpenAI Codex, Anthropic Claude, and LLaMA. The role of LLMs is also very important because it ultimately determines performance and security. Different tasks require the use of different LLMs to achieve optimal results, and locking in on a particular LLM provider can have negative consequences from that perspective.

We recommend creating a modular ecosystem where multiple LLM providers can be used, including 3rd party models such as OpenAI GPT4 and private open-source-based models such as LLaMA. The IDEs and other interfaces should be able to connect to different providers. The importance of the modular approach is also justified by the following considerations:

- **Custom tools might be needed:** Specialized use cases such as codebase analysis and question answering require building custom tools using frameworks like Langchain.
- **Integration with version control and CI:** GenAI can be used for code quality and readability checks, pull request validation, and other use cases that require integration with continuous integration (CI) processes.
- **Specialized services might be needed:** Some tasks such as static code analysis (review) and test generation require specialized services.
- **Data privacy might be compromised:** Some GenAI tools present data privacy threats. This issue can be mitigated by proper tools usage policies and private LLM deployments.
- **Tools can be outdated:** LLMs that back the tools are trained on code repositories and API documentation collected at specific points in time, so one can experience issues with tools lagging behind the latest API and library versions.

The high-level layout of the GenAI tooling ecosystem is presented in Figure 4.

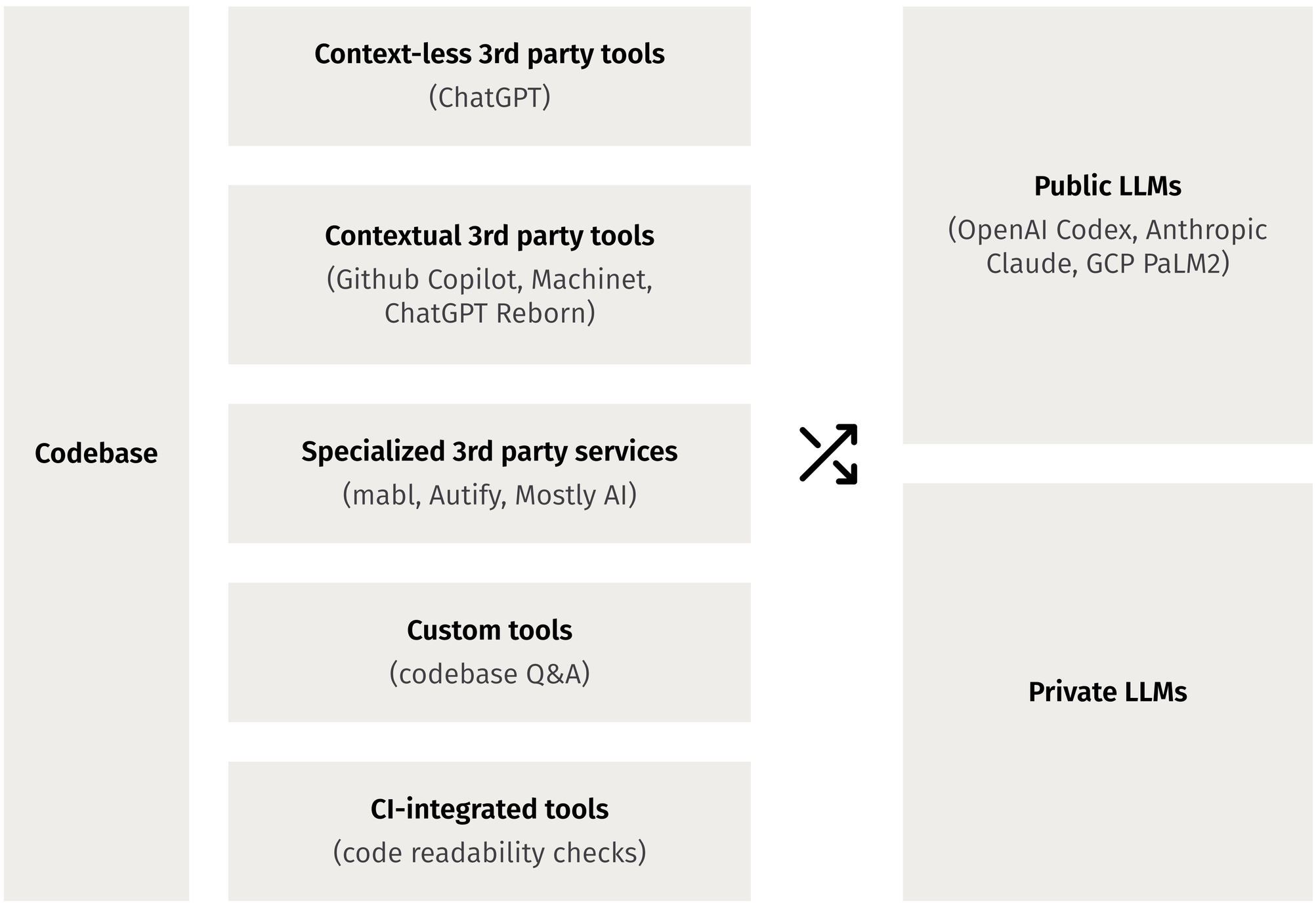


Figure 4. High-level layout of the GenAI tooling ecosystem.

What are the notable limitations of GenAI-powered development?

GenAI tools deliver significant productivity boosts for simple and routine tasks, but performance rapidly degrades on tasks that involve complex business logic or mathematical reasoning, and those that require a deep understanding of the domain, or require broad knowledge of the codebase and dependencies.

The code produced by GenAI tools needs to be carefully reviewed by engineers with good analytical skills. As noted previously, GenAI tools are not recommended for Junior developers.

Accelerating individual development tasks using GenAI

In this section, we provide more details about the individual development tasks that we have evaluated, including the observed challenges, and recommended guidelines. We focus here on the low-level usage techniques for GenAI tools, and discuss more strategic implications in the next section.

Code translation

We define code translation as the task of converting an implementation in one programming language to another language. This task holds immense practical significance for legacy migration projects where huge amounts of routine work is necessary to migrate from obsolete languages such as COBOL to modern languages such as Java.

Task examples

We employed tasks that require the translation of relatively large classes with business logic or technical components typical of enterprise applications. An example of such a task is the translation of the LRUCache class from Python to Java. The following code snippet illustrates the input provided to the evaluation participants:

```
    class LRUCache(Generic[T, U]):

        def put(self, key: T, value: U) → None:

            if key not in self.cache:
                if self.num_keys ⩾ self.capacity:
                    first_node = self.list.head.next
                    assert first_node is not None
                    assert first_node.key is not None
                    assert (
                        self.list.remove(first_node) is not None
                    )
                    del self.cache[first_node.key]
                    self.num_keys -= 1
                self.cache[key] = DoubleLinkedListNode(key, value)
                self.list.add(self.cache[key])
                self.num_keys += 1

            else:
                node = self.list.remove(self.cache[key])
                assert node is not None
                node.val = value
                self.list.add(node)
    ...
```

Evaluation results

Evaluating the results of the translation required a comparison of the respective solutions' code quality. For this purpose, the resulting code was analyzed to identify any kind of errors. The average error rate in the raw GenAI tools output was around 15 errors per 100 lines of code (including functional bugs as well as language and code quality issues). After accounting for the time spent and time required to fix the errors above, usage of GenAI tools can yield **performance enhancements of up to 45%**.

Recommended guidelines

We faced two issues when we were trying to straightforwardly apply the GenAI tools to code translation tasks: results with errors and incorrect or irrelevant results. To some extent, this can be mitigated by adding constraints such as "Follow the original contract" or "Add missing methods" to the prompts. This approach explicitly forbids AI tools from rewriting the code in completely new ways. However, the productivity gain that can be achieved using these techniques is somewhat limited.

We discovered that much better results can be achieved by using multi-stage translation chains. For example, the legacy code can be converted into step-by-step natural language description of the business logic. This description can be converted into a block diagram, and the final code can be generated based on this diagram. This approach can help to achieve productivity gains of up to 90%. However, the development of custom tools is necessary to use such techniques efficiently.

Code writing

We define code writing as creating new modules or components that extend the existing codebase. This is the most common backend development task.

Task examples

We employed tasks that required the implementation of relatively small components of business logic. For example, one of the tasks required the participants to implement a robust BankAccount class with the following interface (participants were allowed to use the programming language of their preferences such as Java, Python, or JavaScript):

```
deposit // Allows users to add funds to their account
withdraw // Enables users to withdraw funds from their account
getBalance // Retrieves the current balance of the account
getStatsForPeriod(startDate, endDate) // Obtain the balance
difference for a specified period effortlessly
getTransactionHistory // Provides a record of past transactions
for the account
getTransactionHistory(startDate, endDate) // Obtain a detailed
account of past transactions within a specified period
lock // Prevents any further transactions from being processed
while in this state
unlock // Restores the ability to process transactions
```

Evaluation results

The results were evaluated using the following metrics: time spendings, code conciseness, and code quality/error rate. In general, AI is definitely useful for code generation tasks. However, while AI-driven code generation offers several benefits, it is crucial to remember the importance of human intervention to ensure the quality and correctness of the generated code. It is important to understand that GenAI "predicts" the answer according to statistics of the previously seen examples. GenAI tools are not always capable of "understanding" the issue/question/task, but rather capable of guessing. By training on vast repositories of existing code, these AI models can learn patterns, structures, and best practices, enabling them to generate functional code snippets and even entire programs.

While AI code generation brings remarkable advantages to common tasks, it is crucial to emphasize human intervention throughout the process. Here's why:

- **Code review and quality assurance:** AI-generated code should never be blindly accepted without thorough human review. Human developers possess critical domain knowledge, context, and intuition, enabling them to identify potential issues, edge cases, and security vulnerabilities that AI may overlook.
- **Contextual understanding:** AI models lack the ability to comprehend broader project requirements, business logic, or user expectations. Human developers play a pivotal role in translating these requirements into code, ensuring the AI-generated code aligns with the desired outcomes.
- **Debugging and troubleshooting:** In the event of errors or unexpected behavior, human developers possess the expertise to debug and troubleshoot the code. AI may not always provide insightful error messages or solutions, making human intervention essential for effective issue resolution.

Upon investigation, the following results were aggregated:

- **6.25%** performance enhancement is achievable when utilizing AI-generated results that require extensive error correction.
- **50%** performance improvement can be attained when employing AI-generated results of exceptional quality.
- **25-50%** performance boost can be obtained to achieve satisfactory results.
- **28-37.5%** notable performance improvement can be expected for tasks of easy to medium complexity.

Recommended guidelines

We recommend the following guidelines for code writing tasks:

- **Incorporate precise programming terminology:** Use specific programming-related words and terms such as "function," "object," "array," etc. Ask questions with clearly defined expectations, for example: "How to achieve X in programming language Y?"
- **Specify the relevant version:** Always specify the version of the framework, programming language, platform, or other technology you are interested in.
- **Prioritize precision and detail:** Ask questions with precision and attention to detail. This will help reduce the chances of receiving an incorrect answer or an answer to a different question.
- **Ensure high-quality code standards:** AI models should be trained on lots of code that follows high-quality code standards.
- **Acknowledge human involvement:** Recognize the importance of human intervention throughout the process. Human expertise is crucial for understanding complex code, handling corner cases, following best practices, and conducting proper code reviews. AI should complement human efforts rather than replace them entirely.

Unit test writing

We designed three unit test writing tasks to conduct the survey, each corresponding to a different programming language (Java, Python, JavaScript). All tasks were slightly different, but with approximately the same complexity. The respondents of the survey were expected to cover some particular class with tests.

Illustrative examples

One of the tasks we designed requires covering an EmployeeService class implemented using Java/Spring Boot with unit tests. The implementation of the class looks as follows:

```
public class EmployeeService {
    private final EmployeeRepository employeeRepository;
    private final EmployeeMapper employeeMapper;
    public EmployeeDTO save(EmployeeDTO employeeDTO) {}
    public EmployeeDTO update(EmployeeDTO employeeDTO) {}
    public Optional<EmployeeDTO> partialUpdate(EmployeeDTO
employeeDTO) {}
    public List<EmployeeDTO> findAll() {}
    public List<EmployeeDTO> findAllWherePaymentRecordIsNull() {}
    public Optional<EmployeeDTO> findOne(Long id) {}
    public void delete(Long id) {}
}
```

Evaluation results

We evaluated the results using the following metrics:

- Overall time spent on test creation;
- Code coverage with tests (% of lines of code covered);
- Public contract coverage (% of publicly available methods that were covered);
- Assertion density. Total number of assertions per number of lines of code that is tested, which gives us a % of code that is covered with assertions.

We can conclude that AI solutions have better assertion density **(2-2.5x higher)**, which theoretically means that these tests would be more likely to detect any changes in code behavior. Both AI and non-AI solutions have 100% public contract coverage and code coverage, accentuating the importance of comparing time metrics. On average, it takes **~30% less time** to finish the task when GenAI is used.

Recommended guidelines

We expect that the results should be consistent in the real-world application if used properly. We recommend the following usage guidelines:

- **Generate tests for each method separately:** In this case, this tool will try to cover not only happy-path scenarios. It usually means that 3 scenarios would be generated: happy-path, some dependency throws an exception (e.g. if we test the controller layer, then this test assumes that the service layer throws an exception), and null-safety.
- **Always do the sanity check:** Read the test and verify that it is relevant. If not, the test is already written and configured, so it is easy to adjust it to your needs.
- **Perform complex tests via code generation:** Some intricate tests can be produced through the code-generation facet of the tool by outlining the test using plain text.
- **Use IDE plugins:** We strongly recommend using IDE plugins as they have access to the whole project, which seems to drastically improve the quality of the outcome.

API schema generation

We defined the API schema generation task as creation of the OpenAPI schema based on the service controller implemented in Java. The schema includes DTO definitions, API contract, and security configuration.

Illustrative examples

The code snippet below shows a service the schema needs to be generated for:

```
@RestController
@RequestMapping("/users")
@RequiredArgsConstructor
public class CustomerController {
  private final CustomerService customerService;

  @PageableAsQueryParam
  @GetMapping
  public Page<CustomerDto> getAll(
      @Parameter(hidden = true) @ModelAttribute CustomerFilter
filter,
      @Parameter(hidden = true) Pageable pageable) {
    return customerService.findAllByFilter(filter, pageable);
  }
  ...
}
```

Evaluation results

Since we strived for a perfect result, the only metric we were interested in was performance enhancement. The solution generated by the GenAI tool allowed us to achieve the result in 45 minutes, a notable contrast to the projected 120 minutes required to achieve the same without the aid of AI. Consequently, the cumulative performance enhancement equates to approximately **62.5% time savings**.

Recommended guidelines

Given the straightforward nature of this task, it aligns perfectly with the utilization of GenAI tools. Our suggestion is to employ prompts that are as precise as feasible, encompassing particulars such as the OAS version and model requirements (lombok, Java records, etc.).

Data access layer creation

This task is defined as generation of Java model classes, Spring Jpa/ Hibernate repositories, and Flyway scripts based on the input SQL schema.

Illustrative examples

The evaluation participants were asked to create the Java classes and Flyway scripts for the following DBML schema from the eCommerce domain:

```
Table customers {
   id integer [primary key]
   email varchar
   first_name varchar
   last_name varchar
   country varchar
    ...
}
Table reviews {
   id integer [primary key]
   rating integer
   body text [note: 'Content of the post']
   customer_uuid integer
    ...
}
Table feedback {
   id integer [primary key]
   overall_experience integer
   service_quality integer
   body text [note: 'Content of the post']
   status integer
    ...
}
Ref: reviews.customer_uuid > customers.id // many-to-one
Ref: feedback.customer_uuid > customers.id // many-to-one
```

Evaluation results

The solution derived from the GenAI tool enabled us to accomplish the task within 30 minutes, a significant improvement over the projected 75 minutes needed to achieve the same outcome without AI. The overall performance enhancement equates to approximately **60% time savings.**

Recommended guidelines

Again, due to the inherent simplicity of this task, it fits seamlessly with the integration of GenAI tools. We recommend using prompts that are exceptionally precise, incorporating particulars such as the Flyway version, model requirements (lombok, Java records, etc.), and any other relevant details.

Deployment script writing

This task is defined as creating Terraform templates to automate deployment of a new microservice, adjusting existing templates, or migrating from one cloud provider to another.

Illustrative examples

The evaluation participants were asked to convert a Cloudformation template to deploy a microservice in AWS to Terraform:

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Deploy an EC2 Service with Autoscaling Group, SSH
Tunnel, and IAM Role

Parameters:
   InstanceRole:
      Description: IAM Role for EC2 instance
      Type: String

Resources:
   VPC:
      Type: 'AWS::EC2::VPC'
      Properties:
         CidrBlock: '10.0.0.0/16'
         EnableDnsSupport: true
         EnableDnsHostnames: true
         InstanceTenancy: default
    Subnet:
       Type: 'AWS::EC2::Subnet'
       Properties:
          VpcId: !Ref VPC
          CidrBlock: '10.0.0.0/24'
          AvailabilityZone: us-east-1a
...
```

Evaluation results

As a result of research conducted, it is possible to achieve a performance boost of around **20%** for Terraform's initial infrastructure setup, and **25-30%** for Terraform's cross provider migration.

Recommended guidelines

According to the performed tests, GenAI is able to efficiently generate appropriate Docker/Cloudformation/Terraform templates for commonly used application setups (e.g. Amazon EC2 service with autoscaling group and specific network access rules). The best practice for such templates involves crafting concise, dedicated templates for distinct system components, thereby facilitating more effective utilization of GenAI. The generated template is easy to validate with enough prior experience in the respective field, and the AI is able to keep track of all the requirements for that template within this constrained context.

However, the usage of GenAI for these tasks is limited, as GenAI's knowledge of such templates is limited to the date when the model was trained (e.g. ~2021 for ChatGPT at the time of evaluation). For any newly released cloud services or any updates to service specifications, existing GenAI models would give incorrect results. For example, at AWS Re:Invent in 2022, 119 new features were presented (either new services, or updates to old ones). These features are not supported by models which are trained prior to the release of respective documentation and code snippets.

The following guidelines are recommended on GenAI usage for deployment script writing:

- **Initial setup:** This includes quick configuration of Jenkins, new infrastructure parts or new services within existing infrastructure.
- **Provider migration:** GenAI can help migrate infrastructure templates from one cloud provider to another (e.g. AWS to GCP or vice versa)
- **Learning:** GenAI could help engineers to get started with deployment script writing and provide some examples and guidelines on this process.

It is important to mention that the recommendations above are only fully applicable for cloud services which have not changed since the release of the GenAI tool that is being used. Otherwise, more manual intervention might be required to fix the discrepancies between current standards and the ones that the GenAI tool is aware of.

Performance analysis and optimization

Performance optimization is the process of identifying and fixing bottlenecks or suboptimal parts of code in software applications. This can be a challenging task, as it often requires a deep understanding of the code and the underlying system. GenAI can help to automate this process by identifying potential bottlenecks and proposing solutions.

The following activities were chosen to asses GenAI efficiency:

- **Static code analysis:** Detecting and fixing performance bottlenecks at the level of individual methods.
- **Whole project analysis:** This puts more pressure on AI tools, as the code base of a project is larger than the desired input of a tool, thus testing if it will still be able to keep track of context on such a large scale.
- **Metrics analysis:** Analysis of the application performance statistics and providing insights.

Illustrative examples

The following is an example of a method with performance bottlenecks that need to be detected and fixed (static code analysis problem):

```
public List<User> findUsersByIds(List<Long> ids) {
   return Optional.of(ids)
        .stream()
        .parallel()
        .flatMap(List::stream)
        .map(userRepository::findById)
        .filter(Optional::isPresent)
        .map(Optional::get)
        .collect(Collectors.toList());
}
```

Evaluation results

Static code analysis: GenAI was able to detect issues in the small snippets of code and provide some reasonable arguments, though in limited capacity:

- For more complicated code snippets, GenAI often misses some issues.
- For some issues, the reasoning is poor and misleading.
- Solutions for code fixes are not always applicable.

Whole project analysis: When the size of code that needs analysis increases, the results get worse, as very few issues were found and all the other problems still apply. However, GenAI was able to answer questions about code, which might be applicable for different use cases.

Metrics analysis: GenAI is not yet able to perform any meaningful analysis on data due to poor quantitative and numerical reasoning skills. It tends to just point out highest/first/random numbers from the provided data that are not applicable for any scenarios.

Recommended guidelines

The results show that not all performance optimization tasks can be enhanced by AI yet. Right now, the best applications are in the following scenarios:

Static code analysis: Although GenAI is not able to detect and explain code issues as well as other non-AI tools (e.g. SonarQube), it can still be used as a consultant in scenarios when the issue is known, and the developer is looking for quick recommendations on possible solutions. These recommendations might not always be correct, but developers with enough expertise could quickly verify them, and use them if any seem applicable.

Also, GenAI could be used to check any new code automatically during the Pull Request phase, and suggest changes. If any of the changes are applicable, it will improve code quality and could help avoid or delay any future performance issues.

Whole project analysis: In this area, GenAI could be beneficial for onboarding new engineers to a project. GenAI could be given a particular project and act as an automatic Q&A system. While the actual time gain varies for projects and individuals, it is still expected to speed up the onboarding process.

As the GenAI models are not yet aware of the concept of mathematics, it is not recommended to use such tools for activities which involve calculations and number analysis, like metrics analysis tasks. The quality of the GenAI results in this scenario could be described as random, misleading and potentially harmful.

API documentation writing

This task is defined as reference API documentation generation based on the Java service implementation. This documentation is important for helping users understand how to use the application, and for making it easier to maintain the application in the future.

Illustrative examples

We evaluated the API documentation writing capabilities using eCommerce-like services, similar to those used in the previous sections.

Evaluation results

During the test task, the solution produced by the GenAI tool enabled us to attain the desired outcome in just 5 minutes, a substantial contrast to the projected 60 minutes required to accomplish the same task without GenAI, all while maintaining a very high level of quality. The overall performance enhancement equates to an impressive **90% time savings**.

Recommended guidelines

GenAI is efficient for generating reference API documentation provided that the tool has enough context and the prompt clearly specifies the task and desirable level of details. We also recommend trying GenAI for more complex documentation writing tasks such as creation of user guides, provided that the context can be efficiently fed into the tool (e.g. small projects, POCs, or individual components).

Additional notes on tool usage techniques

In general, providing the tool with more details increases the relevance of results. For standalone (non-contextual) tools like ChatGPT, it is a good practice to set a context using prompt templates such as the following:

```
You are a professional software engineer, specializing in
{LANGUAGE_NAME}. You possess deep knowledge of {AREA_NAME}. You
have twenty years of experience, and you need to update the
library {LIBRARY_NAME} from version {VERSION_1} to {VERSION_2}.
First, outline the summarization of the major changes that
occurred, in a clear, understandable and readable way. Second,
outline API changes and deprecations that happened. Third, do a
risk analysis. Make sure your answer is correct and
comprehensive. Here is the changelog: {CHANGELOG}Based on the
given changelog provide analysis for following code and make a
decision if it needs to be fixed or not in format:
Decision: OK, NEED_FIX, NOT_SURE
```

Furthermore, It is advisable to use smaller tasks. Decompiling your task into smaller tasks and then running them through the tool is likely to yield more beneficial results than executing the entire task in a single instance. Even though most of the LLM models for GenAI are capable of handling hundreds or thousands of “tokens” of context, they can get distracted and overwhelmed.

We strongly suggest limiting the usage of GenAI tools to engineers with enough expertise to quickly assess generated solutions and enough knowledge to watch-out for tricky cases. The outputs of GenAI tools can contain issues that can be easily missed. It requires a fair share of experience to be able to find and fix those issues, as the generated solution often seems fine at the first glance.

Transforming the development process using GenAI

In the previous section, we established that GenAI delivers significant gains for a broad range of individual development tasks. In this section, we discuss how GenAI can change project structures and schedules.

Legacy migration

Legacy application migration yields significant value by reducing ownership costs, improving system quality, and adding new features, but it is a complex process. From a software engineer's point of view, the key activities are legacy code assessment/understanding, reverse engineering, including its documentation, and actual code translation/rewriting and testing. Among these activities, code translation and testing stand out as the most time-consuming, requiring significant resources and effort. This is precisely where GenAI can help. For a clearer visualization, the transformation of the development process for legacy migration projects is depicted in Figure 5.

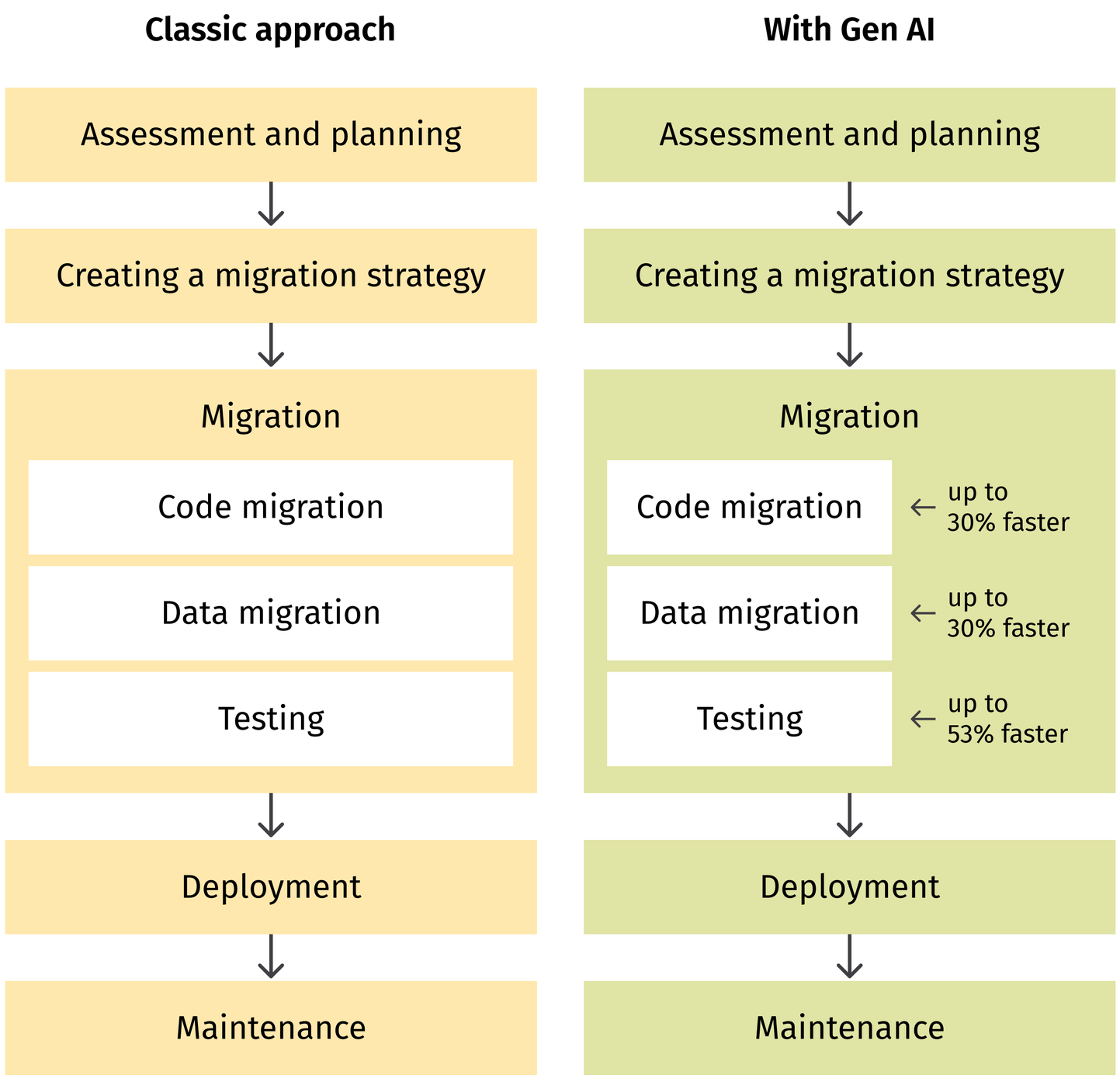


Figure 5. How GenAI transforms legacy migration projects.

1. Assessment and planning

In this phase, the existing legacy platform is thoroughly assessed, including its codebase, architecture, and dependencies. The migration goals are defined, outlining the specific reasons for migration and the desired outcomes. A target platform or technology stack is chosen based on compatibility and the ability to meet migration goals. No GenAI involvement.

2. Creating a migration strategy

A migration strategy is developed in the next step. The approach for migration, such as a "big bang" or incremental approach, is chosen. Resource allocation, time estimates, and risk assessment are performed, along with planning for potential challenges. No GenAI involvement.

3. Code migration

The existing codebase is reviewed to identify areas that need improvement or optimization during this phase. Code refactoring is performed to align the code with best practices and design patterns of the new platform. Outdated or deprecated code is replaced or updated, and dependencies are addressed to ensure compatibility. GenAI can assist developers with code translation **(15-45% savings)**, code refactoring **(10-30% savings)**, and explaining legacy code **(10-20% savings)**.

4. Data migration

During the data migration step, the data schema of the legacy system is analyzed, and the data migration process is planned. Data from the old platform is transferred to the new platform while ensuring data integrity and consistency. Verification of the migrated data is performed to confirm its accuracy. AI-powered data migration tools can automate data mapping and transformation, reducing the risk of data integrity issues and expediting the migration **(15-45% savings)**.

5. Testing

Comprehensive testing is conducted in the testing step to ensure the functionality and performance of the migrated system. Unit testing checks individual components, integration testing ensures seamless interactions between components, and system testing validates the overall system behavior. User acceptance testing requires end-users to verify that the migrated system meets their requirements. AI-driven testing tools can generate test cases automatically based on code analysis and historical test data **(10-20% savings)**. GenAI also showed great results in unit test generation **(36-70% savings)**.

6. Deployment

The deployment process is planned to ensure a smooth transition to the new platform. The migrated codebase is deployed in a production-like environment, and the system is monitored post-deployment to detect and address any issues that may arise. No major GenAI involvement.

7. Maintenance

After migration, post-migration support is offered to resolve any unforeseen issues that may arise in the initial stages of using the new platform. This ensures a stable and successful transition to the modern environment. No major GenAI involvement.

Benefits and challenges

The main benefits of using GenAI in legacy migration:

- **Faster migration:** GenAI can automate code translation and optimization, significantly reducing the time required for migration. Manual code translation can be time-consuming and error-prone, but GenAI can expedite the process.
- **Improved accuracy:** With its ability to analyze patterns and context, GenAI can provide more accurate and reliable translation options. This can lead to a smoother migration with fewer issues and bugs.
- **Reduced human errors:** Manual code translation involves the risk of human errors, which can lead to costly mistakes and system malfunctions. GenAI can help mitigate these risks by automating the process and maintaining consistency.
- **Resource optimization:** By automating repetitive and time-consuming tasks, GenAI allows developers to focus on more complex and critical aspects of the migration. Junior developers can be assigned to handle less intricate tasks, while experienced developers concentrate on challenging areas.
- **Enhanced system performance:** GenAI can identify opportunities for code optimization and refactoring, leading to improved system performance and efficiency. This can result in better user experience and overall system quality.

The main challenges of using GenAI in legacy migration:

- **Domain understanding:** GenAI tools may not always have a deep understanding of the specific domain or business logic, leading to potential misinterpretation of certain code constructs or requirements.
- **Code complexity:** Legacy systems can be highly complex and convoluted, making it challenging for GenAI tools to accurately translate and optimize the code without introducing new issues.
- **Lack of context:** While GenAI can analyze patterns and context, it may still struggle to fully grasp the context of the entire legacy system, leading to suboptimal translation suggestions.
- **Limited tool support:** As of now, GenAI tools for legacy migration are still evolving, and there might be limitations in terms of language support, codebase compatibility, and effectiveness for certain types of legacy systems.
- **Quality assurance:** While GenAI can improve accuracy, it is not infallible. Proper QA processes are still essential to ensure that the migrated code meets functional requirements, adheres to coding standards, and does not introduce new bugs or vulnerabilities.

Overall, while GenAI offers promising benefits for legacy migration, it is essential to approach its implementation with careful consideration of the challenges involved and the need for proper quality assurance and human oversight to ensure a successful migration process. The chained transformation techniques discussed earlier in this report are also very important.

New application development

There are several steps that have to be executed to create a new software application. And while GenAI is not yet capable of solving them all, it can still speed up the most time-consuming stage: development. The diagram in Figure 6 summarizes the process transformations and the main gains:

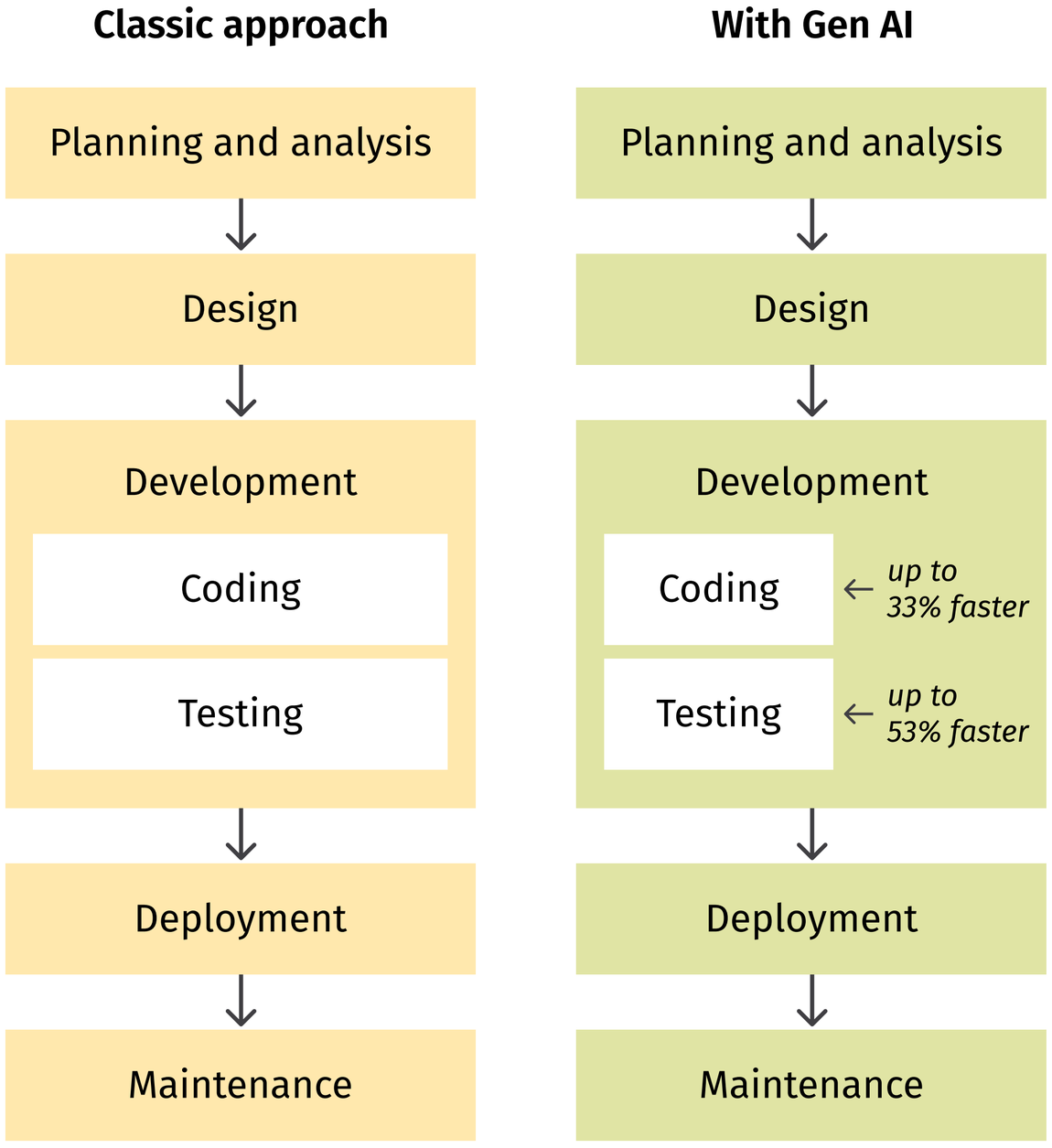


Figure 6. How GenAI transforms new development projects.

1. Planning and analysis

This stage is critical to the success of the project, as it ensures that the application is developed in a way that meets the needs of the users. The project manager will work with stakeholders to define the scope of the project, identify the requirements, and create a project plan. No GenAI involvement.

2. Design

The design stage is where the application is brought to life. The team will create detailed designs for the user interface, the database, and the architecture. The designs will be reviewed by stakeholders to ensure that they meet their expectations. No GenAI involvement.

3. Development

This is the step where the application is actually built. The team will use a variety of programming languages and frameworks to create the application. The application will be tested throughout the development process to ensure that it is free of bugs. This is the stage where the use of GenAI is beneficial, as GenAI can enhance coding **(~28-38%)** and testing **(36–70% savings)**.

4. Deployment

This step involves making the application available to users. The application can be deployed to a cloud server or installed on users' computers. The team will also provide training to users on how to use the application. No GenAI involvement.

5. Maintenance

This step involves fixing bugs, adding new features, and improving the performance of the application. The team will also monitor the application for security vulnerabilities. No GenAI involvement.

Benefits and challenges

The development phase consists of several activities, some of which may be dramatically improved. For example, for code generation, GenAI can be used to generate code, which can save developers time and effort. This can be especially useful for repetitive tasks, such as writing boilerplate code. Similar impact could be expected for testing, where GenAI can be used to automate test creation. Overall, GenAI can be used to improve the efficiency and effectiveness of the software development process. This can lead to faster time-to-market, lower costs, and better quality applications.

However, there are also some challenges associated with using GenAI in software development:

- **Complexity:** GenAI tools can be complex to use, and it can be difficult to find qualified personnel who are able to use them effectively.
- **Accuracy:** GenAI tools are not always accurate, and they can sometimes generate incorrect or misleading results.

Overall, the benefits of using GenAI in software development outweigh the challenges. However, it is important to carefully consider the specific needs of the organization before deciding whether or not to use GenAI.

Backend vs frontend development

We performed separate evaluations for backend and frontend development scenarios, and concluded that the average gains are approximately the same. The most beneficial frontend-specific applications of GenAI include the following:

- **Bootstrapping a web page layout:** Generating the initial HTML layout based on a high-level description.
- **Generating UI components:** Generating basic UI components for frameworks like React based on a textual description.

Existing application modification

Modifying an existing application often requires significant time and effort from software engineers and development teams. The process involves understanding the existing codebase, identifying areas for improvement or enhancement, making changes, and thoroughly testing the modifications to ensure they do not introduce new issues. In this research, we explore the potential benefits and challenges of utilizing GenAI-powered tools for application modification, aiming to streamline the process and improve efficiency. The actual process usually consists of the steps presented in Figure 7.

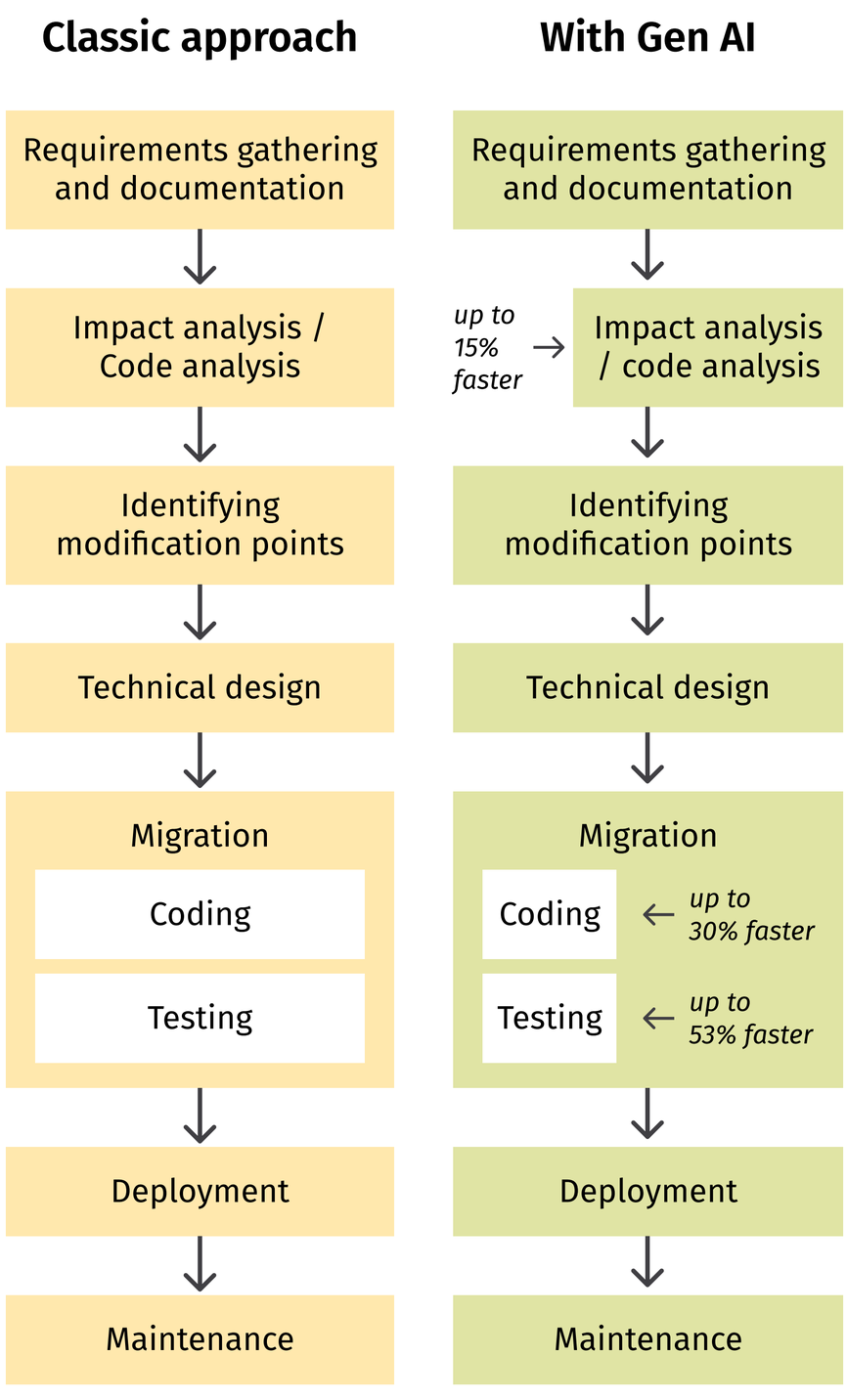


Figure 7. How GenAI transforms application modification projects.

1. Requirements gathering

This involves understanding the reasons and objectives behind the modification, collaborating with stakeholders to gather detailed requirements, including new features, enhancements, or bug fixes, and defining product specifications and sets of stories and epics. GenAI can assist with writing user stories, though this is a very small improvement in relation to the total effort required at this step.

2. Impact analysis/code analysis

This step entails conducting a thorough impact analysis to assess the implications of the proposed changes on the existing codebase, functionality, and other dependent modules. GenAI tools can help developers understand the existing codebase faster by providing insights into its structure, dependencies, and potential modification points **(10-20% time savings).**

3. Identifying modification points

This step includes identifying areas that require changes or enhancements. No GenAI involvement.

4. Technical design

The design process involves developing a detailed design for the modifications based on points from above, including architectural changes, database schema updates, and code refactorings, as well as planning the implementation steps and estimating the effort required. No GenAI involvement.

5. Development

Development entails implementing the modifications following the design and planning phase. For straightforward modifications or repetitive tasks, GenAI-powered tools can automatically generate code snippets, reducing the time and effort needed for implementation **(15-45%)**.

6. Testing

This phase includes testing the changes in a staging or development environment to identify and fix any issues. GenAI tools can assist in generating test cases based on the modifications, expediting the testing process and identifying potential issues **(10-20% savings)**. GenAI also showed great results on unit test generation **(36–70% savings)**.

7. Deployment

Deployment involves integrating the modified code into the application's build and deployment process, as well as automating the testing and deployment pipeline to ensure consistent and reliable releases. No GenAI involvement.

Benefits and challenges

The main benefits of GenAI in the context of the application modification projects include:

- **Time savings:** GenAI-powered tools can speed up the modification process, allowing developers to focus more on complex tasks rather than mundane changes.
- **Improved efficiency:** Automation of code generation and testing can enhance the overall efficiency of the modification process. In certain cases, GenAI tools may suggest best practices and patterns, leading to improved code quality and maintainability.

The main challenges with using GenAI in application modification projects include:

- **Context awareness:** GenAI tools might lack context awareness, leading to potential misunderstandings or suboptimal solutions.
- **Limited decision making:** AI-generated modifications might not consider domain-specific knowledge or business requirements, necessitating human intervention.
- **Risk of errors:** Depending solely on AI-generated solutions without careful validation may introduce bugs or security vulnerabilities.

Conclusions

GenAI is applicable to the absolute majority of software development tasks. The highest gains are observed for low-complexity, low-risk, routine tasks such as code translation, test writing, and documentation writing, where productivity improvements can easily reach **70-90%**. Some of these tasks can be accomplished in a relatively straightforward way using standard tools, but some tasks, such as legacy migration, may require creating custom tools.

The performance gains and applicability of GenAI greatly depend on tools and usage techniques, and different tools are optimal for different tasks. It is advisable to build a modular infrastructure that allows the use of 3rd party and private LLMs, and integrate them with various IDE plugins.

GenAI enables organizations to reallocate resources and engage in new projects thanks to increased developer productivity. It is advisable to consider starting new PoC projects, conducting legacy migrations, and improving tech debt and internal infrastructure.

Appendix: Evaluation methodology

The results presented in the previous sections were obtained using crowdsourcing-like evaluation that involved about 50 engineers at Grid Dynamics. In this section, we provide details about the evaluation methodology.

Evaluation process

For the purpose of this research, a number of developers were engaged to participate in different kinds of activities. These developers were split into 2 groups, where one group used GenAI tools, and the other group did not. The objective was to compare the results of these teams on a set of day-to-day tasks. The participants of this research were of different backgrounds (levels of seniority, technology stacks, projects, programming languages, etc.) to obtain a more comprehensive estimate of the AI impact in different conditions.

The evaluation workflow is outlined in Figure 8. The main steps of this workflow are as follows:

1. Survey is sent to engineers. Engineers complete entry questions (background, expertise, etc.) and become familiar with the instructions.
2. Engineers go through the task and solve it using AI. The engineers make necessary fixes on top of the AI-generated code to ensure that the result is correct and meets industrial coding standards.
3. The results are submitted for further analysis.
4. Engineers go through the exit questions (how useful AI seemed, how much they had to rework the code, etc.).
5. The other team of developers performed the same exact tasks without using AI-tools to receive the baseline (golden) results.
6. All the results are analyzed for quality (e.g. test coverage, errors number, etc.). Time to fix those issues is added as a penalty.
7. Results for an individual task are aggregated and analyzed.
8. Results for all tasks are reviewed, more general conclusions about the GenAI impact and best practices are made.

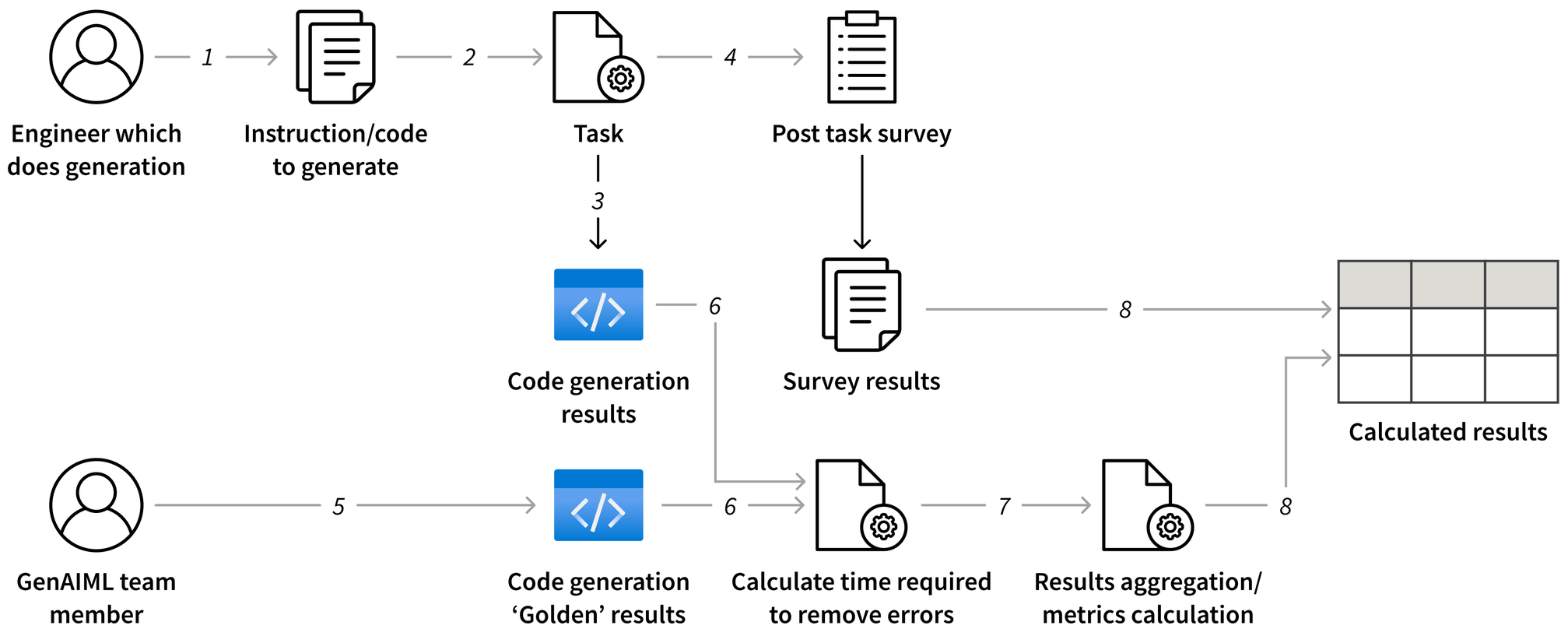


Figure 8. Evaluation workflow.

Task design

We designed the tasks with real-world enterprise development scenarios in mind rather than abstract standalone code problems. More specifically, we assumed the following scenarios:

Backend development

The scenario can be described as follows:

- The goal is to build a customer service for the typical eCommerce microservices environment to provide user registration and authentication functionalities, enabling secure account creation and login.
- Customers can manage their profile information, such as name, address, contact details, and preferences. Integration with the loyalty service enables handling loyalty program features like points accumulation, rewards redemption, and tier upgrades. Additionally, the service integrates with a 3rd party email notification system for sending notifications on events like customer creation, profile updates, and feedback/review submission.
- Feedback and reviews can be submitted by customers, stored, and displayed, with options for filtering and pagination.
- The service collaborates with the marketing system to emit profile changes, allowing accurate marketing campaigns.

Frontend development

The scenario can be summarized as follows:

- The goal is to build a typical eCommerce admin application to display eshop statistics on orders/sales. The user interface is presented in Figure 9.
- The application is designed using the micro frontend approach.

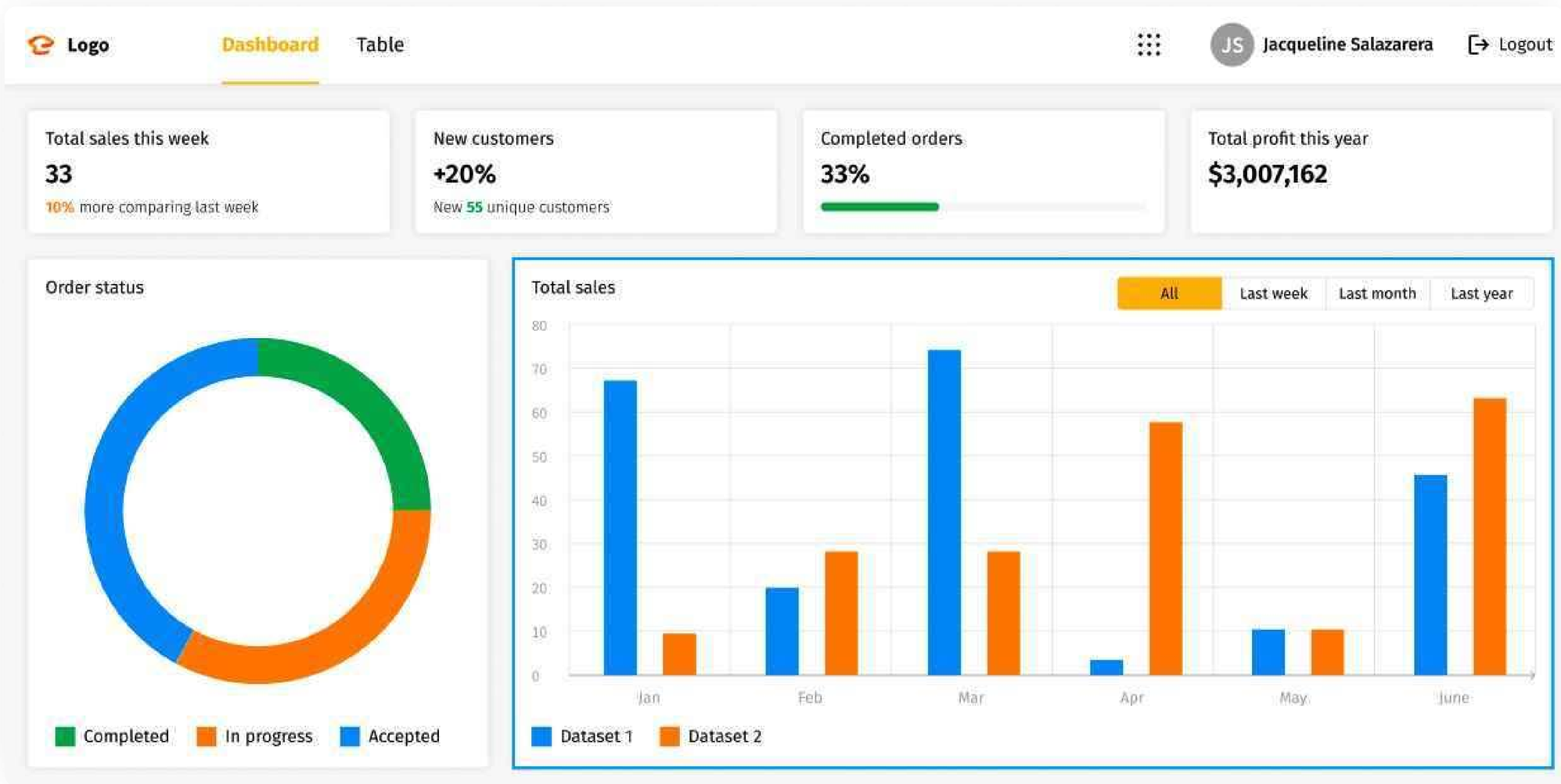


Figure 9. Interface of the admin application that was built to evaluate frontend development tasks.

Task evaluation

The goals of this research were to evaluate the benefits of using AI tools in software development and determine whether these tools are viable for the most common tasks, and if so, what approximate benefits and caveats it might yield.

The methodology used allows us to evaluate several different metrics of each of the software development processes in question (manually solved and with the use of GenAI tools). The most important part of the research is that the tasks and scenarios are designed based on everyday tasks developers tend to face.

For each of the scenarios, tasks were prepared with similar complexity, and often in several programming languages, to try and cover many different aspects. The evaluation process varies slightly depending on the scenario, but in general, two main metrics were compared:

- **The quality metric** depends heavily on the scenario, but overall it gives an understanding of how much any given solution aligns with production development requirements (e.g. test coverage, code quality, number of errors, etc.).
- **The time-spent metric** shows how much time participants needed to get to the solutions. After that each of the solutions is reviewed against a predefined set of rules to measure code quality (e.g. after code translation, the new code is able to pass all of the same tests as the initial code).

These metrics are evaluated for each of the survey results and for the solution we achieved after solving that same task without the usage of AI tools. Metrics are compared to determine benefits/caveats of the approaches.

Tools

Developers had access to most tools available on the market and many of them were familiar with at least one GenAI tool based on their job experience or personal experiments. In the evaluation process we did not prescribe which tools to use, but most of the engineers used OpenAI ChatGPT web interface, JetBrains Machinet plugin, and GitHub Copilot. Some engineers also used specialized services such as Autify, mabl, and Gretel.

We believe that this approach to evaluation is optimal because the tools and their capabilities change very rapidly, and by allowing the developers to use the tools they like, we obtain a representative picture with biases averaged out.

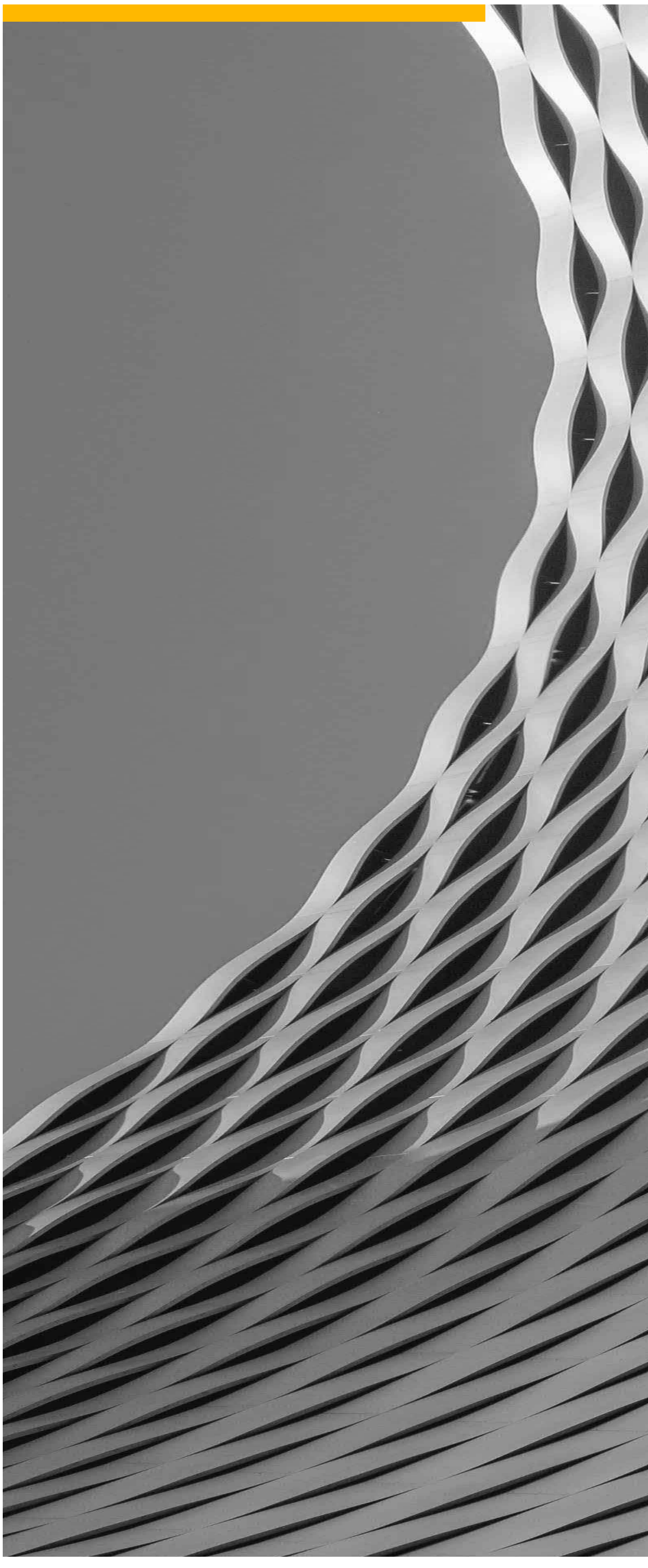
About Grid Dynamics

Grid Dynamics is a global digital engineering company that co-innovates with the most respected brands in the world to solve complex problems, optimize business operations, and better serve customers. Driven by business impact and agility, we create innovative, end-to-end solutions in digital commerce, AI, data, web UI and UX, and cloud to help clients grow.

Headquartered in Silicon Valley, with delivery centers located throughout the globe, Grid Dynamics is known for architecting revolutionary digital technology platforms for 7 of the 25 largest retailers in the US and 3 of the 10 largest consumer goods companies in the world, as well as leading brands in the digital commerce, manufacturing, finance, healthcare, and high tech sectors.

Our secret sauce? We hire the top 10% of global engineering talent and employ our extensive expertise in emerging technology, lean software development practices, a high-performance product and agile delivery culture, and strategic partnerships with leading technology service providers like Google, Amazon, and Microsoft.

In 2020, Grid Dynamics went public and is trading on the NASDAQ under the GDYN ticker.



About Grid Dynamics

Key facts

- Offices across the US, Mexico, UK, Europe, and India
- Thousands of employees across the globe
- Proprietary starter kits developed in collaboration with AWS, Google Cloud, Microsoft Azure, and others

Areas of expertise

- **Experience engineering**
  Web UI | Mobile | UX | AR/VR
- **Data science and AI**
  Generative AI | Search | Personalization | Supply chain | IoT
- **Platform engineering**
  Microservices | MACH | Composable
- **Data engineering**
  Big data | Streaming | MLOps
- **Cloud and DevOps**
  CI/CD | AIOps | SRE | QE

Clients

	Google	JABIL
align	Levi's	StanleyBlack&Decker
RAYMOND JAMES	fiserv.	AMERICAN EAGLE
Boston Scientific	INSPIRE Brands	HELLO FRESH

Grid Dynamics

trusted engineering partner for digital transformation

Grid Dynamics Holdings, Inc.

5000 Executive Parkway,

Suite 520 / San Ramon, CA

650-523-5000

www.griddynamics.com