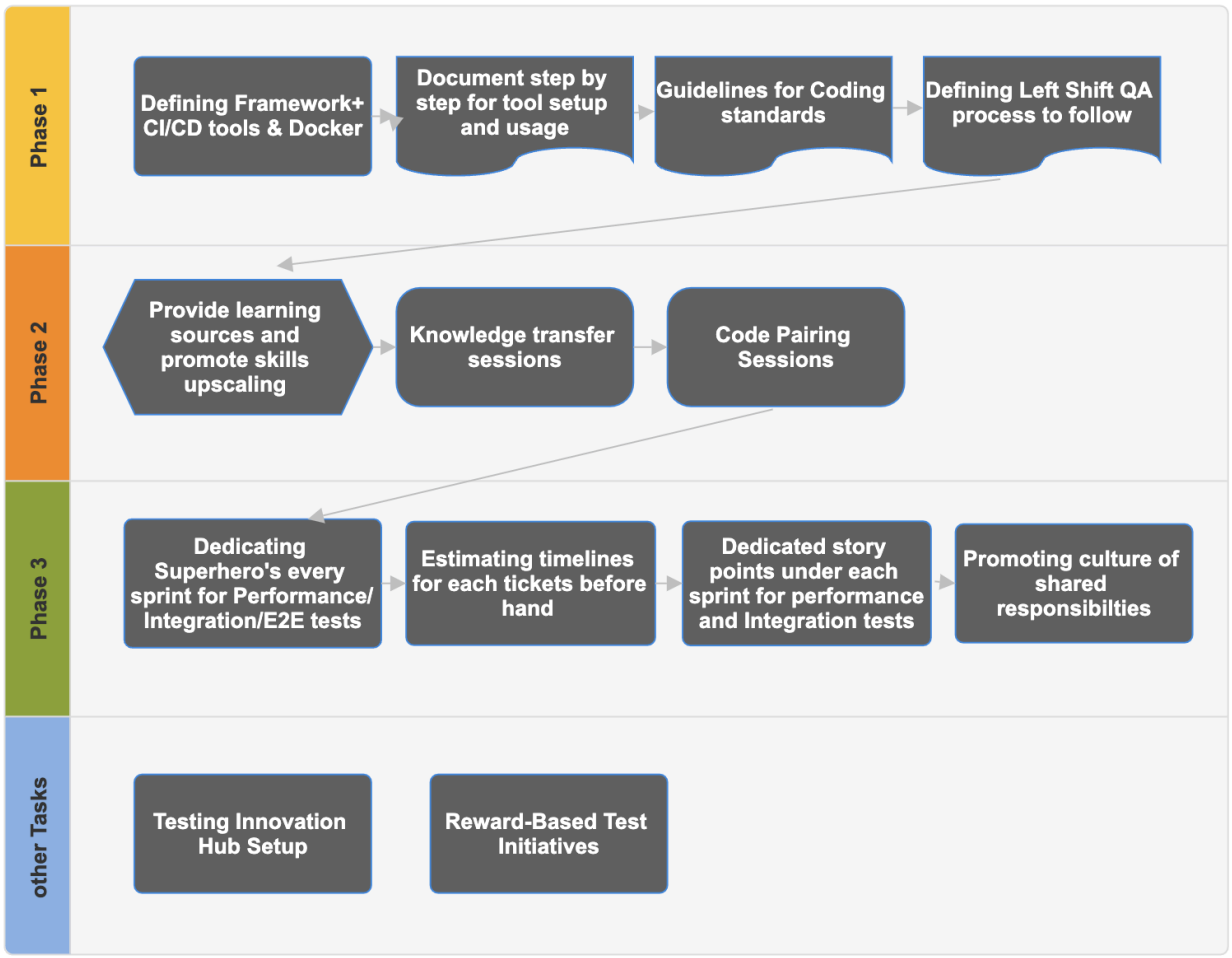
Critical gaps in the current system:

- Very less adoption of Integration test (approximately 5%)
- Very less adoption of Performance test (approximately 5%)
- Less adoption of E2E tests (approximately 5%)

Aim of this Document:

- Higher adoption rates for integration tests, performance tests, and E2E tests.
- Overall timelines & milestones achieved.
- Potential risks and their mitigation plan.

Proposed Plan for Higher Adoption Rate for Integration test , Performance test & E2E Test



Phase I: Defining Tools and Procedure:

1. Introducing the Framework, CI/CD Tools, and Docker:

Step I: <u>Consolidating Framework for Integration test</u>: Having multiple frameworks in the system makes context switching a hassle. Additionally, we have QA and 40% developers skilled in Java, and we have 40% developers skilled in NodeJS, so we can either switch to one framework entirely or switch to NodeJS entirely.

Pros of this solution:
1. ***There are fewer frequent context changes***: Using a single framework can simplify things and lessen the need for frequent, frequent context switches, which could eventually be advantageous for testing and development.
2. ***Leveraging existing skills:*** Most team members already know Java and Node.js, so consolidation into one language may increase productivity and reduce learning curves.
3. ***Less maintenance:*** By using fewer tools, a single framework eliminates the need for constant upkeep to handle updates, dependencies, and compatibility problems.
4. ***Single expertise:*** By sticking to single technology, the QA team can more easily concentrate their efforts on developing a deep understanding of it rather than squandering their resources on several frameworks.

However, things to consider:

- ☐ **Business requirements and team competence**: advantages and disadvantages of Java and Node.js for project scalability, speed, testing tools, and other requirements.
- ☐ **Pilot migration:** Begin by transferring a small, less crucial portion of the system to the business. This will typically provide more realistic insight into the true impact of consolidation as well as better insights into where the challenges will lie.
- ☐ **Gradual Standardisation to the Standard Framework:** Rather than making a radical change, standardise teams and projects according to comfort level, importance, and priority. This will reduce the initial shock of change and allow it to transform gradually.

As a result, the choice to move to a Java or Node.js-based framework will be determined by the project's requirements, and the preparedness of the team. This tactic, when properly thought out, can result in higher output, improved teamwork, and a more controllable testing framework.

Step II: <u>Integrating framework with CI/CD tools and Docker</u>:

After deciding between a Java- or Node.js-based solution, we can select a framework.

A. For **Java**, use **JUnit** or **TestNG** for integration tests, and **Testcontainers** for managing external dependencies during tests.
B. For **Node.js**, use **Cypress** or **Playwright** for E2E and integration testing, and **Jest** or **Mocha** as the main test runners for integration scenarios.
C. For Performance test, keep expanding the current K6 framework.

Testcontainers: This library works incredibly well for executing integration tests that rely on third-party components like message brokers, databases, and other systems. During testing, it enables the spinning up of lightweight, disposable containers that guarantee a reliable, repeatable environment.
Jest & Mocha is an all-in-one tool that is commonly used for testing JavaScript applications. It is especially useful for integration testing in the Node.js ecosystem because it can run tests, make assertions, and mock objects.

D. **Continuous Integration with Jenkins or CircleCI:**

   Test Execution in CI/CD Pipeline: Jenkins or CircleCI should be configured to automatically trigger integration tests with every new commit or PR. Within the CI/CD pipeline, the majority of problems are found early on. Stages should be defined using Jenkinsfiles and/or.circleci/config.yml, and tests for Java should be run using JUnit/TestNG and Node.js using Jest/Mocha.

   Tests can be parallelised with Jenkins or CircleCI to speed up feedback processing. Parallel test execution is already possible with JUnit/TestNG; CircleCI gives you the option to start tests concurrently to expedite the process.

E. **Integrate CI/CD Pipeline with Docker:** Multiple containers can be orchestrated to spin up test environments, databases, and application services all at once.

   Docker makes it simple to integrate Java testcontainers into continuous integration and delivery pipelines by managing the provisioning of environments required for test execution. For instance, Docker images for services like databases (such as MySQL and PostgreSQL) are pulled by Jenkins or CircleCI, and you are assured of a clean, repeatable, and simple-to-run test environment.
   In the Node.js ecosystem, Cypress and Playwright are also hosted in Docker containers. This allows easy development in consistent testing environment isolated across platforms.

By integrating the suggested frameworks with Docker, Jenkins, or CircleCI, testing can be made more efficient, consistent, and automated, leading to quicker feedback and higher-quality results across the development lifecycle.

2. Document step-by-step for tool setup and usage:

Following the proper tool and technology selection and end-to-end framework configuration with CI/CD integration, engineers should record all setup and usage procedures. Any new team member will be able to set up and operate the frameworks with ease thanks to this documentation, which also acts as a guide. It guarantees that anyone who is unfamiliar with these tools can begin using the entire framework by understanding the procedure and adhering to explicit instructions.

3. **Follow guidelines for coding standards:** The purpose of these coding standards is to ensure that the codebase is:

1. **Consistent:** Encourage the team as a whole to read, comprehend, and maintain the code.
2. **High Quality:** Making sure the code is efficient and minimising the possibility of adding bugs.
3. **Maintainable:** Encouraging simpler updates, debugging, and developer cooperation.
4. **Collaborative:** Increase team member communication and streamlining code reviews.

The development process is ultimately enhanced by these standards, producing software that is more dependable and durable.

4. **Defining the Left Shift QA process to follow:**

It is always advantageous to shift QA to the left in order to find bugs early. Here, though, our main concern will be making sure that thorough performance, integration, and end-to-end testing protocols are adhered to consistently and nothing is missed.

- **Before development begins, plan to include test design for new features.** Developers and QA should talk about which use cases—such as unit tests, performance tests, and end-to-end tests—will be automated during the test design phase. Make separate tests for each set of requirements that fall under performance and end-to-end automation tickets during test design, link them to the feature's epic, and mark the epic as done once all automation coverage for that feature is complete.
- **For Existing/Legacy Features:** Examine current use cases to determine which manual tests—performance and end-to-end tests included—should be automated. Create tickets for these automation tasks, prioritize them, and add them to the backlog. These tickets can be picked up according to priority during sprint planning.

Phase II: Capability Building and Cultural Transformation:

1. **Provide learning resources and help to upskill**: Most of the Engineers or QA team members might be new to the frameworks like Node.js, Java, K6. Jenkins and Docker etc. So we should provide them the required supports to upskill as per the existing framework we select. This will help more team members to contribute for building the scripts, increase in the codebase obviously helps increasing the test coverage and overall process.

2. **Knowledge transfer sessions:** Throughout the organisation, we can set up knowledge transfer sessions wherein engineers can swiftly pick up new skills while more seasoned team members guide and assist those who are not familiar with the tools and technologies being used.

3. **Code Pairing Sessions:** Encourage a culture where team members assist and support one another when utilising new tools so they can learn more quickly and perform better.

Phase III: Adopting in Sprint Cycle:

1. **Dedicating Superhero's every sprint for Performance/ Integration/E2E tests:** To speed up the automation process, assign one or two team members to work solely on increasing the automation coverage for end-to-end, performance, and integration tests during each sprint. The number of committed "superheroes" should be decided upon by taking into account the capacity of the sprint and the importance of the tasks. Therefore, choosing the right number of superheroes for each sprint requires careful capacity planning.

2. **Estimating timelines for each ticket before hand:** We can calculate the number of automation tickets to pull based on capacity planning and story points by estimating the story points for each automation task.

3. **Dedicated story points under each sprint for performance and integration tests:** Every sprint, we should try to add 10–20 story points of tickets that are specifically focused on increasing the number of automation tests for end-to-end, performance, and integration testing. If the team is overloaded with other high-priority tasks, we can reduce this allocation accordingly. On the other hand, we can increase the allotment beyond 20 story points if other tickets are of lower priority or have fewer story points.

4. **Promoting culture of shared responsibilities**: By delegating tasks like testing, automation, and code reviews to team members, developers, QA, and stakeholders, you can encourage everyone to contribute to maintaining quality. Encourage pair

programming, cross-functional cooperation, and shared accountability for quality objectives. This reduces silos, enhances collective learning, and builds a more cohesive team where everyone contributes to both development and quality assurance.

Other Implementation Work:

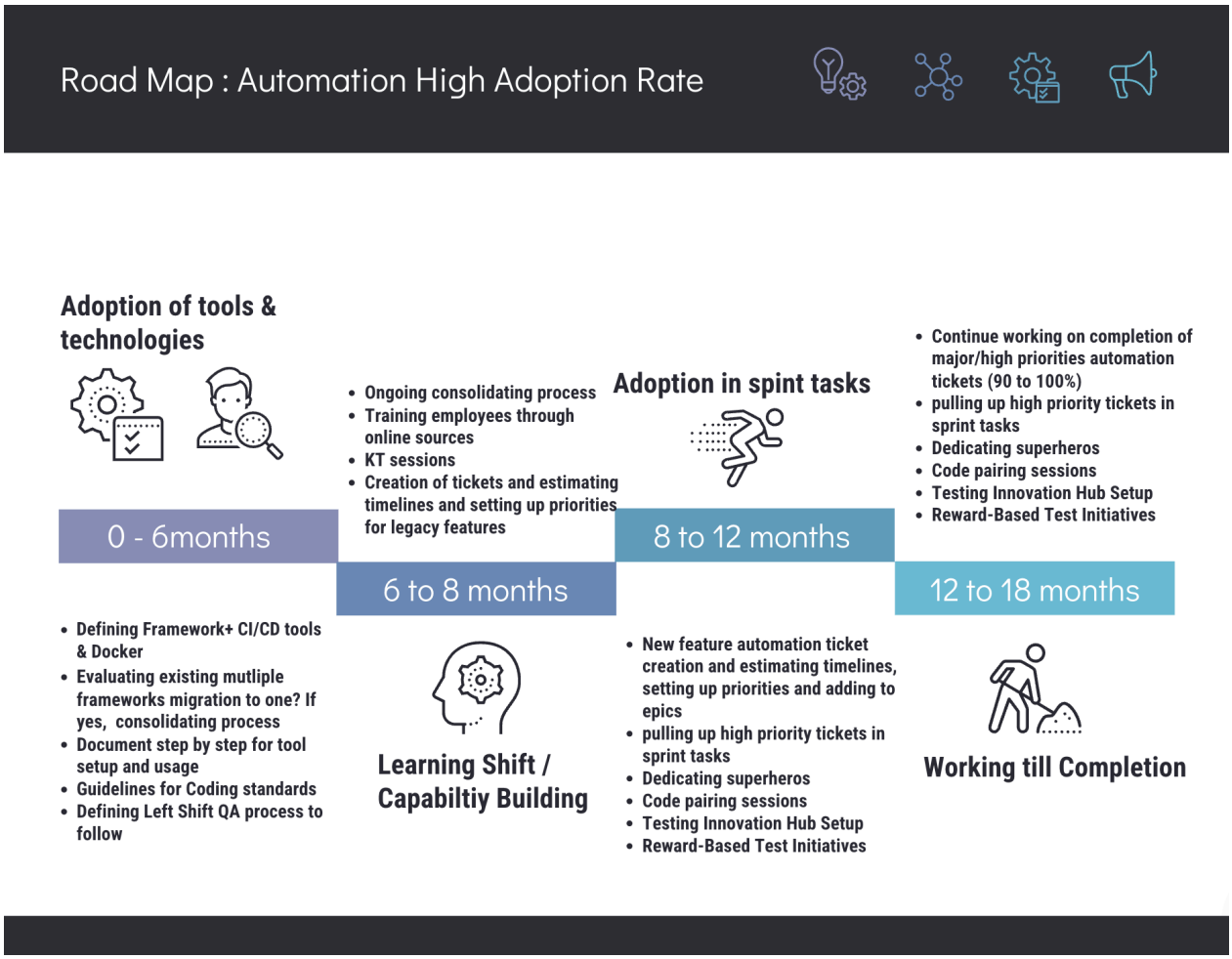
1. **Testing Innovation Hub Setup:**

    - Within the company, form a specialised unit centred on performance, integration, and end-to-end (E2E) testing best practices, tools, and frameworks, such as a Centre of Excellence. In order to assist engineers in effectively implementing testing procedures, this team can create reusable templates and support training programmes.
    - Reusable Test Libraries: Establish a central repository for scripts, templates, or parts that can be reused for E2E, performance, and integration testing. Engineers will find it easier to use this shared resource, which will cut down on the amount of work needed to begin implementing thorough testing.

2. **Reward-Based Test Initiatives**

- To promote the adoption of tests, make a leaderboard for teams or individuals who write the most integration and performance tests, find the most bugs early, or maintain the highest degree of test coverage. Give top performers recognition and rewards through internal contests, bonuses, or company-wide shoutouts.
- **Internal Hackathons:** Plan testing sprints or internal hackathons where engineers spend a predetermined amount of time working exclusively on enhancing integration and performance tests. Provide awards or recognition to winning teams.

Timelines & Milestones achieved:



Risks & Mitigation Plan:

- Engineers may be afraid to take on additional testing responsibilities for new changes

  Mitigation Plan:

  - *Already added reward system in my plan (Fun and Incentive-based)*
  - *Rewards and recognition*

- Engineers may think that there is no time for writing automation tests for the product

Mitigation Plan:

- *QA and dev collaboration: Culture of shared responsibility*
- *Capacity planning by Sprint Manager/ Engineering Manager to incorporate separate points just for automation work*

- May be at some point this strategy fails to achieve the adoption rate

  Mitigation Plan:

  - *It might be useful for the organization to establish a Management KPI in the Monitoring of Adoption Rates: Every quarter there should be set up review meetings to evaluate how effective these strategies have turned out to be and make adjustments based on adoption rates. Involve relevant personnel (tech leads, managers) in the evaluation process.*
  - *In case certain technologies or frameworks will have difficulties in adoption, then there has to be a fallback plan which does not impose restrictions on the choice of tools. Keep in touch with the teams in order to collect their feedback early and prevent people from opposing to the change.*