CHAPTER 12

“-ilities” Fitness Function

After designing a solution, you must evaluate your design by using the fitness function to ensure it can solve the problems under consideration. To check the functional requirements fitness, you might be using test-driven development, but what about the “-ilities”? How will you check the fitness function for the “-ilities”? This chapter gives you insight into the step-by-step approach for the “-ilities” fitness function.

In a modern cloud native environment, the architecture will evolve constantly to support business changes. How do you support evolution? You have everything to support and test the functional use cases, but what about the architecture, design, and the “-ilities”? Does your designed software support all the architecture elements and decisions, not with theory but with actual data points? For example, how do you shift left the PowerPoint version of the architecture block diagram into the real implementation?

In the age of cloud native and modern-day architecture, you need to predict the performance and behavior of your architecture during the development time, not at the end of the development lifecycle; it is about being proactive, not reactive.

In the previous chapter, I explained how to design the architecture for the “-ilities” and assess the health of a system.

In this chapter, you will gain more insight into how to conduct a fitness check of the designed “-ilities.”

- What is fitness in architecture?
- How do you create a fitness function?
- How do you test the fitness function?
- How do you measure the fitness function?

S. R. Goniwada, *Cloud Native Architecture and Design*, https://doi.org/10.1007/978-1-4842-7226-8_12

What Is a Fitness Function?

In evolutionary computing, a fitness function is a type of objective function that is used to determine how close a given solution is to achieving the desired result. The function returns the fitness of your architecture. These functions take the solution to the problem as input, and they produce as output details about how to fit and how good the solution is concerning the problem under consideration.

The fitness function is being used in genetic programming and genetic algorithms to guide simulations toward an optimal design solution.

A genetic algorithm is a machine learning technique that attempts to solve a problem from a pool of candidate solutions. These generated candidates are iteratively evolved and mutated and selected for survival based on a grading criterion, called the fitness function. For example, when using a genetic algorithm to optimize a driverless car, the fitness function assesses the identification of safety, signboards, objects, zebra crossings, and other characteristics that are desirable to create a 100 percent safe driverless car.

A fitness function can be applied to a cloud native architecture to determine how close the designed architecture is to achieving the desired characteristics. Fitness functions are an objective way to assess architectural characteristics.

In cloud native architecture, the fitness functions are used to evaluate the design for "-ilities," and the defined architecture must be evaluated using a fitness function algorithm to ensure its ability to meet the required service level agreements (SLAs), service level indicators (SLIs), and service level objectives (SLOs) under consideration.

The fitness function is not generic; each system's "-ilities" varies. Some systems require more security, some systems require high scalability and availability, and some might require more resilience to failure. Therefore, your input and output of a fitness function are system-specific.

As shown in Figure 12-1, a fitness function takes target input and applies the fitness algorithm for all the required "-ilities" based on input and generates the output with metrics. The fitness function represents every requirement of your system. You can consider the fitness function as a metric or test case. Some "-ilities" require a test case; for example, the performance fitness requires you to run performance test cases to identify fitness metrics.

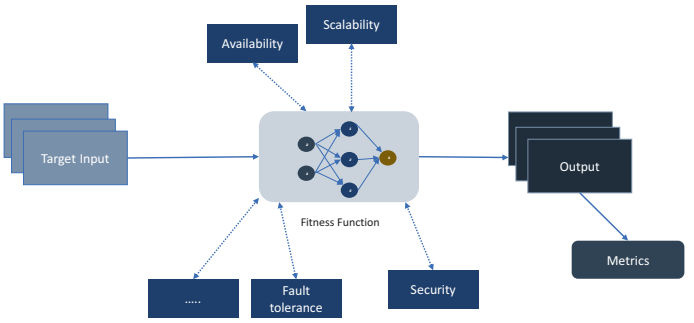

Figure 12-1. *Fitness function*

A fitness function should be clearly defined and provide a quantitative measure of how fit a solution is for a particular problem. A quantitative measure is how fit a solution is for a particular problem, while a quantitative result matrix is what will allow you to compare the architecture before and after a change is introduced.

Categories of Fitness Functions

A fitness function protects the various architectural constraints of the system. The constraints are not the same across the system. They vary depending on the nature of the system. You can check the fitness function in various dimensions such as scope, frequency, domain, global presence, architecture type, and other ways.

Atomic vs. Holistic

The atomic fitness function focuses on a single context and one architectural characteristic. For example, you can have a single "-ilities" unit test that is designed to test cohesion, coupling, etc., and is atomic.

A holistic fitness function takes multiple architectural characteristics into consideration at the same time, for example, by conducting security and performance fitness functions together and calculating the quantitative matrix.

Triggered vs. Continuous

Triggered fitness functions are executed based on some event. For example, the "-ilities" unit test is executed as part of the build.

A continuous fitness function runs constantly, and its execution is not based on some occurrence of some event. For example, the monitoring tool monitors continuously, which will send an alert when certain conditions are met.

Static vs. Dynamic

A static fitness function is one in which the value for the condition that we are testing for is constant. A test is looking to ensure that the result is less than some static numeric value or that a test that returns true or false returns the value that you expect.

The dynamic fitness function change is based on a different context. For example, the performance test might be different depending on the current level of scalability. At a much higher level of scalability, a lower level of performance might be acceptable.

Automated vs. Manual

Automated fitness functions are triggered automatically. They could be part of the automated unit test or part of the continuous integration (CI) pipeline. In cloud native, the preferred approach is automated. However, there are many times you may require executing fitness tests manually.

Temporal

The temporal fitness function is based on a designated amount of time. Other fitness functions are focused on architectural change but are triggered based on time. For example, the fitness function is created for a system patch on certain days. This executes based on time.

International vs. Emergent

Many fitness functions can be defined during the discovery phase of a project; these are known as *international* fitness functions. However, some characteristics of the architecture are not known right from the beginning but emerge as the system continues its development. These fitness functions are known as *emergent* ones.

Domain-Specific

Domain-specific fitness functions are based on specific concerns related to the business domain such as compliance, regulatory, security, etc. A domain-specific fitness function can ensure that the architecture continues to conform to these requirements.

All these categories are executed either during design or at runtime; they are further classified here.

Design-Time Fitness Function

At design time, you need to run fitness functions related to atomic elements like a unit of code or static security. For the code fitness function, you have to write a unit test specific to the architectural concerns such as coupling and cohesion, and write a domain-driven fitness function to check against the domain modularity of your system.

Runtime Fitness Function

In the runtime fitness function, you need to consider running a fitness function for the context of one "-ility" or implement a holistic approach by combining more than one "-ility." In the single context, you can examine the fitness of runtime security, that is, dynamic security testing (DAST) on OWASP vulnerabilities or scalability testing against the SLA.

In a holistic runtime fitness function, you need to combine more than one "-ility" to conduct a fitness function, for example, combining dynamic security and scalability, security, and performance. This helps you to identify whether your system can meet the target SLA holistically. Here you need to execute both security and scalability fitness together.

Execution of the Fitness Function

The fitness test can execute either as a single manual or as a continual part of the DevSecOps pipeline.

Manual Execution

The design-time and runtime fitness function tests are executed manually by the engineers either in the development environment or in the test environments. Many projects globally still follow a manual approach to developing, testing, and deploying; therefore, these systems are required to conduct fitness tests by using certain tools. For example, the "-ilities" unit test can be run by the developer on their machine or in a development environment, and the performance engineer can execute the performance test by using tools in a QA or performance environment.

Even though you are using automation in your project, some aspects of fitness functions resist automation; therefore, you require a manual execution.

Automated Execution

In the automated context shown in Figure 12-2, there are both-design time and runtime fitness test within an automated context, like a continuous integration (CI) and continuous delivery (CD) pipeline. In the pipeline, you can execute the "-ilities" test cases and implement a single and holistic approach of a runtime fitness test.

After collecting the fitness functions, configure them in a testing framework. Ideally, the fitness function should address the requirements of the "-ilities" in terms of an objective metric that is meaningful to stakeholders. Regular fitness function reviews can focus architectural efforts on meaningful and quantifiable outcomes.

You can configure unit test jobs and domain-driven bounded context test jobs as part of the CI/CD pipeline for the continual execution of design-time fitness. As a result of this automation, every new and major change in service is developed in a way to pass the fitness functions.

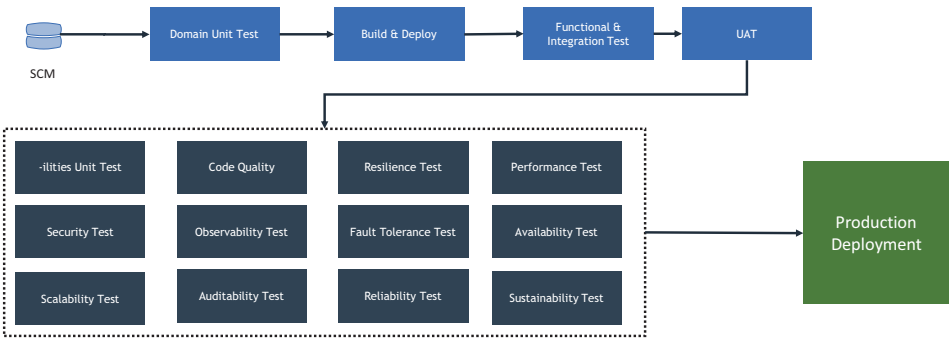

Figure 12-2. *Automated fitness function*

For a runtime fitness function, you configure a single-context approach to execute every "-ility" to make sure each one meets the SLAs and later executes a holistic approach by combining various "-ilities." For example, combine security cases like API authentication, security at transit, and data encryption along with the performance of state and intercommunication of services. Once you execute the functions, you can create a matrix of both and compare the results.

Fitness Function Identification

You need to define most of the fitness functions in the project discovery phase as they are characteristics of architecture and design, but this is not final. As your project evolves, you need to revisit your fitness function to accommodate evolution. As I mentioned, the fitness function is not generic; it has to be project-specific and industry-specific. For example, the financial industry has a lot more compliance than other industries, and an ecommerce application and trading platform has more spikes than other industries.

During the identification of the fitness functions, you need to categorize them based on relevance to your project because these fitness functions directly impact your design decisions. The categorization based on relevance is as follows. Each fitness function must have objectives and quantifiable results.

Key: These categories of fitness functions directly impact your design decisions and architecture choice.

Relevance: These categories do not directly impact the design and architecture decisions but relevance during the realization of a design.

Not Relevant: These categories are not of much importance but are nice-to-have fitness functions.

Fitness Function: Coupling and Cohesion

To conduct a fitness test on coupling and cohesion "-ilities," you need to write a "-ilities" unit test that verifies against the developed code. Two strategies are available to conduct fitness test.

- By layer
- By feature

The layered classical approach is followed by the Model View Control (MVC) pattern; this strategy is based on horizontal layering. The by-feature strategy is when the features are organized by vertical layer. All domains or features related to a single domain reside in a single layer. This matches the layout of a cloud native service.

To illustrate the by feature fitness function, as shown in Figure 12-3, I have created a small Java project that contains a package structure with dummy classes and components.

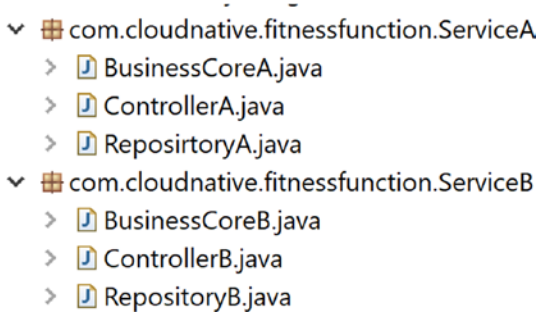

Figure 12-3. *Code package of cloud native services*

As shown in Figure 12-4, I defined two services and defined the rules. If you write code that invokes service A and service B cyclically, the unit test fails. I prefer to write these test cases along with the domain test cases. You need to make sure that all these "-ilities" unit test cases are executed separately and owned by the architecture team.

```
public class DesignFitnessTest {
@Test
public void VerifyDesignilities()

{
    DependencyConstraint fitness = new DependencyConstraint();
    JavaPackage serviceA = fitness.addPackage("com.cloudnative.fitnessfunction.ServiceA");
    JavaPackage serviceB = fitness.addPackage("com.cloudnative.fitnessfunction.ServiceB");

    serviceA.dependsUpon(serviceB);
    serviceB.dependsUpon(serviceA);
    jdepend.analyze();
    assertEquals("Dependency mismatch", true, jdepend.dependencyMatch(fitness));


}

}
```

Figure 12-4. *Jdepends verify of fitness function for coupling and cohesion*

To execute these test cases, you need to make sure to configure the CI pipeline as a separate job to track the metrics of the fitness function.

Fitness Function: Security

To conduct your architecture and design fitness function, there are two strategies. Each strategy evaluates various fitness functions for your architecture.

- Static security system testing (SAST)
- Dynamic security system testing (DAST)

SAST conducts a fitness test of your encryption, SQL injection, input validation, stack buffer overflows, and false-positive analysis. DAST conducts a fitness test of the OWASP Top 10 vulnerabilities like cross-site scripting, broken authentication, broken access control, and more.

To verify SAST, there are various tools like IBM App Scan, the Fortify static code analyzer, Code Scan, etc. To conduct DAST, tools like OWASP ZAP, Burp Suite, Checkmarx, etc., can be used.

The design-time fitness function is executed along with the CI pipeline, as shown in Figure 12-4. As I mentioned, you need separate unit test cases for functional requirements and the "-ilities," the both the test cases need to execute separately to get the execution metrics, and need to run SAST and DAST along with the pipeline.

The execution of functional testing before or after the fitness function depends on each pod team.

Fitness Function: Extensibility, Reusability, Adaptability, and Maintainability

A code quality test helps you identify fitness functions for extensibility, reusability, and maintainability. These tests are added to the pipeline to determine the relevant "-ilities." This set of fitness functions can serve as quality gates to prevent unmaintainable code in production.

Quality gate tools like SonarQube can be used to create fitness function for the "-ilities." You can configure a maintainability rating and reliability rating fitness function in the tool.

Fitness Function: Performance

The fitness function for performance should be defined during the discovery phase; not all services are required to perform in a similar way. Various tools and frameworks provide mechanisms to build tests and test load in a variety of scenarios. The performance fitness function should be executed as part of the CI/CD pipeline in a separate environment, and the configuration of the environment should mimic the production environment.

Tools like JMeter, Load Runner, etc., can be used to test the fitness function, and these tools should be configured as part of the CI/CD pipeline and use tools like Gatling to execute the fitness function early in the programmer development environment. Use a configuration environment that mirrors production for the performance fitness test.

Fitness Function: Resiliency

Use a fitness function for resiliency to identify and ensure the availability of an application during failure. This fitness function configures the code to handle tolerance and then retires. You can use load test tools to check the resiliency of your service. The metrics can be calculated as several successful versus unsuccessful requests. You can use Chaos Monkey tool to test the resilience fitness function.

Fitness Function: Scalability

Use a fitness function for scalability to ensure a service can scale based on user spikes. This fitness function configures containers and Kubernetes to handle the user load. The code must manage the state, configuration, etc., during the scalability of an application. You can use load testing to check the scalability fitness function. Create a matrix that shows the number of successful transactions versus unsuccessful transactions with the transaction round-trip time.

Fitness Function: Observability

A fitness function for observability ensures all the services in a system are monitored and send alerts, catch errors, and meet the architectural standards of observability. It will collect metrics across the application, infrastructure, and security environment. The metrics, such as all the observability parameters, are collected for successful versus missing parameters.

Fitness Function: Compliance

The fitness function for compliance ensures that domain-specific and country-specific compliances or regulations are met. A matrix can display whether compliance has been met or not (true or false).

A fitness function may come in the form of tests, monitoring, and the collection of metrics. Not all tests are fitness functions; only those that assess the "-ilities" are fitness functions.

The fitness function can be used to calculate various software metrics to determine whether an architecture continues to meet the "-ilities" requirements. For example, for cohesion, coupling, and maintainability, the fitness functions are cyclomatic complexity details and unit test as measurement to identify fitness. This fitness function helps you to identify whether refactoring is required.

Performance tests ensure that the architecture continues to meet your requirements and that any recent changes to the services have not negatively impacted its performance. Security tests can focus on the security parameters of your system to ensure that changes have not introduced any new vulnerabilities.

Using these various types of fitness functions provides an architect with information on the quality of the overall architecture as changes are introduced and it continues to evolve. These fitness functions provide a way to give a software architect confidence that the system continues to be capable and informs you if it is starting to decline in quality. Fitness functions facilitate the creation of an evolvable architecture.

For holistic fitness, the functions test multiple parts of the system all the time. An example of a continual holistic fitness function is Netflix's Chaos Monkey, which tests latency, availability, elasticity, resilience, scalability, and so on, in the cloud.

Fitness Function Metrics

Using the fitness measurements and a matrix of fitness functions provides a software architect with information about how fit the overall architecture is. The measurements give a software architect a way to calculate how fit their systems are. Table 12-1 provides insight into the requirements that can be collected for the fitness function test of one of my projects.

Table 12-1. *Fitness Function Metrics*

Fitness Function	**Details**	**Requirement**	**Measurement**
Cohesion and Coupling	Check coupling and cohesion through code quality.	Quality gates Unit test success: 100% Maintainability rating Reliability rating	Use SonarQube to measure against the quality gates. Write unit tests to check coupling and cohesion.
Availability	Check for high availability of your service.	Availability is 99.99% (four nines)	Measure by using **X=(n-y) *100/n** n = total number of minutes y = total number of minutes unavailable For example, 31 days/month N=31*24*60= 44,640 minutes, if a server is not available for 15 minutes in a month **X=(44640-15)*100/44640** **=99.96%**
Scalability	Scale in and out depending upon the load on your system.	Annual connection = 5,000,000 Average per day = 13700 Peak per day = 25000 Average day peak hour = 12500 Peak day peak hour = 13,500	Little's law: X=N/R The law says that if the box contains an average of N users and the average user spends R seconds, then the throughput is X. N = transaction R = seconds X = throughput transaction/per second (tps) X = 100/1200ms = 83.33tps
Performance	Check the overall performance of your system.	API server-side response time: <1 seconds DB calls: < 2 seconds Initial page load: < 4 seconds	Use load testing tools to measure the performance along with monitoring.

(*continued*)

Table 12-1. (*continued*)

Fitness Function	Details	Requirement	Measurement
Security	Check the overall security of an application; the security is different for each system.	OWASP Top 10 vulnerabilities Static security test Threat model Firewall Encryption	Use SAST and DAST tools to measure the security and use the threat model to create a threat analysis.
Observability	Monitor and alert across applications.	Integrated observability across application, infrastructure, and security.	Check observability dashboards, alerting, events, etc.

Review Function Metrics

After identifying and calculating the measurements of the fitness function, you need to schedule a meeting with the key stakeholders about the goal of conformance. In the meeting, you can check the relevance of the current fitness function, determine a change in the scale or magnitude of each fitness function, and decide if there is any better approach to measuring the fitness function.

Summary

In this chapter, I covered fitness functions and how they can be used in your product or project development. I covered how to determine whether the architecture continues to achieve the required "-ilities" and also provided a few examples on how to identify and measure the fitness functions.

You must adopt the following best practices when using fitness functions:

- You must define the fitness functions clearly with no ambiguity, and the relevant stakeholder must understand the fitness function for the project.
- The fitness function must be implemented efficiently.
- Each fitness function must be measured to demonstrate how fit a created architecture is when solving the problem.
- The fitness function must generate intuitive results.