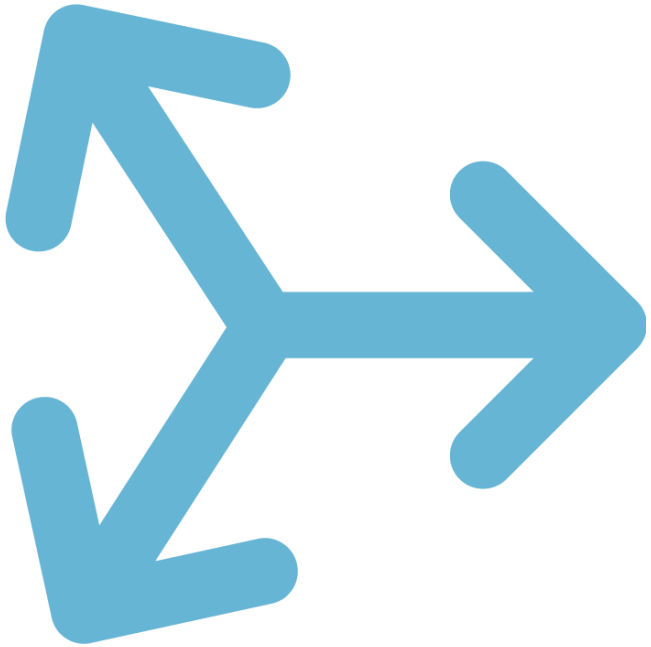


SOFTWARE ARCHITECTURE ASSIGNMENT 1

AUTOMATED FLOW EXECUTION (AFE) SYSTEM ARCHITECTURE

A SYSTEM THAT AUTOMATES EXECUTION OF FLOWS SUCH AS TEST SUITES FOR APPLICATIONS AND FLOWS FOR RESOURCE DEPLOYMENTS

AKHIL SUDHAKARAN - 2021MT12054 - 2021MT12054@WILP.BITS-PILANI.AC.IN

PURPOSE OF THE AUTOMATED FLOW EXECUTION (AFE) SYSTEM

The following are the goals of the AFE system:

- Automate running test flows on services and resources through a UI/command line to aid developers, testers and dev-ops engineers in an organization.
- Ability to run flows to deploy resources such as servers and databases that can be used by the test flows or even operation teams.
- To run unit tests already written for applications via the automated test flows.
- Integrate with other IT automation tools to synergize CI/CD pipelines.

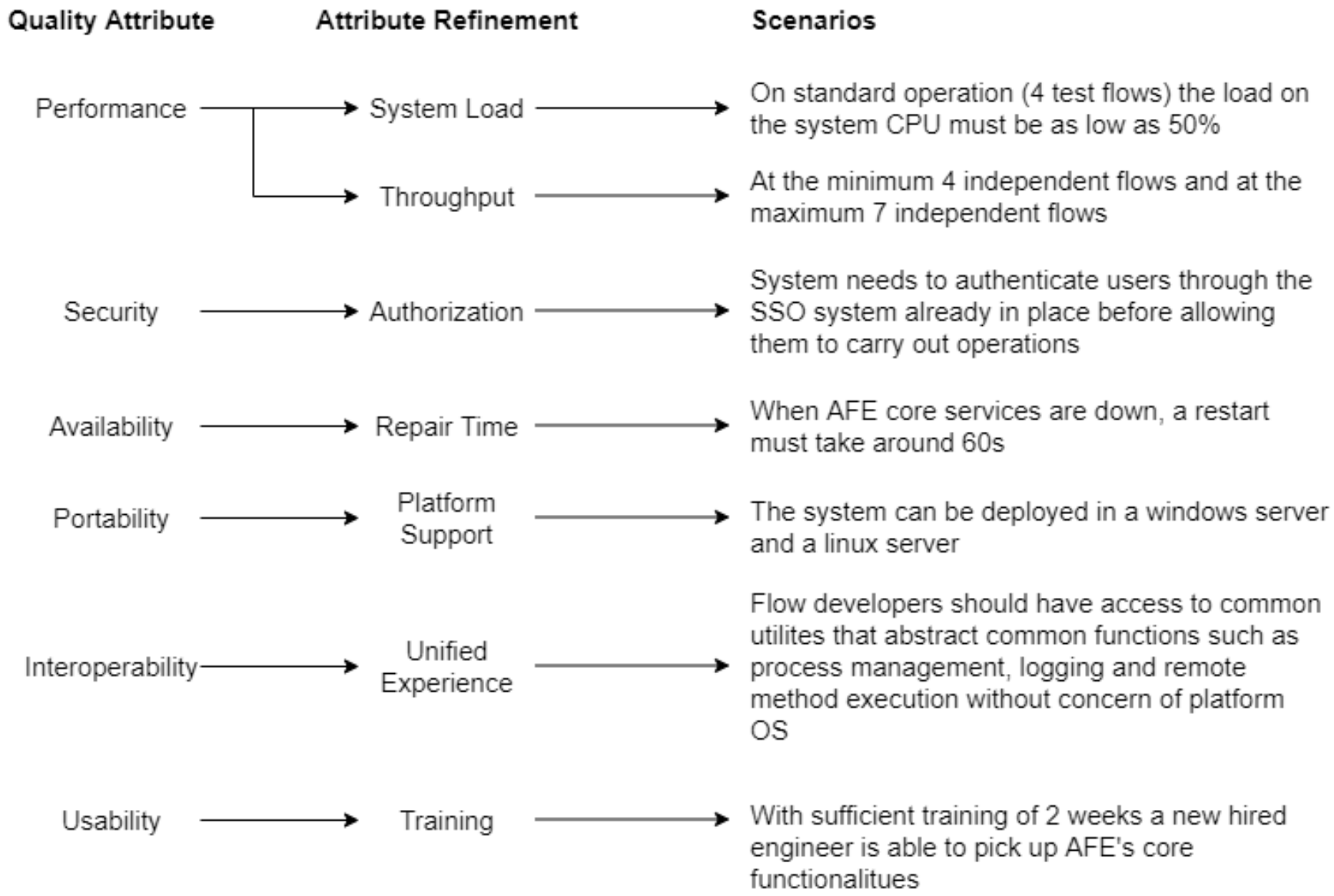
FUNCTIONAL REQUIREMENTS:

FR No.	FR Description
FR 1	System must run flows that can be written in languages such as Python. Flows can be test cases or a set of instructions to deploy services that developers need.
FR 2	System must have a UI that can be used by developers/testers/devops where they can initiate, monitor flow runs and view results.
FR 3	System must also have a command line tool to allow other IT automation tools such as Jenkins and Ansible to interact with it.
FR 4	System must allow users to run flows only if they are authenticated.
FR 5	System must have capability to notify stakeholders(through email) the status of flow runs.

NON-FUNCTIONAL REQUIREMENTS:

Quality	NFR Description
Performance	• System shall run test flows parallelly (At least 4 flows at a time). • Maintain CPU usage as low as 50% in standard workloads.
Security	The system must not allow certain operations based on permissions given to the user logged in.
Availability	MTTR of Less than 60 seconds shall be needed
Portability	Should be able to deploy system in both Windows and Linux platforms
Usability	System shall be easy to pick up and be used by new hires in less than two weeks of training
Interoperability	System shall abstract common services such as logging, remote communication to ease flow development for flows.

UTILITY TREE OF ASR



TACTICS TO ACHIEVE TOP 5 ASRS

Quality Attribute	Scenario (Attribute Refinement)	Tactics
Performance	On standard operation of 4 test flows the CPU usage must be as low as 50% to let the other services to use. (System Load)	• Caching data of older flow results so that requests for them are handled quickly and not affect other flow execution requests. • Making sure resource arbitration is done by placing flow execution requests on a FIFO queue since all flow requests are of equal priority and have mechanisms in place (such as timeouts and exception handling) to avoid other flows requests from getting stuck in the queue.
	A minimum of 4 independent flows should be able to run. (Throughput)	Can be achieved by using multi core processors on the app server and implement the flow execution engine with multithreading to make sure flows are run in separate threads.

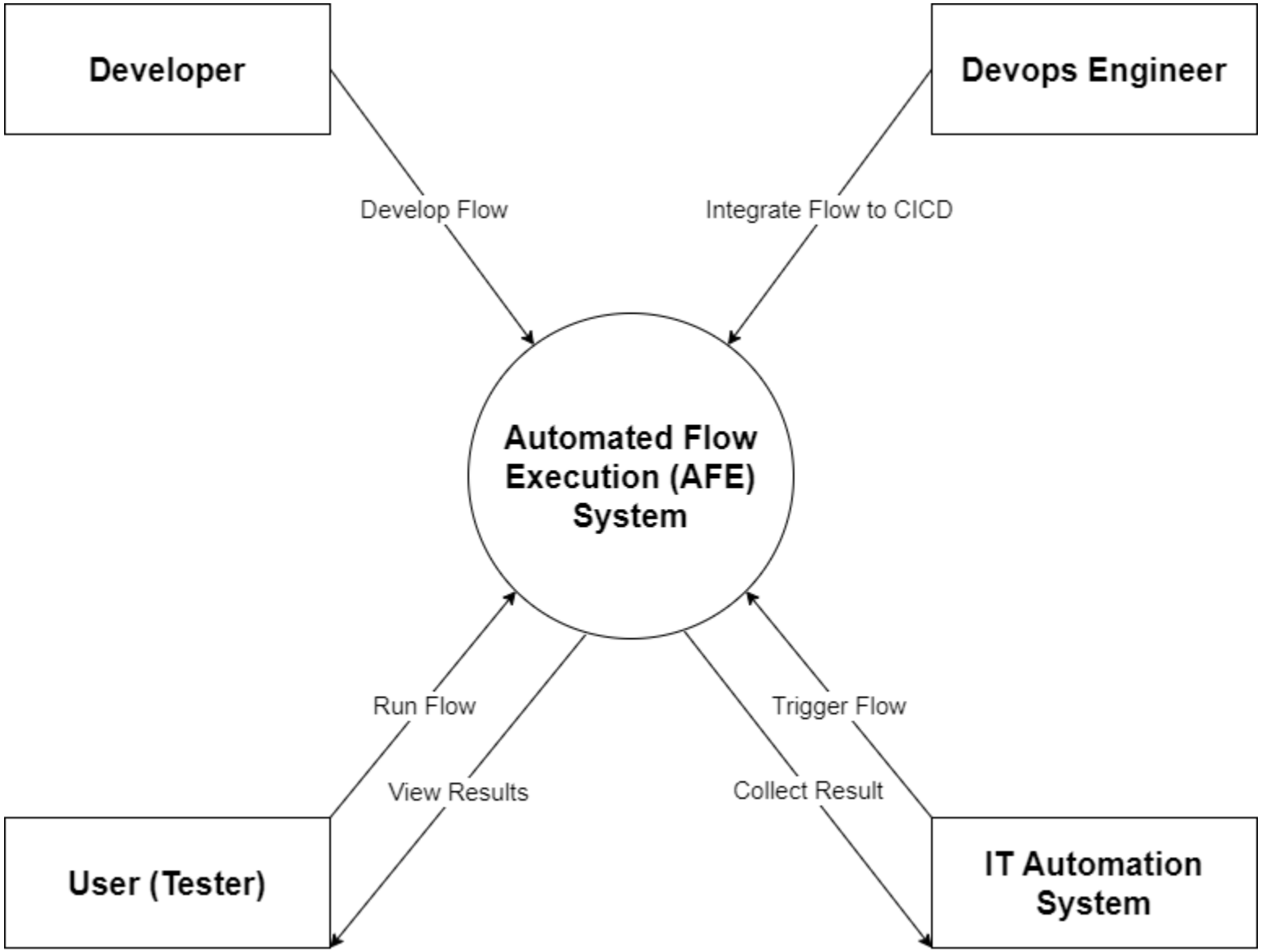
TACTICS TO ACHIEVE TOP 5 ASRS (CONT.)

Quality Attribute	Scenario (Attribute Refinement)	Tactics
Security	Authorized AFE users are assigned to groups which decide what flows the user can execute and what they cannot (Authorization)	• The controller that will handle requests to AFE system must be able to authenticate the requests by integrating it with the organization's SSO (Single Sign On). • When flows are developed one can assign role permissions to it. • Users can be put into the roles and groups which further helps in segregating the kind of flows one can execute.
Availability	When core AFE services are down the system must come back up in 60 seconds	• Make the core AFE system modules responsible for flow execution be deployed in fast readable SSD based storage. • Caching application state data so that the service restart is faster.

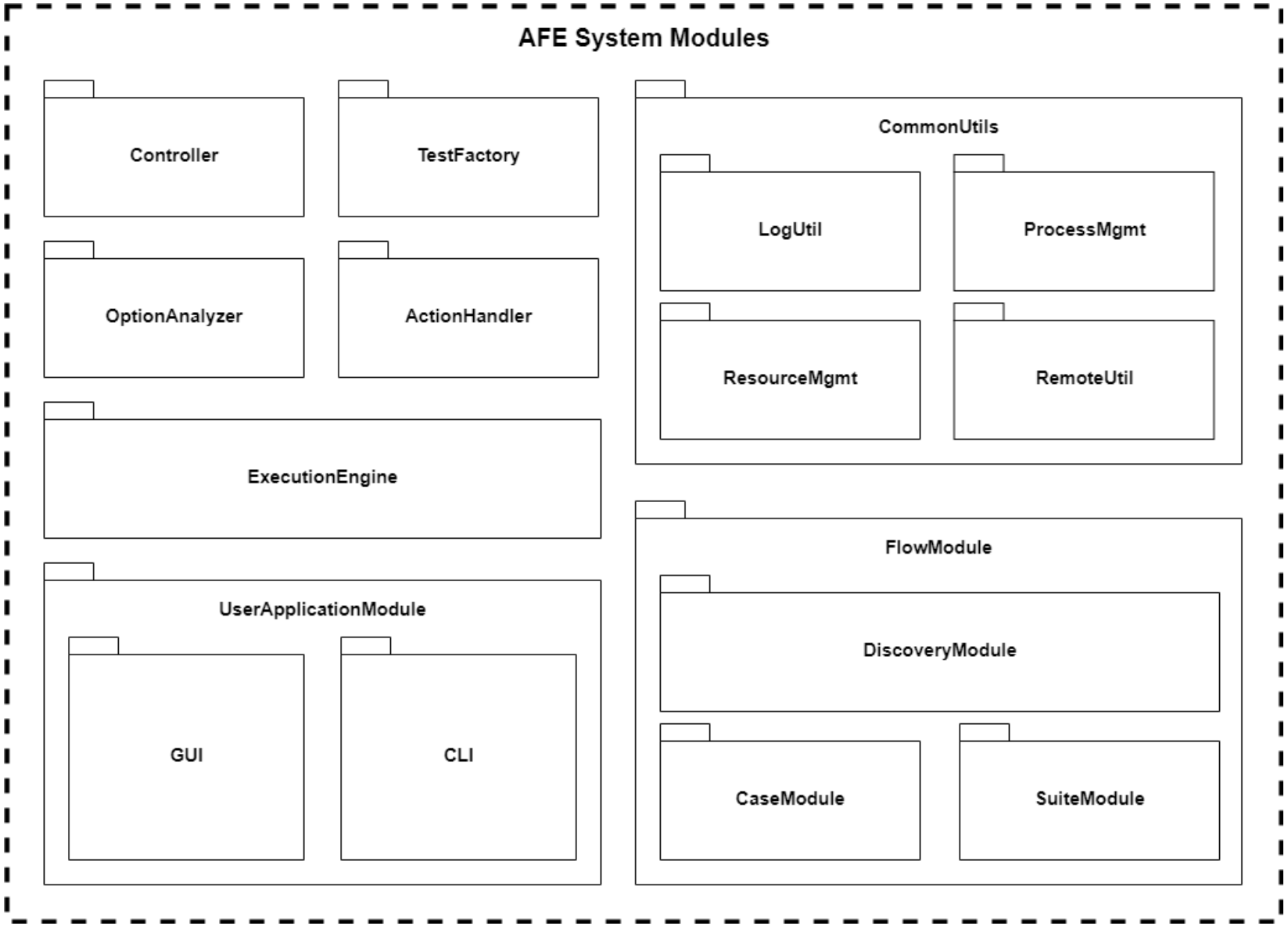
TACTICS TO ACHIEVE TOP 5 ASRS (CONT.)

Quality Attribute	Scenario (Attribute Refinement)	Tactics
Portability	AFE Services can be deployed in both Windows and Linux servers and the GUI is accessible on popular platforms.	• The AFE application core services can be implemented with cross platform frameworks such as spring in case of Java so that it can be deployed on both Windows and Linux servers alike. • The front-end GUI can be a web-based interface making it accessible on any device with a web browser.
Interoperability	When developers create automated flows that behave similarly across platforms, then the flow must work on all these platforms (Unified Experience)	Utilities for logging, remote command execution, process management and file management must be abstracted so that flows can use these functions without worrying about the target platform these flows would run on

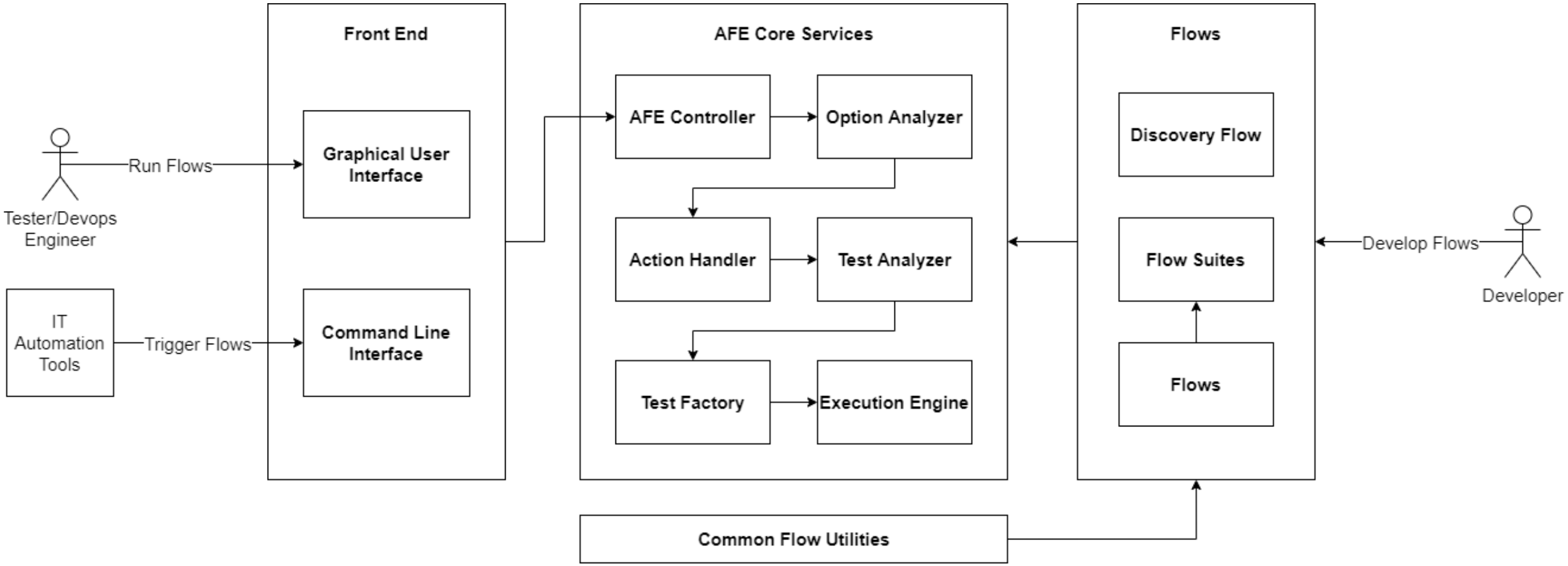
CONTEXT DIAGRAM



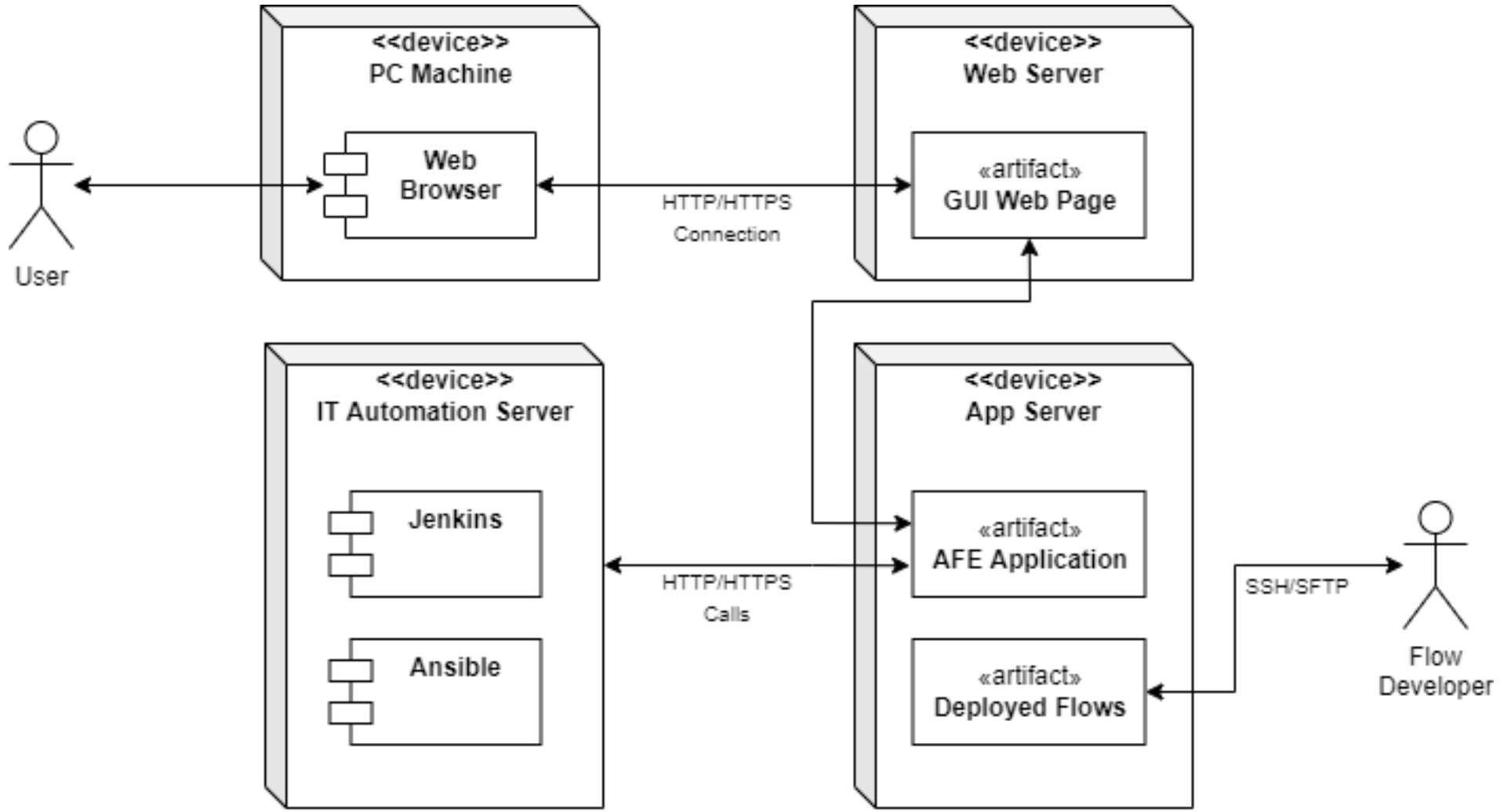
MODULE DECOMPOSITION



COMPONENT CONNECTION DIAGRAM



DEPLOYMENT DIAGRAM



HOW DOES AFE WORK?

The AFE system runs flows on resources. This can be test cases or instructions to deploy services or even cleanup tasks. This is achieved by the interactions of the following components:

- The developers develop **flows (written in Python/Java)** and aggregate related flows into **flow suites** which can be triggered by the users of AFE system. These flows are packaged and deployed in the app server.
- The **AFE Controller** has endpoints that takes in the requests from the GUI/CLI tools and sends the parameters of the flow execution request to the **Option Analyzer.**
- The **Option Analyzer** validates all the option combinations provided by the user through the GUI/CLI and if the user can run the requested flow. If the options are valid then it is sent to the **Action Handler,** else error is sent back to the controller to notify users.
- The **Action Handler** is responsible for pre flow execution tasks such as checking if the resources on which the flow needs to be run on exists and to even check if AFE is functioning properly by checking the logs. After these checks are done the handle goes to the **Test Analyzer**.

HOW DOES AFE WORK? (CONT.)

- The **Test Analyzer** checks for issues with the flow packages that are deployed on the AFE system. If there are issues with the flow packages deployed, then the flows are not run, and a notification is sent to teams responsible for flow deployments. If there are no issues with the flow packages, then the request is sent to the **Test Factory.**
- The **Test Factory** builds the execution queue, analyzes the dependencies between flows and flow suites to be executed and constructs the best sequence for executing the flows and then hands over the control to the **Execution Engine**
- The **Execution Engine** is the most critical part of the AFE system. It is responsible for scheduling the run order of flows, parallelism handling, reservation of resources, updating relevant log files with status and handling failures. The status of the flows can be checked via the GUI/CLI via the **exposed controller endpoints**.
- Apart from the above core requirements for AFE to work, there would be **common utilities** developed which help in abstracting things such as logging, process management, remote command execution etc. so that same flows can be run on multiple platforms. This improves code maintenance.

KEY LEARNINGS

- Apart from thinking about functional requirements it is important to figure out the nonfunctional requirements that can potentially change the way the system is modelled. For example, performance requirements can affect how an entire system is designed and knowing about it early and having the intuition to think about such potential requirements is very important.
- Knowing how the system acts as a backbox through a context diagram helps all stakeholders understand the bigger picture. It helps in clearly showing what belongs in a system and what does not.
- The analysis of the system in different views (Module decomposition, utility trees, component connection etc.) can be templated and be reused for other similar systems. This helps in deriving more cost vs benefit of different architecture approaches for similar problems in the future.
- The module view and component connection diagrams help developers visualize implementation strategies much earlier in system designing and makes planning (especially in AGILE) much easier.
- The deployment view gives a clear image to operation teams on how the application modules are deployed and helps in rough estimation of cost early on easing planning.