A.Y. 2025-2026 Software Engineering for HPC Project: goal, schedule, and rules

Table of content

1. Goal and approach

The objective of this project is to apply in practice what you have learned during the entire course about requirement engineering, architectural design, automated testing, containerization, and the usage of HPC clusters through SLURM.

The final goal is not necessarily to deliver working software/scripts, but, rather, to try possible solutions, reflect on conceptual and technical problems, and, in general, focus on learning by doing.

The exercise simulates a response to a "Call for Tenders" by the European Space Agency (ESA).

The use of Generative AI/Large Language Models (e.g., ChatGPT, Claude, Copilot) is allowed for writing code, tests, and pipeline scripts. However, a strict rule of **accountability** applies: during the project discussion, which will also consist of a code review, you must be able to explain, justify, and live-edit the code, test, or configuration file you have submitted. The detailed code of conduct for the usage of Generative AI (GenAI) tools is available in Section 7.

Remember that you are the sole authors and owners of the artifacts you deliver.

2. Rules

- The project is meant to be developed in groups composed of two to four students each. If a student, for any reason, is aware that they cannot coordinate with others and, therefore, cannot be part of a group, they can take the project as an individual and inform us in advance. This, however, should be considered a last resort, as the work of a software engineer is inherently collaborative and benefits from discussion with multiple stakeholders. For this same reason, larger groups are preferable.

- You must provide your artifacts within the stated deadline. A delay of a few days will be tolerated, possibly resulting in a penalty in the final score. Special cases will have to be discussed with the professor.
- The project is composed of two parts. Part 1 concerns the development of a requirement analysis and design document and assigns up to 8 points. Part 2 can be taken in two different variants:
    - *Standard track*: this concerns creating a Continuous Integration/Continuous Deployment pipeline to automate the execution of a reference implementation of the project we will provide. This variant assigns up to 8 points.
    - *Full track*: this includes developing a viable prototype of the proposed project and creating the corresponding tests and a Continuous Integration/Continuous Deployment pipeline. Such an implementation can be developed in either C++ or Python. The software structure should be explained, referring to the software architecture defined in Part 1. Differences and simplifications should be described in an accompanying document (the README in the GitHub repository). You are free to use any library or framework you find useful for your case. You should privilege stability and quality of your code, rather than the implementation of a large quantity of features.

      This variant assigns up to 16 points for the implementation and up to 8 points for the CI/CD pipeline. Therefore, this variant allows you to complete the exam without taking the written part.

3. The problem: "AstraLog-HPC"

Modern space missions generate a massive, continuous stream of telemetry data. To ensure spacecraft safety, we focus on developing **AstraLog-HPC**, a software that identifies anomalies in aircraft subsystems. AstraLog-HPC acquires data streamed from sensors installed in each subsystem and executes a set of monitoring rules based on the absolute values of the received data, as well as their temporal variations and correlations.

To simplify time management while keeping the problem realistic, we assume a unified synchronous system clock. Moreover, we assume that measurements across all sensors are equally spaced in time (e.g., exactly 1 measurement per second) and perfectly aligned (all sensors produce data for the entire mission duration without any interruptions). Therefore, the *n*-th row in each sensor stream corresponds to the same timestamp.

Each spacecraft can contain multiple sensor types (e.g., temperature, pressure, voltage, ...), each potentially available in multiple instances. The specific type and number of sensors to be managed are available in a configuration file called sensors.yaml (to be made available by the instructors) and do not change during operation. Each sensor continuously produces data, and the spacecraft broadcasts a continuous data stream to the ground station. Each piece of data is transmitted as a **JSON packet** with the following structure:

```
{
 "timestamp": "2025-11-15T12:00:00Z",
 "sensor_id": "TEMP-01",
 "value": 25.5,
 "priority": "HIGH",
}
```

In real-world space communication, data transmission is often noisy and subject to corruption. Corruption cases to be considered are the following:

- **Malformed JSON:** truncated strings or invalid JSON syntax.
- **Schema Errors:** packets missing mandatory fields (e.g., missing sensor_id).
- **Type Errors:** invalid data types (e.g., a string instead of a numerical value).

Monitoring rules can be of the following types and are stored in a rules.json file readable by AstraLog-HPC.

1. Simple Rule (Absolute Threshold) checks if a value exceeds a fixed threshold at a given time.

Example: The temperature measured by sensor TEMP-01 exceeds 50.0 °C.

```
{
  "rule_id": "R1",
  "type": "simple",
  "sensor_id": "TEMP-01",
  "operator": ">",
  "value": 50.0,
  "priority": "MEDIUM"
 }
```

2. Step Difference Rule (Relative Variation): Since measurements are equally spaced, this rule completely bypasses explicit time management. It simply checks the difference between the current measurement and the previous measurement at T_{n-1} . are equally spaced, this rule completely bypasses explicit time management. It simply checks the difference between the current measurement at T_n and the previous measurement at T_{n-1} .

Example: The pressure drops by more than 2.0 units compared to the immediately preceding measurement.

```
{
  "rule_id": "R2",
  "type": "step_difference",
  "sensor_id": "PRES-01",
  "operator": "<",
  "value": -2.0,
  "priority": "LOW"
 }
```

3. Stateful Rule (Measurement Persistence): An anomaly is triggered only if a specific simple condition persists uninterrupted for a given number of consecutive measurements. This introduces the concept of state and memory over the data arrays.

Example: The main voltage drops below 20.0V and stays below this threshold for at least 5 consecutive measurements. The alarm is triggered exactly on the 5th measurement (and on any subsequent continuous violation).

```
{
  "rule_id": "R3",
  "type": "stateful",
  "sensor_id": "VOLT-MAIN",
  "operator": "<",
  "value": 20.0,
  "consecutive_measurements": 5,
  "priority": "HIGH"
 }
```

4. Logical Correlation Rules: Combines the Boolean outcomes of other evaluated rules at the same timestamp T_n using standard logical operators (AND / OR).

Example: Critical risk triggered if, at the same timestamp, the temperature is > 50°C (Rule R1 evaluates to TRUE) AND the pressure just dropped rapidly (Rule R2 evaluates to TRUE).

```
{
  "rule_id": "R4",
  "type": "correlation",
  "logic": "AND",
  "conditions": ["R1", "R2"],
  "priority": "HIGH"

}
```

Each monitoring rule should belong to one of the four types above and should be identified by a unique identifier (rule_id). Optionally, a rule can have a priority, which can assume the values low, medium, or high. Such priority applies within a single rule type and means that rules with higher priority are evaluated before the others.

AstraLog-HPC should receive streamed data, filter out syntactically invalid packets and accumulate valid packets into a local batch file; then, when the batch reaches the size you specify, it should apply the four types of rules to all collected valid data. Two cases are possible:

- If no rules are violated (none of the simple, step, stateful, or correlation rules trigger an alarm), the overall system state is considered nominal. In this case, the AstraLog-HPC appends a single aggregated line to the output file valid_data.csv, representing the spacecraft's healthy state and including the values collected by all sensors for that timestamp.
  Format:
  TIMESTAMP;NOMINAL;[SENSOR_1]:[VALUE_1]|[SENSOR_2]:[VALUE_2]|...

Example:
2025-11-15T12:00:00Z;NOMINAL;TEMP-01:25.5|PRES-01:101.3|VOLT-MAIN:24.1

- If one or more rules are violated, the system is in an anomalous state. Nothing is written to valid_data.csv for this timestamp. Instead, AstraLog-HPC appends one line to the file alarms.log for *each* rule that evaluated to TRUE.
  Format:
  TIMESTAMP;RULE_ID;PRIORITY;VIOLATED_SENSOR(S);CURRENT_VALUE(S)
  (Note: For correlation rules, list all sensors and values involved in the parent rules).
  Examples:
  2025-11-15T12:00:05Z;R1;MEDIUM;TEMP-01;51.2
  2025-11-15T12:00:05Z;R4;HIGH;TEMP-01,PRES-01;51.2,98.0

Note: To ensure the software's verifiability, the output formats must be strictly adhered to.

The execution environments for AstraLog-HPC include classical machines/virtual machines, as well as an HPC cluster accessible through the SLURM job scheduler. As part of the project, you should define how the AstraLog-HPC components are deployed within these environments.

4. Phase 1 – Requirement analysis and design document

The result of this phase will be a document that will include at least the following elements. You can extend this structure if you think it is needed:

- **A front page** that includes the project title, the version of the document, your names, and the release date.
- **A table of contents** that includes the headers of the first three levels of headings in your document, with the corresponding page number.
- **Section 1. The project and project goals.** You can copy and freely re-elaborate (if needed) on the description you find in this document.
- **Section 2. Requirement analysis.** It will include:
  - 2.1. Relevant human and non-human actors
  - 2.2. Use cases (include a use case diagram and the detailed description of the use case you feel is the most critical)
  - 2.3. Domain assumptions
  - 2.4. Requirements (2.4.1. Functional requirements, 2.4.2. Non-functional requirements)
- **Section 3. Design.** It will include:
  - 3.1. General description of the architecture. It will include a component diagram and a description of components.

    - 3.2. Sequence diagrams. Include at least two sequence diagrams that describe the most relevant interactions (e.g., evaluating a stateful or a correlated rule).
    - 3.3. Critical points and design decisions. Document here the aspects of the system you felt were important and drew particular attention; describe and motivate your main design decisions and a rationale.
- **Section 4. Work distribution and effort**. Include a table describing how the work was split among team members and how many hours you spent per task.
- If applicable, **Section 5. Usage of AI**. Describe how you used AI for the requirement analysis and design (see the rules in Section 7).

5. Phase 2 – Standard track

The objective of this track is to create a CI/CD pipeline for a simple AstraLog-HPC implementation, which we will make available on GitHub. To create your own GitHub repository, you will use a template that will be made available on April 10 (an announcement will be published on Webeep). Name your repository using the surnames of all group participants, for instance: BrownRossiXiang if the surnames of the three group members are Brown, Rossi, and Xiang, respectively, and set it as private. Add all team members and the instructors' accounts dinitto and SimoneReale as contributors.

The template will include a simple implementation of AstraLog-HPC, test cases, and a README file with additional information.

You will go through the following steps.

Step 1 – From build to release and manual job execution

Your tasks:

Automating the build, test, and release processes:

- Create a CI/CD pipeline that builds and tests the project on every push.

Containerizing the application:

- Write a Singularity container definition for the telemetry processing tool and include it in your repo.
- Extend the CI/CD pipeline to build the container image from its definition.

Executing on the Galileo 100 cluster:

- Write a job.sh script to run your containerized application on SLURM.
- Ensure standard output and error are redirected to text files.
- Transfer the script and container to Galileo100.
- Submit the job and verify the results.
- Push job.sh and the output files to your GitHub repository.

Step 2 – Automating Job Submission with Containerization:

You will extend the CI/CD pipeline to fully automate the process from a GitHub push to execution of the containerized application on SLURM.

You must:

- Move the container from the GitHub runner to the cluster (e.g., using scp or a Singularity registry).
- Handle credentials securely using GitHub Secrets (**never hard-code tokens or passwords**).

Step 3 – Finalization

- Clean up your repository as needed.
- Create a docs folder to store the latest version of your requirement analysis and specification document.
- Associate an open-source license with your repository and the files you have created in this repository.
- Create a README file that includes the following: the names of all team members, the selected track (standard in your case), the role of each team member in the project, the detailed activities performed, and the effort spent in hours. Moreover, describe your pipeline and present the difficulties you have faced. For those you have overcome, explain how you did it; for the others, describe the attempts you made. **Crucially, include here also a detailed paragraph on how you used GenAI during the project, if any.**
- Fill out the submission form at https://forms.office.com/e/mfBZqj4gB2, including, once again, information about all team members and the link to your repository. Only one representative of each team will fill in this form on behalf of all their mates.

This part will be assessed by analyzing the defined pipeline and executing it to verify that it works as expected. The analysis will be conducted during the project discussion and will take into account your ability to describe the issues you faced and to identify possible solutions. Remember that, according to the declared goal of the project, you should show that you have tried possible solutions, reflected on conceptual and technical problems, and, in general, learnt from the project.

6. Phase 2 – Full track

The objective of this phase is to create your own implementation of AstraLog-HPC, define and implement the corresponding test cases, and create the CI/CD pipeline for your project.

Preliminarily, create your GitHub repository either from scratch or using the template made available for the standard track (see Section 5). Name the repository using the surnames of all group participants, for instance: BrownRossiXiang if the surnames of the three group members are Brown, Rossi, and Xiang, respectively, and set it as private. Add all team members and the accounts dinitto and SimoneReale of the instructors as contributors. Include in the repository at least the following folders:

- src, where you will store the code you will produce;

- test where you will store all test cases;
- docs where you will store the latest version of your requirement analysis and specification document.

Groups composed of two to three students will focus on the functionality of rule processing using the input data we provide in .csv files. The rule-processing component will run on G100.

Groups composed of three students should also focus on improving rule processing performance by distributing/parallelizing tasks as much as possible.

Groups composed of four students, in addition to what is required for groups of two to three students, will also manage the ingestion of real-time data from a spacecraft digital twin we will make available by April 10. The digital twin broadcasts this data through a HiveMQ MQTT Broker. Spacecrafts broadcast their telemetry to a specific MQTT topic (e.g., esa/astralog/telemetry). Note that, as mentioned, the system assumes a unified synchronous system clock, but, in practice, packets from different sensors may arrive at slightly different sub-second intervals. This should not be a problem as we threat time at the second granularity.

As part of the development process, all groups are expected to develop the software needed to automate test execution.

In the finalization phase, you will follow these steps:

- Clean up your repository as needed.
- Associate an open-source license with your repository and the files you have created in this repository.
- Create a README file where you will include the names of all team members, the selected track (full in your case), the role of each team member in the project, the detailed activities performed, and the effort spent in hours. In the README you will also:
    - Specify the programming language you have used and any library/framework you have adopted;
    - Describe the organization of your code, explaining how it relates to the software architecture you have defined in the requirement analysis and specification document;
    - Make explicit any simplification or variation with respect to that architecture;
    - For groups of three or four students, how you handled distribution/parallelization of your code;
    - Present your test cases and the rationale behind their choice;
    - Describe your pipeline;
    - Present the difficulties you have faced, for those you have overcome, how you did it, for the others, the attempts you made;
    - **Crucially, include here a detailed paragraph on how you used GenAI during the project, if applicable.**
- Fill out the submission form at https://forms.office.com/e/mfBZqj4gB2, indicating, once again, the information for all team members and the link to your

repository. Only one representative from each team will fill out this form on behalf of all their teammates.

This part will be assessed by analyzing the code and the corresponding documentation, running the test cases, analyzing the defined pipeline, and executing it to verify that both the pipeline and the software work as expected. The evaluation will be conducted during the project discussion. It will take into account your ability to describe the issues you faced and to identify and defend the solutions you propose. Remember that, according to the declared goal of the project, you should show that you have tried possible solutions, reflected on conceptual and technical problems, and, in general, learnt from the project.

7. Code of Conduct for the use of Generative AI tools

You are welcome to use any tools that help you prepare high-quality documentation and software. However, please keep in mind the following key principles:

1. **Responsibility for content:** You are solely responsible for the entire content of your software and documents, including every line of code, all text, UML diagrams, figures, and references. While you may use any tool to assist your work, you must ensure that all material is correct, original, and fully understood by you. During the project discussion, you should be able to explain, justify, and defend every aspect of your work in detail.
2. **Use of Generative AI**: Generative AI tools—including, but not limited to, Large Language Models (LLMs)— can be useful for certain supporting tasks, such as improving writing style, generating summaries, suggesting code snippets, or drafting diagrams. However, they cannot replace your own reasoning or development work.

   If you use Generative AI tools in your work, you must include a dedicated section in your documents clearly describing:

   - Which tool(s) did you use (e.g., specific chatbots, such as ChatGPT, Gemini);
   - The inputs you provided (e.g., prompts, datasets, source materials, parameters, or constraints);
   - The outputs you obtained;
   - How did you verify, refine, and integrate those outputs into your project?

Important notes:

- Some tools—including free or online tools—may store input data for future model training. Use them cautiously and ensure privacy and confidentiality are maintained.
- Generative AI tools may produce inaccurate or misleading (“hallucinated”) outputs, including errors in code, diagrams, or figures. It is your responsibility to verify the accuracy and integrity of all generated material.

All tools must be used ethically and responsibly. Compliance with this code of conduct will be evaluated, and violations may affect your final grade, up to and including failure of the project in severe cases.

8. Steps and deadlines

- **Group formation**: constitutes the project groups. If needed, you can use the course forum to post messages to look for mates. Ensure all group members agree on the variant to use for Part 2. After forming the group, fill in thoutform httpat s://forms.office.com/e/6k3kHgiT39. This is expected to be filled in by a single representative for the whole group. You will include there the names, emails, and person codes (please ensure they are correct) of all group members. Deadline: **15/04/2026.**
- **Access to the CINECA cluster**. You must register for the CINECA UserDB portal at https://userdb.hpc.cineca.it/, using your Polimi email and indicating Polimi as your reference institution. During the registration procedure, you will be asked to upload a scan of your identity card. Detailed instructions are available here https://docs.hpc.cineca.it/general/getting_started.html#get-str-card. The task to be completed now corresponds to the first step in the tutorial. After completing the registration, please fill out the following form: https://forms.office.com/e/r1GRNKvaqd so we can add your account to the project dedicated to our experiments. Deadline: **18/04/2026.** Note that each group member should do this task individually and independently from the other members. After adding your account to the project, we will inform you. Your next step will be to configure 2FA (two-factor authentication).
- **Requirement analysis and design document**: preliminary submission by **29/04/2026**. Uploading your PDF file on webeep, here. This will serve as an intermediate checkpoint. You can continue improving your document after this submission, if needed. We will provide informal feedback in the case we spot issues in the intermediate versions. You can also ask questions explicitly in case of doubts.
- **Standard track**: to be completed by **8/06/2026**; discussion with the instructor and score assignment to be agreed upon in the **week of June 8, 2026**. The discussion will concern both parts of the AstraLog-HPC project.
- **Full track**: to be completed by **14/06/2026**; discussion with the instructor and score assignment to be agreed upon in the period **June 15-July 3, 2026**. The discussion will concern both parts of the AstraLog-HPC project.