17th International Conference on Space Operations, Dubai, United Arab Emirates, 6 - 10 March 2023.

SpaceOps-2023, ID # 322

Applying Continuous Integration for Operational Products in the Mission Preparation Environment

Annabell Langs[a*], Jan Oertlin[b], François Trifin[c]

[a] *Telespazio Germany GmbH, Europaplatz 5, 64293 Darmstadt, Germany*, annabell.langs@telespazio.de
[b] *Telespazio Germany GmbH, Europaplatz 5, 64293 Darmstadt, Germany*, jan.oertlin@telespazio.de
[c] *European Space Agency / ESOC, Robert-Bosch-Str. 5, 64293 Darmstadt, Germany*

Abstract

During the mission preparation phase a significant part of the effort of Flight Control Teams (**FCT**) is spent on the production of operational products, such as the operational spacecraft tailoring data and flight operations plan, followed by the verification and validation of the related mission operations systems.

As the European Space Agency (**ESA**) is migrating to the ESA Ground Operation System based on the European Ground Systems Common Core (**EGOS-CC**) for its missions, modern software design and development principles are being applied to its infrastructure software and to the preparation and validation of the related operational products.

The software industry's established principles behind Continuous Integration (**CI**) allow for an increased efficiency of software production. Similar efficiency gains are therefore expected for the production of operational products managed by the Flight Control Teams.

In this paper, an integrated workflow across the preparation environment suitable for all ESA missions using EGOS-CC based systems is presented and discussed. It is outlined how existing ESA software systems have been adapted and integrated in order to provide a continuous integration pipeline allowing the automatic verification and validation of the operational products.

Using the OPEN Preparation Environment (**OPEN**) tools, the Flight Control Teams can edit the operational products, such as the tailoring data, and push them into the version control system. This triggers the continuous integration, verification and validation pipeline using the GitLab DevOps platform. Management of the pipeline and its associated workflow is covered through the web-based Portal-M system as part of OPEN-M (Operations Preparation Environment for EGS-CC-based missions). This portal allows the planning and management of change requests for the operational products covering the EGOS-CC specific tailoring data formats, which are used by and are part of these products. Further, it supports the user by providing a holistic view of the ongoing mission's changes. From this the state of any change, including validation results, can easily be assessed.

While the focus of the Assembly, Integration and Validation Environment (**AIV**) so far has been the validation of the software infrastructure such as the mission control system, the extended integration with the OPEN-M environment allows the evolution of the AIV activities towards continuous validation and delivery of operational products thereby further automating the Flight Control Teams' workflows in the mission preparation environment.

Keywords: Continuous Integration, Operational Products, Preparation Environment

Acronyms/Abbreviations

AIV ..... Assembly, Integration and Validation Environment
API ..... Application Programming Interface
APID ..... Application Process Identifier
ARVALIS ..... Automated Rule-Based Cross-Validation of Operational Data
CD ..... Continuous Deployment
CDe ..... Continuous Delivery
CI ..... Continuous Integration
CLI ..... Command Line Interface
EGOS-CC ..... ESA Ground Operation System based on the European Ground Systems Common Core
EGS-CC ..... European Ground Systems Common Core
EMF ..... Eclipse Modeling Framework

ESA ..... European Space Agency
ESOC ..... European Space Operations Centre
FCT ..... Flight Control Team
FOP ..... Flight Operation Plan
GSTP ..... General Support Technology Programme
HTTPS ..... Hypertext Transfer Protocol Secure
OPEN ..... OPEN Preparation Environment
OPEN-M ..... Operations Preparation Environment for EGS-CC-based missions
RCP ..... Rich Client Platform
REST ..... Representational State Transfer
SA ..... Spacecraft Analyst
SCOS-2000 ..... Spacecraft Control & Operation System 2000
SOE ..... Spacecraft Operation Engineer
SOM ..... Spacecraft Operations Manager
YAML ..... YAML Ain't Markup Language

1. Introduction

The complexity of software systems has been increasing rapidly in recent years. Parallels to this evolution can be seen in the operation of spacecraft as the spacecraft have become more capable with multiple subsystems each fulfilling a different functionality and entailing parameters, commands, and flight operations procedures. This leads to an increase in data dependencies and complexity which in turn leads to a higher risk of hazardous, unintended consequences when implementing changes to that data. In operations, typically most of this operational data is prepared before the launch of the spacecraft in the preparation environment, a process that so far takes person-years of effort to validate the databases and flight control procedures through manual reviews and (often manual) tests. The software industry, equally being exposed to increased complexity and testing requirements, has found solutions in terms of tooling, processes and methodologies in order to alleviate associated risks.

One of these approaches is the adoption of CI which is generally understood as a development best practice where developers integrate code into a shared repository frequently. Each submitted change is automatically built and tested. Beyond automatic integration, Continuous Delivery (**CDe**) extends this model by a release process that allows providing the application reliably at any time. Continuous Deployment (**CD**) even automatically deploys the application to the production environment following all other process of CI and CDe.

The CI principles can also be applied in the preparation environment: changes made to operational artefacts are regularly pushed to their mission (source) repositories from where they are automatically tested, and the results presented to the FCT members. By validating all changes automatically, including verification within the AIV environment, it is ensured that problems (e.g. side effects that are hard to detect because they are caused by transitive data dependencies) can be detected early before ending up in the production environment. This streamlines the process of integrating new changes and makes it more efficient since it reduces the manual effort. It thereby allows the FCT members to concentrate on issues that cannot be automated and that bring real business value. Applying these CI/CD concepts to mission tools thus will help with reducing manual work, improving the deployment of operational products, and speeding up the integration and testing stage of changes.

Changes must be made in several operational products which are required to configure and instantiate activities with the spacecraft and ground systems. In the context of this activity, an operational product is a package of operational artefacts which is used in operational environments. Operational artefacts include but are not limited to the spacecraft tailoring data containing telemetry, telecommands, activity lists, flight control procedures (FCP), and scripts. Operational products might also contain artefacts that are only used for verification purposes such as simulator scripts, breakpoints, verification specification files and will only be deployed in an AIV environment. Packaging these artefacts into operational products allows for traceability, versioning, and release management. In terms of software development, the packaging can be thought of as the output of a build step processing source files (operational artefacts) into a product for easier dissemination.

Ground and Flight Control teams are working with a variety of mission operation systems to produce, validate and manage operational products. These systems are often focused on solving a particular step of the overall workflow.

They fit a specific purpose but do not necessarily integrate with other systems. This has resulted in a landscape of many specialised products and tools where the gaps between them often have to be filled by manual steps to be executed by the end users. In order to apply continuous integration, processes and tools need to be automated and integrated which in turn leads to an increased efficiency of the FCT which is relieved of some manual work. This trend can also be seen in the software industry where DevOps platforms such as GitLab have gained popularity which integrate all development lifecycle steps into a single easy to access web interface.

The objective of this work is on creating a CI environment that integrates existing systems and workflows to manage changes to operational products followed by automatic verification and deployment. This paper focuses on the concepts and integrations of the different systems and processes of the mission preparation environment, particularly those used by the FCT members at the European Space Operations Centre (**ESOC**).

This paper starts by giving a brief overview of the current state of CI/CD in the preparation environment and continues by describing the relevant environments and software systems that have been considered. It then outlines how CI is applied in the operational product development lifecycle and finishes with a conclusion and outlook.

2. From Manual to Continuous Integration

Operational artefacts such as the spacecraft tailoring data and flight control procedures undergo rigid testing before being put into operations. During this preparation phase, the FCT members make use of various tools such as data editors, management tools, scripts and automation systems. With the move from the Spacecraft Control & Operation System 2000 (**SCOS-2000**) to EGOS-CC - and this work's implementations are targeted towards the future system - it is possible to improve current tools and processes by consolidating them without too much interruption. This is because end users will expect changes in tooling anyway and are thereby more open to changes affecting their day-to-day routines. FCT members of the Sentinel missions have been included during the activities' development process with the goal of enhancing existing concepts with continuous integration features rather than completely changing them. Thereby later end users can better draw from their existing experience when using the new EGOS-CC software landscape. In the following, the (necessary) changes from manual to continuous integration are outlined, highlighting detected shortcomings and how these can be alleviated based on user feedback (see also Figure 1).

As a first step, the FCT plans the changes to be integrated using a change management system in order to synchronize the work of the team members. There are different types of so-called change requests that can be handled by the system (e.g. with regards to flight control procedures or the spacecraft data). While the system integrates some information from the Flight Operation Plan (**FOP**) and the spacecraft data, its integration is rather high-level and in parts manual (e.g. when the data changes). In addition, it does not integrate with any version control system where the changes themselves are managed.

To achieve continuous integration, the changes need to be managed in a single mission (source) repository. This is integrated with the change management system so that each change can be traced in the version control system and dependencies can be more easily detected and handled. Pushing changes in operational artefacts to a single mission repository allows to trigger automatic processes as part of a CI/CD pipeline and display its results in the scope of the change request so that immediate feedback is accessible. The web-based Portal-M system as part of the Operations Preparation Environment for EGS-CC-based missions (**OPEN-M**) [1] extends the existing change management system with these integrations and provides a holistic view on the changes of operational artefacts and products.

After having planned the necessary changes for the next versions of the products, the FCT members use different editors tailored towards the data to be modified. This results in many applications which all need to integrate with each other for the best user experience. For example, when making modifications to flight control procedures, a link to the spacecraft data is established in the respective editors so that the FCT can more easily make changes in accordance with the available telemetry and telecommands of the spacecraft.

The OPEN framework reduces repeated software development work by providing common functionalities (such as version control, viewing and editing any data of EGOS-CC based systems) to any tools based thereon. The OPEN-M Eclipse Rich Client Platform (**RCP**) based application is the main tool used by the FCT members to modify operational artefacts. Through its integration with both the mission repository and change management system, the amount of tool and context switches can be reduced.

Before being used operationally, the changes undergo rigorous testing. This typically happens in a dedicated test environment that mimics the operational environment as closely as possible. Testing the changes in such an environment often requires some sort of manual interaction:

- The environment needs to be set up and configured on a (virtual) machine. Configuration management systems (such as Chef) allow to reproduce the environments in a consistent way.
- The verification and validation tools need to be started and the respective test scenarios loaded.
- The test environment needs to be connected to a test instance of the mission operation system and a spacecraft operational simulator. Both of these systems need to be loaded with the respective spacecraft data, which might have undergone changes, as well as be put into a specific test state.
- Finally, the tests are executed.

While there are some scripts and processes available to overcome certain setup and configuration steps (and are run with the help of platforms such as Jenkins or GitLab), there is no holistic and end-to-end approach enabling automated testing. In order to achieve continuous integration with testing at its heart, it is important to automate testing and shorten the feedback loop by providing easy access to test reports. Implementing automated tests requires a lot of effort and still, since there are hundreds of procedures for all kinds of contingency scenarios, not all flight control procedures might get tested in an automated manner. Typically, only the most crucial and often used procedures are covered by automated tests while most other procedures are only manually tested. Ultimately, the FCT members need to be confident that their changes do not have any unintended side effects and some manual testing might still be required until test coverage reaches 100 percent.

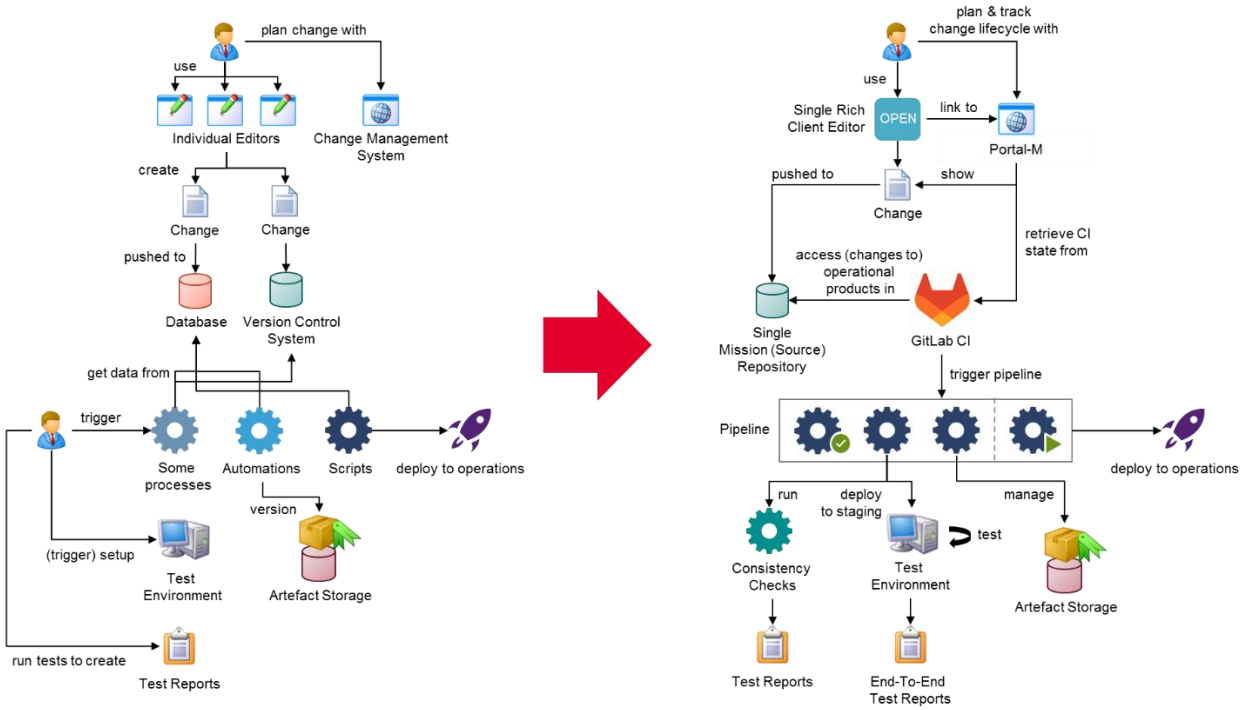

Figure 1. Manual integration (left) to continuous integration (right).

Finally, the changes need to be deployed to operations which entails proper storage and versioning of the operational products as well as deployment to the operational environment. With the help of CD, companies such as Amazon [2] deploy their software updates automatically multiple times a day. Until satisfiable test coverage is achieved, such automated deployment of operational products is unthinkable in mission operations as it may mean the loss of a mission. Hence, a CDe approach can be used where the deployment to operations is still triggered by the Spacecraft Operations Managers (**SOM**) in a reproducible and automated but controlled manner.

In the following sections, the preparation environment and its integrated tools as well as the continuous integration related processes are further detailed.

3. OPEN Preparation Environment for Missions

Applications based on the OPEN framework support the preparation of tailoring data consumed by European Ground Systems Common Core (**EGS-CC**) based systems. As a software framework it provides a foundation of common basic functionality such as version control, data compare and merge, scripting and reporting. OPEN-M supports the flight control teams in defining and managing all EGS-CC tailoring data such as monitoring and control activities, telemetry and telecommand packets, parameters, calibrations, checks, operation procedures, user defined displays and others. Plugins for importing different data formats (such as from the previously used SCOS-2000 based systems) and exporting operational products for eventual usage in EGS-CC based monitoring and control systems are also provided as part of the OPEN-M Eclipse RCP application. Notably, before deploying such products to the runtime environment, consistency checks are automatically executed on the data ensuring that they adhere to the data model, specifications, and other potential mission specific constraints. In addition, the web-based portal for missions, called Portal-M, provides an easily accessible means of managing and visualizing the tailoring data changes.

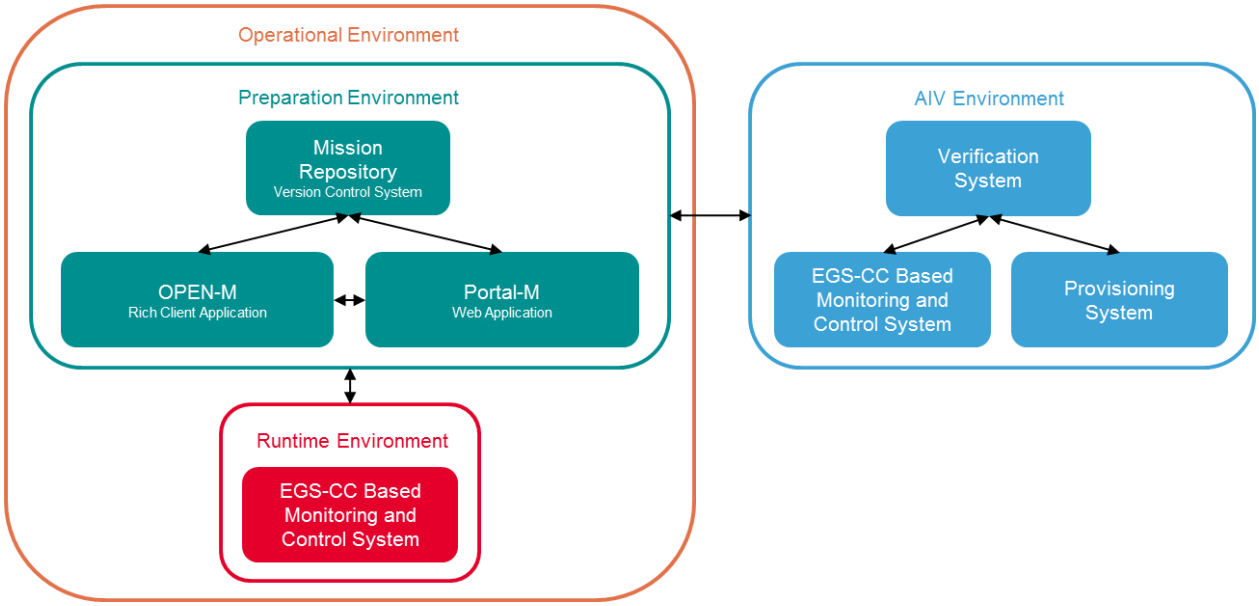

Figure 2. Environments and their relationship. The preparation environment as part of the operational environment interacts with the AIV and runtime environments.

The OPEN-M Rich Client and Portal-M applications are running in the OPEN-M preparation environment, an environment separate from the runtime (containing the monitoring and control system) and AIV environments. Both applications directly interact with the mission repository whereas the other environments rather retrieve the versioned and packaged operational products from a separate binary repository. Thereby it is ensured that the final products that are verified in the AIV environment are also those that are put into operations. The AIV environment is set up in such a way as to verify the operational products in a realistic runtime-like environment but without interfering with it. Contrary to the runtime environment, it contains additional systems for provisioning and verification purposes. Interaction between the environments is done via well-defined interfaces, e.g. to trigger execution of tests for change requests in the AIV environment and to push the results back asynchronously to the preparation environment (see also section 5.1).

4. Integrating Software Systems and Harmonizing Processes

The preparation environment specific software systems interact with a number of external software systems, which are integrated as much as possible without reinventing the wheel. While some software systems such as the source repository and CI/CD ones can be seen as part of the preparation environment, re-using existing external software

systems such as the GitLab DevOps platform for the CI/CD reduces the overhead of maintaining yet another custom system. An end-user is only confronted with the preparation environment systems thus the complexity of the system landscape is hidden. Figure 3 highlights the external systems in the context of the preparation environment and also shows the end-users targeted by the preparation environment.

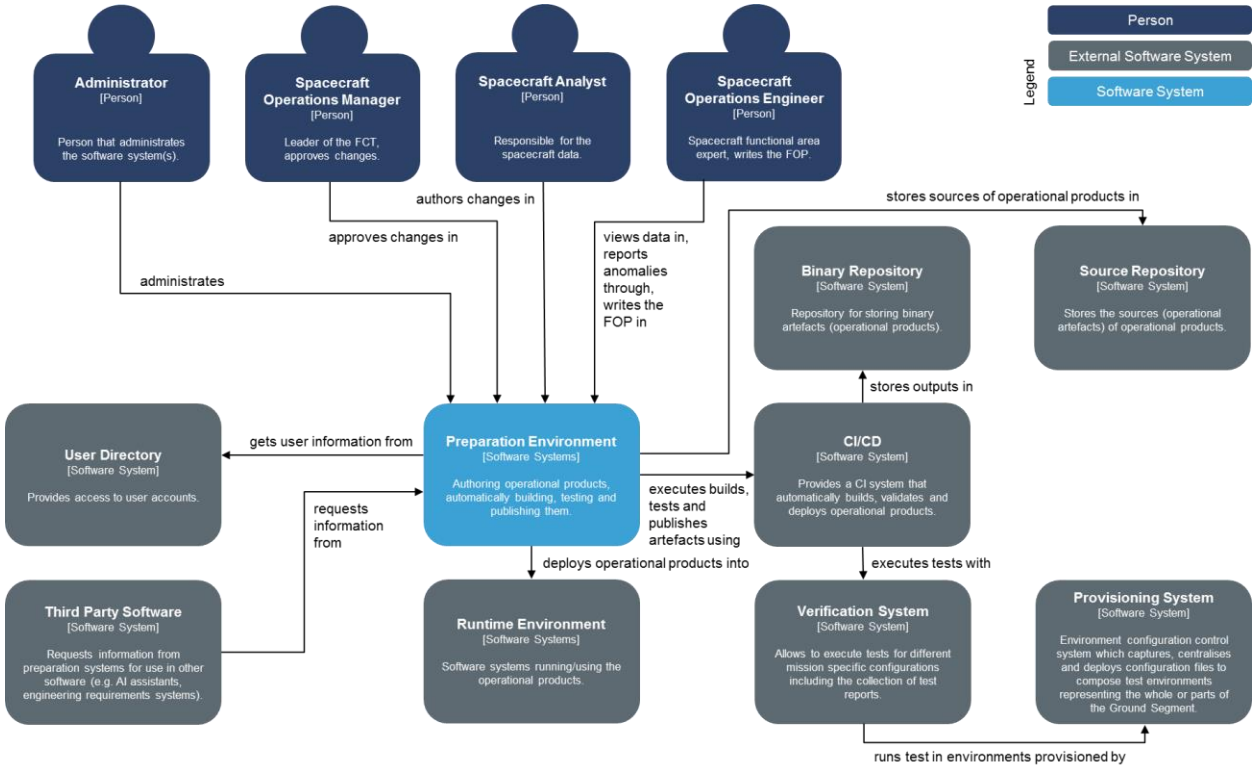

Figure 3. C4 high-level system context diagram [3] for preparation environment systems.

In this section, the preparation environment systems are detailed further including the underlying mission source repositories that allow to version control the operational artefacts. In section 5, the focus is on the CI/CD system as well as the processes for the verification system.

4.1. Mission Repositories

Any data used by missions is version controlled and hence stored in a number of Git repositories. A mission consists of data maintained by different teams and used in different contexts. Any data that is not mission specific is referenced rather than copied to allow re-usability and increase maintainability. Since EGS-CC has been designed to be applicable for different mission phases and types, missions using a monitoring and control system that is based on EGS-CC require customisations (through software extensions for functionality not supported by the EGS-CC software), tailoring (through the so-called tailoring data for the controlled system and any potential additional software components) and configuration to adapt the system to the specific needs. Data for this adaptation process are stored in the source repositories. The following data is kept under version control:

- **data** tailoring data including automated flight control procedures and displays
- **configuration** of OPEN applications, simulators etc.
- **checks** to ensure consistency and correctness of the tailoring data
- **scripts** that allow to automate tailoring data interaction
- **reports** which are generated and need to be retained
- metadata indicating the references (monitoring and control system data, ground station data and common OPEN data, each in their specific applicable version) of the mission

Due to the tailoring principles of EGS-CC outlined above, the data may be produced as part of different development lifecycles which together with the different data types makes it necessary to segregate the data on three levels:

1. Dedicated repositories: Data of different ownership are kept in different repositories. This allows to re-use it more easily and to have clear ownership of the data.
2. Dedicated branches: "Same" data of different versions is kept on different Git repository branches. Thereby it is possible to switch between different patch levels of the same data which in itself is consistent and complete.
3. Dedicated folders: Data of different purpose are kept in different folders. Different data types are kept in different folders of the same repository. This also applies to different purposes of the same data type, e.g. tailoring data of one spacecraft is kept in a separate folder from the tailoring data of another spacecraft even though both are tailoring data (and are otherwise in the same top-level folder). Keeping these data in the same repository has the advantage that only one repository needs to be modified if a change is implemented affecting different data types, e.g. tailoring data and scripts.

Repository segregation is also a consequence of the hierarchy of systems as seen in Figure 4: The ESOC specific adoption of the EGS-CC - called EGOS-CC - adds a layer of software extensions to the common EGS-CC infrastructure. In addition, similar missions (e.g. all earth observation missions) may share the same extensions on top of EGOS-CC which are maintained separately. Since the more specific repositories (e.g. the control system mission family repository) extend the less specific one (e.g. the EGOS-CC control system repository), the mission repository only depends on the most specific control system repository which is declared in the mission metadata as a reference. A mission can also add additional extensions which are under responsibility of that mission and are hence kept in the mission repository directly. With this repository segregation it is possible to re-use extensions more easily across missions.

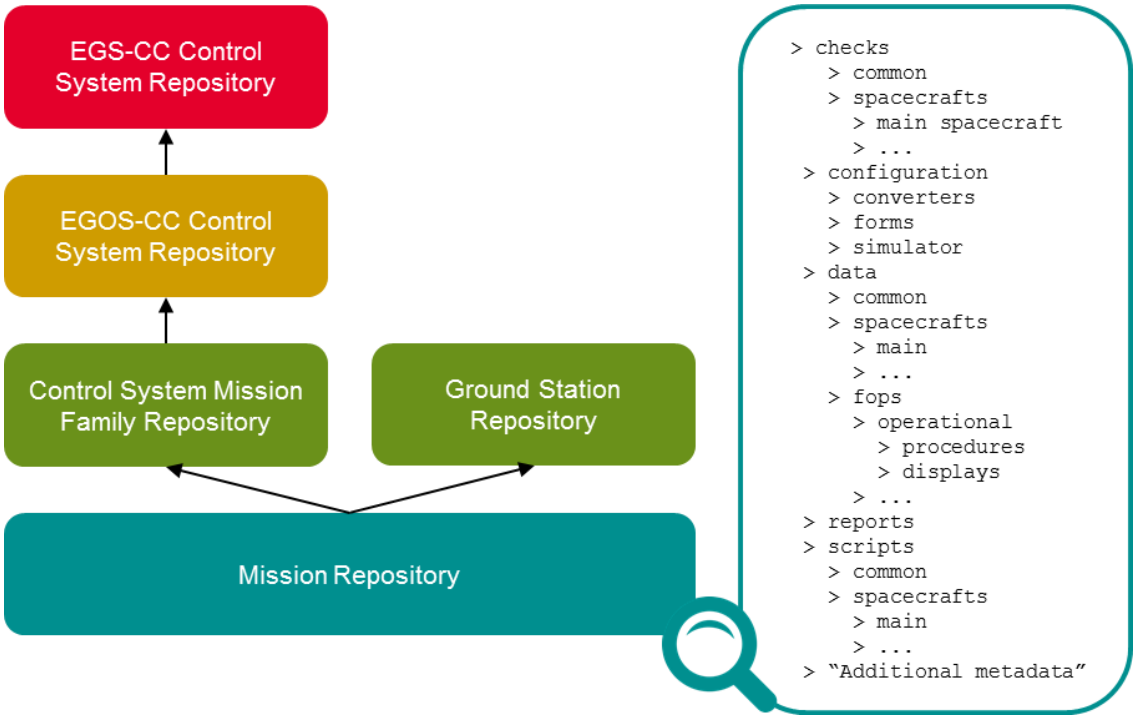

Figure 4. Repository hierarchy and content structure. The mission repository depends on other repositories.

While the mission repository depends on a number of other repositories, it does not depend on a specific CI platform: All infrastructure data to enable continuous integration is stored outside of the mission repository in a separate infrastructure repository. As the current implementation of the CI processes relies on GitLab, this includes for example the GitLab specific definition files (in the form of YAML Ain't Markup Language (**YAML**) files) but also

general scripts for the build processes. The infrastructure repository is then configured as part of the GitLab configuration of each mission project.

4.2. A Portal for Harmonized Change Management

In software development it is common practice to manage changes to the software in a project/change management system such as JIRA. It provides dashboards to get an overview of open feature or change requests and their states according to a configured workflow as well as links the individual tickets to the software changes in their source repositories. As a common platform to provide the same functionality for mission preparation, a web application named Portal-M has been developed at ESOC. It allows to:

- Manage multiple missions with multiple mission repositories
- Manage different types of change requests for mission data
- Get an overview of the change requests and their state according to a standardized workflow
- View the changes in the EGOS-CC tailoring data directly in the browser
- Trace the change requests to the changes on their associated branches in the repository
- View the results of CI execution as part of each change request
- Version the change requests in the context of the changes in the repository

Contrary to other change management platforms, Portal-M has been specifically tailored to the needs and existing workflows of the FCT. In the following, the branching strategy, workflow, and versioning approach is described, all of which eventually lead to the CI processes as described in section 5.

4.2.1. Branching Strategy

A branching strategy or model describes which branches exist in a repository, what their use cases are and how to handle them. The chosen branching strategy must support the workflow. There are different well-known branching strategies available and the presented branching model is generally following a GitHub flow [4] approach, sometimes also referred to as a scaled trunk-based development [5] approach:

- There is a single trunk (in Git often called `main` or `master`, here called `main`) which only contains validated changes and is always deployable. Since it can be tagged at any point in time for operational usage, a version number shall not be defined in any of the files on the branch as it would require additional release preparation (such as "massaging" of files).
- Features (e.g. data or procedure changes) are developed on feature branches branched off from `main` and merged back into `main` with the help of so-called merge requests. These merge requests undergo review and validation before being merged into the `main` branch for everyone else. Feature branches need to be kept up-to-date with the `main` branch meaning that the feature branch needs to contain the latest changes from the `main` branch - if changes have been merged into `main` in the meantime - by merging the `main` branch into the feature branch. As changes of a feature branch are often contained within a single subsystem, conflicts between different feature branches are not likely and, in any case, need to be avoided by the sole implementer of that spacecraft subsystem. This is especially true to avoid repetitive synchronization (merge from `main`) situations that might require a re-validation and can also be avoided when the changes are small enough to allow for a short lifecycle of the feature branch. With this verification-first approach, the necessity of hotfixes should be prevented but if needed can be implemented in the same way as any other change though potentially prioritised.
- The continuous `ops` branch is never changed directly but always contains the last released version. This is a convenience branch in addition of having tags for releases on the `main` branch which allows other tooling to easily refer to the latest release without going through the list of tags.

There might be additional feature branches for internal usage (e.g. holding the last delivered data from industry) but their merging into `main` always follows the feature approach outlined above. Figure 5 shows the branching model.

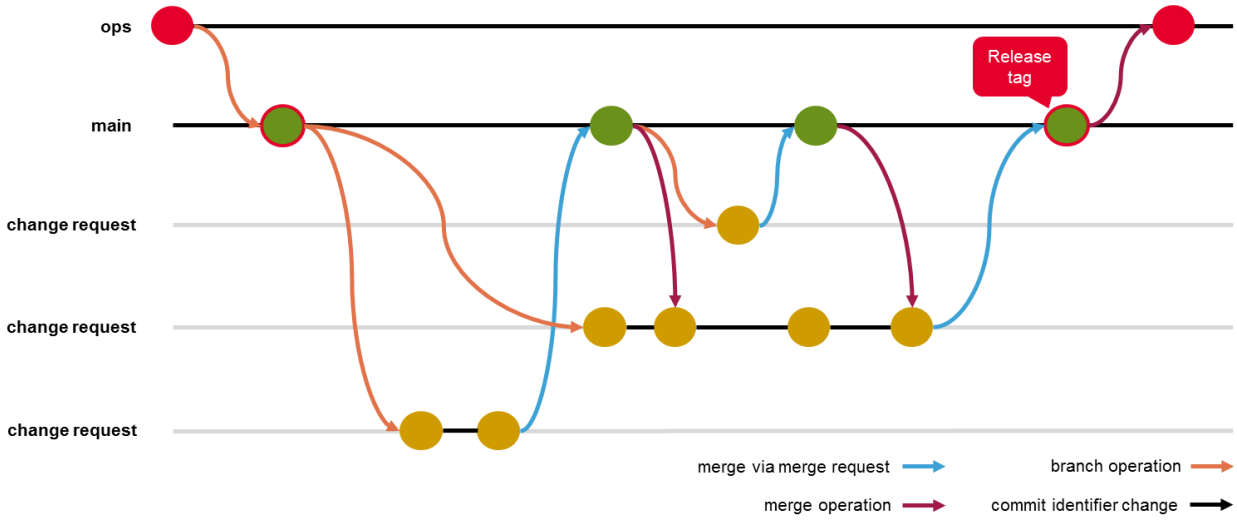

Figure 5. Branching strategy using a single `main` branch and feature branches for development. The `ops` branch contains the last released version.

This branching strategy was chosen for the following reasons:

- It is simple enough yet fits with the workflow of the FCT.
- It allows to restrict what goes into the `main` branch, specifically it ensures that only validated changes are ever brought into production.
- CI is central to this solution rather than secondary.

4.2.2. Workflow

A workflow defines how the change request is handled by the FCT members. Different members are responsible for certain steps in the processing of the change request, which is depicted through states in the workflow. The change request transitions from state to state until a final state is reached. The following actors for the workflow are relevant:

- The Spacecraft Analyst (**SA**) is responsible for the spacecraft tailoring data and configuration specific to a mission.
- The Spacecraft Operation Engineer (**SOE**) is an expert of the spacecraft's functional area and writes amongst others the FOP.
- The Spacecraft Operations Manager (**SOM**) leads the FCT and is responsible for the definition, implementation, and execution of the entire mission related activities.

In addition to these actors, Portal-M respectively the CI plays a central role as it automates certain processes and related state transitions. The overall workflow is depicted in Figure 6.

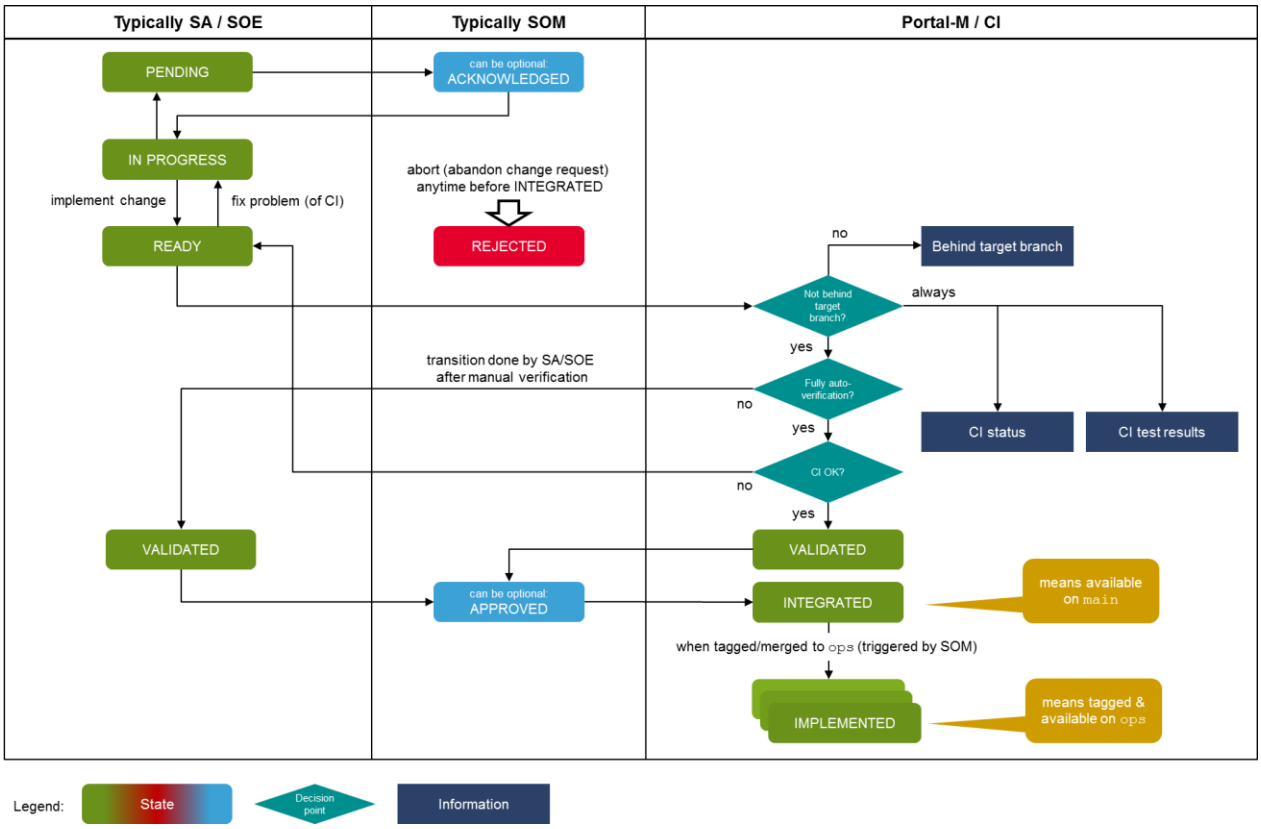

Figure 6. State transition workflow of Portal-M. The diagram is structured into three columns, depicting the drivers of the state transitions.

The workflow considers both the current and future automation maturity:

- At present, not all changes can be fully verified by automated tests, either because the test is not yet available or it is not feasible to automate the test.
- In the future, all changes shall be covered by automated tests such that no further manual testing would be necessary.

It also caters for different mission phases: During production, change requests are acknowledged and approved before work can start or before changes go into production. In non-production mode, approvals are relaxed leading to faster turn-around times. In general, the FCT members retain control over the critical state transitions that relate to their work responsibilities while those state transitions that relate to the CI are automated. As this solution only implements continuous delivery but not deployment, the decision to deploy changes to production (i.e. transition related change requests to the `IMPLEMENTED` state) still lies with and is triggered by the SOM. The actual processes behind it are automated as well within Portal-M.

4.2.3. Versioning

The changes shall be traceable through the change requests and the operational products in operation. To achieve this traceability, the operational products are versioned both as a versioned product in the binary repository (see section 5.2) and in their "source" form as changes to operational artefacts in the source repository. After a change request has been validated and integrated into the `main` branch, it will be part of the next release of the operational product. The `main` branch is tagged with the version number once the SOM decides to release a new version of the operational product and all its integrated changes through Portal-M. The version number is a simple increasing integer as a differentiation of fixes or breaking changes (as in semantic versioning) is currently not planned.

4.3. Integrating OPEN-M with Change Management

The OPEN-M Rich Client application is central for browsing and editing the data stored in the mission repositories. Changes pushed into the repositories shall be done in the context of the planned change requests in order to trace individual changes back to their original feature request or bug report. As part of this work, an add-on was developed that allows to connect to the Portal-M change management system as seen in Figure 7. Other change management systems are potentially also supported if they implement the supported interfaces. This add-on also links with the task management system of OPEN-M which allows end users to create, commit and push branches (called tasks) to the repository, hiding the complexity of all available Git commands and focusing on the workflows used by the FCT. Therefore, the processes can be summarised as follows:

1. The user can see all change requests of Portal-M in an overview table.
2. When the change request is confirmed for implementation, the FCT member creates a task in OPEN-M from that change request. This automatically creates the correct branch and transitions the change request to the `IN PROGRESS` state in Portal-M.
3. In the context of the task or change request, the user pushes the changes to the repository.

There is no longer a need to switch to any other (web) application as all the workflow related actions such as branch creation or change requests editions can be executed directly from the OPEN-M application. This supports the FCT members in applying the workflow and thereby supporting the standardisation of such workflows across missions.

Another principle of continuous integration is to fail fast so that problems can be found early and fixed quickly. OPEN-M also supports the FCT members in this regard by allowing to run the same consistency checks as run by the CI in the local instance. These checks are the first line of defence against faulty changes made to the tailoring data and therefore can be run even before pushing changes. More extensive and longer duration testing is then triggered by the CI system.

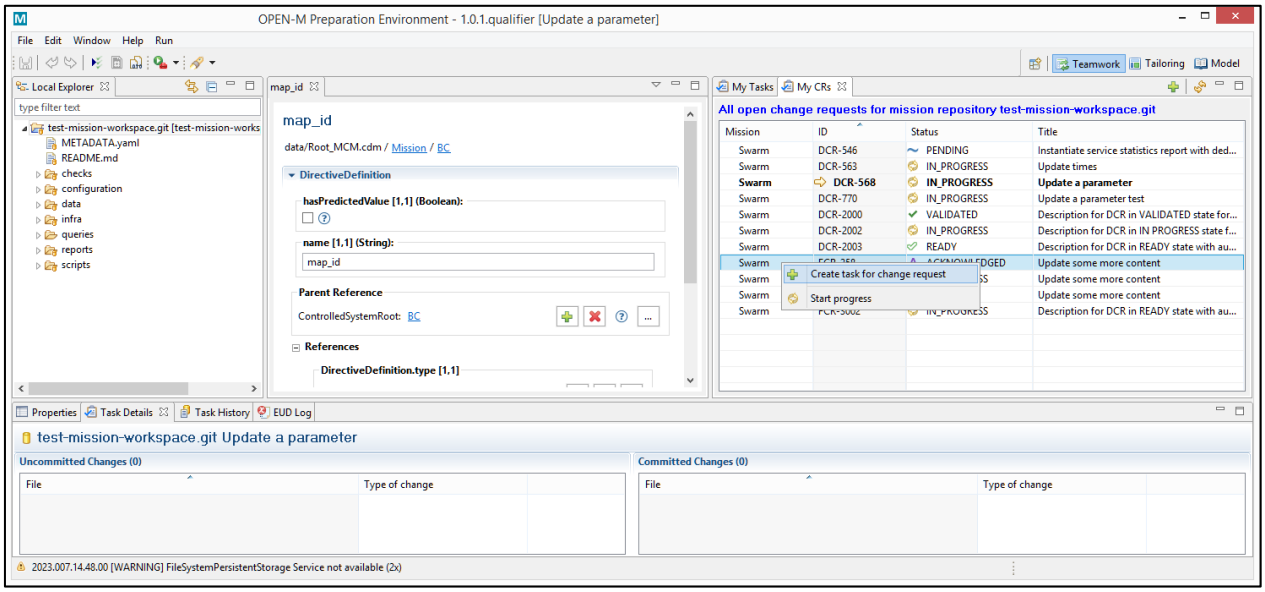

Figure 7. OPEN-M with change requests overview on the right.

5. Applying Continuous Integration

In software development, a system can be set up so that each feature and bug-fix is continuously integrated and continuously deployed. Usually, it follows a cycle depicted in Figure 8 where the steps "build", "test", "release", and "deploy" are automated. The release process describes the generation of the artefact containing the just built and tested increment of the software. The deploy-step is putting the generated release into operation. Since this cycle is performed for each single modification, the system provides rapid feedback to the implementer. A prerequisite is a high coverage with well-written tests to ensure the software's stability and security.

Such a CI/CD system requires several components including but not limited to the source repository, the pipeline executors, and the binary repository for the deployment of the successful build. A large variety of off-the-shelf systems is available on the market providing a wide range of functionalities and aiming at various scenarios. For the continuous integration of operational products, GitLab has been selected. It integrates the source repository, the pipeline executors, and the binary repository. This reduces maintenance and manual integration effort and allows to setup, inspect, and monitor the components from the same user interface. Additionally, GitLab is already used at ESOC for software projects. This omits the need of another software system and the experience of administrating GitLab is already established.

The CI/CD for operational products uses an adapted cycle, see Figure 9. The step "test" has been split into a consistency check step and two verification steps. The "consistency check" step performs checks on the data and its dependencies of the source repository; the verification step executes in-depth simulations in a representative environment (AIV environment). This step requires a proper build and versioned operational product within the binary repository. Therefore, the step "generate operational product" has been added. For cases where automated tests are not present or their execution is not cost-effective[1] and hence does not justify a heavy test in the AIV environment, a manual verification is performed.

Like in software development, also for operational products the test reports created during the CI are made available to the users for inspection. The users need a quick and easy-to-understand visualization of the report as well as a fast way to filter the reported items. This is crucial for a streamlined workflow because reports often incorporate several ten-thousand items. While certain items may be false-positives e.g. due to broken test scripts, others are critical and have to be identified. Therefore, Portal-M visualizes the consistency check results in combination with the change request's details. In order to make the data available within Portal-M, the CI authenticates with the portal after the consistency checks and after the verification tests were performed to transfer the data via Hypertext Transfer Protocol Secure (**HTTPS**) to a Representational State Transfer (**REST**) endpoint.

For operational products, the CI/CD cycle and its transitions are not automated to their maximum. Especially the release process has manual interaction points. It is desired that a planned collection of modifications is released in a controlled manner to ensure the mission's safety.

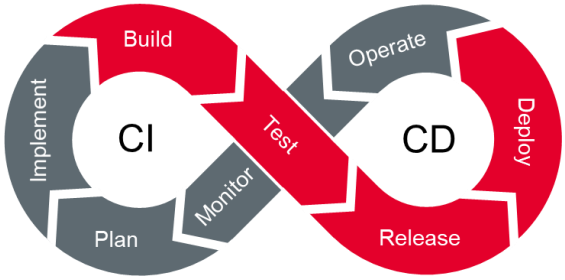

Figure 8. In software development, a CI/CD cycle usually contains the depicted steps. The steps coloured in red are executed in an automated manner. Since this cycle is executed for each single modification, a rapid feedback loop is created: The system detects problems early before they are deployed.

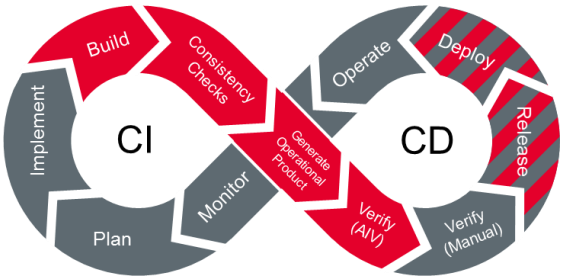

Figure 9. For operational products, the CI/CD has been adapted and additional manual steps are introduced. In contrast to software development, here a controlled deployment is always needed but requiring for verification tests a built and published operational product. The hatched steps are automated but have to be triggered manually to leave control about new data increments to the FCT.

[1] For instance, if the modification is small and the implementer is confident of their modification's impact.

5.1. Continuous Integration and Verification

Continuous integration and verification are applied in accordance with the workflow and branching model. When the implementation of a change is deemed ready on a feature branch, a CI pipeline is triggered which performs two levels of testing:

First, the consistency checks can be seen as unit level testing and typically execute fast. They validate that the tailoring data adheres to the data model and any specific constraints imposed by the mission. These checks are implemented as groovy scripts which can be run within OPEN-M Rich Client or through a Command Line Interface (**CLI**) from the CI. The scripts use an interface to traverse the data and its data model which is based on the Eclipse Modeling Framework (**EMF**). Such a check can be, for instance, that the Application Process Identifiers (**APID**) are unique.

Secondly, comprehensive verification in a representative environment is conducted. While consistency checks are inherently automated, this is not necessarily the case for verification testing as it requires a lot of infrastructure and automation. Due to this fact, a change request can be marked as either being covered by full auto-verification or not. In the first case, the CI would deploy the necessary (software) infrastructure, install the operational product under test therein and run end-to-end tests to verify the changes on the branch from where the operational product originates. The test results are then pushed to Portal-M and displayed alongside the overall status of the CI execution. As part of the CI automation, the associated change request is automatically transitioned to the `VALIDATED` state (see also Figure 6) and the changes integrated into the `main` branch. In case there are not enough tests available to fully verify the changes (this is to support the existing incomplete test coverage), the CI still deploys the infrastructure and installs the operational product, but leaves the test execution to the FCT member working on the change request. This approach already eases the work of the FCT because the setup of the test environment often involves a lot of work. The deployment and installation are done via the (external) provisioning system, typically using containerization techniques.

End-to-end verification testing is conducted via the (external) verification system. This activity has defined the testing approach and the necessary interfaces to integrate with the preparation environment tooling. Instead of asking the verification system to run a specific test, the concept is to request to verify a certain change and the verification system derives the tests to execute. For this, three specifications are needed:

1. Specification of the change which defines what is changed given a specific type of change (e.g. to test a parameter check type of change, the parameter name and out of limit values need to be given).
2. Specification for the test which gives information (in addition to those used from the change specification) what shall be verified in addition by the test e.g. log messages, telemetry retrieved etc.
3. Specification of the mission (i.e. its mission name and specific version reference) in order to identify the exact operational products to be tested. All dependencies are retrieved by the provisioning system based on the mission specification.

The change and test specifications are defined by the FCT in Portal-M when creating the change request respectively when implementing the change. An interface to the verification system allows Portal-M to validate the specifications against their schemas and support the user in defining them. A REST Application Programming Interface (**API**) has been created for the interfaces as well as example YAML specifications and schemas. While the definition of such schemas to cover all different kinds of changes is outside the scope of this work, the usage of YAML allows to easily exchange and create specifications by end-users, these being either the FCT members or test engineers.

The specifications are input to the verification system which

- Maps the change/test specification to the test case (i.e. the verification system knows which test case(s) verify a certain type of change)
- Requests a test environment from the provisioning system given the mission specification
- Runs the identified test case in the test environment
- Collects the test results

As the test execution triggered by the CI may take an extended amount of time, the test execution may be asynchronous. Another interface in Portal-M allows the verification system to push the results back when finished. Subsequently, the test results are shown in Portal-M's change request.

Once the feature has been merged into the `main` branch (either automatically or after manual verification), the CI pipeline for the `main` branch is executed. The processes are slightly different for the `main` (and `ops`) branch and are further described in the next section. In particular, a release process can be triggered at any time because the `main` branch only contains verified changes and does not need an additional verification.

5.2. Continuous Packaging, Release and Deployment

The process from packaging the operational product to releasing and deploying it to the operational environment embraces multiple steps. Those ensure the data integrity and avoid repetitive cost-intensive consistency checks and verification execution. Another goal is to keep the operational product file self-contained to avoid the need to load other operational products as dependencies.

5.2.1. Packaging

Generating self-contained operational product files requires that all dependencies are known at packaging time. As mentioned in section 4.1, the mission repository contains a list of all dependencies in its metadata file. This file is parsed during the generation process and all dependencies and transitive dependencies are downloaded. The actual packaging process is performed with a dedicated packaging application that is called within the CI via a command line interface. This application packages the tailoring data from an OPEN based environment into a file format that is meant to be used by external systems, specifically for EGS-CC bases monitoring and control systems. It contains alongside the tailoring data further metadata like checksums to ensure its integrity and consistency.

Immediately after the operational product file has been generated, the next step is checking the product file for integrity. This check is sensitive to the branch (compare section 4.2.1) the CI is currently running on as shown in Table 1:

- if this check is executed on a feature branch a simple check is performed. That means it is tested whether the file can be un-packed and the content's integrity is kept.
- if this check is performed on the `main` or `ops` branch, an extended check is executed (which includes the simple check). Since on those branches no consistency checks and no verification tests (neither automated nor manual) are executed within the CI pipeline (compare Table 1), the system still needs to ensure that the generated operational product file contains actual well-tested data.

As described in section 4.2.1, the `main` branch is always in a well-tested state because everything that is merged into that branch is tested properly on its feature branch beforehand. Consequently, after a feature branch is merged, the pipeline that works on the `main` branch can skip all consistency checks and verification tests. To ensure that the now on the `main` branch generated operation product file contains the same data as the product that has been tested on the feature branch, this step compares the content in the product files between the `main` branch and the latest merged feature branch.

Table 1. Detailed excerpt from the CI/CD for operational products. The steps are not executed on all branches defined in the branching model. Testing is only performed on feature branches to ensure an always-validated `main` branch. On the other hand, the product file validation is more in-depth on the `main` and `ops` branch. A deployment to the operational environment is only performed if the CI is triggered on the `ops` branch.

	Feature branch	main-branch	ops-branch
Consistency checks	✓		
Operational product generation	✓	✓	✓
Product file check	✓ (simple)	✓ (extended)	✓ (extended)
Verification tests	✓		
Deployment to operation			✓

The pipeline works on the `ops` branch if a release is created and the `main` branch is merged into the `ops` branch. In this case the system executes the same extended check as on the `main` branch except that it compares the product files between the `ops` and the `main` branch. This approach ensures the well-tested state of the data and avoids running redundant tests. Only if this check succeeds, does the pipeline continues with the next step.

Each generated operational product file follows a specific naming convention that traces the file back to its origin, naming the Git branch, Git tag, and Git short hash. This is important since all successfully generated product files are published within the GitLab binary repository, which is called package registry. As mentioned above, this is already required at this step to allow the verification step to work on a properly packaged product file. The package registry can be configured to work with a range of supported package managers. In this case, the generic package types is used to allow working with a custom file name convention.

After the file has been published in the package registry, it is already ready to be pulled by users. The first user is usually the AIV environment which is often but not always provisioned automatically by the pipeline if it runs on a feature branch (see section 5.1 and Table 1).

5.2.2. Release and Deployment

In order to give the FCT the main control about the timing of a new data increment, releases, and deployments are not automatically triggered. Instead, the FCT can actively manage the creation of a new release in the Portal-M user interface. A new release can be created if one or more change requests have been successfully integrated. This means that the corresponding feature branch has been merged into the `main` branch. Usually, the SOM triggers the release process. In the user interface, they can inspect which changes are integrated and are hence ready to be released. If a release is triggered, the following actions are performed automatically:

1. Creation of a corresponding tag in the mission's Git repository (on the `main` branch) to identify the release;
2. Merge of the `main` branch into the `ops` branch so that the `ops` branch is always in the state of the latest release;
3. Pipeline execution on the `ops` branch.

In the following, the third action is addressed in more detail.

A pipeline execution of the `ops` branch is similar to an execution on the `main` branch. Both branches contain the same data after the merge. As described above for the `main` branch, on the `ops` branch also no consistency checks and no verification tests are executed (see details in section 5.2). The main difference between the pipeline that works on the `main` and on the `ops` branch is that on the `ops` branch the deployment to the operational environment is triggered (compare Table 1). In fact, the operational environment is informed about the new release and its new operational product file in the binary repository via an HTTPS request to a well-defined REST endpoint. The operational environment is now informed about the new increment and can start the deployment by pulling the new product file. When exactly the new operational product is deployed is outside of the control of the CI/CD system and is left to the implementation and process that is applicable to the operational environment.

6. Conclusions

With the introduction of technologies and methodologies borrowed from software development, it was possible to create an integrated and more automated workflow together with the FCT. The whole workflow has been streamlined and points have been identified that now have been automated. The critical parts were to automate as much as possible but still leave the control in the hands of the FCT. Since it is ESA's first system that allows to automate the whole test, release, and deploy phases of operational products, it was important to be able to allow enough manual intervention to

- ensure high user acceptance;
- mitigate non-automated cases;
- provide the means for a transition phase to a fully automated process.

It has been shown that tools used successfully in software development can also be used for preparing and validating operational products. This not only reduces costs and the administrative effort but also speeds up the development of such systems. Smart short cuts have been taken to further reduce costs during the test phase by avoiding repetitive tests and simulations.

A key aim was to provide one integrated and consistent graphical interface to the users to inspect and control the workflow as well as to inspect the current and future states of the operational products. In addition, the editor of operational products has been integrated with Portal-M to avoid the need of unnecessary manual steps within the preparation environment. This allows the users to focus on the actual implementation of changes while the platform

supports them by guiding them through the process. At the same time, it is ensured to fail fast so that the implementation process is sped up as much as possible.

7. Outlook

Beyond more efficient processes through continuous integration and delivery, continuous deployment promises to put changes frequently into production. While continuous deployment does not look feasible or desired at this point, with more automated tests available the confidence into rapid deployments is expected to increase in the future. For this, additional types of tests or test frameworks could help to comprehensively cover test scenarios that are currently executed manually. ESA has several test frameworks at its disposal, less so for verification and validation of operational products but rather for infrastructure software. Their acceptance and usage are often diminished by the fact that they require manual setup and configuration. The Automated Rule-Based Cross-Validation of Operational Data (**ARVALIS**) framework is a "rule-based system for end-to-end automated testing and validation of operational data" [6]. It could be integrated into the OPEN preparation environment and ARVALIS tests executed as part of continuous integration, thereby reducing manual setup work. Overall, the need for manual steps in existing end-to-end verification solutions used at ESA is hampering their usage in fully automated continuous integration processes. While this paper has presented the interfaces necessary between the preparation and AIV environment, the existing verification systems must be adapted to be usable in an automated way. Since CI/CD has become so popular and common in software development, a number of solutions are available to support automation, such as containerization, configuration management systems or complete DevOps platforms. While the latter is not specifically targeted towards mission preparation, methodologies and processes can be applied to operational products and will increase the efficiency of the FCT members even further in the future.

Acknowledgements

We would like to thank Jaeho Shin for his incredible contributions to Portal-M, Chris Marpe for her constant support maintaining the agile process during the development, Marco Forsinetti for his positive attitude and invaluable feedback from the user's side, and the whole FOPPER team for their constant support.

This work is part of the General Support Technology Programme (**GSTP**) of and funded by the European Space Agency.

References

[1] European Space Agency. About OPEN. *OPEN Preparation Environments*. Available at: https://open.space-codev.org/. (Accessed: 17th January 2023)

[2] McKendrick, J. At Amazon, it's a 'hands-off' approach to continuous integration and continuous deployment of software. *ZDNET* (2020). Available at: https://www.zdnet.com/article/amazon-has-hands-off-approach-to-continuous-integration-and-continuous-delivery-of-software/. (Accessed: 28th December 2022)

[3] Brown, S. Introduction. *The C4 model for visualising software architecture*. Available at: https://c4model.com/. (Accessed: 17th January 2023)

[4] GitHub. GitHub flow. *GitHub Docs*. Available at: https://docs.github.com/en/get-started/quickstart/github-flow. (Accessed: 17th January 2023)

[5] Hammant, P. Trunk based development. *Introduction*. Available at: https://trunkbaseddevelopment.com/. (Accessed: 17th January 2023)

[6] Lucia, D. et al. Towards automated end-to-end testing and validation of operational data. *SpaceOps 2016 Conference* (2016). Available at: https://arc.aiaa.org/doi/pdf/10.2514/6.2016-2390. (Accessed: 17th January 2023)