OpenStudyBuilder Hub (OSB-Hub)

OSB-Trail-SystemEngineering Action Planning

Innovating Through Community Collaboration
June 2nd 2025

Pascal Bouquet

Agenda

- ➢ Introduction
- ➢ USDM Importer in OSB
- ➢ OSB Testing Strategies

OSB Hub

Mission and Vision:

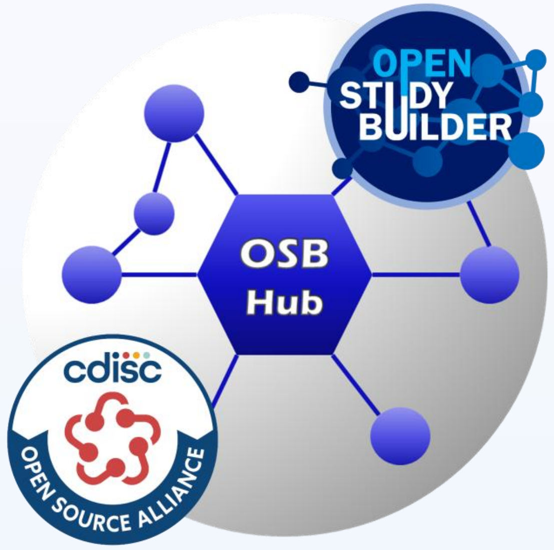
Support the utilization and enhancement of the OpenStudyBuilder open-source tool.

Core Objectives:

Collect feedback, run focused projects, drive innovation through community engagement.

Participants:

Everybody is invited!



Purpose of OSB Hub

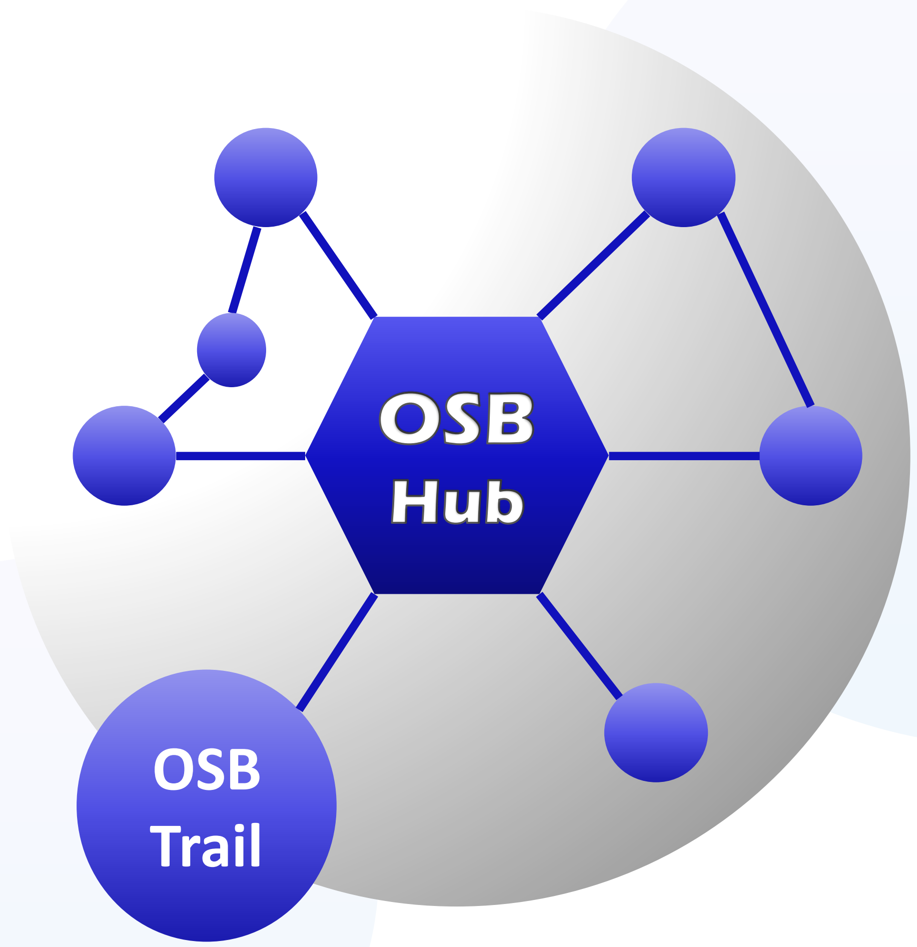
- COSA Community
    - List, add and discuss use-cases
    - Feedback & community interest
    - Prioritise use-cases of interest
    - Manage and run focus projects
    - Utilization and enhancement of the OpenStudyBuilder

Join us on Slack: [Invite]

Feedback on Use-Cases: [Discussions]

Checkout Information: [Wiki]

OSB Trails – Focus Projects



OSB-Trail-System Engineering: **SCOPE**

Priority1 Scope

1. Optimizing deployment workflows for OSB across diverse environments. First deployment and deployment of new release
   - ✓ Cloud: Azure, AWS, Google
   - ✓ Containerization of OSB: EKS, AKS, GKE, OpenShift, Vmware Tanzu, Fargate, ACI …
   - ✓ Develop a Terraform-based deployment template for easy replication across cloud environments (AWS, Azure, GCP).
2. Exploring and implementing robust authentication methods tailored to OSB.
3. Facilitating seamless integrations between OSB and other enterprise systems.
4. Monitoring of OSB/Observability
5. Inventory of OSB implementations
6. Archiving, Back-up, DR
7. Share Performances/Scalability Testing
8. Validation approach of OSB and Interest in developing UI testing tools to support validation.
9. Applying GraphRag on OSB
10. Changing OSB CSS

OSB Strategic Roadmap and Vision

OSB is advancing towards full end-to-end automation in clinical trials by integrating structured metadata across various processes:

Protocol Authoring

Structured elements like the SoA are used to automatically populate Word-based protocol templates.

Data Collection Enablement

Facilitates the setup of Electronic Data Capture (EDC) systems through standardized metadata.

Data Transformation and Analysis

Supports the generation of SDTM datasets and statistical analyses (ADaM/TFLs) by leveraging consistent metadata.

Regulatory Submissions

Enhances the preparation of clinical study reports and submissions by maintaining data integrity and traceability.

These efforts align with the CDISC 360i initiative, aiming to harmonize data standards and improve tool integrations throughout the clinical trial lifecycle..

Some OSB News

Migration of the OpenStudyBuilder project from GitLab to GitHub completed: Github code repo

Version 0,15 of OSB:

Release of Word Add-In	The OSB Word Add-In introduces a new ribbon in Word, allowing users to populate structured protocol content from OpenStudyBuilder into a standardized Word template
Copy Study Structure	Users can now duplicate existing study structures to create new studies, promoting efficiency and consistency across related studies.
Enhanced SoA Interface	The SoA interface now supports drag-and-drop functionality and is optimized for smaller screens, improving usability.
Wizard for Activity Instances	A guided setup (wizard) for creating Numerical Findings Activity Instances simplifies the configuration process.
Audit Trail Enhancements	A new dashboard allows users to compare different versions of a study, enhancing traceability and version control.
CRF Visualization Options	Users can specify vendor extensions when generating Case Report Forms (CRFs), allowing for tailored representations based on study requirements.

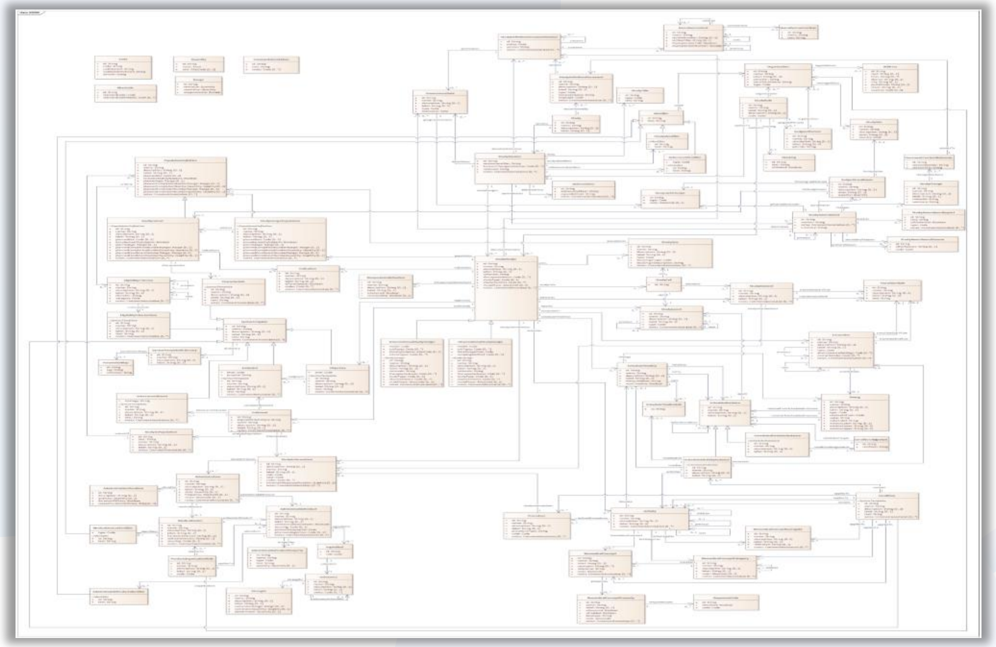
Agenda

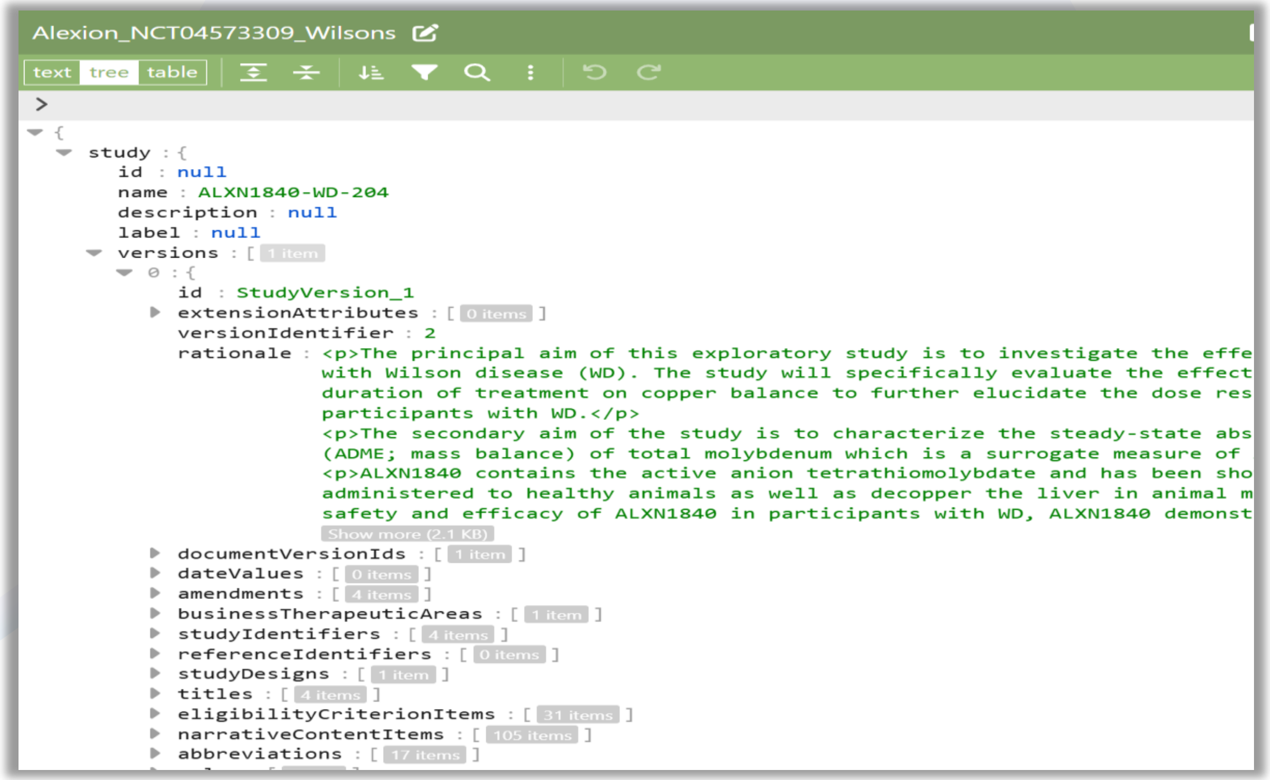
- ➢ Introduction
- ➢ USDM Importer in OSB
- ➢ OSB Testing Strategies

USDM for Dummies

- ✓ USDM aims to establish a standardized, **machine-readable model for representing the definition of a clinical study.**
- ✓ It ensures **interoperability and consistency across clinical trial tools and processes**, particularly in support of initiatives like TransCelerate's Digital Data Flow (DDF).
- ✓ USDM facilitates reuse, supports regulatory compliance, and enhances collaboration between sponsors, CROs, and tech vendors.
- ✓ The USDM Json model includes **core study components** such as **study design, eligibility criteria, objectives, endpoints, arms, visits, activities, and data collection events**.
- ✓ **USDM is owned and maintained by CDISC,** developed in collaboration with stakeholders from the TransCelerate DDF initiative, technology vendors, and regulators.
- ✓ **Current release is USDM v3.0**, published in **November 2023**, which aligns with ICH M11 specifications.
- ✓ CDISC conducted a public review of the draft USDM v4.0, which closed on April 3, 2025.

USDM 3.0





Alexion_NCT04573309_Wilsons

```
>
{
  study : {
    id : null
    name : ALXN1840-WD-204
    description : null
    label : null
    versions : [ 1 item
      0 : {
        id : StudyVersion_1
        extensionAttributes : [ 0 items ]
        versionIdentifier : 2
        rationale : <p>The principal aim of this exploratory study is to investigate the effe
                    with Wilson disease (WD). The study will specifically evaluate the effect
                    duration of treatment on copper balance to further elucidate the dose res
                    participants with WD.</p>
                    <p>The secondary aim of the study is to characterize the steady-state abs
                    (ADME; mass balance) of total molybdenum which is a surrogate measure of
                    <p>ALXN1840 contains the active anion tetrathiomolybdate and has been sho
                    administered to healthy animals as well as decopper the liver in animal m
                    safety and efficacy of ALXN1840 in participants with WD, ALXN1840 demonst
                    Show more (2.1 KB)
        documentVersionIds : [ 1 item ]
        dateValues : [ 0 items ]
        amendments : [ 4 items ]
        businessTherapeuticAreas : [ 1 item ]
        studyIdentifiers : [ 4 items ]
        referenceIdentifiers : [ 0 items ]
        studyDesigns : [ 1 item ]
        titles : [ 4 items ]
        eligibilityCriterionItems : [ 31 items ]
        narrativeContentItems : [ 105 items ]
        abbreviations : [ 17 items ]
```

[1] DDF-RA/Deliverables/UML/USDM_UML.png at main · cdisc-org/DDF-RA · GitHub
[2] DDF-RA/Documents/Examples/EliLilly_NCT03421379_Diabetes/EliLilly_NCT03421379_Diabetes.json at main · cdisc-org/DDF-RA · GitHub
[3] DDF-RA/Documents/Examples/Alexion_NCT04573309_Wilsons/Alexion_NCT04573309_Wilsons.json at main · cdisc-org/DDF-RA · GitHub

Value of a USDM Importer

Completes the Bidirectional Workflow

OSB already supports **USDM export**—an importer would close the loop, enabling **bi-directional integration**.

Accelerates AI-Driven Study Setup

AI tools can generate USDM from protocols and extract **controlled terms** (e.g., **SNOMED CT**), making import into OSB seamless.

Supports Legacy Study Initialization

Importing historical USDM files helps **train users** and pre-populate OSB with legacy studies for demo and QA environments.

Fast-Tracks Migration from Legacy SDRs

Facilitates **automated transition** from older study definition repositories into the OSB environment.

Improves Efficiency and Data Quality

USDM's machine-readble format **reduces manual data entry, minimizes errors,** and **accelerates timelines** across systems.

USDM-OSB: Our experimentation

01

STUDY CREATION

This step creates the initial study shell under the project.

Steps:

1. The script builds a study payload and posts to `/studies`.
2. A study UID is returned for further processing.

02

STUDY PATCH

After creating the study shell, the script updates it with detailed metadata.

Payload structure: Includes registry identifiers, version metadata, design details, and high-level metadata.

Steps:

1. Builds the metadata patch payload by mapping from the USDM JSON data.
2. Sends a PATCH request to `/studies/{study_uid}`.

03

POSTS STUDY DESIGN

- Series of post calls to add /study-objectives[primary,secondary], /study-endpoints[primary,secondary],/study-criteria [inclusion,exclusion]
- OBJECTIVE,ENDPOINTS,CRITERIA-First, post request to create template-return template uid, use template uid at study level to post respective study component
- Adds study elements, epochs, and visits
- Visits require Epoch uid, so script posts epochs gets uid and loops through each epoch to post visits before moving to the next

USDM-OSB: Achievements and challenges

04

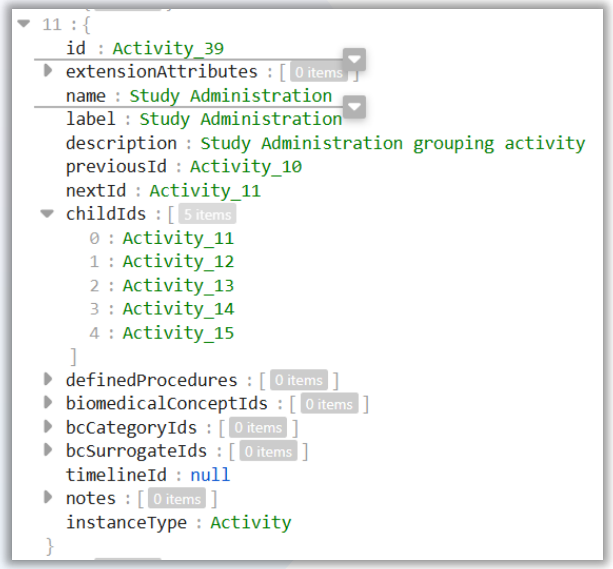
Study Activities

[Post in Library]

This step checks if activity in USDM is in this get request from front end /concepts/activities/activities

Steps:

1. If activity is present in returned json items , returns the activity groupings associated I,e activity_group_uid, activity_subgroup_uid and activity uid.
2. If activity has BC , checks it under the activity returned from /concepts/activities/activities and returns uid as in step [1]
3. If the activity in USDM has a grouping i.e "see snip"
   - Then it loops through each childIDs items, searches front end for the activities, and returns their activity groupings,
   - Else it creates a new activity group, and sub group and creates new activity under this subgroup, for now each activity is created with a suffix _"<study_number>" for traceability or for testing
4. Else: If any activity is totally not in front end, script creates a new group and sub group for the activity under name "TBD" approves them in library then gets the associated activity_group_uid, activity_subgroup_uid and activity uid.



```
11 : {
  id : Activity_39
  extensionAttributes : [ 0 items ]
  name : Study Administration
  label : Study Administration
  description : Study Administration grouping activity
  previousId : Activity_10
  nextId : Activity_11
  childIds : [ 5 items
    0 : Activity_11
    1 : Activity_12
    2 : Activity_13
    3 : Activity_14
    4 : Activity_15
  ]
  definedProcedures : [ 0 items ]
  biomedicalConceptIds : [ 0 items ]
  bcCategoryIds : [ 0 items ]
  bcSurrogateIds : [ 0 items ]
  timelineId : null
  notes : [ 0 items ]
  instanceType : Activity
}
```

05

Study Activities

[Post in Study Level]

Once we have activity_group_uid, activity_subgroup_uid and activity uid , we get the soa_group_term_uid, & activity_instance_uid and post the study activities

Challenges

- SOA group terms are not in USDM
- Only activities with BC can be assigned instance class dynamically, instance can be determined based on class of domain found in properties section in USDM i.e LBCAT=Findings instance class
- Hence created activities under TBD Group and sub group have no instance class and determining one is not dynamic yet
- UID mapping vs Content De-duplication. Calling "CriteriaTemplate_000981" returns metadata on retired/inactive version "CriteriaTemplate_000951"

Take Away and next steps

01 **Importing Legacy Study in OSB has a lot of Value**

02 **USDM provides a useful framework for structuring study elements and** importing in OSB

03 **USDM Importer can take what is in USDM and load in OSB**

04 **A preliminary step is required to ensure good data quality in USDM:** Data harmonization using multiple sources is essential to overcome the challenges of lack of standards terminology and concept in legacy trials

NEXT STEPS (WORKING BACKWARD)

02 Define an approach to data harmonization before USDM creation

01 Completing USDM import

Agenda

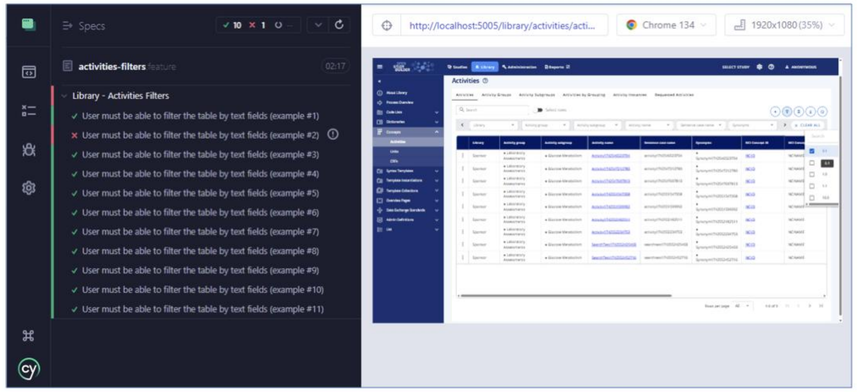
- Introduction
- USDM Importer in OSB
- OSB Testing Strategies

Version 0.14.2 Release (March 2025)

End-to-End Testing Suite: Introduction of a comprehensive testing framework with 93 test modules covering 1,247 test cases ensures system robustness and reliability.

The most impactful addition in OpenStudyBuilder 0.14.2 is the inclusion of comprehensive end-to-end test scripts. This robust testing framework consists of 93 test modules, covering an impressive 1,247 individual test cases.

These tests are built using Cypress, a modern testing framework known for its interactive execution. With Cypress, every step of the test - including clicks, inputs, and validations - is visually displayed, allowing users to see exactly what happens in real-time. This means you can watch tests unfold as they verify expected outcomes, making debugging faster and more intuitive.

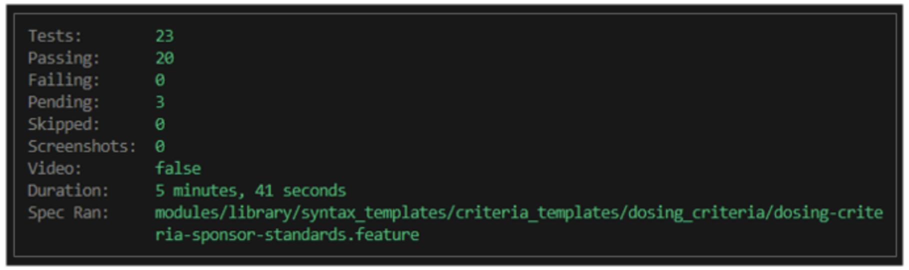


activities-filters.feature 02:17

Library - Activities Filters
- ✓ User must be able to filter the table by text fields (example #1)
- ✗ User must be able to filter the table by text fields (example #2)
- ✓ User must be able to filter the table by text fields (example #3)
- ✓ User must be able to filter the table by text fields (example #4)
- ✓ User must be able to filter the table by text fields (example #5)
- ✓ User must be able to filter the table by text fields (example #6)
- ✓ User must be able to filter the table by text fields (example #7)
- ✓ User must be able to filter the table by text fields (example #8)
- ✓ User must be able to filter the table by text fields (example #9)
- ✓ User must be able to filter the table by text fields (example #10)
- ✓ User must be able to filter the table by text fields (example #11)

Figure 7: Interactive Testing with Cypress using End-to-End Tests

Cypress also supports batch testing, enabling users to execute large test suites automatically in a single run. This drastically reduces manual testing time and ensures thorough validation across the system. Additionally, Cypress tests can be seamlessly integrated into CI/CD pipelines, facilitating continuous testing and deployment.



```
Tests:        23
Passing:      20
Failing:      0
Pending:      3
Skipped:      0
Screenshots:  0
Video:        false
Duration:     5 minutes, 41 seconds
Spec Ran:     modules/library/syntax_templates/criteria_templates/dosing_criteria/dosing-crite
              ria-sponsor-standards.feature
```

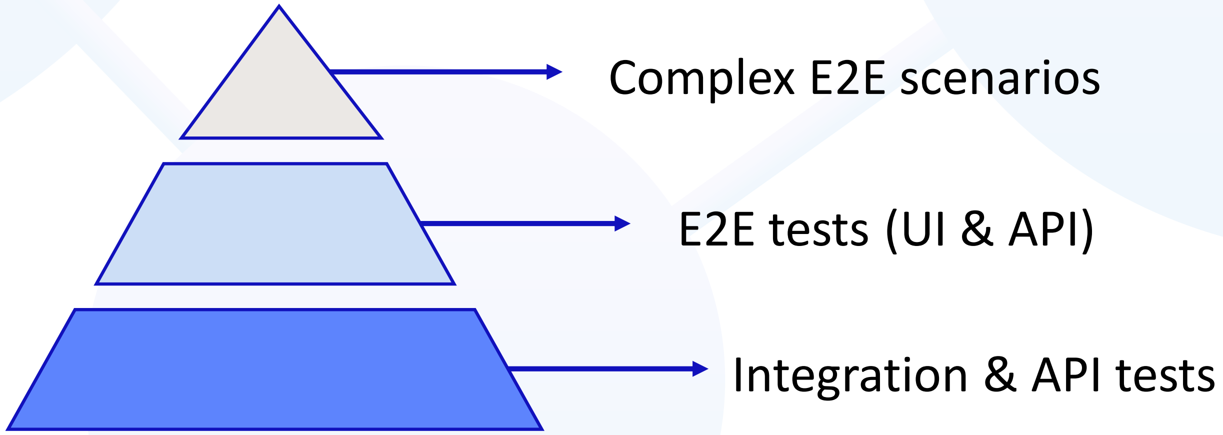
Figure 8: Batch Testing with Cypress using End-to-End Tests

With these new end-to-end test scripts, companies adopting OpenStudyBuilder can now establish a robust automated testing strategy, ensuring system reliability while streamlining study setup. This makes moving to OpenStudyBuilder a smart and future-proof choice for a clinical metadata and study definition repository.

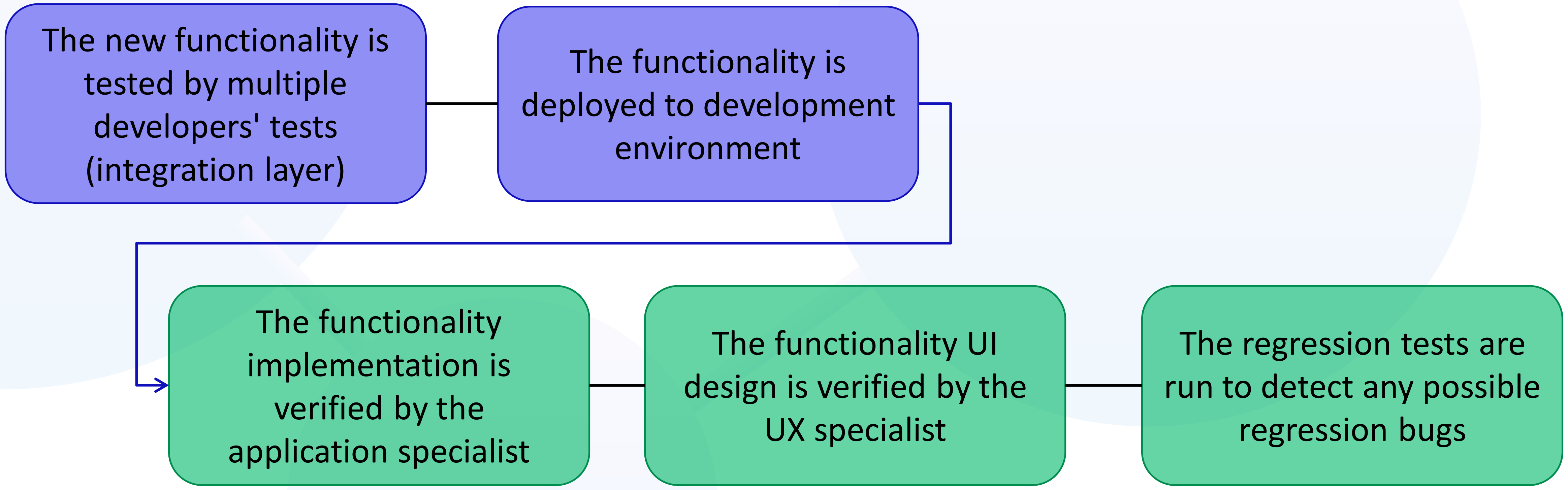
OSB testing process – high level overview

Our testing strategy can be summarized as „Test as early in the process as possible”. This shift-left testing approach allows us to detect most of the bugs right after they’re introduced. Thanks to that the bug-fixing costs are lower.

Moreover, we are following the recommended test pyramid model. Looking at the number of tests we can clearly see that most of them are executed on integration/API level.



OSB testing process – high level overview


The new functionality is tested by multiple developers' tests (integration layer)
The functionality is deployed to development environment
The functionality implementation is verified by the application specialist
The functionality UI design is verified by the UX specialist
The regression tests are run to detect any possible regression bugs

Test Levels – API & Integration tests - overview

These tests are ensuring that the integration between back-end modules is working as expected. They're also verifying the API services behaviour.

- Responsible team: developers
- Run as part of the merge to the main branch (usually a few times a day)
- If any test fail - changes are rejected until fixes are provided
- If a bug is reported - proper tests are added or existing ones are updated to avoid regression in the future

Test Levels – API & Integration tests - statistics

- **number of tests**: 8,132
- **average run time**: 1h
- **code coverage (lines):** 94%
- **code coverage (branches – possible code execution paths)**: 79%

Test Levels – E2E tests (UI layer) - overview

These tests are ensuring that the integration between back-end and front-end is working as expected.

- responsible team: testers
- main goal is to simulate the end-user behaviour and to verify that the application is implemented in accordance with the user requirements
- complex E2E scenarios (~10% of full test scope) are executed manually by the application specialists
- atomic approach – we try to make our tests as independent as possible. Each test is focusing on specific functionality instead of verifying multiple ones. This gives us a clear overview of what is working and what is not.

Test Levels – E2E tests (UI layer) – execution frequency

- run on **daily** basis as part of the nightly build on development environment
- run **locally** by the test team members (a couple times a day, but usually not in the full scope)
- run **on demand** every time a release candidate is created and deployed to test environment

Test Levels – E2E tests (UI layer) - statistics

- **number of tests**: 1,048
- **average run time**: 3h
- **functionalities coverage:** ~70-80%
- **flakiness**: < 1%

Test Levels – E2E tests (UI layer) – tech stack

JavaScript

test code implementation

Cypress

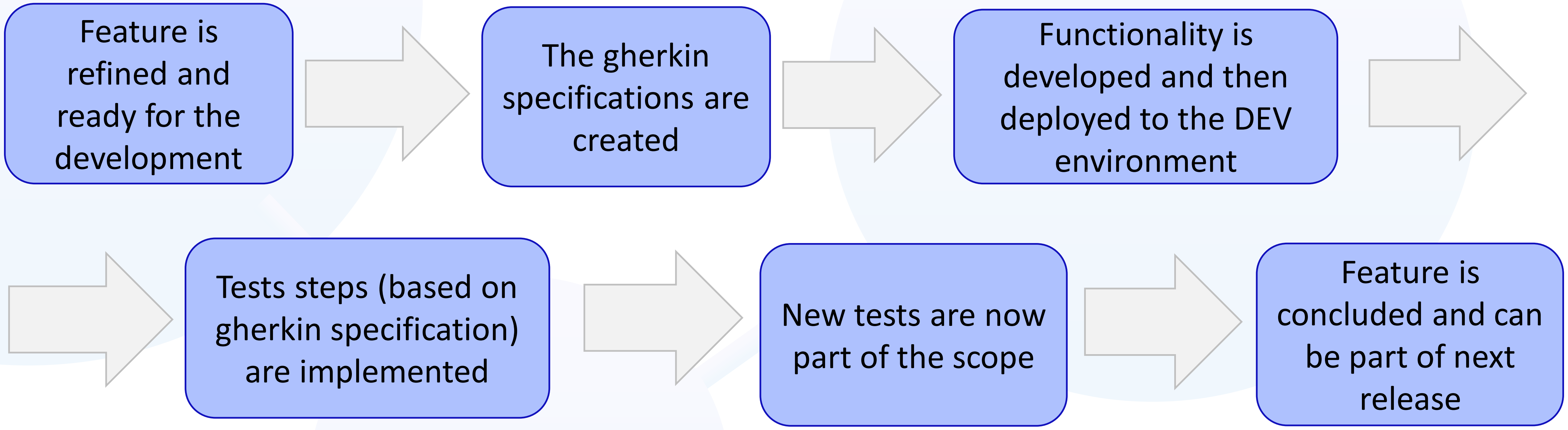
testing framework

Allure

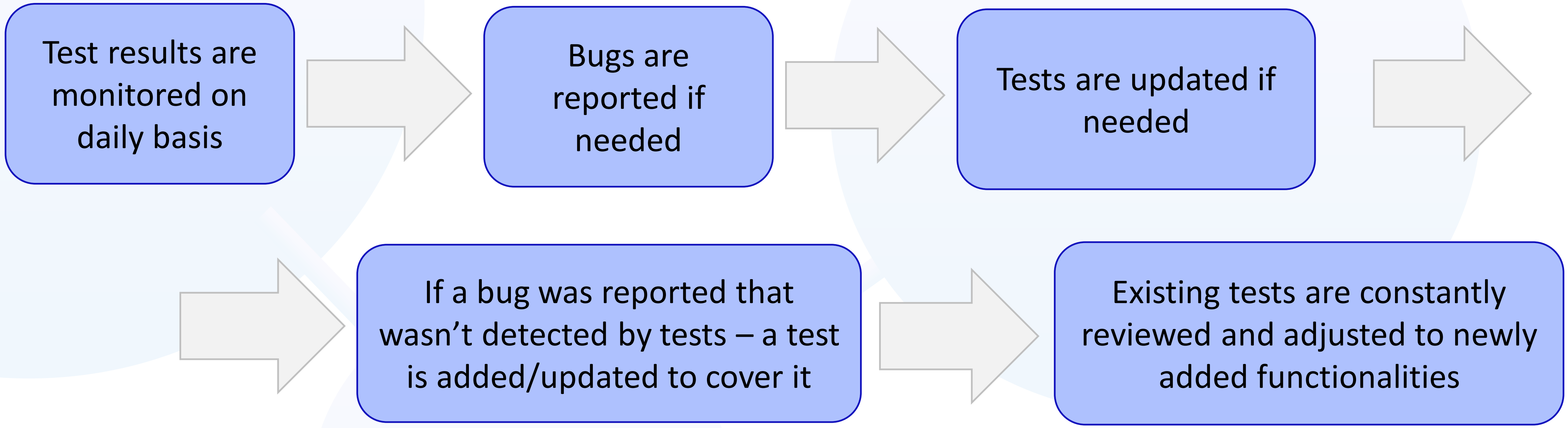
reporting

Cucumber tracking tests steps and their expected results

E2E Tests Process - development


Feature is refined and ready for the development
The gherkin specifications are created
Functionality is developed and then deployed to the DEV environment
Tests steps (based on gherkin specification) are implemented
New tests are now part of the scope
Feature is concluded and can be part of next release

E2E Tests Process - maintenance



Test results are monitored on daily basis

Bugs are reported if needed

Tests are updated if needed

If a bug was reported that wasn’t detected by tests – a test is added/updated to cover it

Existing tests are constantly reviewed and adjusted to newly added functionalities

E2E Tests Process - improvements

Test code is constantly improved to ensure low maintenance costs

If any test starts showing instability – we're immediately investigating it

We're migrating tests pre-conditions to the API calls to speed-up the execution and ensure transparent results

E2E Tests – let’s dive into the technicalities

Code structure

1. **Feature files –** this is the place where all user requirements are described in the human readable language. They’re written in Gherkin and are following Given-When-Then pattern. Their naming is reflecting the actual application structure.
2. **Steps definition files –** here you can find our tests code. It’s written using Cypress and JavaScript. Similarly to the feature files the naming pattern is reflecting the application structure (POM).
3. **Support functions** (front-end commands) - many components are reused across the application (e.g. tables, dropdowns, search). In such cases we are preparing separate files to store generic functions. This way we’re avoiding code repetition, which in turn lowers maintenance costs.

E2E Tests – let’s dive into the technicalities

Code structure

4. **Support functions** (API requests) – to avoid hardcoding values (which may vary depending on used environment) we are often fetching it via API request. Additionally, we are using API requests to inject our custom test data.
5. **Support functions** (browser operations) – the functions that are stored here are helping is selecting specific study by its id.
6. **Fixtures** – the files stored here are holding the test data that is used for creating new objects in the application

E2E Tests – let's dive into the technicalities

Test data for our tests is created via:

1. **Import scripts –** data is injected directly into the database. It must be run before tests start.
2. **API requests –** data is created via API POST calls. This is performed as part of the test.
3. **Performing actions directly in UI** –we are avoiding this approach if we can, since it slowing down the execution and can produce false-negative results. It's performed as part of the test.

E2E Tests – let’s dive into the technicalities

Running tests can be done by:

- **powershell commands –** this way tests can be run both in foreground and background. User can execute specific test suite (feature file) or run whole tests scope.
- **Cypress desktop tool –** user can select which test suite to run. The tests can only be run in the foreground mode. If the used machine has poor RAM the tests might unexpectedly crash.

E2E Tests – let’s dive into the technicalities

Report is generated using Allure

- **Test results are grouped by the feature files (suits) –** it’s easy to map failing test to the specific page in the application
- **Functionalities keywords –** to improve readability we are specifying functionality keywords (e.g. create, edit, delete) at the beginning of test name. It also improves tests sorting in the report.
- **Screenshot on fail –** if something goes wrong and test fails a screenshot is taken to help in the investigation process

E2E Tests – What's next?

Continue to improve tests coverage

Our aim is to reach 90% coverage functionality wise

Always keep the flakiness level below 2%

If we want to have reliable tests, they must be stable

Extend traceability in Allure Report

We want to include data on execution history (pass-fail rate), functionality coverage, bugs etc.

OSB Hub

THANK YOU!

https://github.com/cdisc-org/osb-hub/wiki

We need YOUR Feedback!