

# Bringing CI/CD to Digital Engineering

Unit Testing, Model Assessments, & Build Automation

**Robert Peters**

**Principal Engineer**

**Catherine Haggerty**

**Software Engineer**

**Mark Petrotta**

**Principal Engineer**

# Agenda

- About Us
- What We Can Learn From Software
- Applying CI/CD to MBSE
- "Hello World" Example
- Limitations & Future Considerations

# About Us



- 40 years combined experience in software, 20 years in MBSE.
- Have collectively seen the benefits of Continuous Integration Continuous Delivery (CI/CD) in the software world. When we were each brought to the MBSE world, we saw that these things were missing.
- We value quality and efficiency. Early detection, yay!

# What We Can Learn From Software

# Software engineering nearly broke 20 years ago

- Team Structure
- Version Control Systems
- Review Process
- Merge / Integration Issues
- Build & Test Challenges
- Traceability & Requirements
- Tooling Lock-in

1991-2005



*decentralized growth  
without tools*

2001–2010s



*heavyweight,  
centralized, contractor-  
driven processes*



**Modern era of distributed version  
control, modular architectures, CI/CD,  
and DevSecOps**

# What We Can Learn From Software

What challenges do engineers face and how does software address them?

## Common Challenges

- Manual validation leads to inefficiencies and missed errors.
- Slow feedback loops hinder agility and collaboration.
- Testing processes lack standardization and repeatability.
- Development process needs traceability across lifecycle.

## The Software CI/CD Solution

- Automated pipelines catch common errors, inefficiencies, and security risks.
- Automation enables quicker feedback loops fostering agility and collaboration.
- Automated pipelines ensure the same tests run on a standardized environment.
- Entire development process is documented, allowing for strong traceability.

Reduced Errors

Faster Feedback

Standardized Process

Enhanced Traceability

# The Software CI/CD Process

Software has developed efficient processes for addressing these engineering challenges.

- Feature requests, bugs, change requests, etc. are recorded in an issue tracking tool
- Source code version history is recorded, and branches are created for each issue
- Source code and test code are developed in conjunction, known as test-driven development
- Source code is automatically tested in a common environment using automation tools
- Releases are published to an integrated artifact repository
- Releases are deployed to active environment

# Applying CI/CD to MBSE

# Applying CI/CD to MBSE

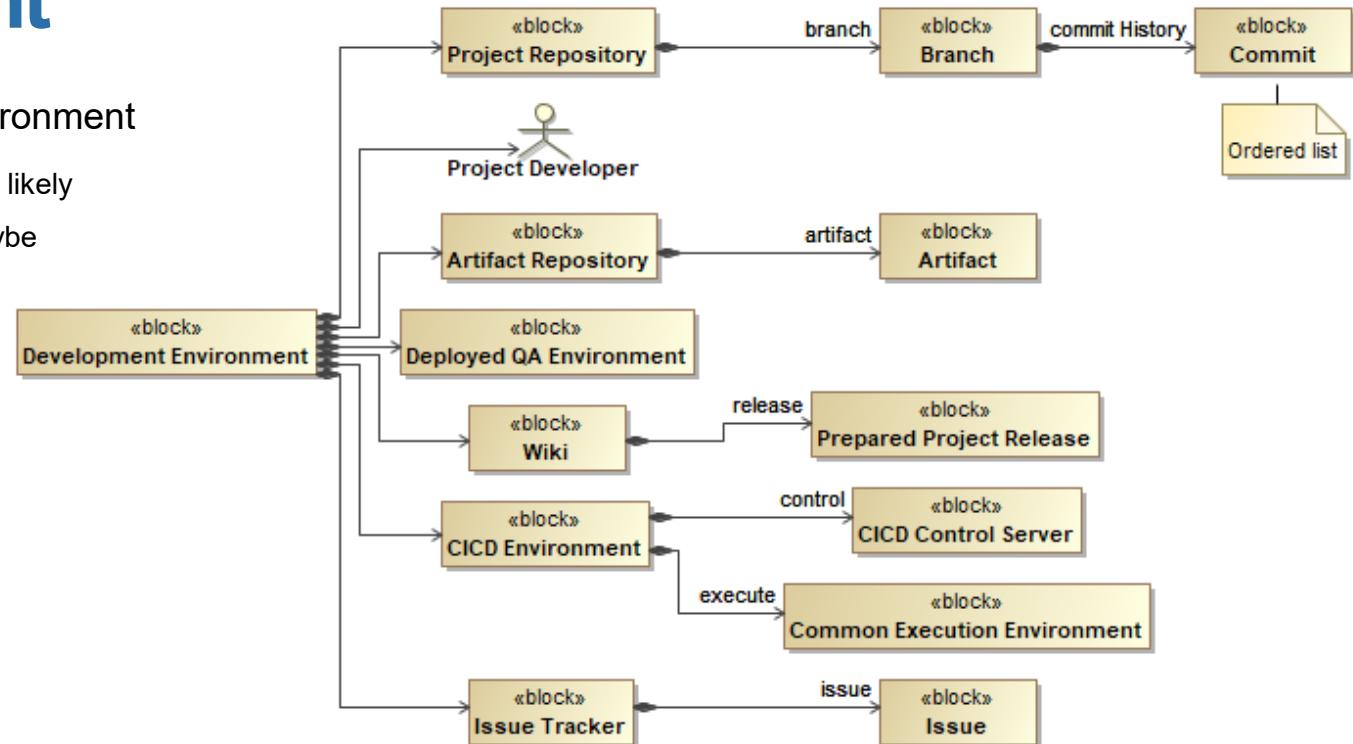
Comparing current software CI/CD solutions with current and future MBSE CI/CD solutions.

Topic	Current Software	Current MBSE	Future MBSE
Issue Tracking	Well-managed	Less formal	Improved traceability
Version Control	Feature-based branching	Minimal branching	Issue-based branching
Test-Driven Development	Unit tests for each method	Validation suites and simulation	Create model with testing in mind
Automated Testing	Runs for every push or pull request, blocks merge	Not enforced	Expanded testing, blocks merge
Integrated Artifact Repository	Build images are stored with strong traceability	Models are stored with limited traceability	MBSE artifacts are stored with strong traceability

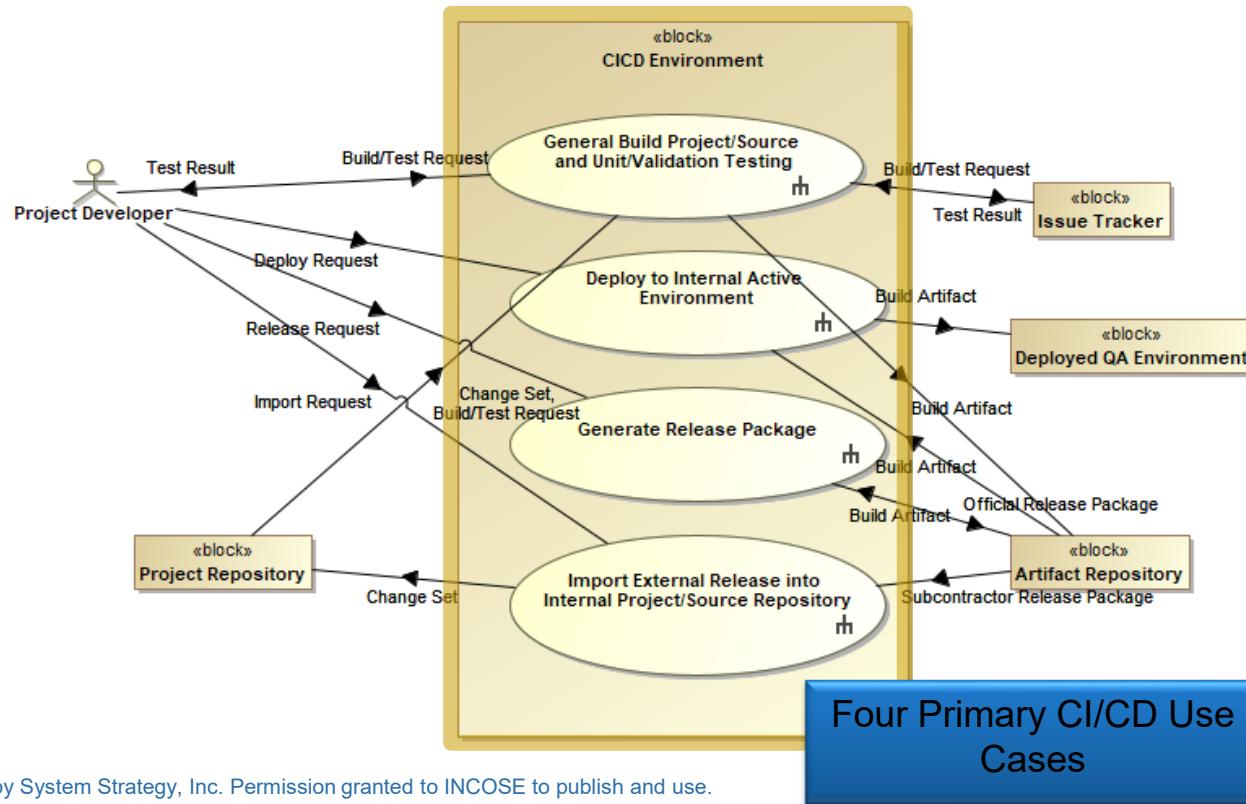
# Common Development Environment

Focus on the CI/CD Environment

- Every software or MBSE team likely has each of these, except maybe the CICD Environment



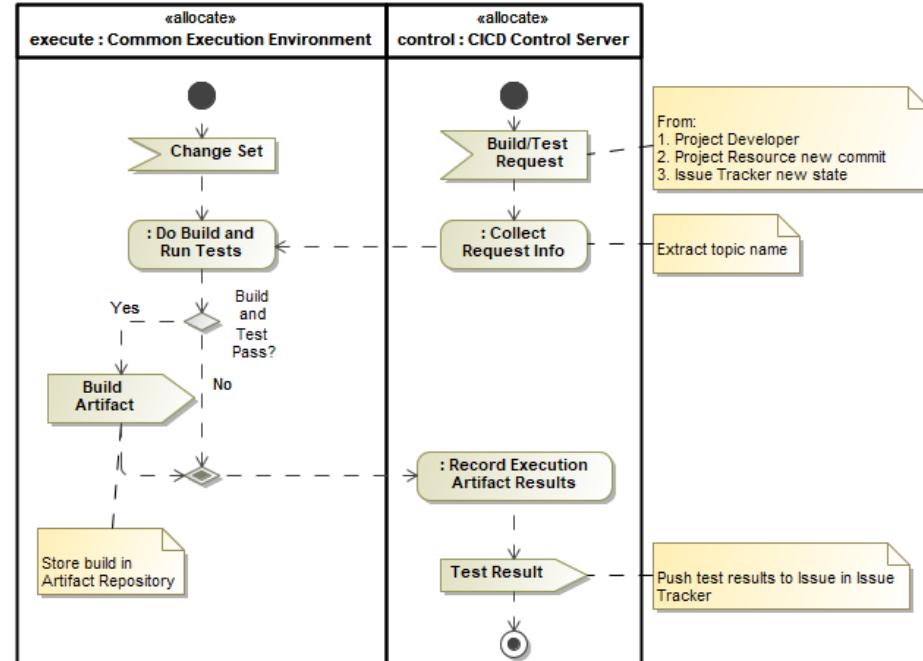
# Four Common CI/CD Use Cases for Software and MBSE



# General Build Project/Source and Unit/Validation Testing



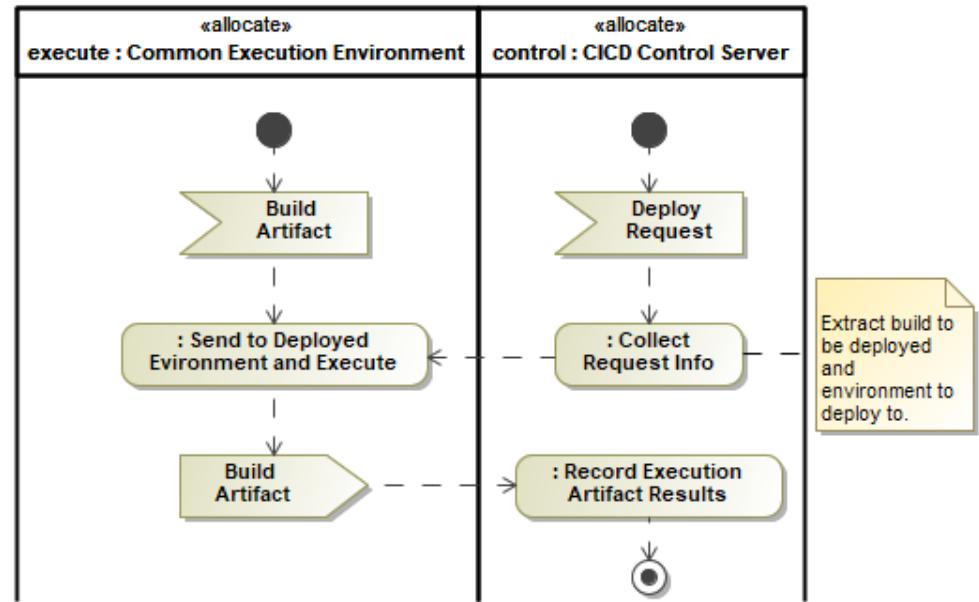
- Software Examples
  - Build SW, check for build errors
  - Run Unit Test, check for logic errors
- MBSE Examples
  - Run simulation
  - Run automatic validation checks
- Hybrid Example (Model + Code)
  - Build code that interacts with model
  - Check against test model



# Deploy to Internal Active Environment

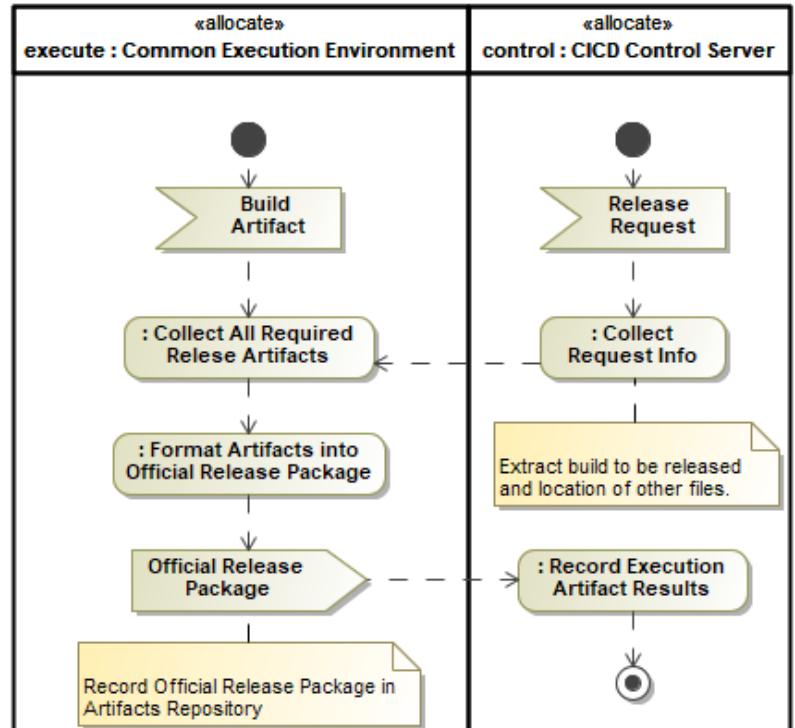


- Makes the latest changes accessible to the whole team
- In software, this may be pushing latest code on the develop branch to the dev runtime environment
- In MBSE, this may be publishing to Cameo Collaborator



# Generate Release Package

- Collect from project repository and other artifacts to create a consistent release package that is traceable to the commit history, issue history, tests, etc. and accessible via standard artifact repository
- Generate documentation for release or review
  - Extract diagrams and documentation from the model
  - Create list of change sets
  - Create list of closed issues
  - Generate the diff between two versions
- Alternate:
  - Generate metrics
  - Extract detailed custom information via scripts

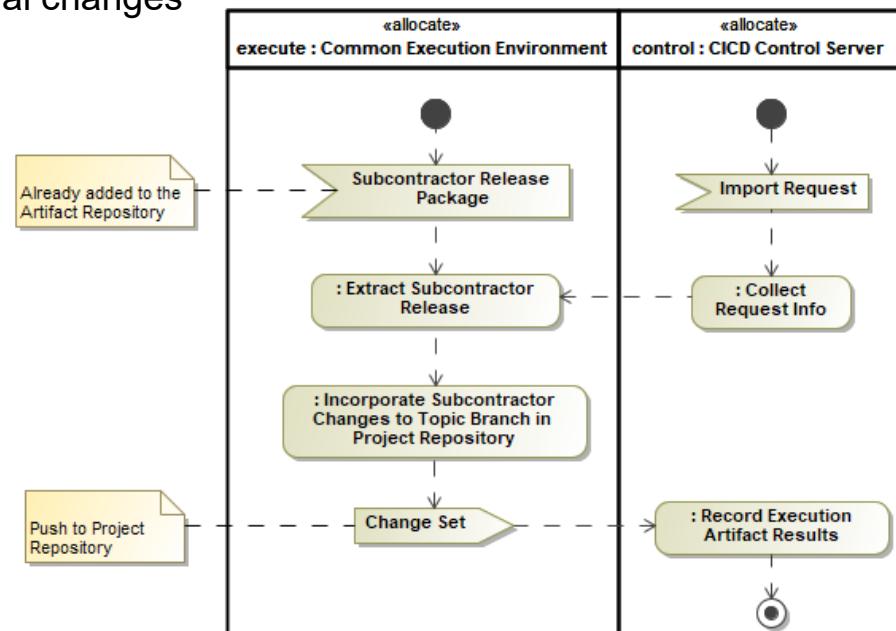


# Import External Release into Internal Project/Source Repository



## Process for receiving and accepting external changes

- External suppliers may not have direct access to the Project Repository.
- This process is to integrate the release into a change set that is part of the Project Repository that can be handled like any other change.
- Process may also run automated acceptance tests including RoC.



# “Hello World” Example of CI/CD for MBSE

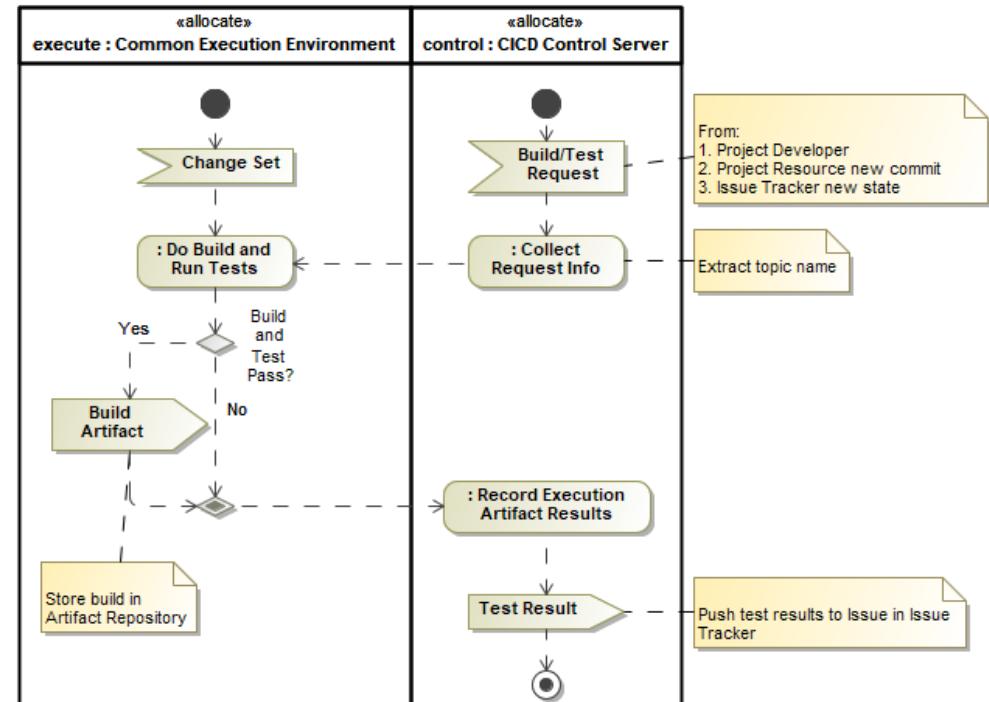
# “Hello World” Example Overview

Doing a basic test of a model

- Follows the behavior for use case “General Build Project/Source and Unit/Validation Testing”
- This test will use Jenkins as the CI/CD control server to kick off a Cameo simulation test from the command line and return the results to Jenkins



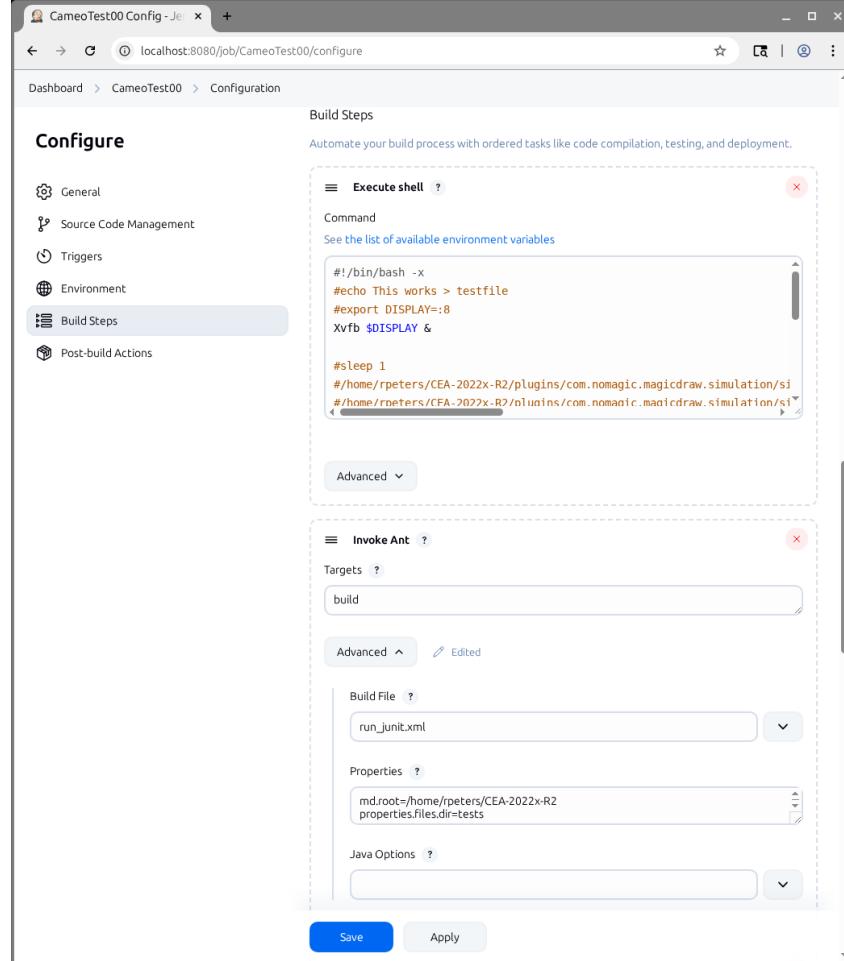
**Jenkins**



# Jenkins Project Configuration 1/2

Shows the unique configuration

- Xvfb is needed to run Cameo headless (in the background)
  - Xvfb: X virtual framebuffer
- The DISPLAY environment variable is configured on the Node
- The Ant scripts are on the local filesystem, but they would eventually be in git (or similar text-based project repository)



Configure

Build Steps

Automate your build process with ordered tasks like code compilation, testing, and deployment.

Execute shell

Command

See the list of available environment variables

```
#!/bin/bash -x
#echo This works > testfile
#export DISPLAY=:8
Xvfb $DISPLAY &

#sleep 1
#/home/rpeters/CEA-2022x-R2/plugins/com.nomagic.magicdraw.simulation/si
#/home/rpeters/CEA-2022x-R2/plugins/com.nomagic.magicdraw.simulation/si
```

Advanced

Invoke Ant

Targets

build

Advanced

Edited

Build File

run\_junit.xml

Properties

md.root=/home/rpeters/CEA-2022x-R2  
properties.files.dir=tests

Java Options

Save Apply

# Jenkins Project Configuration 2/2

Shows the unique configuration

- Kill Xvfb to cleanup for next time
- Collect the test results as build artifacts

The screenshot shows the Jenkins 'Configure' screen for the 'CameoTest00' job. The 'Post-build Actions' section is selected. It contains two steps: 'Execute shell' and 'Publish JUnit test result report'. The 'Execute shell' step has the following command:

```
#export  
#export DISPLAY=:8  
#ant -file run_junit.xml -Dmd.root=/home/rpeters/CEA-2022x-R2 -DDISPLAY=:8  
killall Xvfb
```

The 'Publish JUnit test result report' step has the following configuration:

- Test report XMLs: test-reports/\*\*TEST\*.xml
- Test output retention: All tests
- keep all the properties:
- Keep original test names, even when reporting from multiple stages:



# Example Cameo Project

The Cameo Simulation Toolkit example model.



CATIA | Cameo Enterprise Architecture 2022x - TestVerdictKind.mdzip [C:\Users\usofr\Downloads\TestSimulationCommandLine\tests\]

File Edit View Layout Diagrams Options Tools Analyze Collaborate 3DEXPERIENCE Window Help

Preview: - no preview - Full Model Create Diagram

Model [Read-Only] x

GeneratePassResult x

GenerateFailResult x

Model [Model]

«SimulationConfig» Run GeneratePassResult

```
addControlPanel = false
animationSpeed = 95
autoStart = true
autoStartActiveObjects = true
cloneReferences = false
constraintFailureAsBreakpoint = false
executionTarget = «GeneratePassResult»
fireValueChangeEvent = true
initializeReferences = false
numberOfRuns = 1
runForksInParallel = true
silent = false
solveAfterInitialization = true
startWebServer = false
timeVariableName = "simtime"
treatAllClassifiersAsActive = true
```

«SimulationConfig» Run GenerateFailResult

```
addControlPanel = false
animationSpeed = 95
autoStart = true
autoStartActiveObjects = true
cloneReferences = false
constraintFailureAsBreakpoint = false
executionTarget = «GenerateFailResult»
fireValueChangeEvent = true
initializeReferences = false
numberOfRuns = 1
runForksInParallel = true
silent = true
solveAfterInitialization = true
startWebServer = false
timeVariableName = "simtime"
treatAllClassifiersAsActive = true
```

act [Test Case] GeneratePassResult [GeneratePassResult]

```
result
```

«valueSpecification» pass

result

«valueType» verdict : VerdictKind

Case] GenerateFailResult [GenerateFailResult]

«valueSpecification» fail

result

«valueType» verdict : VerdictKind

Tools

Code Engine

Ready 100% 100%

# Jenkins Project Results 1/5

## Test Result History

- When the build is running correctly, 1 will pass and 1 will fail.
- 2 fails is when the build isn't running correctly

CameoTest00 - Jenkins

localhost:8080/job/CameoTest00/

Jenkins

Dashboard > CameoTest00 >

Status

Changes

Workspace

Build Now

Configure

Delete Project

Rebuild Last

Rename

CameoTest00

Test bed to get Jenkins to execute Cameo Unit Tests

Test Result Trend

Passed (Green), Skipped (Grey), Failed (Red)

Builds

Filter

April 19, 2025

#83 3:35 PM

March 7, 2025

#82 10:24 AM

December 2, 2024

#81 7:55 PM

November 28, 2024

#80 6:57 PM

#79 1:33 PM

#78 11:53 AM

#77 11:51 AM

#76 11:17 AM

#75 11:16 AM

#74 11:12 AM

#73 11:11 AM

#72 11:09 AM

#71 11:07 AM

Latest Test Result (2 failures ±0)

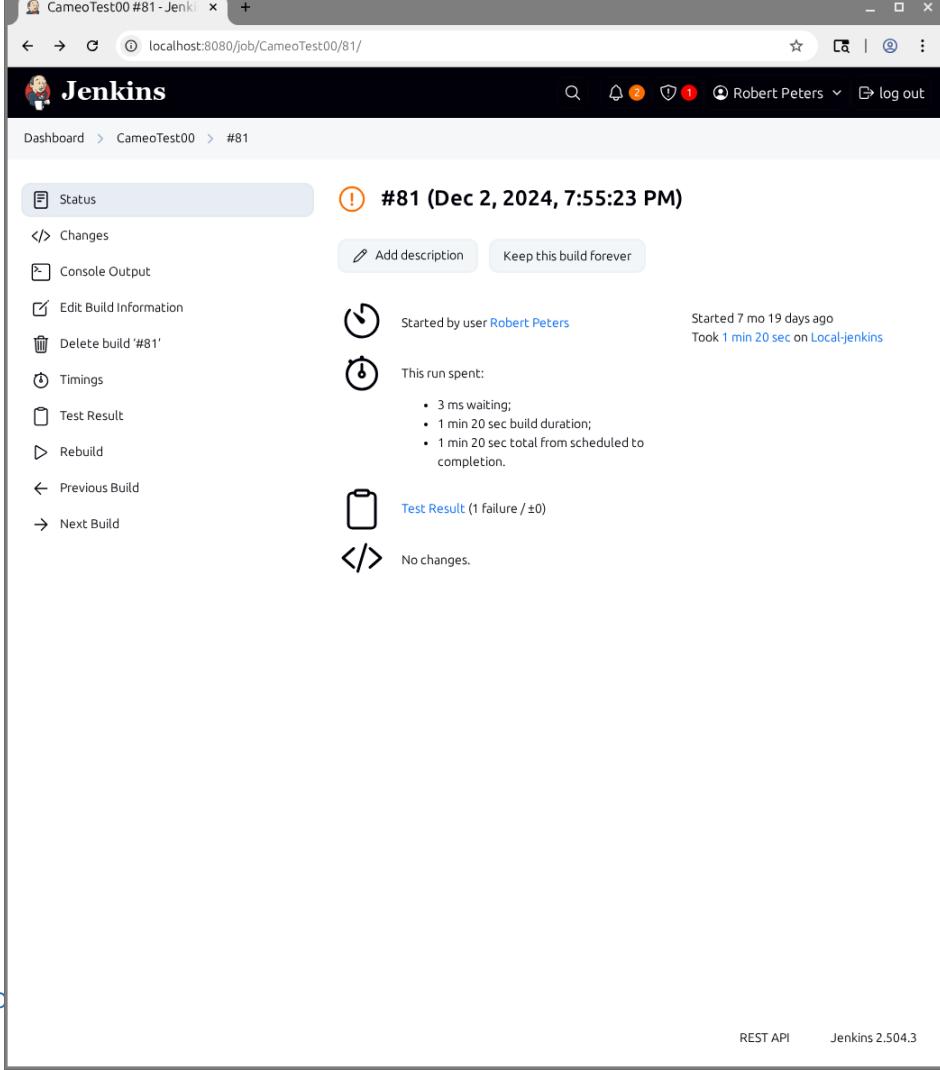
Permalinks

- Last build (#83), 3 mo 2 days ago
- Last stable build (#2), 7 mo 27 days ago
- Last successful build (#83), 3 mo 2 days ago
- Last failed build (#48), 7 mo 24 days ago
- Last unstable build (#83), 3 mo 2 days ago
- Last unsuccessful build (#83), 3 mo 2 days ago
- Last completed build (#83), 3 mo 2 days ago

# Jenkins Project Results 2/5

## Single build result

- Shows build time
- Has link to Test Result



The screenshot shows a Jenkins build results page for build #81. The page header includes the Jenkins logo, a user icon for 'Robert Peters', and a 'log out' link. The main content area displays the build number (#81), the build date (Dec 2, 2024, 7:55:23 PM), and an 'Add description' button. On the left, a sidebar lists build-related actions: Status (selected), Changes, Console Output, Edit Build Information, Delete build '#81', Timings, Test Result, and Rebuild. Below these are links for Previous Build and Next Build. The main content area on the right shows the build was started by 'Robert Peters' 7 months and 19 days ago, took 1 min 20 sec on 'Local-Jenkins', and spent 3 ms waiting, 1 min 20 sec build duration, and 1 min 20 sec total from scheduled to completion. It also shows a 'Test Result' link (1 failure / ±0) and a note that there were no changes.

Dashboard > CameoTest00 > #81

Status #81 (Dec 2, 2024, 7:55:23 PM)

Changes

Console Output

Edit Build Information

Delete build '#81'

Timings

Test Result

Rebuild

Previous Build

Next Build

Started by user Robert Peters Started 7 mo 19 days ago Took 1 min 20 sec on Local-Jenkins

This run spent:

- 3 ms waiting;
- 1 min 20 sec build duration;
- 1 min 20 sec total from scheduled to completion.

Test Result (1 failure / ±0)

No changes.

# Jenkins Project Results 3/5

## Test Result Details

- Expandable details about
  - Error Details
  - Stack Trace
  - Standard Output

The screenshot shows a Jenkins test report for 'CameoTest00 #81 Test R'. The 'Test Result' summary indicates 1 failure (±0) out of 2 tests (±0), completed in 1 min 19 sec. The 'All Failed Tests' section details a failure in 'com.nomagic.magicdraw.simulation.TestSimulationCommandLine.testGenerateFailResult'. The error details show a properties file being loaded, a simulation starting, and a 'Run GenerateFailResult' failing. The 'All Tests' section shows a summary table with 1 failure, 0 skips, 1 pass, and 2 total tests.

Package	Duration	Fail	(diff)	Skip	(diff)	Pass	(diff)	Total	(diff)
com.nomagic.magicdraw.simulation	1 min 19 sec	1		0		1		2	

# Jenkins Project Results 4/5

## All Tests

- See links for the two tests
- Time Duration
- Pass/Fail Status

Test Result : TestSimulationCommandLine

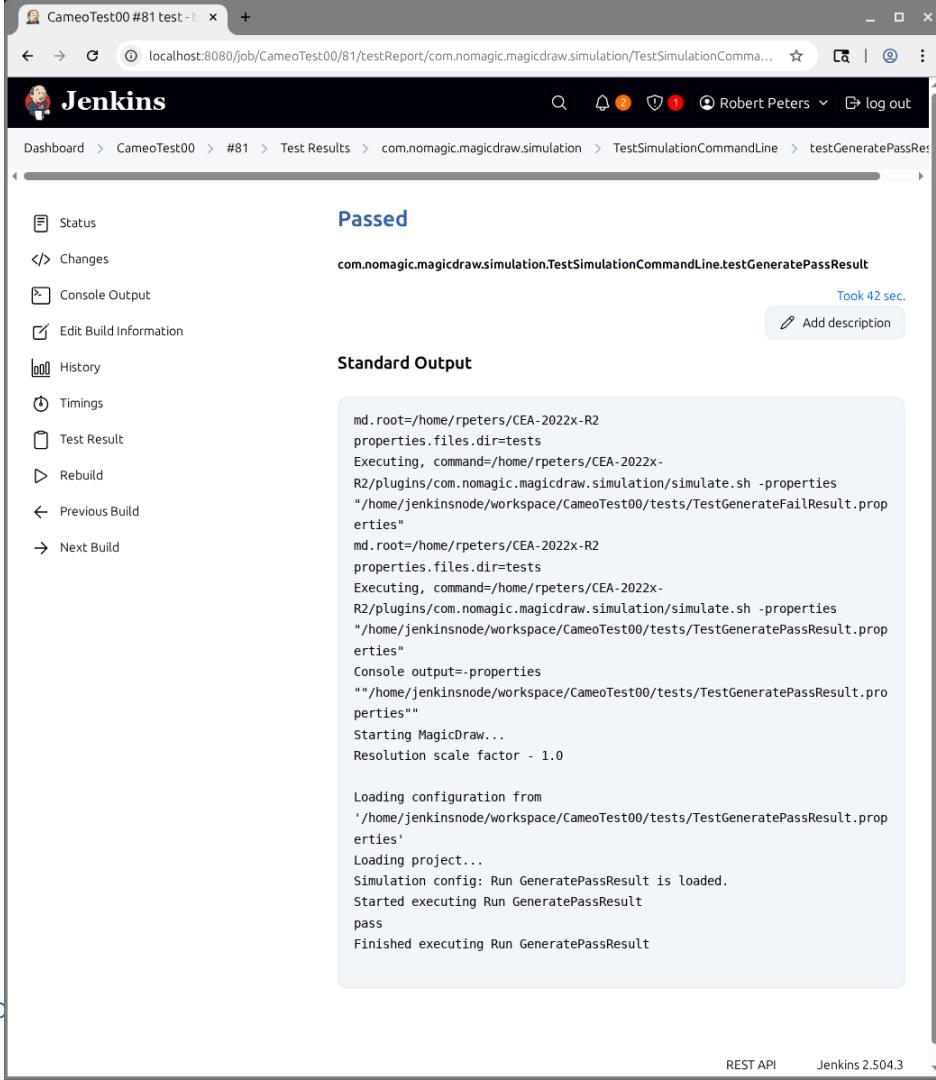
1 failures (±0) 2 tests (±0)  
Took 1 min 19 sec. Add description

Test name	Duration	Status
testGenerateFailResult	37 sec	Failed
testGeneratePassResult	42 sec	Passed

# Jenkins Project Results 5/5

## Pass Test

- Standard Output
- Note the “pass” on the line second from the bottom



The screenshot shows a Jenkins test results page for a build named 'CameoTest00 #81'. The test 'com.nomagic.magicdraw.simulation.TestSimulationCommandLine.testGeneratePassResult' has passed. The standard output shows the command being executed and the resulting output, including the word 'pass'.

**Passed**  
com.nomagic.magicdraw.simulation.TestSimulationCommandLine.testGeneratePassResult  
Took 42 sec. Add description

**Standard Output**

```
md.root=/home/rpeters/CEA-2022x-R2
properties.files.dir=tests
Executing, command=/home/rpeters/CEA-2022x-
R2/plugins/com.nomagic.magicdraw.simulation/simulate.sh -properties
"/home/jenkinsnode/workspace/CameoTest00/tests/TestGenerateFailResult.prop
erties"
md.root=/home/rpeters/CEA-2022x-R2
properties.files.dir=tests
Executing, command=/home/rpeters/CEA-2022x-
R2/plugins/com.nomagic.magicdraw.simulation/simulate.sh -properties
"/home/jenkinsnode/workspace/CameoTest00/tests/TestGeneratePassResult.prop
erties"
Console output=properties
"/home/jenkinsnode/workspace/CameoTest00/tests/TestGeneratePassResult.pro
perties"
Starting MagicDraw...
Resolution scale factor - 1.0

Loading configuration from
'/home/jenkinsnode/workspace/CameoTest00/tests/TestGeneratePassResult.prop
erties'
Loading project...
Simulation config: Run GeneratePassResult is loaded.
Started executing Run GeneratePassResult
pass
Finished executing Run GeneratePassResult
```

# Conclusion

# Conclusion - CI/CD to MBSE with Example

Comparing future MBSE goals to our “Hello World” example.

Topic	Future MBSE	What We Showed
Issue Tracking	Improved traceability	NA
Version Control	Issue-based branching	NA
Test-Driven Development	Create model with testing in mind	Check
Automated Testing	Expanded testing, blocks merge	Check
Integrated Artifact Repository	MBSE artifacts are stored with strong traceability	Partial (Jenkins)

# Conclusion

There are a lot of opportunities, but still some challenges to implementing this in your environment.

- Implementing CI/CD offers the ability to manage multiple concurrent development lifecycles across a team of developers/modelers.
- Test cases are a powerful tool for enabling quality and conformance in software and are now being realized for MBSE.
- MBSE has broader use cases. The solutions for each program using models may be different and hard to reuse as a community but should be reusable within a single company.
- The current supported mechanisms for executing MagicDraw from the command line are limited, particularly when using a CAC for login.
- SysML v2 textual notation will likely bring CI/CD for MBSE closer to software.

## Scalability

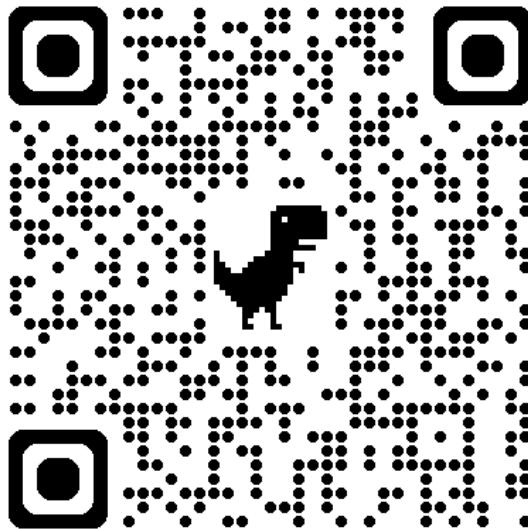
## Quality & Conformance

## Highly Configurable

## Technical Challenges

## Interdisciplinary Convergence

# Summary



CI/CD is great! Let's all start using it to automate repeated quality checks.

← QR Code to this presentation and additional materials (Will be available next week.)