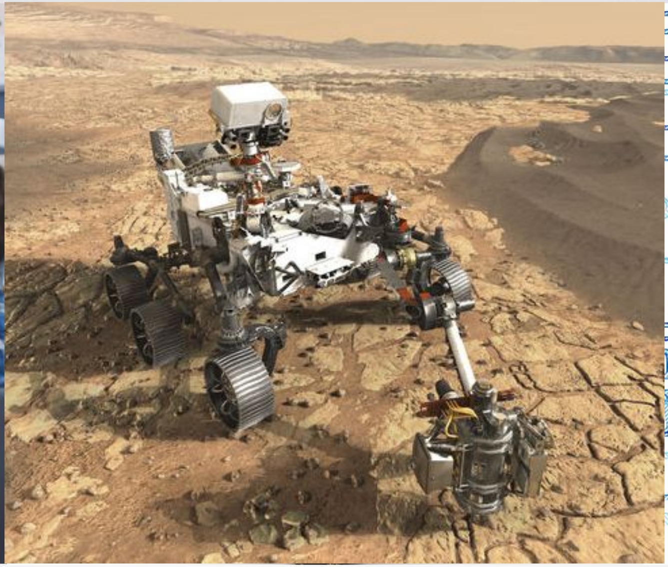
Taming Monsters with Dragons

Towards a Model-Based Product Development Process from Early Concepts to Engineering Implementation

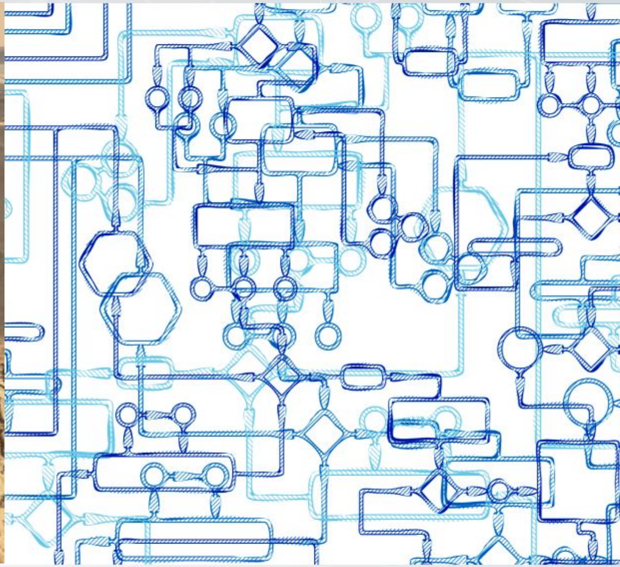
Robert Karban and Myra Lattimore
Jet Propulsion Laboratory, California Institute of Technology

Systems Engineers guide the concurrent collaborative design of complex technical systems

Leadership



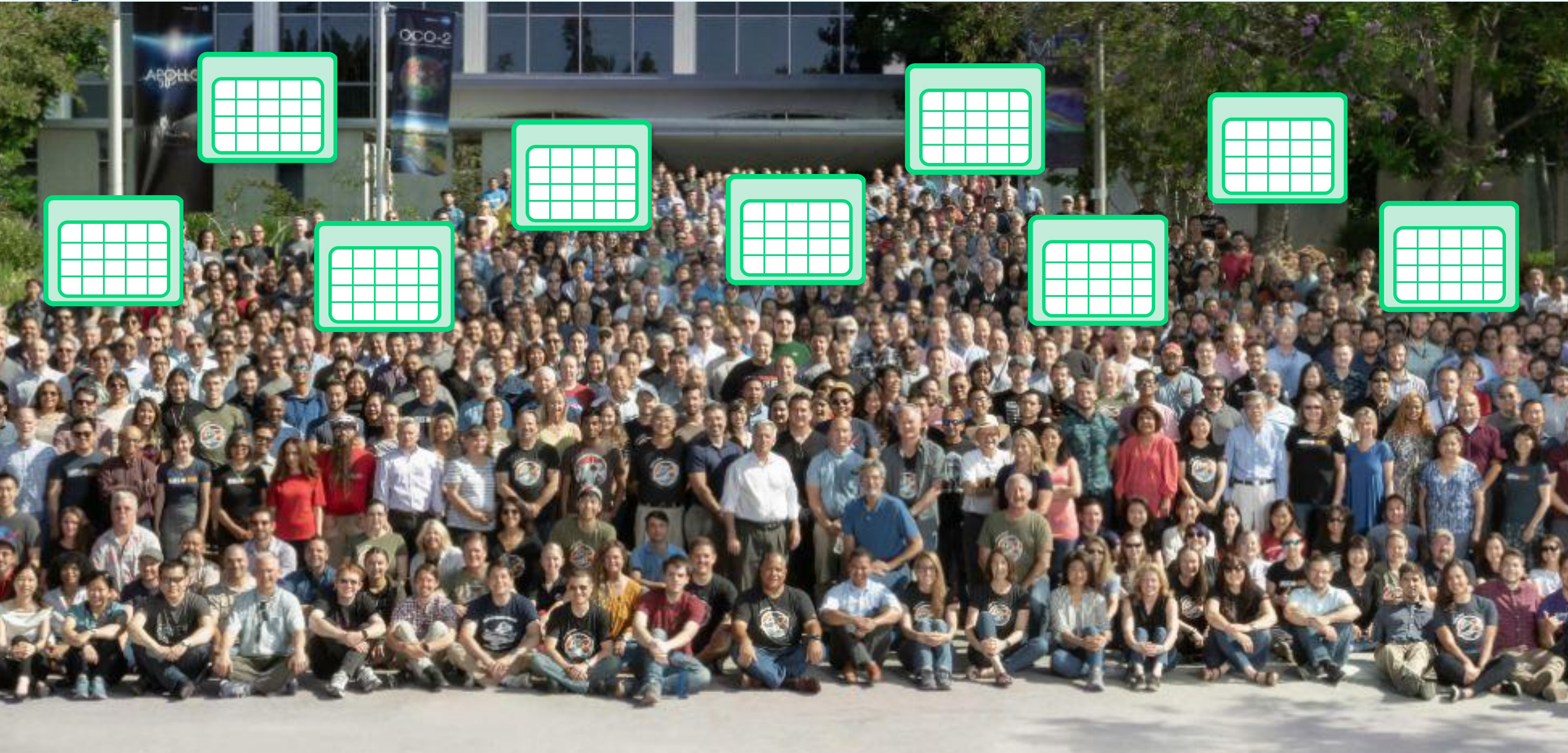
Architect and Design
Cyber-Physical Systems



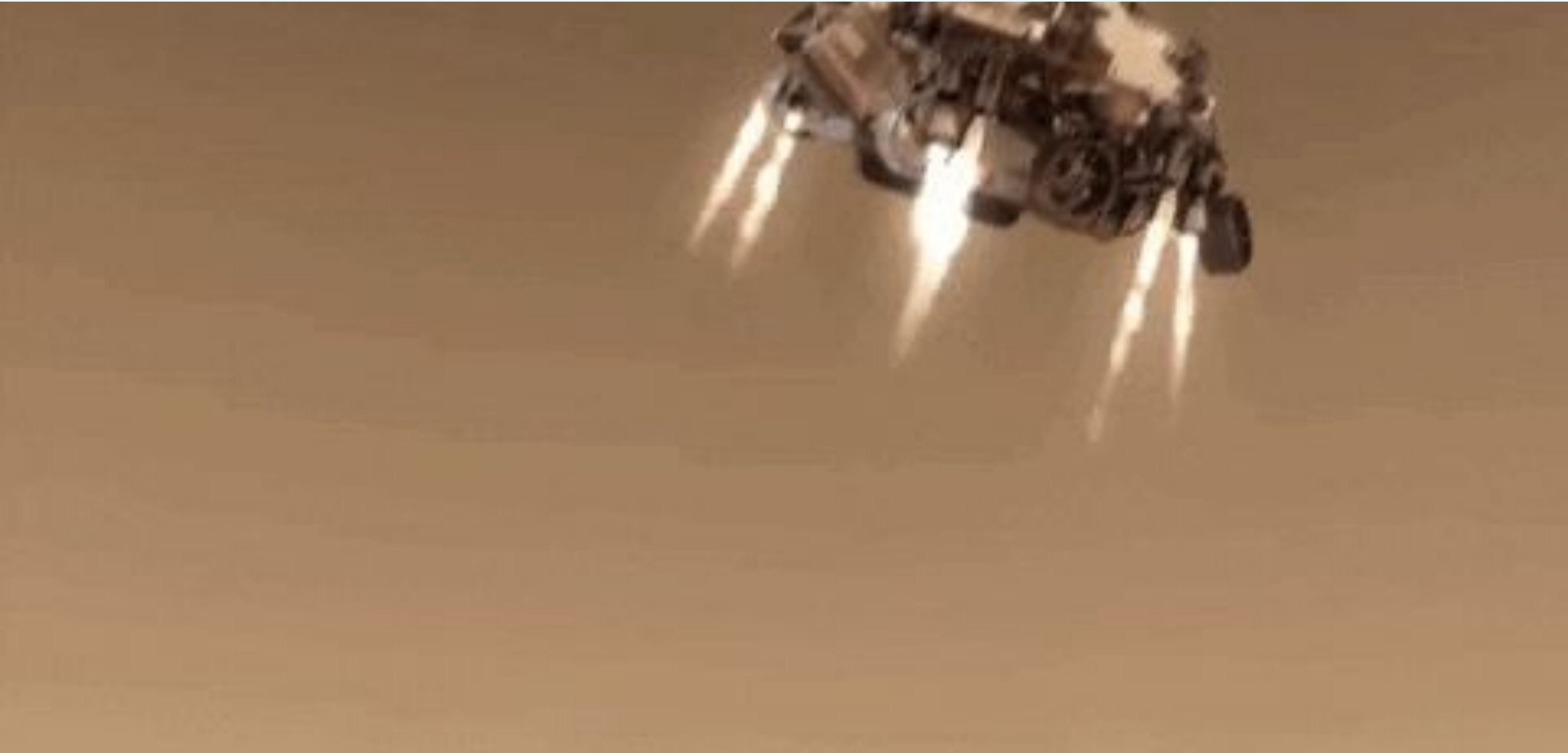
Manage Complexity

Project teams are large. Lots of people work on one project...

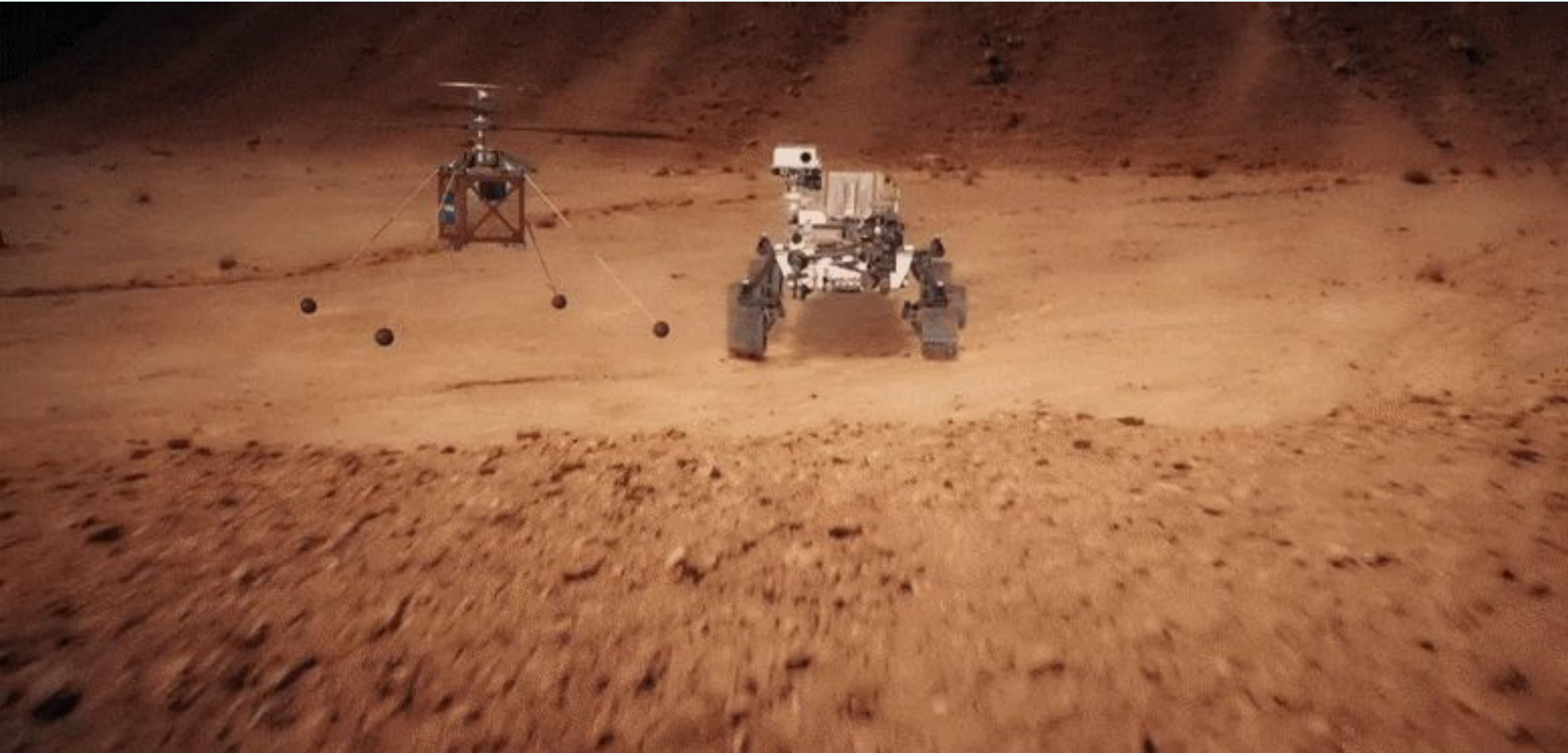
...and they communicate their project needs often through spreadsheets.



To eventually do this...

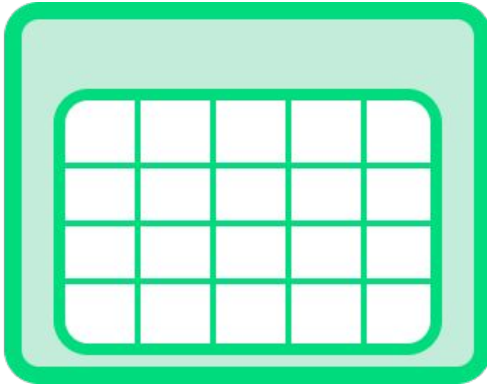


And this.   But it takes a lot to get here.



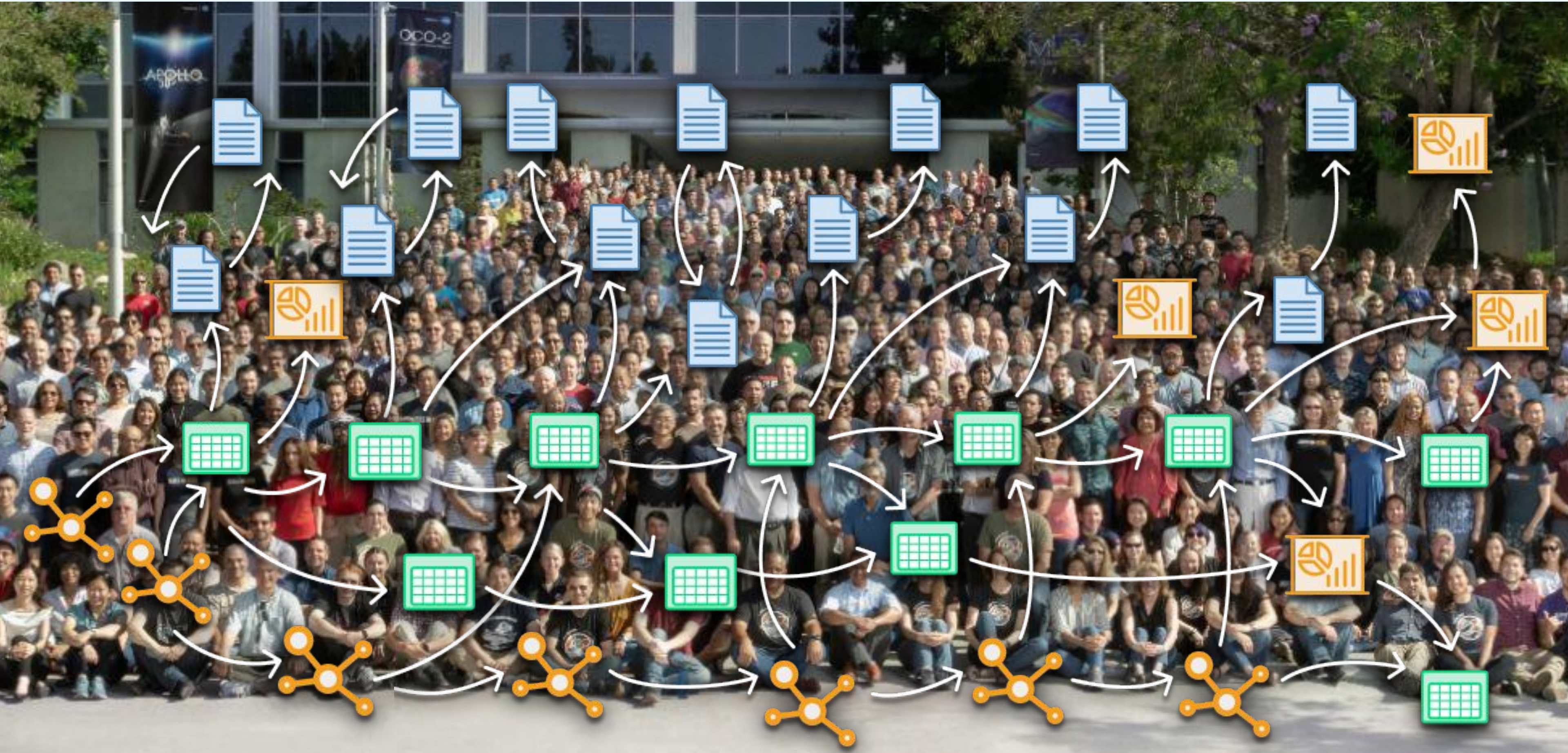
How can the systems environment help to bridge this gap?

So how do we get from this...

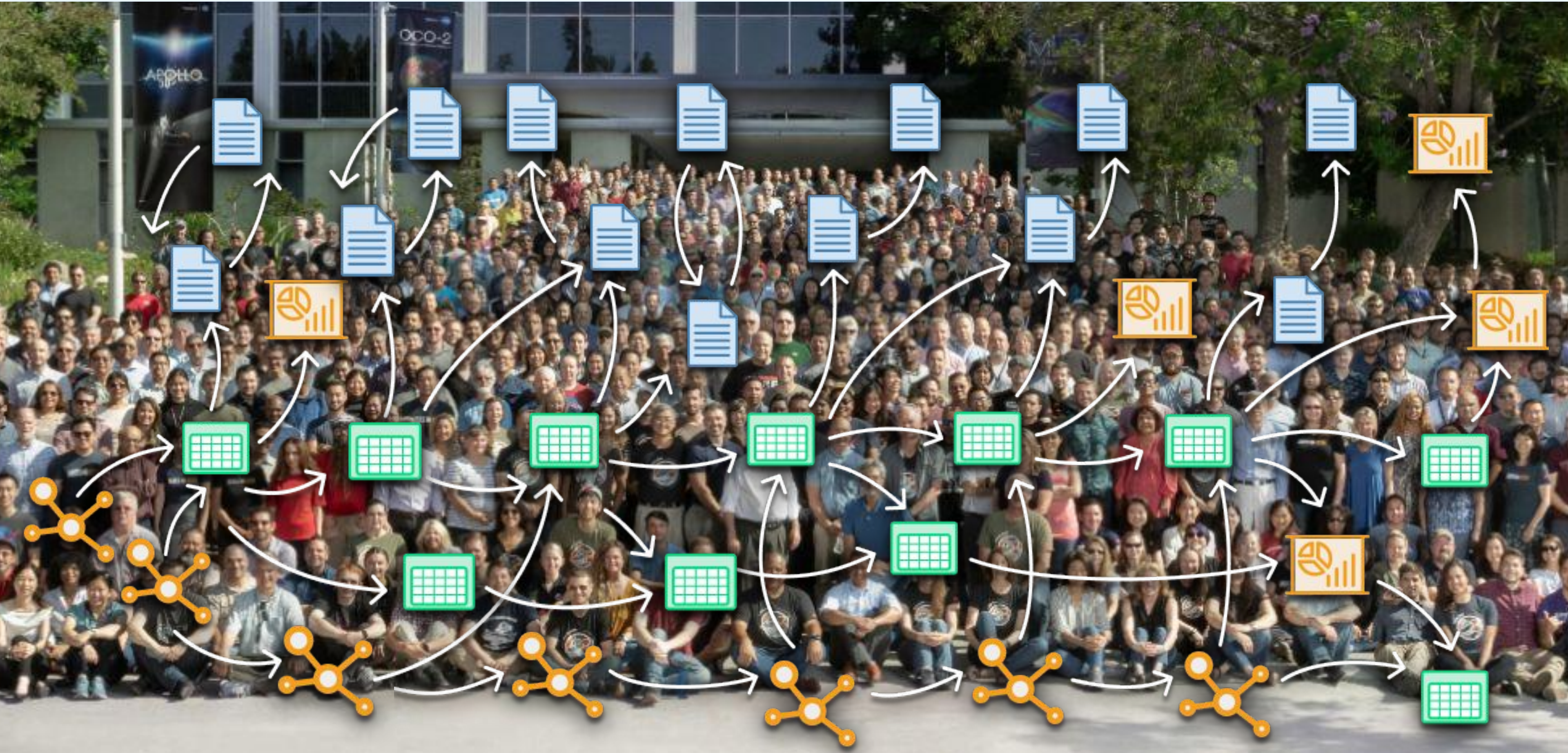


to this?

It takes people, and requirements, and spreadsheets, and documents, and processes, and workflows, and tests...



Unfortunately, all of these things create silos of information that lead to miscommunication and duplicate work.



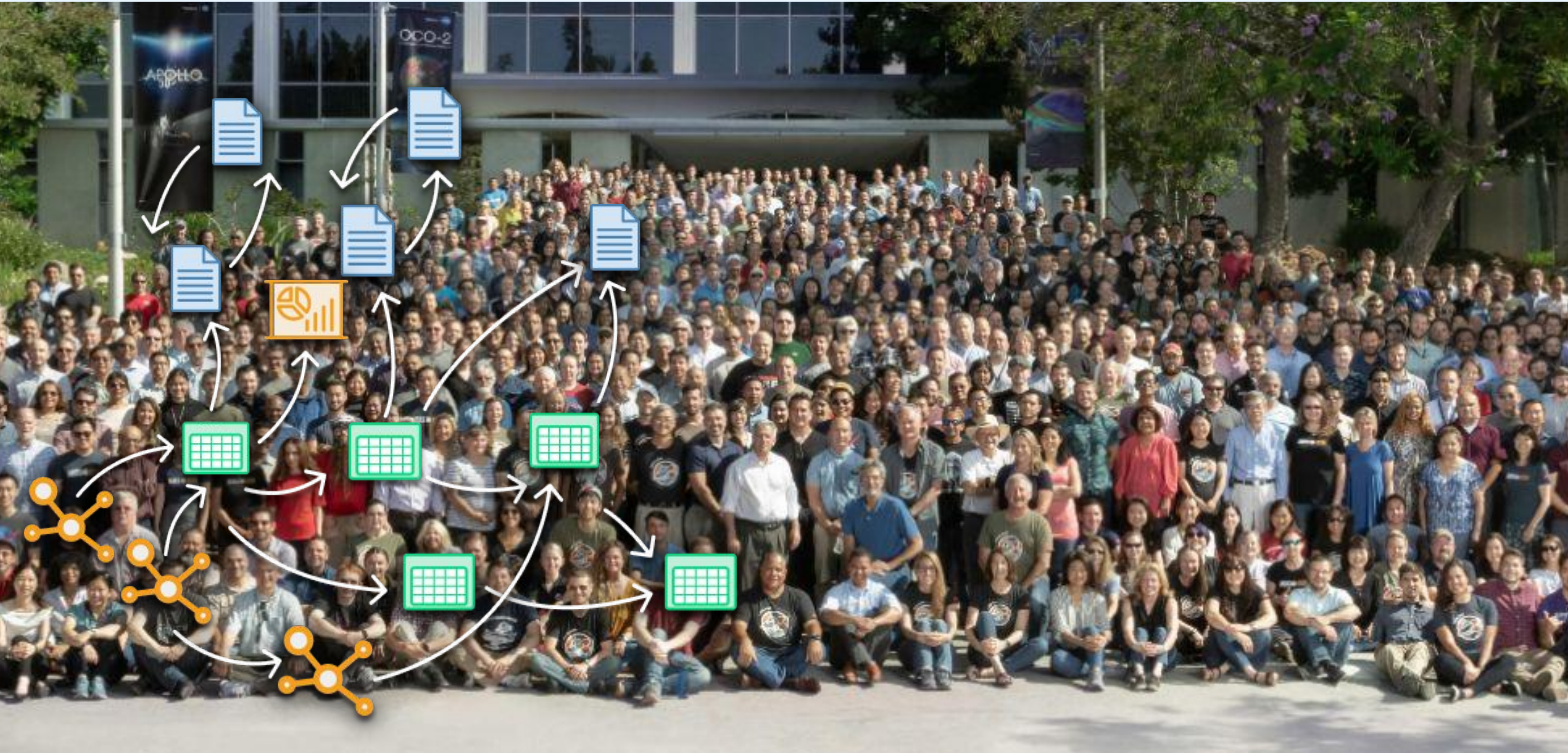
A project starts simple.

Engineers iterate on their models.

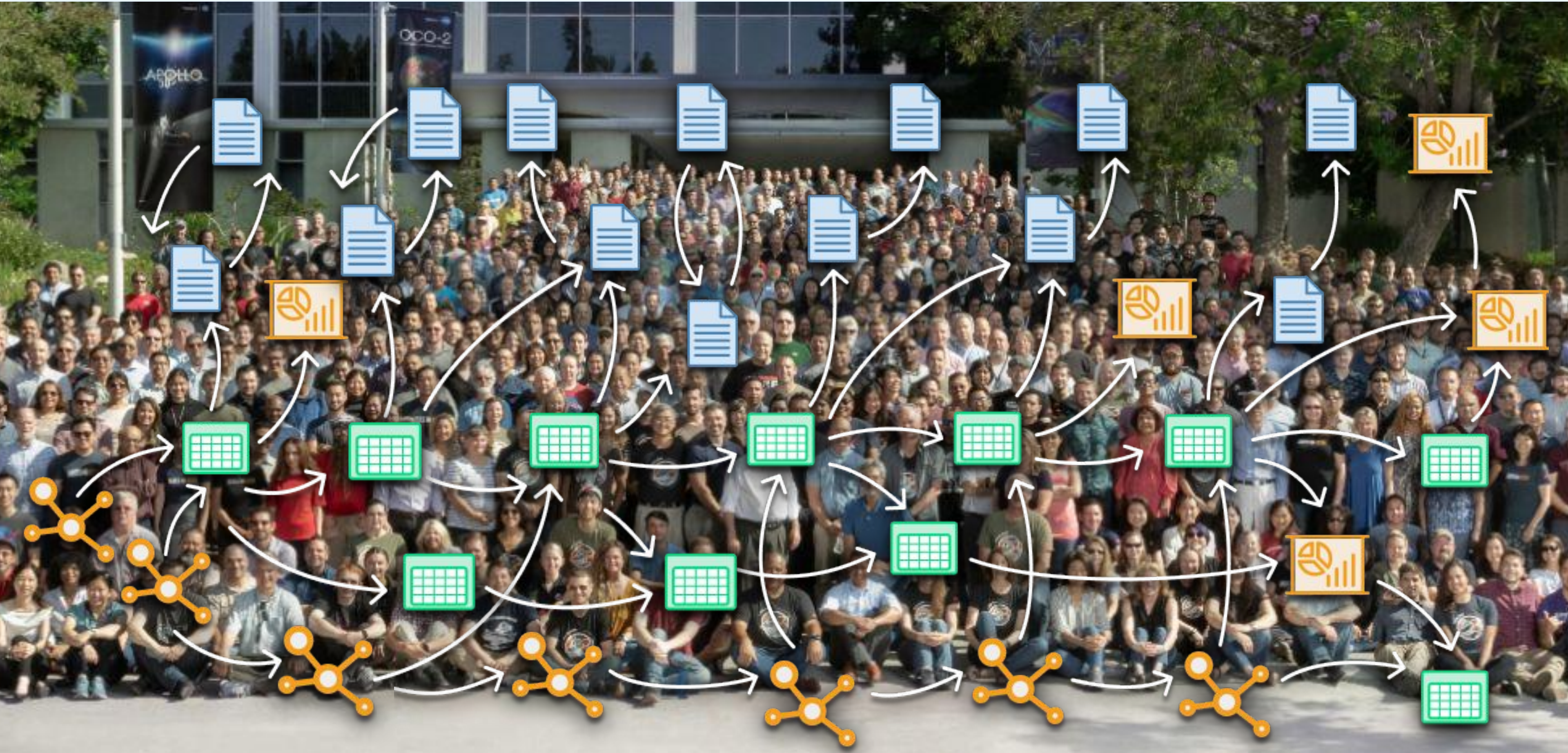
They add it to a spreadsheet to track it over time.

They add it to a document.

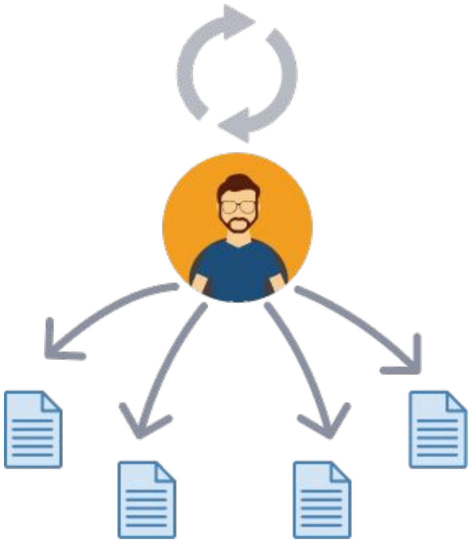
And get input from others.



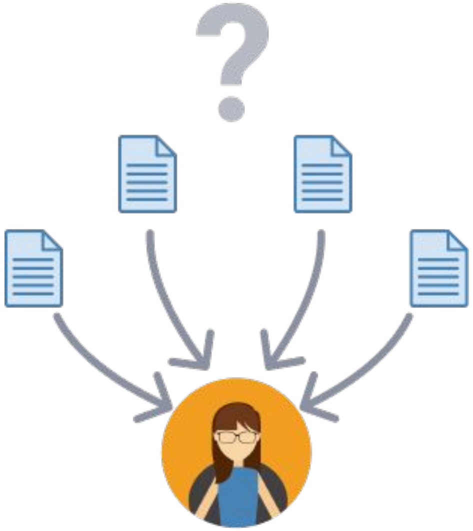
And it gets complicated pretty quickly...



Bad Ratio of Real engineering vs. overhead



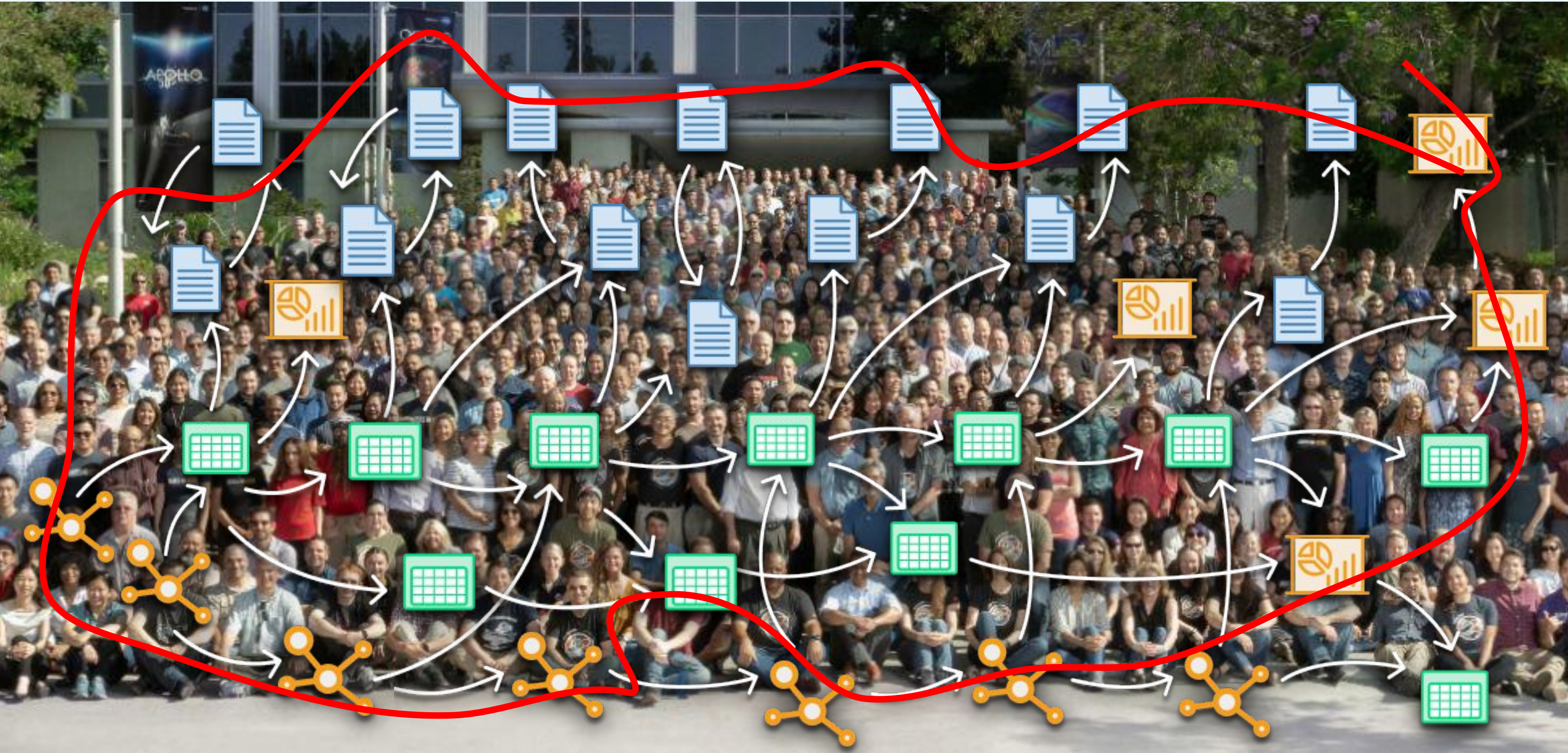
Repetitive Data Entry



Version Confusion

Constant Searching

How do we connect all of these things together?



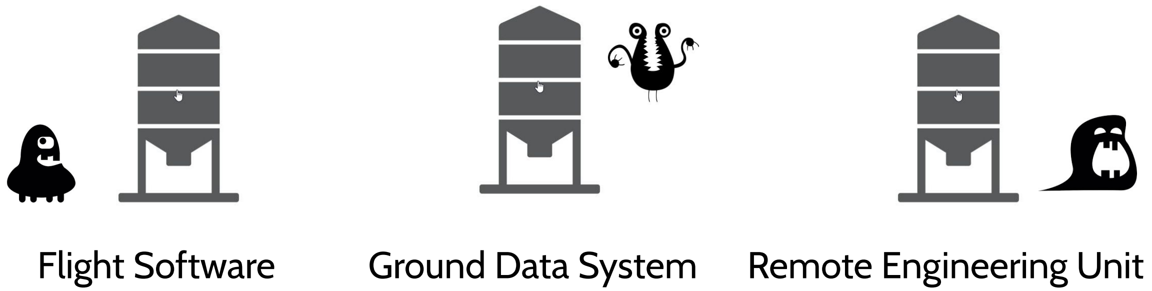
Dragons beating Monsters

Systems engineering the development process

Jet Propulsion Laboratory
California Institute of Technology

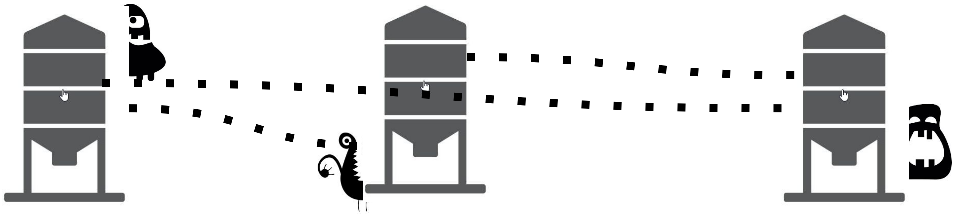
Monsters occur and reproduce

- Ever growing complexity of spacecrafts
  - More functions
  - More hardware
  - More software
- Observation that silos (the Monsters) occur, for example:



Monsters have Implicit connections

- Hard to understand relationships between silos, e.g. how to check requirements against test
- Difficulty when something changes and perform impact analysis across the board
- Monsters can be resolved by using **models** to capture relationships – turns monster into a solution

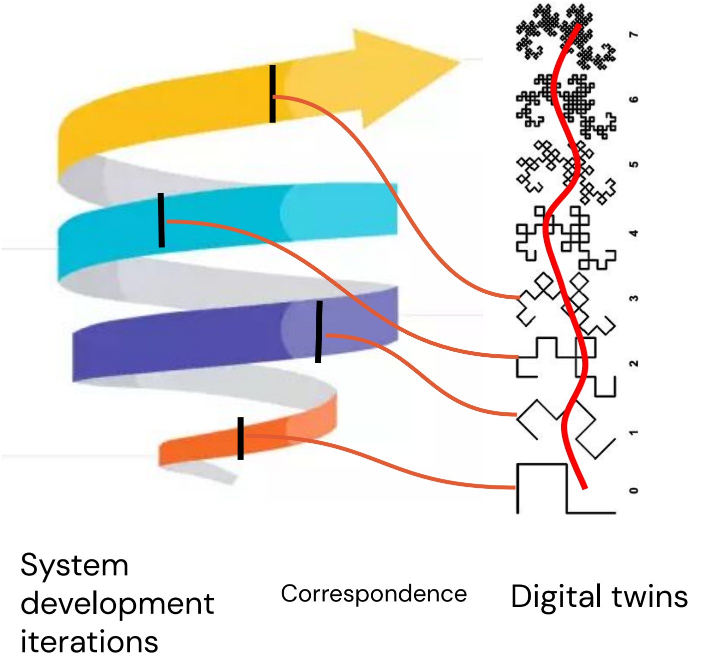


Flight Software Ground Data System Remote Engineering Unit

Monsters live in a discontinuous space

- **Overhead** to connect everything manually limits breaking up the silos
- **Disconnected** areas, implicit (e.g. in Excel).
- Qualification process does not work systematically

Digital twin pipelines evolve with system development



Digital twin - progression of detail, fidelity and clarity of what we build - the product, and produce companion products of system, e.g. model of system under control
Progress towards implementation until we have high enough fidelity so we can build the system - software defined systems

Qualification - property of twin,
Connect twins only when qualified

Lifecycle axis - how twins connect up
Progression of pipelines to complete the qualification of the system for flight, e.g. MBSE with Model checking & Simulation
Each of fractal branches is a digital twin product

Follow an evolutionary pattern to keep things connected

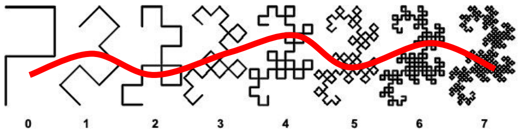
- **Expand pipelines** evolving them as a digital twin
- Manage **Change packages**
- **Fractal approach**: Each qualification step remains valid over time or elaborated, e.g. scenarios in simulation are still valid for SW testing later on; levels of requirements
- **Spiral build** of system representation

Make process of systematically connecting information a commodity

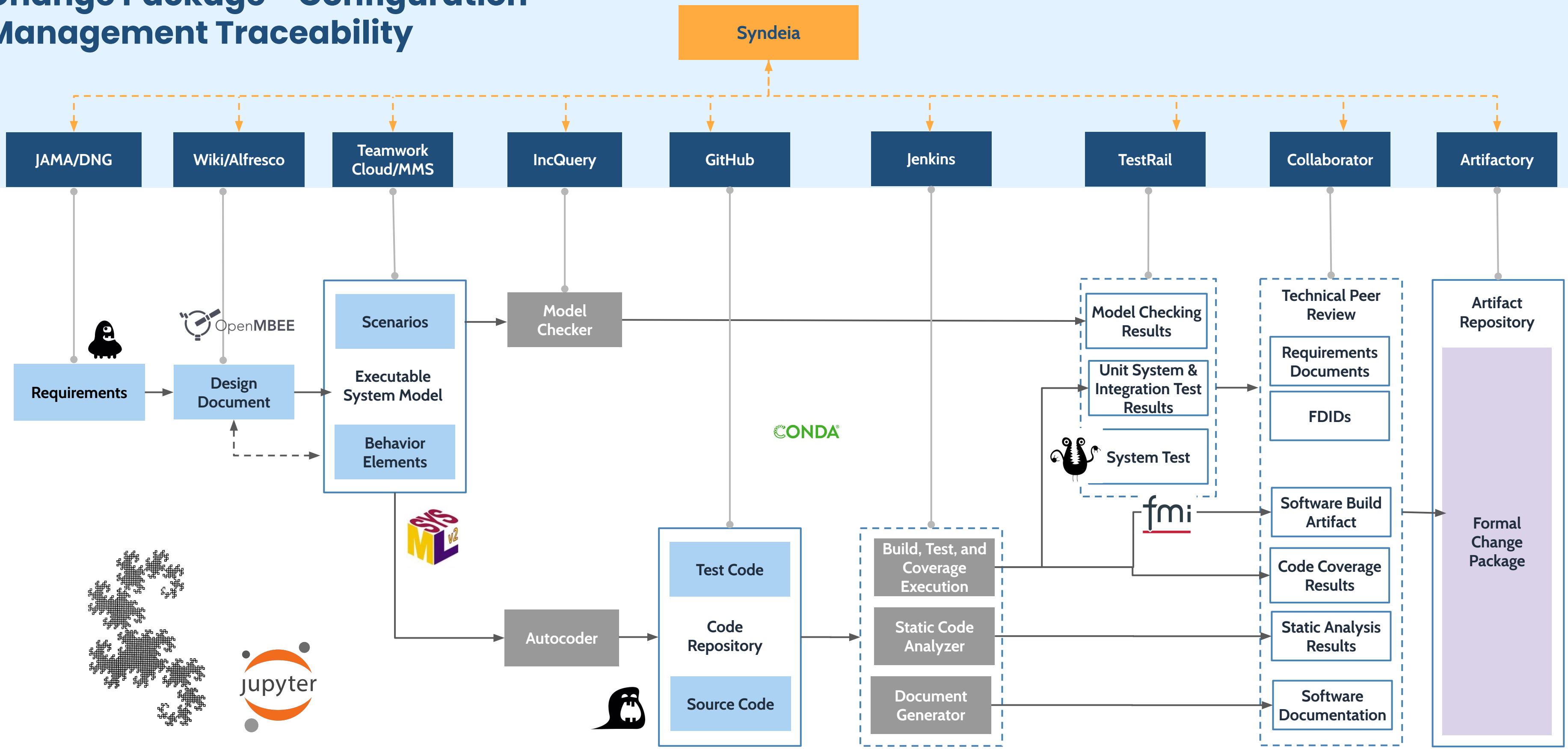
- Project **spiral of system development** into **fractal pipelines** with increasing detail and fidelity
- Have a degree of **qualification comprehensiveness** for each twin (increment in the spiral)
- **Each pipeline builds on top of the other –** progress in terms of fidelity

Dragon: Formal Qualification of Systems Modeling and Software

- **Repeatable, executable representation** of the system and its relationships at different levels of fidelity
- **Explicit** qualification record
- Relationship between representation – the **progression**
- **Systematic** process for developing the system



Change Package – Configuration Management Traceability



DevOps – Continuous Integration – Simulation

Process: Issue Management • Continuous Integration • Process Orchestration

JIRA | Jenkins | XLRelease

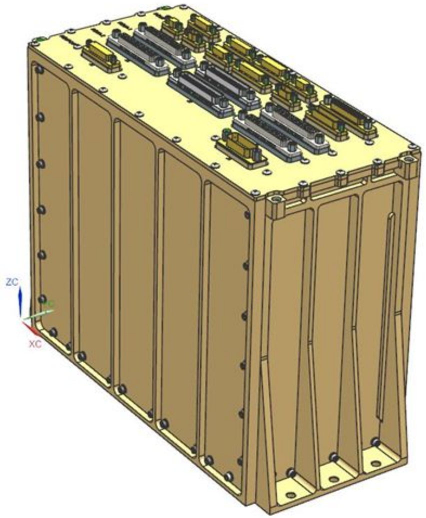
Example of Dragon

Europa Clipper REU Use Case as a starting point

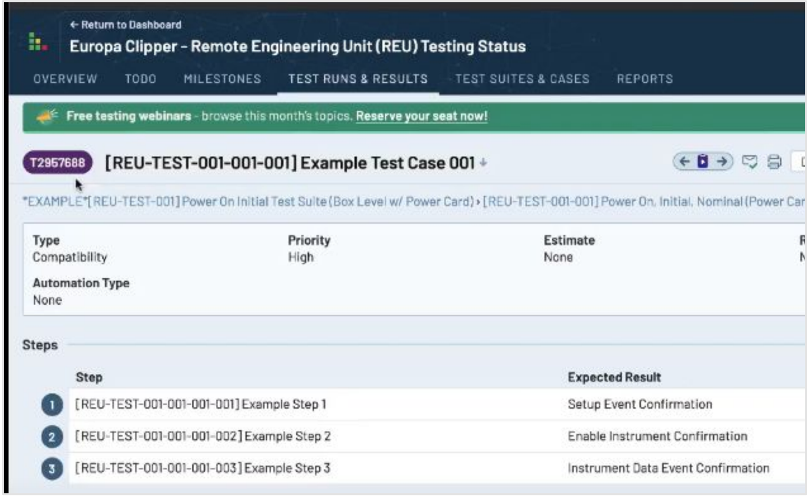
Europa Clipper Remote Engineering Unit (REU)

Capabilities:

- Manage and develop test suites and cases
- Trace requirements to test suites
- Analyze and report traceability matrices
- Explicitly traced and querible artifacts
  - Requirements
  - Test cases
  - Log cases
  - Test results
  - Generated reports



Test Management

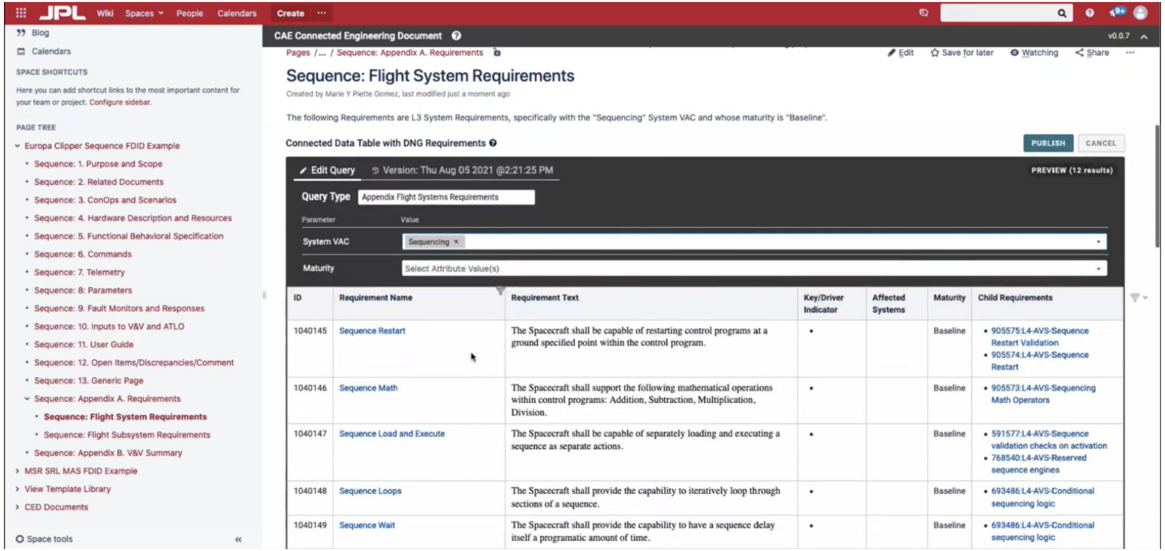


Requirements Traceability

Nexus/OpenMBEE ViewEditor

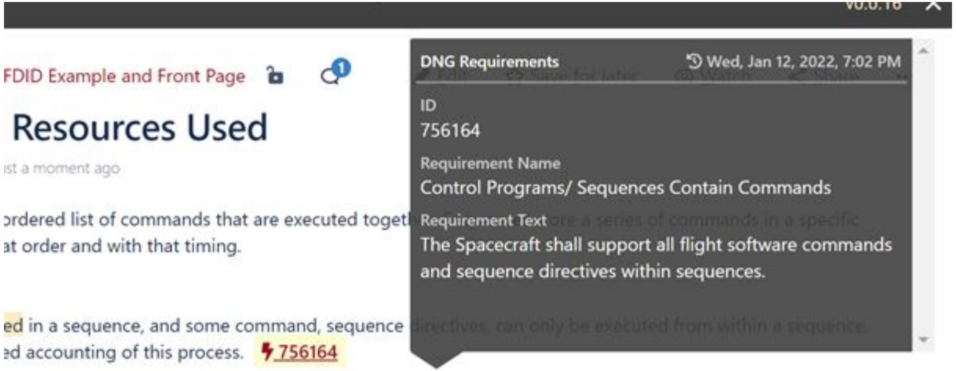
The Nexus product is a collaborative platform for Systems Engineers to automatically generate documents and integrate data from other repositories in a coherent fashion

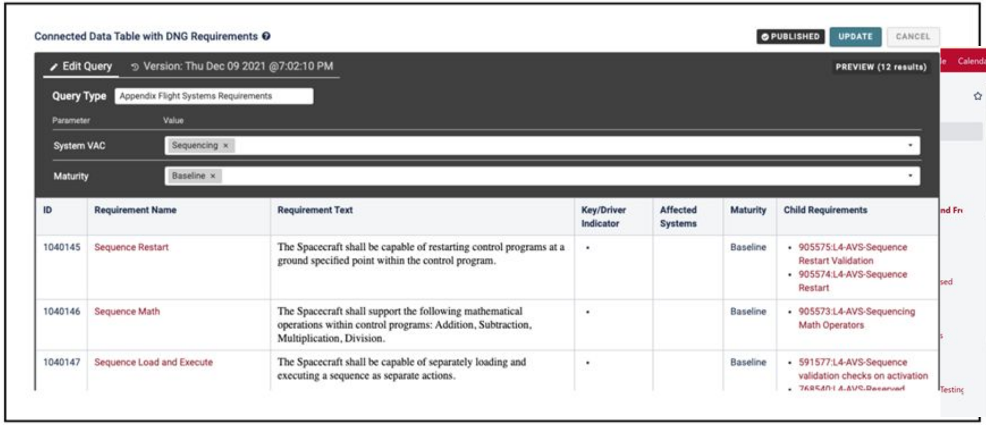
- Connects data to web-based engineering documents on Confluence Wiki
- Manages updates and changes to connected information
- Supports the document collaboration, review, and export process
- Provides document and requirement visualization
- Maintains the single, authoritative source in a collaborative environment

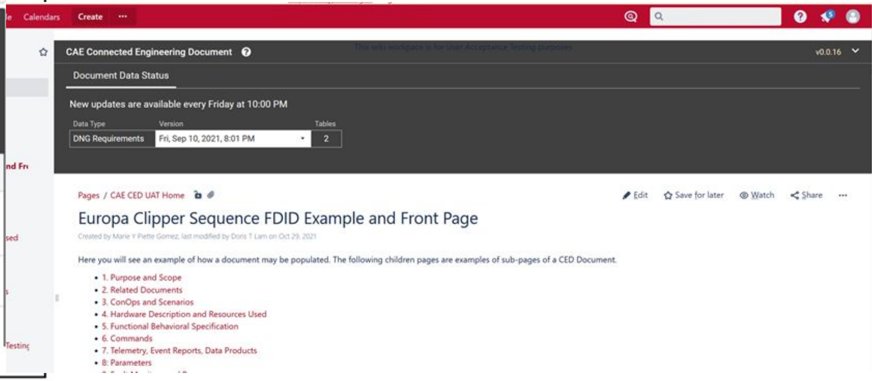


Nexus helps authoring Model Based engineering documents

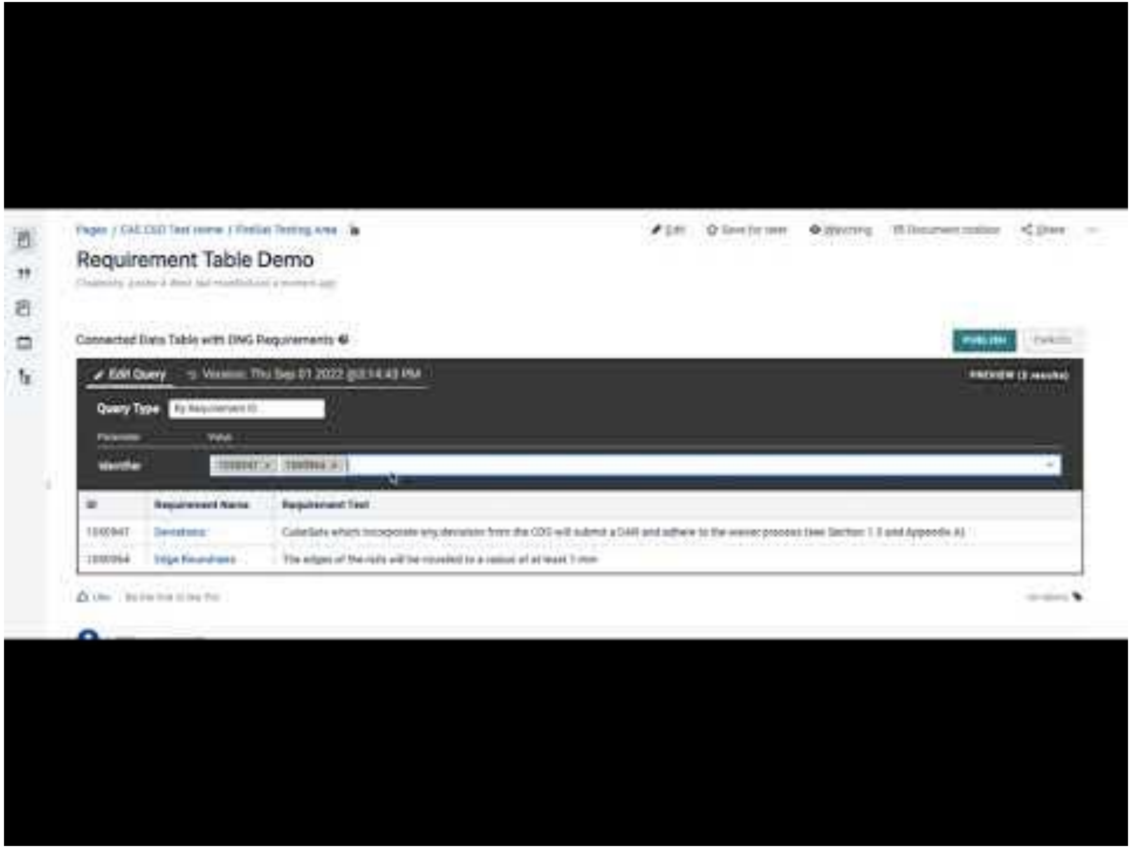
- Insert rich hover references to either DNG requirements or other Content.
- Live tables for Requirements and Dictionary data
- Configuration Managed Data updates







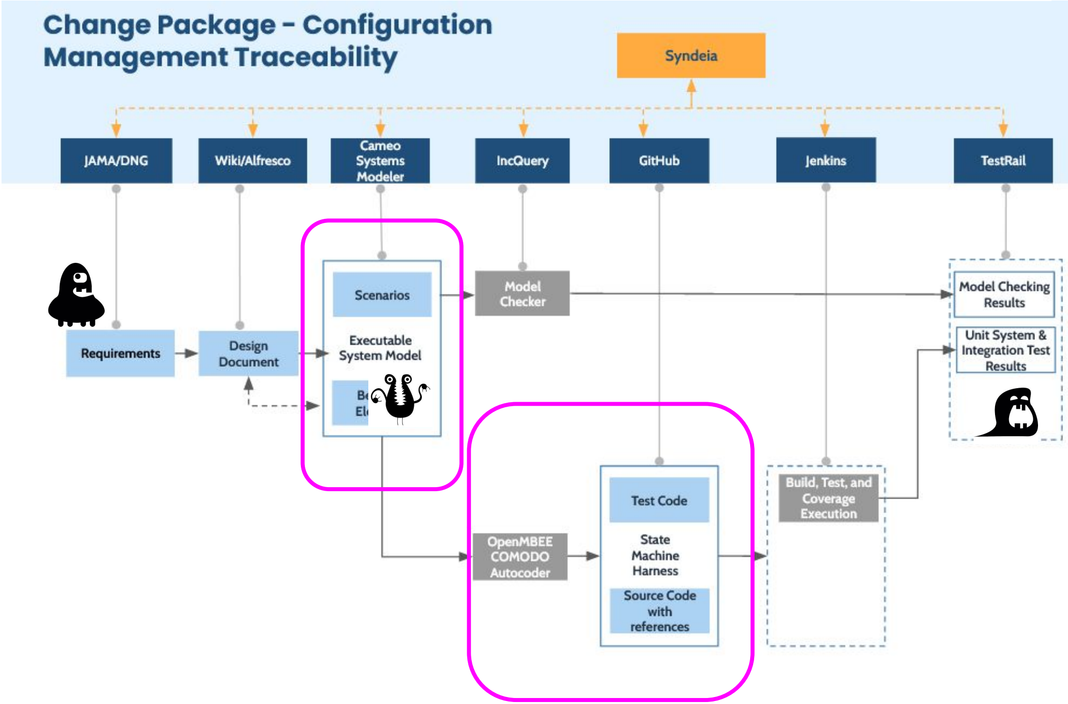
Demo: FDIDs as linked documents



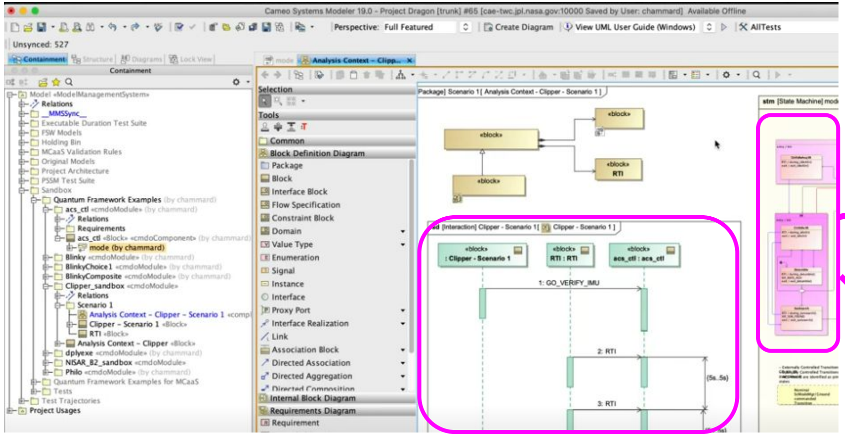
Pipeline Prototypes

Model Execution, Code Generation and Test

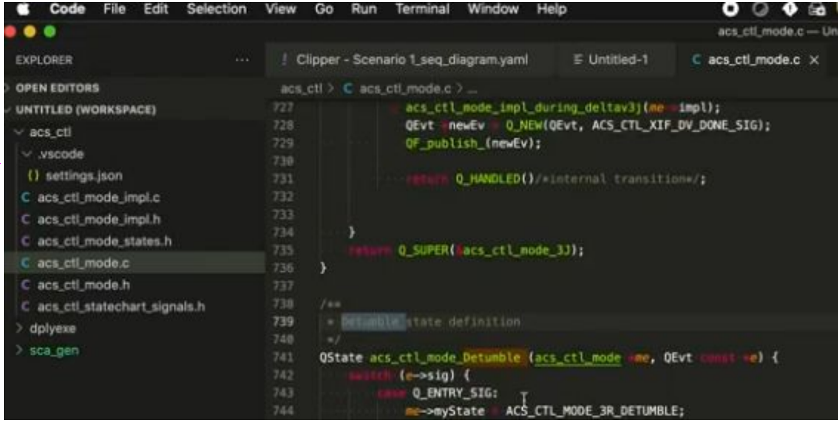
- FSW models revised to incorporate **executable simulation**
- Requirements drive **scenario tests** in the form of sequence diagrams
- Syndeia **traces requirements** and Testrail to systems model and scenario tests
- **Traverse the model** (state machine and sequence diagrams).
- **Generate** source code & test sequences.



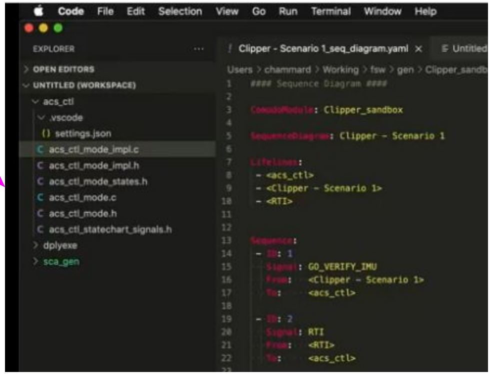
Code Generation with OpenMBEE COMODO



App code

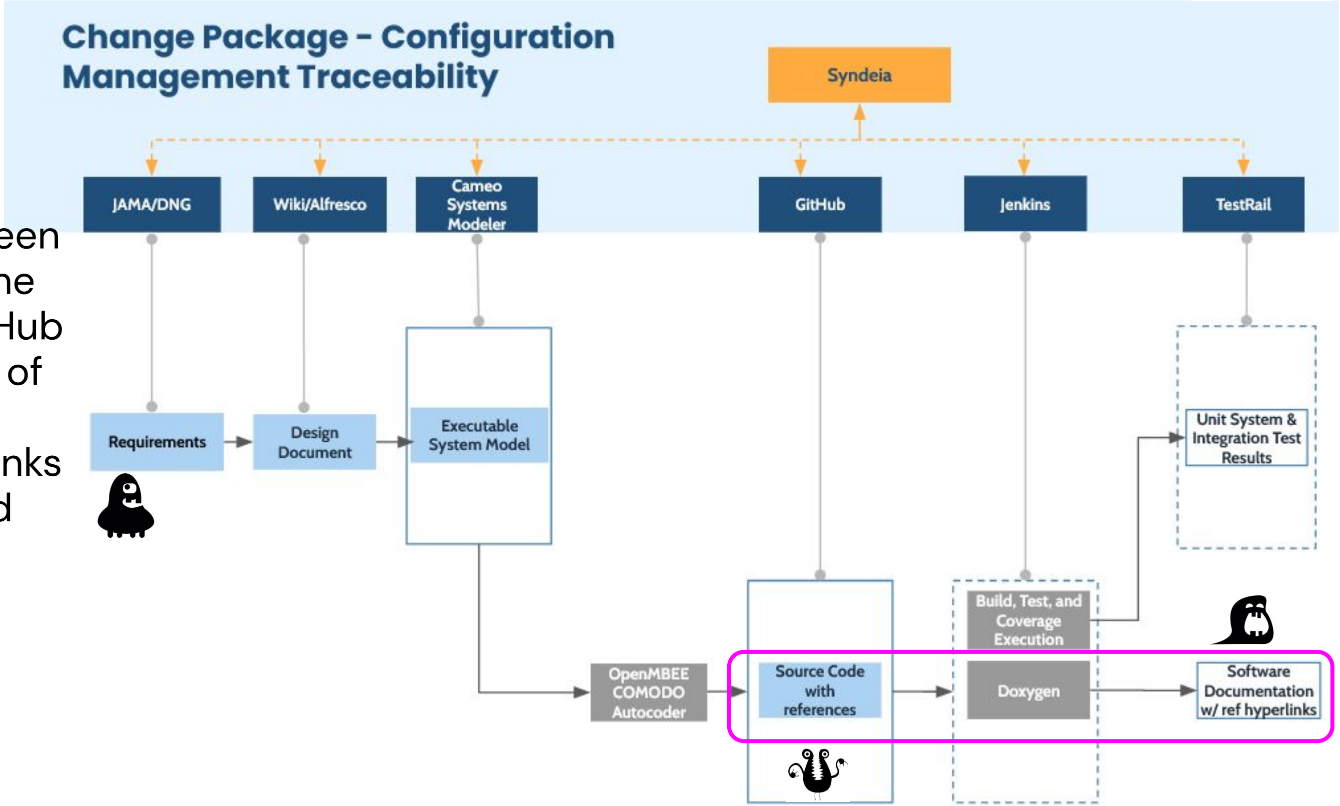


Test code

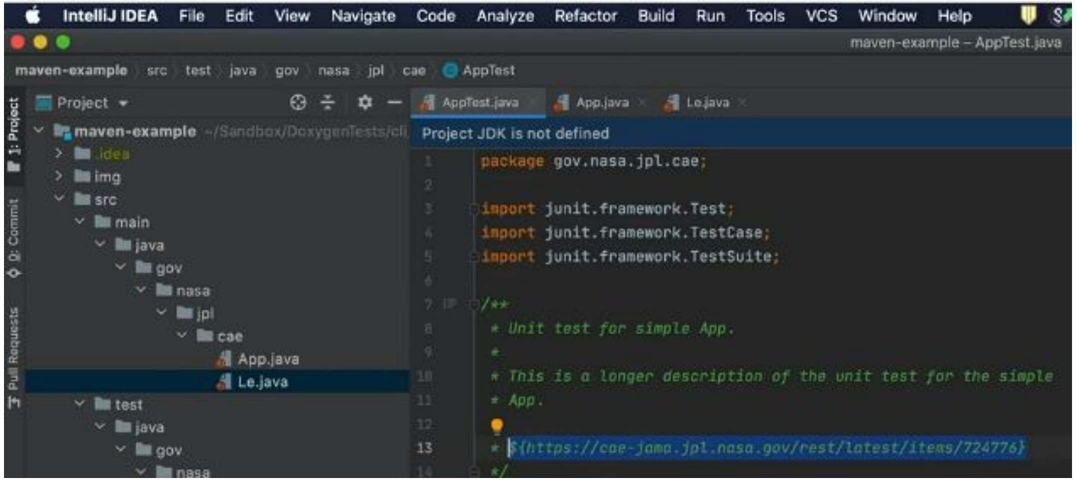


Document Generation

- **Detect reference** to requirement in source code files
- Create or update a Syndeia **relation** between the requirement and the source code file in GitHub
- **Display** the endpoints of this relation in the documentation with links to the requirement and the source code file in GitHub where it is referenced

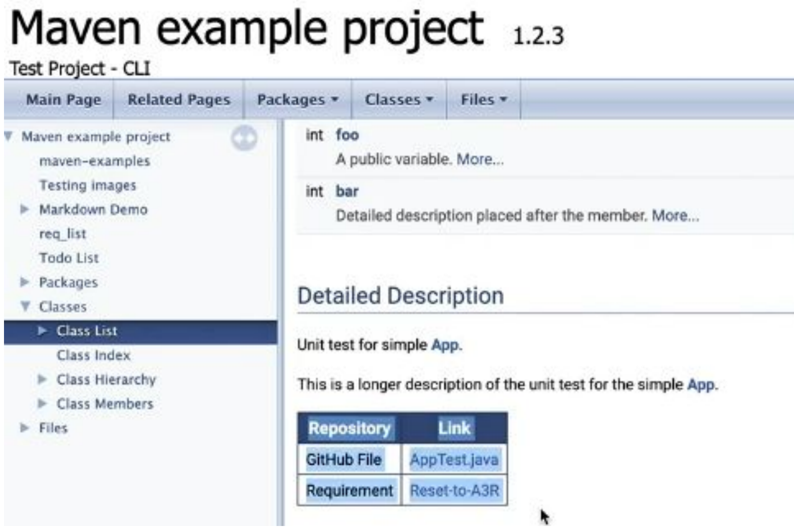


Document Generation



Auto-include links into code

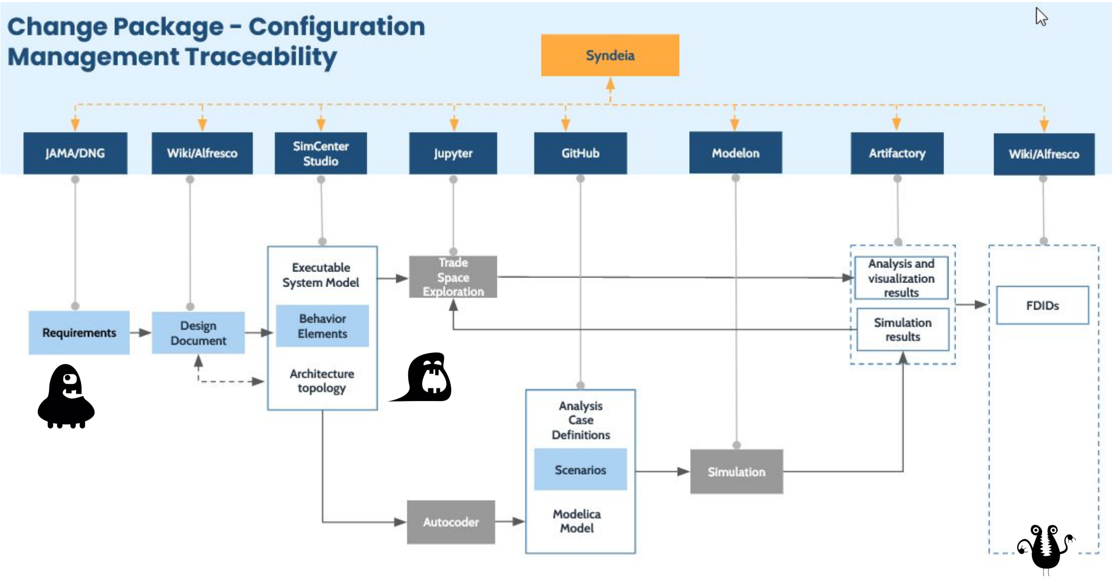
Generated linked documentation



Model Based Product Development

Model Based Product Development Pipeline

- **Cover product lifecycle** from early concept development to implementation
- Bring (executable) **systems/physics based modeling earlier** into the design
- **Evolve fidelity** of simulation as architecture matures
- Common, flexible, integrated, collaborative engineering **platform**
- **Track** parameters across product lifecycle
- Consistent set of models to **keep trade space open** longer before committing to hardware
- **Develop and validate** the architecture before buying parts
- Allow for **transition** into detailed **low level discipline specific** simulation capabilities
- Determine implementation risk



Iterative

Workflow across product lifecycle

Collaboration & Capture

Set up Architecture topology

Deconfliction

Discovery

Design & Constraint Capture

Architecture & Design

(Generative) Architecture exploration

System level & multi-physics Co-simulation

Behavioral Specifications

Variants & Configurations

(Quantitative) Analysis

Multi-attribute comparison & constraint evaluation

Visualizations (e.g. telemetry)

Behavioral Analysis

Functional & physical trade

VnV

Traceability

Parameter tracking

Metrics Reporting

Audit trails

Publication

Products

Documents

Issue Management   Continuous Integration   Process Orchestration

Process (Traceable, Auditable, Repeatable)

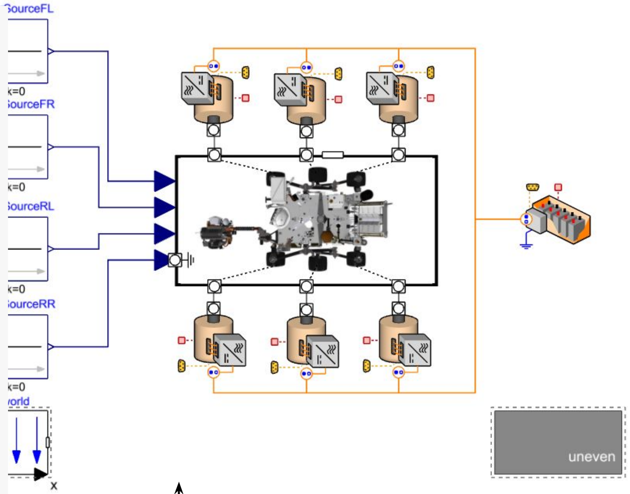
OpenCAE

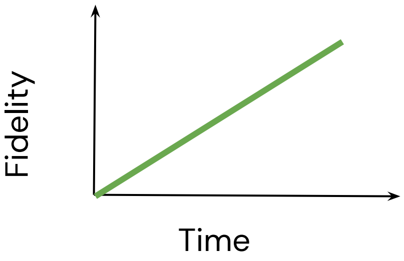
Iterative

Collaboration & Capture

```
system simple_rover:
    owns:
        Chassis
        Wheel
        PowerSupply
        Terrain

component Chassis:
    ports::group wheels:
        out::wheel_structure w1
        out::wheel_structure w2
        out::wheel_structure w3
        out::wheel_structure w4
        (out)::wheel_structure w5
        (out)::wheel_structure w6
    ports:
        out::power_structure p1
component Wheel:
    properties:
        multiplicity = 4..6
    ports:
        in::wheel_structure w1
        out::terrain t1
component PowerSupply:
    ports:
        in::power_structure p1
component Terrain:
    ports::group terrains:
        in::terrain t1
        in::terrain t2
        in::terrain t3
        in::terrain t4
        (in)::terrain t5
        (in)::terrain t6
```

System model





Multi-physics model

Pages / ... / CAE Systems Environment Wiki and Discussion

Engineering Document

Created by Robert Karban just a moment ago

This document captures the design and analysis of a rover.

MEL

Component	Mass
Chassis	500 kg
Wheel	10 kg
PowerSupply	50 kg

Architectures

Analysis of attributes

Wiki

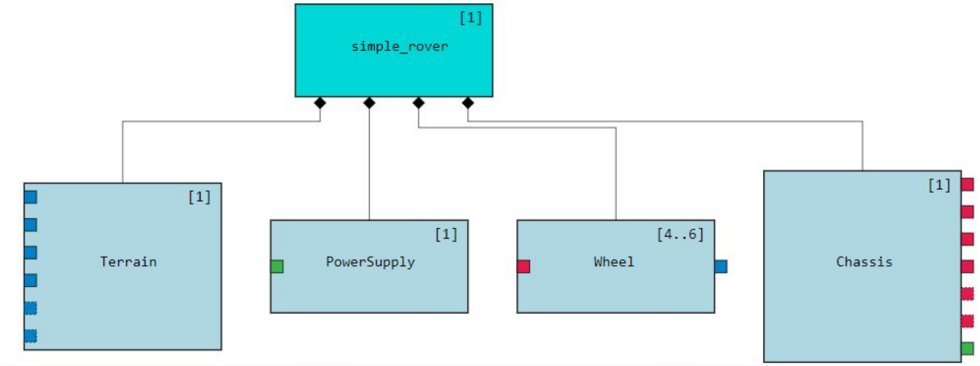
Collaboration & Capture

- Set up Architecture topology
- Deconfliction
- Discovery
- Design & Constraint Capture

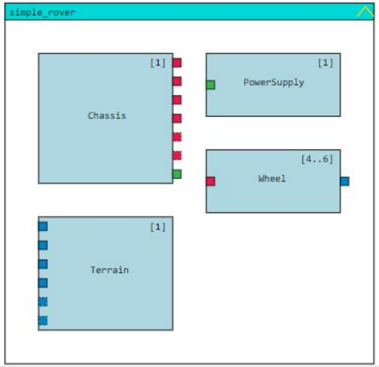
OpenCAE

Iterative

Architecture and Design

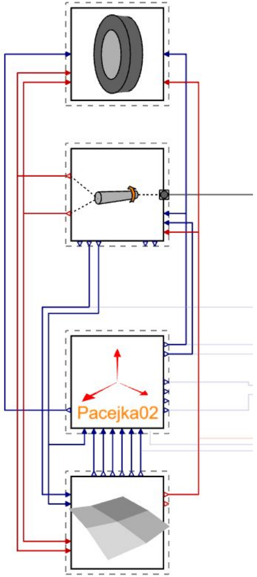



```
    out::terrain t1
component PowerSupply:
    ports:
        in::power_structure p1
component Terrain:
    ports::group terrains:
        in::terrain t1
        in::terrain t2
        in::terrain t3
        in::terrain t4
        (in)::terrain t5
        (in)::terrain t6
```




```
==============================
Run: simple_rover -
- - - - - - - - - - - - - - -
Creating ensembles...
... done in 0.136seconds.
 - 3 total ensembles
... Instantiating ensembles...
 ... 100.00% instantiated in 0.024 seconds
 - 3 ensembles
... done in 0.170 seconds, maximum tree width = 0, maximum multiplicity = 6.
- - - - - - - - - - - - - - -
Creating architectures...
 - 3 unique architectures in 3.560 seconds
... done in 3.561 seconds.
```

Project functional into physical architectures



Architecture & Design

(Generative) Architecture exploration

System level & multi-physics Co-simulation

Behavioral Specifications

Variants & Configurations

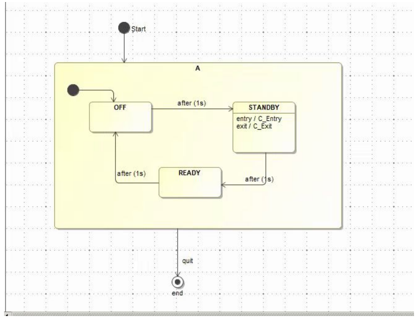
- Articulation of Design Space
- Determine required connectivity constraints

OpenCAE

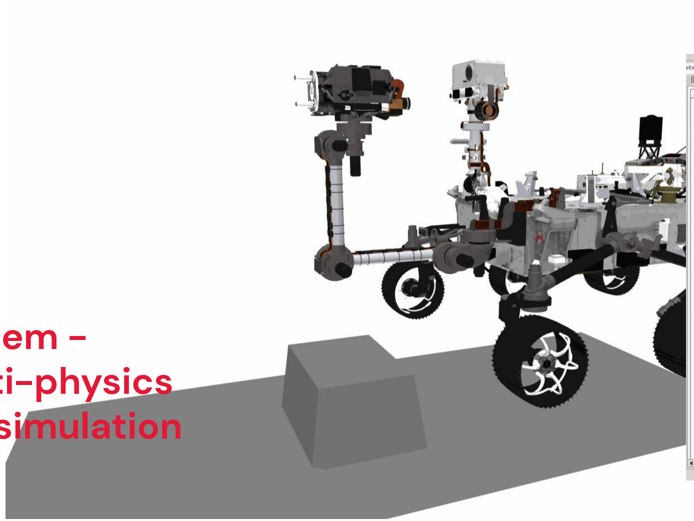
Quantitative Analysis

- Baseline Analysis
- Adding new Components
- Swapping Components

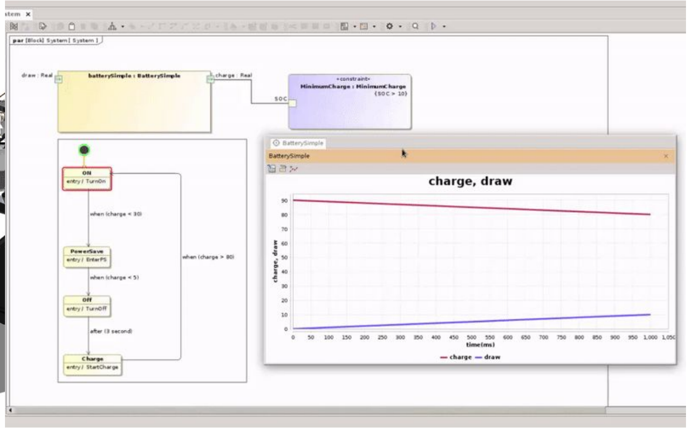
Guided Architecture Selection in an integrated engineering environment



FSW simulation



System – Multi-physics Co-simulation



Constraint evaluation

Quantitative Analysis

Multi-attribute comparison & constraint evaluation

Visualizations (e.g. telemetry)

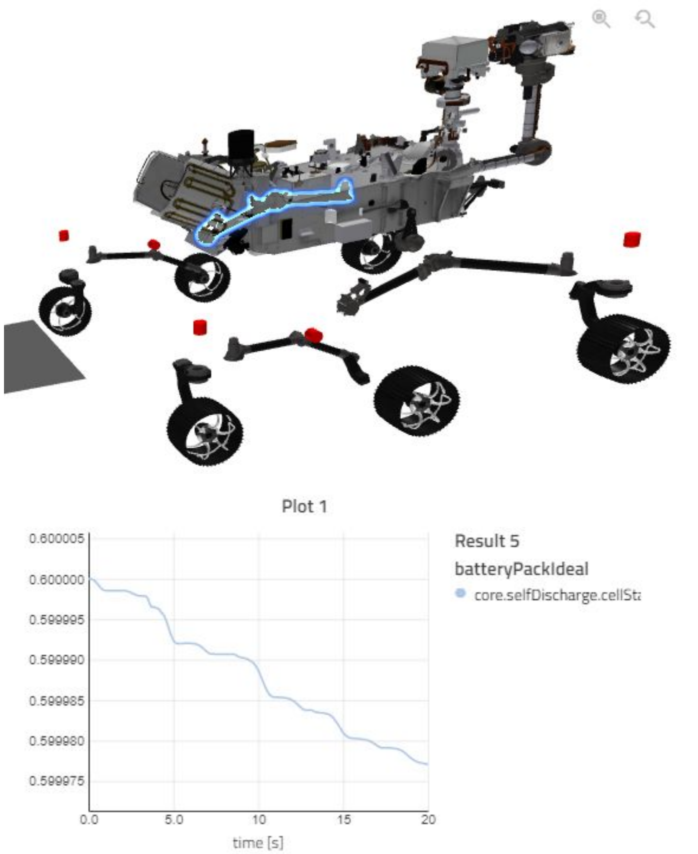
Behavioral Analysis

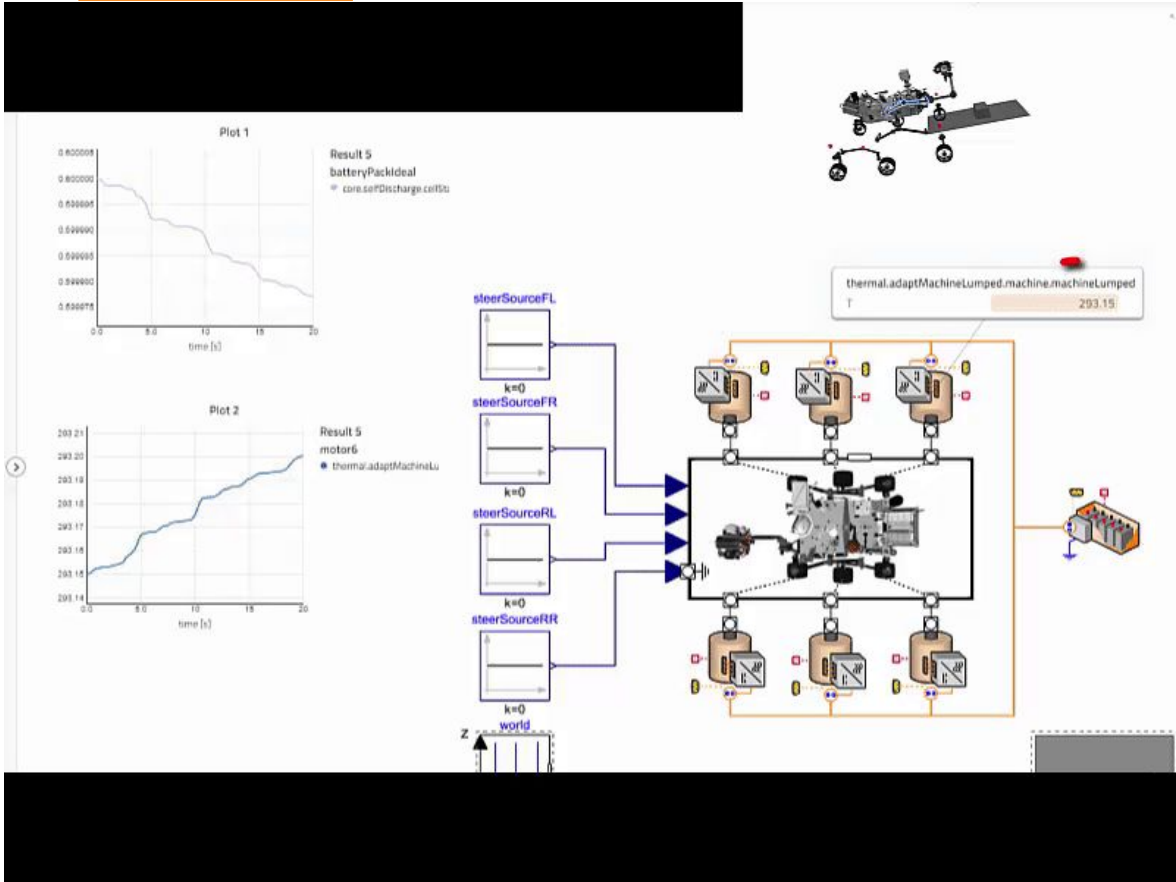
Functional & physical trade

https://github.com/Open-MBEE/perseverance-modelica

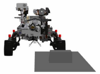
OpenCAE

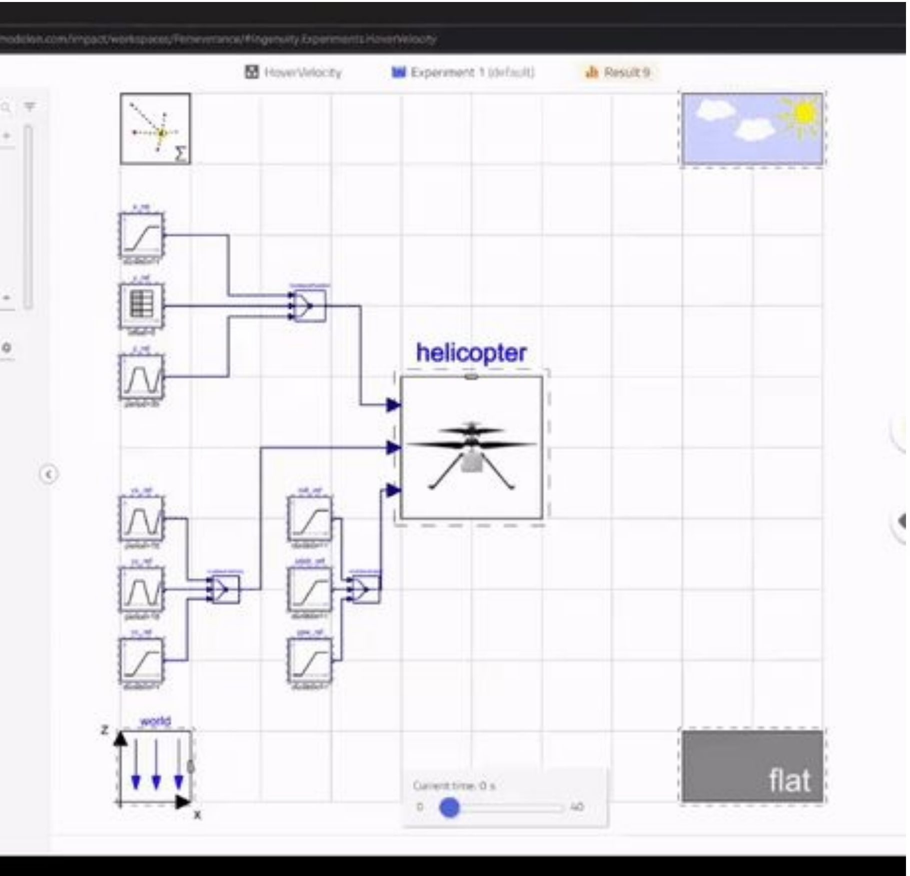
Telemetry Visualization

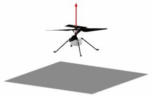




Sophisticated Simulations







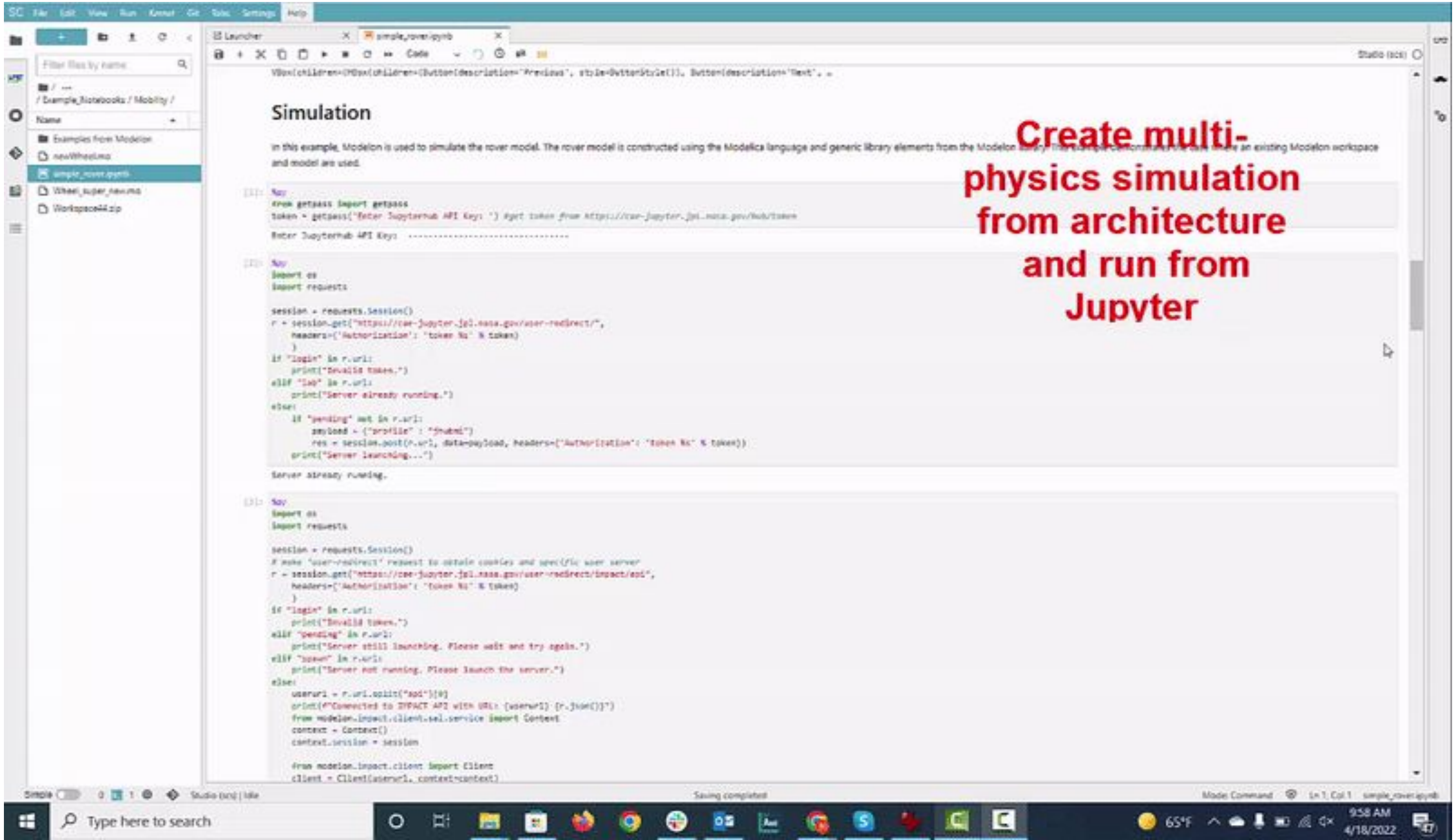
OpenCAE

Varying the physical architecture with Jupyter

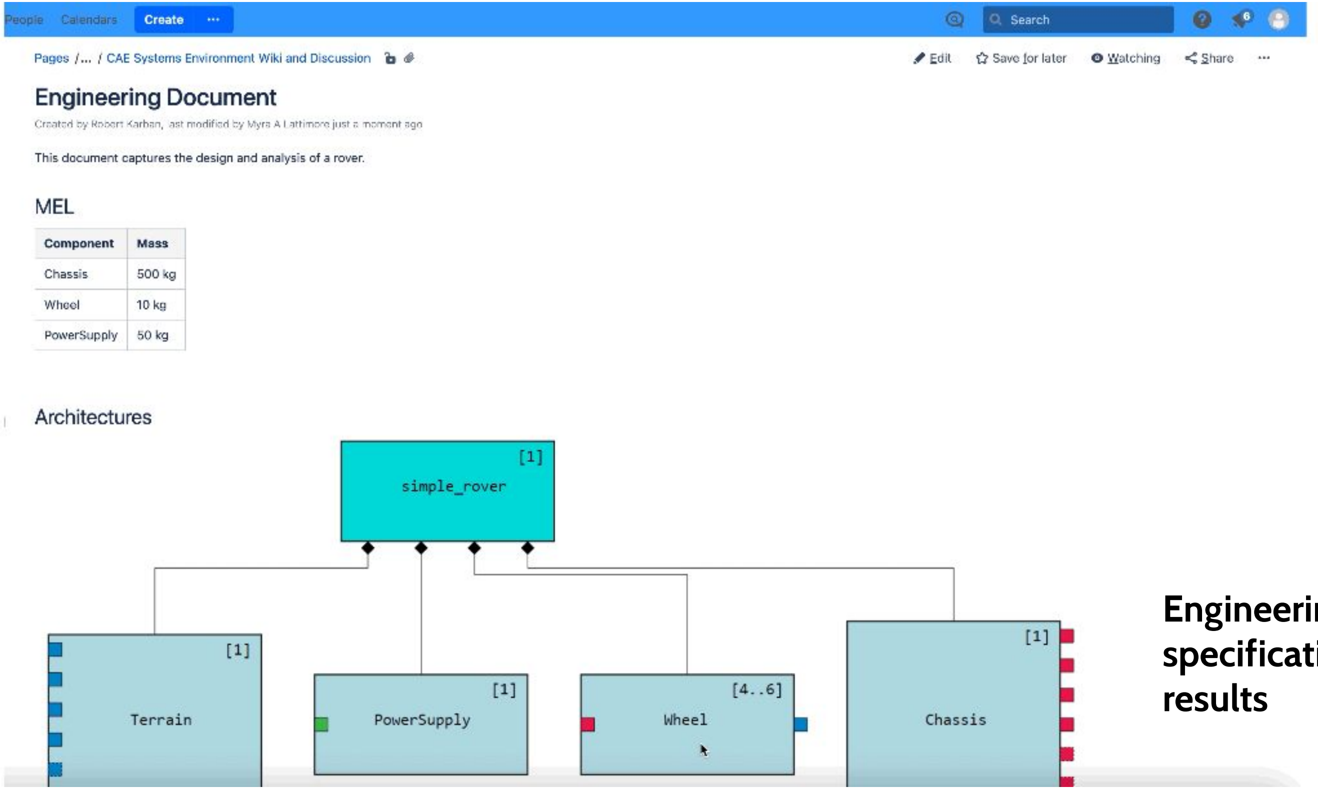
For example
Trade science return

by exploring
- Power topologies
- Sensing options
- Mass allocations

and
Evaluate against constraints



Publication



People Calendars Create ···

Search

Pages / ... / CAE Systems Environment Wiki and Discussion

Edit Save for later Watching Share ···

Engineering Document

Created by Robert Karban, last modified by Myra A Lattimore just a moment ago

This document captures the design and analysis of a rover.

MEL

Component	Mass
Chassis	500 kg
Wheel	10 kg
PowerSupply	50 kg

Architectures

Engineering documents aggregate design specification, definition, and analysis results

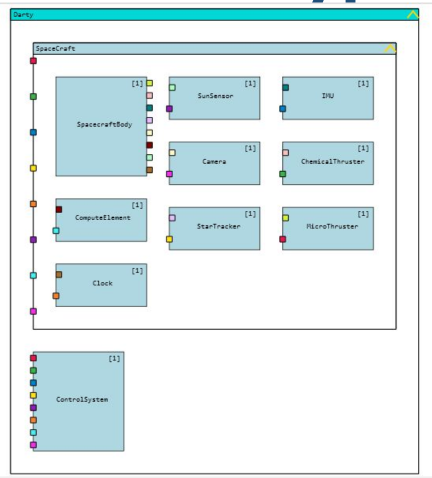
OpenCAE

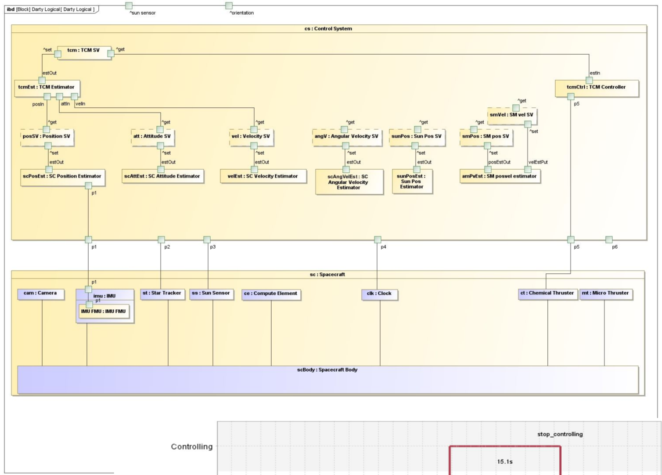
Autonomy Trades Prototype with “Darty”

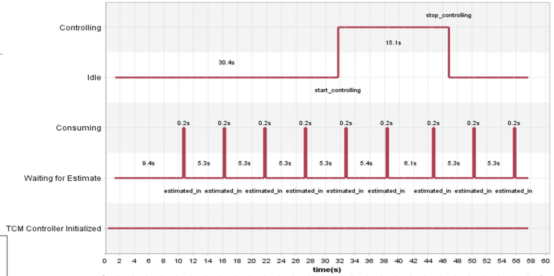
- Prototype demonstrating the process with a DART like mission – crash into a small body
- Explore trade space to determine if autonomy is needed and which architectures require it

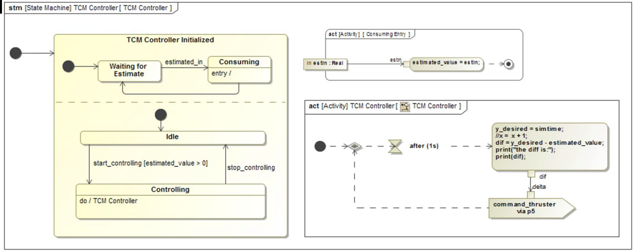
Steps:

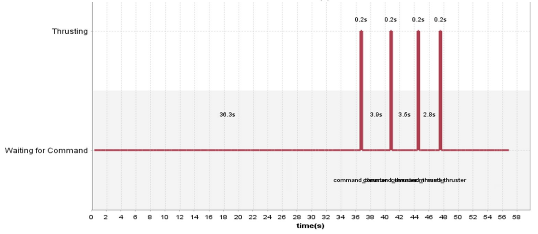
- Articulate architecture space with different types of thrusters
- Generate architecture structures as SysML models
- Elaborate required control system behavior with a goal-based architecture in an executable SysML model
- Capture multi-physics model of spacecraft in Modelica
- Co-simulate System level behavior and multi-physics with different scenarios
- Determine when trajectory correction maneuvers require autonomy



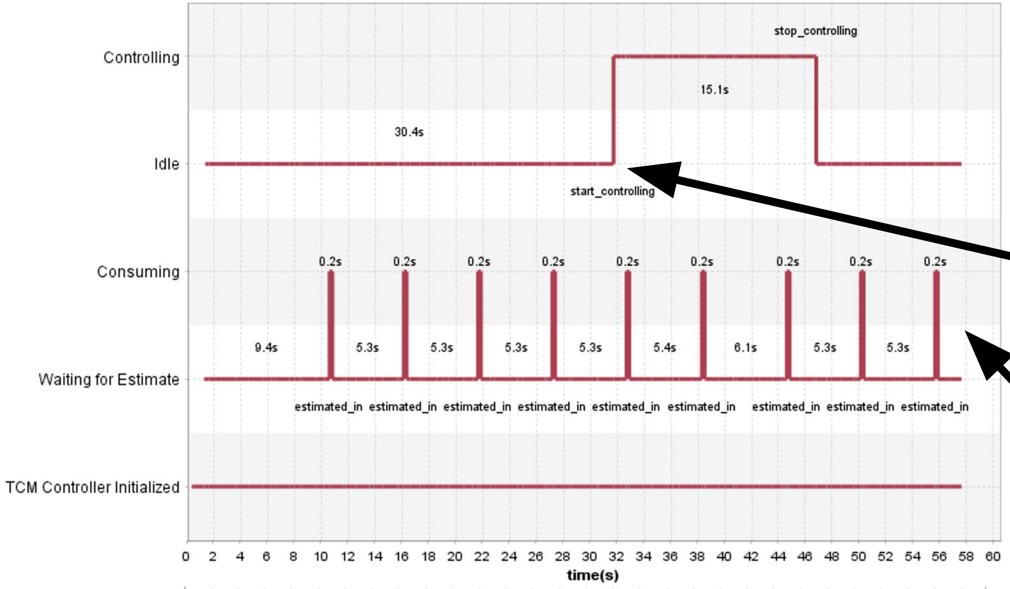






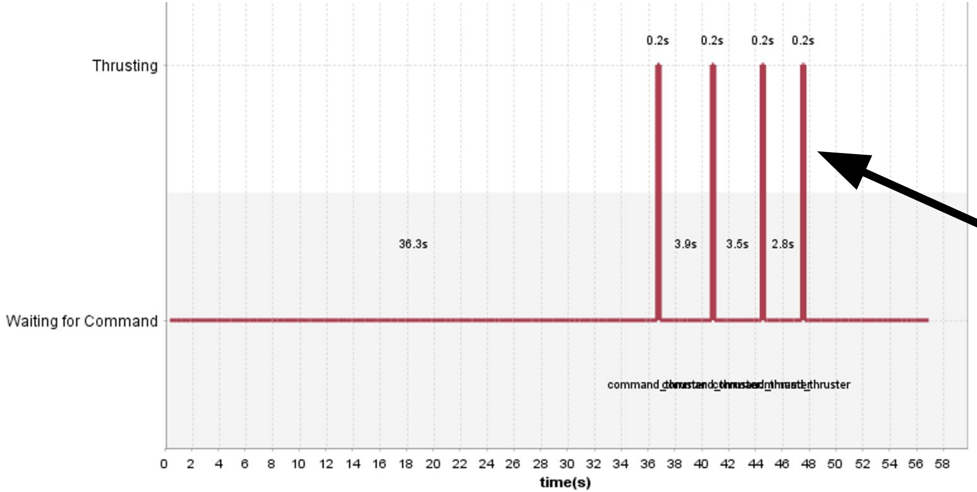


Simulation Results – State Machines



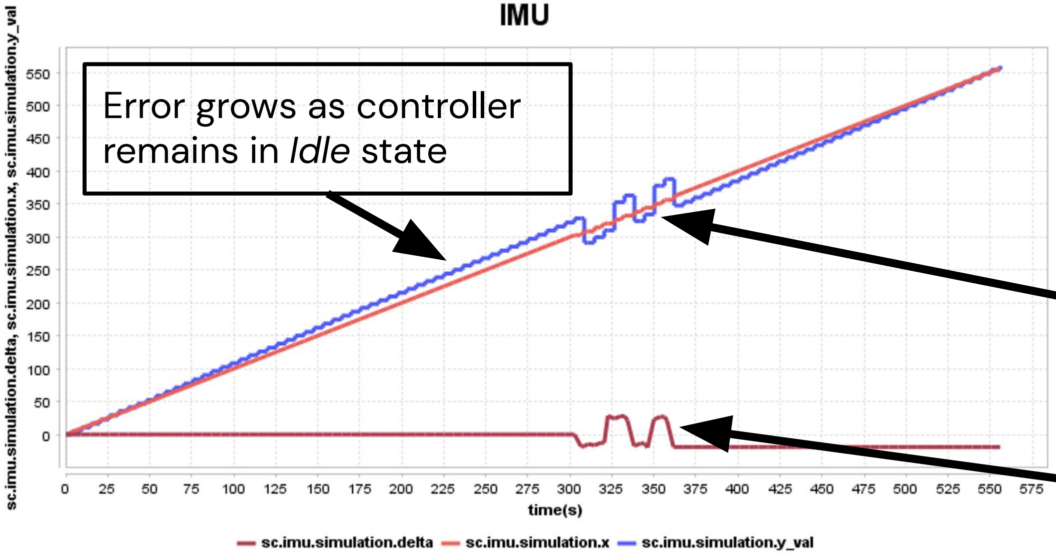
Control is applied upon receipt of the *start_controlling* signal

Controller is always consuming new measurements as they arrive from the the estimator



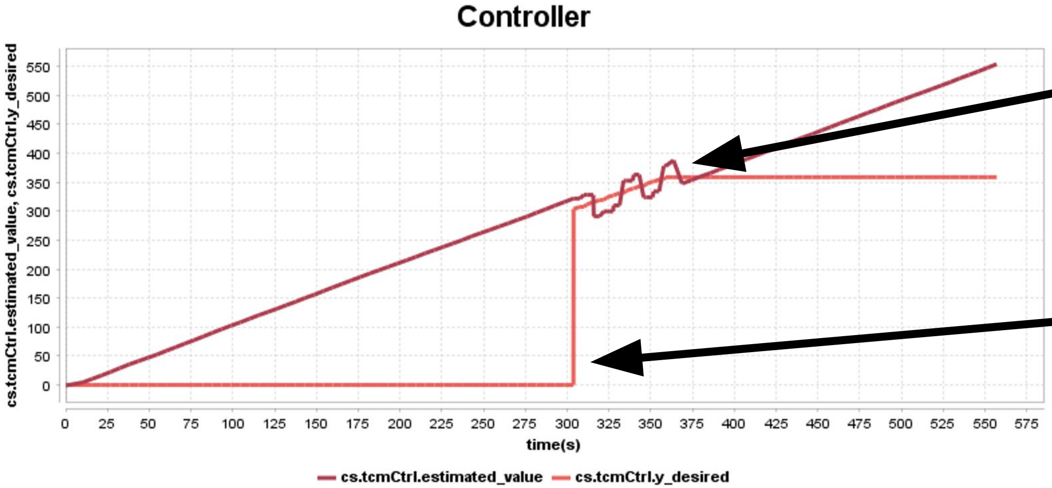
Thruster is active when controller is in the *Controlling* state and provides *command_thruster* signal

Simulation Results – Value properties



Measured value responds to changes induced by the thruster [low fidelity sim produces oscillations around desired value]

Correction delta is applied by thruster when system is in the *Controlling* and *Thrusting* states



Estimated value responds to changes induced by the thruster when in *Thrusting* state.

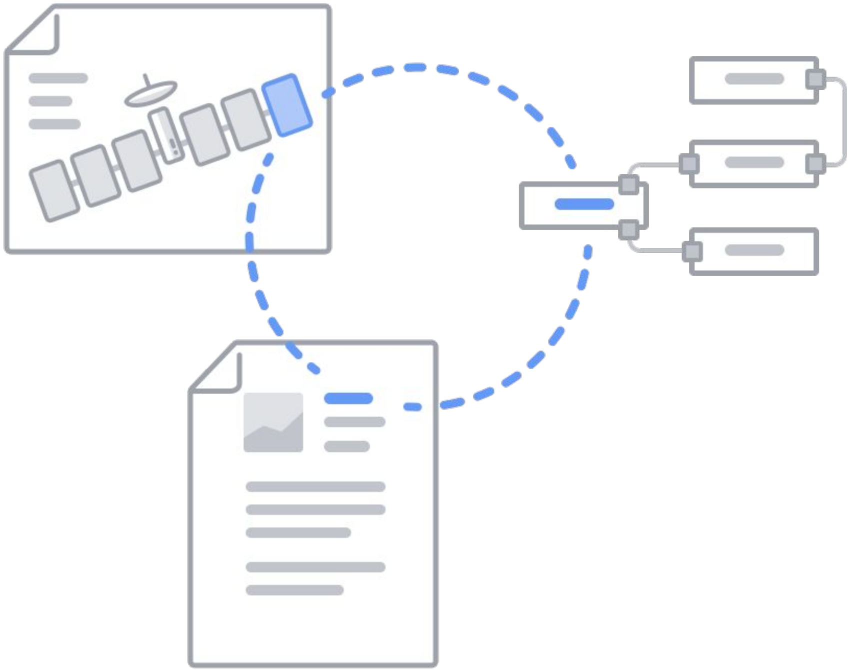
Initially, controller is in *Idle* state so no desired value is calculated. Desired value is calculated when controller is in *Controlling* state. Desired value remains unchanged after controller returns to *Idle*.

Topic: Enabling Technology

We are almost there

Open Model-Based Engineering Environment

- OpenMBEE is a **community** for open source modeling software and models
  - Open source software activities
  - Open source models
  - Open source exchange of ideas
- Participants and adopters:
  JPL, Boeing, Lockheed Martin, OMG, NavAir, Ford, Stevens, Georgia Tech, ESO, ...
- > 500 members



Linked Data Documents with OpenMBEE

Ecosystem Vision

- Augment Jupyter's **multi-language analysis** capabilities with modeling and connected engineering
- Enable novel data-driven analyses with advanced capabilities, as a **service**
- Unlock value through **commoditization**
- **Standards** powered engineering **platform** using SysML v2 + API & Services

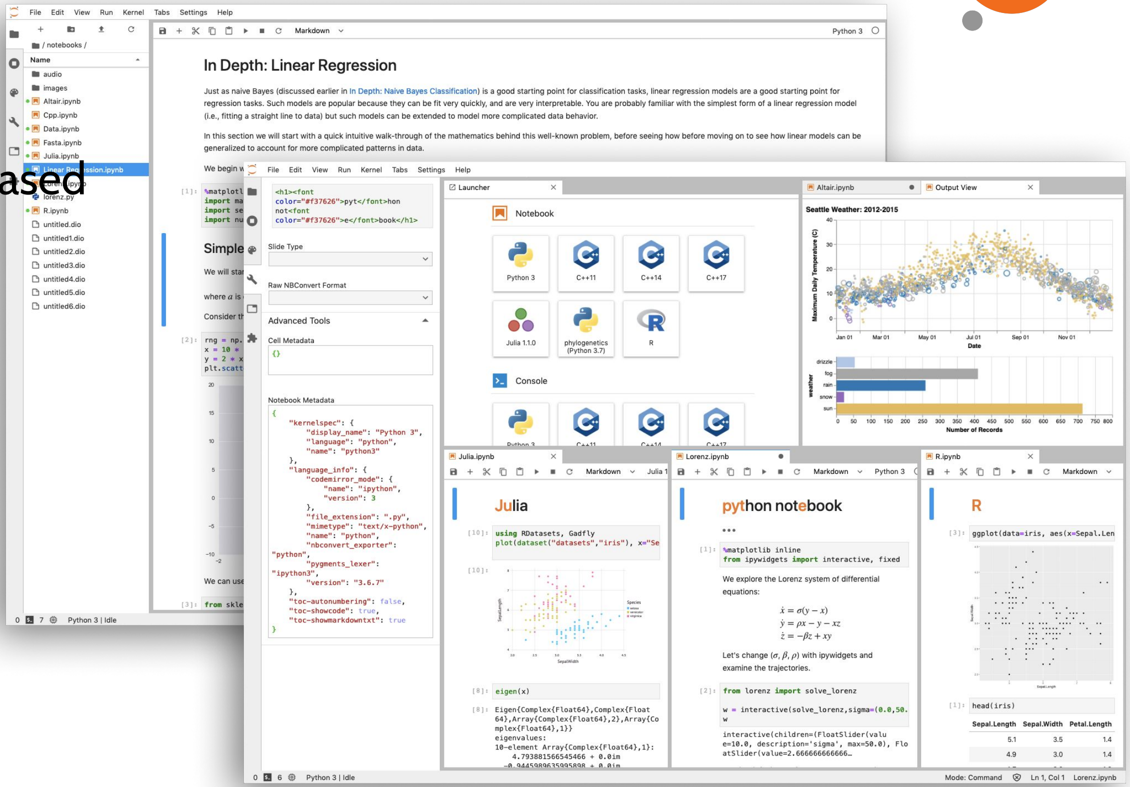
J

Global Engineering Ecosystem

Jupyter as Analysis and Visualization hub for Models

Interactive, exploratory, browser-based computing environment for:

- engineering
- data science
- scientific computing
- ML/AI
- and so much more...



SysML v2 Key Elements

- New metamodel that is not constrained by UML
  - Grounded in formal semantics
- Robust visualizations based on flexible view & viewpoint specification and execution
  - Equivalent textual and graphical
- Standardized API to access the model

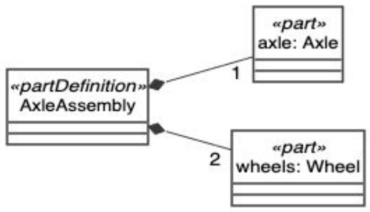
In [1]:

```
import ISQ::TorqueValue;

part def Axle;

part def Wheel {
    part lugbolt {
        attribute torque : TorqueValue;
    }
}

part def AxleAssembly {
    part axle : Axle[1];
    part wheels : Wheel[2];

    connection : Mounting connect axle to wheels;
}

connection def Mounting {
    end axle: Axle[1];
    end wheel: Wheel[*];
}

attribute def FuelCmd;

action def providePower (
    in fuelCmd : FuelCmd,
    out wheelTorque : TorqueValue[2]
);

part axle : AxleAssembly;
```

In [2]:

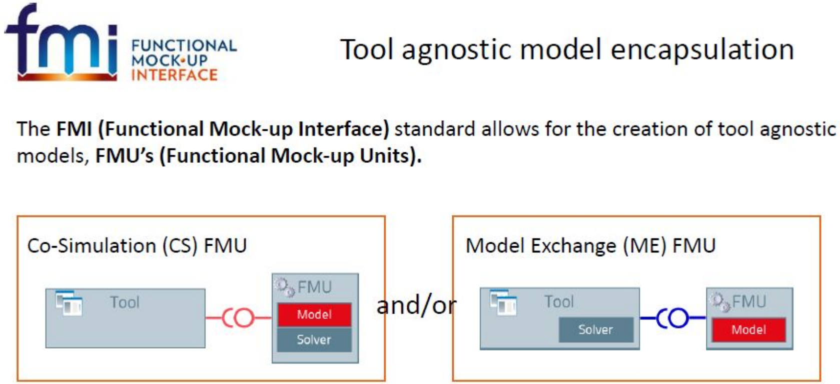
```
%viz AxleAssembly --style lr
```

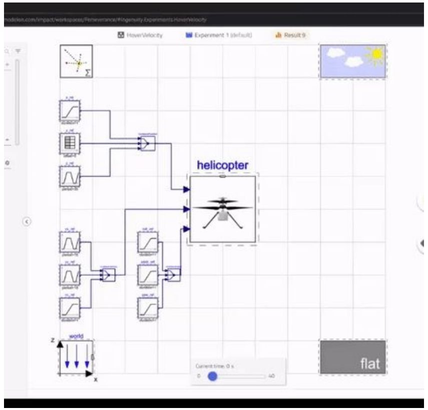
Out[2]:



Functional Mock Up Interface

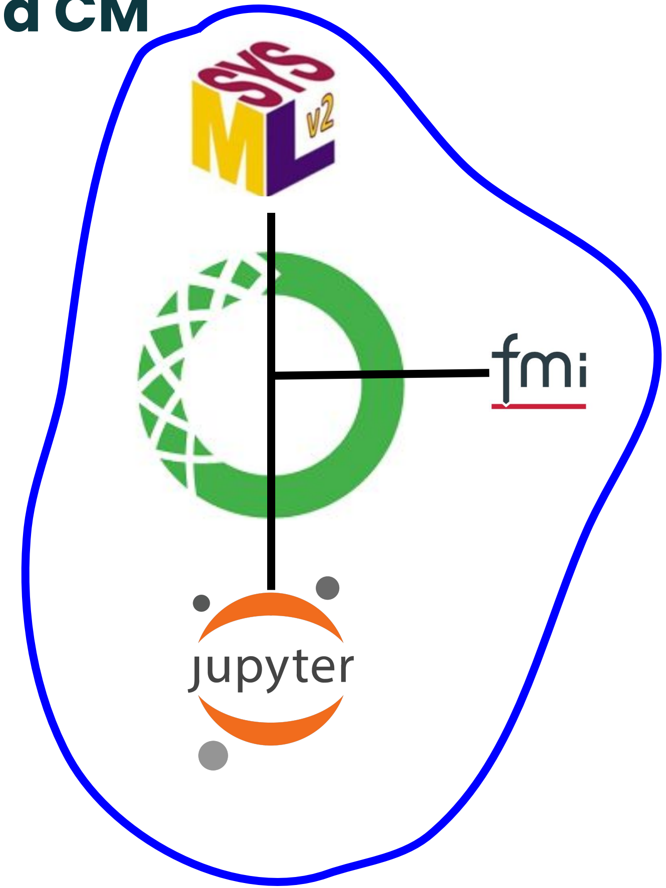
- Supported by more than 100+ tools (https://fmi-standard.org/)
- Custom IP protection
- Cost-effective deployment
- Compiled models
- Parameters can be changed
- Structure cannot be changed





Conda enables packaging of and CM

- Package and **dependency management** for models
- Enable sharing **distribution** of models as self-contained packages
- Model **re-use**



The Next Generation Systems Engineer’s Dream Car

Conclusions

- SE paradigm shift to formal languages and automation, i.e. MBSE
- Systematic qualification and audit trail
- Close gap between engineering documents and models
- Break up the engineering silos with digital twin pipelines
- Standards based and commoditized

Questions?

OpenCAE

Backup