Flywheel

The Biomedical Research Data Platform
Powering Healthcare Innovation

# The HPC-client:
# A Pathway from POC to Production

The Craftsmanship of Code

Copyright Flywheel Exchange, LLC 2022

# Outline

- What is the HPC-Client?
- What Was Done
  - POC -> Deployable Product
- Intermission
  - Live Demo (Jesus)
  - Questions
- How it Was Done
  - Preliminaries
  - Foundations
  - The Feature Cycle
- Final Words

Flywheel

## What is the "HPC-Client"?

- Definition
- Workflow

## The "HPC Client" is...

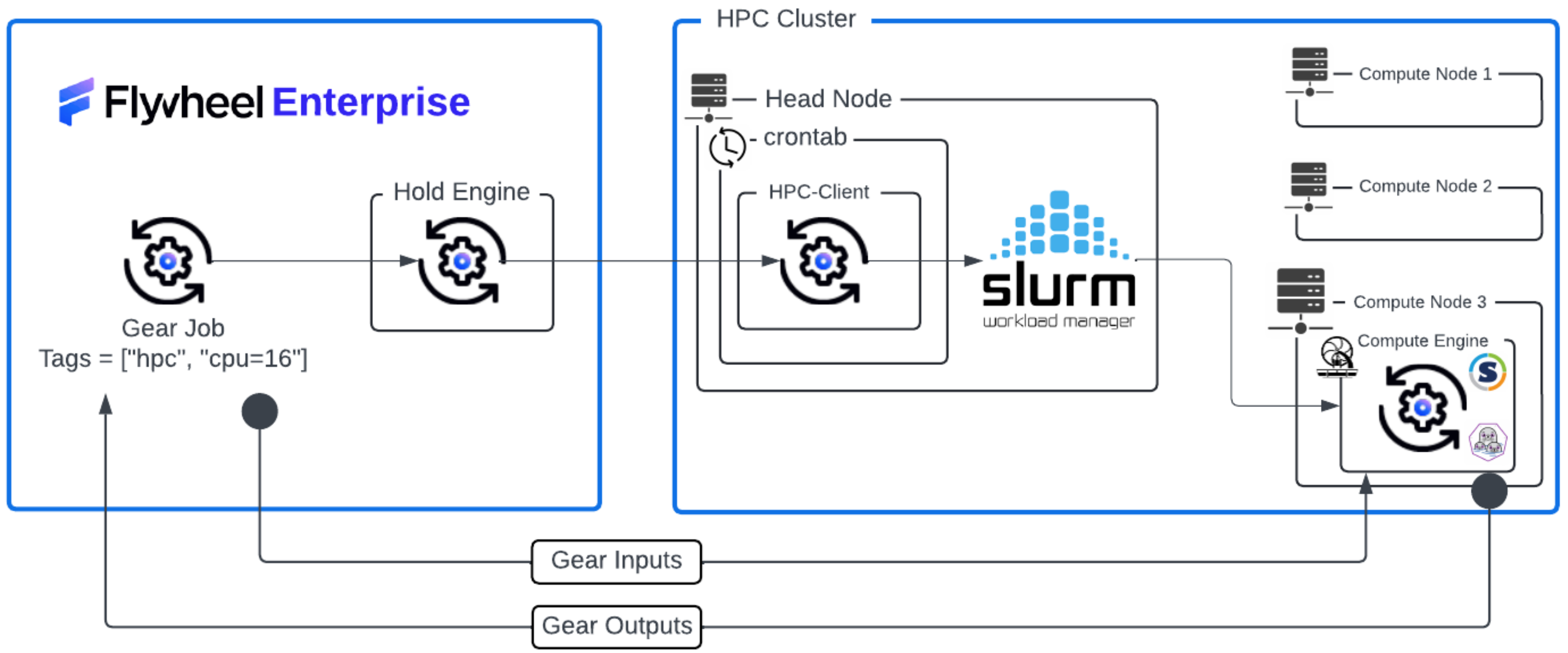
The HPC Client is an orchestration application that enables Flywheel Gears to be scheduled and executed on the nodes of a High-Performance Computing cluster.

The HPC Client enables specifying required resources (e.g. RAM, CPUs, GPU) of the execution node at gear launch. Default resource requirements are defined in the HPC Client configuration.

Flywheel

# HPC Client Workflow

An Over-Simplified View



## POC -> Deployable Product

- From There...
- To Here...

# Starting with...

A Proof-of-Concept

- GitHub Repository
- "Installed" with `git clone …`
- Pipenv dependency management
- 34% test coverage
- 0% docstring coverage
- Bash script
  - Setup/Reset
  - Code Formatting
- Flywheel-supported configuration process
- Detailed markdown instructions

```
hpc-client
├── doc/
├── examples
│   └── settings
│       ├── cast.yml
│       ├── credentials.sh
│       └── start-cast.sh
├── process
│   ├── check-syntax.sh
│   ├── factory-reset.sh
│   └── setup.sh
├── src
│   ├── cluster/
│   ├── util/
│   ├── __init__.py
│   ├── cast.py
│   ├── Pipfile
│   └── Pipfile.lock
├── tests/
├── installer.sh
└── readme.md
```

# Ending with…

## Foundations

- GitLab Repository
- Pip Installed and updated
  - `pip install fw-hpc-client`
  - option for stand-alone Linux executable
- Poetry dependency management
- 100% unit-test coverage
- 100% docstring coverage
- Self-service Python-integrated setup
  - Guided setup interview (***in-review***)
- CI/CD and pre-commit hooks
- Markdown instructions rendered with mkdocs
- Updates and documentation of the **engine code**

```
fw-hpc-client
├── docs/
├── examples
│   └── settings
│       ├── cast.yml
│       ├── credentials.sh
│       └── start-cast.sh
├── fw_hpc_client
│   ├── assets/
│   ├── cluster/
│   ├── interview_templates/
│   ├── util/
│   └── cast.py
├── tests/
├── .gitlab-ci.yml
├── .pre-commit.yaml
├── mkdocs.yml
├── pyproject.toml
└── README.md
```

# Ending with...

## Customer-Driven Features

- Slurm-Scheduled GPU job submission
    - Singularity (CIBM-confirmed!)
    - Podman (***in-review)***
- Map Flywheel Job Priority to Slurm Priority
- Tag-driven Slurm CPU/RAM setting
- Tag-driven submission to specific HPC Cluster

Attempted:

- Schedule Slurm Job by Posix-User
    - Requires Core-API changes for API-Key permissions

      **OR**
    - Every user reads the SCITRAN_CORE_DRONE_SECRET

```
fw-hpc-client
├── docs/
├── examples
│   └── settings
│       ├── cast.yml
│       ├── credentials.sh
│       └── start-cast.sh
├── fw_hpc_client
│   ├── assets/
│   ├── cluster/
│   ├── interview_templates/
│   ├── util/
│   └── cast.py
├── tests/
├── .gitlab-ci.yml
├── .pre-commit.yaml
├── mkdocs.yml
├── pyproject.toml
└── README.md
```

Flywheel

# Intermission

- Live Demo (Jesus)
- Questions

Flywheel

## Live Demo (Jesus)

- Install
- Basic Setup
- Configuration
- Execution

Flywheel

## Questions

On the Story So Far?

Questions?

Flywheel

## How It Was Done

More Method than Madness

- Preliminaries
- Foundations
- Feature Cycle

Flywheel

## Preliminaries

- Project Setup
- Note-taking
- Read the Docs
- Kick the Tires

Flywheel

# Project Setup

Structuring my projects helps with coordinating

- Multiple Related Repositories (**Code**)
- Project-Specific **Data**
- **Downloads**
- Notes (**Docs**)

Project Creation Script

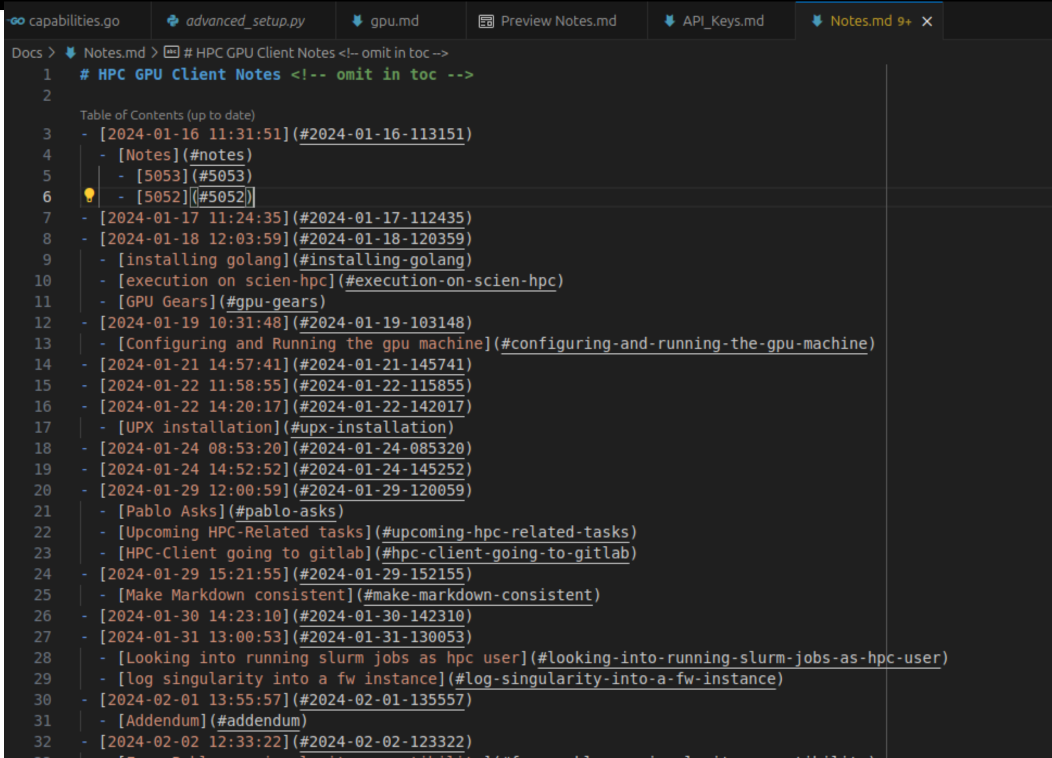
```
2024.04.15.INCLUDE.Phase.II
├── Code
│   └── reference_repositories/
│   └── active_repositories/
│   └── experiments/
├── Data/
├── Docs
│   └── Notes.md
└── Downloads/
```

# Note-Taking

## The Art of Progressive Summarization

Notes in Markdown is a ***Game-Changer***.

- Automated Table of Contents (code_ext)
- Date-String Headers (code_ext)
- Capturing Code Snippets in Code Blocks
- Links to project-relevant articles
  - Better than 3-dozen open tabs
- Searchable from the command line
  - `rg "fold " –g "*.md"`
- "Distillable" into end-user documentation
- My project "Capture" system
  - See "Getting Things Done"


capabilities.go
advanced_setup.py
gpu.md
Preview Notes.md
API_Keys.md
Notes.md 9+
Docs > Notes.md > # HPC GPU Client Notes <!-- omit in toc -->
Table of Contents (up to date)

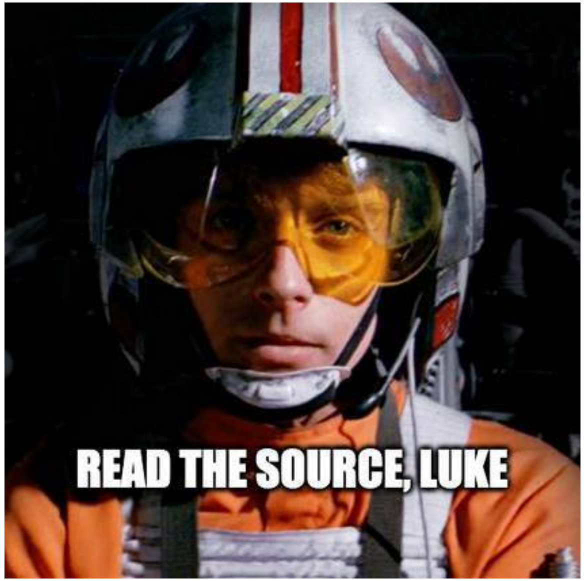
```
# HPC GPU Client Notes <!-- omit in toc -->

- [2024-01-16 11:31:51](#2024-01-16-113151)
  - [Notes](#notes)
    - [5053](#5053)
    - [5052](#5052)
- [2024-01-17 11:24:35](#2024-01-17-112435)
- [2024-01-18 12:03:59](#2024-01-18-120359)
  - [installing golang](#installing-golang)
  - [execution on scien-hpc](#execution-on-scien-hpc)
  - [GPU Gears](#gpu-gears)
- [2024-01-19 10:31:48](#2024-01-19-103148)
  - [Configuring and Running the gpu machine](#configuring-and-running-the-gpu-machine)
- [2024-01-21 14:57:41](#2024-01-21-145741)
- [2024-01-22 11:58:55](#2024-01-22-115855)
- [2024-01-22 14:20:17](#2024-01-22-142017)
  - [UPX installation](#upx-installation)
- [2024-01-24 08:53:20](#2024-01-24-085320)
- [2024-01-24 14:52:52](#2024-01-24-145252)
- [2024-01-29 12:00:59](#2024-01-29-120059)
  - [Pablo Asks](#pablo-asks)
  - [Upcoming HPC-Related tasks](#upcoming-hpc-related-tasks)
  - [HPC-Client going to gitlab](#hpc-client-going-to-gitlab)
- [2024-01-29 15:21:55](#2024-01-29-152155)
  - [Make Markdown consistent](#make-markdown-consistent)
- [2024-01-30 14:23:10](#2024-01-30-142310)
- [2024-01-31 13:00:53](#2024-01-31-130053)
  - [Looking into running slurm jobs as hpc user](#looking-into-running-slurm-jobs-as-hpc-user)
  - [log singularity into a fw instance](#log-singularity-into-a-fw-instance)
- [2024-02-01 13:55:57](#2024-02-01-135557)
  - [Addendum](#addendum)
- [2024-02-02 12:33:22](#2024-02-02-123322)
```

# Read the Docs

[Read the Source, Luke!]

- Familiarize yourself with the documentation
- Familiarize yourself with the codebase
- Take copious notes

  (***No, you can't keep it "all in your head"!!!***)
  - How you think it works (best guesses are OK!)
  - What looks funny/weird/non-compliant
  - What you would add/change
    - "- [  ]" This is a checkbox in MD
    - It is a "searchable marker"
- Don't try to understand too much
  - This will come with time
- CHANGE NOTHING!!!



Flywheel

# Kick The Tires

## Yes, Try to BREAK It!!!

Take the existing code for a test drive.

Take notes on:

- What works
- What doesn't
- What's clunky
- What BREAKS!!!
    - What you might do to prevent it from breaking
- What you would improve
- Again, CHANGE NOTHING!!!

Flywheel

# Foundations

- Setup Automation
- Linting
- Docstrings
- Debugging
- Unit Tests

Flywheel

## Setup Automation

Pre-commit and CI/CD are your Friends...Really!!!

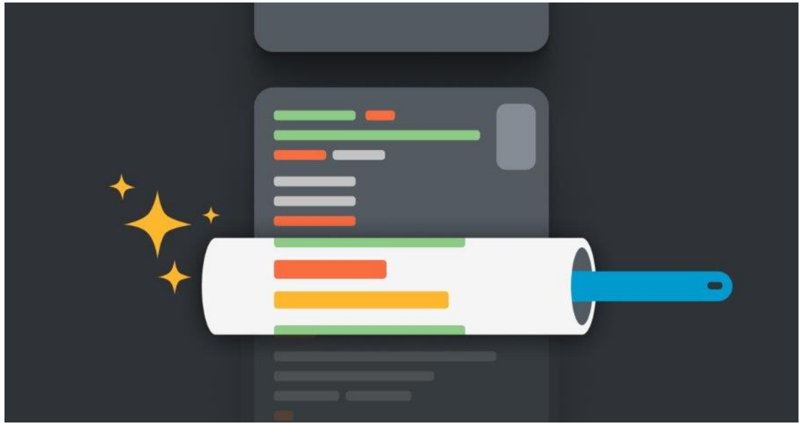
Introduce "Bare Bones"

- **.pre-commit-config.yaml**
  - Introduce a hook per MR
  - *poetry_export/requirements.txt* may be needed for some hooks
- **.gitlab-ci.yml**
  - You may need to install packages
    - APT_INSTALL
    - PIP_INSTALL
- **pyproject.toml**
  - The above tooling works best with poetry

# Linting

Increasing Readability

- Markdown Linting
  - Enable *markdownlint* pre-commit hook
  - Use IDE Linters (code_ext)
  - fold -w 88 -s Markdown.md
  - Rewrap (code_ext)
  - Merge Request
- Yaml Linting
  - Enable *yamllint* pre-commit hook
  - Resolve violations
  - Merge Request

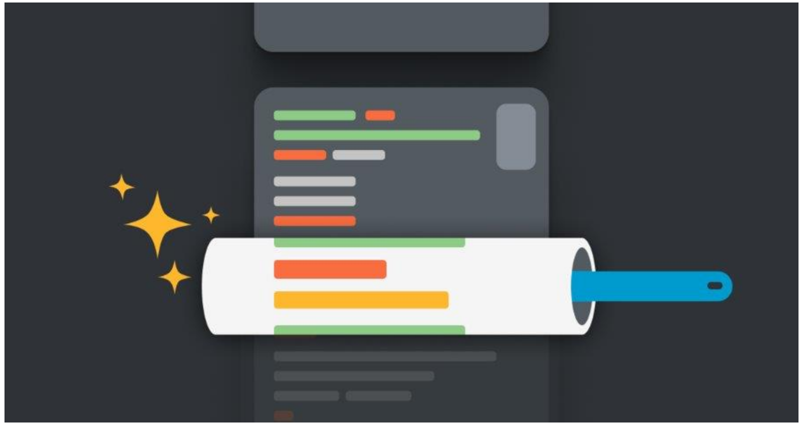


Flywheel

# Linting

Increasing Readability

- Python Linting
  - Visual Linting (code_ext)
  - isort (code_ext)
    - See extension for vscode settings
  - Commands:
    - `isort –profile black .`
    - `black .`
  - vscode settings
    - "editor.rulers":[88]
- Enable *ruff_tests* hook
- Merge Request



Flywheel

## Docstrings

...and comments promote insight and understanding

- Give Every Function/Method a Docstring
  - AutoDocstring (code_ext)
  - Copilot*
  - ChatGPT*
- Add comments for
  - Clarity
  - To call attention (# NOTE: …)
  - To mark a TODO (# TODO: …)
    - "Distill" to a  "TODO.md"
  - Other "Codetags"
- Merge Request

* depending on sensitivity

ALWAYS SPRINKLE ***Comments*** in your Code

Reference

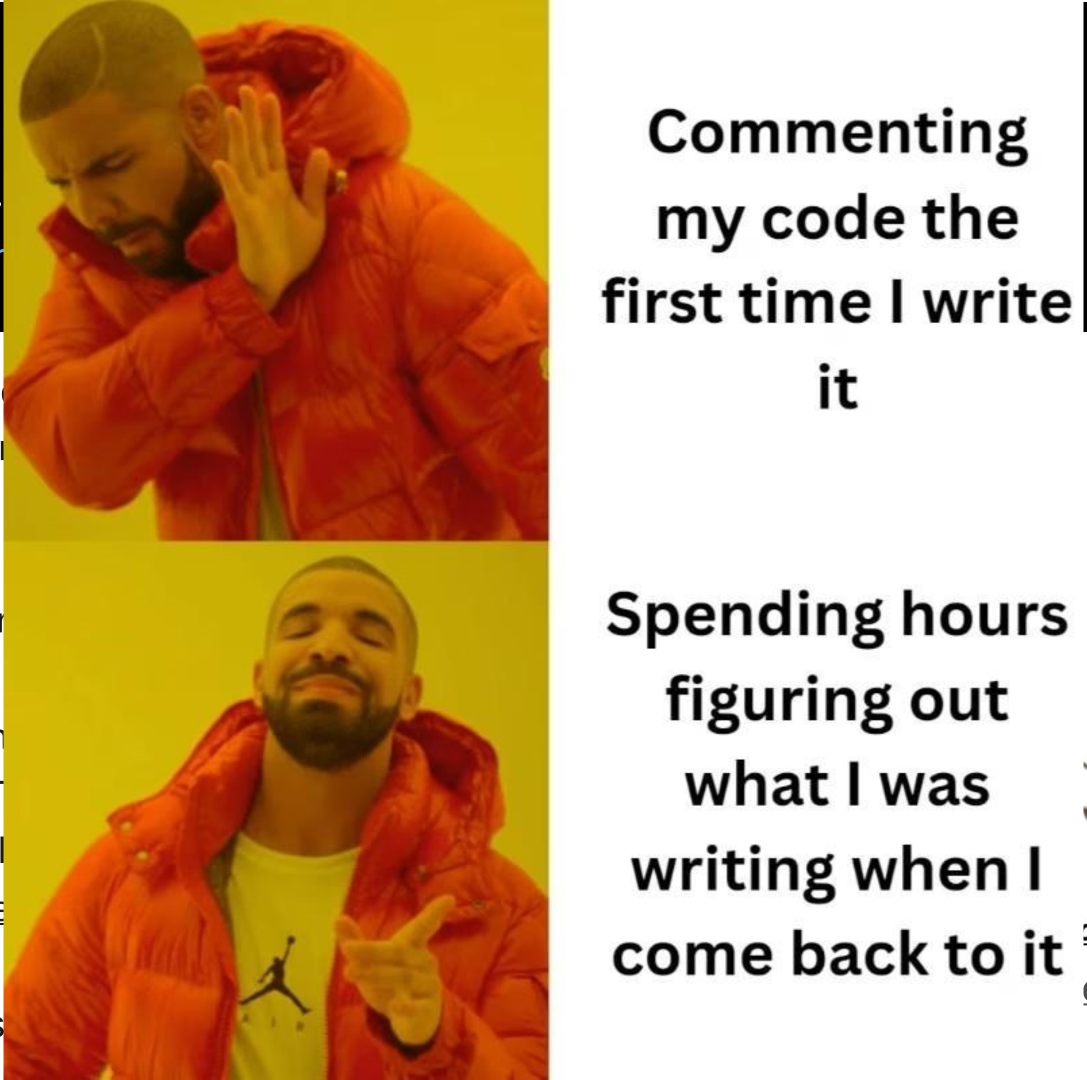
Flywheel

# Docstri

...and commen

- Give Every Functi
  - AutoDocstri
  - Copilot*
  - ChatGPT*
- Add comments fo
  - Clarity
  - To call atten
  - To mark a T
    - "Distill
  - Other "Code
- Merge Request

* depending on sens


Commenting my code the first time I write it
Spending hours figuring out what I was writing when I come back to it

*omments* in your Code

e

Flywheel

# Debugging

## No Substitute

- Stepping through code is essential for understanding
- `launch.json` configuration (vscode) allows for
    - Command-line arguments
    - Environment variables
- "Debug Console"/"Watch Panes" allows for variable observation and manipulation

Flywheel

# Unit Testing

## Patching in Layers

- Each module (e.g. *my_python_file.py*) gets its own test file (e.g. *test_my_python_file.py*).
- Each function gets its own tests.
    - Patch referenced functions or built-ins
        - Test those functions separately!!!
    - Give representative data/Mocks for inputs
    - Execute function
    - Assertions
    - Teardown/Reset Mocks
- Use classes to setup and test classes
- Check on coverage regularly
    - `pytest --cov=<package> --cov-report=term-missing .`
- Multiple MRs—depending on size
    - Update PYTEST_COV_FAIL_UNDER accordingly (.gitlab-ci.yml)

Flywheel

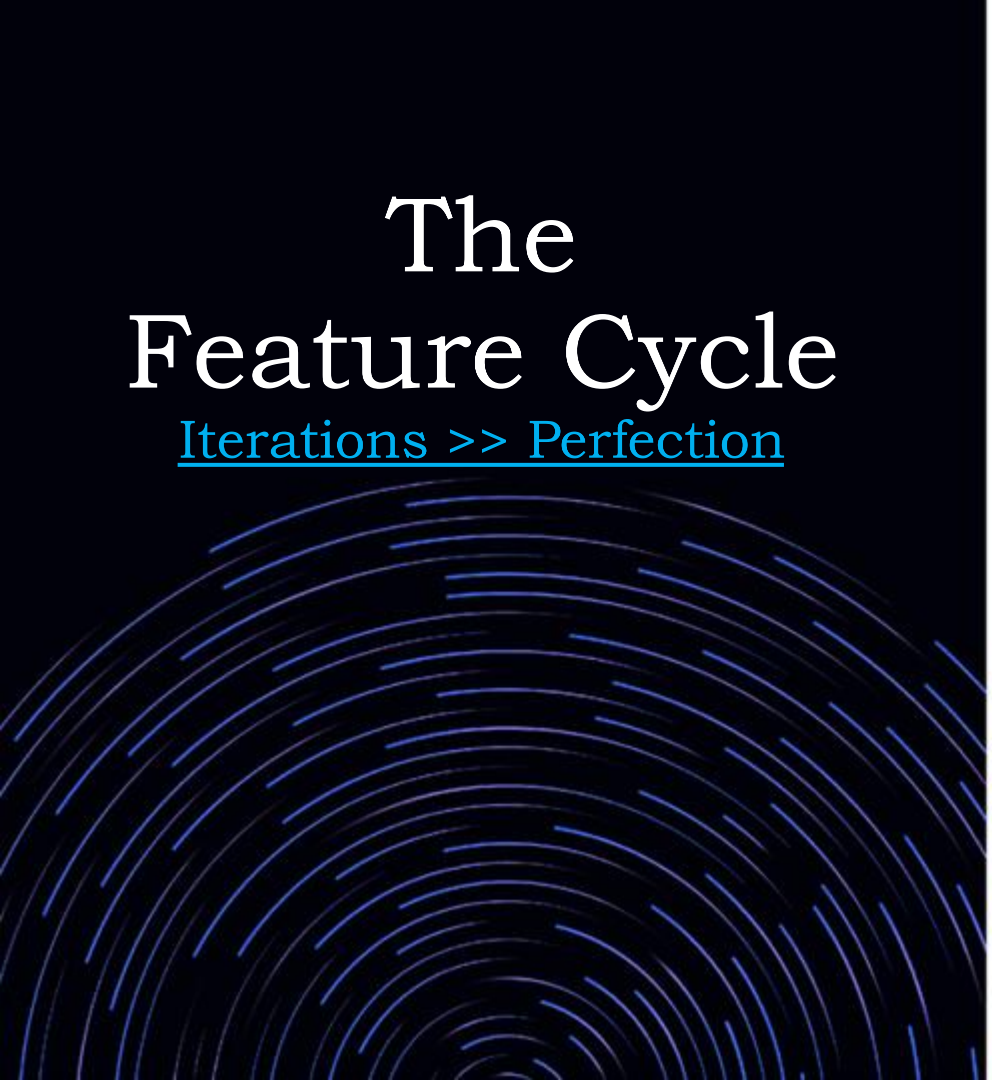
## Unit Testing

AI-Assist

- ChatGPT* and Copilot* can streamline unit-test creation.
- Will require ***very specific*** instructions
  - "/tests patch all referenced functions, …"
- Will require correction and clarification
- Add comments to the tests for clarity
  - # Setup, # Resetting Mocks, # Cleanup
  - # Case 1: Do this thing
  - # Case 2: Do this other thing

* depending on sensitivity

Flywheel



## The Feature Cycle

Iterations >> Perfection

- Cataloging Proposed Changes
- Prioritizing
- Writing "Stories"
- Execution

Flywheel

# Cataloging Proposed Changes

## Data Mining Your Notes

- Search for proposed changes in your notes
  - "- [ ]"
- Catalog them in a file
  - "Proposals.md"
  - ...
- Refine them for prioritization and presentation
  - Flesh them out
  - How does it add value?
  - How does it increase quality?
  - How does it make it easier to use?
  - ….
- "Clarify" of GTD

Flywheel

# Prioritizing Changes

## The Needs of the Customer Come First

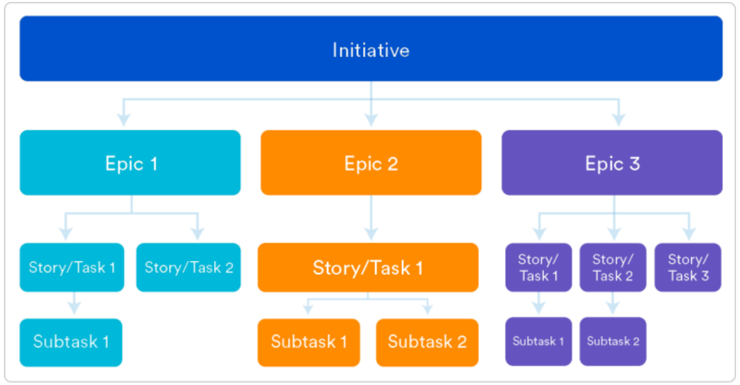
- Customer Feature Requests take priority
  - "Foundations" enable quality
- Group other proposals into
  - FIX, ENH, BUG, ...
  - Major Changes
  - Medium-Scale Changes
  - Minor Changes
    - May be "bundled" with other changes
- Discuss with stake-holders
  - What changes are priority?
  - When can they be scheduled?
  - How can they be delegated?
- "Organize" of GTD

Flywheel

# Writing "Stories"

## "Who", "What", "Why" and "How"

- Value Statement:
  - As a <role> (customer, developer,...) (**Who**)
  - I want <feature> ("GPU Gears") (**What**)
  - Such that <value> is delivered. (**Why**)
  - "As a developer I want 100% test coverage such that high code quality is assured."
- Acceptance Criteria (**How**):
  - A research-spike is performed
  - It is demonstrated that...
    - Aspect of feature is delivered
    - This is actively "***demo***"-nstrated
  - A Code Review is performed through a Merge Request
  - Successful execution is demonstrated
- "Groomed"/Reviewed with stake-holders of the project



Flywheel

# Writing "Stories"

## A Problem Well Stated is a Problem Half-Solved

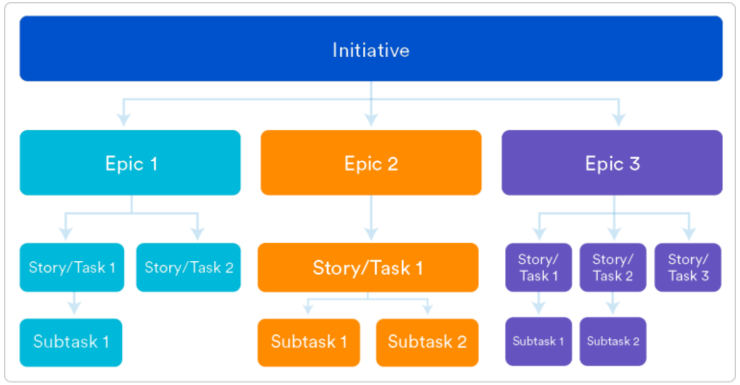
As an hpc-client user I want a configuration interview to guide me through a setup process such that successfully installing and configuring the hpc-client on my HPC Cluster is quick and easy.

**Acceptance Criteria:**

- A research spike is performed to identify necessary tools
- The hpc-client is demonstrated to have a configuration interview with default values
- Unit tests are passing with above 80% coverage
- Documentation is altered to reflect the above changes
- The above changes pass Code Reviews in one or more Merge Requests.

**Implementation Details:**

....



Flywheel

# Execution

## Where the Rubber Meets the Road

- Research
  - Remember to take notes!
- Experiment and record
  - What works, what doesn't, what BREAKS!
  - AND WHY!!!
- Implement
- Test
- Document
- Code Review
- Resolve Acceptance Criteria w/ Stakeholders
  - Demonstrate how the AC are achieved!!!
- "Engage" of GTD

Flywheel

# Final Words

- On Craftsmanship
- You Can't Do Everything

# On Craftsmanship

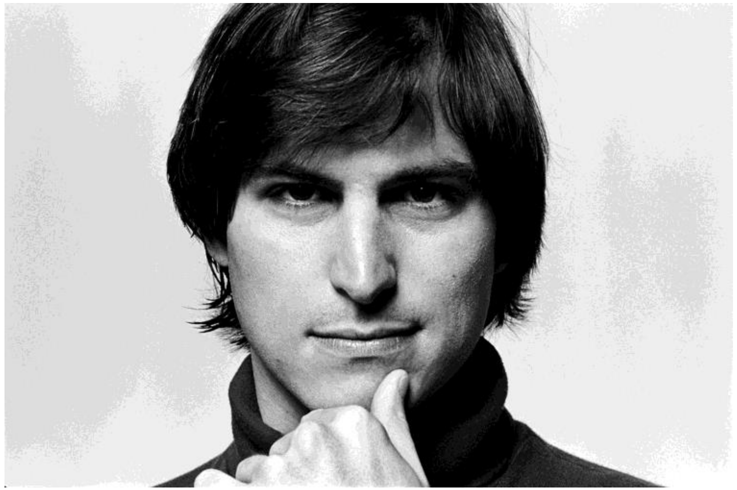
## Grueling through the Grit to Delivering Quality

"After I left Apple, John Scully caught a very serious disease. This disease is thinking that a really great idea is 90% of the work. And if you just tell people 'here's this really great idea' then of course they can go off and make it happen.

"The problem with this is that there is a tremendous amount of craftsmanship in between a great idea and a great product.... It never comes out like it starts...there are tremendous trade-offs....

"It's the process that's the magic."

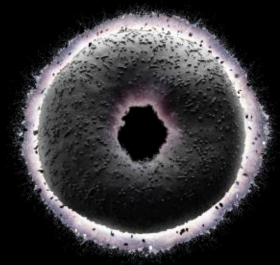
- Steve Jobs, on Craftsmanship



Flywheel

# You Can't Do Everything!!!

## Rely on your Team for Perspective



- Craftsmanship is an evolving system of habits, skills, and relationships
  - It will never be "***Perfect***"
- You will make mistakes
  - Forgive yourself
- "Linger Not on the Struggle Bus"
  - Ask for Help at the Right Time
- "Navigating Relationships is the Master Skill"
  - Be Polite
  - Be Professional
  - ***Be Persistent!***
- No plan survives contact with "Reality"
  - Adapt
- "Steal like an Artist"!