Scientific Computing

How to approach and conduct code reviews of scientific software

Harry Swift

Who am I?

Research Software Engineer

Team: Biomolecular Simulation Group, Computational Biology and Imaging Theme, Scientific Computing

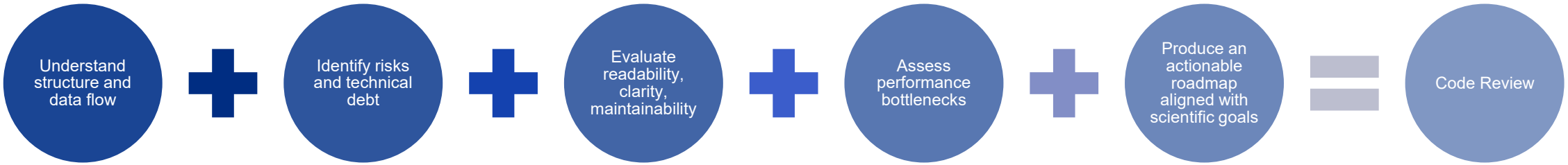
Science and Technology Facilities Council

Scientific Computing

Why scientific software is challenging?

- Often prioritises results over engineering quality
  - Quick prototypes become long-term tools
- Lacks modern practices
  - Few or no tests
  - Minimal documentation
  - Inconsistent coding styles
- Needs HPC-level performance
- Hard to onboard new contributors
  - Unclear architecture
  - Hidden assumptions

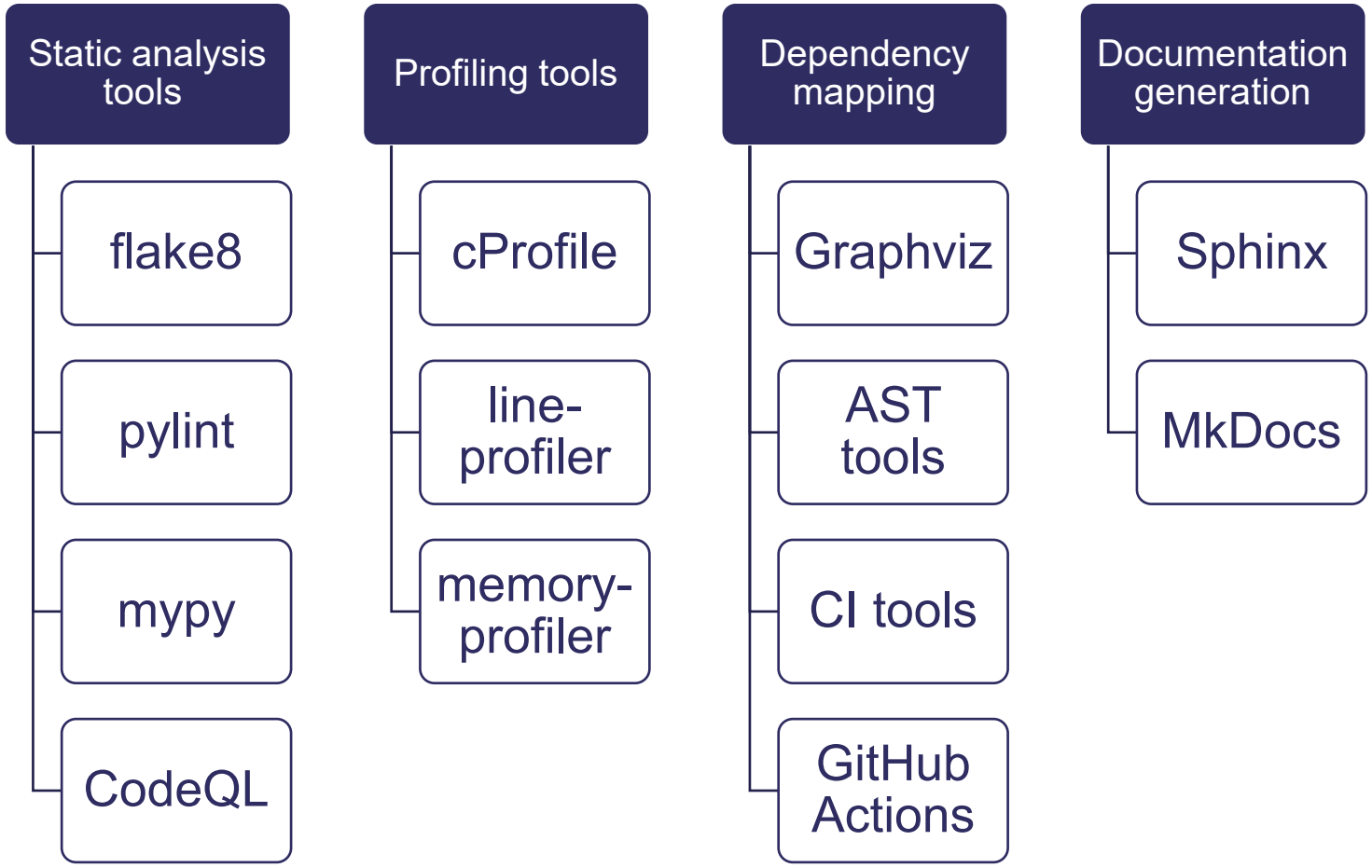
Goals of a code review



Code review methodology

Architecture review	Module boundaries
	Data and control flow
	Extensibility
Code quality review	Naming and consistency
	Function/module responsibilities
	Duplicate or redundant code
Scientific correctness	Numerical stability
	Validation of scientific assumptions
	Reproducibility of results
Performance assessment	Profiling hotspots
	Memory usage
	Parallelism and vectorisation strategy

What tools can we utilise?



How would we approach a codebase evaluation?

Complexity and architecture	Dependency graph
	Module interactions
Identification of fragile areas	Heavily coupled modules
	Oversized functions
Performance profile	Hotspot identification
Test coverage	Unit tests
	Integration tests
	Regression tests
	Reproducibility checks

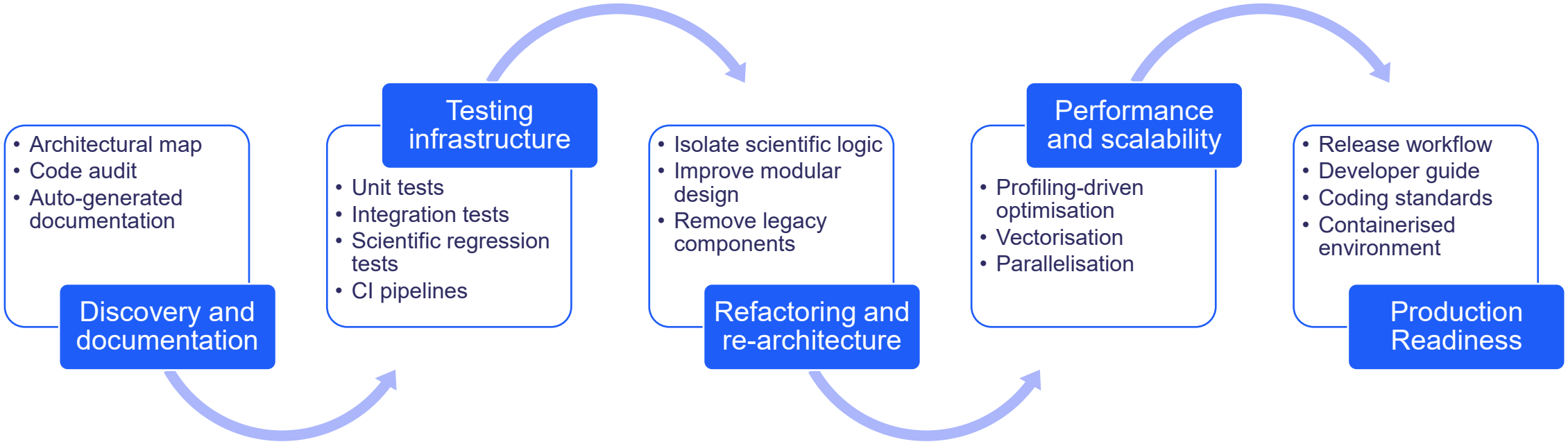
Prioritising issues

- Impact vs effort matrix
  - High-impact, low-effort → do first
  - High-impact, high-effort → plan as milestones
  - Low-impact, high-effort → deprioritise
  - Low-impact, low-effort → opportunistic
- Align work with scientific deadlines
  - Maintain accuracy-sensitive modules
  - Avoid destabilising key components

Modernisation goals

Clean modular architecture	Clear separation of concerns
Testable and reproducible scientific results	Regression tests
	Automated workflows
Scalable performance	Efficient numerical kernels
Reduced technical debt	Remove dead or duplicated code
	Improve clarity and maintainability
Robust developer workflows	CI, versioning, documentation, coding standards

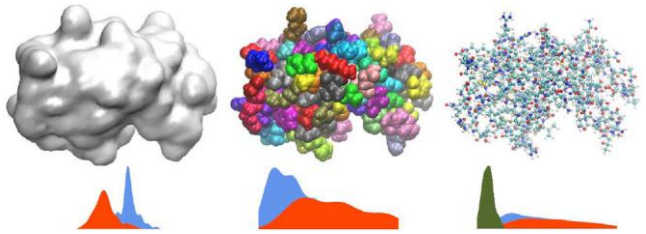
Modernisation milestones



Case Study - CodeEntropy

What is entropy?

Entropy measures the disorder or uncertainty in a system. In molecular simulations, it reflects how atoms explore different configurations, helping us understand flexibility, stability, and binding



Case Study – CodeEntropy

- Python package for macromolecular entropy prediction
- Implements the Multiscale Cell Correlation (MCC) method
- Part of a broader effort merging structural and fluidic entropy
- Towards a unified “super-theory”
- Aims for high-accuracy binding energy prediction
- Approximately 10× cheaper computationally than alternatives

CodeEntropy Initial State

First-pass theoretical integration of two tools

No tests

No CI

Minimal documentation

High technical debt

Hard to extend or add new theory

Script-like structure

Poor modular separation

Hidden assumptions and ad-hoc choices

Issues identified during review

No extensible architecture

Entropy models tightly coupled

Mixed responsibilities

- Theory, numerics, I/O, CLI intertwined

No validation or correctness checks

- No regression tests
- No accuracy benchmarks

Inconsistent coding practices

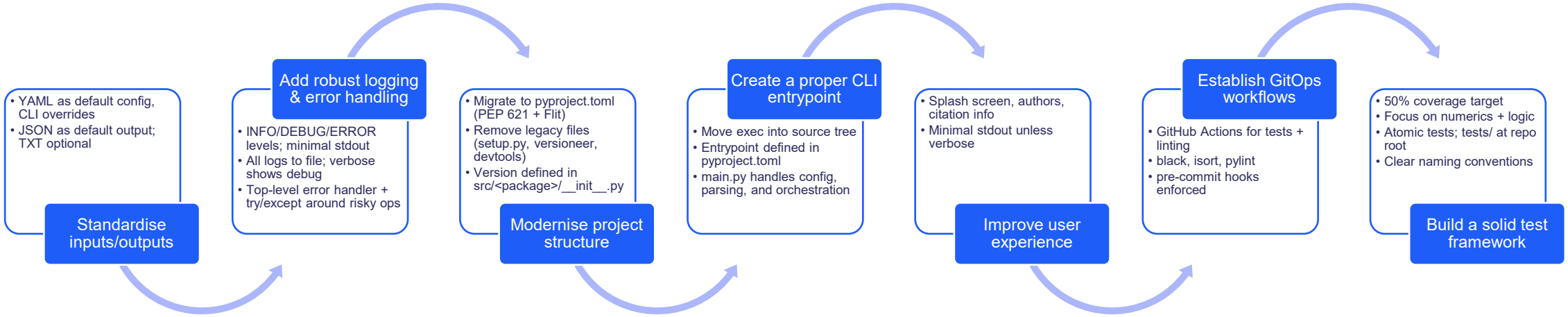
- Naming, duplication, layout

Performance issues

- Inefficient Python loops
- Duplicate computations

Scientific assumptions not documented

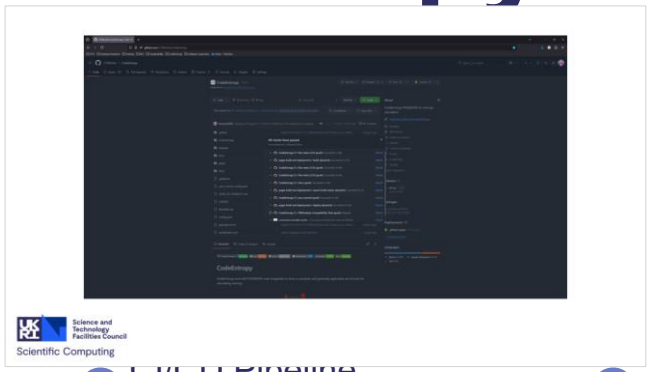
Case Study – CodeEntropy - Modernisation strategy



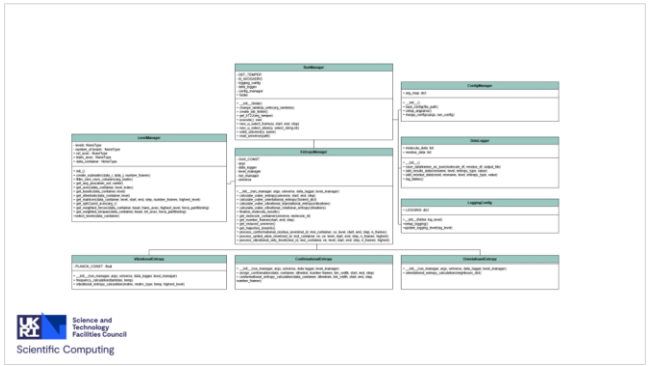
Science and Technology Facilities Council

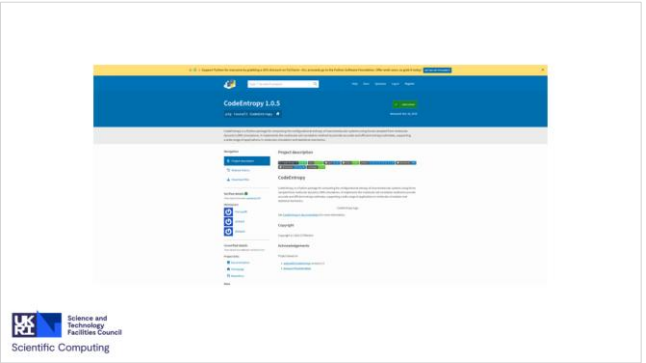
Scientific Computing

Delivering CodeEntropy

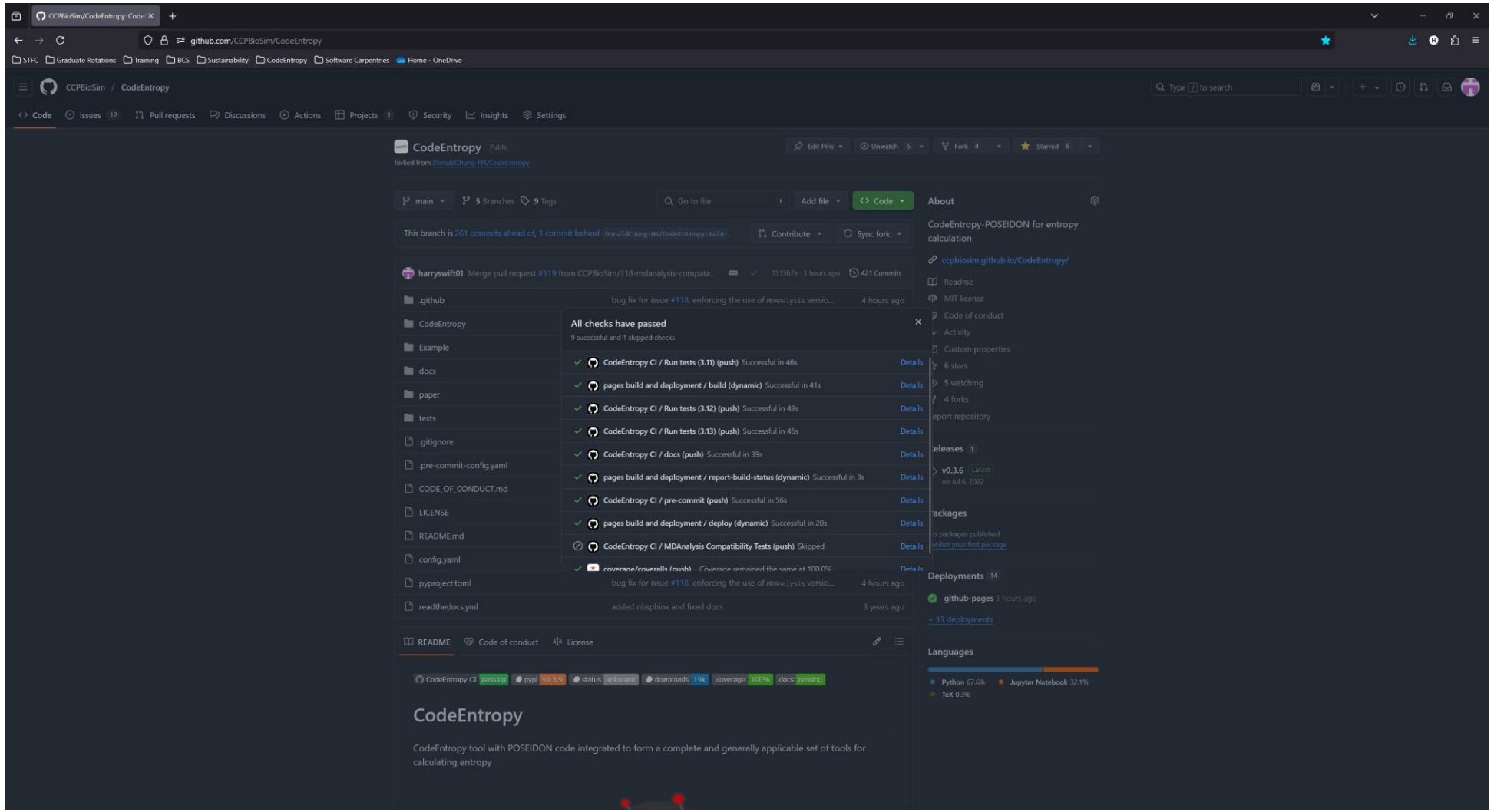


- Input System Overhaul
- CI/CD Pipeline Modernisation
- Testing and Code Quality
- Logging and Error handling
- Modular Code Architecture
- Release





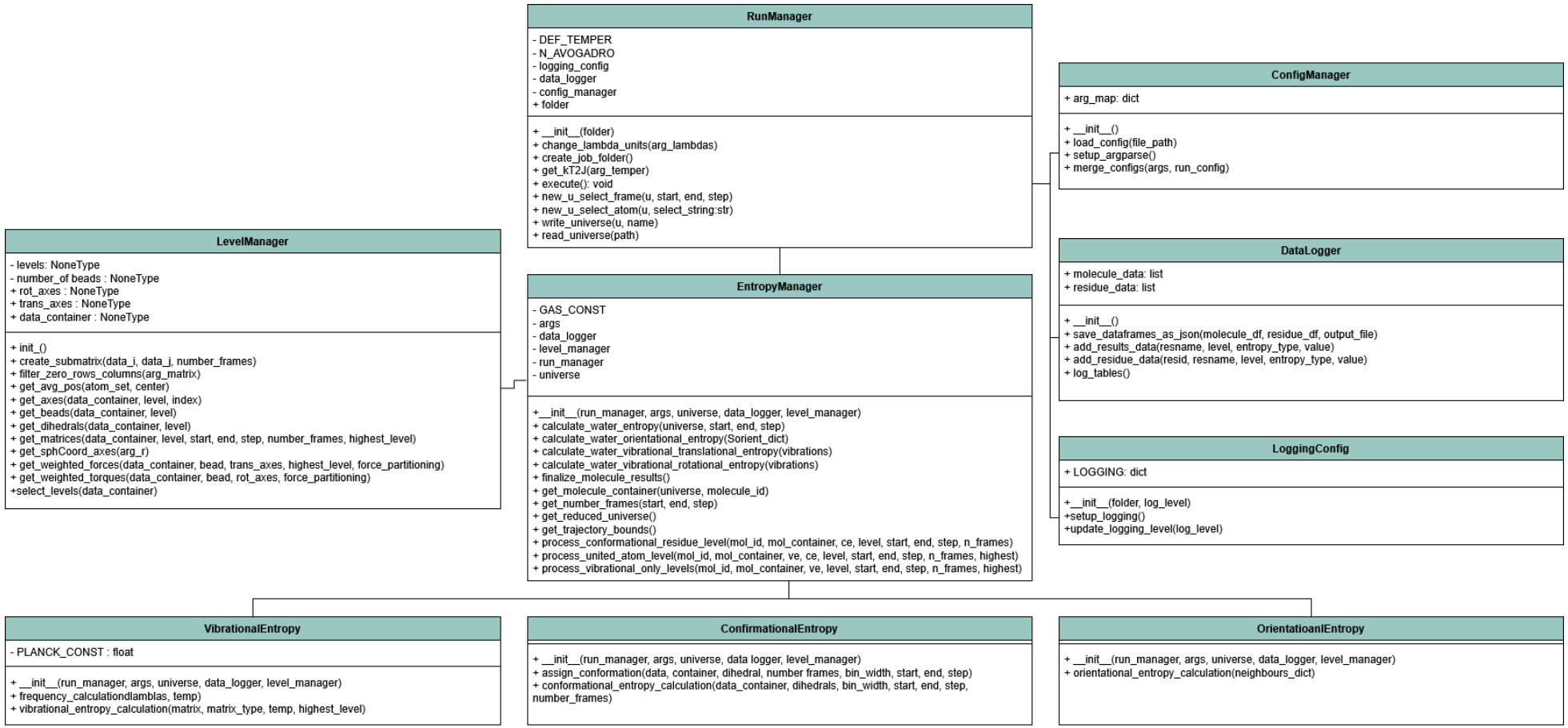
Science and Technology Facilities Council
Scientific Computing

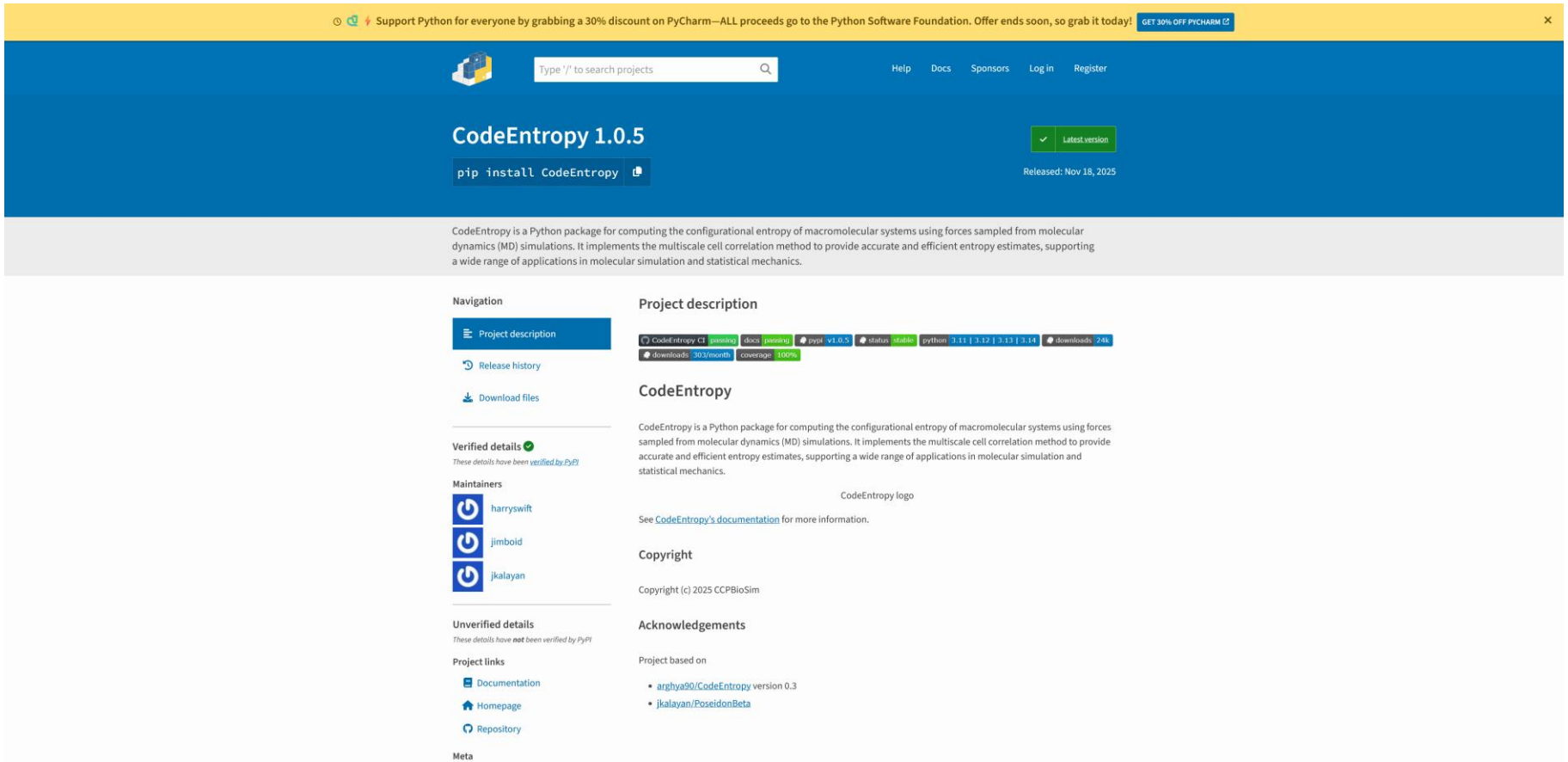


Science and Technology Facilities Council

Scientific Computing


CCPBIOSIM / CODEENTROPY
100%


RunManager
- DEF_TEMPER
- N_AVOGADRO
- logging_config
- data_logger
- config_manager
+ folder
+ __init__(folder)
+ change_lambda_units(arg_lambdas)
+ create_job_folder()
+ get_kT2J(arg_temper)
+ execute(): void
+ new_u_select_frame(u, start, end, step)
+ new_u_select_atom(u, select_string:str)
+ write_universe(u, name)
+ read_universe(path)
ConfigManager
+ arg_map: dict
+ __init__()
+ load_config(file_path)
+ setup_argparse()
+ merge_configs(args, run_config)
LevelManager
- levels: NoneType
- number_of beads : NoneType
+ rot_axes : NoneType
+ trans_axes : NoneType
+ data_container : NoneType
+ init_()
+ create_submatrix(data_i, data_j, number_frames)
+ filter_zero_rows_columns(arg_matrix)
+ get_avg_pos(atom_set, center)
+ get_axes(data_container, level, index)
+ get_beads(data_container, level)
+ get_dihedrals(data_container, level)
+ get_matrices(data_container, level, start, end, step, number_frames, highest_level)
+ get_sphCoord_axes(arg_r)
+ get_weighted_forces(data_container, bead, trans_axes, highest_level, force_partitioning)
+ get_weighted_torques(data_container, bead, rot_axes, force_partitioning)
+select_levels(data_container)
EntropyManager
- GAS_CONST
- args
- data_logger
- level_manager
- run_manager
- universe
+__init__(run_manager, args, universe, data_logger, level_manager)
+ calculate_water_entropy(universe, start, end, step)
+ calculate_water_orientational_entropy(Sorient_dict)
+ calculate_water_vibrational_translational_entropy(vibrations)
+ calculate_water_vibrational_rotational_entropy(vibrations)
+ finalize_molecule_results()
+ get_molecule_container(universe, molecule_id)
+ get_number_frames(start, end, step)
+ get_reduced_universe()
+ get_trajectory_bounds()
+ process_conformational_residue_level(mol_id, mol_container, ce, level, start, end, step, n_frames)
+ process_united_atom_level(mol_id, mol_container, ve, ce, level, start, end, step, n_frames, highest)
+ process_vibrational_only_levels(mol_id, mol_container, ve, level, start, end, step, n_frames, highest)
DataLogger
+ molecule_data: list
+ residue_data: list
+ __init__()
+ save_dataframes_as_json(molecule_df, residue_df, output_file)
+ add_results_data(resname, level, entropy_type, value)
+ add_residue_data(resid, resname, level, entropy_type, value)
+ log_tables()
LoggingConfig
+ LOGGING: dict
+__init__(folder, log_level)
+setup_logging()
+update_logging_level(log_level)
VibrationalEntropy
- PLANCK_CONST : float
+ __init__(run_manager, args, universe, data_logger, level_manager)
+ frequency_calculationdlamblas, temp)
+ vibrational_entropy_calculation(matrix, matrix_type, temp, highest_level)
ConfirmationalEntropy
+ __init__(run_manager, args, universe, data logger, level_manager)
+ assign_conformation(data, container, dihedral, number frames, bin_width, start, end, step)
+ conformational_entropy_calculation(data_container, dihedrals, bin_width, start, end, step, number_frames)
OrientatioanlEntropy
+ __init__(run_manager, args, universe, data_logger, level_manager)
+ orientational_entropy_calculation(neighbours_dict)



Science and Technology Facilities Council

Scientific Computing

Science and Technology Facilities Council

Scientific Computing

Questions?

Science and Technology Facilities Council

Scientific Computing

Thank you

scd.stfc.ac.uk

@SciComp_STFC