The original `cosmic_integration` package on GitHub is powerful but suffers from several architectural and stylistic issues. Many modules are monolithic scripts (e.g., `ClassCosmicIntegrator.py` and `FastCosmicIntegration.py`) with long functions and heavy use of global state or print-based reminders. Classes such as `COMPASData` are declared in files with names beginning with “Class*”, and constructors rely on prints to remind users to call other methods 【368244662012357†L59-L64】. The integrator itself prints instructions about what to do next【670899451913518†L89-L129】, and `MSSFR` prints reminders to call setup functions【738272119352158†L74-L83】. These ad-hoc prints make automation difficult and violate the principle of separation of concerns. Additionally, functions and variables use inconsistent naming conventions and lack type hints. Files like `FastCosmicIntegration.py` are more than a thousand lines long and mix command-line parsing, astrophysical calculations and plotting, while scattering debugging prints throughout【361501502538014†L0-L11】. PEP8 recommends snake_case for file and function names and CamelCase for classes, yet the current code uses CamelCase for file names and inconsistent indentation. There are a few well-structured modules—`cosmology.py` includes a proper docstring with parameters and return values【347454130662587†L10-L32】, and `selection_effects.py` demonstrates clear function documentation 【327604978656573†L20-L54】—but these are exceptions.

Proposed refactoring plan

1. **Reorganise package structure** – Create a clear namespace hierarchy. Instead of monolithic scripts, split concerns into modules named with snake_case. Top-level files should expose only public APIs (e.g. `compas_data.py` for loading COMPAS output, `cosmic_integrator.py` for integration logic, `mssfr.py` for metallicity-specific star-formation models, `mass_evolution.py` for IMF-related functions, `selection_effects.py` for detection probability utilities, etc.). A `binned` subpackage can encapsulate the two-dimensional grid integrator, housing classes such as `BinaryPopulation`, `CosmologicalModel`, `DetectionMatrix` and helper modules.

2. **Rename classes and methods to follow PEP8** – Remove the “Class” prefix from file and class names. Use CamelCase for class names (`CompasData`, `CosmicIntegrator`, `MSSFR`) and snake_case for functions (`set_dco_mask` instead of `setCOMPASDCOmask`). This improves readability and consistency.

3. **Replace print statements with logging** – Constructors currently print instructions (e.g., `COMPASData.__init__` prints reminders【368244662012357†L59-L64】). Such reminders should be conveyed through docstrings or raised exceptions. Use Python’s `logging` module for optional verbosity and remove side effects in initialisation.

4. **Use dataclasses and type hints** – Most classes merely store data; refactoring them as `@dataclass` simplifies attribute definitions and automatically generates `__init__`. Annotate all arguments and return values with type hints to aid both readability and static analysis.

5. **Minimise global state** – Global variables like `_interpolator` in `selection_effects.py` and `COSMOLOGY` in `cosmology.py` should be encapsulated. Provide functions that return new objects or use caching decorators rather than mutating module-level variables.

6. **Decompose long functions** – Break complex methods (e.g. the nested loops in `ClassCosmicIntegrator.cosmologicalIntegration`【670899451913518†L283-L320】 or the giant `FastCosmicIntegration.py`) into smaller, well-named functions. Pull out argument parsing into a separate CLI module, separate plotting functions into a `plotting.py` module, and isolate heavy numerical routines.

7. **Introduce clear docstrings** – Every public function and class should have

a docstring following NumPy or Google style, explaining parameters, return values, units and side effects. The docstrings in `cosmology.py` 【347454130662587†L10-L32】 and `selection_effects.py`【327604978656573†L20-L54】 provide good examples.

8. **Provide enumerations and constants** – Replace magic strings such as “BBH”, “BHNS” with `Enum` classes (as demonstrated in the `stellar_type.py` enumeration). Similarly, define constants for default cosmological parameters rather than scattering numbers through the code.

9. **Add unit tests and continuous integration** – After refactoring, write tests for all modules (e.g. verifying mass conversions, metallicity binning, detection probability interpolation). Configure CI with `pytest` and `pre-commit` to enforce PEP8 compliance and run tests automatically.

10. **Document usage and examples** – Provide example notebooks or scripts demonstrating typical workflows. Move the command-line parser out of `FastCosmicIntegration.py` into a top-level `cli.py` using `argparse` or `click`, leaving the integration logic importable.

Updated package skeleton

To illustrate the proposed organisation, a folder of Python stubs has been created. Each file outlines the intended public API with type hints and docstrings. For example:

- `compas_data.py` defines a `CompasData` dataclass with methods `load_data`, `set_dco_mask`, `compute_delay_times` and `compute_metallicity_grid`.
- `cosmic_integrator.py` provides a `CosmicIntegrator` dataclass with methods `create_redshift_shells`, `compute_birth_arrays` and `integrate`.
- `mssfr.py` defines a `MSSFR` model where users can set metallicity, galaxy mass function and SFR prescriptions.
- `mass_evolution.py` holds IMF-related utility functions such as `analytical_star_forming_mass_per_binary_using_kroupa_imf`.
- `selection_effects.py` stubs detection probability functions.
- A `binned` subpackage includes `binary_population.py`, `cosmological_model.py`, `detection_matrix.py`, `detection_rate_computer.py`, `snr_grid.py`, `plotting.py`, `gpu_utils.py`, `io.py`, `stellar_type.py` and `bin_2d_data.py`. These modules mirror the original “binned_cosmic_integrator” but adopt clear naming and encapsulate GPU handling, I/O helpers, plotting, and detection rate computation. For example, the `BinaryPopulation` class includes a `from_compas_h5` factory method, and `DetectionMatrix` cleanly exposes `compute`, `save`, and `plot` methods.

The complete skeleton of the refactored package is provided in the attached archive for reference. It demonstrates how the codebase could be reorganised to improve clarity, modularity and compliance with PEP8.