157 lines - 2 Removals

```
"""
    juliet_pipeline_steps: Implementation of pipeline steps to access the Juliet test suite and select testcases.
"""

import logging
import typing

from pipeline_framework import PipelineState, PipelineStep


class EnumerateCWEsPipelineStep(PipelineStep):
    step_key = "enumerate_cwes"

    def execute(self, input: dict, output: dict) -> None:
        import juliet_access

        output["cwes"] = juliet_access.enumerate_cwes()




        if self.state.smoke_test:
            self.logger.warning("Smoke test, only enumerating first 30 CWEs")
            output["cwes"] = output["cwes"][:30]

    def output_ready(self, output: dict) -> bool:
        return "cwes" in output.keys() and (
            len(output["cwes"]) == 118 or self.state.smoke_test
        )


class EnumerateTestcasesPipelineStep(PipelineStep):
    step_key = "enumerate_testcases"

    def input_ready(self, input: dict) -> bool:
        return "cwes" in input.keys()

    def execute(self, input: dict, output: dict) -> None:
        import juliet_access

        self.logger.debug(f'Enumerating testcases for {len(input["cwes"])} CWEs')
        output["testcases"] = sum(
            [juliet_access.enumerate_testcases(cwe) for cwe in input["cwes"]], []
        )




```

165 lines + 13 Additions

```
"""
    juliet_pipeline_steps: Implementation of pipeline steps to access the Juliet test suite and select testcases.
"""

import logging
import typing

from pipeline_framework import PipelineState, PipelineStep


class EnumerateCWEsPipelineStep(PipelineStep):
    step_key = "enumerate_cwes"

    def execute(self, input: dict, output: dict) -> None:
        import juliet_access

        if self.state.malware:

            output["cwes"] = juliet_access.enumerate_cwes_malware(self.state.malware)
        else:
            output["cwes"] = juliet_access.enumerate_cwes(self.state.malware)


        if self.state.smoke_test:
            self.logger.warning("Smoke test, only enumerating first 30 CWEs")
            output["cwes"] = output["cwes"][:30]

    def output_ready(self, output: dict) -> bool:
        return "cwes" in output.keys() and (
            len(output["cwes"]) == 118 or self.state.smoke_test
        )


class EnumerateTestcasesPipelineStep(PipelineStep):
    step_key = "enumerate_testcases"

    def input_ready(self, input: dict) -> bool:
        return "cwes" in input.keys()

    def execute(self, input: dict, output: dict) -> None:
        import juliet_access

        self.logger.debug(f'Enumerating testcases for {len(input["cwes"])} CWEs')
        if self.state.malware:
            output["testcases"] = sum(

                [juliet_access.enumerate_testcases_malware(cwe) for cwe in input["cwes"]], []
            )
        else:
            output["testcases"] = sum(
                [juliet_access.enumerate_testcases(cwe) for cwe in input["cwes"]], []
            )


```

```
        if self.state.smoke_test:
            self.logger.warning("Smoke test, only enumerating random 1000 testcases")
            shuffled_list = list(output["testcases"])
            import random

            random.shuffle(shuffled_list)
            output["testcases"] = shuffled_list[:1000]

    def output_ready(self, output: dict) -> bool:
        return "testcases" in output.keys() and (
            len(output["testcases"]) == 64099 or self.state.smoke_test
        )


class PrintTestcasesPerCWEPipelineStep(PipelineStep):
    step_key = "print_testcases_per_cwe"

    def __init__(
        self,
        config: dict,
        state: PipelineState,
        logger: logging.Logger,
        comment: typing.Optional[str] = None,
    ):
        super().__init__(config, state, logger)

        self.comment = comment

    def input_ready(self, input: dict) -> bool:
        return "cwes" in input.keys() and "testcases" in input.keys()

    def execute(self, input: dict, output: dict) -> None:
        import utils

        utils.print_testcases_per_cwe(input["cwes"], input["testcases"], self.comment)


class FilterTestcasesPipelineStep(PipelineStep):
    step_key = "filter_testcases"

    def __init__(
        self,
        config: dict,
        state: PipelineState,
        logger: logging.Logger,
        function: str,
        arguments: typing.Optional[list] = None,
    ):
        super().__init__(config, state, logger)

        import juliet_access

        if arguments is not None:
            self.filter_fn = lambda cwes, testcases: juliet_access.filter_testcases(
                cwes,
                testcases,
                lambda testcase: getattr(juliet_access.filters, f"testcase_{function}")(
                    testcase, **arguments
```

```
        if self.state.smoke_test:
            self.logger.warning("Smoke test, only enumerating random 1000 testcases")
            shuffled_list = list(output["testcases"])
            import random

            random.shuffle(shuffled_list)
            output["testcases"] = shuffled_list[:1000]

    def output_ready(self, output: dict) -> bool:
        return "testcases" in output.keys() and (
            len(output["testcases"]) == 64099 or self.state.smoke_test
        )


class PrintTestcasesPerCWEPipelineStep(PipelineStep):
    step_key = "print_testcases_per_cwe"

    def __init__(
        self,
        config: dict,
        state: PipelineState,
        logger: logging.Logger,
        comment: typing.Optional[str] = None,
    ):
        super().__init__(config, state, logger)

        self.comment = comment

    def input_ready(self, input: dict) -> bool:
        return "cwes" in input.keys() and "testcases" in input.keys()

    def execute(self, input: dict, output: dict) -> None:
        import utils

        utils.print_testcases_per_cwe(input["cwes"], input["testcases"], self.comment)


class FilterTestcasesPipelineStep(PipelineStep):
    step_key = "filter_testcases"

    def __init__(
        self,
        config: dict,
        state: PipelineState,
        logger: logging.Logger,
        function: str,
        arguments: typing.Optional[list] = None,
    ):
        super().__init__(config, state, logger)

        import juliet_access

        if arguments is not None:
            self.filter_fn = lambda cwes, testcases: juliet_access.filter_testcases(
                cwes,
                testcases,
                lambda testcase: getattr(juliet_access.filters, f"testcase_{function}")(
                    testcase, **arguments
```

```
                ),
            )
        else:
            self.filter_fn = lambda cwes, testcases: juliet_access.filter_testcases(
                cwes, testcases, getattr(juliet_access.filters, f"testcase_{function}")
            )

    def input_ready(self, input: dict) -> bool:
        return "testcases" in input.keys()

    def execute(self, input: dict, output: dict) -> None:
        output["testcases"] = self.filter_fn(input["cwes"], input["testcases"])


class FilterCWEsPipelineStep(PipelineStep):
    step_key = "filter_cwes"

    def __init__(
        self,
        config: dict,
        state: PipelineState,
        logger: logging.Logger,
        function: str,
        arguments: typing.Optional[list] = None,
    ):
        super().__init__(config, state, logger)

        import juliet_access

        if arguments is not None:
            self.filter_fn = lambda cwes, testcases: juliet_access.filter_cwes(
                cwes,
                testcases,
                lambda cwe: getattr(juliet_access.filters, f"cwe_{function}")(
                    cwe, **arguments
                ),
            )
        else:
            self.filter_fn = lambda cwes, testcases: juliet_access.filter_cwes(
                cwes, testcases, getattr(juliet_access.filters, f"cwe_{function}")
            )

    def input_ready(self, input: dict) -> bool:
        return "cwes" in input.keys() and "testcases" in input.keys()

    def execute(self, input: dict, output: dict) -> None:
        output["cwes"] = self.filter_fn(input["cwes"], input["testcases"])


__all__ = [
    "EnumerateCWEsPipelineStep",
    "EnumerateTestcasesPipelineStep",
    "PrintTestcasesPerCWEPipelineStep",
    "FilterTestcasesPipelineStep",
    "FilterCWEsPipelineStep",
]
```


```
                ),
            )
        else:
            self.filter_fn = lambda cwes, testcases: juliet_access.filter_testcases(
                cwes, testcases, getattr(juliet_access.filters, f"testcase_{function}")
            )

    def input_ready(self, input: dict) -> bool:
        return "testcases" in input.keys()

    def execute(self, input: dict, output: dict) -> None:
        output["testcases"] = self.filter_fn(input["cwes"], input["testcases"])


class FilterCWEsPipelineStep(PipelineStep):
    step_key = "filter_cwes"

    def __init__(
        self,
        config: dict,
        state: PipelineState,
        logger: logging.Logger,
        function: str,
        arguments: typing.Optional[list] = None,
    ):
        super().__init__(config, state, logger)

        import juliet_access

        if arguments is not None:
            self.filter_fn = lambda cwes, testcases: juliet_access.filter_cwes(
                cwes,
                testcases,
                lambda cwe: getattr(juliet_access.filters, f"cwe_{function}")(
                    cwe, **arguments
                ),
            )
        else:
            self.filter_fn = lambda cwes, testcases: juliet_access.filter_cwes(
                cwes, testcases, getattr(juliet_access.filters, f"cwe_{function}")
            )

    def input_ready(self, input: dict) -> bool:
        return "cwes" in input.keys() and "testcases" in input.keys()

    def execute(self, input: dict, output: dict) -> None:
        output["cwes"] = self.filter_fn(input["cwes"], input["testcases"])


__all__ = [
    "EnumerateCWEsPipelineStep",
    "EnumerateTestcasesPipelineStep",
    "PrintTestcasesPerCWEPipelineStep",
    "FilterTestcasesPipelineStep",
    "FilterCWEsPipelineStep",
]
```