University of
South Australia

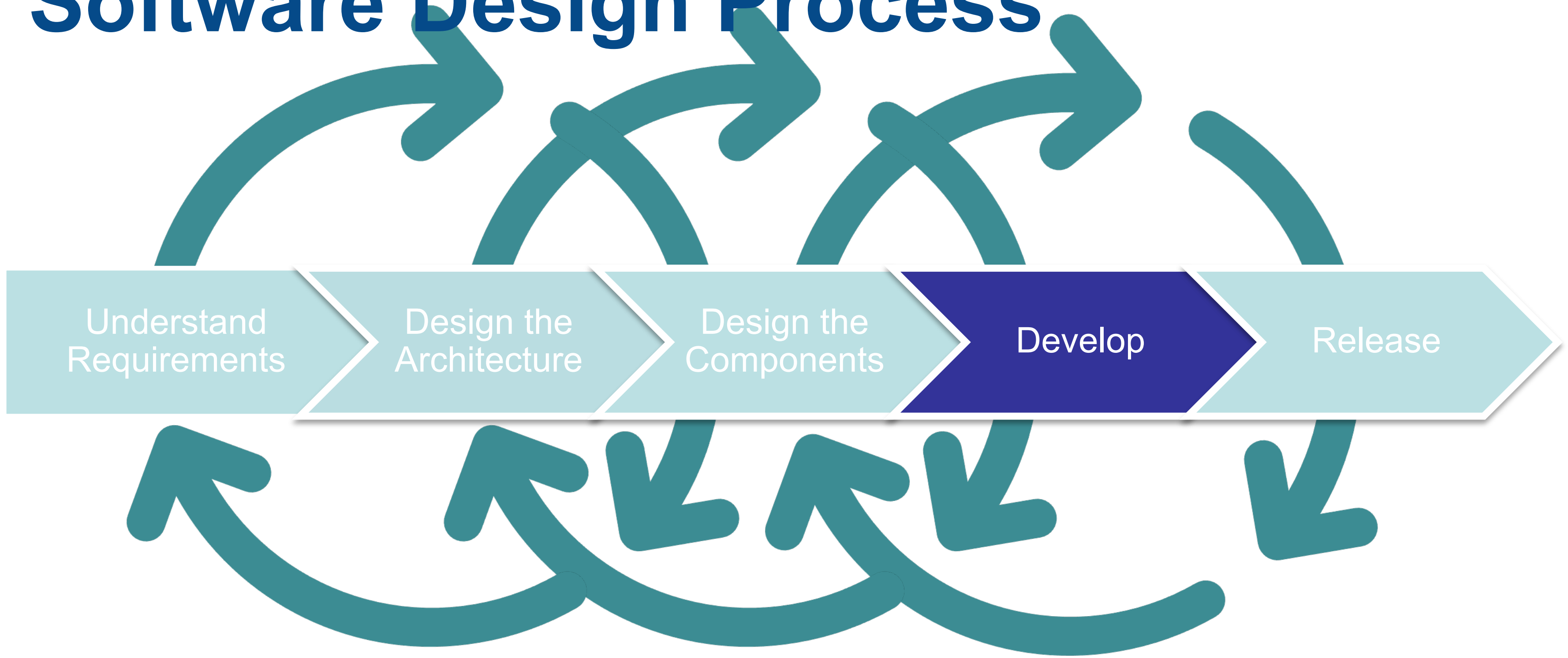
INFS 2044

Practical 4

Preparation

- Read the required readings
- Watch the Week 13 Lecture

Software Design Process



Learning Objectives

- Detect code smells (CO4)
- Improve code quality through refactoring (CO4)
- Use automated tools to check adherence to coding standards (CO6)

Task 1. Identify Code Smells

- Download the code prac4_code_initial.py from the course website
- Review the code and identify any code smells

Task 2. Create a Test Suite

- Write unit tests to cover normal operations for function aggregate_table
- Ensure tests for boundary conditions are included
  - Boundary conditions are scenarios at the boundary between normal scenarios and those that trigger errors
  - For example, rows that only have an operation name, but no numbers, only one number, etc
- Ensure that all tests pass

Task 3. Refactoring Step 1

- Apply the “Extract Method” refactoring to break the long function into smaller functions. This also addresses code duplication.
- Aim to separate into different functions:
  - Splitting the string into lines
  - Parsing each line
  - Converting the string values into numbers
  - Apply the requested operation to the numbers
  - Print error messages
- Ensure all variables and functions have descriptive names
- Verify that the unit tests still pass

Task 4. Examine the Revised Code

- Does it resolve the code smells identified earlier?
- Which code smells are still present?

Task 5. Refactoring Step 2

- Refactor the code to make it extensible without modifying existing code.
- See Hints on the subsequent slides.

Hints (1)

- Introduce an abstract class for operations and create separate subclasses for each operation
- Create a factory that creates an object of the correct type given the requested operation key (string in the table) and the numeric data
  - The factory can use a dictionary to map the operation keys to the corresponding classes
  - This mechanism allows to extend by adding entries to that dictionary instead of modifying conditional logic.
- Use the factory in do_calculation to create the correct object

Hints (2)

- You can further improve the design by separating parsing from doing computation.
  - Rename do_calculation to create_operation
  - Do not invoke the calculation from do_calculation
  - Instead, return the operation objects you created
  - These can be invoked in aggregate_table to then carry out the computation.
- All refactorings combined separate the parsing of the string, the selection of the computation, and the execution of the computation from each other.

Task 6. Check Code Style

- Install pylint

```
python -m pip install pylint
```

- Run *pylint* to see if your code has issues

```
python -m pylint prac4_code_final.py
```

You Should Know

- Detect signs of poor code
- Refactor code to improve its quality
- Use pylint to check coding style

Activities this Week

- Continue working on Assignment 2

University of
South Australia