Test-Driven Development (TDD)

Instead of writing code, then writing tests …

Write tests, then write code

“After implementing a small part of the feature, I found that some of the implemented functionality was similar to something in another class, therefore I started changing that class so I could use it instead. During that work I discovered a defect that I then began to fix. However, to fix it I had to change the internal data structure a lot, which led to adding a parameter to a few of the class methods. Therefore I had to change all the places where these methods were called. Then I noticed …” [Christensen]

TDD tries to avoid these kinds of wild (coding) goose chases …
Test-Driven Development (TDD)

Instead of writing code, then writing tests …

Write tests, then write code

The basic TDD five-step process …

1. (Re-)Write a test (or tests)
2. Watch it fail (ensure test works)
3. Add enough code to pass test (stay focused)
4. Watch test succeed (and regressions)
5. Refactor as needed (non-feature changes)

Test-Driven Development (TDD)

Instead of writing code, then writing tests …

Write tests, then write code

Some basic TDD ideas …

- improve design as you go (refactoring)
  - includes your testing code as well
  - e.g., using JUnit fixtures
- stay focused on task at hand by coding to tests
- take small steps as you code
Exercise

Break into same “subgroups” and do the next set of exercises

Options if you already have been doing unit testing …

- Create more tests
- Try TDD
- Create new types of tests (e.g., integration or UI tests)
- Add unit testing to your checklist
- Add support for automating unit tests in your build (e.g., in GitHub, Continuous Integration, etc.)