Test-Driven Development

CS 485/698: AI-Assisted SE

Today's Agenda

- Team meeting (~15 minutes)
  - Standup: how is P5 going? What else do you plan to build before the end of the semester (i.e., for P7)?
  - Make sure that your **issue tracker** is up-to-date and contains at least one known issue (needed for today's in-class activity)
    - If you don't know of any issues, use this time to find one
    - Michael and I will be coming around to check
- Lecture on test-driven development
- In-class activity

Test driven development

Definition: ***test driven development*** (TDD) is a software development process that relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, then the software is improved so that the tests pass.

Test driven development

Definition: ***test driven development*** (TDD) is a software development process that relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, then the software is improved so that the tests pass.

- **key idea**: using TDD **guarantees** that you have a test for each line of code that you write

Test driven development

Definition: ***test driven development*** (TDD) is a software development process that relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, then the software is improved so that the tests pass.

- **key idea**: using TDD **guarantees** that you have a test for each line of code that you write
- research shows that TDD **dramatically improves** software quality (as measured by defect density) when humans use it
  - implication: **always use TDD** if possible

Test driven development: steps

1. “think of a test that will **force** you to add the next few lines of production code”

Test driven development: steps

1. “think of a test that will **force** you to add the next few lines of production code”

requirement: the test must **fail** when first written!

- “run your entire suite of tests and watch the new test fail”

Test driven development: steps

1. “think of a test that will **force** you to add the next few lines of production code”

requirement: the test must **fail** when first written!

- “run your entire suite of tests and watch the new test fail”
- what if your new test ***doesn’t*** fail?
    - actually a very common problem! you’ll have to try again

Test driven development: steps

1. “think of a test that will **force** you to add the next few lines of production code”
2. write the test and **observe** the test failure

Test driven development: steps

1. “think of a test that will **force** you t
   production code”
2. write the test and **observe** the test failure

Common mistake: don’t actually run the tests, just assume that your test will fail

Test driven development: steps

1. “think of a test that will **force** you to add the next few lines of production code”
2. write the test and **observe** the test failure
3. write **just enough** code to get the test to pass

Test driven development: steps

1. “think of a test that will **force** you to
   production code”
2. write the test and **observe** the test failure
3. write **just enough** code to get the test to pass

Don’t worry too much about elegance - goal in step 3 is to get back to **working code**

Test driven development: steps

1. “think of a test that will **force** you to add the next few lines of production code”
2. write the test and **observe** the test failure
3. write **just enough** code to get the test to pass
4. **refactor** your code to improve its quality/elegance, re-running the test after each change to make sure that it still passes

Test driven development: steps

1. “think of a test that will **force** you to add the next few lines of production code”
2. write the test and **observe** the test failure
3. write **just enough** code to get the test to pass
4. **refactor** your code to improve its quality/elegance, re-running the test after each change to make sure that it still passes
5. commit the new code **and the test**; make a PR

Test driven development: steps

1. “think of a test that will **force** you to add the next few lines of production code”
2. write the test and **observe** the test failure
3. write **just enough** code to get the test to pass
4. **refactor** your code to improve its quality/elegance, re-running the test after each change to make sure that it still passes
5. commit the new code **and the test**; make a PR
6. go back to step 1

Test driven development w/ LLMs

- Recent work [1] has shown that LLM agents using TDD **with human-written tests** significantly outperform… :
  - the same agents (that do TDD) with LLM-generated tests
  - baseline agents (that aren't explicitly instructed to use TDD) with access to the same human-written tests
- … at resolving issues in real GitHub repositories

[1] Han, Kevin, et al. "TDFlow: Agentic Workflows for Test Driven Development." Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers). 2026.

Test driven development w/ LLMs

- Recent work [1] has shown that LLM agents using TDD **with human-written tests** significantly outperform… :
  - the same agents (that do TDD) with LLM-generated tests
  - baseline agents (that aren't explicitly instructed to use TDD) with access to the same human-written tests
- … at resolving issues in real GitHub repositories

Today's in-class activity:
try to test this ourselves!

[1] Han, Kevin, et al. "TDFlow: Agentic Workflows for Test Driven Development." Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers). 2026.

In-Class Activity: Setup

- Locate your partner. Earlier in the alphabet, come to the front of the class.
- Each partner should **choose one issue** from their project team's issue tracker
- Make sure that both of you can access and work with each other's project repositories
  - Your LLM will need to do something with your partner's project repo at some point during this in-class activity
  - Bonus: this is a test of your project team's **ability to onboard a new developer**. If it isn't quick and easy, you have a problem.

In-Class Activity: TDD w/ AI vs human tests

In-Class Activity: TDD w/ AI vs human tests

- Pick one partner to go first. You'll both use their project + issue.
  - Call the partner your pick ***P1*** and the other partner ***P2***

In-Class Activity: TDD w/ AI vs human tests

- Pick one partner to go first. You'll both use their project + issue.
  - Call the partner your pick ***P1*** and the other partner ***P2***
- ***P1***: help ***P2*** make sure they can access + edit the project

In-Class Activity: TDD w/ AI vs human tests

- Pick one partner to go first. You'll both use their project + issue.
  - Call the partner your pick ***P1*** and the other partner ***P2***
- ***P1***: help ***P2*** make sure they can access + edit the project
- Once ***P2*** can work with the project:
  - ***P1*** should write a test for the chosen issue **by hand**
  - ***P2*** should generate a test for the issue **with their LLM**

In-Class Activity: TDD w/ AI vs human tests

- Pick one partner to go first. You'll both use their project + issue.
  - Call the partner your pick ***P1*** and the other partner ***P2***
- ***P1***: help ***P2*** make sure they can access + edit the project
- Once ***P2*** can work with the project:
  - ***P1*** should write a test for the chosen issue **by hand**
  - ***P2*** should generate a test for the issue **with their LLM**
- ***P1*** and ***P2*** **agree on a prompt** to debug + fix the issue via TDD. The only difference between the prompts should be the target test.

In-Class Activity: TDD w/ AI vs human tests

- Pick one partner to go first. You'll both use their project + issue.
  - Call the partner your pick ***P1*** and the other partner ***P2***
- ***P1***: help ***P2*** make sure they can access + edit the project
- Once ***P2*** can work with the project:
  - ***P1*** should write a test for the chosen issue **by hand**
  - ***P2*** should generate a test for the issue **with their LLM**
- ***P1*** and ***P2*** **agree on a prompt** to debug + fix the issue via TDD. The only difference between the prompts should be the target test.
- Both partners run the prompt using their test separately

In-Class Activity: TDD w/ AI vs human tests

- Pick one partner to go first. You'll both use their project + issue.
  - Call the partner your pick ***P1*** and the other partner ***P2***
- ***P1***: help ***P2*** make sure they can access + edit the project
- Once ***P2*** can work with the project:
  - ***P1*** should write a test for the chosen issue **by hand**
  - ***P2*** should generate a test for the issue **with their LLM**
- ***P1*** and ***P2*** **agree on a prompt** to debug + fix the issue via TDD. The only difference between the prompts should be the target test.
- Both partners run the prompt using their test separately
- If either of you fix the issue, PR it, switch ***P1*** and ***P2***, repeat

In-Class Activity: Discussion

- Completion grade for **opening at least one PR** during class today
  - If you didn't fix an issue, PR the best test case that you have (even though it's failing…right?)
  - Submit link to PR + your partner's name to Canvas assignment

In-Class Activity: Discussion

- Completion grade for **opening at least one PR** during class today
  - If you didn't fix an issue, PR the best test case that you have (even though it's failing…right?)
  - Submit link to PR + your partner's name to Canvas assignment
- **Discussion questions:**
  - How different were the human- vs AI-written tests?
  - Was TDD more successful with the human-written tests?
  - Was TDD actually helping your LLM?
  - Did the quality of the issue description matter?
  - Any other observations?

Wrapup and Reminders

- Sign up for A5 ASAP if you haven’t done so already
  - Nastygrams will come later today