Retinal Detachment Risk Calculator Application Development

A Test-Driven Development Approach

Luke Herbert

January 14, 2025

Presentation Overview

Contents

Project Overview

- I wanted to make a little application to help me with my retinal detachment risk assessment. It was a side quest...

Project Overview

- I wanted to make a little application to help me with my retinal detachment risk assessment. It was a side quest...
- I had available a paper with various risk factors for retinal detachment.

Project Overview

- I wanted to make a little application to help me with my retinal detachment risk assessment. It was a side quest...
- I had available a paper with various risk factors for retinal detachment.
- In my head was a simple little app I could use on my phone or computer, put in parameters, draw a retinal detachment and get a risk output.

Project Overview

- I wanted to make a little application to help me with my retinal detachment risk assessment. It was a side quest...
- I had available a paper with various risk factors for retinal detachment.
- In my head was a simple little app I could use on my phone or computer, put in parameters, draw a retinal detachment and get a risk output.

Contents

Initial Constraints

Core Requirements

- Had to be obvious to use
- No data sent off device
- Appear instantaneous to the user
- Graphic interface

Constraint implications

Had to be obvious to use	

Constraint implications

Had to be obvious to use	Down to my design

Constraint implications

Had to be obvious to use	Down to my design
No data sent off device	

Constraint implications

Had to be obvious to use	Down to my design
No data sent off device	Javascript? WASM?

Constraint implications

Had to be obvious to use	Down to my design
No data sent off device	Javascript? WASM?
Appear instantaneous to the user	

Constraint implications

Had to be obvious to use	Down to my design
No data sent off device	Javascript? WASM?
Appear instantaneous to the user	Calculation on device

Constraint implications

Had to be obvious to use	Down to my design
No data sent off device	Javascript? WASM?
Appear instantaneous to the user	Calculation on device
Graphic interface	

Constraint implications

Had to be obvious to use	Down to my design
No data sent off device	Javascript? WASM?
Appear instantaneous to the user	Calculation on device
Graphic interface	Typescript and React?

Because I had been doing more object oriented stuff I chose TypeScript which is a superset of Javascript with types. React is a library for building user interfaces in TypeScript and seemed to be the most popular framework for building web applications, so I chose that. I didn't know either TypeScript or React before and I wanted this to be quick so I decided to use LLM to help me.

Contents

LLM-Assisted Development Process

- Work out the logic from the paper
- Design the calculation part
- Design the graphical part

Logic from the paper

- Supply the paper and ask LLM to extract the parameters
- Confirm the output manually
- Extract the regression equation from the paper
- Use the equation to calculate the risk and check against examples in the paper

This was pretty straightforward and I got the logic from the paper.

Design the calculation part

- I checked the maths separately
- Asked the LLM to create code to reproduce the calculation
- Checked against hand calculations and examples in the paper

This was straightforward.

Design the graphical part

- Initially asked the LLM to create code to produce a graphic of a clock face
- Added highlighting and the ability to select segments
- Added the logic around the selection of tears and detachment

This was far from straightforward.

(Re-)Design the graphical part

- Added highlighting and the ability to select segments
- Added the logic around the selection of tears and detachment

This was far from straightforward!

Contents

LLM Integration Challenges

Key Challenges

- If something wasn't quite right there was a tendency to go round in circles
- Hard to know when to stop
- Need for structured approach to avoid endless iterations

Eventual LLM working pattern

Documentation Strategy

- Have a "meta" folder to contain:
  - Design documents
  - Examples
  - Documents generated by the LLM
  - A "next steps" document

Testing Approach

- Set up a testing framework:
  - Make minimal tests to start
  - Have the tests run after every change
  - Use the tests to drive the development

Using tests to drive development

As an example, the way things displayed had to be different for mobile and desktop. Working in parallel was too complicated so I got it working in one format and then wrote tests to check that the other format worked. These obviously failed, but I could feed back the results of the failing tests to the LLM until the tests passed.

Contents

Testing Strategy Overview

1. Test Organization
   - Tests organized by view (mobile/desktop) and responsibility
   - Clear separation between component, utility, and hook tests
   - Tests grouped using describe blocks with clear descriptions
   - Proper test isolation and setup/teardown

Test-Driven Development cycle

Key TDD Steps

1. Write/update test for a single change
2. Verify test fails (red)
3. Make minimal code changes
4. Verify test passes (green)
5. Refactor if needed
6. Get approval before next change

Component Testing Structure Example

Test Organization

- RetinalCalculator tests split into:
  - Container tests (view switching, layout)
  - Mobile view tests (touch interactions)
  - Desktop view tests (mouse interactions)
  - Shared functionality tests

Testing Principles Phase 1

- Never modify source code to make tests pass
- Document needed changes in meta/*.md files
- Skip failing tests with detailed comments
- Wait for explicit approval before implementation changes
- Maintain test isolation and clear boundaries

I woudl run a session on an aspect with the intructions for the LLM as above. When I had read and understood the needed changes, I would switch to Phase 2.

Testing Principles Phase 2

- Never modify code to make tests pass
- Make the changes described in meta/*.md files
- Reintroduce previously failing tests when addressed
- Wait for explicit approval before implementation changes
- Maintain code isolation and clear boundaries

Testing Strategy

- Comprehensive test cases
- Edge case handling
- Debug capabilities
- Automated validation

Contents

Key Development Achievements

Technical Implementation

- React-based implementation
- Modular component structure
- Utility functions for time conversion
- Robust testing framework

Future Enhancements

Planned Improvements

- Additional time formats
- Enhanced user interactions
- Visual improvements
- Extended test coverage