Testing

Test Test Testing!

Today's Agenda

- More talks on testing
- Test-Driven Development
- Monster! Exercise
- Project Planning with Testing
- Connect-4 Exercise

Testing, a list

The order of a properly tested thing:

- unittests
  - make sure all the pieces work on their own
- integration tests
  - make sure all the pieces work together
- verification& validation tests
  - make sure the sum of pieces gives you correct results

Integration Testing

- Individual modules and units are assembled and tested together.
- Integration testing ensures that units that rely on other units are using those units
  - main() generally needs to be integration tested, as it is the joining method of all the other methods
  - integration testing is also generally automated, though it’s tricker to write

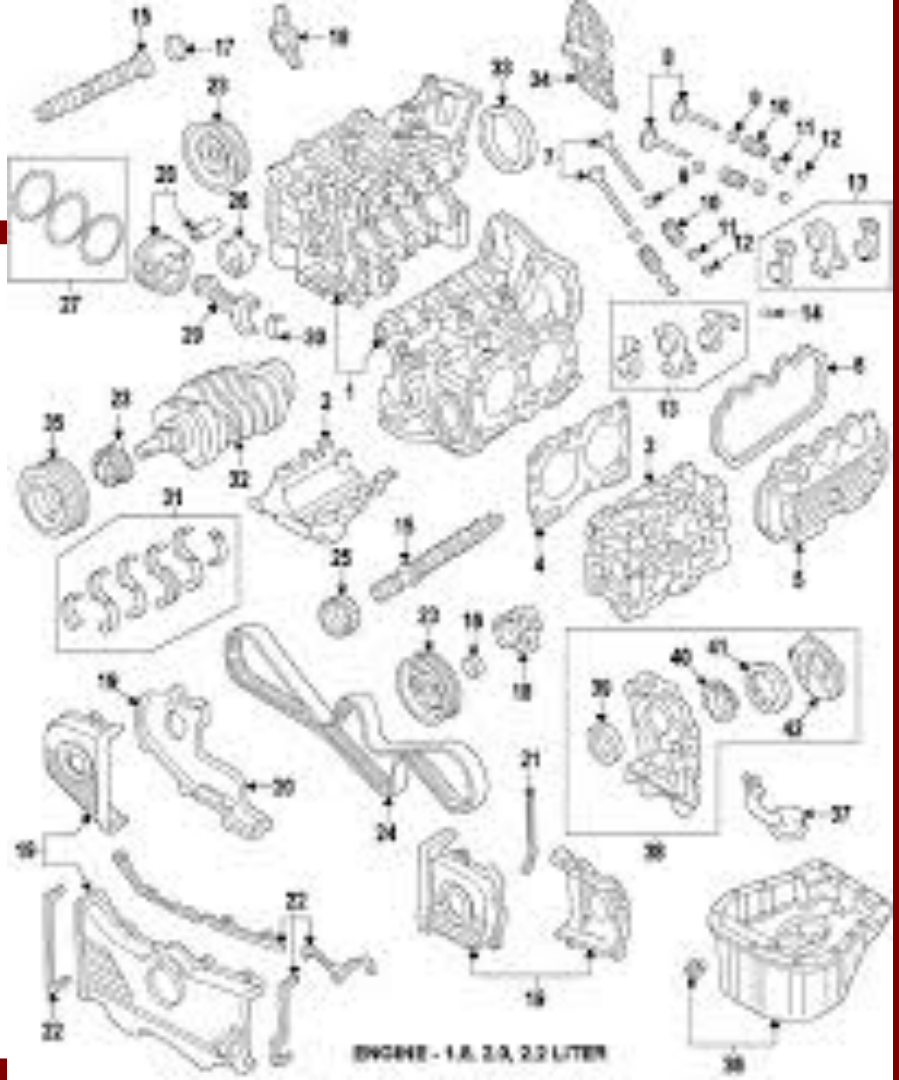
Verification & Validation Tests

- process of checking that the software as a whole meets specifications and does what it’ s supposed to do
  - Automated testing for this is very difficult to do, so manual V&V is more common
- Verification: Did you build it right?
- Validation: Did you build the right thing?

My Car's Engine

Unittest:

- Does each of these pieces work like they're supposed to?



My Car's Engine

Integration Test:

- Do all the pieces work together?

My Car's Engine

Verification & Validation:

- Did we make the engine right?
- Is it a Subaru Engine?
- Does it actually do what it's supposed to do?

Vroom Vroom

Test Driven Development

Short, quick development bursts that revolve around writing tests, then writing the least amount of code needed to pass those tests.

- No more, no less

TDD in Short

Once your main Design is done (requirements and UML)

1. pick a small unit to work on
2. write a test
3. run all the tests
4. write code until all your tests pass
5. clean up your code
6. repeat from 1

Benefits of TDD

- Developers are more productive, as there are clear goals and next-steps in place
- Less debugging is needed
- Can help ensure that you're coding to design, not to "until it works"
- Yes, you're writing more code (including tests), but you're spending less time fixing code

Monster!

Create a Monster based off this UML using TDD:

Monster
name: String (set on init) status: “Out of Tokyo” health: 10 victory_points: 0
reset() – reset Monster to initial stats in_tokyo():Bool – returns True if Monster status is “In Tokyo” flee():Bool – prompts monster to see if they want to flee Tokyo. If “y”, return True. If “n”, return false heal(int) – add the passed int to the Monster’s health, up to but not exceeding 10 attack(int):int – subtract the passed int from the Monster’s health, returning health. If health is <=0, set status to “K.O.’d” score(int):int – add passed int to Monster’s victory_points, and return victory_points. If a Monster’s VP >= 20, set status to “WINNING”

Model View Controller (MVC)

- software application architecture pattern for programs that have some sort of user interface
- divides the software into three interconnected parts
  - model
  - view
  - controller

Model

- stores the data that the controller leverages and the view displays
- you can have multiple models in a program

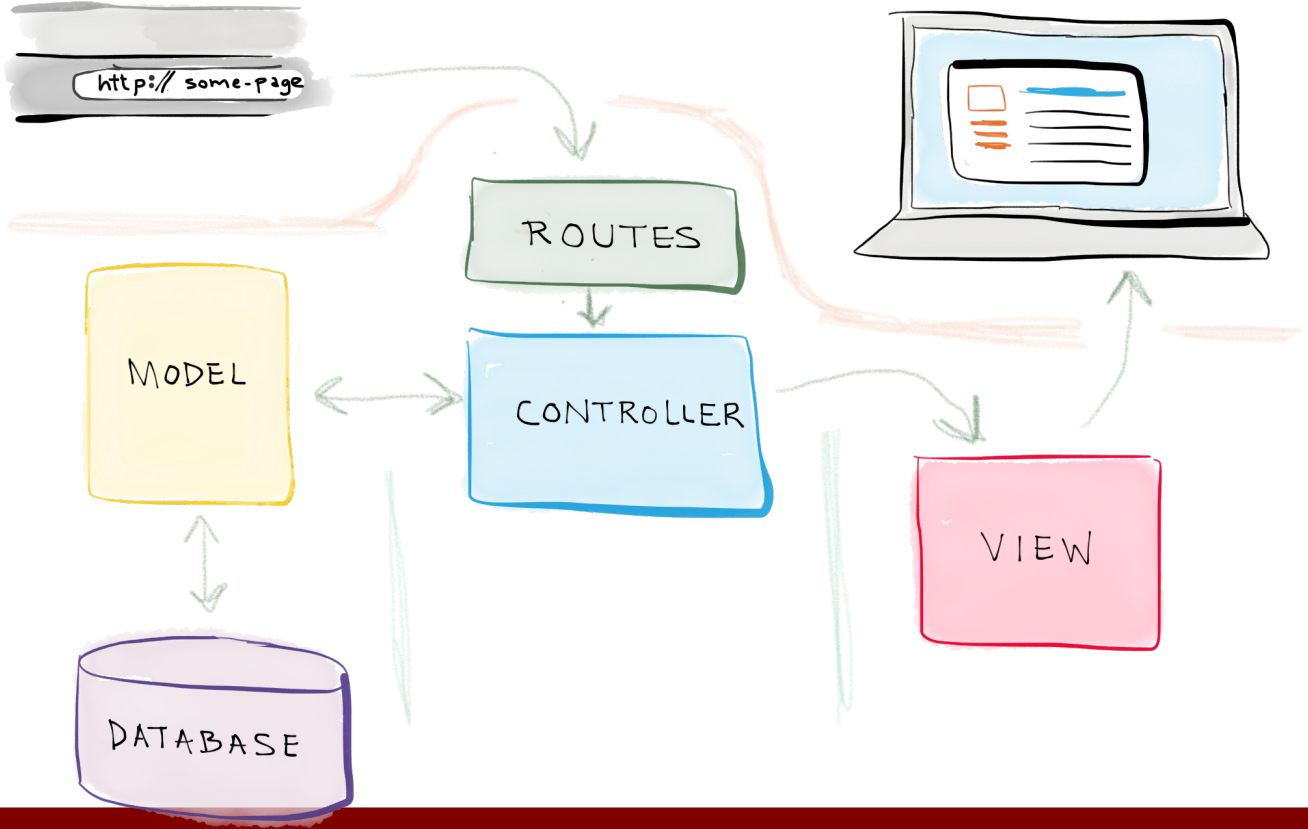
View

- Controller fetches the data from the Model and then uses the View to display it
- Views should be swappable; you could have multiple views, and which you use depends on the Controller

Controller

- Does most the program's heavy lifting; it *controls* the program
- You can only have ONE Controller
- Is responsible for all of the communication between your Models and Views

A Pretty Picture



Design & Develop using TDD

- Planning: What you're going to do
- Analysis: What are the requirements?
- Design: How are you going to do it?
- Test Driven Development: Make it with tests!
- Validation & Verification
- Ship*
- Profit**

*This is I grade your assignments **Not guaranteed

Partners!!

These are the partner assignments for the next exercise:

- Summer & Attila
- Max & Daniel
- John & Nehemiah
- Michael & Chelsea

Remember, you’re being graded on your design, development, and co-working skills

Connect 4 (via Wikipedia)

A two-player connection game in which the players first choose a color and then take turns dropping colored discs from the top into a seven-column, six-row vertically suspended grid. The pieces fall straight down, occupying the next available space within the column. The objective of the game is to connect four of one's own discs of the same color next to each other vertically, horizontally, or diagonally before your opponent.