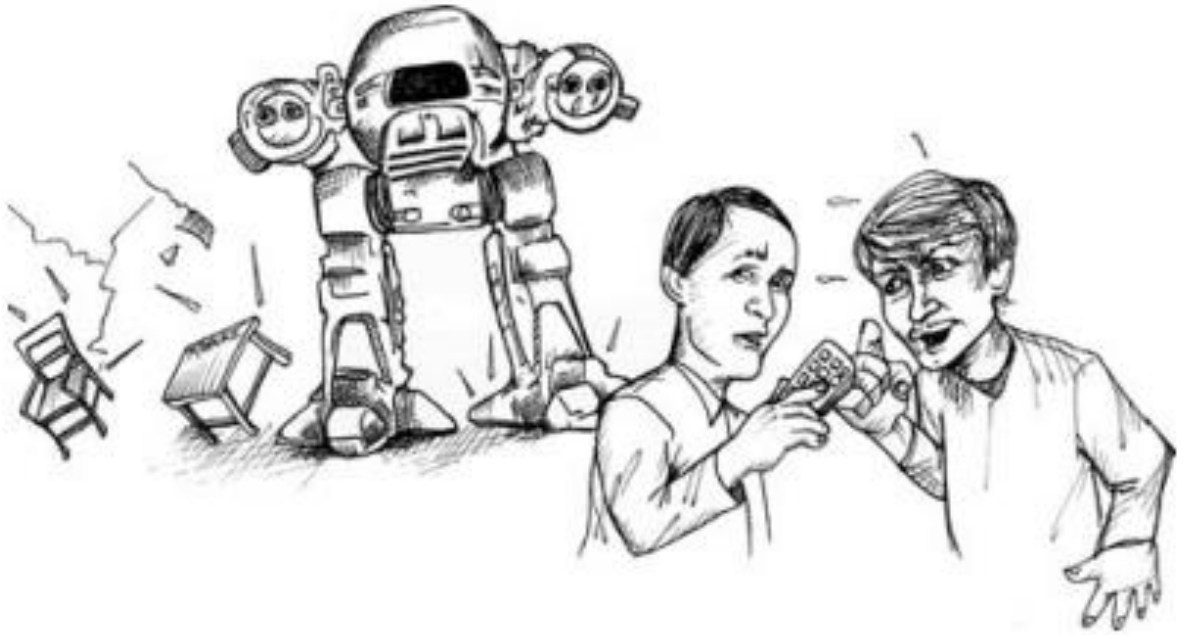
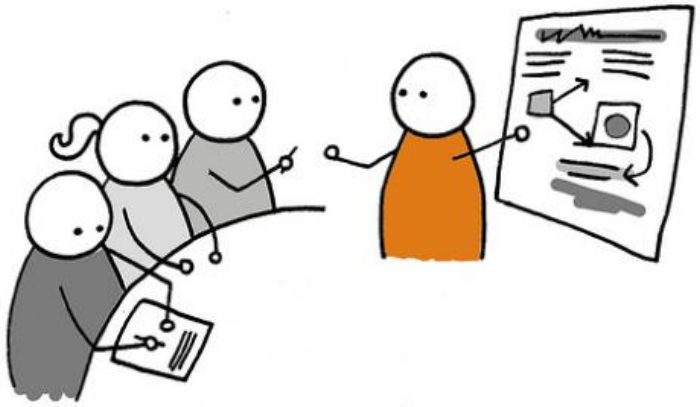
Chapter 7: Acceptance Testing



Tuesday, February 11, 2025

Communicating Requirements & Premature Precision

- Most common communication issues between programmers and business is the requirements.
- Communication of requirements is extremely difficult
- Premature Precision
  - Business people want to know exactly what they are going to get before the authorize the project
  - Developers want to know exactly what they are supposed to deliver before they estimate the project

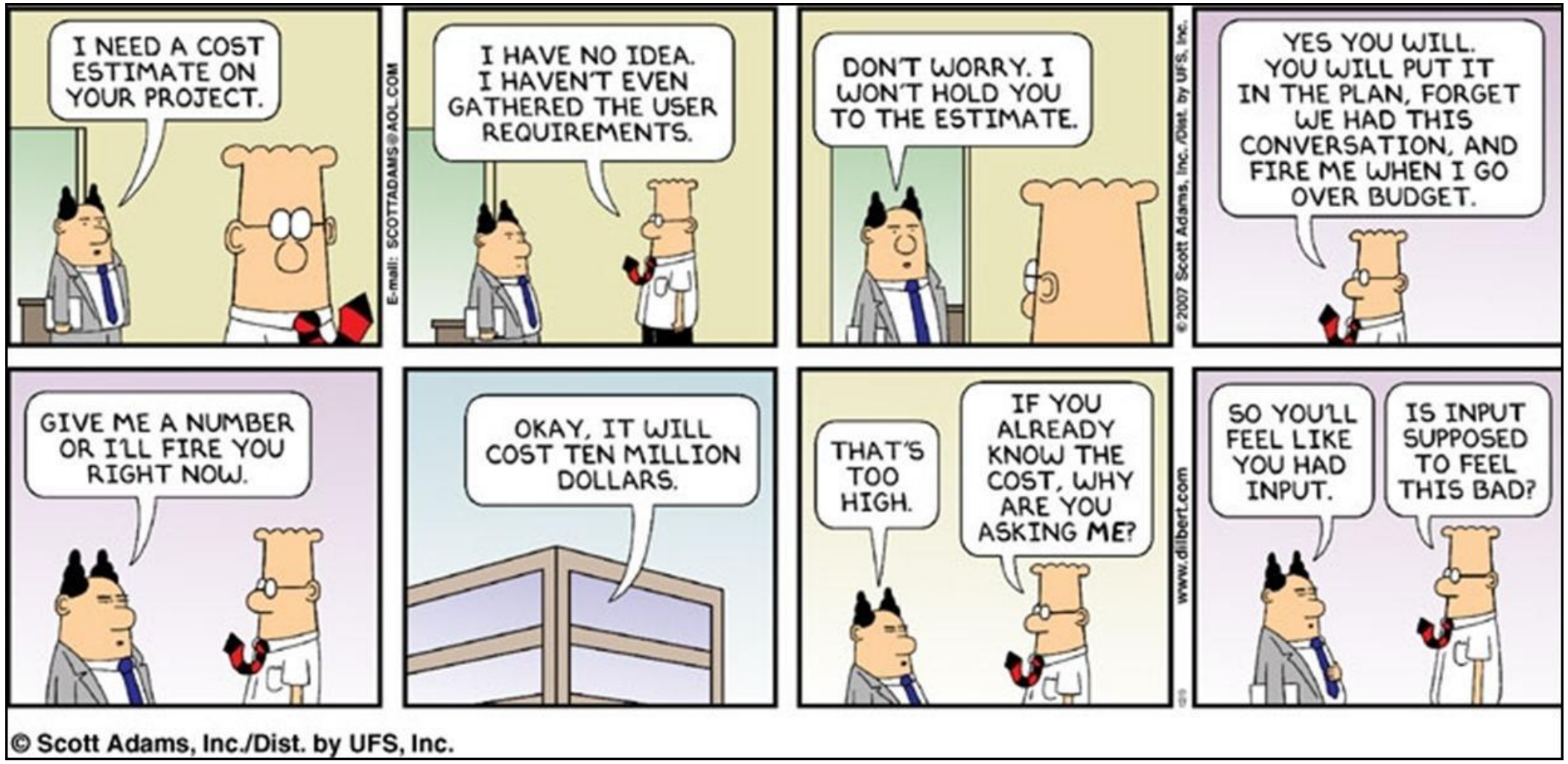


The Uncertainty Principle

- Things appear different on paper than they do in the system
- When you demonstrate a feature to the business, it gives them more information than they had before, and that new information impacts how they see the whole system.
- The more precise you make your requirements, the less relevant stakeholders become as the system is implemented.

"Based on my observations, this is how the Business Analysts do their job."

Estimation Anxiety



Late Ambiguity

- Professional developers don't flesh out a requirement until they are just about to develop it, which may lead to late ambiguity

> *"An ambiguity in a requirements document represents an argument amongst the stakeholders."*
>
> *-Tom DeMarco, XP Immersion*

- Stakeholders simply assume that their readers know what they mean

Electron Tic-Tac-Toe…

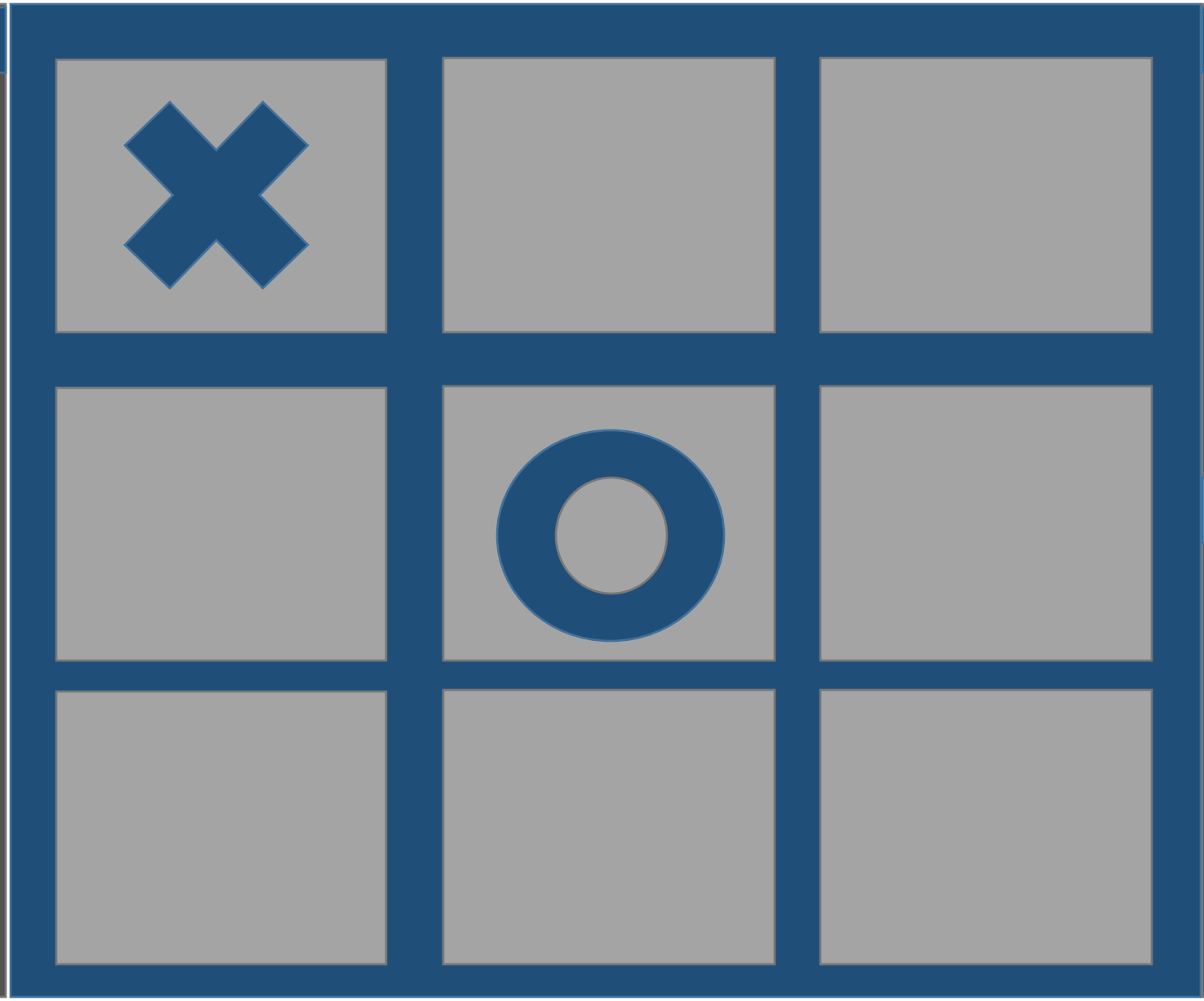
- Please create me a UI for an electronic Tic-Tac-Toe Game
  - There should be 2 players
  - We should track who wins
  - Each Player should be able to place an x & o on the screen

3 minutes…

Tic-Tac-Toe

Leader Board

Lindsey……..100
Tom……..75
Robin……10
Ryan……5
Emily……2



Lindsey X

It is your turn…

Turns Remaining: 4

Number of Games Won: 5

Tom O

Turns Remaining: 3

Number of Games Won: 5

Did you detect the ambiguity?

- I said I wanted to track player wins but I didn’t say for how long
- I was not specific enough for what I wanted so you had to make assumptions
- Responsibility of all professional developers and stake holders to make sure that all ambiguity is removed from the requirements

Acceptance Testing

- Acceptance tests are tests written by a collaboration of the stakeholders and the programmers in order to define when a requirement is done
- What is the definition of “Done”?
  - For professional developers done means all code has been written, all tests pass, and the stakeholders and QA have accepted
  - Automated tests pass

Example of an Acceptance Test

- *As a tic-tac-toe player, I want to be able to track how many games I have won so my score can be placed on the leader board*

  - Players should be able to log in so we can identify them as players
  - When a player wins a game a win should be added to their total wins
  - The leader board should display the top 5 players with the most total wins

Acceptance Testing Cont.

- Communication
  - The purpose of acceptance tests is communication, clarity, and precision.
  - By agreeing to them, the developers, stakeholders, and testers all understand what the plan for the system behavior is.
- Automation
  - Acceptance tests should always be automated
  - The cost of automating acceptance tests is so small in comparison to the cost of executing manual test plans that it makes no economic sense to write scripts for humans to execute.
- Extra Work
  - Specifying at this level of detail is the only way we, as programmers, can know what “done” means.
  - Specifying at this level of detail is the only way that the stakeholders can ensure that the system they are paying for really does what they need
  - Specifying at this level of detail is the only way to successfully automate the tests.

Who Writes Acceptance Tests?

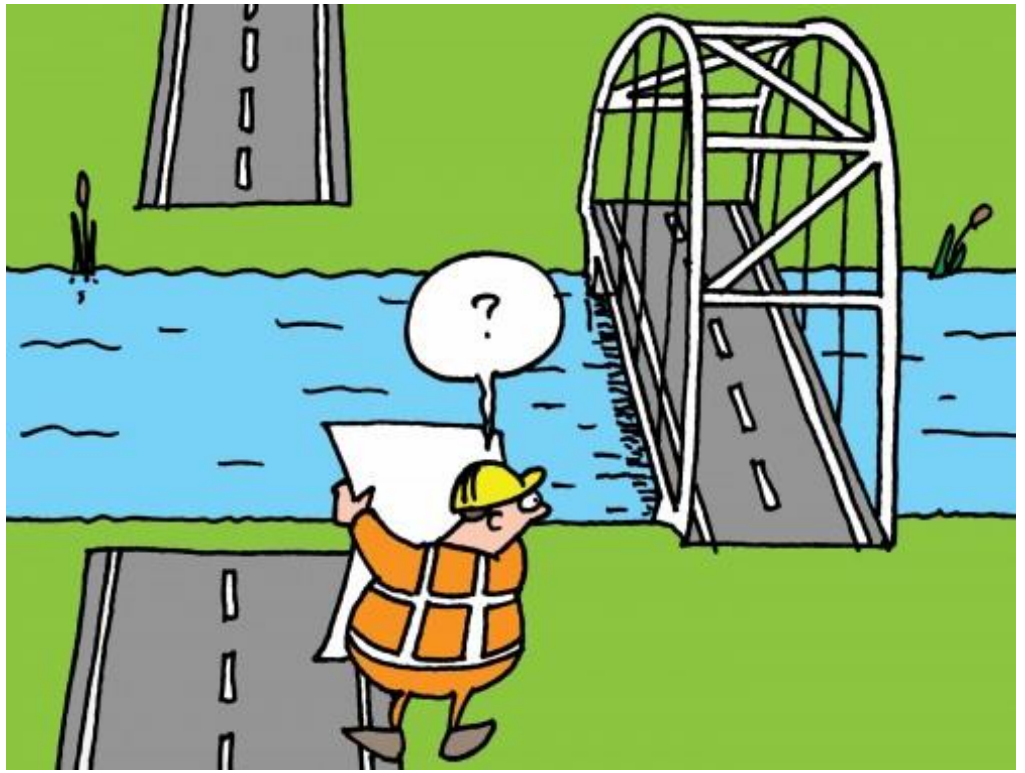
- Ideally, the stakeholders and QA would collaborate to write these tests, and developers would review them for consistency
- Business analysts write the “happy path” versions of the tests, because those tests describe the features that have business value.
- QA typically writes the “unhappy path” tests, the boundary conditions, exceptions, and corner cases. This is because QA’s job is to help think about what can go wrong.

When do you write Acceptance Tests?

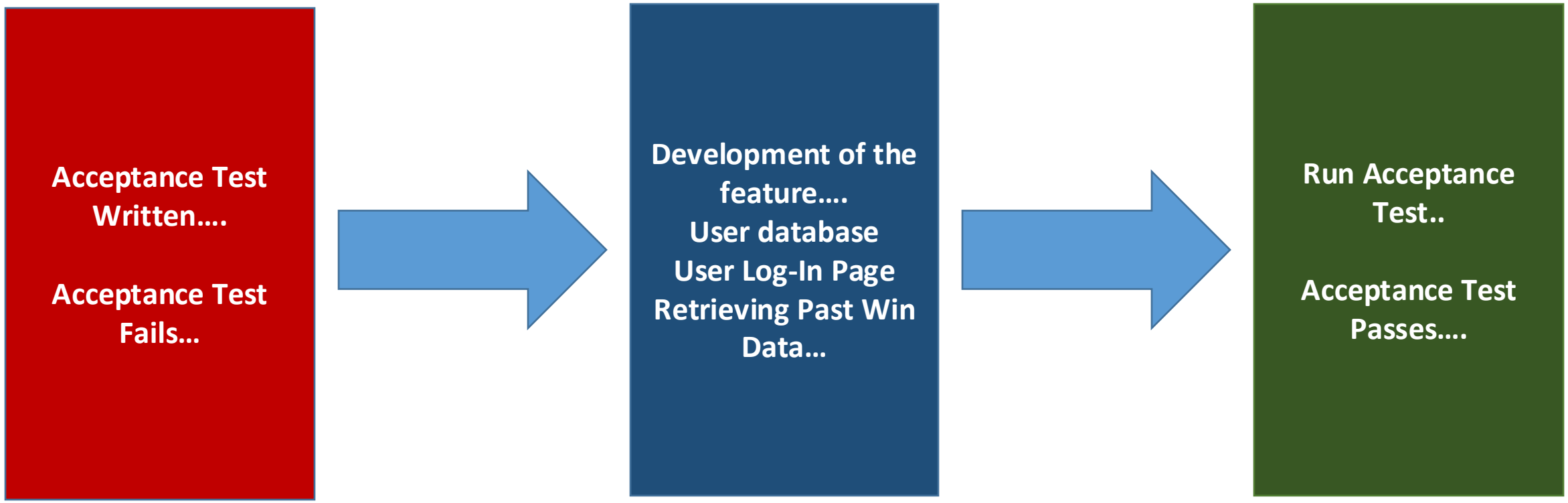
- Acceptance tests should be written as late as possible, typically a few days before the feature is implemented.
- In Agile projects, the tests are written after the features have been selected for the next Iteration or Sprint.
- First Day of Iteration: few acceptance tests
- More completed each day until the midpoint of the interaction when all of them should be ready

The Developer Role

- Implementation work on a feature begins when the acceptance tests for that feature are ready.
- The developers execute the acceptance tests for the new feature and see how they fail.
- Then they work to connect the acceptance test to the system, and then start making the test pass by implementing the desired feature.
- That it is the developer's job to connect the acceptance tests to the system, and then to make those tests pass.



Players should be able to log in so we can identify them as players….



The Negotiation and Passive Aggression

- Sometimes the tests as written don't make a lot of sense once you start implementing them.
  - Too complicated
  - Awkward
  - Contains silly assumptions
  - Or they might just be wrong
- As a professional, it is your job to negotiate with the test author for a better test.
- What you should never do is take the passive-aggressive option and say to yourself, "Well, that's what the test says, so that's what I'm going to do."

Acceptance Tests and Unit Test

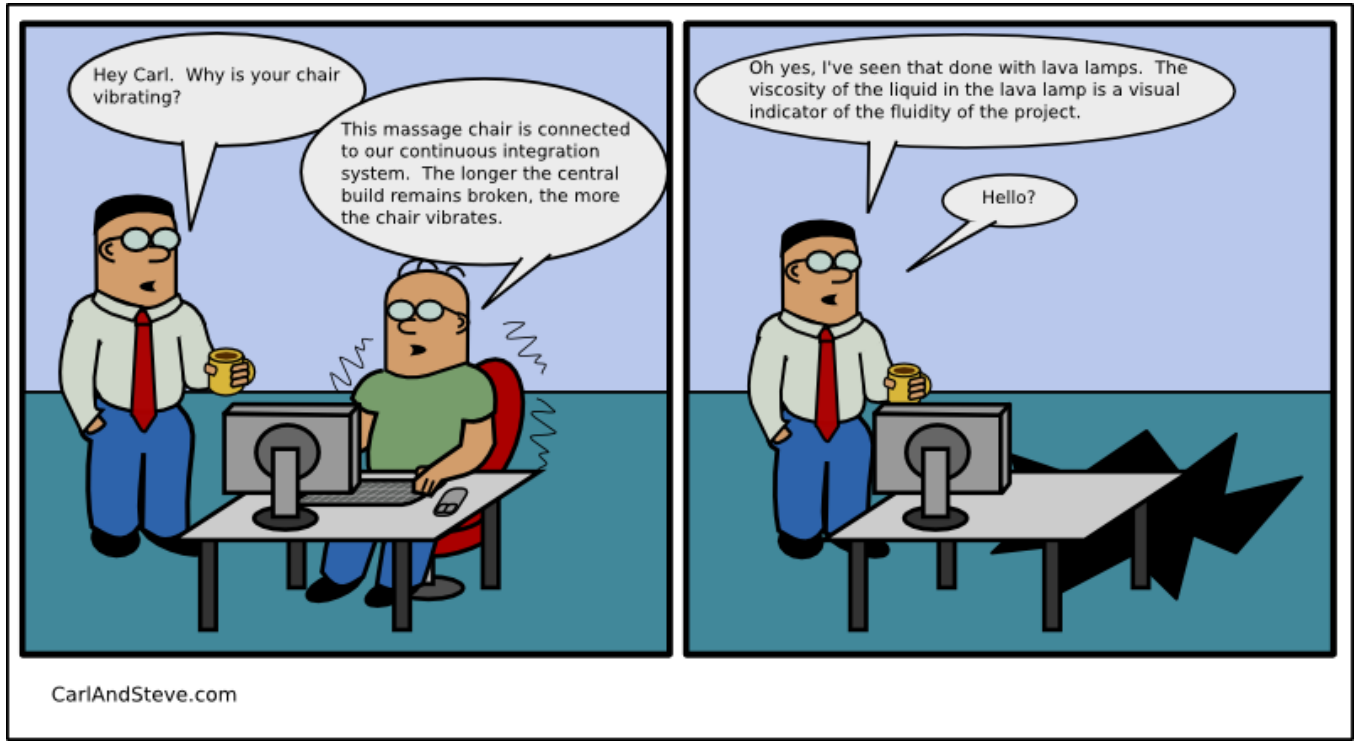
- Acceptance tests are not unit tests - Unit tests are written by programmers for programmers
- Acceptance tests are written by the business for the business
  - They are formal requirements documents that specify how the system should behave from the business' point of view.
- May test the same things, just through different mechanisms and pathways
- Primary purpose is to formally document the design, structure, and behavior of the system.

GUIs and Other Complications

- GUIs are hard to test because the aesthetics are subjective and therefore volatile
- Design the system so that you can treat the GUI as though it were an API rather than a set of buttons, sliders, grids, and menus.
- The layout, format, and workflow of the GUI will change for aesthetic and efficiency reasons, but the underlying capability of the GUI will remain the same despite these changes.
  - when writing acceptance tests for a GUI you take advantage of the underlying abstractions that don’t change very frequently.

Continuous Integration

- Make sure that all your unit tests and acceptance tests are run several times per day in a continuous integration system
- Should never fail, if they do the whole team should stop what they are doing and focus on getting the broken tests to pass again



Chapter 7: Key Points

- Communication about details is hard, especially true for programmers and stakeholders communicating about the details of an application.
- The only way I know of to effectively eliminate communication errors between programmers and stakeholders is to write automated acceptance tests.
- Unit tests are written for programmers by programs, acceptance tests are written for business by business