
# Leslie Brooks

## Finding Zero Bugs

PACIFIC NW SOFTWARE QUALITY CONFERENCE

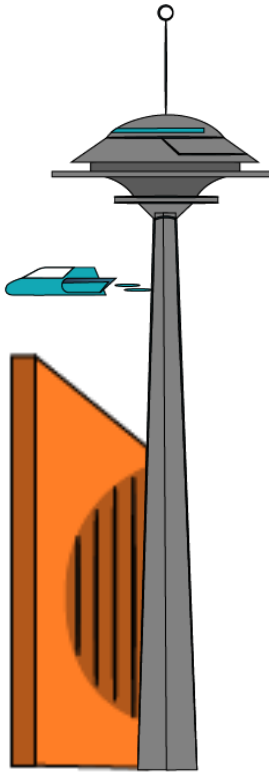
THE FUTURE IS NOW

PNSQC.ORG

OCTOBER 14-16 2024

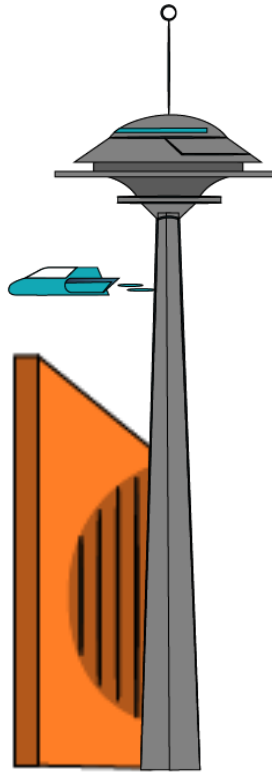
# Our World Today

- We write requirements
- The developers write code – and bugs
- The QA team is supposed to catch all of the bugs
- But some escape to Production



# The Future is Finding Zero Bugs

- Why do we write requirements?
- Why does writing requirements not always work?

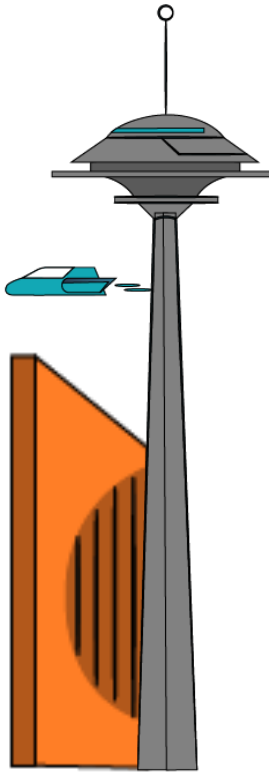


# The Problem

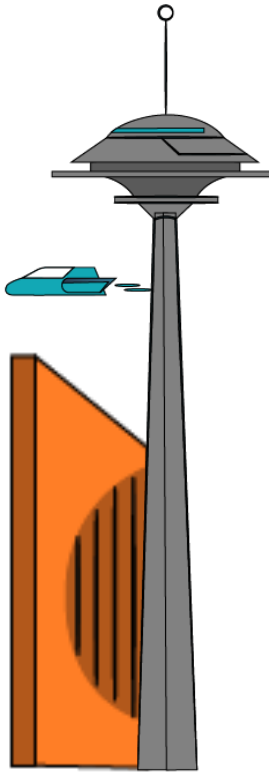
What the Customer Wants

What the Developers Think the Customer Wants

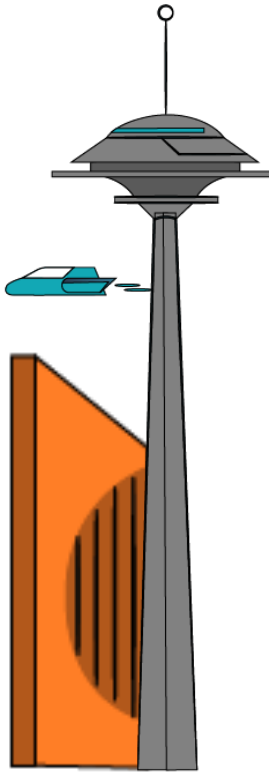
What QA Thinks the Customer Wants



# Little ‘b’ bugs

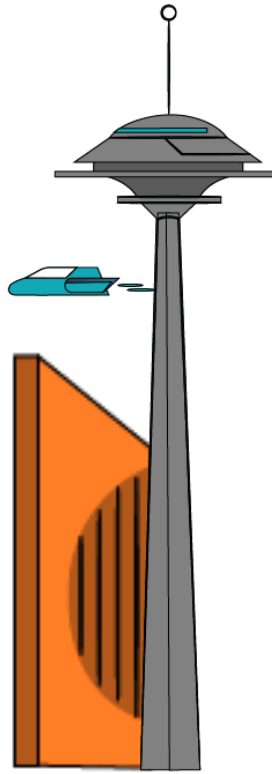


# Big ‘B’ Bugs



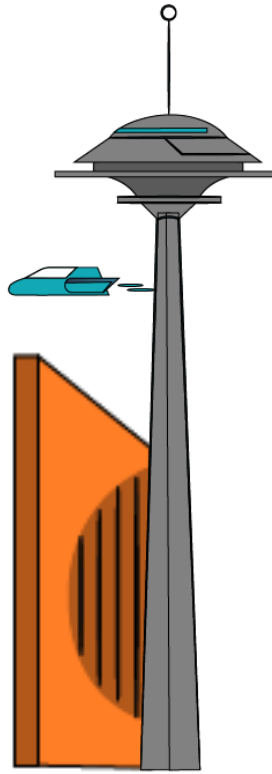
# The Future is Finding Zero Bugs

- Our current Agile methodologies:
  - Make it too easy to create bugs
  - Assume that we must write test cases to catch the bugs
  - Assume that we must have a QA team to catch the bugs
  - Assume that some bugs will escape into Production and be found later



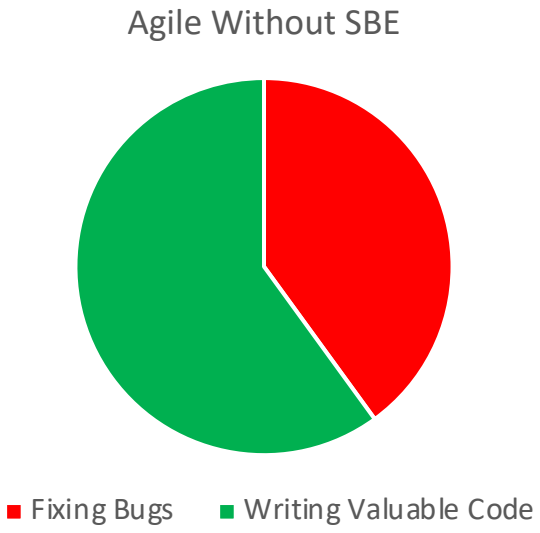
# The Future is Finding Zero Bugs

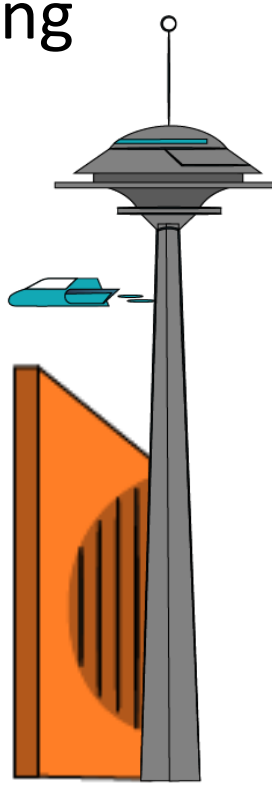
- Specification by Example (SBE) is an agile methodology that is designed to fix these problems
- Specification by Example (SBE) is a better name for BDD – Behavior Driven Development



# The Future is Finding Zero Bugs

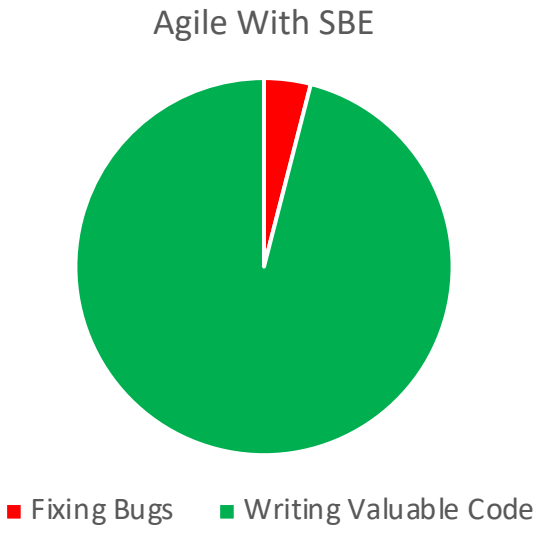
- At BNP Paribas, the developers were spending 35% of their time fixing bugs

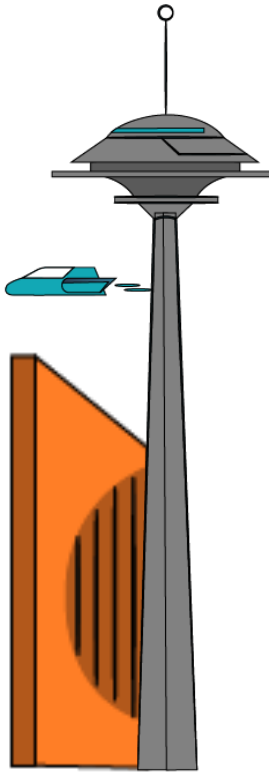


# The Future is Finding Zero Bugs

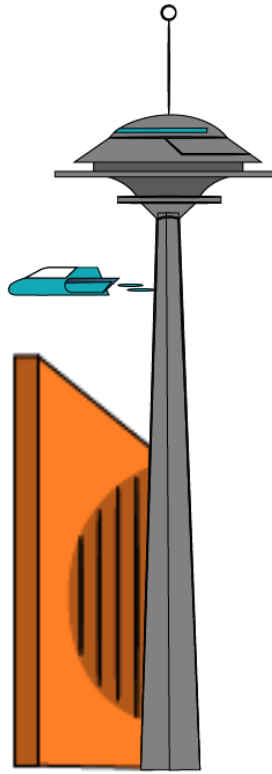
- Nine months later the developers were spending 4% of their time fixing bugs



# The Future is Finding Zero Bugs

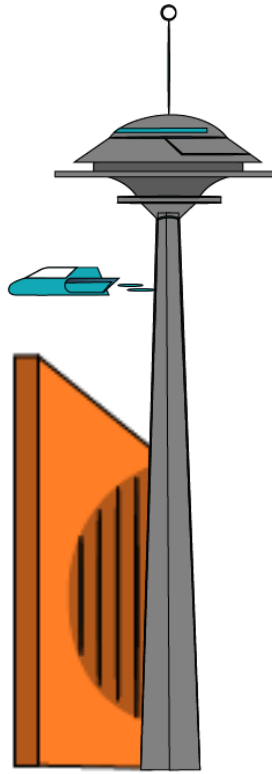
- At Raytheon I introduced SBE in a Data Science group



# The Future is Finding Zero Bugs

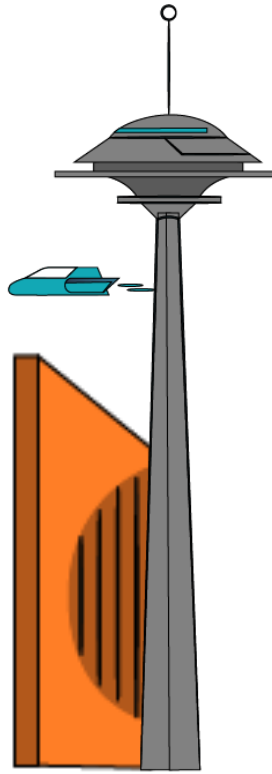
Over a one-year period, they delivered two bugs into QA

One Year



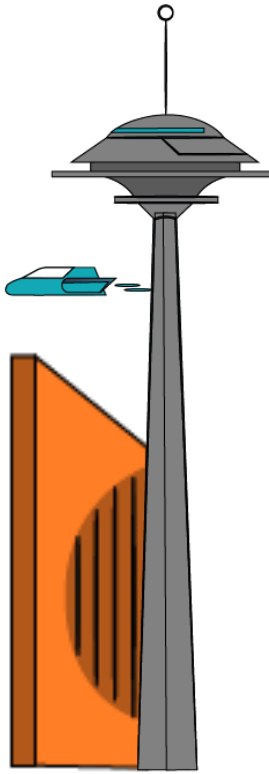
# The Future is Finding Zero Bugs

- How can SBE possibly invalidate all of these assumptions?
  - Traditional narrative requirements are too easy to misunderstand
  - Since we are going to have bugs, we must also have test cases
  - The test cases may have errors, or may get out of sync with the requirements
  - Our test cases won't be complete
  - We must have a QA team to execute those test cases

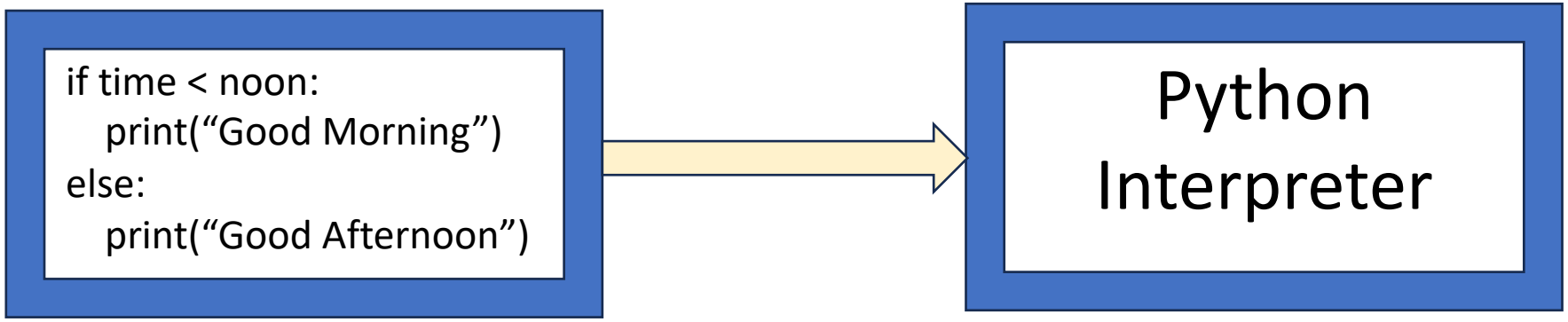


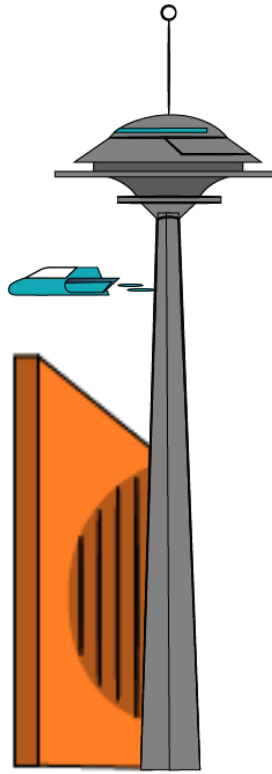
# The Future is Finding Zero Bugs

- How can the requirements also be the test cases?
- We must write executable requirements
- We must be able to run the requirements against the application



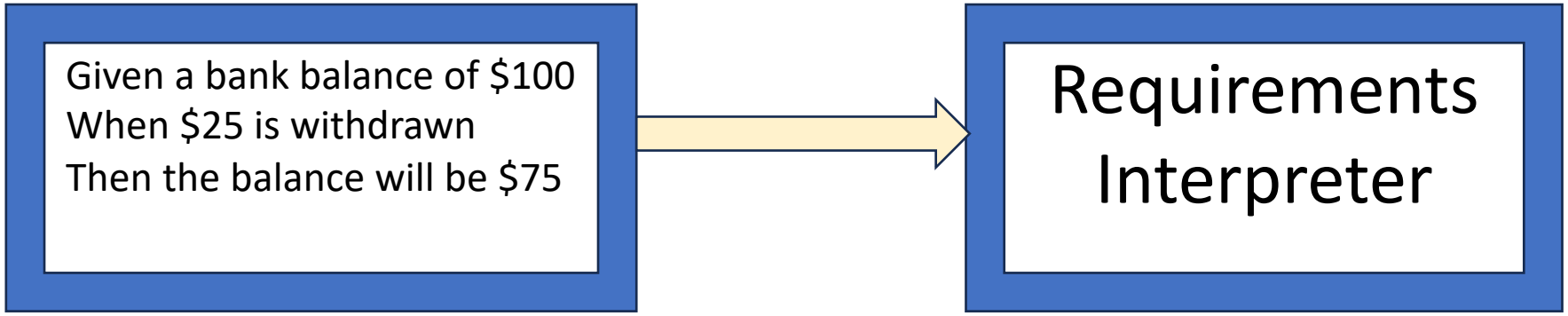
# What Are Executable Requirements?



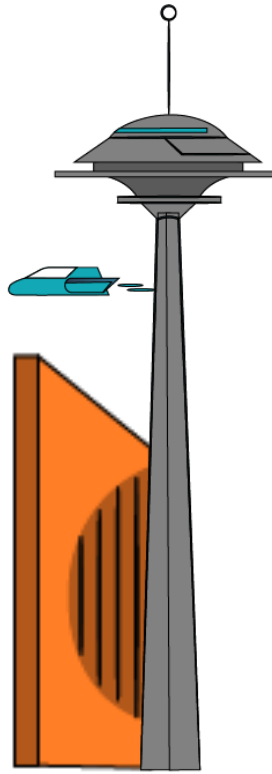



- This is executable Python
- By itself it doesn't do anything
- If we have a Python interpreter, this Python code becomes executable – we can run it

# What Are Executable Requirements?

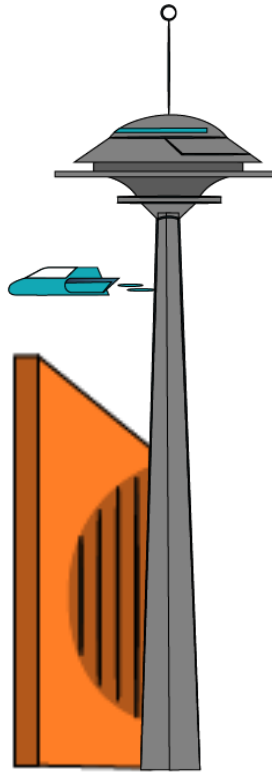



- This is an executable requirement
- By itself it doesn't do anything
- If we have a requirements interpreter, this requirement becomes executable – we can run it against our application



# Where Do We Get A Requirements Interpreter?

- We can’t ‘get’ a requirements interpreter, we must write it
- ‘Cucumber’ is an open source tool that makes it easy to write a requirements interpreter
- The requirements interpreter won’t be ready the first day



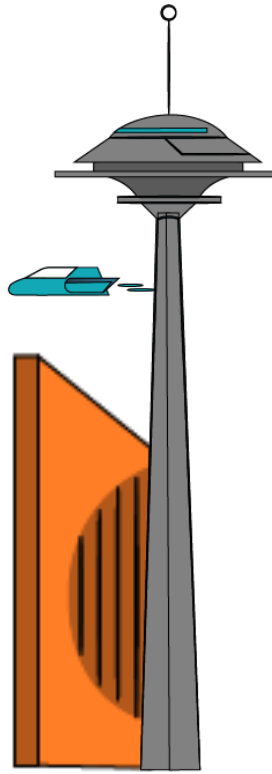
# The Future is Finding Zero Bugs

- Let’s look at a simple banking requirement

Scenario Outline: Withdrawing money debits the account balance
Given an account with a <Beginning_Balance> balance
When <Amount> is withdrawn
Then the balance will be <Ending_Balance>

**Examples:**

| Beginning_Balance | Amount | Ending Balance |
|---|---|---|
| $ 99 | $10 | $89 |
| $100 | $10 | $90 |



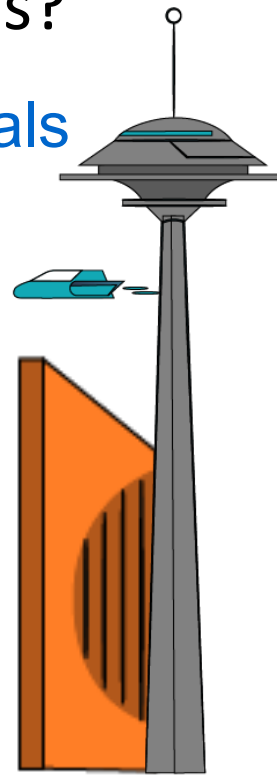
# The Future is Finding Zero Bugs

- What happens when we introduce a new concept – transaction fees?

  Scenario Outline: If the account balance is less than $100, withdrawals are charged a $5 transaction fee
  Given an account with a <Beginning_Balance> balance
  When <Amount> is withdrawn
  Then the balance will be <Ending_Balance>

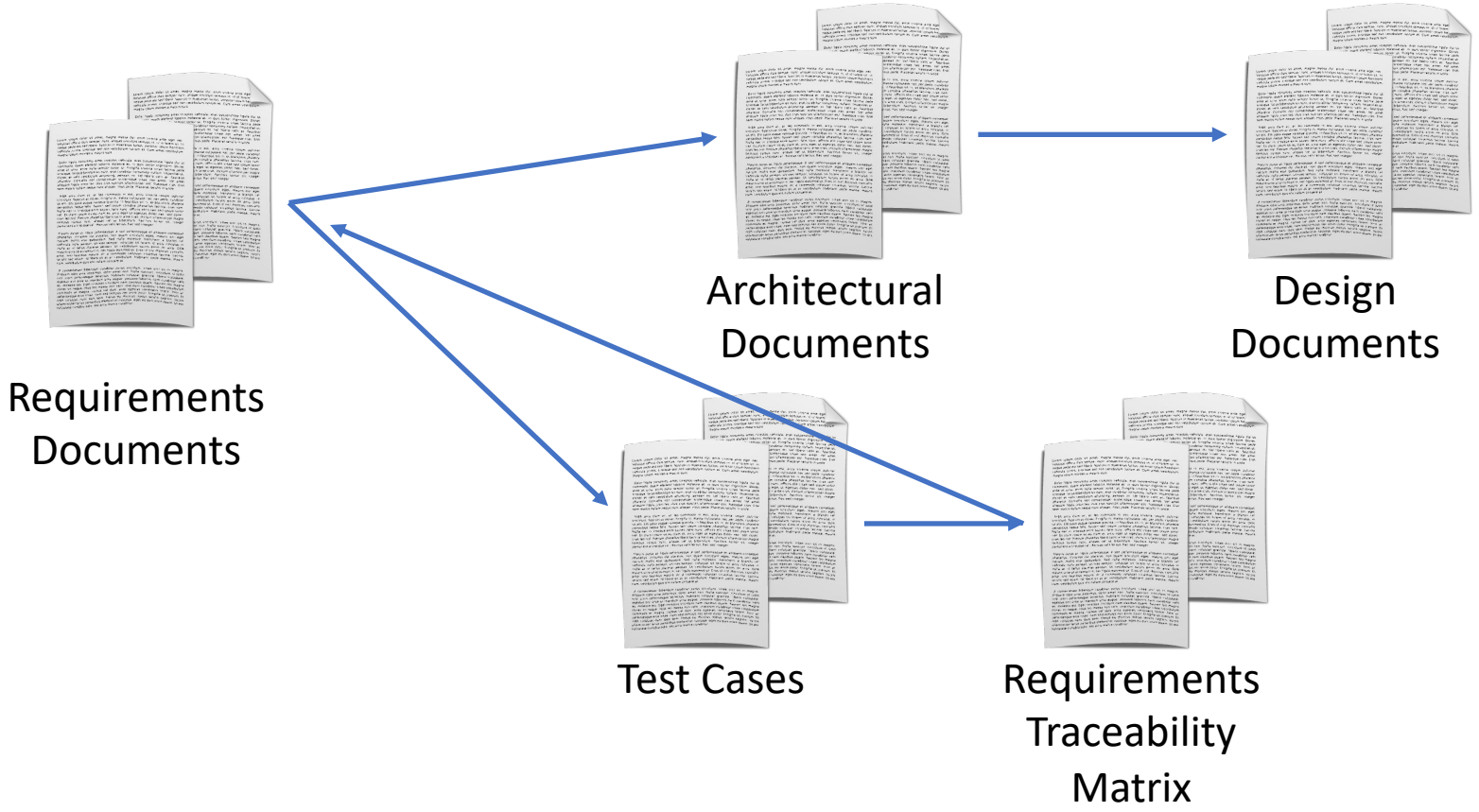
  **Examples:**

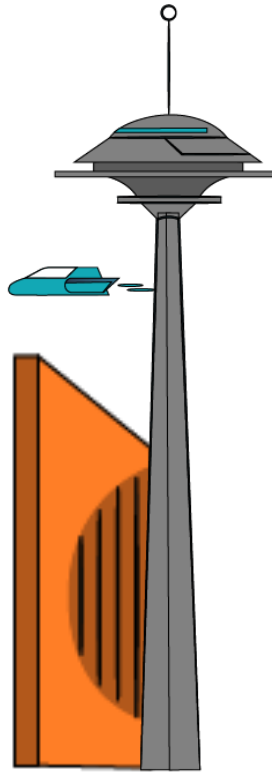
| Beginning_Balance | Amount | Ending Balance |
|---|---|---|
| $ 99 | $10 | $84 |
| $100 | $10 | $90 |



# The Future is Finding Zero Bugs

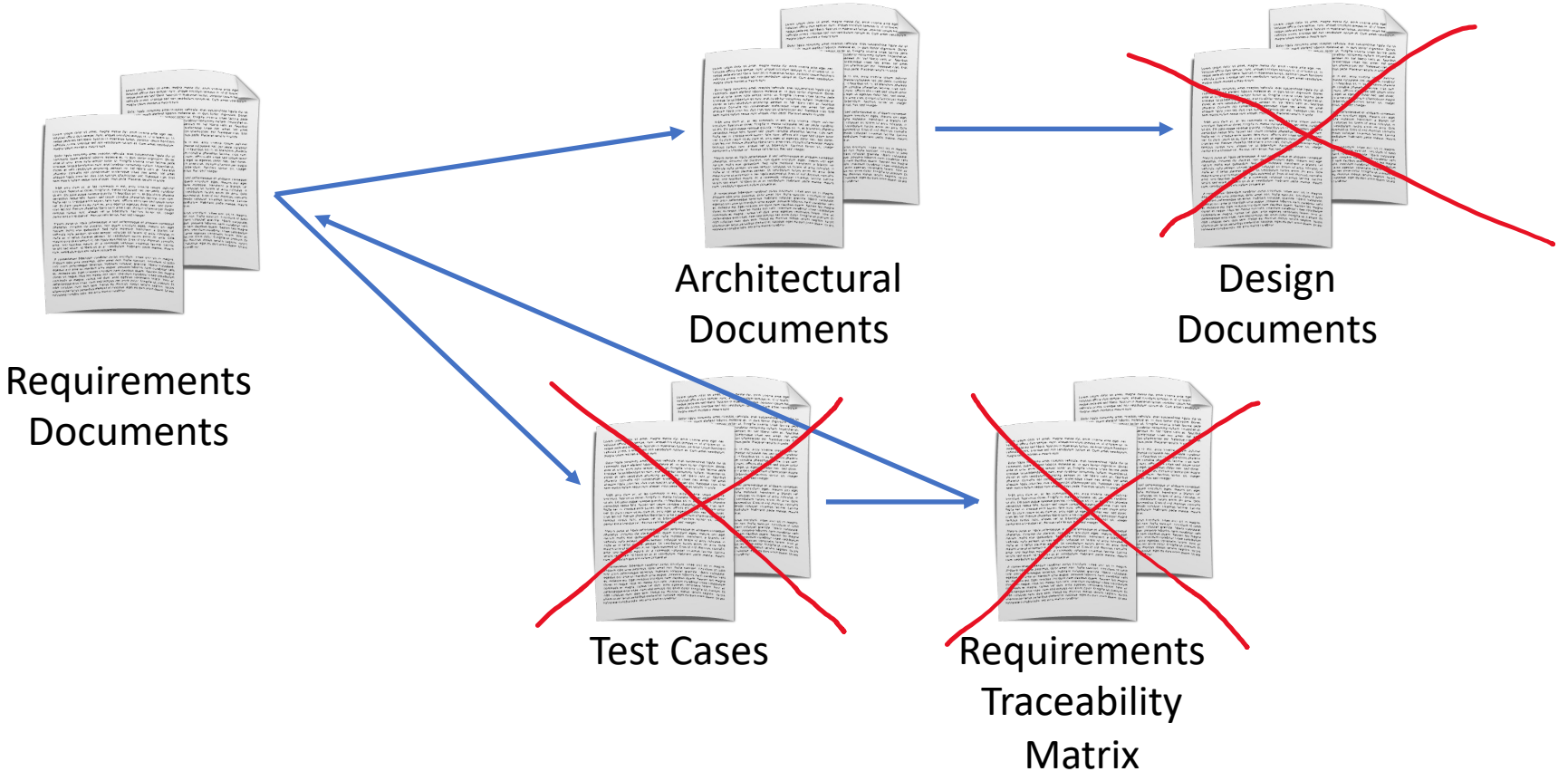
- Traditionally, a development project requires multiple documents:

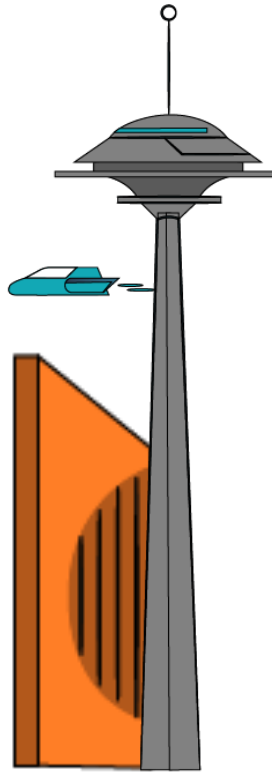


# The Future is Finding Zero Bugs

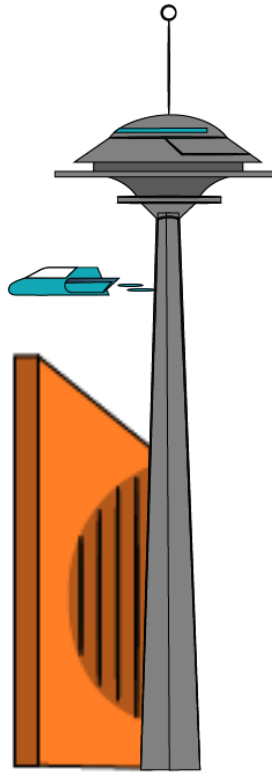
- SBE has just two documents:



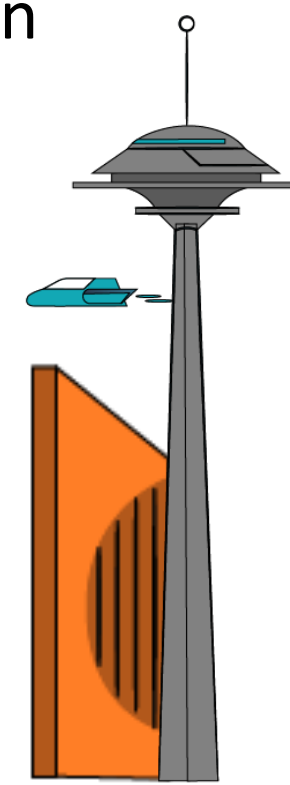
# How Can We Implement This Methodology?

- We must have (at least) five things:
  1) Knowledge
  2) Discipline
  3) Practice
  4) Support
  5) Time



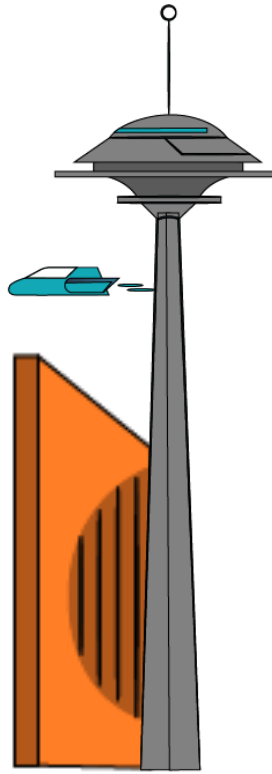
# SBE – Two Minute Drill

- SBE says that we must write requirements in a Given – When – Then format:
  - Given these preconditions
  - When this event happens
  - Then this is the expected result
- This Given – When – Then format is called Gherkin



# SBE – Two Minute Drill

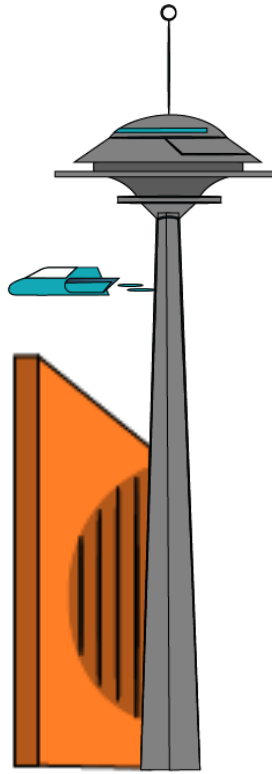
1) Write a business rule from a user story:
   - Business Rule: Bills more than 60 days late will be charged a 5% penalty
   - Business Rule: Late bills will be charged a penalty on a sliding scale
2) Illustrate the business rule with the Gherkin
3) Illustrate the Gherkin with concrete examples



# SBE – Two Minute Drill

- User Story: As the accounting department, I want late bills to be charged a 5% penalty
- Business rule: Bills more than 60 days late will be charged a 5% penalty

**Scenario:** Bills more than 60 days late will be charged a 5% penalty
Given a bill that is 61 days past due
When the total is calculated
Then a 5% penalty will be added



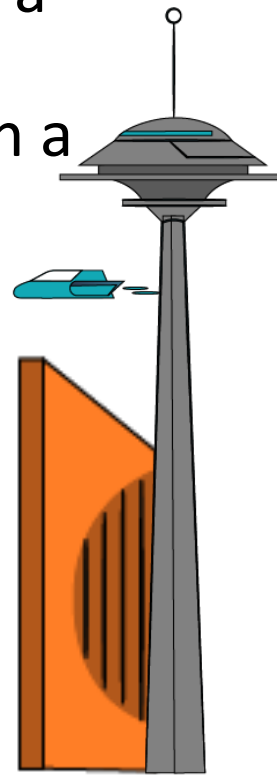
# SBE – Two Minute Drill

- User Story: As the accounting department, I want late bills to be charged a penalty on a sliding scale
- Business rule: Bills more than 60 days late will be charged a penalty on a sliding scale

**Scenario Outline:** Late bills will be charged a penalty on a sliding scale
Given a bill that is <Days_Past_Due> days past due
When the total is calculated
Then a <Penalty> penalty will be added

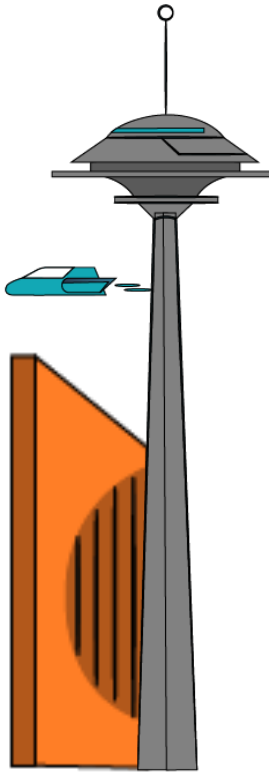
**Examples:**

| Days_Past_Due | Penalty |
|---|---|
| 60 | 0% |
| 61 | 5% |
| 90 | 5% |
| 91 | 7% |



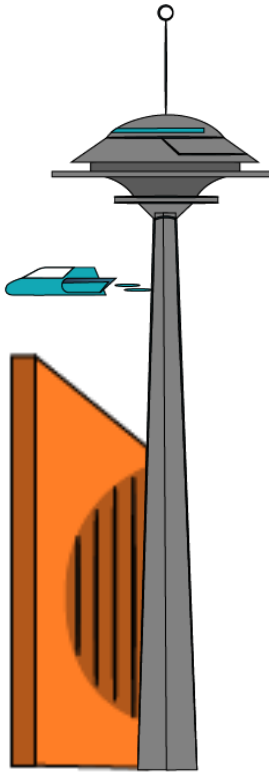
# SBE – Two Minute Drill

- What is so hard about this?
- We just write Given, When, Then
- Easy!
- We are creating a Domain Specific Language – a DSL
- This DSL is crafted for writing requirements about our business
- Creating a DSL is actually pretty easy
- Unless you want it to be a *good* DSL!



# SBE – Two Minute Drill

- The scenario title should state a business or product rule
  - The scenario title does not have to be concrete
- Illustrate the title with the Gherkin
- Illustrate the Gherkin with concrete examples
- Write the ‘Given’ in the past tense
  - It should talk about something that happened in the past
  - Or conditions that exist now
- Write the ‘When’ in the present tense
  - It should talk about an action or event taking place now
- Write the ‘Then’ in the future tense
  - It should talk about the outcome or expected result



# SBE – Two Minute Drill

- Always be consistent with the language you use!

  NOT This:

  **Given** an active customer account

  **Given** a customer account that is expired

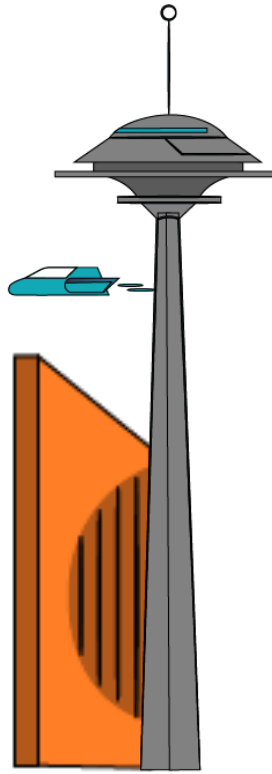
  **Given** a customer account that is no longer active

  But This:

  **Given** an active customer account

  **Given** an expired customer account

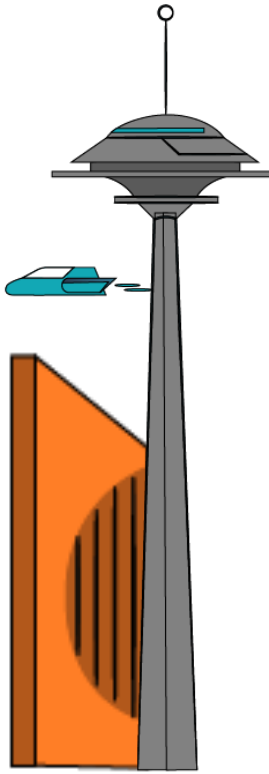
  **Given** an inactive customer account



# Additional Learning

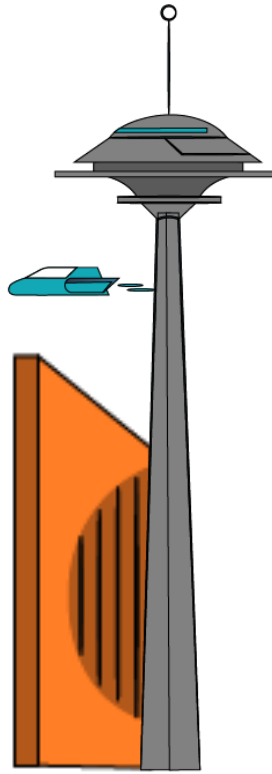
Before you Google ‘BDD Tutorial’, please read my blog post titled “Never Google ‘BDD Tutorial’”.

- https://thebddcoach.com/post/never-google-bdd-tutorial-because-you-will-get-bad-advice/



# Additional Learning

- Thebddcoach.com
- Cucumber.io
- Gojko.net
- Specflow.org
- ReqnRoll.net




# THANK YOU

PACIFIC NW SOFTWARE QUALITY CONFERENCE

THE FUTURE IS NOW

PNSQC.ORG

OCTOBER 14-16 2024