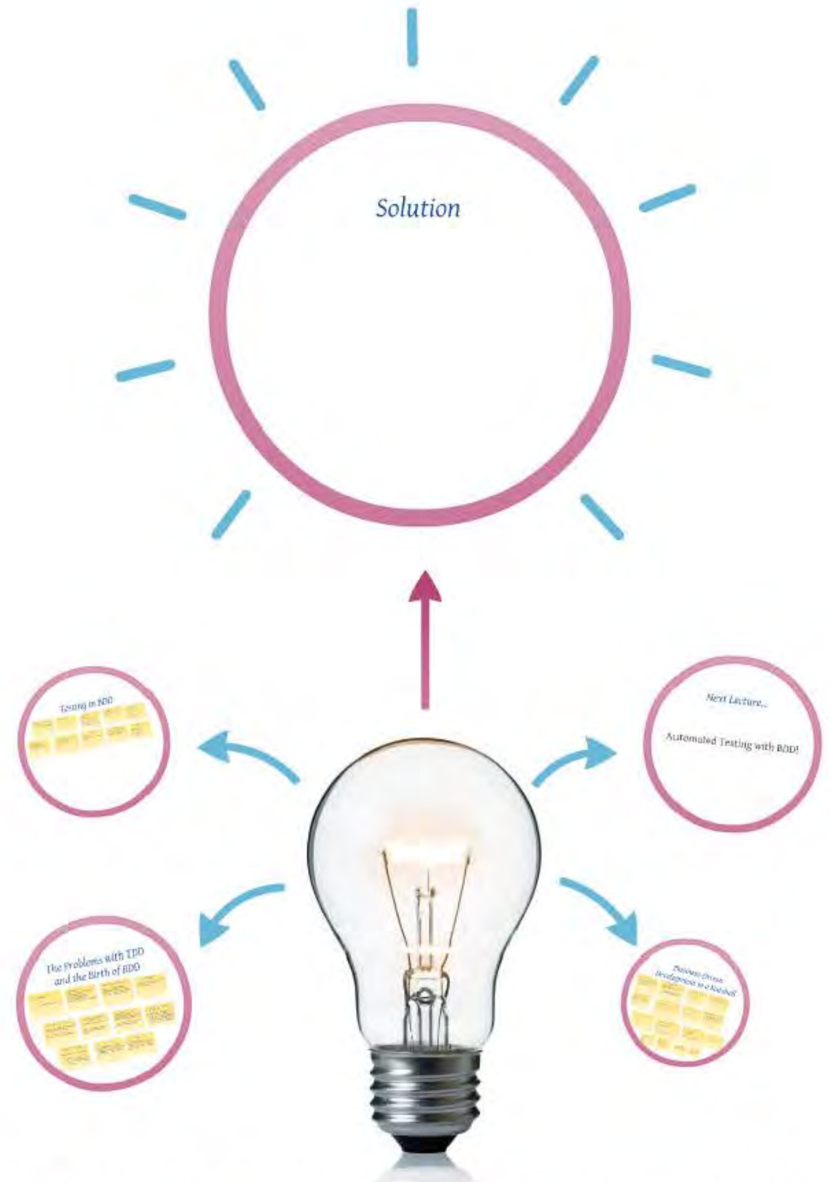
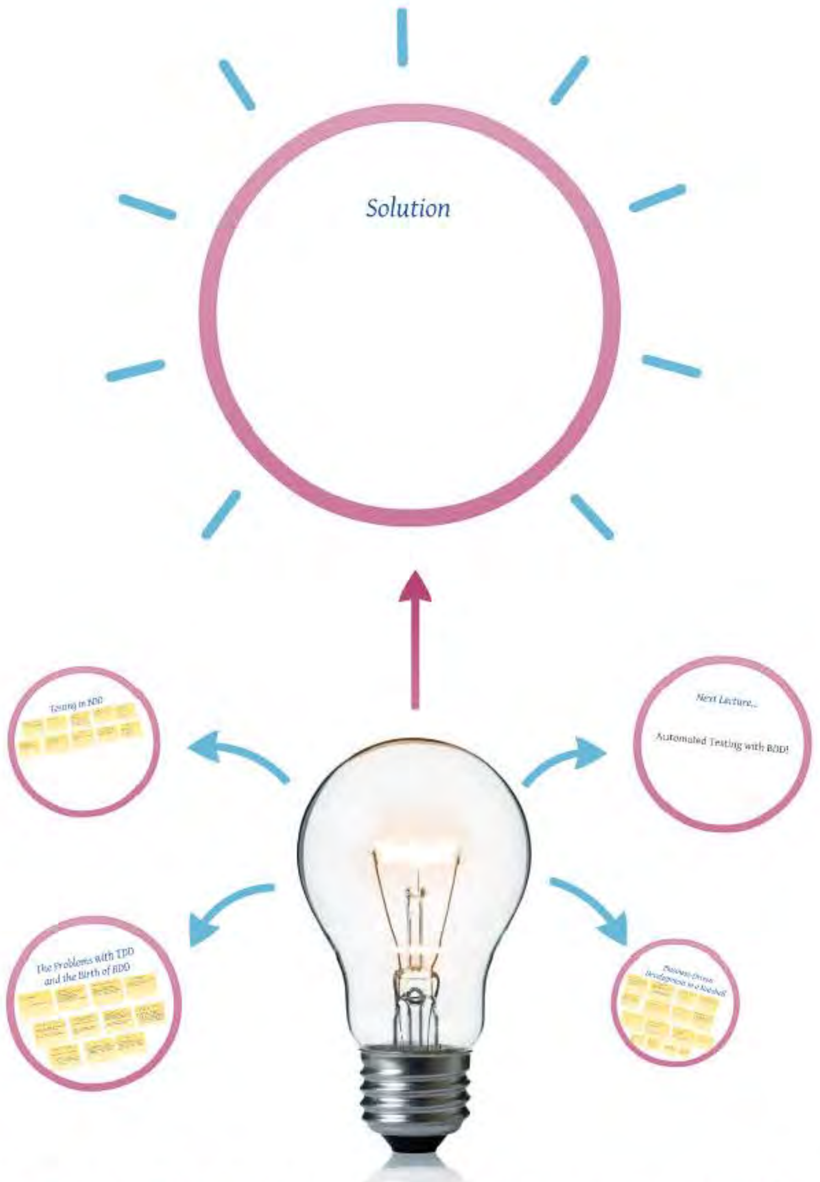

Solution
Testing in BDD
Next Lecture...
Automated Testing with BDD!
The Problems with TDD and the Birth of BDD
Business-Driven Development in a Nutshell

CS1699: Lecture 12 - Intro to Behavior-Driven Development


Solution
Testing in BDD
Next Lecture...
Automated Testing with BDD!
The Problems with TDD and the Birth of BDD
Business-Driven Development in a Nutshell

CS1699: Lecture 12 - Intro to Behavior-Driven Development

The Problems with TDD and the Birth of BDD

TDD is powerful....

... but it leaves many questions.

Where should I start?
What should I test... or avoid testing?
How big should my tests be?
What should I name my tests?
Why are my tests failing?
How do I figure out why a test is failing?

2006: Dan North shows a different way

In an article in Better Software magazine, Dan North explained an evolution of TDD that he called BDD, or "Behavior-Driven Development."

Evolution

Note that this an evolution of TDD, but it can also be used by itself, or with "traditional" TDD.

BDD is supposed to avoid the pitfalls of TDD.

There are some successes and failures at this, in my opinion. We'll go over them.

TDD is focused on building good *code*.
BDD is focused on building a good *product*.

In order to understand BDD, you need to understand ATDD.

Remember the red-green-refactor loop in TDD?

What if it applied to features of the software instead of methods and functions?

```
public void testCanAccessSite() { }
public void testCanLogin() { }
public void testCanPurchaseWidget() { }
public void testDisplayAmazonData() { }

// These look more like your manual
// tests in Deliverable 1, right?
```

Each Acceptance Test (AT) may have numerous unit tests...

```
public void testDisplayAmazonData() { }
-> public void testConnectAmazon() { }
-> public void testConnectBadPW() { }
-> public void testConnectionError() { }
-> public void testRetrieveJSON() { }
-> public void testRecoverBadJSON() { }
```

Eventually, the higher-level test passes.

I think of ATDD as "hierarchical TDD".

You're still doing TDD, just at two levels. Once you finish enough code at the lower level, you move up and see if it works.

It's easier to deal with understanding the level of "I need to display Amazon data" than remembering every single error condition to check for when connecting to Amazon's API.

However....

You're still looking at this like a programmer. BDD helps you see the problem as a user of the system, and how the user would like to use the system.

TDD is powerful....

... but it leaves many questions.

Where should I start?
What should I test... or avoid testing?
How big should my tests be?
What should I name my tests?
Why are my tests failing?
How do I figure out why a test is failing?

2006: Dan North shows a different way

In an article in Better Software magazine, Dan North explained an evolution of TDD that he called BDD, or "Behavior-Driven Development."

Evolution

Note that this an evolution of TDD, but it can also be used by itself, or with "traditional" TDD.

BDD is supposed to avoid the pitfalls of TDD.

There are some successes and failures at this, in my opinion. We'll go over them.

TDD is focused on building good *code*.
BDD is focused on building a good *product*.

In order to understand BDD, you need to understand ATDD.

Remember the red-green-refactor loop in TDD?

What if it applied to features of the software instead of methods and functions?

```
public void testCanAccessSite() { }
public void testCanLogIn() { }
public void testCanPurchaseWidget() { }
public void testDisplayAmazonData() { }

// These look more like your manual
// tests in Deliverable 1, right?
```

Each Acceptance Test (AT) may have numerous unit tests...

```
public void testDisplayAmazonData() { }
-> public void testConnectAmazon() { }
-> public void testConnectBadPW() { }
-> public void testConnectionError() { }
-> public void testRetrieveJSON() { }
-> public void testRecoverBadJSON() { }
```

Eventually, the higher-level test passes.

I think of ATDD as "hierarchical TDD".

You're still doing TDD, just at two levels. Once you finish enough code at the lower level, you move up and see if it works.

It's easier to deal with understanding the level of "I need to display Amazon data" than remembering every single error condition to check for when connecting to Amazon's API.

However....

You're still looking at this like a programmer. BDD helps you see the problem as a user of the system, and how the user would like to use the system.

Business-Driven Development in a Nutshell

ATDD is useful, but really it's just a variant on TDD.

Think of TDD as UTDD (Unit-Test Driven Development). ATDD is just the same thing with more far-reaching tests.

It's not really something you could easily talk about with a non-technical person.

This Limits Us

If you and your users/customers/project managers are all speaking different languages, communication becomes more difficult.

Try explaining P vs NP to a non-CS or non-Math major sometime. Think about the language we take for granted: polynomial, big-O, algorithm, etc.

BDD depends on an ubiquitous language

Think of it as Esperanto!

a "Common Language"

In order to do so, let's take a step back...

... and re-consider the concept of "requirements".

Is there a better way to describe what the user wants?

"The system shall enable the LOWTEMP warning light when two out of three internal thermometers agree that the ambient temperature is below -10 degrees Celsius (14 degrees F, 263 degrees Kelvin), as indicated by the INTHERM1, INTHERM2, and INTHERM3 registers."

Does this describe what the user wants, or what the engineer wants to see?

It's a well laid-out list of things to do, which is great to for the engineer.

But will the user understand it?

Probably not.

What if... we could describe this in a language that both sides could understand?

We could easily go back to customers and see that we're on the right track, previous to development.

This helps us be "Agile".

The Connextra Template/Format

As a <role>
I want <function/functionality>
So that <reason/benefit>

also called a "As A.. I Want.. So that" template/format

or a "Role/Function/Reason" template/format

Example

As a systems administrator
I want to create users
So that users in my domain can log in

As a user
I want to see my bank account balance
So that I know how much money I have

We Take Advantage of Our Common Humanity

Roles

[illegible]

Example

[illegible]

Role-Playing

[illegible]

[illegible]

ATDD is useful, but really it's just a variant on TDD.

Think of TDD as UTDD (Unit-Test Driven Development). ATDD is just the same thing with more far-reaching tests.

It's not really something you could easily talk about with a non-technical person.

This Limits Us

If you and your users/customers/project managers are all speaking different languages, communication becomes more difficult.

Try explaining P vs NP to a non-CS or non-Math major sometime. Think about the language we take for granted: polynomial, big-O, algorithm, etc.

BDD depends on an ubiquitous language

Think of it as Esperanto!

a "Common Language"

In order to do so, let's take a step back...

... and re-consider the concept of "requirements".

Is there a better way to describe what the user wants?

"The system shall enable the LOWTEMP warning light when two out of three internal thermometers agree that the ambient temperature is below -10 degrees Celsius (14 degrees F, 263 degrees Kelvin), as indicated by the INTHERM1, INTHERM2, and INTHERM3 registers."

Does this describe what the user wants, or what the engineer wants to see?

It's a well laid-out list of things to do, which is great to for the engineer.

But will the user understand it?

Probably not.

What if... we could describe this in a language that both sides could understand?

We could easily go back to customers and see that we're on the right track, previous to development.

This helps us be "Agile".

The Connextra Template/Format

As a <role>
I want <function/functionality>
So that <reason/benefit>

also called a "As A.. I Want.. So that" template/format

or a "Role/Function/Reason" template/format

Example

As a systems administrator
I want to create users
So that users in my domain can log in

As a user
I want to see my bank account balance
So that I know how much money I have

We Take Advantage of Our Common Humanity

Roles

1. User
2. Administrator
3. Domain-Specific
   a) Salesperson
   b) Stock trader
   c) Banker
   d) Trainer
   e) Twitter user
   f) Student
   g) Teacher
   h) Soldier

Example

As a soldier
I want to see a map of my current location
So that I can accurately navigate

As a soldier
I want to see where the enemy is
So that I can attack the enemy or defend against him/her

As a commanding officer
I want to see where my unit is located
So that I can accurately plan my attacks

Role-Playing

CatShare++ has succeeded beyond our wildest dreams, as it was certain to do. Our company is now expanding into direct cat rental, with its new subsidiary....

Rent-A-Cat

Role-Playing

1. Users who want to rent cats for companionship
2. Users who want to rent cats for mousing
3. Users who want to rent cats for homework help
4. Cat trainers
5. Administrators
6. Marketing personnel
7. Rent-A-Cat social media personnel

Testing in BDD

We now have "requirements", but testing these are slightly different.

We can use another template to put them into words the users, engineers, and testers can understand.

These are called "scenarios".

Example

As a user
I want to log in
So that I can access my banking account

Scenario 1: I log in with correct username and password
Scenario 2: I log in with correct username, but incorrect password
Scenario 3: I log in with incorrect username

They're basically "subsets" of functionality that arise from the user story.

Each of these can be individually tested using yet another template.

Given (preconditions / input values)

When (execution steps)

Then (postconditions / output values)

(you can also add "And" to make more than one of a given type of value)

Given / When / Then template

Example

Given a correct username
And an incorrect password
When I try to log in with those credentials
Then I should receive an error page with "incorrect password entered" on it

Example

Given one cat with name "Fluffernins"
When a user is logged in
And searches for "Fluffernins"
The cat named "Fluffernins" should appear on the search results

Role-Playing

You are now testers.

Develop three scenarios for one of the user stories you developed earlier.

Specifications

These are the unit test equivalents of BDD.

They specify very specific, very technical behavior, usually at the method level.

Example

Specification: Linked List

When a Linked List is created
Then it will have no nodes
And have a length of 0

Given an empty Linked List
When a node is added
Then it will have one node
And a have a length of 1

We now have "requirements", but testing these are slightly different.

We can use another template to put them into words the users, engineers, and testers can understand.

These are called "scenarios".

Example

As a user
I want to log in
So that I can access my banking account

Scenario 1: I log in with correct username and password
Scenario 2: I log in with correct username, but incorrect password
Scenario 3: I log in with incorrect username

They're basically "subsets" of functionality that arise from the user story.

Each of these can be individually tested using yet another template.

Given (preconditions / input values)

When (execution steps)

Then (postconditions / output values)

(you can also add "And" to make more than one of a given type of value)

Given / When / Then template

Example

Given a correct username
And an incorrect password
When I try to log in with those credentials
Then I should receive an error page with
"incorrect password entered" on it

Example

Given one cat with name "Fluffernins"
When a user is logged in
And searches for "Fluffernins"
The cat named "Fluffernins" should appear on the search results

Role-Playing

You are now testers.

Develop three scenarios for one of the user stories you developed earlier.

Specifications

These are the unit test equivalents of BDD.

They specify very specific, very technical behavior, usually at the method level.

Example

Specification: Linked List

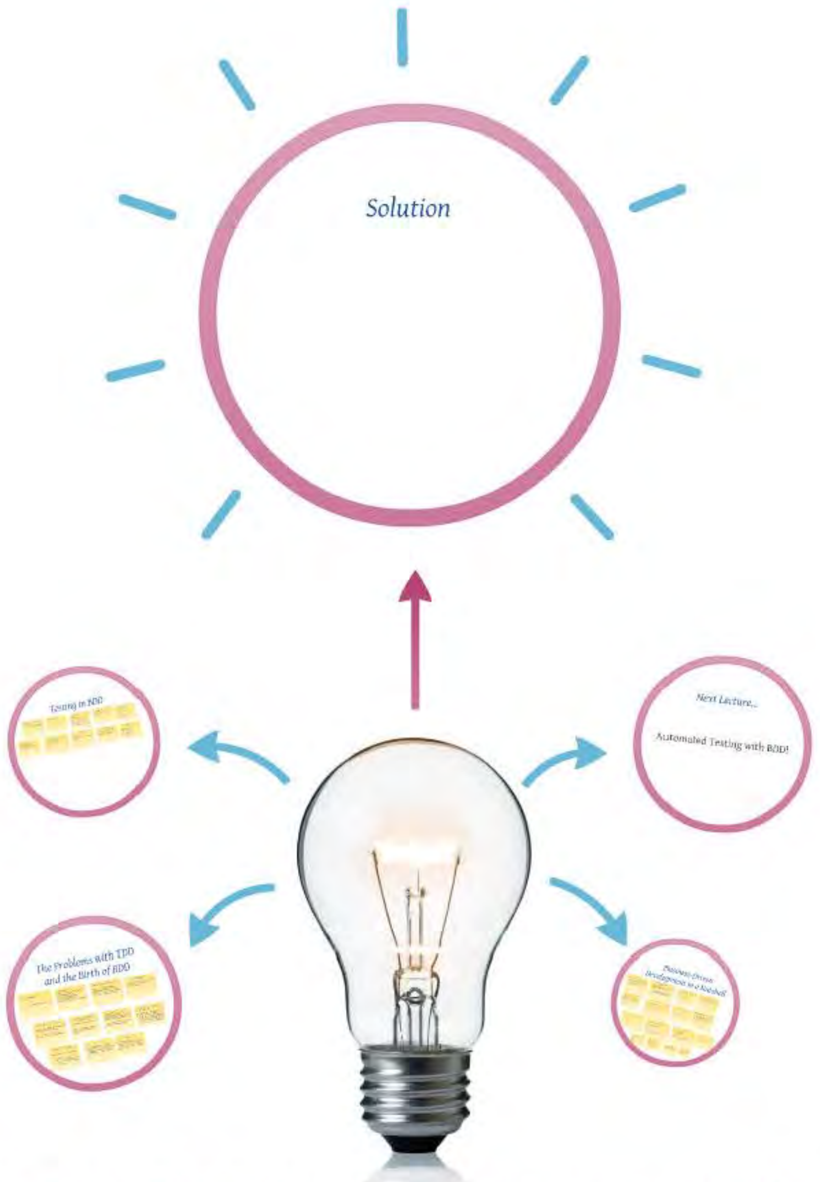
When a Linked List is created
Then it will have no nodes
And have a length of 0

Given an empty Linked List
When a node is added
Then it will have one node
And a have a length of 1

Next Lecture...

Automated Testing with BDD!

PREZI


Solution
Testing in BDD
Next Lecture...
Automated Testing with BDD!
The Problems with TDD and the Birth of BDD
Behavior-Driven Development in a Nutshell

CS1699: Lecture 12 - Intro to Behavior-Driven Development

Solution