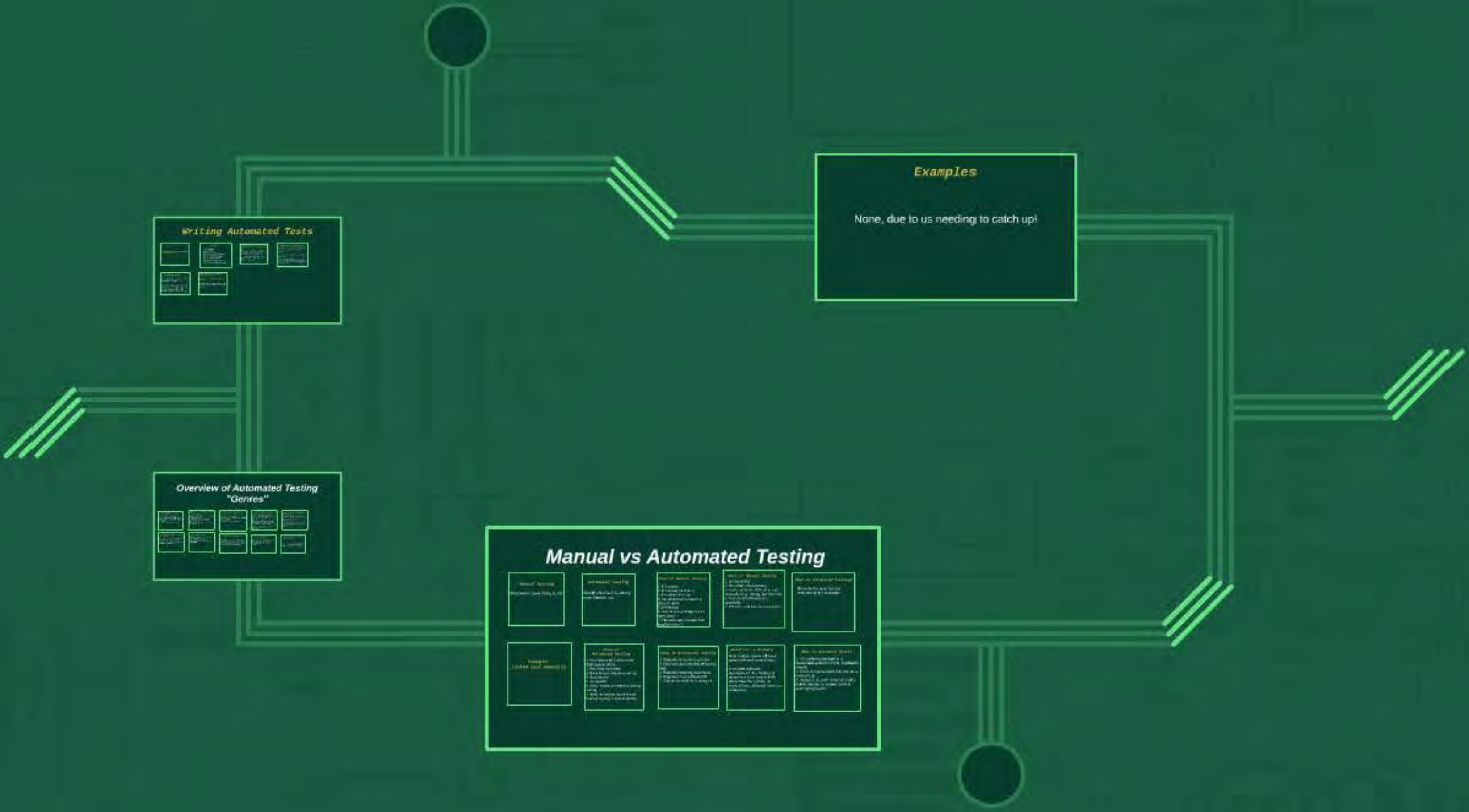
CS1699 Lecture 6: Automated Testing



Writing Automated Tests

Examples

None, due to us needing to catch up!

Overview of Automated Testing "Genres"

Manual vs Automated Testing

CS1699 Lecture 6: Automated Testing

Writing Automated Tests

Examples

None, due to us needing to catch up!

Overview of Automated Testing "Genres"

Manual vs Automated Testing

Manual vs Automated Testing

Manual Testing

What we've been doing so far.

Automated Testing

Mostly what we'll be doing from here on out.

Pros of Manual Testing

1. It's simple
2. It's cheap (at first...)
3. It's easy to set up
4. No additional software to learn or write
5. It's flexible
6. Can focus on things users care about
7. Humans catch issues that programs don't

Cons of Manual Testing

1. It is BORING
2. It is often unrepeatable
3. Some tasks are difficult to test manually (e.g., timing, low-level ifc)
4. Human error becomes a possibility
5. It's time- and resource-intensive

What is Automated Testing?

You write the tests that are executed by the computer.

Example: Linked List Equality

Pros of Automated Testing

1. No chance for human error (during execution)
2. Fast test execution
3. Easy to execute once set up
4. Repeatable
5. Analyzable
6. Less resource-intensive during testing
7. Ideal for testing some things manual testing is bad at testing

Cons of Automated Testing

1. Requires extra time up-front
2. May not catch some user-facing bugs
3. Requires learning more tools
4. Requires more skilled staff
5. Only tests what it's looking for

Solution: A Mixture

Most product teams will have automated and manual tests.

In modern software development, the number of automated tests tend to FAR outnumber the number of manual tests, although there are exceptions.

When To Automate Tests?

1. If it can be automated in a reasonable amount of time, it probably should.
2. If not, it may be worth it to just do a manual test.
3. It pays to do some manual testing before release, to ensure minimal user-facing issues.

Manual Testing

What we've been doing so far.

Automated Testing

Mostly what we'll be doing from here on out.

Pros of Manual Testing

1. It's simple
2. It's cheap (at first...)
3. It's easy to set up
4. No additional software to learn or write
5. It's flexible
6. Can focus on things users care about
7. Humans catch issues that programs don't

Cons of Manual Testing

1. It is BORING
2. It is often unrepeatable
3. Some tasks are difficult to test manually (e.g., timing, low-level ifc)
4. Human error becomes a possibility
5. It's time- and resource-intensive

What is Automated Testing?

You write the tests that are executed by the computer.

Example:
Linked List Equality

Pros of Automated Testing

1. No chance for human error (during execution)
2. Fast test execution
3. Easy to execute once set up
4. Repeatable
5. Analyzable
6. Less resource-intensive during testing
7. Ideal for testing some things manual testing is bad at testing

Cons of Automated Testing

1. Requires extra time up-front
2. May not catch some user-facing bugs
3. Requires learning more tools
4. Requires more skilled staff
5. Only tests what it's looking for

M
au

In
de
au
ou
m
ex

ing

cing

s

or

Solution: A Mixture

Most product teams will have automated and manual tests.

In modern software development, the number of automated tests tend to FAR outnumber the number of manual tests, although there are exceptions.

1.
rea
sho
2.
ma
3.
bef
use

When To Automate Tests?

1. If it can be automated in a reasonable amount of time, it probably should.
2. If not, it may be worth it to just do a manual test.
3. It pays to do some manual testing before release, to ensure minimal user-facing issues.

e

Overview of Automated Testing "Genres"

Unit Tests

Test low-level "units" (e.g. classes, methods, functions) of code
Written by developers or white-box testers (require intimate knowledge of the codebase)
The earliest tests that code faces

Integration / Acceptance Tests

Test functionality, not code. Access the system through a test harness or directly through an interface.
No knowledge of code necessary.
Black-box testing.
Should be written before code is, or at least independently.

Performance Testing Software

Tests the speed or resource usage of a function, system, or process.
Can be a part of integration testing; rarely part of unit tests.
Can also be run independently.

Property-Based Testing Tools

Test determine which properties output values should have given certain input value properties, then passes in many of those input values (usually randomly generated).

For example, a .sort function should always return values where the first value is smaller than the second, second is smaller than third, etc. with the same number of elements as those passed in.

This is usually at the unit test level, and is also much more commonly seen in functional programming.

Behavior-Driven Development (BDD) Tools

Tests (often written in English or other natural language) that are converted into tests of functionality.
Example:
Scenario 1: Items added are displayed in the cart
Given a customer is logged in
And the customer currently has three widgets in his/her cart
When the customer adds a widget
Then the customer should see four widgets in his/her cart

Used often in Agile development, usually at integration (not unit) level.

Continuous Integration (CI) Test Tools

Ensure all tests pass before integrating code to baseline. Used often for Frequent Integration (FI) and Continous Integration (CI) setups.

GUI Testing Tools

Allow tests to interact with a GUI (web page, application screen, etc.)
Often used at integration level, not unit testing.

Model-Based Testing Tools

Generate a model of the system (e.g., a finite state machine) and ensure that the system you designed follows the model.
Often used for embedded systems.

Fuzz Testing Tools

Ensure a system handles random data, random downtime of subsystems, etc.
Popularized by Netflix's "Chaos Monkey."

Lots of tools, but it all boils down to...

What did I *expect* to happen?
What *actually* happened?

Unit Tests

Test low-level "units" (e.g. classes, methods, functions) of code

Written by developers or white-box testers (require intimate knowledge of the codebase)

The earliest tests that code faces

Integration / Acceptance Tests

Test functionality, not code. Access the system through a test harness or directly through an interface.

No knowledge of code necessary.

Black-box testing.

Should be written before code is, or at least independently.

Performance Testing Software

Tests the speed or resource usage of a function, system, or process.
Can be a part of integration testing; rarely part of unit tests.
Can also be run independently.

Property-Based Testing Tools

Test determine which properties output values should have given certain input value properties, then passes in many of those input values (usually randomly generated)

For example, a .sort function should always return values where the first value is smaller than the second, second is smaller than third, etc. with the same number of elements as those passed in.

This is usually at the unit test level, and is also much more commonly seen in functional programming.

Behavior-Driven Development (BDD) Tools

Tests (often written in English or other natural language) that are converted into tests of functionality.
Example:
Scenario 1: Items added are displayed in the cart
Given a customer is logged in
And the customer currently has three widgets in his/her cart
When the customer adds a widget
Then the customer should see four widgets in his/her cart

Used often in Agile development, usually at Integration (not unit) level.

Continuous Integration (CI) Test Tools

Ensure all tests pass before integrating code to baseline. Used often for Frequent Integration (FI) and Continous Integration (CI) setups.

GUI Testing Tools

Allow tests to interact with a GUI (web page, application screen, etc.)
Often used at integration level, not unit testing.

Model-Based Testing Tools

Generate a model of the system (e.g., a finite state machine) and ensure that the system you designed follows the model. Often used for embedded systems.

Fuzz Testing Tools

Ensure a system handles random data, random downtime of subsystems, etc. Popularized by Netflix's "Chaos Monkey."

Lots of tools, but it all boils down to...

What did I *expect* to happen?
What *actually* happened?

Writing Automated Tests

Remember the Rules of Writing Manual Tests...

You need:

IDENTIFIER:
TEST CASE:
PRECONDITIONS (if any):
INPUT VALUES (if any):
EXECUTION STEPS:
OUTPUT VALUES (if any):
POSTCONDITIONS (if any):

Minimize External Dependencies

You should test your code, not Amazon's web service, or your network connection, or another service you depend on.

Write tests that can avoid these. We'll talk about exactly how in the next few lectures. For now, fake them.

Minimize External Dependencies

For example, run the server and client on the same machine and have the IP address be 127.0.0.1.

Create a fake service which responds 200 OK to everything.

NB You have now generated more areas for bad code to appear (in your testing harnesses). Such is life.

Minimize Randomness

These tests should be repeatable, in general. Use specific values instead of a random one. Be as specific as possible.

NB Fuzz testing and property-based testing are obvious exceptions. However, in these cases the failed input/output values should be logged (e.g. "Sort monotonically increasing - Falsifiable with [0,3,2] -> [3,2,0]")

Very well-defined expected vs. observed behaviors

Remember the peanut butter and jelly example.

You need to be very precise - more precise than in manual tests - about what you expect to happen.

Prezi

Remember the Rules of Writing Manual Tests...

You need:

IDENTIFIER:
TEST CASE:
PRECONDITIONS (if any):
INPUT VALUES (if any):
EXECUTION STEPS:
OUTPUT VALUES (if any):
POSTCONDITIONS (if any):

Minimize External Dependencies

You should test your code, not Amazon's web service, or your network connection, or another service you depend on.

Write tests that can avoid these. We'll talk about exactly how in the next few lectures. For now, fake them.

Minimize External Dependencies

For example, run the server and client on the same machine and have the IP address be 127.0.0.1.

Create a fake service which responds 200 OK to everything.

NB You have now generated more areas for bad code to appear (in your testing harnesses). Such is life.

Minimize Randomness

These tests should be repeatable, in general. Use specific values instead of a random one. Be as specific as possible.

NB Fuzz testing and property-based testing are obvious exceptions. However, in these cases the failed input/output values should be logged (e.g. "Sort monotonically increasing - Falsifiable with [0,3,2] -> [3,2,0]")

Very well-defined expected vs. observed behaviors

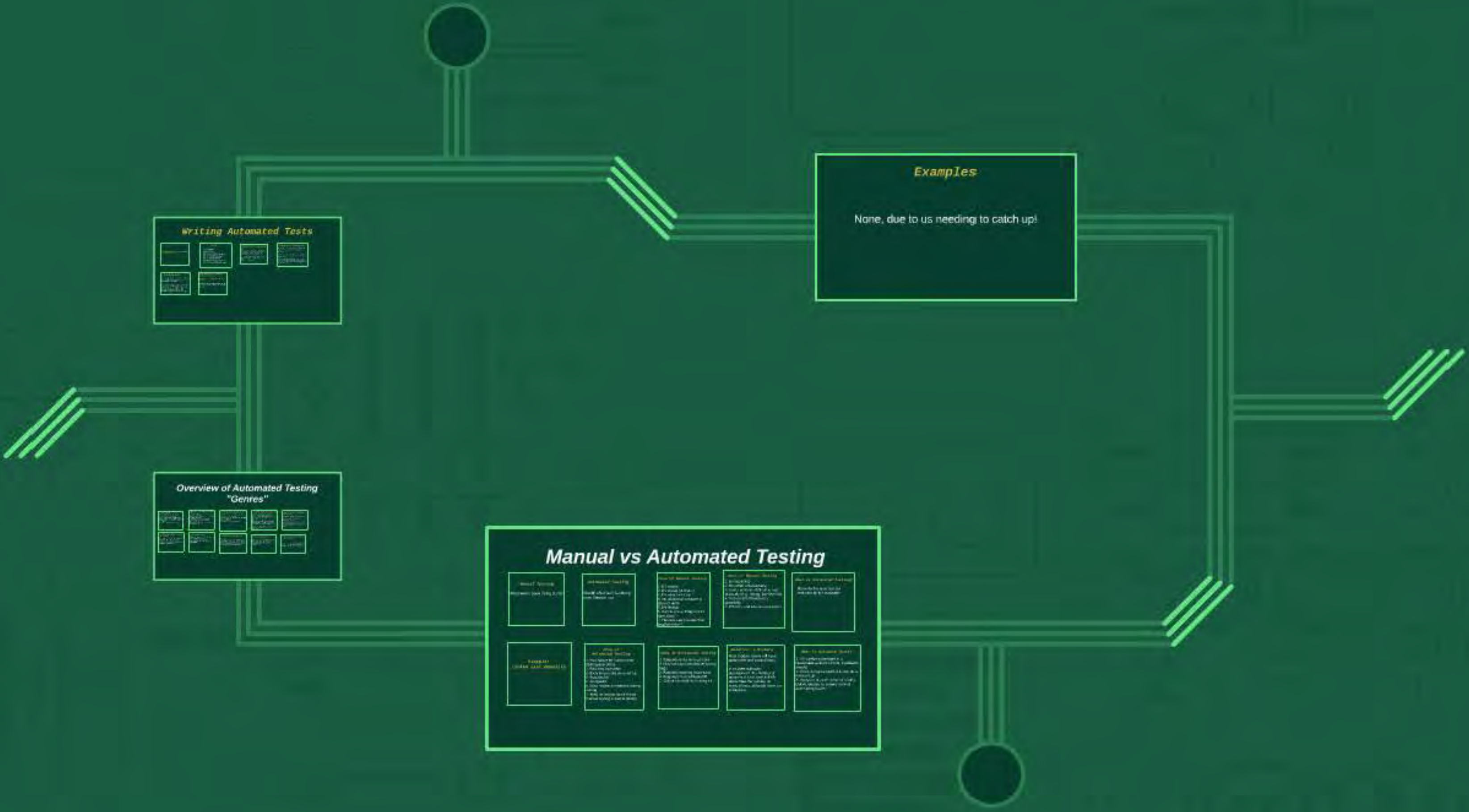
Remember the peanut butter and jelly example.

You need to be very precise - more precise than in manual tests - about what you expect to happen.

Examples

None, due to us needing to catch up!

CS1699 Lecture 6: Automated Testing



Writing Automated Tests

Examples

None, due to us needing to catch up!

Overview of Automated Testing "Genres"

Manual vs Automated Testing