COMP110: Principles of Computing

Software Quality

Today's lecture

Today's lecture has **three parts**

- Software quality and quality assurance
- Pathfinding and the A* algorithm
    - Introducing the next worksheet
- Live coding: applications of OOP techniques

Software testing

FALMOUTH UNIVERSITY

In this section

In this section you will learn how to:

- **Discuss** the importance of software testing in game development
- **Identify** the different types and levels of testing
- **Apply** test-driven development practices to your own programming projects

Further reading

- Pressman, R.S. (2009) Software Engineering: A Practitioner's Approach. 7th Edition. McGraw-Hill.

Quality

Last time:

Quality

Last time:

- There are many ways of measuring the **quality** of a game or piece of software

Quality

Last time:

- There are many ways of measuring the **quality** of a game or piece of software
- **Quality assurance** is important to ensure that the software is of sufficiently high quality to provide benefit to developers and end users

Testing

- Finding **inadvertent errors** in the design and implementation of software

Testing

- Finding **inadvertent errors** in the design and implementation of software
- Often takes more time and effort than any other part of development

Testing

- Finding **inadvertent errors** in the design and implementation of software
- Often takes more time and effort than any other part of development
- ... but letting errors slip into the final product can be even more costly

Testing

- Finding **inadvertent errors** in the design and implementation of software
- Often takes more time and effort than any other part of development
- ... but letting errors slip into the final product can be even more costly
- Testing $\neq$ quality assurance

Testing

- Finding **inadvertent errors** in the design and implementation of software
- Often takes more time and effort than any other part of development
- ... but letting errors slip into the final product can be even more costly
- Testing $\neq$ quality assurance
  - Testing is an important part of QA, but **not the only part**

Who is responsible?

- Last time, we discussed that **designers**, **developers**, **publishers**, and maybe even **players** share the responsibility for software quality in games

FALMOUTH UNIVERSITY

Who is responsible?

- Last time, we discussed that **designers**, **developers**, **publishers**, and maybe even **players** share the responsibility for software quality in games
- Who should take responsibility for **testing**?

Who is responsible?

- Last time, we discussed that **designers**, **developers**, **publishers**, and maybe even **players** share the responsibility for software quality in games
- Who should take responsibility for **testing**?
  - "**Developers** write the code, so they should make sure it works"?

Who is responsible?

- Last time, we discussed that **designers**, **developers**, **publishers**, and maybe even **players** share the responsibility for software quality in games
- Who should take responsibility for **testing**?
  - "**Developers** write the code, so they should make sure it works"?
  - "**Everyone** is responsible for quality, so everyone should pitch in"?

Who is responsible?

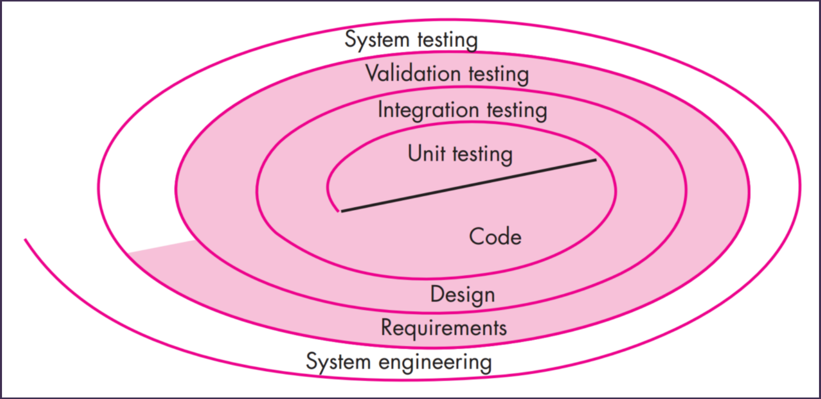
- Last time, we discussed that **designers**, **developers**, **publishers**, and maybe even **players** share the responsibility for software quality in games
- Who should take responsibility for **testing**?
    - "**Developers** write the code, so they should make sure it works"?
    - "**Everyone** is responsible for quality, so everyone should pitch in"?
    - "Code should be tested by **someone other** than the developer who wrote it"?

Socrative `6E8NSW3IN`

So who should test game software?

- In pairs.
- Discuss for 2-minutes.
- **Suggest** which parties should take responsibility for testing **and justify** your answer.

Testing strategy



(Pressman, 2009) Figure 17.1

Testing strategy

- Development starts with system engineering and works **inwards**

Testing strategy

- Development starts with system engineering and works **inwards**
  - The **waterfall model**

Testing strategy

- Development starts with system engineering and works **inwards**
  - The **waterfall model**
  - Agile doesn't quite work like this

Testing strategy

- Development starts with system engineering and works **inwards**
  - The **waterfall model**
  - Agile doesn't quite work like this
- Testing starts with unit testing and works **outwards**

Testing strategy

- Development starts with system engineering and works **inwards**
  - The **waterfall model**
  - Agile doesn't quite work like this
- Testing starts with unit testing and works **outwards**
- **White box testing**: testing the software **with** knowledge of its internal workings

FALMOUTH UNIVERSITY

Testing strategy

- Development starts with system engineering and works **inwards**
  - The **waterfall model**
  - Agile doesn't quite work like this
- Testing starts with unit testing and works **outwards**
- **White box testing**: testing the software **with** knowledge of its internal workings
- **Black box testing**: testing the software **without** knowledge of its internal workings

Unit testing

- A **unit test** is a piece of code that verifies a **unit** (e.g. a function or class) of a program

Unit testing

- A **unit test** is a piece of code that verifies a **unit** (e.g. a function or class) of a program
- E.g. verifies that a function called with a particular set of parameters returns the expected result

Unit testing

- A **unit test** is a piece of code that verifies a **unit** (e.g. a function or class) of a program
- E.g. verifies that a function called with a particular set of parameters returns the expected result
- E.g. verifies that a function called with invalid parameters throws the expected error

Designing user tests

- Test the **edge cases**

Designing user tests

- Test the **edge cases**
  - Programming errors often occur at the **boundary** between valid and invalid input, or the boundary between one case and another

Designing user tests

- Test the **edge cases**
  - Programming errors often occur at the **boundary** between valid and invalid input, or the boundary between one case and another
  - E.g. for an *n*-element data structure, test accessing elements $n - 1$, n, $n + 1$

Designing user tests

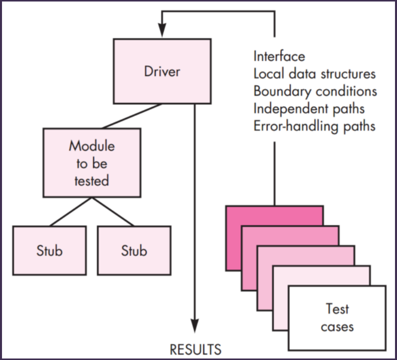
- Test the **edge cases**
  - Programming errors often occur at the **boundary** between valid and invalid input, or the boundary between one case and another
  - E.g. for an n-element data structure, test accessing elements $n - 1$, n, $n + 1$
- Aim for high **coverage**

Designing user tests

- Test the **edge cases**
  - Programming errors often occur at the **boundary** between valid and invalid input, or the boundary between one case and another
  - E.g. for an *n*-element data structure, test accessing elements $n - 1$, n, $n + 1$
- Aim for high **coverage**
  - Ideally, **every line of code** should be executed in **at least one** unit test

FALMOUTH UNIVERSITY

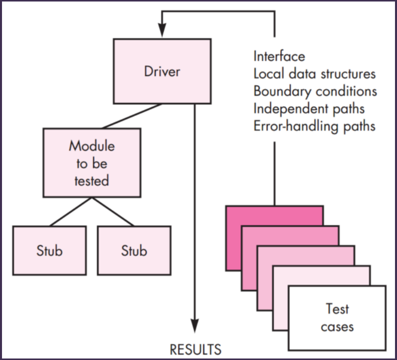
Drivers and stubs



(Pressman, 2009) Figure 17.4

- Unit testing generally requires extra code to be written

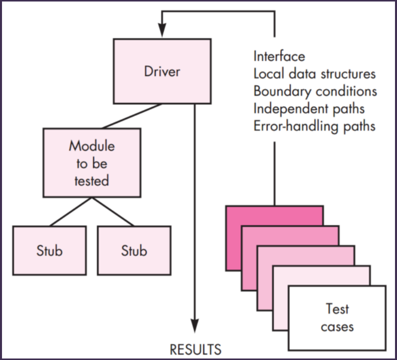
Drivers and stubs



(Pressman, 2009) Figure 17.4

- Unit testing generally requires extra code to be written
- **Driver** — to set up any required state and run the test

Drivers and stubs



(Pressman, 2009) Figure 17.4

- Unit testing generally requires extra code to be written
- **Driver** — to set up any required state and run the test
- **Stubs** — to replace any modules upon which the module under test depends

Integration testing

- Verify that the individual units work **together**

Integration testing

- Verify that the individual units work **together**
- Can be done **top-down** or **bottom-up**

Integration testing

- Verify that the individual units work **together**
- Can be done **top-down** or **bottom-up**
- Either way, the idea is to gradually replace stubs and drivers with actual units, testing as you go

Integration testing

- Verify that the individual units work **together**
- Can be done **top-down** or **bottom-up**
- Either way, the idea is to gradually replace stubs and drivers with actual units, testing as you go
- **Regression testing** is important — re-running tests to ensure that recent additions have not broken anything

Socrative 6E8NSW3IN

If the units have been thoroughly tested individually, why is integration testing needed?

- In pairs.
- Discuss for 2-minutes.
- Give an **example** of a problem that integration testing might uncover, but that unit testing might miss.

Validation testing

- Testing the complete software system from the user's point of view

Validation testing

- Testing the complete software system from the user's point of view
- E.g. playtesting

Socrative `6E8NSW3IN`

If unit testing and integration testing have been done correctly, why is validation testing needed?

- In pairs.
- Discuss for 2-minutes.
- Give an **example** of a problem that validation testing might uncover, but that unit and integration testing might miss.

When is testing “done”?

- The aim of testing is to find bugs, so it’s done when there are no bugs left to be found! ☺

When is testing “done”?

- The aim of testing is to find bugs, so it’s done when there are no bugs left to be found! ☺
- When the software is (quantitatively or qualitatively) “good enough”

When is testing “done”?

- The aim of testing is to find bugs, so it’s done when there are no bugs left to be found! ☺
- When the software is (quantitatively or qualitatively) “good enough”
- Testing is never “done” — the burden just shifts onto the users

Test driven development (TDD)

- A development process that advocates writing the unit tests **first**

Test driven development (TDD)

- A development process that advocates writing the unit tests **first**
- Repeat the following three steps:

Test driven development (TDD)

- A development process that advocates writing the unit tests **first**
- Repeat the following three steps:
    1. **Red**: create a new test case, which should initially **fail**

Test driven development (TDD)

- A development process that advocates writing the unit tests **first**
- Repeat the following three steps:
  1. **Red**: create a new test case, which should initially **fail**
  2. **Green**: write code to make the new test **succeed** (without causing the other test cases to fail)

Test driven development (TDD)

- A development process that advocates writing the unit tests **first**
- Repeat the following three steps:
  1. **Red**: create a new test case, which should initially **fail**
  2. **Green**: write code to make the new test **succeed** (without causing the other test cases to fail)
  3. **Refactor**: **improve** the code, ensuring that all tests still **succeed**

Why TDD?

- Often easier to convert a **user story** into test cases rather than directly into code

Why TDD?

- Often easier to convert a **user story** into test cases rather than directly into code
- Writing the bare minimum of code to make the test "green" lets you **focus on user stories**, not on **over-generalisation** or **non-essential functionality**

Why TDD?

- Often easier to convert a **user story** into test cases rather than directly into code
- Writing the bare minimum of code to make the test “green” lets you **focus on user stories**, not on **over-generalisation** or **non-essential functionality**
    - **KISS**: Keep It Simple, Stupid
    - **YAGNI**: You Aren’t Gonna Need It

Red

- Create a new test case, which should initially **fail**

Red

- Create a new test case, which should initially **fail**
- Write only enough code to allow the test case to compile and run, e.g. write a **stub** function

Red

- Create a new test case, which should initially **fail**
- Write only enough code to allow the test case to compile and run, e.g. write a **stub** function
- What if the test succeeds?

Red

- Create a new test case, which should initially **fail**
- Write only enough code to allow the test case to compile and run, e.g. write a **stub** function
- What if the test succeeds?
  - Maybe you already implemented that feature?

Red

- Create a new test case, which should initially **fail**
- Write only enough code to allow the test case to compile and run, e.g. write a **stub** function
- What if the test succeeds?
  - Maybe you already implemented that feature?
  - Maybe the test case is wrong?

Red

- Create a new test case, which should initially **fail**
- Write only enough code to allow the test case to compile and run, e.g. write a **stub** function
- What if the test succeeds?
  - Maybe you already implemented that feature?
  - Maybe the test case is wrong?
  - Maybe your unit testing code is broken?

FALMOUTH UNIVERSITY

Green

- Add the **bare minimum** of code to make the new test case succeed

Green

- Add the **bare minimum** of code to make the new test case succeed
  - **K**eep **I**t **S**imple, **S**tupid!

Green

- Add the **bare minimum** of code to make the new test case succeed
  - **K**eep **I**t **S**imple, **S**tupid!
- Verify that **all** unit tests now succeed

Green

- Add the **bare minimum** of code to make the new test case succeed
  - **K**eep **I**t **S**imple, **S**tupid!
- Verify that **all** unit tests now succeed
- What if old tests now fail?

Green

- Add the **bare minimum** of code to make the new test case succeed
  - **K**eep **I**t **S**imple, **S**tupid!
- Verify that **all** unit tests now succeed
- What if old tests now fail?
  - Fix it

Green

- Add the **bare minimum** of code to make the new test case succeed
  - **K**eep **I**t **S**imple, **S**tupid!
- Verify that **all** unit tests now succeed
- What if old tests now fail?
  - Fix it
  - **Or** revert and start again — can be faster than debugging

Green

- Add the **bare minimum** of code to make the new test case succeed
  - **K**eep **I**t **S**imple, **S**tupid!
- Verify that **all** unit tests now succeed
- What if old tests now fail?
  - Fix it
  - **Or** revert and start again — can be faster than debugging
  - (you **did** commit before you started, right?)

Refactor

- E.g. remove duplication, improve names, add documentation, apply design patterns, ...

Refactor

- E.g. remove duplication, improve names, add documentation, apply design patterns, ...
- To generalise or not to generalise?

Refactor

- E.g. remove duplication, improve names, add documentation, apply design patterns, ...
- To generalise or not to generalise?
- **Do** generalise if it makes the code **simpler**

Refactor

- E.g. remove duplication, improve names, add documentation, apply design patterns, ...
- To generalise or not to generalise?
- **Do** generalise if it makes the code **simpler**
- **Don't** generalise because you "might" need it later

Refactor

- E.g. remove duplication, improve names, add documentation, apply design patterns, ...
- To generalise or not to generalise?
- **Do** generalise if it makes the code **simpler**
- **Don't** generalise because you "might" need it later
  - **Y**ou **A**ren't **G**onna **N**eed **I**t!
  - Wait until it **is** needed in another cycle

Refactor

- E.g. remove duplication, improve names, add documentation, apply design patterns, ...
- To generalise or not to generalise?
- **Do** generalise if it makes the code **simpler**
- **Don't** generalise because you "might" need it later
  - **Y**ou **A**ren't **G**onna **N**eed **I**t!
  - Wait until it **is** needed in another cycle
- Verify that **all** unit tests still succeed

Socrative `6E8NSW3IN`

How suitable is the test driven approach for game development?

- In pairs.
- Discuss for 2-minutes.
- Suggest **one advantage and one disadvantage** of test driven development in the context of game development

COMP110 Coding Task 2

The assignment brief

LearningSpace: COMP110 assignment 4

The task

- Develop a **component**...

The task

- Develop a **component**...
  - **For example**, non-player character AI
  - **or** procedural content generator
  - **or** physics simulation
  - **or** combat mechanic
  - **or** ...

The task

- Develop a **component**...
  - **For example**, non-player character AI
  - **or** procedural content generator
  - **or** physics simulation
  - **or** combat mechanic
  - **or** ...
- ... for a **game**

The task

- Develop a **component**...
  - **For example**, non-player character AI
  - **or** procedural content generator
  - **or** physics simulation
  - **or** combat mechanic
  - **or** ...
- ... for a **game**
  - BA Digital Games project
  - **or** your COMP150 group project
  - **or** your COMP130 Kivy project

How does this fit with COMP150?

- You will take **ownership** of this component of the game

How does this fit with COMP150?

- You will take **ownership** of this component of the game
  - Essentially as a “consultant” to your own team

How does this fit with COMP150?

- You will take **ownership** of this component of the game
  - Essentially as a “consultant” to your own team
- Members of the same COMP150 team **must not** target the same component of their COMP150 game

Proposal

- For **next Wednesday's COMP110 lecture (9th March)**
- See assignment brief for details