COMP160: Software Engineering

1: Software Quality

FALMOUTH
UNIVERSITY

Software Quality and Quality Assurance

Learning Outcomes

In this section you will learn how to...

- **Explain** what 'quality' is
- **Explain** what 'quality assurance' is
- **Discuss** the role of quality assurance in software engineering

Further Reading

- Pressman, R.S. (2009) *Software Engineering: A Practitioner's Approach*. 7th Edition. McGraw-Hill.
- Kaner, C., Falk, J. And Nguyen, H.Q. (1999) *Testing Computer Software*. John Wiley and Sons.

Software Quality

"Bad software plagues nearly every organisation that uses computers, causing lost work hours during computer downtime lost or corrupted data, missed sales opportunities, high IT support, and maintenance costs, and low customer satisfaction" (ComputerWorld, 2005)

Software Quality

"The Sorry State of Software Quality—quality has gotten worse!" (InfoWorld, 2006)

Software Quality

So, what does quality look like in games?

Software Quality

So, what does quality look like in games?

- "A characteristic or attribute of something."

Software Quality

So, what does quality look like in games?

- "A characteristic or attribute of something."
- Quality may relate to several aspects of games:

Software Quality

So, what does quality look like in games?

- "A characteristic or attribute of something."
- Quality may relate to several aspects of games:
  - **the quality of the design**: the aesthetic is specified to accurately meet the desires and pleasures of the target audience; provides a distinctive experience; etc.

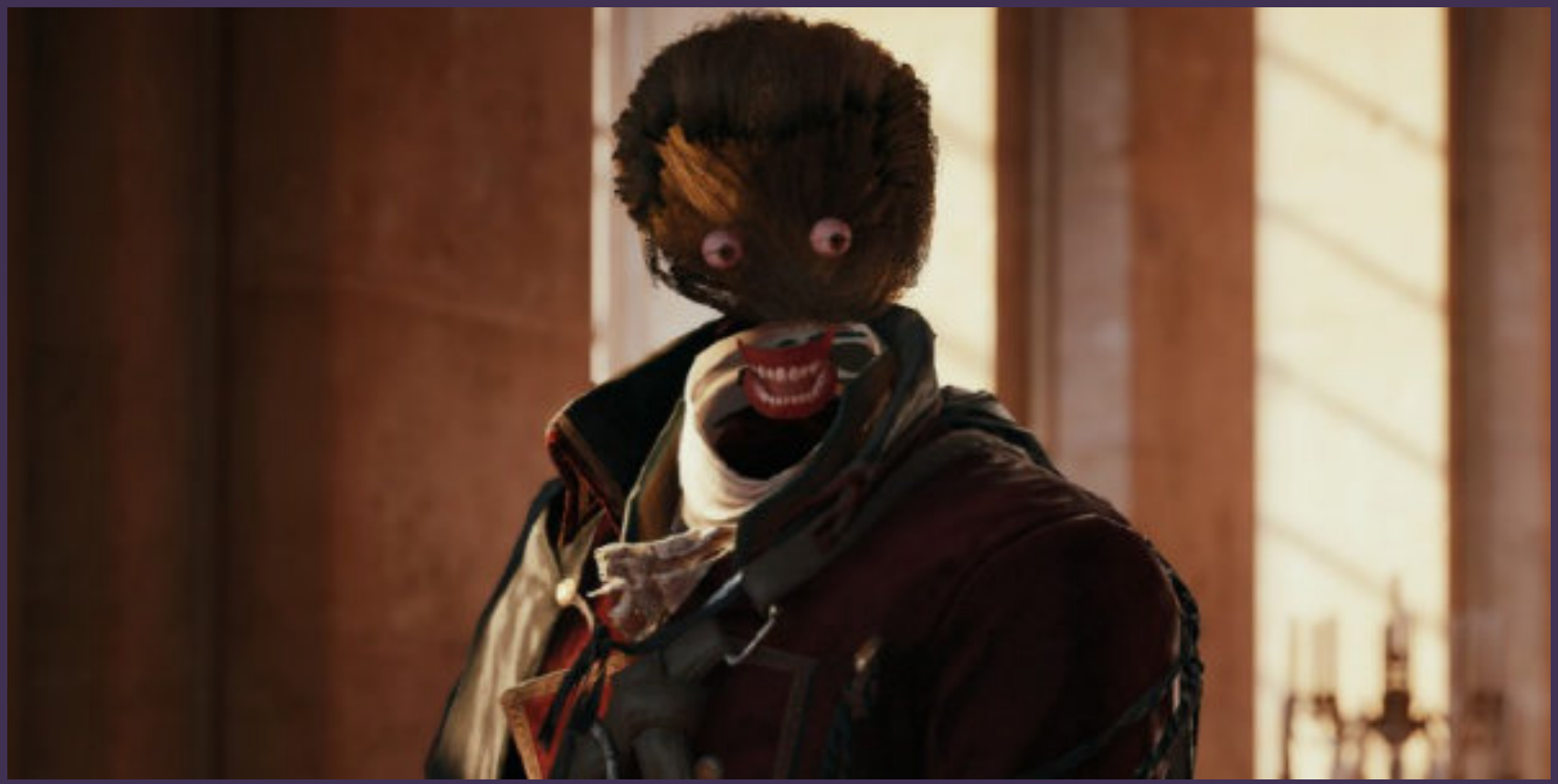
Software Quality

So, what does quality look like in games?

- "A characteristic or attribute of something."
- Quality may relate to several aspects of games:
  - **the quality of the design**: the aesthetic is specified to accurately meet the desires and pleasures of the target audience; provides a distinctive experience; etc.
  - **the quality of the implementation**: the game mechanics are able to achieve the intended aesthetic; the game is well-implemented...

Software Quality

Software Quality



Software Quality

https://www.youtube.com/watch?v=VhenntkfAgU

Software Quality

Today, software quality in games remains an issue, but who is to blame?

Software Quality

Today, software quality in games remains an issue, but who is to blame?

- Players blame developers, arguing that sloppy practices lead to low-quality software.

Software Quality

Today, software quality in games remains an issue, but who is to blame?

- Players blame developers, arguing that sloppy practices lead to low-quality software.
- Investors blame developers for not understanding what players want and what is `good enough' to maximise profit.

Software Quality

Today, software quality in games remains an issue, but who is to blame?

- Players blame developers, arguing that sloppy practices lead to low-quality software.
- Investors blame developers for not understanding what players want and what is `good enough' to maximise profit.
- Developers blame the design team and their publisher, arguing that irrational delivery dates and continuous change force them to deliver software before it can be adequately tested.

Socrative `6E8NSW3IN`

So, who is responsible?

- In pairs.
- Discuss for 2-minutes whether designers, developers, or publishers are responsible for software quality.
- **Suggest** which parties are responsible **and justify** your answer.

Socrative `6E8NSW3IN`

But wait...what exactly is quality?

- In pairs.
- Discuss for 2-minutes what `software quality' means in the context of game development.
- **Give** a definition for `game software quality'.

Software Quality

"Quality...you know what it is, yet you don't know what it is. But that's self-contradictory. But some things are better than others, that is, they have more quality. But when you try to say what the quality is, apart from the things that have it, it all goes poof! There's nothing to talk about. But if you can't say what Quality is, how do you know what it is, or how do you know that it even exists? If no one knows what it is, then for all practical purposes it doesn't exist at all. But for all practical purposes it really does exist. What else are the grades based on? Why else would people pay fortunes for some things and throw others in the trash pile? Obviously some things are better than others...but what's the betterness?...So round and round you go, spinning mental wheels and nowhere finding anyplace to get traction. What the hell is Quality? What is it?"

(Robert Persid, 1974)

Software Quality

- **transcendental view**: quality is something immediately recognisable, but cannot be explicitly defined.

Software Quality

- **transcendental view**: quality is something immediately recognisable, but cannot be explicitly defined.
- **pragmatic view**: relative to utility and specific goals. If something meets our goals, it exhibits quality.

Software Quality

- **transcendental view**: quality is something immediately recognisable, but cannot be explicitly defined.
- **pragmatic view**: relative to utility and specific goals. If something meets our goals, it exhibits quality.
- **commercial view**: the specification is key. If the specification is sound and the product conforms to the specification, it exhibits quality.

Software Quality

- **product view**: quality is tied to inherent characteristics (e.g. functions and features) of a product.

Software Quality

- **product view**: quality is tied to inherent characteristics (e.g. functions and features) of a product.
- **value-based view**: quality is based on how much a customer is willing to pay.

Software Quality

- **product view**: quality is tied to inherent characteristics (e.g. functions and features) of a product.
- **value-based view**: quality is based on how much a customer is willing to pay.
- In practice, perception of quality tends to combine these different views in subtle and nuanced ways.

Software Quality

"An effective software process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it." (Bes, 2004)

Socrative `6E8NSW3IN`

Can we now construct a better definition of software quality in games?

- In pairs.
- Discuss for 2-minutes what `software quality' means in the context of game development.
- **Give** a definition for `game software quality'.

Quality Assurance

- An **effective development process** establishes the infrastructure that supports any effort towards high quality.

Quality Assurance

- An **effective development process** establishes the infrastructure that supports any effort towards high quality.
- The management aspects create checks and balances to help avoid project chaos—a key contributor to poor quality.

Quality Assurance

- An **effective development process** establishes the infrastructure that supports any effort towards high quality.
- The management aspects create checks and balances to help avoid project chaos—a key contributor to poor quality.
- Software engineering practices empower developers to analyse and review their product.

Quality Assurance

- An **effective development process** establishes the infrastructure that supports any effort towards high quality.
- The management aspects create checks and balances to help avoid project chaos—a key contributor to poor quality.
- Software engineering practices empower developers to analyse and review their product.
- Umbrella activities, such as project management and code reviews, are key factors in determining quality and have just as an important role as any other specific source code quality assurance practice.

Quality Assurance

In order to assure quality in games, we need to know what to measure. David Garvin (1987) has some suggestions:

Quality Assurance

In order to assure quality in games, we need to know what to measure. David Garvin (1987) has some suggestions:

- **performance**: does the software deliver all content, functions, and features that are specified as part of the requirements model in a way that provides value to the end-user?

Quality Assurance

In order to assure quality in games, we need to know what to measure. David Garvin (1987) has some suggestions:

- **performance**: does the software deliver all content, functions, and features that are specified as part of the requirements model in a way that provides value to the end-user?
- **features**: does the software provide features that surprise and delight first-time end-users?

Quality Assurance

In order to assure quality in games, we need to know what to measure. David Garvin (1987) has some suggestions:

- **performance**: does the software deliver all content, functions, and features that are specified as part of the requirements model in a way that provides value to the end-user?
- **features**: does the software provide features that surprise and delight first-time end-users?
- **reliability**: does the software deliver all features and capability without failure? Is it available when it is needed? Does it deliver functionality that is error free?

Quality Assurance

- **conformance**: does the software conform to local and external software standards that are relevant to the application? Does it conform to de facto design and coding conventions? For example, does the user interface conform to accepted design rules for menu selection or data input?

Quality Assurance

- **conformance**: does the software conform to local and external software standards that are relevant to the application? Does it conform to de facto design and coding conventions? For example, does the user interface conform to accepted design rules for menu selection or data input?
- **durability**: Can the software be maintained (changed) or corrected (debugged) without the inadvertent generation of unintended side effects? Will changes cause the error rate or reliability to degrade with time?

Quality Assurance

- **serviceability**: Can the software be maintained (changed) or corrected (debugged) in an acceptably short time period. Can support staff acquire all information they need to make changes or correct defects?

Quality Assurance

- **serviceability**: Can the software be maintained (changed) or corrected (debugged) in an acceptably short time period. Can support staff acquire all information they need to make changes or correct defects?
- **look-and-feel**: Most of us would agree that an aesthetic entity has a certain elegance, a unique flow, and an obvious ‘presence’ that are hard to quantify but evident nonetheless.

Quality Assurance

- **serviceability**: Can the software be maintained (changed) or corrected (debugged) in an acceptably short time period. Can support staff acquire all information they need to make changes or correct defects?
- **look-and-feel**: Most of us would agree that an aesthetic entity has a certain elegance, a unique flow, and an obvious 'presence' that are hard to quantify but evident nonetheless.
- **socio-cultural context**: In some situations, you have a set of prejudices that will influence your perception of quality.

Quality Assurance

Other definitions to explore and factors to consider in your own time:

- McCall's Quality Factors
- ISO 9126 Quality Factors
- Targeted Factors
- IEEE 610.12 Software Assurance

Socrative `6E8NSW3IN`

Why are these factors important in games?

- In pairs.
- Discuss for 2-minutes why quality assurance is important to game development.
- **Illustrate TWO** reasons why quality assurance is important. Use examples to support your answer.

Quality Assurance

Why are these factors important?

Quality Assurance

Why are these factors important?

- If you produce a software system that has terrible quality, you lose because no one will want to buy it.

Quality Assurance

Why are these factors important?

- If you produce a software system that has terrible quality, you lose because no one will want to buy it.
- If on the other hand you spend infinite time, extremely large effort, and huge sums of money to build the absolutely perfect piece of software, then it's going to take so long to complete and it will be so expensive to produce that you'll be out of business anyway.

Quality Assurance

Why are these factors important?

- If you produce a software system that has terrible quality, you lose because no one will want to buy it.
- If on the other hand you spend infinite time, extremely large effort, and huge sums of money to build the absolutely perfect piece of software, then it's going to take so long to complete and it will be so expensive to produce that you'll be out of business anyway.
- Either you missed the market window, or you simply exhausted all your resources.

Quality Assurance

- Some aim to that magical middle ground where the product is good enough not to be rejected right away, but also not the object of so much perfectionism and so much work that it would take too long or cost too much to complete.

Quality Assurance

- Some aim to that magical middle ground where the product is good enough not to be rejected right away, but also not the object of so much perfectionism and so much work that it would take too long or cost too much to complete.
- So-called `good enough' software tends to deliver high quality functions and features that end-users desire, but at the same time may risk delivering other more obscure or specialized functions and features that contain bugs.

Quality Assurance

- Some aim to that magical middle ground where the product is good enough not to be rejected right away, but also not the object of so much perfectionism and so much work that it would take too long or cost too much to complete.
- So-called `good enough' software tends to deliver high quality functions and features that end-users desire, but at the same time may risk delivering other more obscure or specialized functions and features that contain bugs.
- Large companies can get away with serious bugs at launch (e.g. Activision-Blizzard and Diablo 3)—but, smaller indies risk permanent damage to their reputation.

Activity

- Organise into your COMP150 groups
- Read `http://blog.codinghorror.com/code-smells/`
- As a group, identify **three** code smells that are present in your COMP150 game code
- Discuss how you might go about refactoring to remove the smells

FALMOUTH
UNIVERSITY

Test driven development

Unit testing

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

Unit testing

- A **unit test** or **test case** 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** or **test case** 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
- The following might be unit tests for a `factorial` function:
  - `factorial(1)== 1`
  - `factorial(2)== 2`
  - `factorial(3)== 6`
  - `factorial(4)== 24`

Why do unit testing?

- Can find problems that normal testing misses

Why do unit testing?

- Can find problems that normal testing misses
- **Bottom-up** testing — if the **parts** work properly, it's easier to make the **whole** work properly

Why do unit testing?

- Can find problems that normal testing misses
- **Bottom-up** testing — if the **parts** work properly, it's easier to make the **whole** work properly
- When code is **changed**, can verify that nothing was broken

Caveats

- Have to spend time writing tests

Caveats

- Have to spend time writing tests
  - Not really a drawback — good unit tests will probably **save** more time in debugging than it takes to write them

Caveats

- Have to spend time writing tests
  - Not really a drawback — good unit tests will probably **save** more time in debugging than it takes to write them
- Can give a false sense of security

Caveats

- Have to spend time writing tests
  - Not really a drawback — good unit tests will probably **save** more time in debugging than it takes to write them
- Can give a false sense of security
  - Unit tests can't cover 100% of a complex program — **not a substitute** for other forms of testing

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?

- All the benefits of **unit testing**, plus...

Why TDD?

- All the benefits of **unit testing**, plus...
- Often easier to convert a **user story** into test cases rather than directly into code

Why TDD?

- All the benefits of **unit testing**, plus...
- 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?

- All the benefits of **unit testing**, plus...
- 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?

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