Falling in LOVE with Unit Testing

Joe Skeen

Slides live at ***https://world-class-engineers.github.io/conf42-rustlang-2023-unit-testing***

Unit testing provides benefits

- Early bug detection
- Regression prevention
- Continuous Integration
- Refactoring support
- Better design
- Better code quality
- Living Documentation
- and more!

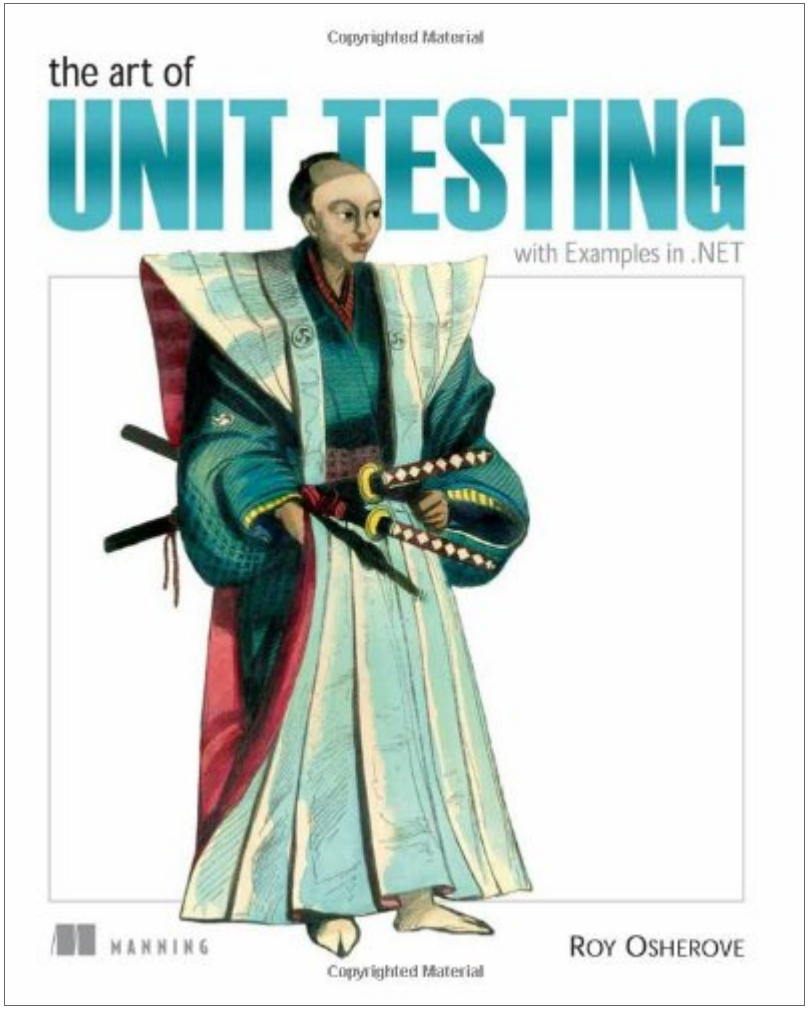
> *The more effort I put into testing the product conceptually at the start of the process, the less effort I [have] to put into manually testing the product at the end because fewer bugs ... emerge as a result.*

Trish Khoo, Director of Engineering at Octopus Deploy

Unit testing friction

- Forced to do it
- Not allowed the time
- Never learned how to do it well

My Journey

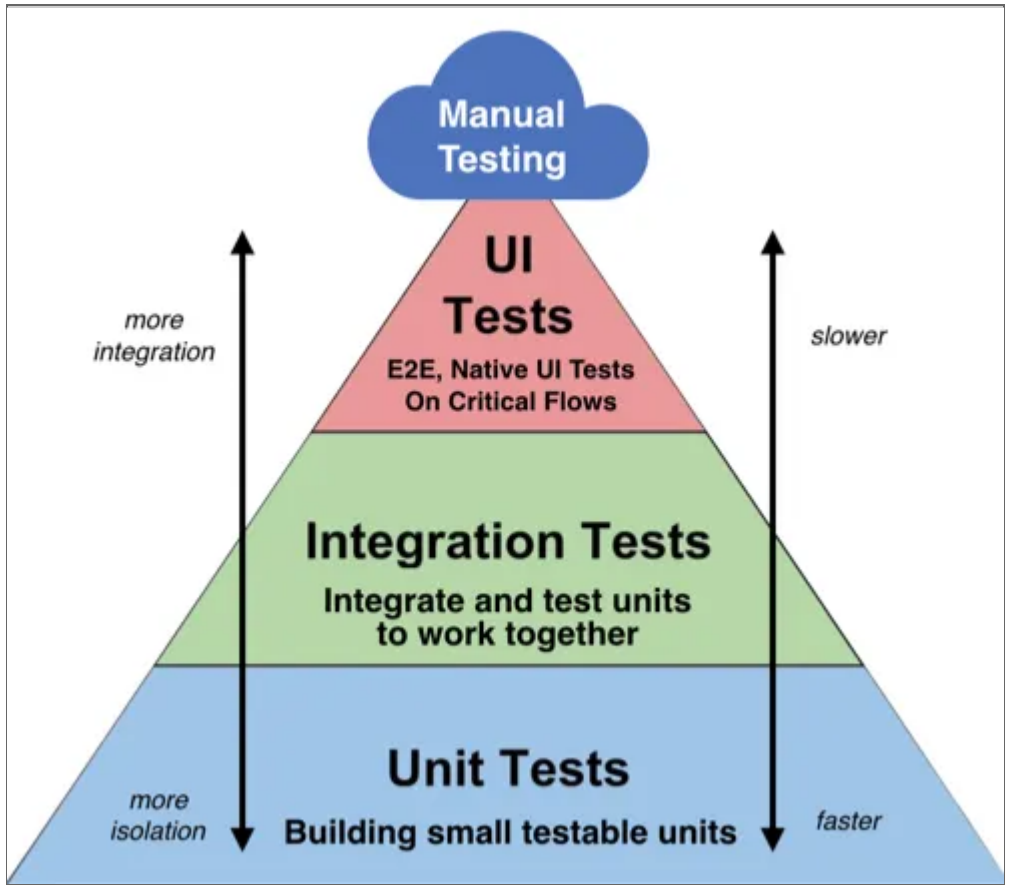


(not a sponsor)

WORLD-CLASS ENGINEERS


WORLD-CLASS
ENGINEERS

What is a unit test?



Credit: Moke Cohn, Martin Fowler, and Lawrence Tan

WORLD-CLASS ENGINEERS

Key 1: Break it down!

image by craiyon.com

WORLD-CLASS ENGINEERS

Privacy Knob Mockup

Image from Amazon (not a sponsor)

WORLD-CLASS ENGINEERS

A privacy doorknob, when the push button is not pressed, when the user turns the inside knob, should also turn the outside knob.

***GIVEN** the push button is not pressed*
***WHEN** the user turns the inside knob*
***THEN** the outside knob should also turn*

A privacy doorknob, when the push button is not pressed, when the user turns the inside knob, ***should also turn the outside knob.***

A privacy doorknob, when the push button is not pressed, when the user turns the inside knob, ***should retract the latch bolt.***

*A privacy doorknob, when the push button is not pressed, when the user turns the inside knob **clockwise**, should also turn the outside knob **counterclockwise**.*

*A privacy doorknob, when the push button is not pressed, when the user turns the inside knob **counterclockwise**, should also turn the outside knob **clockwise**.*

A privacy doorknob, when the push button is not pressed, when the user turns the outside knob clockwise, should also turn the inside knob counterclockwise.

A privacy doorknob, when the push button is not pressed, when the user turns the outside knob counterclockwise, should also turn the inside knob clockwise.

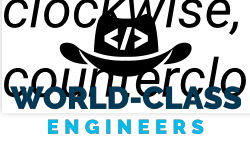
A privacy doorknob, when the push button is not pressed, when the user turns the outside knob, should retract the latch bolt.

A privacy doorknob, when the push button is not pressed, when the user turns the inside knob clockwise, should also turn the outside knob counterclockwise.

A privacy doorknob, when the push button is not pressed, when the user turns the inside knob counterclockwise, should also turn the outside knob clockwise.

A privacy doorknob, when the push button is not pressed, when the user turns the inside knob, should retract the latch bolt.

A privacy doorknob, when the push button is not pressed, when the user turns the outside knob clockwise, should also turn the inside knob counterclockwise.



A privacy doorknob, when the push button is not pressed, when the user turns the outside knob counterclockwise, should also turn the inside knob clockwise.

A privacy doorknob, when the push button is not pressed, when the user turns the outside knob, should retract the latch bolt.

Key 2: Care about test code quality

i.e. DRY

- A privacy doorknob,
- when the push button is not pressed,
  - when the user turns the inside knob clockwise,
  - should also turn the outside knob counterclockwise.
  - should retract the latch bolt.
  - when the user turns the inside knob counterclockwise,
  - should also turn the outside knob clockwise.
  - should retract the latch bolt.
  - when the user turns the outside knob clockwise,
  - should also turn the inside knob counterclockwise.
  - should retract the latch bolt.
  - when the user turns the outside knob counterclockwise,
  - should also turn the inside knob clockwise.
  - should retract the latch bolt.

- A privacy doorknob,
- when the push button is pressed,
  - when the user tries to turn the outside knob clockwise,
  - should not turn the outside knob at all.
  - should not turn the inside knob at all.
  - should not retract the latch bolt.
  - when the user tries to turn the outside knob counterclockwise,
  - should not turn the outside knob at all.
  - should not turn the inside knob at all.
  - should not retract the latch bolt.
  - when the user tries to turn the inside knob clockwise,
  - should pop the push button out.
  - should turn the inside knob clockwise.
  - should turn the outside knob counterclockwise.
  - should retract the latch bolt.
  - when the user tries to turn the inside knob counterclockwise,
  - should pop the push button out.
  - should turn the inside knob counterclockwise.

- should turn the outside knob clockwise.
- should retract the latch bolt.

Other use cases

- A privacy doorknob,
- when the push button is pressed,
  - when the user tries to close the door (press the latch bolt)
  - should retract the latch bolt.
  - when the user inserts a long pin into the hole on the outside knob,
  - should pop the push button out.

Exceptional use cases

- A privacy doorknob,
- when the button is pressed,
  - when the user uses excessive force to try to turn the outside knob,
  - the knob should not break.

Key 3: Focus on what matters

Key 4: Use AAA

- **Arrange:** Initialize object, set properties
- **Act:** Call the function you are testing
- **Assert:** Verify the results

Enough talk, let's see the code!

```
pub struct PrivacyDoorKnob {
  button_is_pushed: bool,
}
impl PrivacyDoorKnob {
  fn new() -> PrivacyDoorKnob { /*TODO*/ }
  pub fn turn_inside_knob(&mut self, direction: RotationDirection) -> KnobInteractionResult { /*TODO*/ }
  pub fn turn_outside_knob(&self, direction: RotationDirection) -> KnobInteractionResult { /*TODO*/ }
  pub fn insert_pin_into_outside_knob_hole(&mut self) { /*TODO*/ }
  pub fn is_button_pressed(&self) -> bool { /*TODO*/ }
  pub fn press_button(&mut self) { /*TODO*/ }
}

#[derive(PartialEq, Debug)]
struct KnobInteractionResult {
  inside_knob: Option<RotationDirection>,
  outside_knob: Option<RotationDirection>,
  latch_bolt: LatchBoltState,
}

#[derive(PartialEq, Debug)]
enum RotationDirection {
  Clockwise,
  Counterclockwise,
}
impl RotationDirection {
  fn opposite(&self) -> RotationDirection { /*TODO*/ }
}

#[derive(PartialEq, Debug)]
```

```
#[cfg(test)]
mod tests {
  use super::*;
}
```

```
#[cfg(test)]
mod tests {
  use super::*;

  // * A privacy doorknob,
  //     * when the push button is pressed,
  //         * when the user tries to turn the outside knob clockwise,
  //             * should not turn the outside knob at all.
  //             * should not turn the inside knob at all.
  //             * should not retract the latch bolt.
  //         * when the user tries to turn the outside knob counterclockwise,
  //             * should not turn the outside knob at all.
  //             * should not turn the inside knob at all.
  //             * should not retract the latch bolt.
  //         * when the user tries to turn the inside knob clockwise,
  //             * should pop the push button out.
  //             * should turn the inside knob clockwise.
  //             * should turn the outside knob counterclockwise.
  //             * should retract the latch bolt.
  //         * when the user tries to turn the inside knob counterclockwise,
  //             * should pop the push button out.
  //             * should turn the inside knob counterclockwise.
  //             * should turn the outside knob clockwise.
  //             * should retract the latch bolt.
}
```

```
#[cfg(test)]
mod tests {
  use super::*;

  mod when_the_push_button_is_pressed {
    use super::*;

    mod when_the_user_tries_to_turn_the_outside_knob_clockwise {
      use super::*;

      #[test]
      fn should_not_turn_the_outside_knob_at_all() { /* TODO */ }

      #[test]
      fn should_not_turn_the_inside_knob_at_all() { /* TODO */ }

      #[test]
      fn should_not_retract_the_latch_bolt() { /* TODO */ }
    }

    //         * when the user tries to turn the outside knob counterclockwise,
    //             * should not turn the inside knob at all.
    //             * should not retract the latch bolt.
    //         * when the user tries to turn the inside knob clockwise,
    //             * should pop the push button out.
    //             * should turn the inside knob clockwise.
    //             * should turn the outside knob counterclockwise.
    //             * should retract the latch bolt.
    //         * when the user tries to turn the inside knob counterclockwise
```

```
#[test]
fn should_not_turn_the_outside_knob_at_all() {
  // Arrange
  let mut knob = PrivacyDoorKnob::new();
  knob.press_button();

  // Act
  let result = knob.turn_outside_knob(RotationDirection::Clockwise);

  // Assert
  assert_eq!(result.outside_knob, None);
}

#[test]
fn should_not_turn_the_inside_knob_at_all() {
  // Arrange
  let mut knob = PrivacyDoorKnob::new();
  knob.press_button();

  // Act
  let result = knob.turn_outside_knob(RotationDirection::Clockwise);

  // Assert
  assert_eq!(result.inside_knob, None);
}

#[test]
fn should_not_retract_the_latch_bolt() {
  // Arrange
```

```
test privacy_door_knob::tests::when_the_push_button_is_pressed::when_the_user_tries_to_turn_the_outside_
test privacy_door_knob::tests::when_the_push_button_is_pressed::when_the_user_tries_to_turn_the_outside_
test privacy_door_knob::tests::when_the_push_button_is_pressed::when_the_user_tries_to_turn_the_outside_
```

```
#[cfg(test)]
mod tests {
  use super::*;

  mod when_the_push_button_is_pressed {
    use super::*;

    mod when_the_user_tries_to_turn_the_outside_knob_clockwise {
      use super::*;

      fn action(knob: &mut PrivacyDoorKnob) -> KnobInteractionResult {
        knob.turn_outside_knob(RotationDirection::Clockwise)
      }

      #[test]
      fn should_not_turn_the_outside_knob_at_all() {
        // Arrange
        let mut knob = PrivacyDoorKnob::new();
        knob.press_button();

        // Act
        let result = action(&mut knob);

        // Assert
        assert_eq!(result.outside_knob, None);
      }

      #[test]
      fn should_not_turn_the_inside_knob_at_all() {
```

```
#[cfg(test)]
mod tests {
  use super::*;

  mod when_the_push_button_is_pressed {
    use super::*;

    fn arrange() -> PrivacyDoorKnob {
      let mut knob = PrivacyDoorKnob::new();
      knob.press_button();
      knob
    }

    mod when_the_user_tries_to_turn_the_outside_knob_clockwise {
      use super::*;

      fn action(knob: &mut PrivacyDoorKnob) -> KnobInteractionResult {
        knob.turn_outside_knob(RotationDirection::Clockwise)
      }

      #[test]
      fn should_not_turn_the_outside_knob_at_all() {
        // Arrange
        let mut knob = arrange();

        // Act
        let result = action(&mut knob);

        // Assert
```

```
#[cfg(test)]
mod tests {
  use super::*;

  mod when_the_push_button_is_pressed {
    use super::*;

    fn arrange() -> PrivacyDoorKnob {
      let mut knob = PrivacyDoorKnob::new();
      knob.press_button();
      knob
    }

    mod when_the_user_tries_to_turn_the_outside_knob_clockwise {
      use super::*;

      fn action(knob: &mut PrivacyDoorKnob) -> KnobInteractionResult {
        knob.turn_outside_knob(RotationDirection::Clockwise)
      }

      #[test]
      fn should_not_turn_the_outside_knob_at_all() {
        let result = action(&mut arrange());
        assert_eq!(result.outside_knob, None);
      }

      #[test]
      fn should_not_turn_the_inside_knob_at_all() {
        let result = action(&mut arrange());
```

```
#[cfg(test)]
mod tests {
  use super::*;

  mod when_the_push_button_is_pressed {
    use super::*;

    fn arrange() -> PrivacyDoorKnob {
      let mut knob = PrivacyDoorKnob::new();
      knob.press_button();
      knob
    }

    mod when_the_user_tries_to_turn_the_outside_knob_clockwise {
      use super::*;

      fn action(knob: &mut PrivacyDoorKnob) -> KnobInteractionResult {
        knob.turn_outside_knob(RotationDirection::Clockwise)
      }

      macro_rules! it {
        ($name:ident, $field:ident, $value:expr) => {
          #[test]
          fn $name() {
            let result = action(&mut arrange());
            assert_eq!(result.$field, $value);
          }
        };
      }
```

```
#[cfg(test)]
mod tests {
  use super::*;

  macro_rules! it {
    ($name:ident, $arrange:ident, $action:ident, $field:ident, $value:expr) => {
      #[test]
      fn $name() {
        let result = $action(&mut $arrange());
        assert_eq!(result.$field, $value);
      }
    };
  }

  mod when_the_push_button_is_pressed {
    use super::*;

    fn arrange() -> PrivacyDoorKnob {
      let mut knob = PrivacyDoorKnob::new();
      knob.press_button();
      knob
    }

    mod when_the_user_tries_to_turn_the_outside_knob_clockwise {
      use super::*;

      fn action(knob: &mut PrivacyDoorKnob) -> KnobInteractionResult {
        knob.turn_outside_knob(RotationDirection::Clockwise)
      }
```

```
#[cfg(test)]
mod tests {
  use super::*;

  macro_rules! it {
    ($name:ident, $arrange:ident, $action:ident, $assertion:expr) => {
      #[test]
      fn $name() {
        let result = $action(&mut $arrange());
        $assertion(result);
      }
    };
  }

  mod when_the_push_button_is_pressed {
    use super::*;

    fn arrange() -> PrivacyDoorKnob {
      let mut knob = PrivacyDoorKnob::new();
      knob.press_button();
      knob
    }

    mod when_the_user_tries_to_turn_the_outside_knob_clockwise {
      use super::*;

      fn action(knob: &mut PrivacyDoorKnob) -> KnobInteractionResult {
        knob.turn_outside_knob(RotationDirection::Clockwise)
      }
```

```
#[cfg(test)]
mod tests {
  use super::*;

  macro_rules! it {
    ($name:ident, $arrange:ident, $action:ident, $assertion:expr) => {
      #[test]
      fn $name() {
        let result = $action(&mut $arrange());
        $assertion(result);
      }
    };
  }

  mod when_the_push_button_is_pressed {
    use super::*;

    fn arrange() -> PrivacyDoorKnob {
      let mut knob = PrivacyDoorKnob::new();
      knob.press_button();
      knob
    }

    mod when_the_user_tries_to_turn_the_outside_knob_clockwise {
      use super::*;

      fn action(knob: &mut PrivacyDoorKnob) -> KnobInteractionResult {
        knob.turn_outside_knob(RotationDirection::Clockwise)
      }
```

Don't get carried away

```
fn remove_vowels(input: String) -> String {/* TODO */}
```

Test:

- the empty String
- a small string with some vowels
- a small string with only vowels
- a small string with no vowels
- a very large String
- a string with complex unicode characters (i.e. emoji)

WORLD-CLASS
ENGINEERS

Go forth and unit test!

Contact me:
joe@worldclassengineers.dev
Thanks for watching!

Speaker notes