Design Principles and Design Patterns

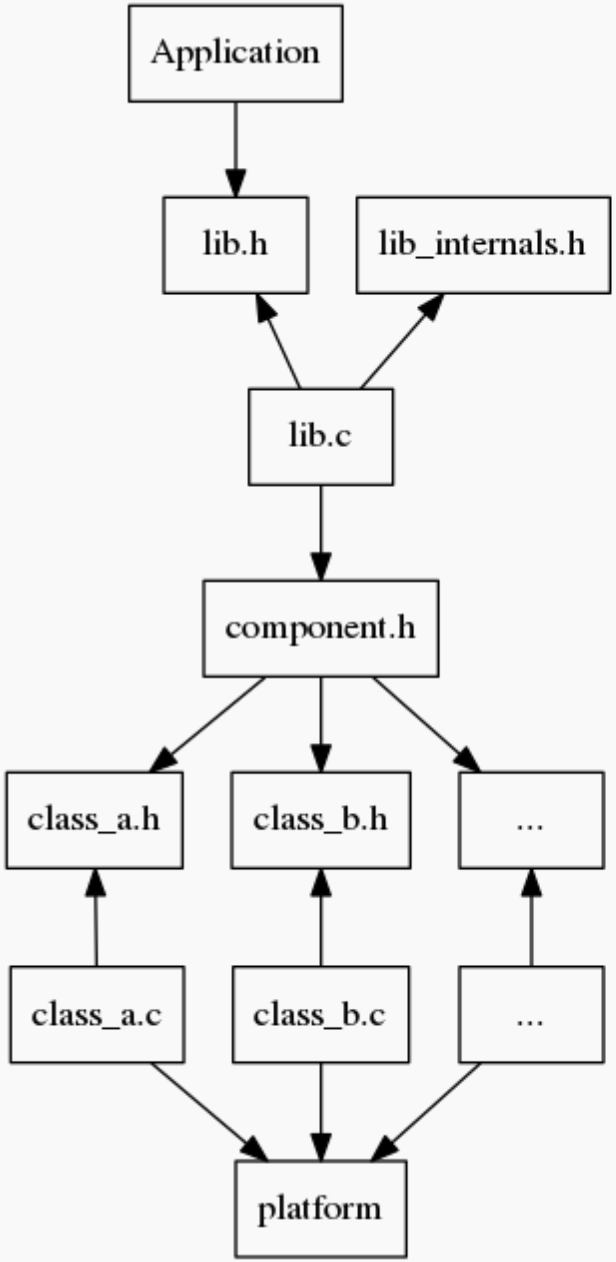
D. Ryan Bartling

May 29, 2018

Outline

Outline for section 1

Architecture and Dependencies



User's application code

Library code

Low level code,
written by implementor

Outline for section 2

Symptoms of Rotting Design

Symptoms of Rotting Design

Symptoms of Rotting Design

Symptoms of Rotting Design

Symptoms of Rotting Design

Symptoms of Rotting Design

1. Rigidity

Symptoms of Rotting Design

Symptoms of Rotting Design

1. Rigidity
2. Fragility

Symptoms of Rotting Design

Symptoms of Rotting Design

1. Rigidity
2. Fragility
3. Immobility

Symptoms of Rotting Design

Symptoms of Rotting Design

1. Rigidity
2. Fragility
3. Immobility
4. Viscosity

Rigidity

Rigidity

- Deficient in or devoid of flexibility

Rigidity

- Deficient in or devoid of flexibility
- Software for which extra effort is expended in order to make changes.

Rigidity

- Deficient in or devoid of flexibility
- Software for which extra effort is expended in order to make changes.
- The system is hard to change because every change forces many other changes to other parts of the system.

Rigidity

How it happens

Rigidity

How it happens

- Code written in a procedural way

Rigidity

How it happens

- Code written in a procedural way
- Lack of abstractions

Rigidity

How it happens

- Code written in a procedural way
- Lack of abstractions
- Solving a generic problem with implementation specific details

Rigidity

How it happens

- Code written in a procedural way
- Lack of abstractions
- Solving a generic problem with implementation specific details
- Spreading a single responsibility throughout several parts

Rigidity

How it happens

- Code written in a procedural way
- Lack of abstractions
- Solving a generic problem with implementation specific details
- Spreading a single responsibility throughout several parts
- When components need a lot of knowledge about each other in order to function

Rigidity

```
1  #include <stdint.h>
2
3  #define ADC_BITS (12)
4  #define ADC_DATA_SHIFT (2)
5  #define ADC_SIGN_CONVERSION (1)
6  #define RAW_ADC_BITS (15) // Sum of the above bits
7
8  #define LFSR_LENGTH (4)
9  #define LFSR_REPEATS (2)
10 #define CORRELATED_BITS (20) // ADC bits + lfsr length + log2(repeats)
11
12 typedef int16_t rpo_raw_adc_t;
13 typedef int24_t rpo_correlated_int_t;
14
15 #if sizeof(rpo_raw_adc_t) < (RAW_ADC_BITS / 2 + 1)
16 #error "rpo_raw_adc_t is too small to store ADC results"
17 #endif
18
19 #if sizeof(rpo_correlated_int_t) < (RAW_ADC_BITS / 2 + 1)
20 #error "rpo_correlated_int_t is too small to store correlated adc results"
21 #endif
```

Rigidity

```
1  #include <stdint.h>
2
3  #define ADC_BITS (14) // Changing this
4  #define ADC_DATA_SHIFT (2)
5  #define ADC_SIGN_CONVERSION (1)
6  #define RAW_ADC_BITS (17) // Changes this
7
8  #define LFSR_LENGTH (4)
9  #define LFSR_REPEATS (2)
10 #define CORRELATED_BITS (22) // Changes this
11
12 typedef int24_t rpo_raw_adc_t; // Changes this
13 typedef int24_t rpo_correlated_int_t;
14
15 #if sizeof(rpo_raw_adc_t) < (RAW_ADC_BITS / 2 + 1)
16 #error "rpo_raw_adc_t is too small to store ADC results"
17 #endif
18
19 #if sizeof(rpo_correlated_int_t) < (RAW_ADC_BITS / 2 + 1)
20 #error "rpo_correlated_int_t is too small to store correlated adc results"
21 #endif
```

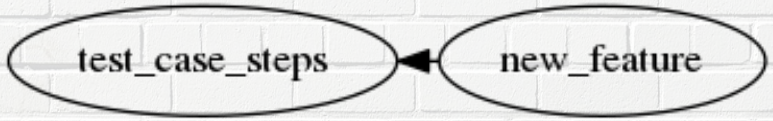
Rigidity

Refactor to reduce rigidity

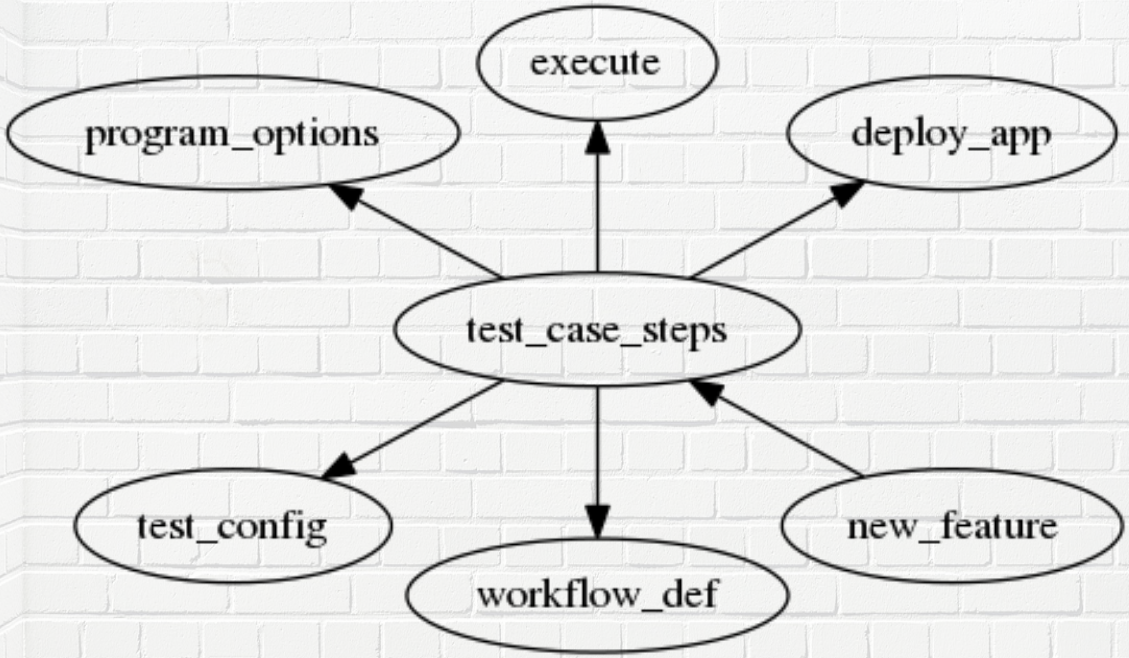
```
1  #include <stdint.h>
2
3  #define ADC_BITS (14)
4  #define ADC_DATA_SHIFT (2)
5  #define ADC_SIGN_CONVERSION (1)
6  #define RAW_ADC_BITS (ADC_BITS + ADC_DATA_SHIFT + ADC_SIGN_CONVERSION)
7  typedef_min_int(rpo_raw_adc_t, RAW_ADC_BITS);
8
9  #define LFSR_LENGTH (4)
10 #define LFSR_REPEATS (2)
11 #define CORRELATED_BITS (RAW_ADC_BITS + LFSR_LENGTH + log_2(LFSR_REPEATS))
12 typedef_min_int(rpo_correlated_int_t, CORRELATED_BITS);
```

Rigidity

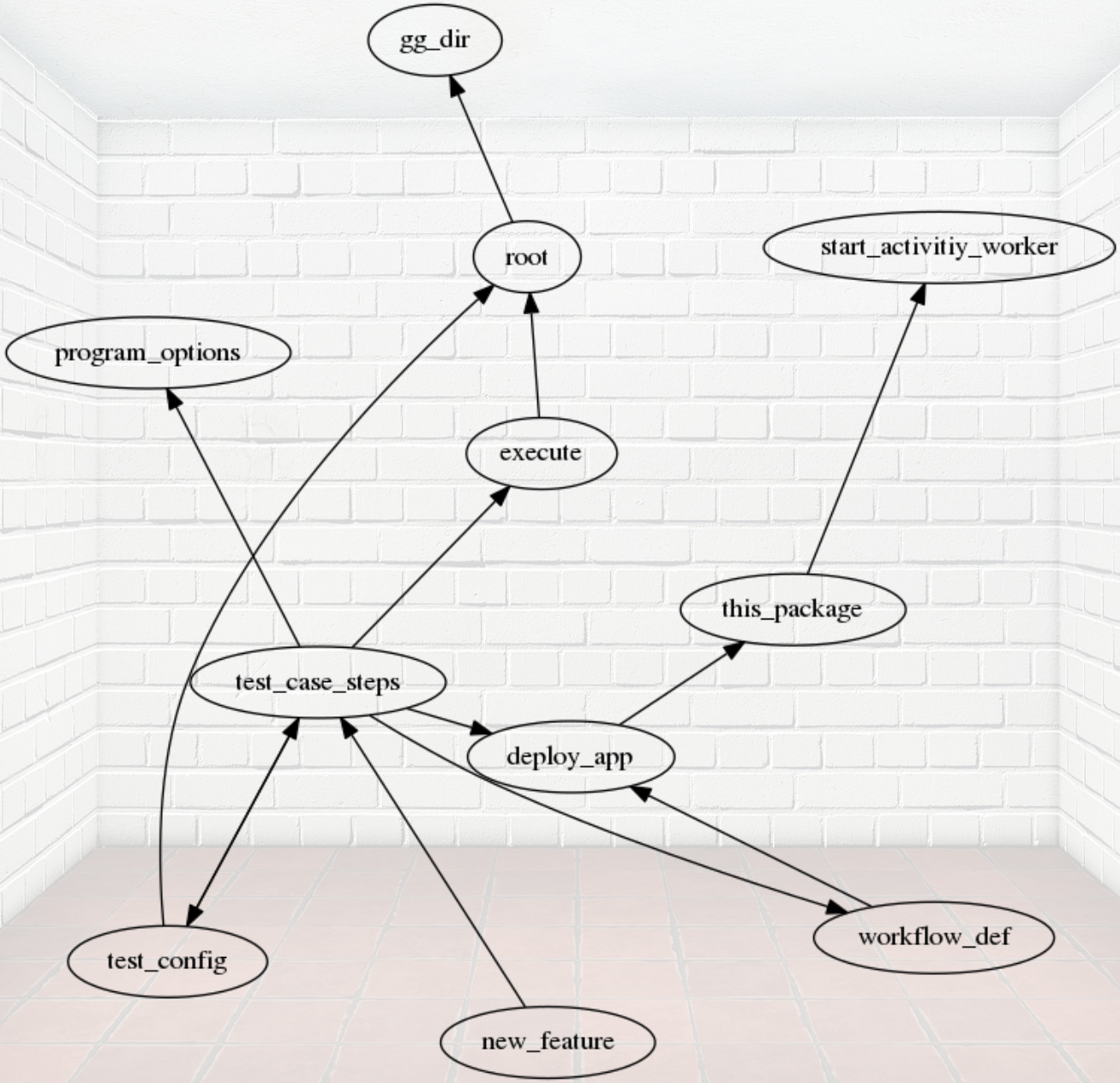
Rigidity



Rigidity


execute
program_options
deploy_app
test_case_steps
test_config
workflow_def
new_feature

Rigidity


gg_dir
root
start_activitiy_worker
program_options
execute
this_package
test_case_steps
deploy_app
workflow_def
test_config
new_feature

Rigidity

How to avoid it

Rigidity

How to avoid it

- Break the code into smaller, self-contained concepts

Rigidity

How to avoid it

- Break the code into smaller, self-contained concepts
- Solve the details and provide a problem oriented abstraction

Rigidity

How to avoid it

- Break the code into smaller, self-contained concepts
- Solve the details and provide a problem oriented abstraction
- Solving a generic problem with implementation specific details

Rigidity

How to avoid it

- Break the code into smaller, self-contained concepts
- Solve the details and provide a problem oriented abstraction
- Solving a generic problem with implementation specific details
- Write DRY code (Don't repeat yourself)

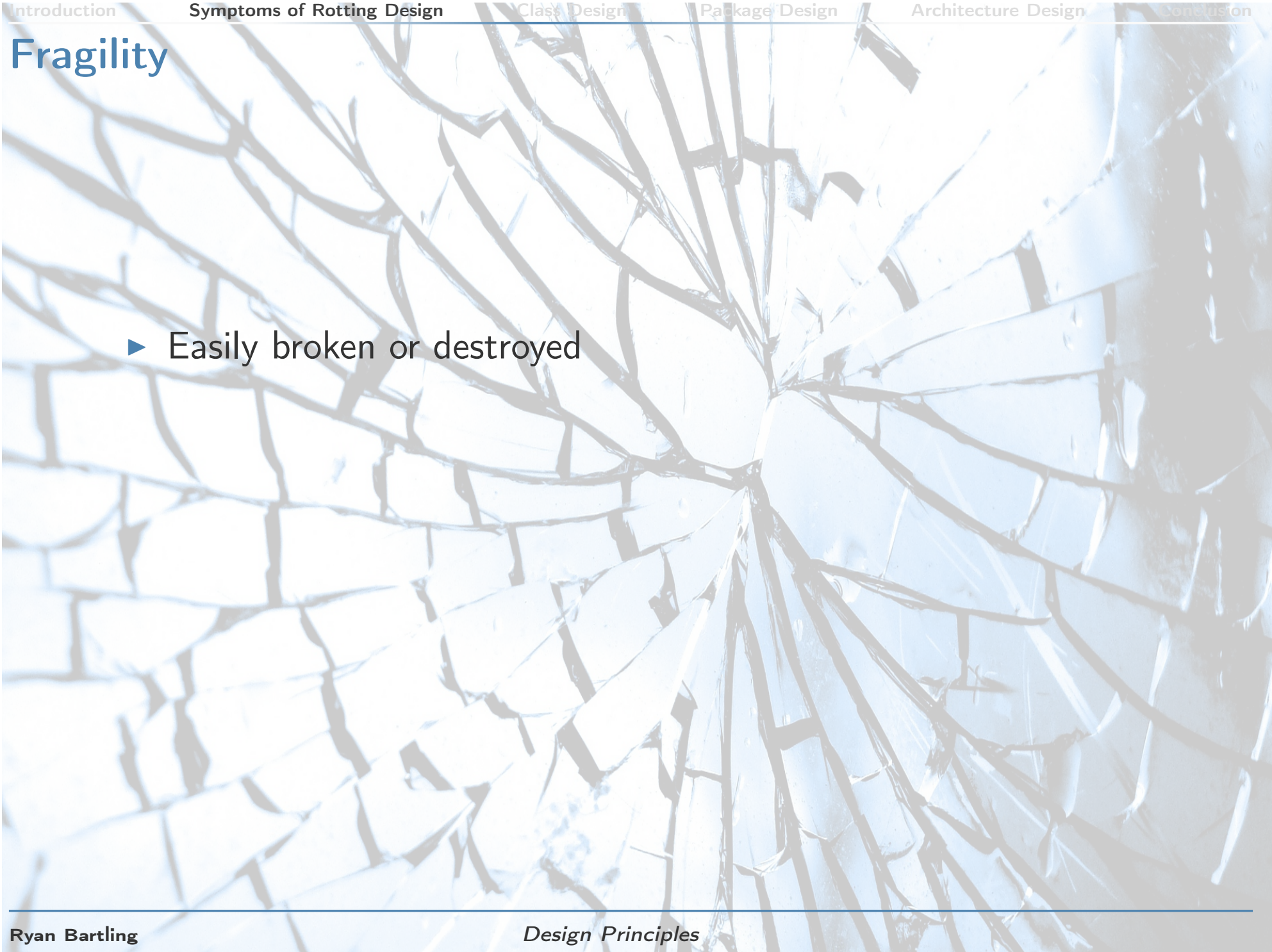
Rigidity

How to avoid it

- Break the code into smaller, self-contained concepts
- Solve the details and provide a problem oriented abstraction
- Solving a generic problem with implementation specific details
- Write DRY code (Don't repeat yourself)
- Define the code in logical pieces. Set boundaries and responsibilities.

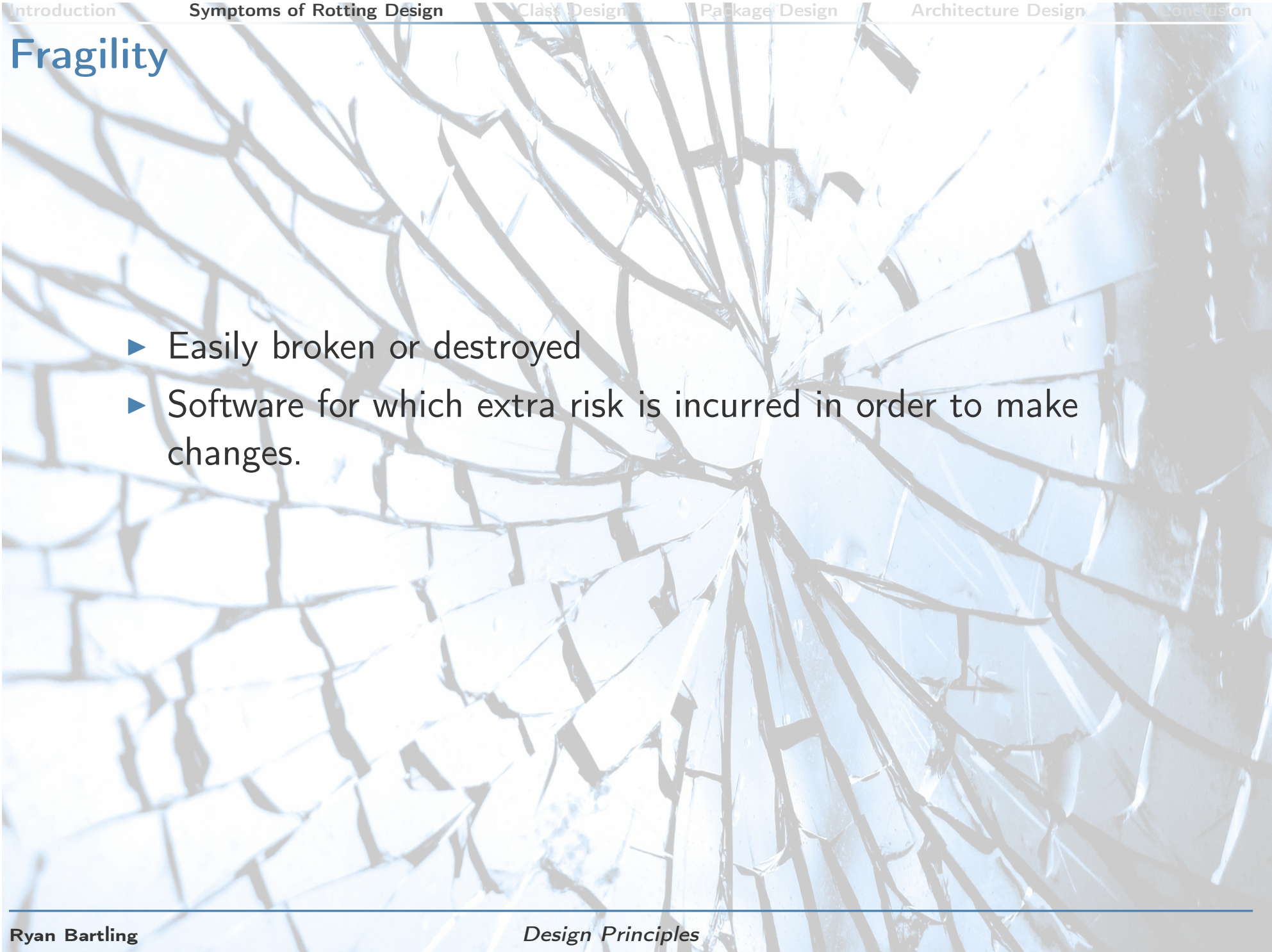
Fragility

Fragility



- Easily broken or destroyed

Fragility

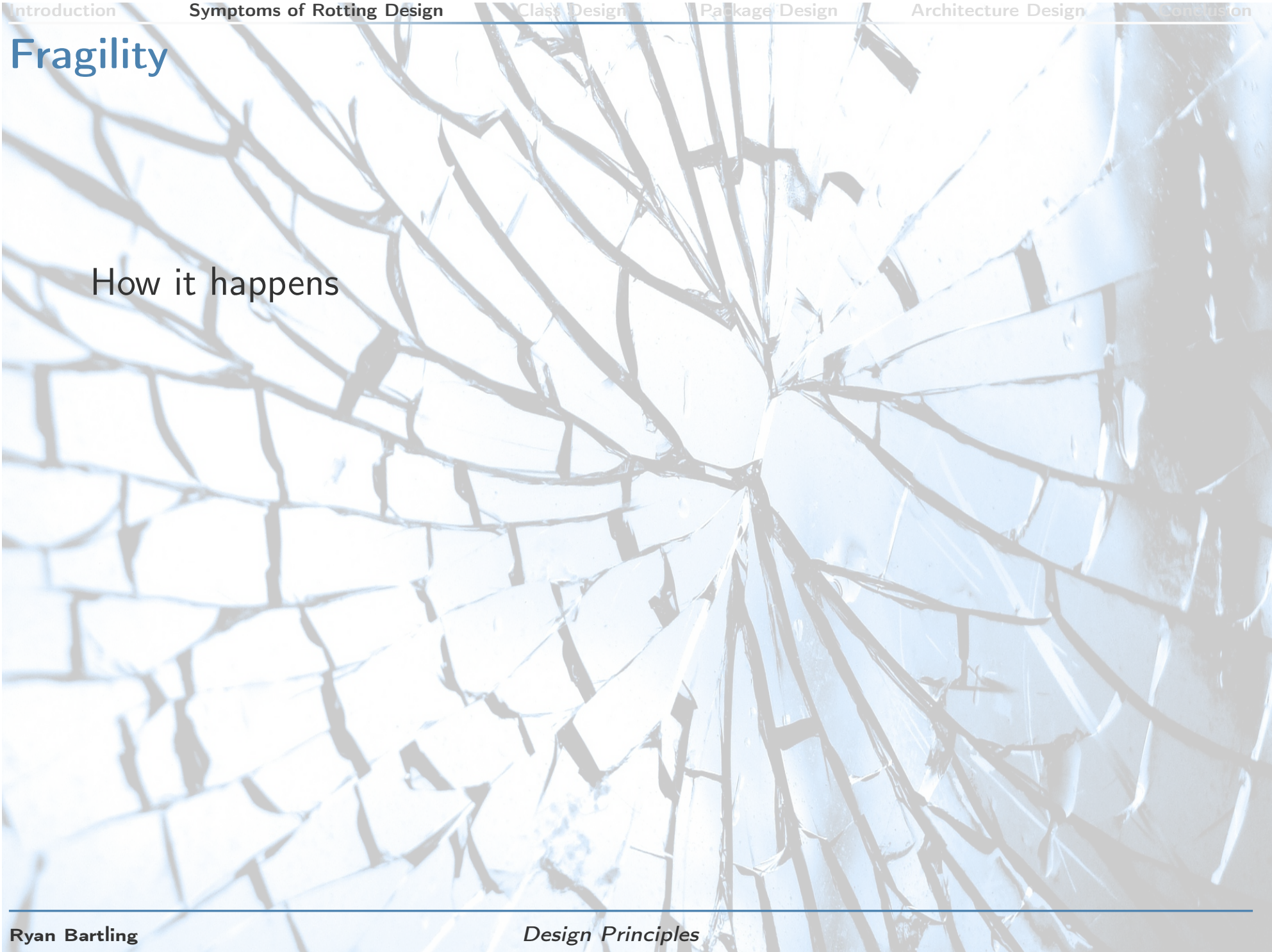


- Easily broken or destroyed
- Software for which extra risk is incurred in order to make changes.

Fragility

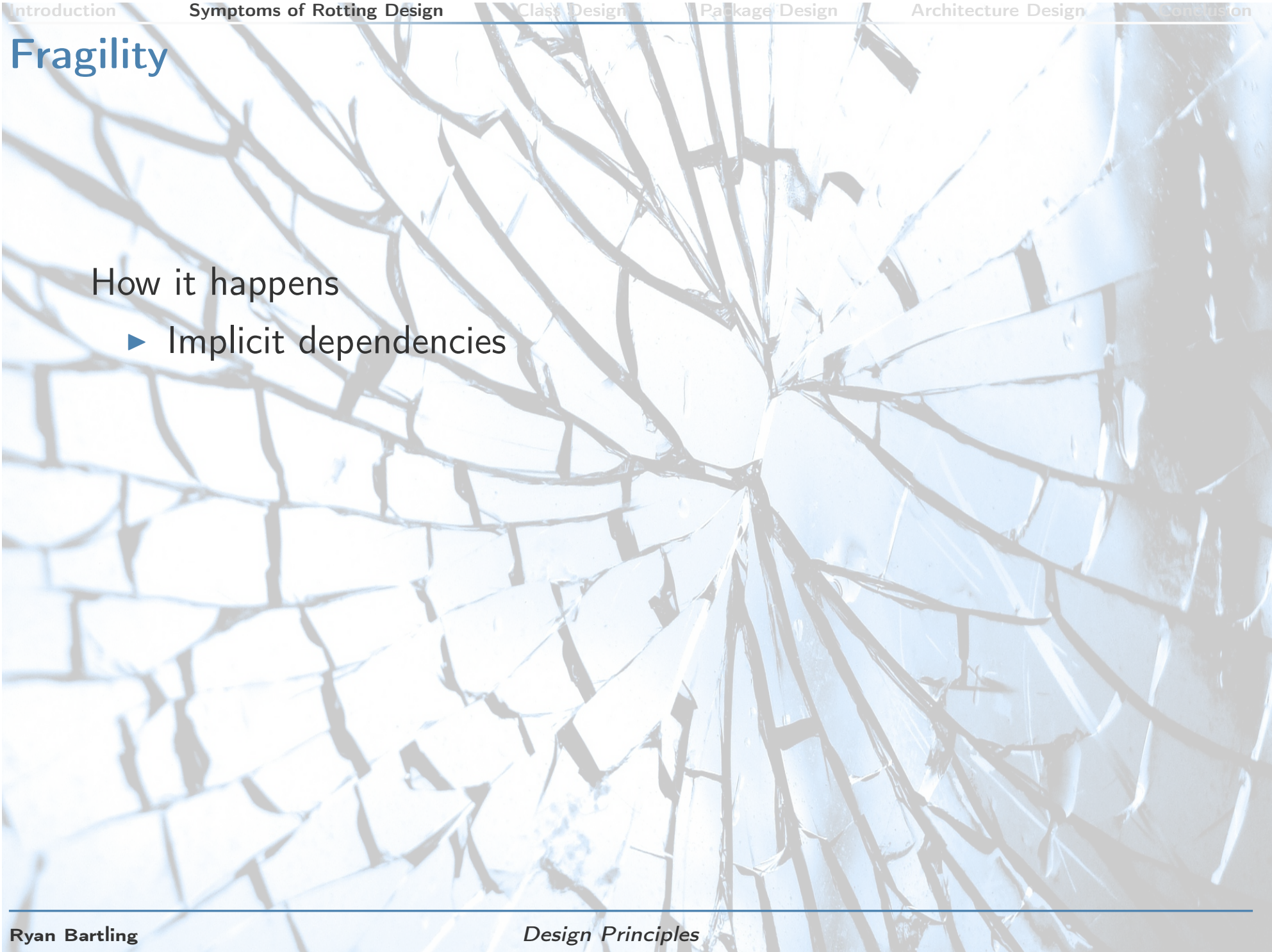
- Easily broken or destroyed
- Software for which extra risk is incurred in order to make changes.
- Changes cause the system to break in places that have no conceptual relationship to the part that was changed.

Fragility



How it happens

Fragility



How it happens

- Implicit dependencies

Fragility

How it happens

- Implicit dependencies
- Unmanaged shared resources

Fragility

How it happens

- Implicit dependencies
- Unmanaged shared resources
- Relying on implementation details

Fragility

How it happens

- Implicit dependencies
- Unmanaged shared resources
- Relying on implementation details
- Relying upon side effects of operations

Fragility

How it happens

- Implicit dependencies
- Unmanaged shared resources
- Relying on implementation details
- Relying upon side effects of operations
- Reaching past abstraction layers

Fragility

How it happens

- Implicit dependencies
- Unmanaged shared resources
- Relying on implementation details
- Relying upon side effects of operations
- Reaching past abstraction layers
- Unmanaged complexity

Fragility

```
1  void sdcard_init(void) {
2      spi_init(mode_0, card_cs_pin);
3      fat_init();
4  }
5
6  void sensor_init(void) {
7      spi_init(mode_0, sensor_cs_pin);
8      spi_write(SENSOR_CONFIGURATION, sensor_cs_pin);
9  }
```

Fragility

Changing the sensor to use mode 1...

```
1  void sdcard_init(void) {
2      spi_init(mode_0, card_cs_pin);
3      fat_init();
4  }
5
6  void sensor_init(void) {
7      spi_init(mode_1, sensor_cs_pin); // Breaks the sd card
8      spi_write(SENSOR_CONFIGURATION, sensor_cs_pin);
9  }
```

...Breaks the sd card (when sensor is initialized after the sd card)

Fragility

How to avoid it

Fragility

How to avoid it

- Implicit dependencies

Fragility

How to avoid it

- Implicit dependencies
- Law of Demeter: principle of least knowledge

Fragility

How to avoid it

- Implicit dependencies
- Law of Demeter: principle of least knowledge
- Avoid side effects, and don't rely on the side effects of other modules

Fragility

How to avoid it

- Implicit dependencies
- Law of Demeter: principle of least knowledge
- Avoid side effects, and don't rely on the side effects of other modules
- Rely on the published API

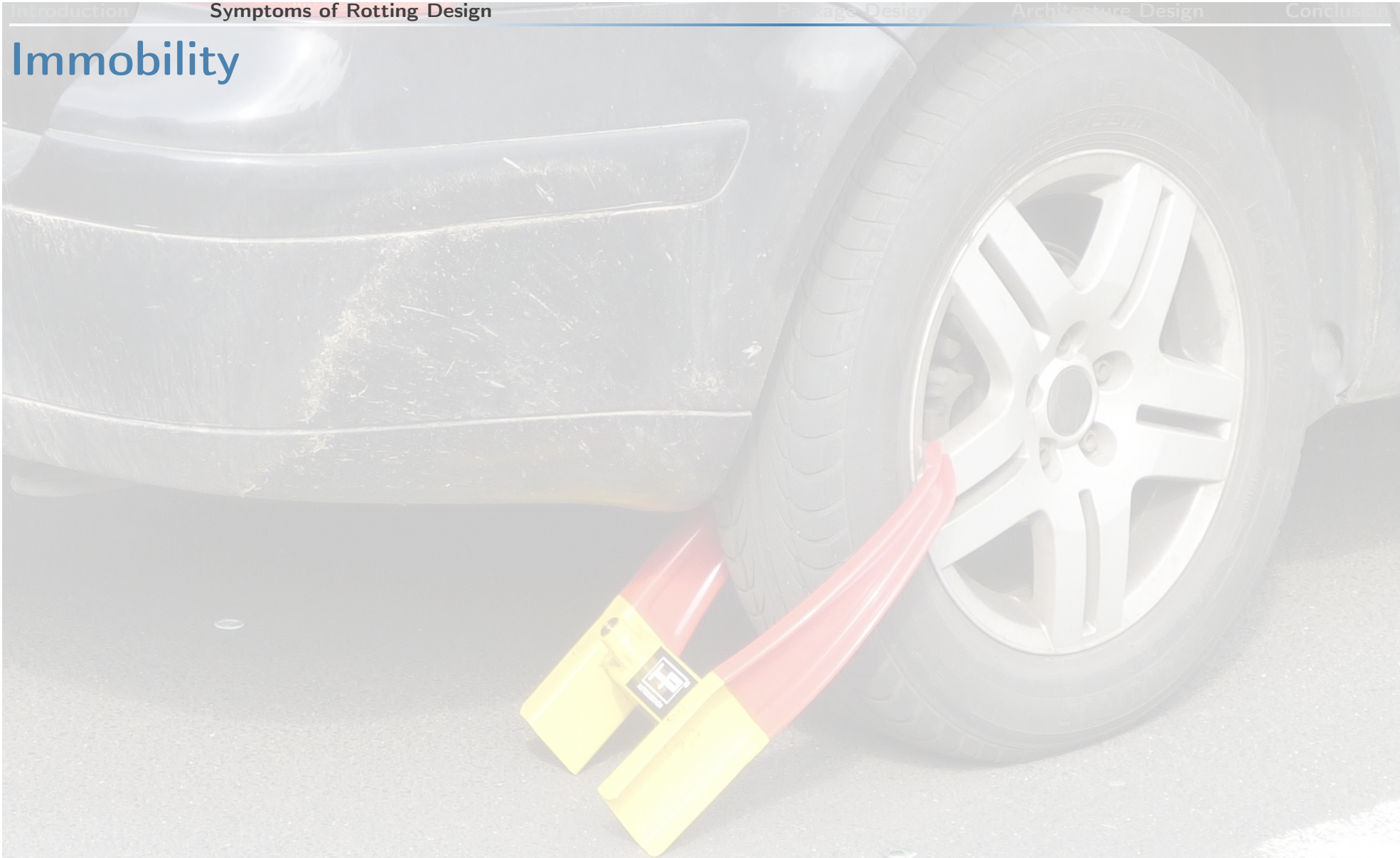
Fragility

How to avoid it

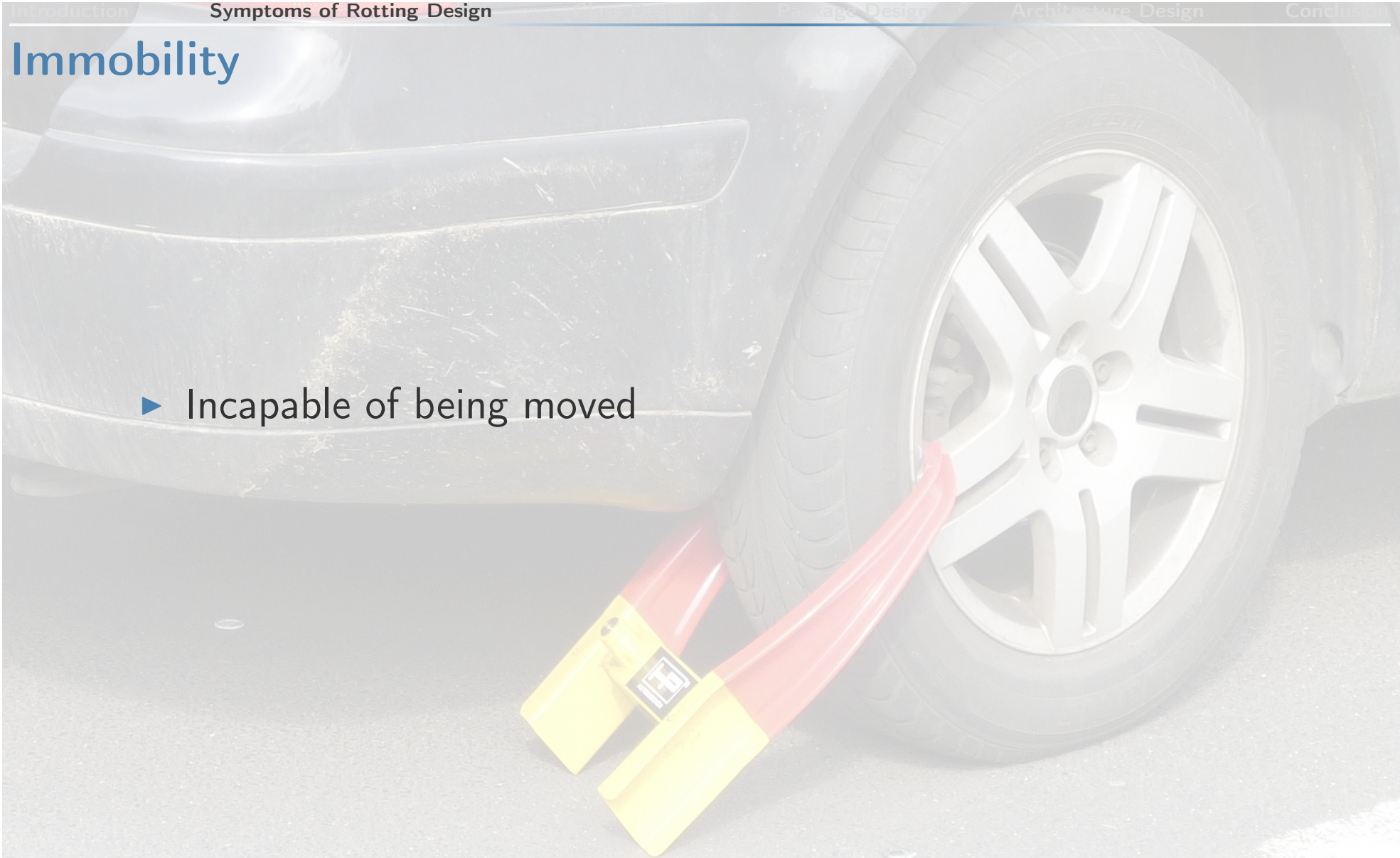
- Implicit dependencies
- Law of Demeter: principle of least knowledge
- Avoid side effects, and don't rely on the side effects of other modules
- Rely on the published API
- Invent and **simplify**

Immobility

Immobility



Immobility



- Incapable of being moved

Immobility

- Incapable of being moved
- Software for which extra effort is required in order to reuse.

Immobility

- Incapable of being moved
- Software for which extra effort is required in order to reuse.
- It is hard to disentangle the system into components that can be reused in other systems.

Immobility

How it happens

- Direct dependency on things you don't own

Immobility

How it happens

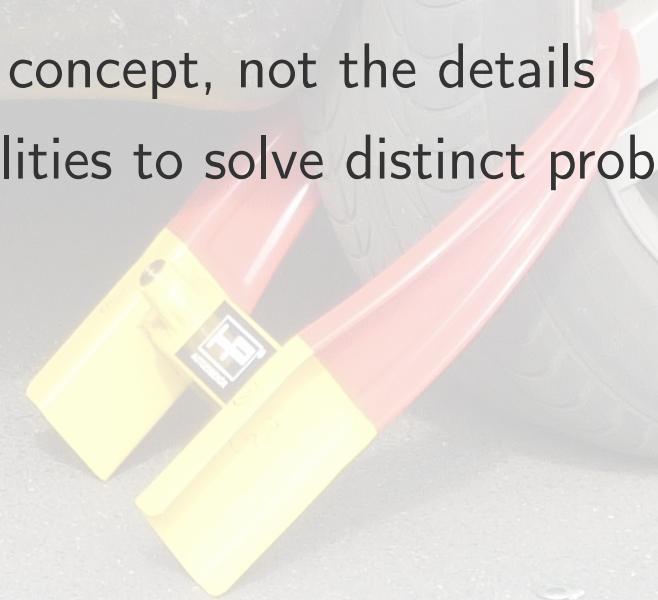
- Direct dependency on things you don't own
- Too many responsibilities

Immobility

How it happens

- Depend upon the concept, not the details

Immobility



How it happens

- Depend upon the concept, not the details
- Reduce responsibilities to solve distinct problems

Viscosity

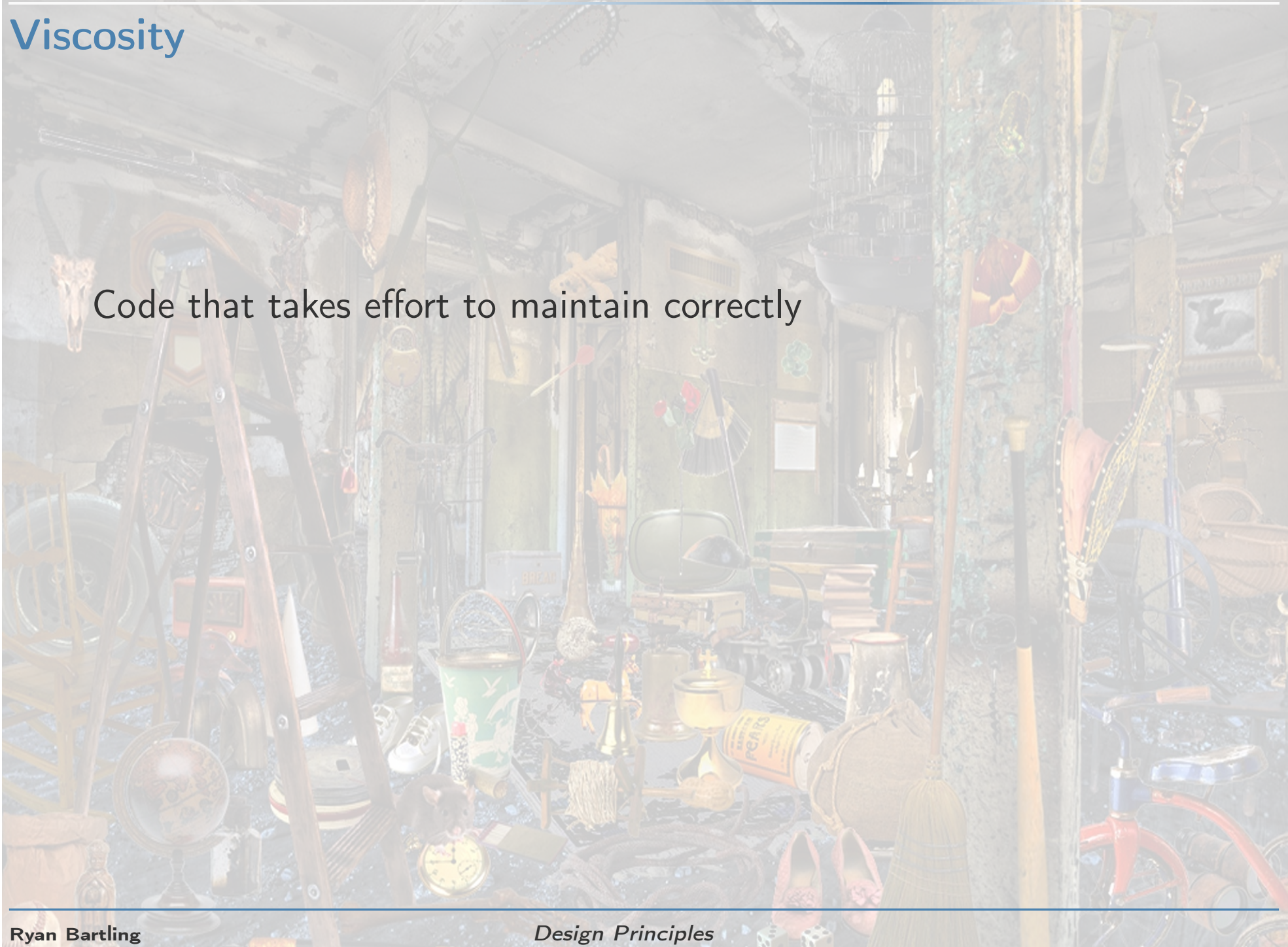
Viscosity

- Having or characterized by a high resistance to flow

Viscosity

- Having or characterized by a high resistance to flow
- Software for which extra effort is required in order to reuse.

Viscosity



Code that takes effort to maintain correctly

Viscosity

Code that takes effort to maintain correctly

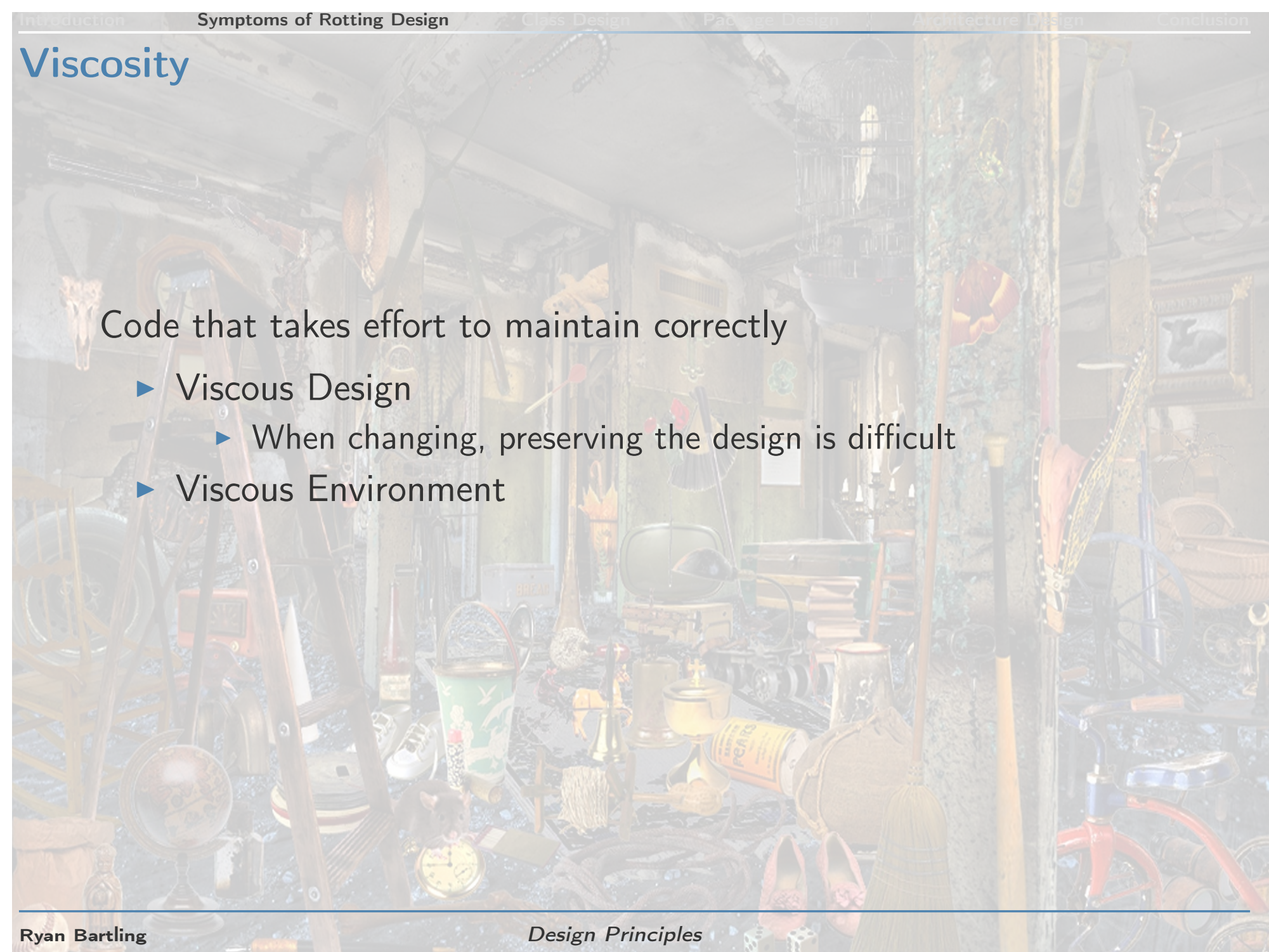
- Viscous Design

Viscosity

Code that takes effort to maintain correctly

- Viscous Design
- Viscous Environment

Viscosity



Code that takes effort to maintain correctly

- Viscous Design
  - When changing, preserving the design is difficult
- Viscous Environment

Viscosity

Code that takes effort to maintain correctly

- Viscous Design
  - When changing, preserving the design is difficult
- Viscous Environment
  - Long builds

Viscosity

Code that takes effort to maintain correctly

- Viscous Design
  - When changing, preserving the design is difficult
- Viscous Environment
  - Long builds
  - Slow Tests

Outline for section 3

Principles of Object Oriented Class Design

SOLID Principles

Principles of Object Oriented Class Design

SOLID Principles

- **S***ingle Responsibility Principle* (SRP)

Principles of Object Oriented Class Design

SOLID Principles

- **S***ingle Responsibility Principle* (SRP)
- **O***pen Closed Principle* (OCP)

Principles of Object Oriented Class Design

SOLID Principles

- **S***ingle Responsibility Principle* (SRP)
- **O***pen Closed Principle* (OCP)
- **L***iskov Substitution Principle* (LSP)

Principles of Object Oriented Class Design

SOLID Principles

- **S***ingle Responsibility Principle* (SRP)
- **O***pen Closed Principle* (OCP)
- **L***iskov Substitution Principle* (LSP)
- **I***nterface Segregation Principle* (ISP)

Principles of Object Oriented Class Design

SOLID Principles

- **S***ingle Responsibility Principle* (SRP)
- **O***pen Closed Principle* (OCP)
- **L***iskov Substitution Principle* (LSP)
- **I***nterface Segregation Principle* (ISP)
- **D***ependency Inversion Principle* (DIP)

Single Responsibility Principle

Single Responsibility Principle

- Cohesion

Single Responsibility Principle

- Cohesion
- Responsibility = Reason to change

title

Outline for section 4

Principles of Package Architecture

Package Principles

Principles of Package Architecture

Package Principles

- Package Cohesion

Principles of Package Architecture

Package Principles

- Package Cohesion
- Package Coupling

Principles of Package Architecture

Package Principles

- Package Cohesion
  - Release Reuse Equivalency Principle (REP)
- Package Coupling

Principles of Package Architecture

Package Principles

- Package Cohesion
  - Release Reuse Equivalency Principle (REP)
  - Common Closure Principle (CCP)
- Package Coupling

Principles of Package Architecture

Package Principles

- Package Cohesion
  - Release Reuse Equivalency Principle (REP)
  - Common Closure Principle (CCP)
  - Common Reuse Principle (CRP)
- Package Coupling

Principles of Package Architecture

Package Principles

- Package Cohesion
  - Release Reuse Equivalency Principle (REP)
  - Common Closure Principle (CCP)
  - Common Reuse Principle (CRP)
- Package Coupling
  - Acyclic Dependencies Principle (ADP)

Principles of Package Architecture

Package Principles

- Package Cohesion
  - Release Reuse Equivalency Principle (REP)
  - Common Closure Principle (CCP)
  - Common Reuse Principle (CRP)
- Package Coupling
  - Acyclic Dependencies Principle (ADP)
  - Stable Dependencies Principle (SDP)

Principles of Package Architecture

Package Principles

- Package Cohesion
  - Release Reuse Equivalency Principle (REP)
  - Common Closure Principle (CCP)
  - Common Reuse Principle (CRP)
- Package Coupling
  - Acyclic Dependencies Principle (ADP)
  - Stable Dependencies Principle (SDP)
  - Stable Abstractions Principle (SAP)

title

Outline for section 5

Principles of Package Architecture

Outline for section 6

Principles of Package Architecture

References

- https://fi.ort.edu.uy/innovaportal/file/2032/1/design_principles.pdf
- http://www.butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod
- http://notherdev.blogspot.com/2013/07/code-smells-rigidity.html
- https://dev.to/bob/how-do-you-know-your-code-is-bad
- http://staff.cs.utu.fi/~jounsmed/doos_06/slides/slides_060321.pdf
- https://softwareengineering.stackexchange.com/questions/357127/clear-examples-for-code-smells

Questions