Design Principles and Design Patterns

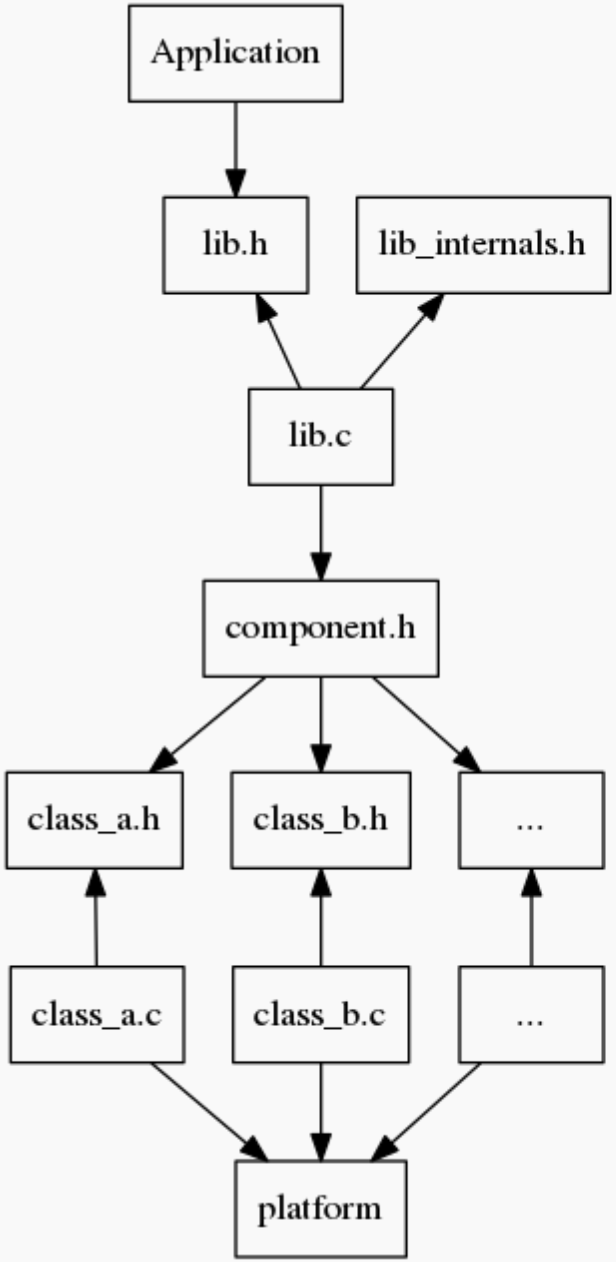
D. Ryan Bartling

May 16, 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 Code

Four Symptoms of Rotting Code

Symptoms of Rotting Code

Four Symptoms of Rotting Code

1. Rigidity

Symptoms of Rotting Code

Four Symptoms of Rotting Code

1. Rigidity
2. Fragility

Symptoms of Rotting Code

Four Symptoms of Rotting Code

1. Rigidity
2. Fragility
3. Immobility

Symptoms of Rotting Code

Four Symptoms of Rotting Code

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

Symptoms of Rotting Code

Four Symptoms of Rotting Code

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

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

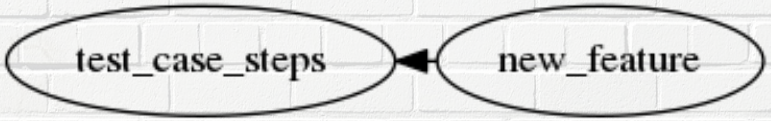
Rigidity

How it happens

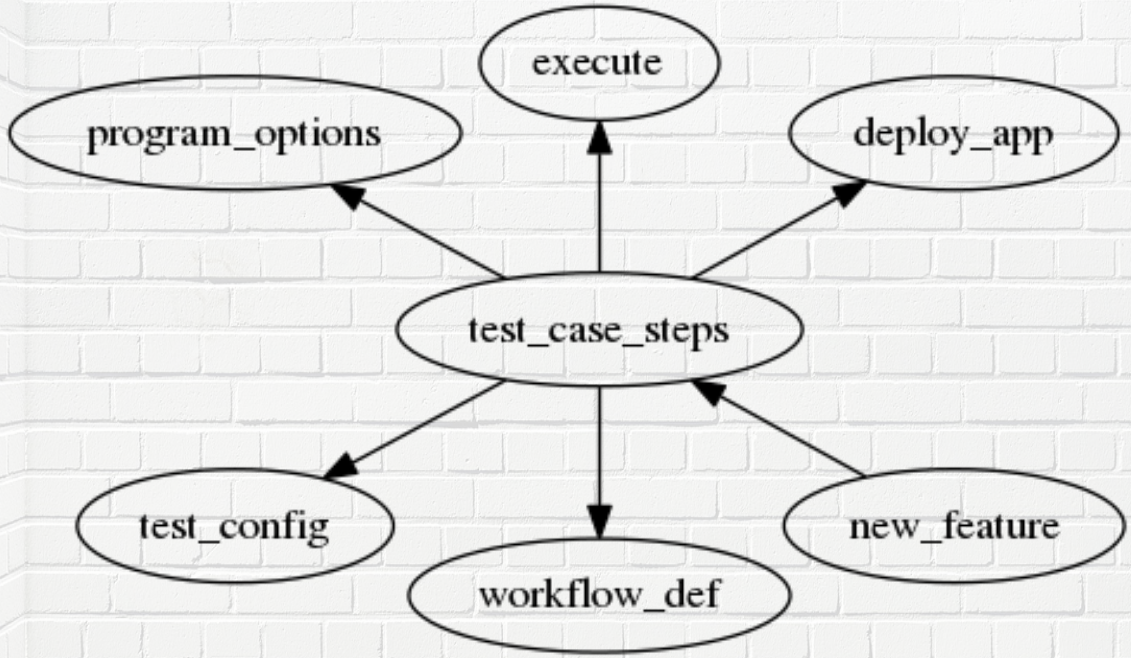
- Code written in a procedural way
- Lack of abstractions
- Solving a generic problem with implementation specific details
- Spreading a single responsibility throughout several parts
- When components need a lot of knowledge about each other in order to function

Rigidity

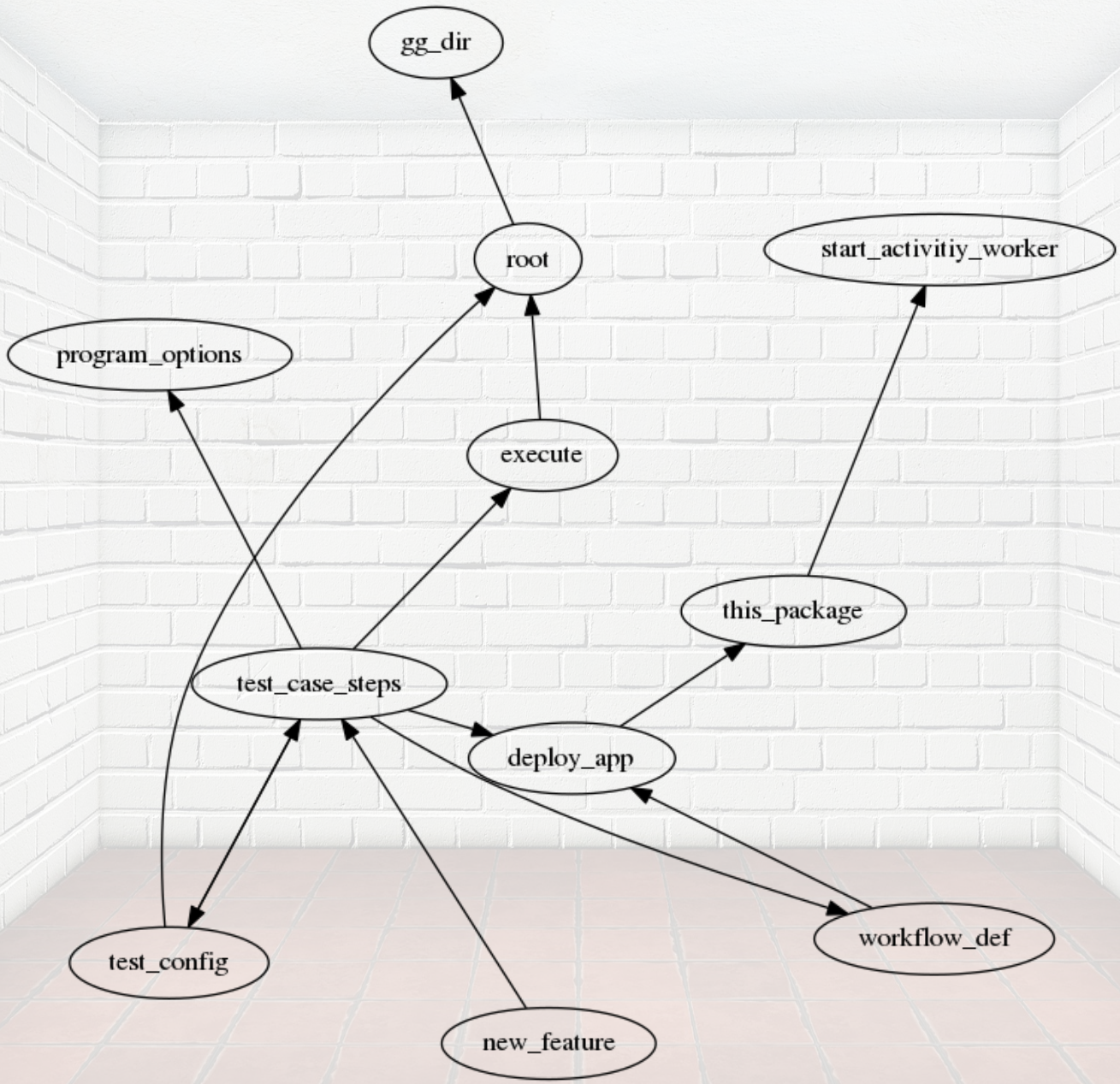
Rigidity



Rigidity


execute
program_options
deploy_app
test_case_steps
test_config
new_feature
workflow_def

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

- 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

How it happens

- Implicit dependencies

Fragility

How it happens

- Implicit dependencies
- Relying on implementation details

Fragility

How it happens

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

Fragility

How it happens

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

Fragility

How it happens

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

Fragility

```
int foo(void)
{
    // TODO: Write example of fragile code
}
```

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

- Incapable of being moved

Immobility

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

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

- Single Responsibility Principle (SRP)
- Open Closed Principle (OCP)
- Liskov Substitution Principle (LSP)
- Interface Segregation Principle (ISP)
- Dependency Inversion Principle (DIP)

Outline for section 4

Principles of Package Architecture

- Package Cohesion

Principles of Package Architecture

- Package Cohesion
- Package Coupling

Principles of Package Architecture

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

Principles of Package Architecture

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

Principles of Package Architecture

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

Principles of Package Architecture

- 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 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 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