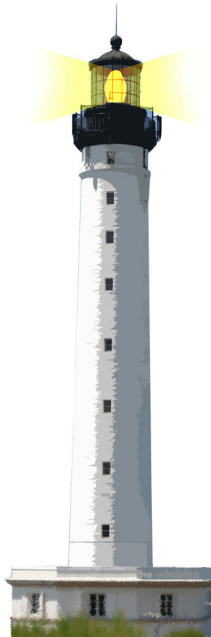
Reverse engineering

A key skill of pros developers

S. Ducasse and G. Polito



20 vs 80

- You get 4 times more probability to work on an old, strange, undocumented, architecturally drifted system than write a new one
- How to survive to
  - **obsolete/irrelevant** documentation,
  - **obscure architecture**, and
  - **lack** of experts?
- How to get insider information?

Reverse engineering

IEEE definition *"the process of analyzing a subject system to identify the system's components and their interrelationships and to create representations of the system in another form or at a higher level of abstraction"*

Different perspectives

- As a user
  - API?
- As an implementor
  - what are the invariants/design choice/tradeoffs?
- As an extender
  - Where to introduce a dispatch hook?

Different perspectives

- Static vs. dynamic
  - Static can be fuzzy (the devil is in the details)
  - Dynamic can simply overwhelm you with details
- Coarse vs. fine-grained
  - architectural element vs. object interactions

Do not dive in the details

- At first do not care about details
  - Do not try to understand everything (you can get lost)
- Look for
  - component interactions
  - big players
  - large responsibilities

Coarse grained

- Architecture?
  - who interacts with what
- Static code
  - dependencies

About package dependencies

- If you have access to application "maps"
- Check package dependencies

e.g....

```
baseline: spec
  <baseline>
  spec
    for: #common
    do: [
      spec
        package: 'Fenster';
        package: 'Fenster−Tests' with: [ spec requires: #(Fenster) ].
      "Core"
      spec
        package: #Bloc with: [
          spec requires: #(#Fenster) ];
        package: #'BlocHost−Mock' with: [
          spec requires: #(#Bloc) ];
        package: #'Bloc−Tests' with: [
          spec requires: #(#Bloc 'BlocHost−Mock' ). ].
```

About package dependencies

- Importing a package does not mean that it is used :)
- Architecture is generally more interesting
  - publish subscribers
  - layers
  - registration

Important classes

- Root of hierarchies
- Most referenced classes
- Root of small hierarchies
- Tested classes

3 navigation (static) pillars

- References to classes
- Senders of messages
- Implementors of messages

Class references

Pay attention you can have factories

- a class not referenced much (basically only by the factory)
- whose instances are created by the factory

Senders

- Who is calling this method?
- Pay attention, quantity may not matter
  - you can have several callers
  - or a single but at the root of a class hierarchy

Implementors

- Who is providing this method?
- Are the implementors in the same hierarchy?
- Spread on multiple hierarchies?
  - Are they part of ’interfaces’?

Important messages

- Redefined messages?
  - Hook for *extenders*?
- Called a lot?

Hierarchy roots

- Check the roots of hierarchies
- Check references

Check tests

- Good for understanding use and scenario
  - black box testing is about use and external behavioral
  - Pay attention tests can test internals (white box)
- Getting fast an idea about the tested classes
- Watch out sometimes people are lazy and test simple classes, not complex ones

Look for senders of instance creation

- Class side methods
- Instance creation can provide the collaborators of classes

Design Assessment

- Conditionals
- Long methods (high cyclomatic is always a smell)
- Duplicated code
- Testing messages

Testing messages

- isMove, isPush...
- They often point to the absence of polymorphism and weak design
- Check their senders

Let us check the class API

Classes define:

- isEmptyBlock
- isWall
- hasPlayer
- hasTarget
- hasBox

Let us check the way this API is used

Too many ifs....

```
GameView >> drawBlock: aBlock on: aCanvas
  aBlock isWall
    ifTrue: [ self drawWall: aCanvas ]
    ifFalse: [ aBlock isEmptyBlock
      ifTrue: [ aBlock hasPlayer
          ifTrue: [ aBlock hasTarget
              ifTrue: [ self drawTargetAndPlayer: aCanvas ]
              ifFalse: [ self drawPlayer: aCanvas ]]
          ifFalse: [ aBlock hasBox
            ifTrue: [ aBlock hasTarget
              ifTrue: [ self drawTargetAndBox: aCanvas ]
              ifFalse: [ self drawBox: aCanvas ]]
          ifFalse: [
            aBlock hasTarget
              ifTrue: [ self drawTarget: aCanvas ]
            ifFalse: [ self drawEmptyBlock: aCanvas ]]]
```

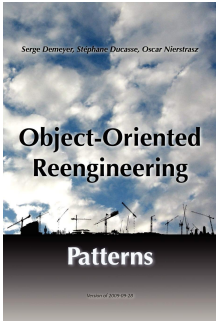
Selected Reverse engineering patterns

More fine-grained understanding

- Speculate about Design
- Refactor to understand
- Step through execution

Let us ""Speculate about Design""

- Apply **Speculate about Design** object-oriented reengineering pattern
- **Intent:** Progressively refine a design against source code by checking hypotheses about the design against the source code.
- Use your development expertise to conceive a hypothetical class diagram representing the design.



""Speculate about Design""

- Think about the kind of objects that should be in the application
- Look at the classes to see if/how they match your lists
- Refine to understand why you do not find the ones you guess
- Refine to understand the extra classes

""Refactor to Understand""

- You have tests! Green Tests! Then ...
- *Iteratively rename and refactor the code to introduce meaningful names and to make sure the structure of the code reflects what the system is actually doing. Run regression tests after each change if they are available, else compile often to check whether your changes make sense.*

Put a breakpoint and step

- If you have tests, or ways to execute,
- Place a breakpoint and check in the debugger
  - but pay attention you can drown in details

Books

- Object-Oriented Reengineering Patterns by Demeyer et al. (free)
  `https://scg.unibe.ch/download/oorp/OORP.pdf`
- Refactoring for Software Design Smells by Suryanarayana et al.
- Refactorings: Improving the Design of Existing Code by Fowler et al.

Conclusion

- Reverse engineering is a cool skill
- Practice as much as you can
- Expert developers know how to walk in the jungle

A course by

S. Ducasse, G. Polito, and Pablo Tesone