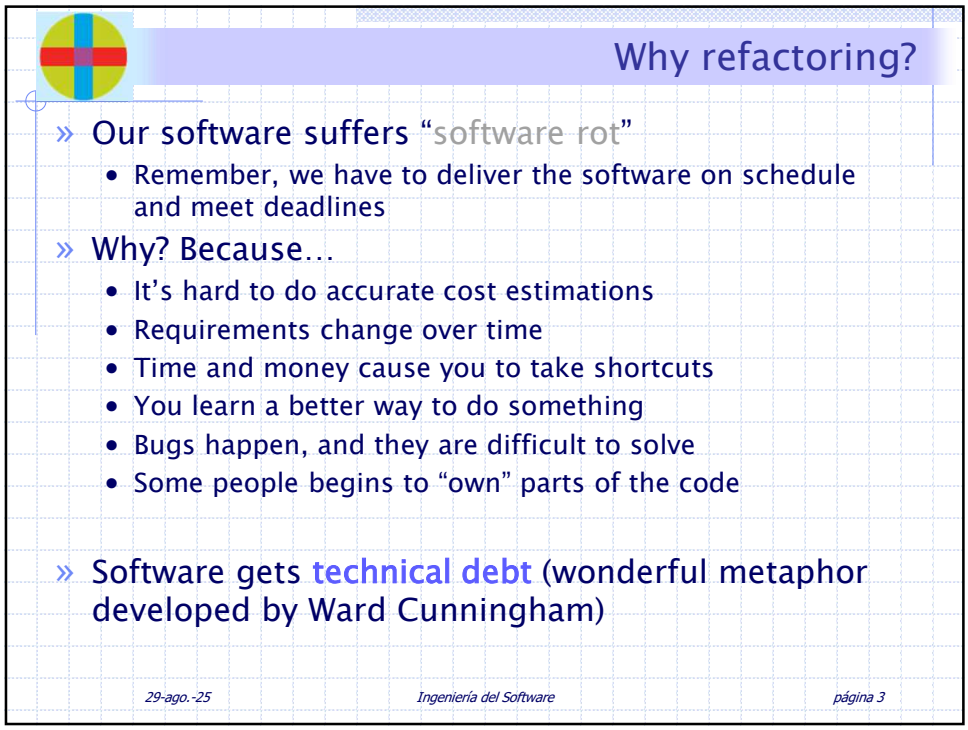
1

What is refactoring?

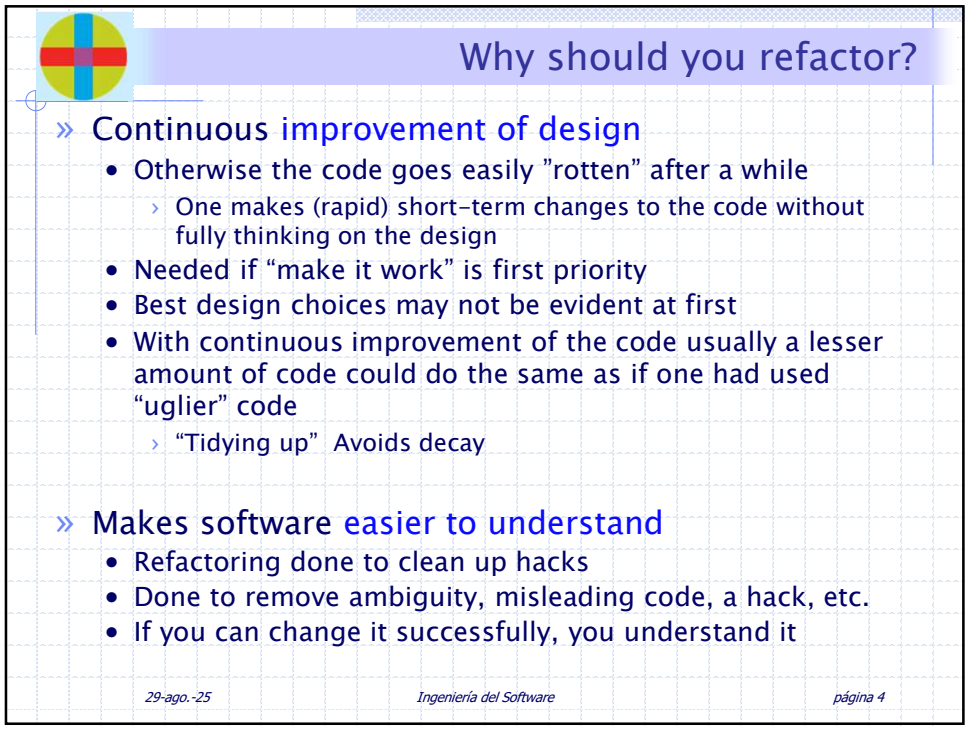
» Martin Fowler (and Kent Beck, John Brant, William Opdyke, Don Roberts): *Refactoring – Improving the Design of Existing Code*. Addison Wesley 1999/2018

- **Refactoring** (noun):
  - a change made to the internal structure of software to make it
    - easier to understand and
    - cheaper to modify
  - without changing its observable behavior.
- **Refactor** (verb):
  - to restructure software by applying a series of refactorings without changing its observable behavior
- So **refactoring** is…
  - Cleaning up code. Improving design. But after the code works
    - Fights against software entropy
- The goal of refactoring is **not** to add new functionality, but make code easier to maintain in the future

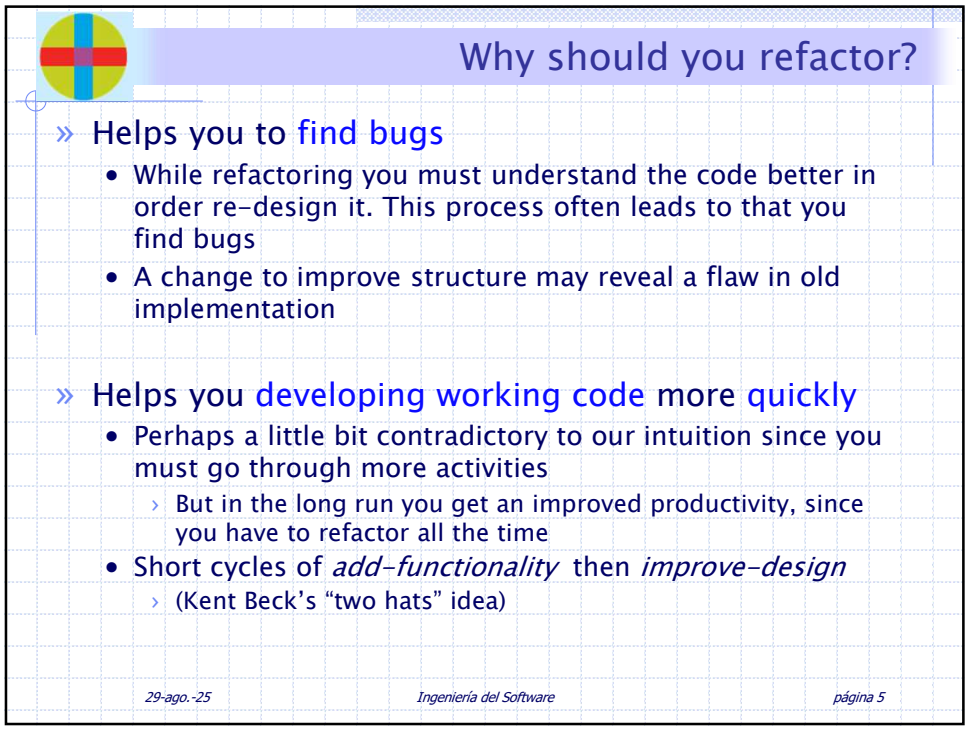
2


Why refactoring?
» Our software suffers "software rot"
• Remember, we have to deliver the software on schedule and meet deadlines
» Why? Because…
• It's hard to do accurate cost estimations
• Requirements change over time
• Time and money cause you to take shortcuts
• You learn a better way to do something
• Bugs happen, and they are difficult to solve
• Some people begins to "own" parts of the code
» Software gets technical debt (wonderful metaphor developed by Ward Cunningham)
29-ago.-25 Ingeniería del Software página 3

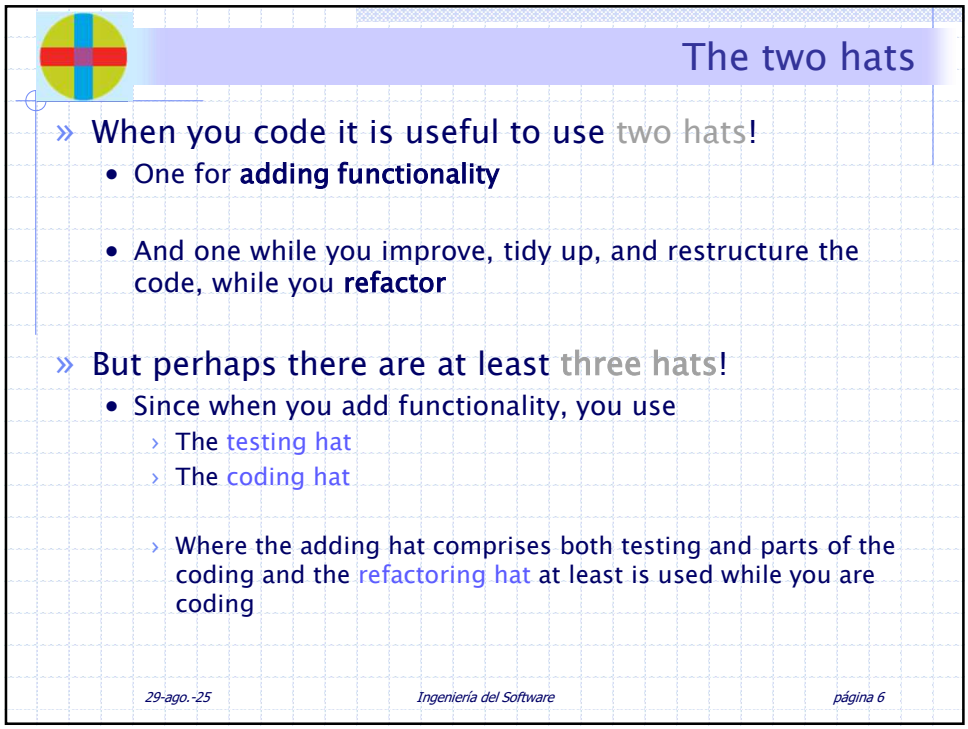
3


Why should you refactor?
» Continuous improvement of design
• Otherwise the code goes easily "rotten" after a while
› One makes (rapid) short-term changes to the code without fully thinking on the design
• Needed if "make it work" is first priority
• Best design choices may not be evident at first
• With continuous improvement of the code usually a lesser amount of code could do the same as if one had used "uglier" code
› "Tidying up" Avoids decay
» Makes software easier to understand
• Refactoring done to clean up hacks
• Done to remove ambiguity, misleading code, a hack, etc.
• If you can change it successfully, you understand it
29-ago.-25 Ingeniería del Software página 4

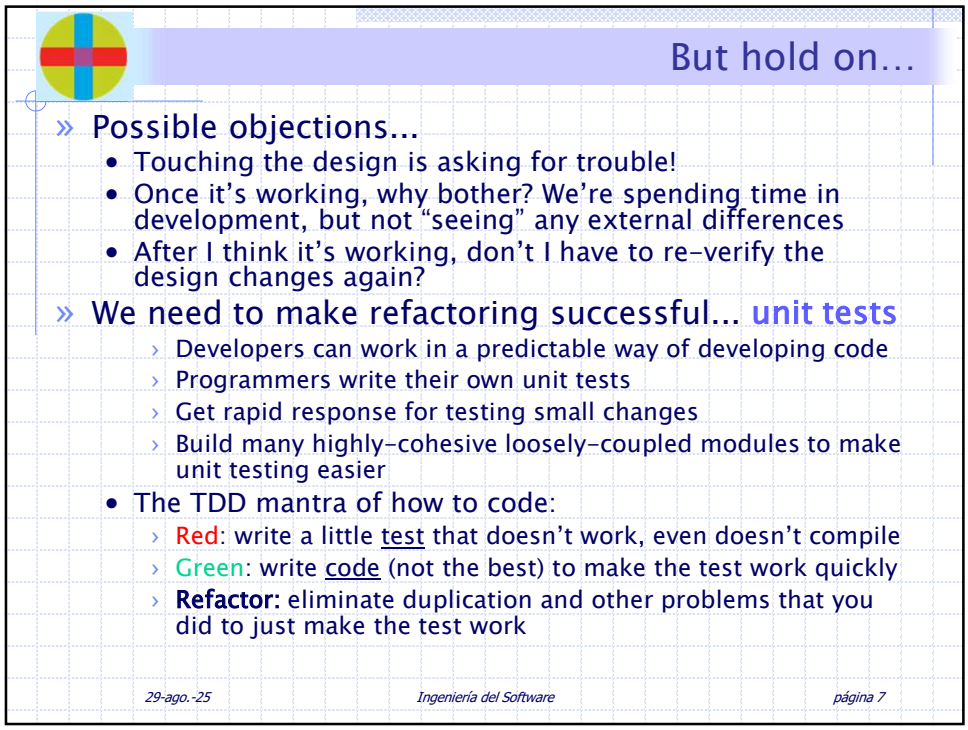
4


Why should you refactor?
» Helps you to find bugs
• While refactoring you must understand the code better in order re-design it. This process often leads to that you find bugs
• A change to improve structure may reveal a flaw in old implementation
» Helps you developing working code more quickly
• Perhaps a little bit contradictory to our intuition since you must go through more activities
› But in the long run you get an improved productivity, since you have to refactor all the time
• Short cycles of add-functionality then improve-design
› (Kent Beck's "two hats" idea)
29-ago.-25 Ingeniería del Software página 5

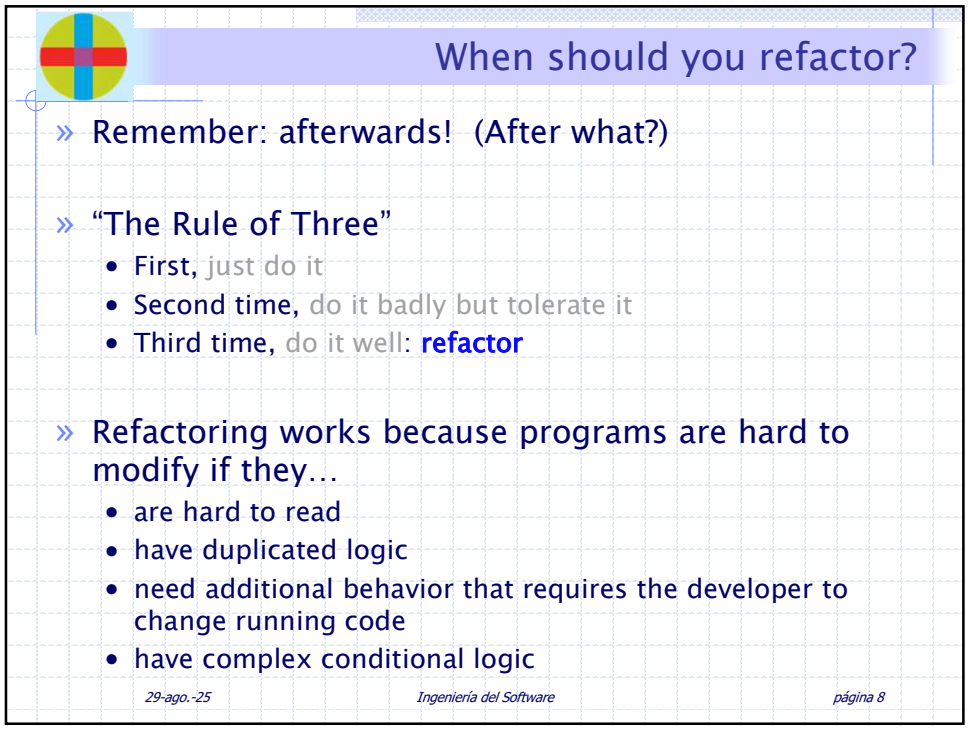
5


The two hats
» When you code it is useful to use two hats!
• One for adding functionality
• And one while you improve, tidy up, and restructure the code, while you refactor
» But perhaps there are at least three hats!
• Since when you add functionality, you use
› The testing hat
› The coding hat
› Where the adding hat comprises both testing and parts of the coding and the refactoring hat at least is used while you are coding
29-ago.-25 Ingeniería del Software página 6

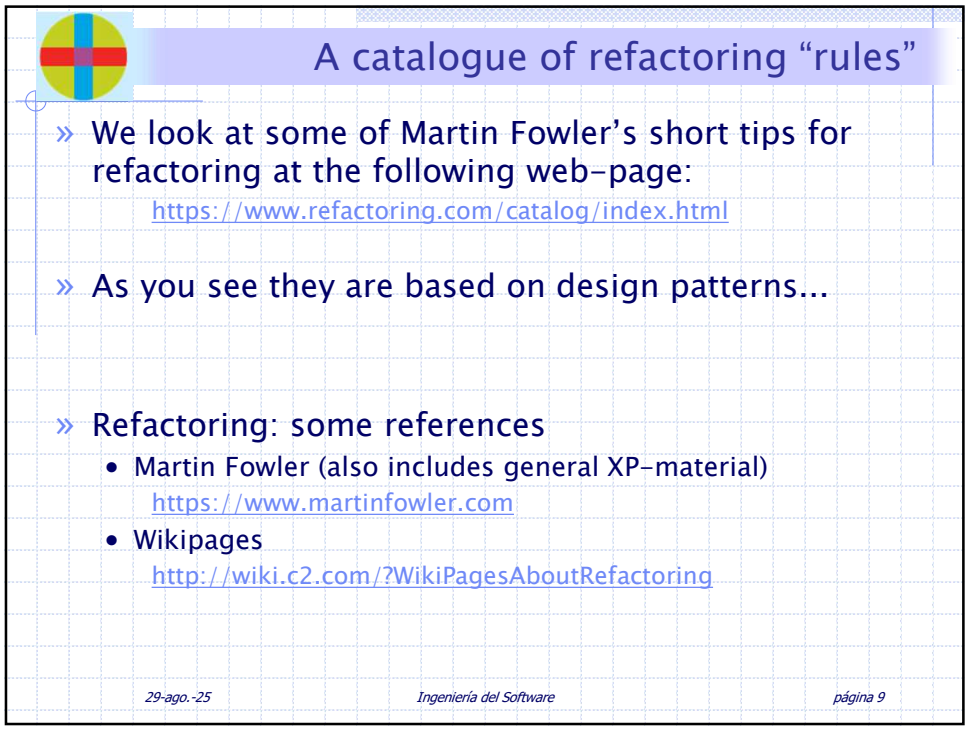
6


But hold on…
» Possible objections…
• Touching the design is asking for trouble!
• Once it's working, why bother? We're spending time in development, but not "seeing" any external differences
• After I think it's working, don't I have to re-verify the design changes again?
» We need to make refactoring successful… unit tests
› Developers can work in a predictable way of developing code
› Programmers write their own unit tests
› Get rapid response for testing small changes
› Build many highly-cohesive loosely-coupled modules to make unit testing easier
• The TDD mantra of how to code:
› Red: write a little test that doesn't work, even doesn't compile
› Green: write code (not the best) to make the test work quickly
› Refactor: eliminate duplication and other problems that you did to just make the test work
29-ago.-25 Ingeniería del Software página 7

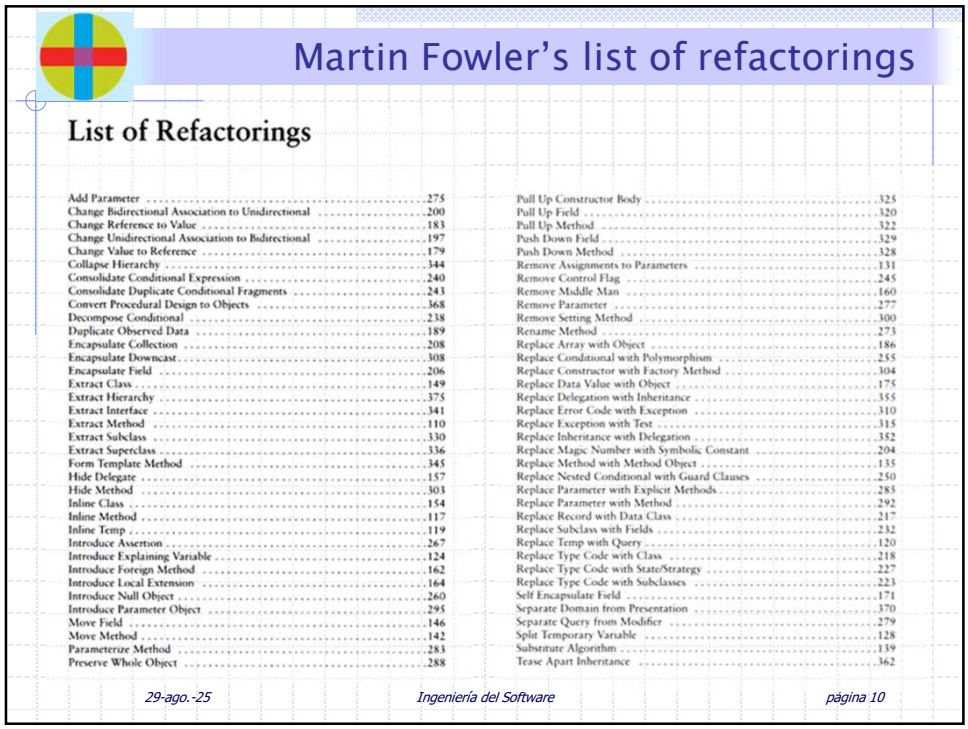
7


When should you refactor?
» Remember: afterwards! (After what?)
» "The Rule of Three"
• First, just do it
• Second time, do it badly but tolerate it
• Third time, do it well: refactor
» Refactoring works because programs are hard to modify if they…
• are hard to read
• have duplicated logic
• need additional behavior that requires the developer to change running code
• have complex conditional logic
29-ago.-25 Ingeniería del Software página 8

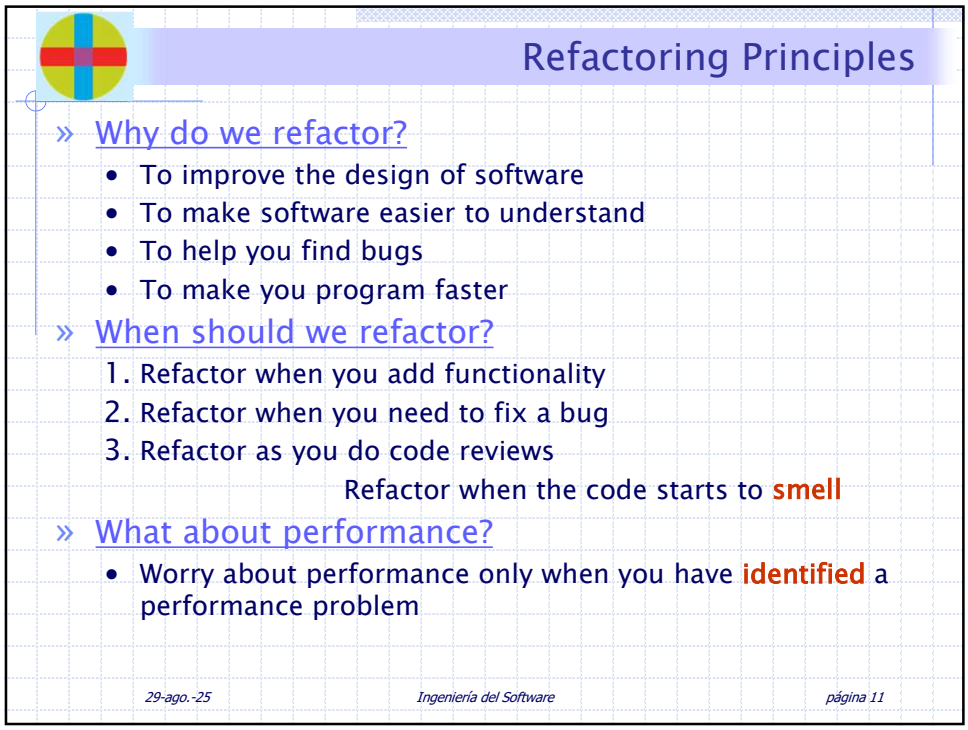
8


A catalogue of refactoring “rules”
» We look at some of Martin Fowler’s short tips for refactoring at the following web-page:
https://www.refactoring.com/catalog/index.html
» As you see they are based on design patterns...
» Refactoring: some references
• Martin Fowler (also includes general XP-material)
https://www.martinfowler.com
• Wikipages
http://wiki.c2.com/?WikiPagesAboutRefactoring
29-ago.-25 Ingeniería del Software página 9

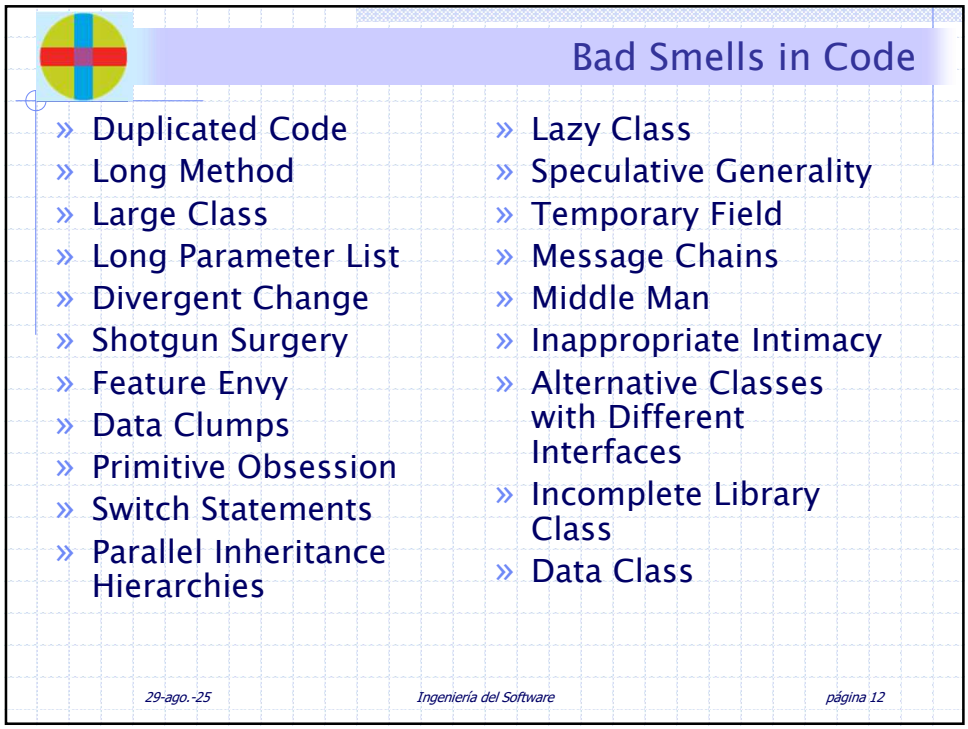
9


Martin Fowler’s list of refactorings
List of Refactorings
Add Parameter 275
Change Bidirectional Association to Unidirectional 200
Change Reference to Value 183
Change Unidirectional Association to Bidirectional 197
Change Value to Reference 179
Collapse Hierarchy 344
Consolidate Conditional Expression 240
Consolidate Duplicate Conditional Fragments 243
Convert Procedural Design to Objects 368
Decompose Conditional 238
Duplicate Observed Data 189
Encapsulate Collection 208
Encapsulate Downcast 308
Encapsulate Field 206
Extract Class 149
Extract Hierarchy 375
Extract Interface 341
Extract Method 110
Extract Subclass 330
Extract Superclass 336
Form Template Method 345
Hide Delegate 157
Hide Method 303
Inline Class 154
Inline Method 117
Inline Temp 119
Introduce Assertion 267
Introduce Explaining Variable 124
Introduce Foreign Method 162
Introduce Local Extension 164
Introduce Null Object 260
Introduce Parameter Object 295
Move Field 146
Move Method 142
Parameterize Method 283
Preserve Whole Object 288
Pull Up Constructor Body 325
Pull Up Field 320
Pull Up Method 322
Push Down Field 329
Push Down Method 328
Remove Assignments to Parameters 131
Remove Control Flag 245
Remove Middle Man 160
Remove Parameter 277
Remove Setting Method 300
Rename Method 273
Replace Array with Object 186
Replace Conditional with Polymorphism 255
Replace Constructor with Factory Method 304
Replace Data Value with Object 175
Replace Delegation with Inheritance 355
Replace Error Code with Exception 310
Replace Exception with Test 315
Replace Inheritance with Delegation 352
Replace Magic Number with Symbolic Constant 204
Replace Method with Method Object 135
Replace Nested Conditional with Guard Clauses 250
Replace Parameter with Explicit Methods 285
Replace Parameter with Method 292
Replace Record with Data Class 217
Replace Subclass with Fields 232
Replace Temp with Query 120
Replace Type Code with Class 218
Replace Type Code with State/Strategy 227
Replace Type Code with Subclasses 223
Self Encapsulate Field 171
Separate Domain from Presentation 370
Separate Query from Modifier 279
Split Temporary Variable 128
Substitute Algorithm 139
Tease Apart Inheritance 362
29-ago.-25 Ingeniería del Software página 10

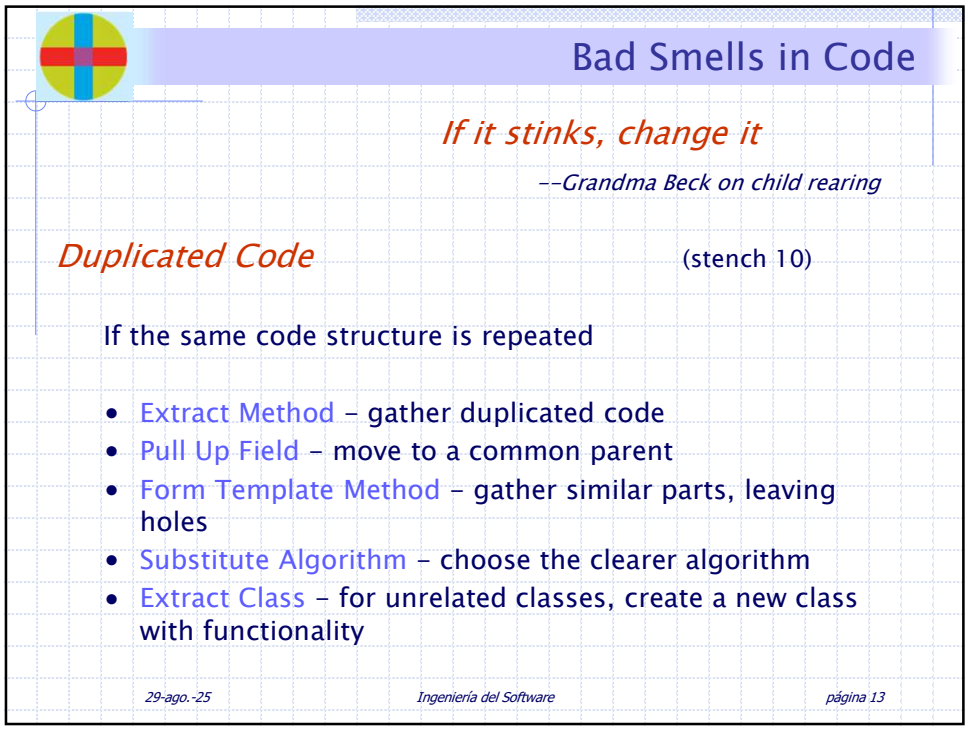
10


Refactoring Principles
» Why do we refactor?
• To improve the design of software
• To make software easier to understand
• To help you find bugs
• To make you program faster
» When should we refactor?
1. Refactor when you add functionality
2. Refactor when you need to fix a bug
3. Refactor as you do code reviews
Refactor when the code starts to smell
» What about performance?
• Worry about performance only when you have identified a performance problem
29-ago.-25 Ingeniería del Software página 11

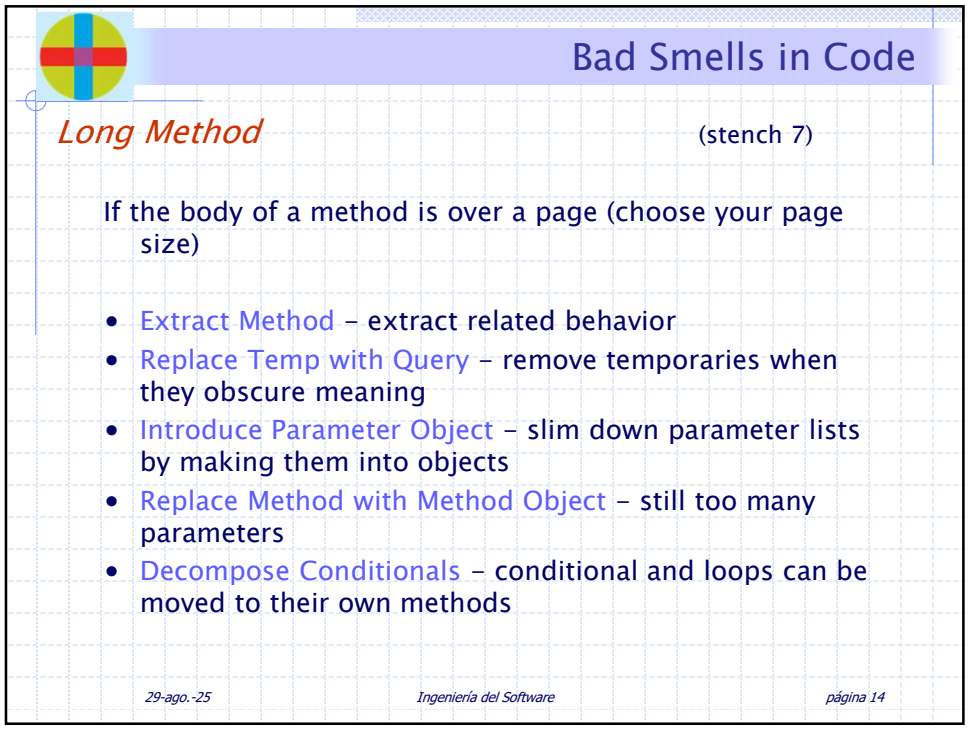
11


Bad Smells in Code
» Duplicated Code
» Long Method
» Large Class
» Long Parameter List
» Divergent Change
» Shotgun Surgery
» Feature Envy
» Data Clumps
» Primitive Obsession
» Switch Statements
» Parallel Inheritance Hierarchies
» Lazy Class
» Speculative Generality
» Temporary Field
» Message Chains
» Middle Man
» Inappropriate Intimacy
» Alternative Classes with Different Interfaces
» Incomplete Library Class
» Data Class
29-ago.-25 Ingeniería del Software página 12

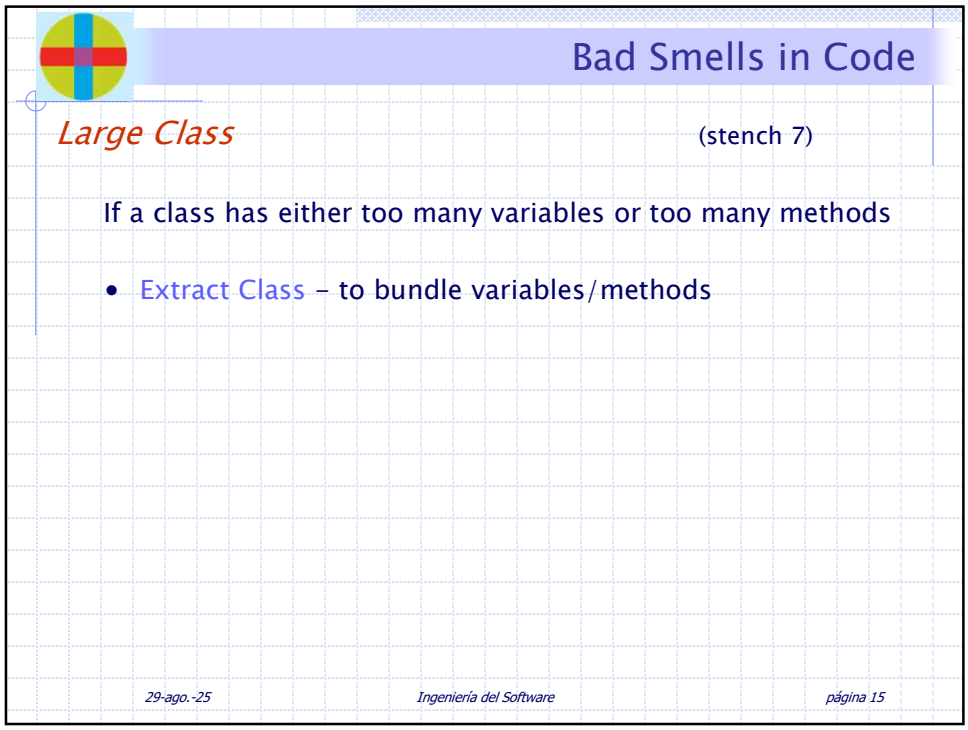
12


Bad Smells in Code
If it stinks, change it
--Grandma Beck on child rearing
Duplicated Code
(stench 10)
If the same code structure is repeated
Extract Method – gather duplicated code
Pull Up Field – move to a common parent
Form Template Method – gather similar parts, leaving holes
Substitute Algorithm – choose the clearer algorithm
Extract Class – for unrelated classes, create a new class with functionality
29-ago.-25
Ingeniería del Software
página 13

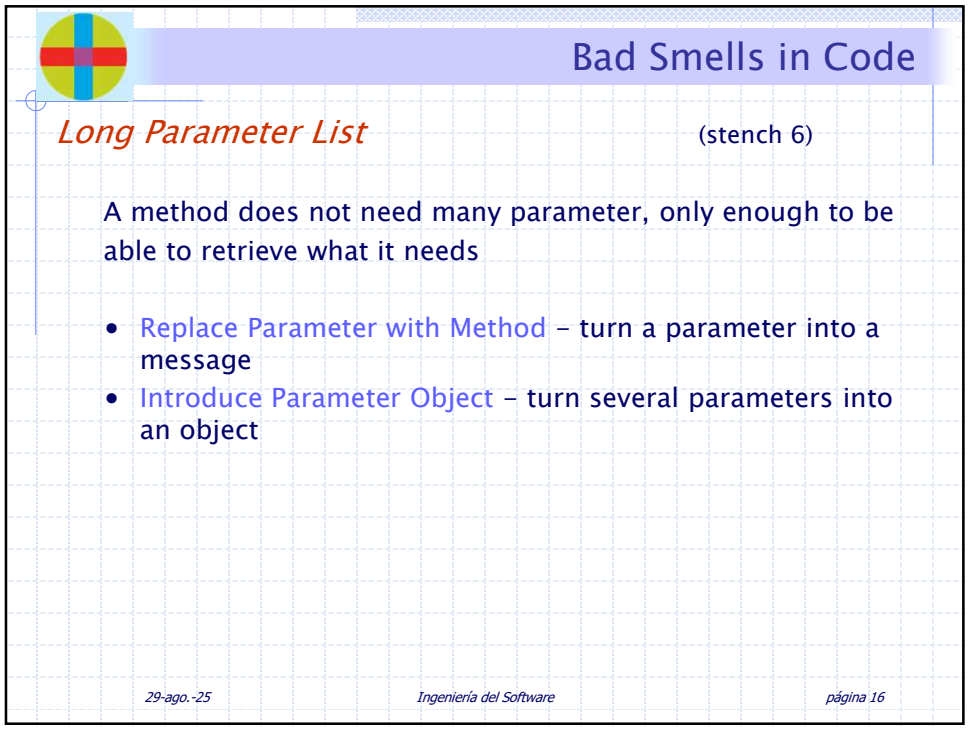
13


Bad Smells in Code
Long Method
(stench 7)
If the body of a method is over a page (choose your page size)
Extract Method – extract related behavior
Replace Temp with Query – remove temporaries when they obscure meaning
Introduce Parameter Object – slim down parameter lists by making them into objects
Replace Method with Method Object – still too many parameters
Decompose Conditionals – conditional and loops can be moved to their own methods
29-ago.-25
Ingeniería del Software
página 14

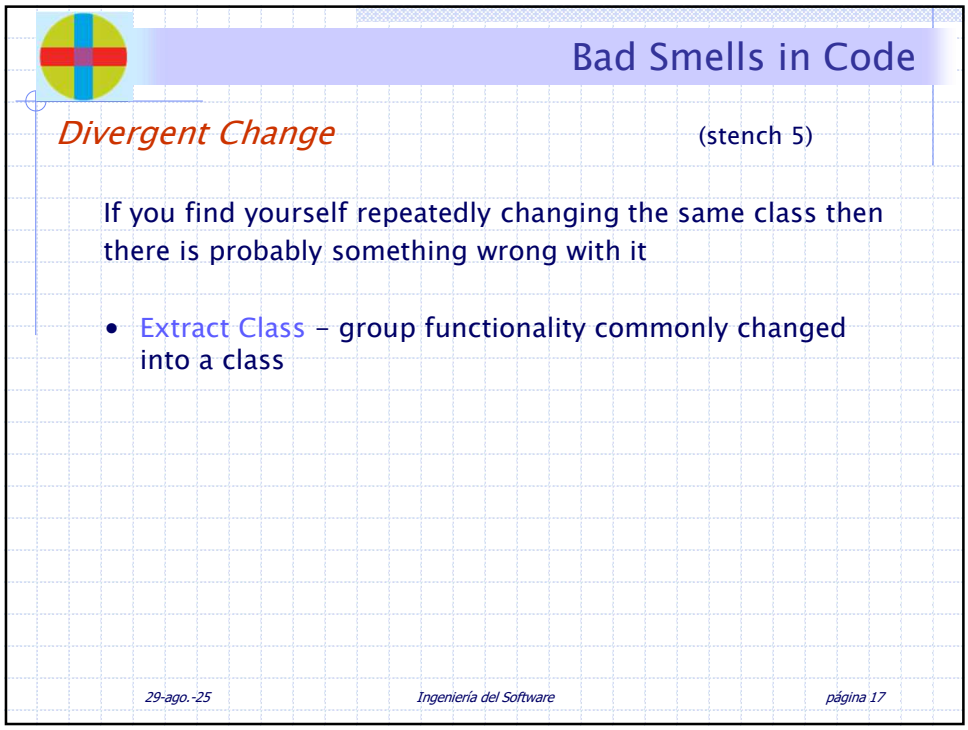
14


Bad Smells in Code
Large Class
(stench 7)
If a class has either too many variables or too many methods
• Extract Class – to bundle variables/methods
29-ago.-25
Ingeniería del Software
página 15

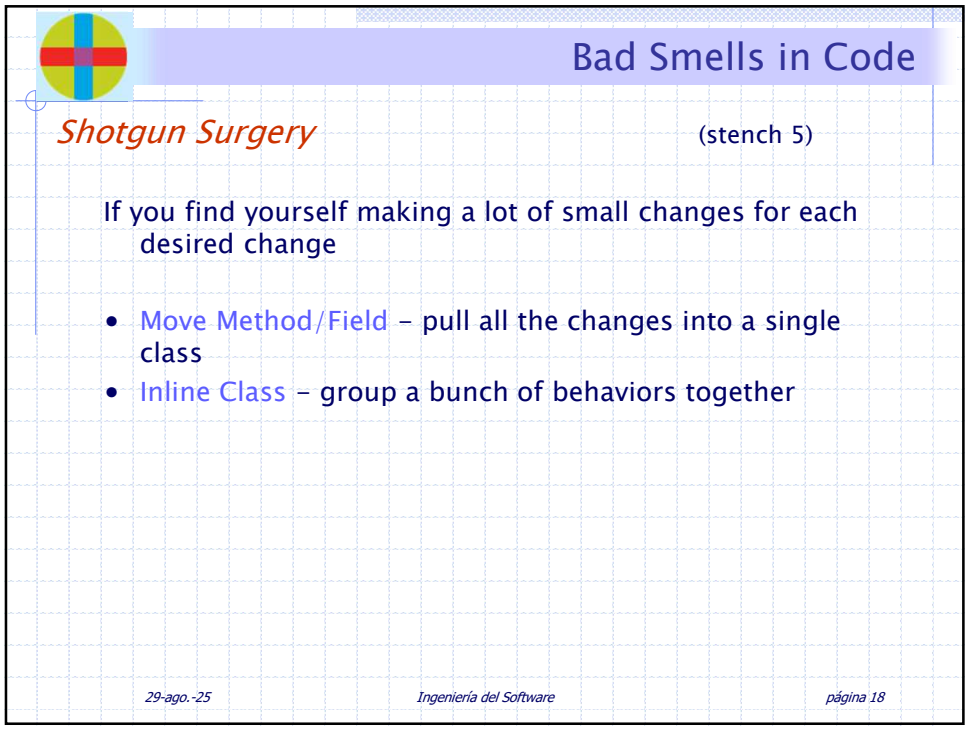
15


Bad Smells in Code
Long Parameter List
(stench 6)
A method does not need many parameter, only enough to be able to retrieve what it needs
• Replace Parameter with Method – turn a parameter into a message
• Introduce Parameter Object – turn several parameters into an object
29-ago.-25
Ingeniería del Software
página 16

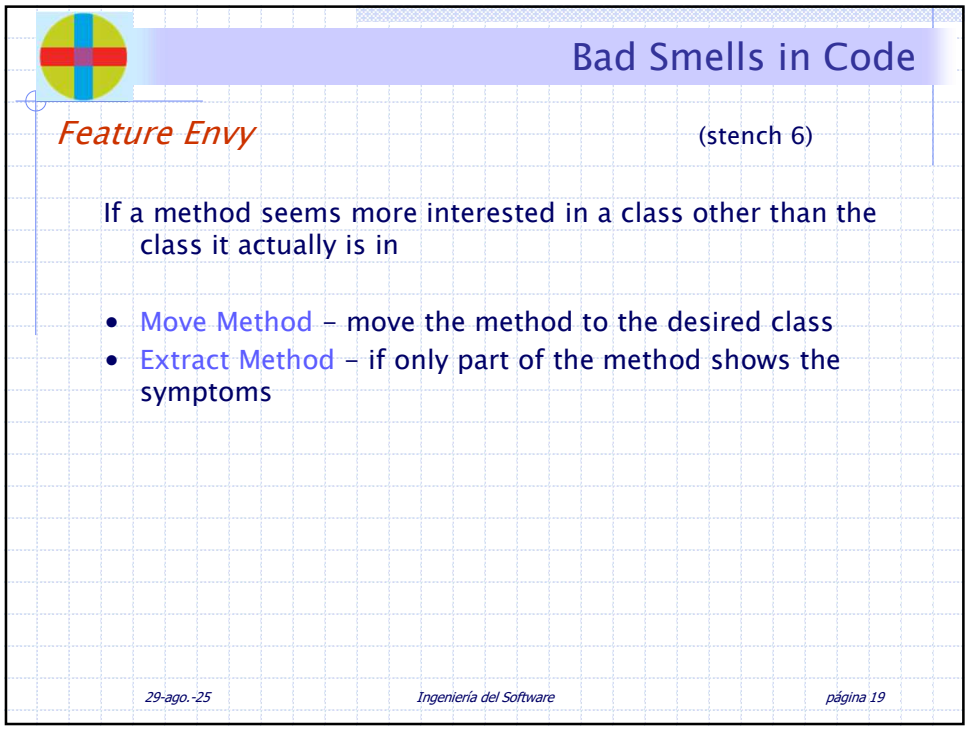
16


Bad Smells in Code
Divergent Change
(stench 5)
If you find yourself repeatedly changing the same class then there is probably something wrong with it
• Extract Class – group functionality commonly changed into a class
29-ago.-25
Ingeniería del Software
página 17

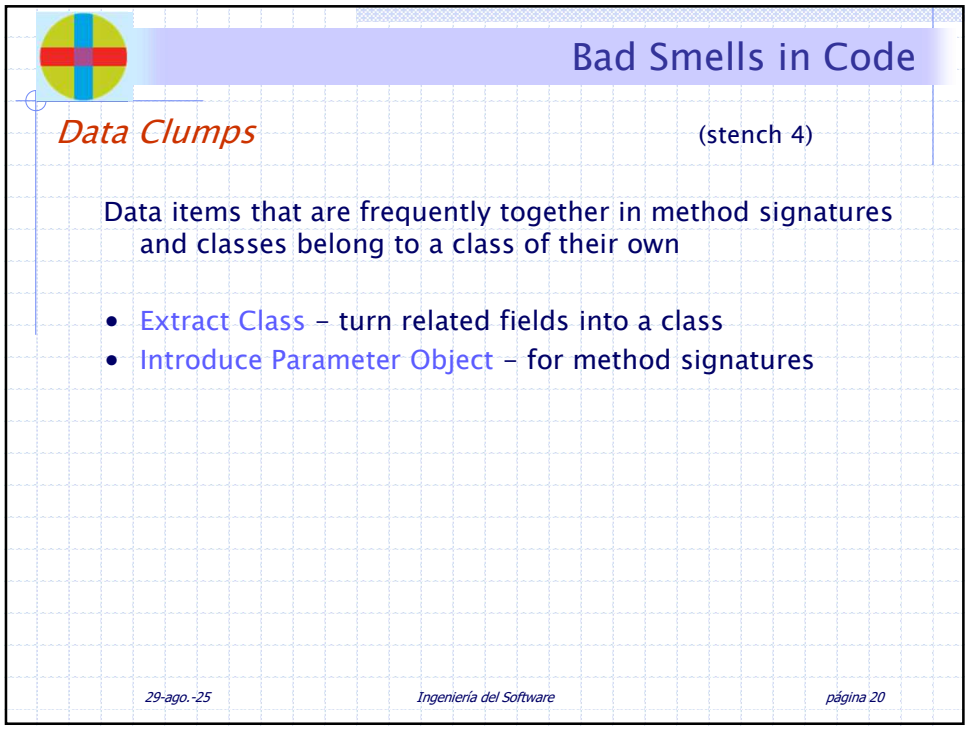
17


Bad Smells in Code
Shotgun Surgery
(stench 5)
If you find yourself making a lot of small changes for each desired change
• Move Method/Field – pull all the changes into a single class
• Inline Class – group a bunch of behaviors together
29-ago.-25
Ingeniería del Software
página 18

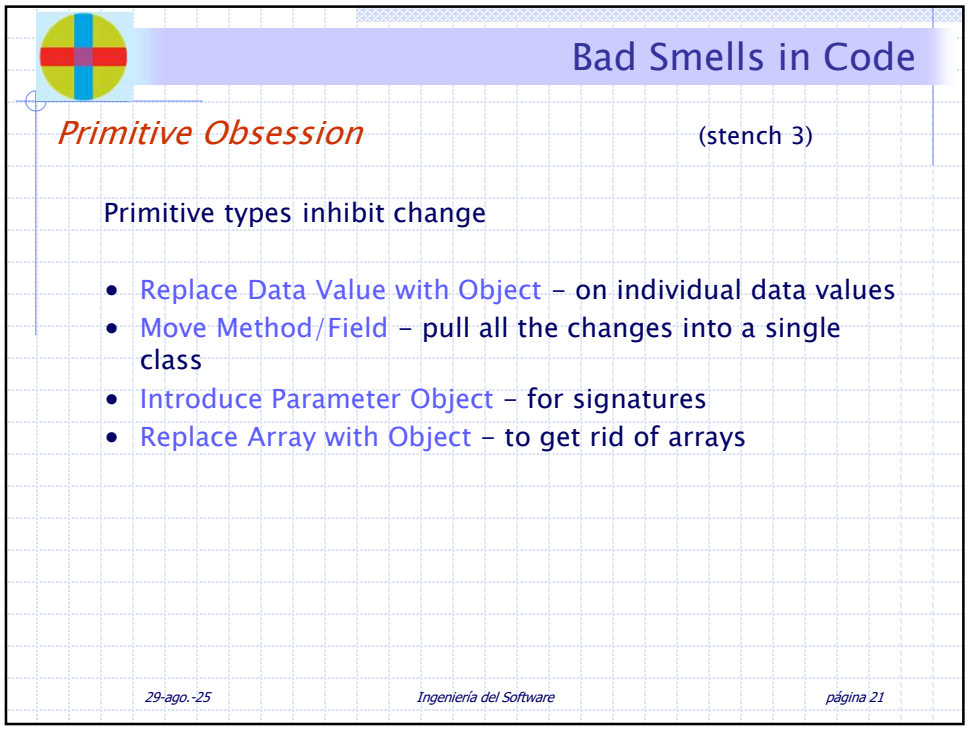
18


Bad Smells in Code
Feature Envy
(stench 6)
If a method seems more interested in a class other than the class it actually is in
Move Method – move the method to the desired class
Extract Method – if only part of the method shows the symptoms
29-ago.-25
Ingeniería del Software
página 19

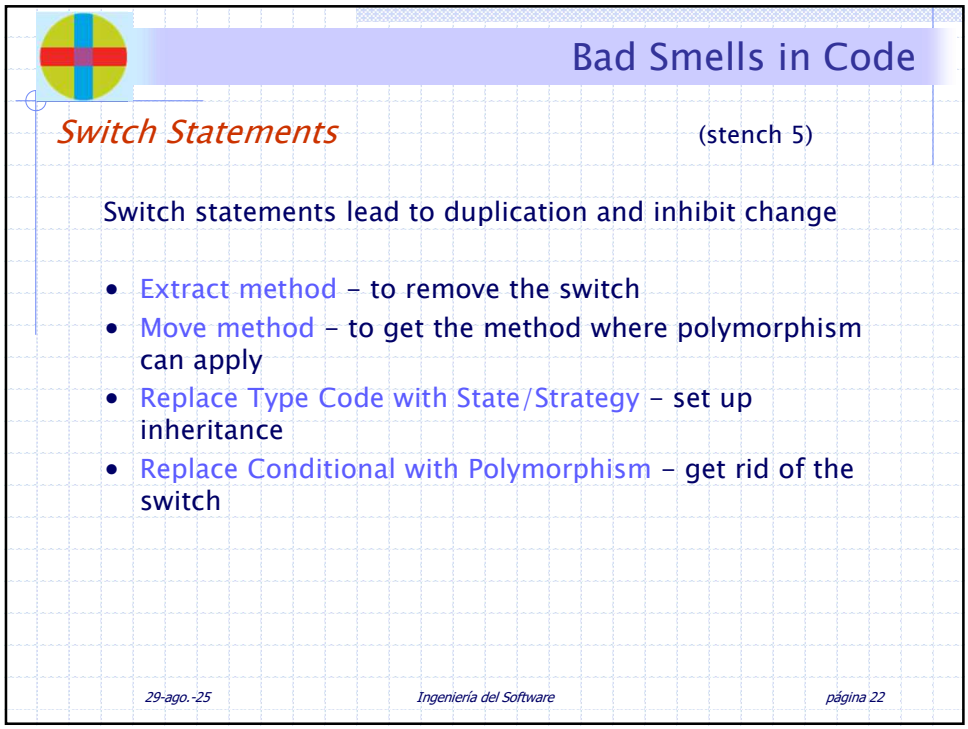
19


Bad Smells in Code
Data Clumps
(stench 4)
Data items that are frequently together in method signatures and classes belong to a class of their own
Extract Class – turn related fields into a class
Introduce Parameter Object – for method signatures
29-ago.-25
Ingeniería del Software
página 20

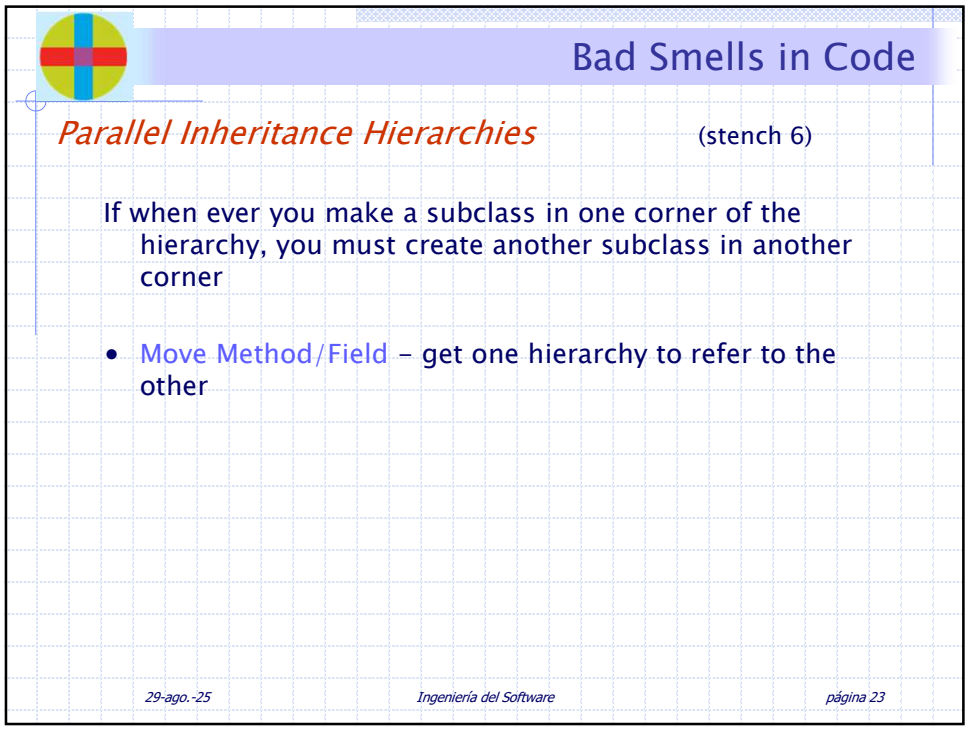
20


Bad Smells in Code
Primitive Obsession
(stench 3)
Primitive types inhibit change
Replace Data Value with Object – on individual data values
Move Method/Field – pull all the changes into a single class
Introduce Parameter Object – for signatures
Replace Array with Object – to get rid of arrays
29-ago.-25
Ingeniería del Software
página 21

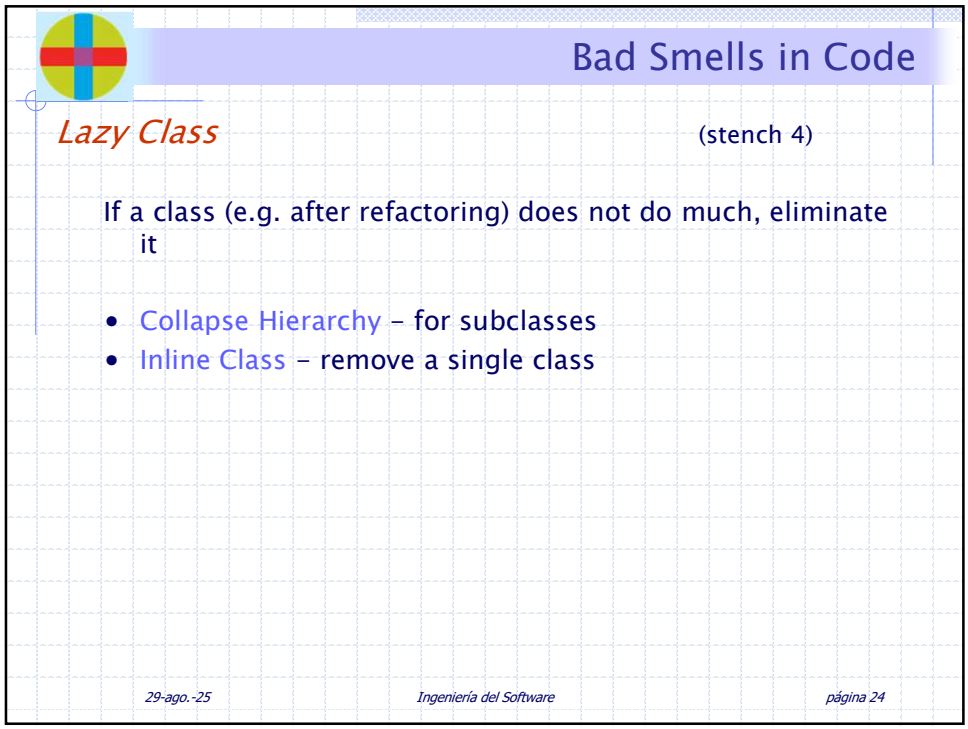
21


Bad Smells in Code
Switch Statements
(stench 5)
Switch statements lead to duplication and inhibit change
Extract method – to remove the switch
Move method – to get the method where polymorphism can apply
Replace Type Code with State/Strategy – set up inheritance
Replace Conditional with Polymorphism – get rid of the switch
29-ago.-25
Ingeniería del Software
página 22

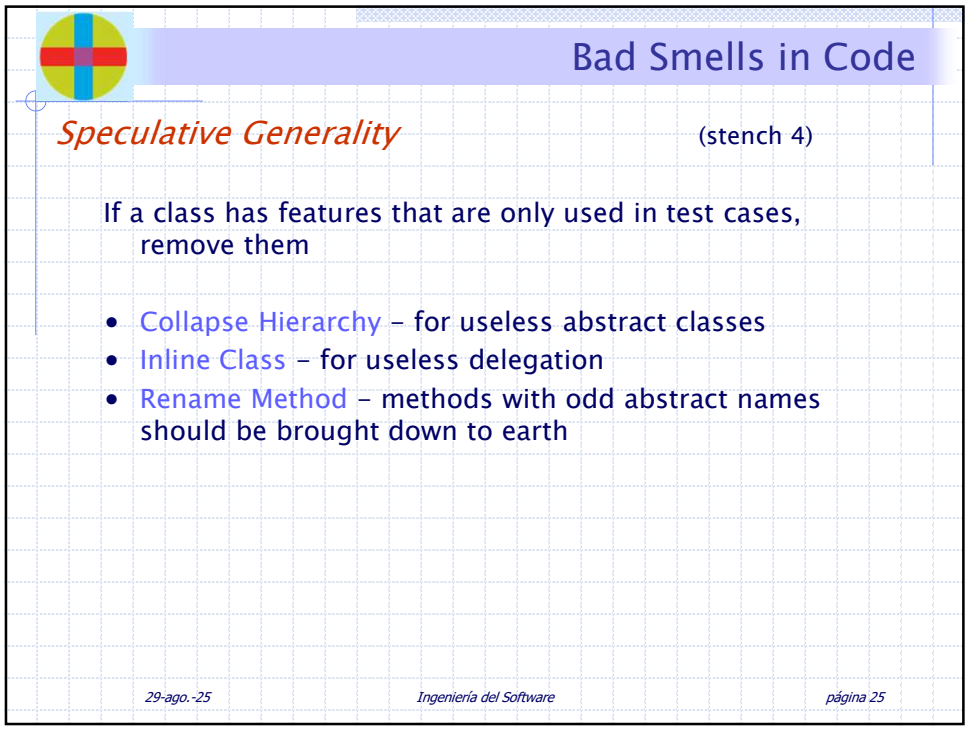
22


Bad Smells in Code
Parallel Inheritance Hierarchies
(stench 6)
If when ever you make a subclass in one corner of the hierarchy, you must create another subclass in another corner
• Move Method/Field – get one hierarchy to refer to the other
29-ago.-25
Ingeniería del Software
página 23

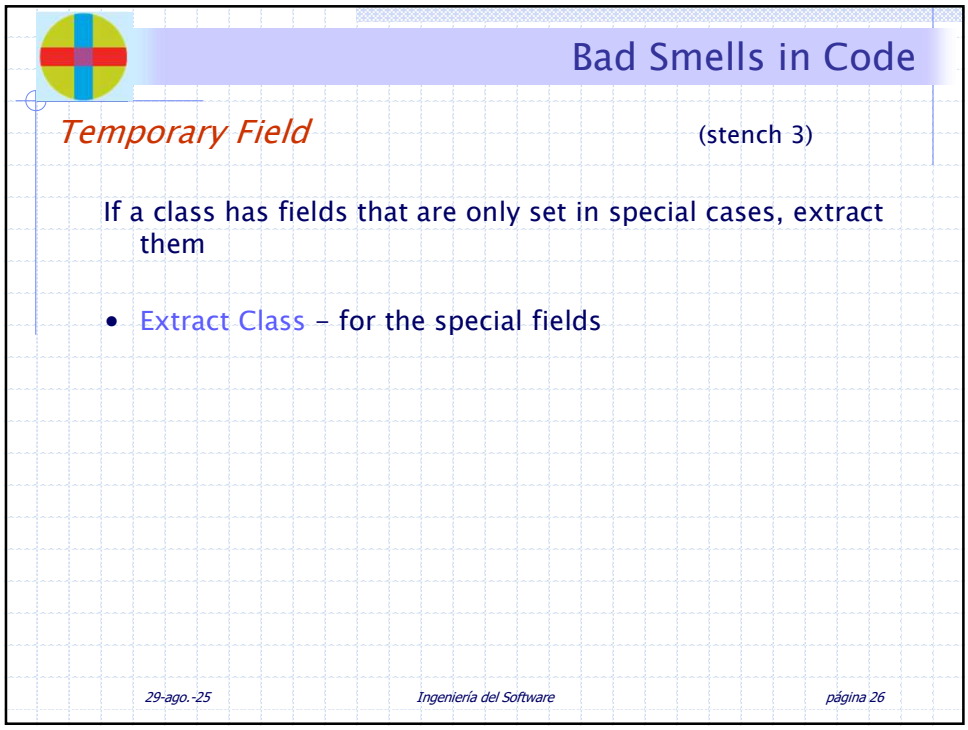
23


Bad Smells in Code
Lazy Class
(stench 4)
If a class (e.g. after refactoring) does not do much, eliminate it
• Collapse Hierarchy – for subclasses
• Inline Class – remove a single class
29-ago.-25
Ingeniería del Software
página 24

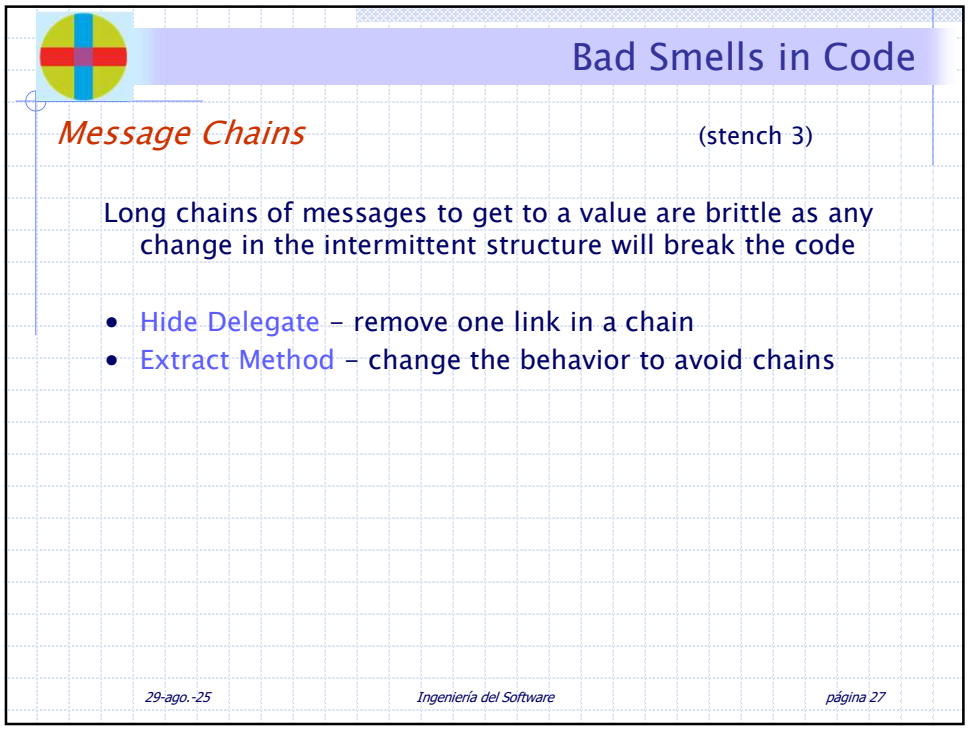
24


Bad Smells in Code
Speculative Generality
(stench 4)
If a class has features that are only used in test cases, remove them
Collapse Hierarchy – for useless abstract classes
Inline Class – for useless delegation
Rename Method – methods with odd abstract names should be brought down to earth
29-ago.-25
Ingeniería del Software
página 25

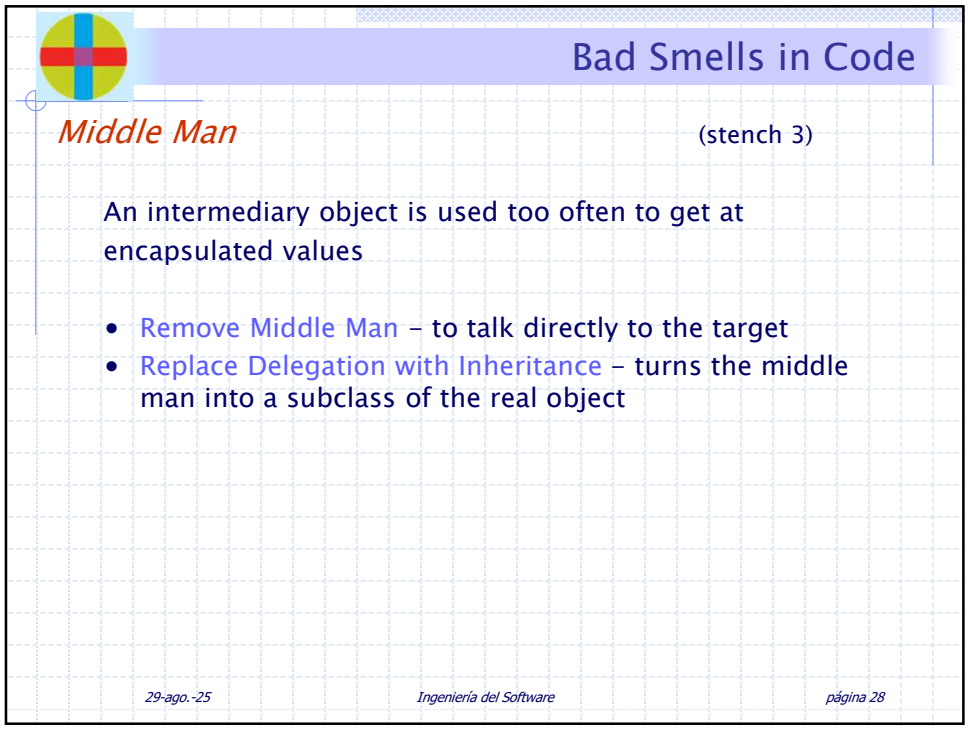
25


Bad Smells in Code
Temporary Field
(stench 3)
If a class has fields that are only set in special cases, extract them
Extract Class – for the special fields
29-ago.-25
Ingeniería del Software
página 26

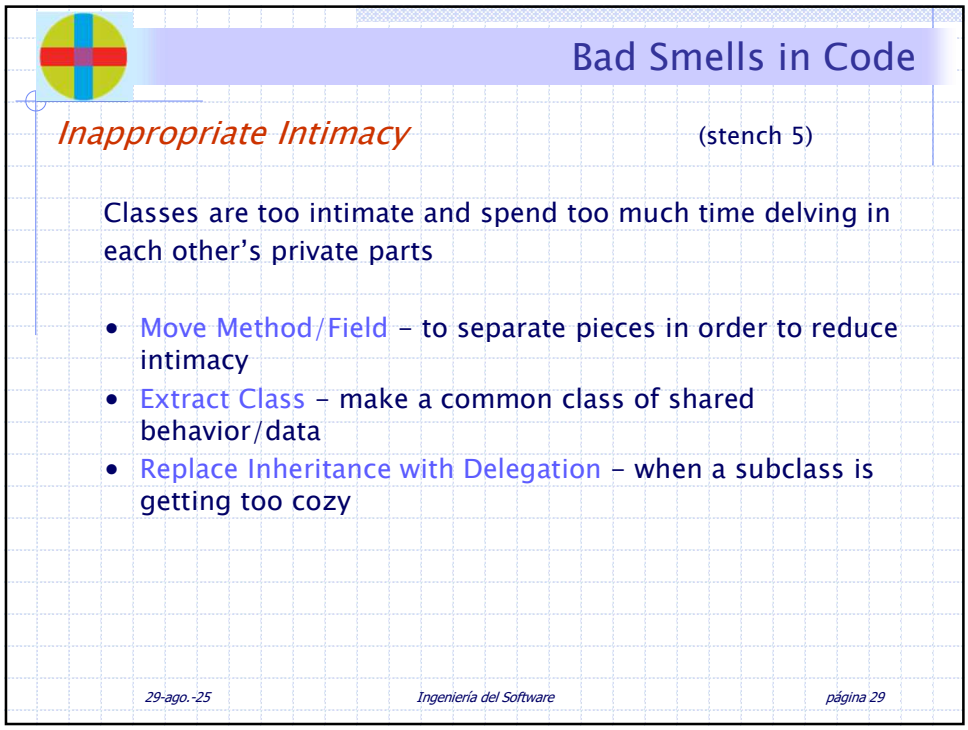
26


Bad Smells in Code
Message Chains
(stench 3)
Long chains of messages to get to a value are brittle as any change in the intermittent structure will break the code
• Hide Delegate – remove one link in a chain
• Extract Method – change the behavior to avoid chains
29-ago.-25
Ingeniería del Software
página 27

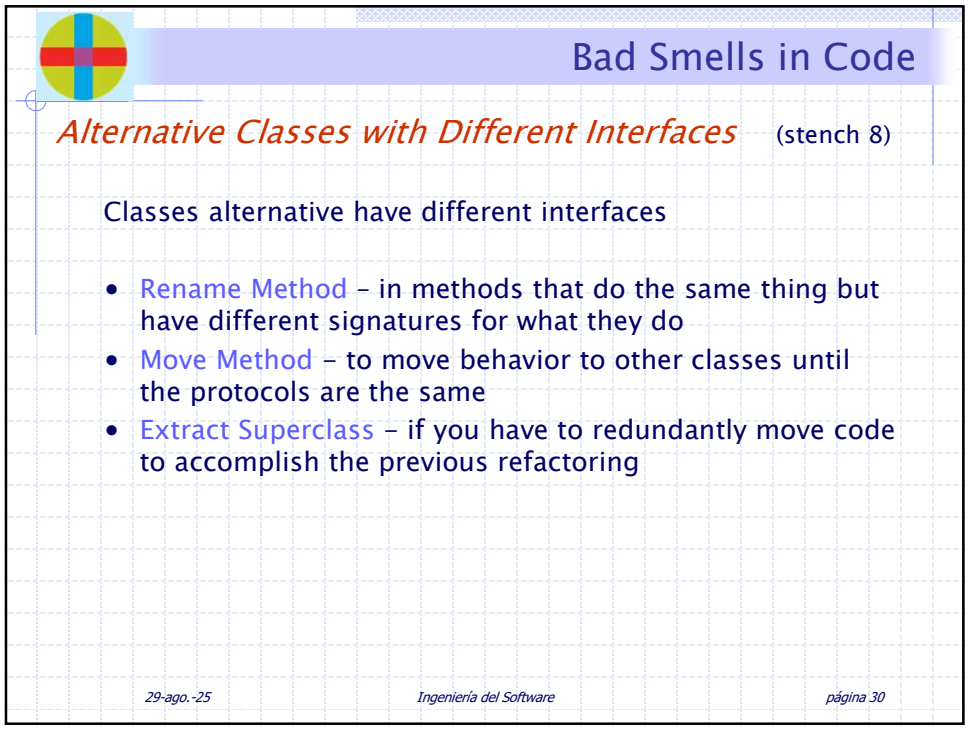
27


Bad Smells in Code
Middle Man
(stench 3)
An intermediary object is used too often to get at encapsulated values
• Remove Middle Man – to talk directly to the target
• Replace Delegation with Inheritance – turns the middle man into a subclass of the real object
29-ago.-25
Ingeniería del Software
página 28

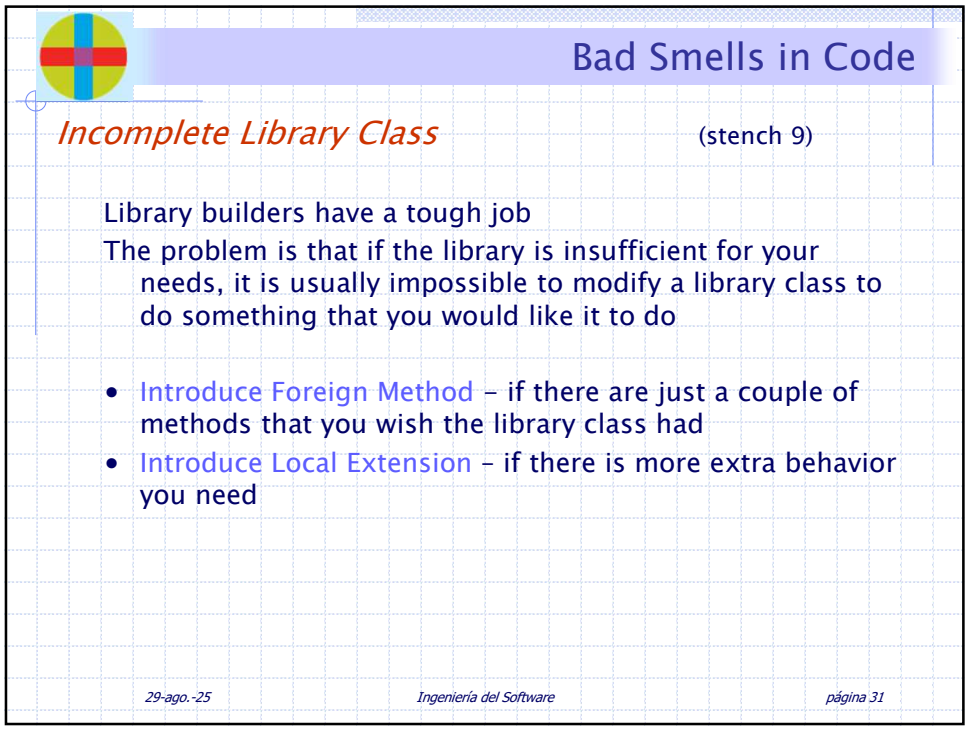
28


Bad Smells in Code
Inappropriate Intimacy
(stench 5)
Classes are too intimate and spend too much time delving in each other's private parts
• Move Method/Field – to separate pieces in order to reduce intimacy
• Extract Class – make a common class of shared behavior/data
• Replace Inheritance with Delegation – when a subclass is getting too cozy
29-ago.-25
Ingeniería del Software
página 29

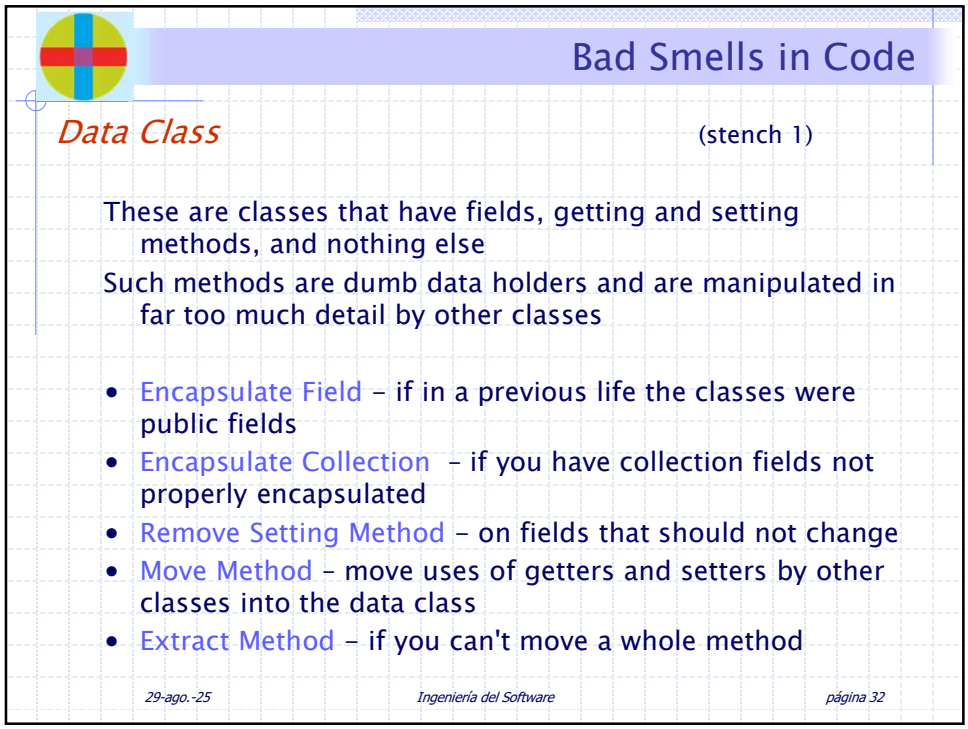
29


Bad Smells in Code
Alternative Classes with Different Interfaces
(stench 8)
Classes alternative have different interfaces
• Rename Method – in methods that do the same thing but have different signatures for what they do
• Move Method – to move behavior to other classes until the protocols are the same
• Extract Superclass – if you have to redundantly move code to accomplish the previous refactoring
29-ago.-25
Ingeniería del Software
página 30

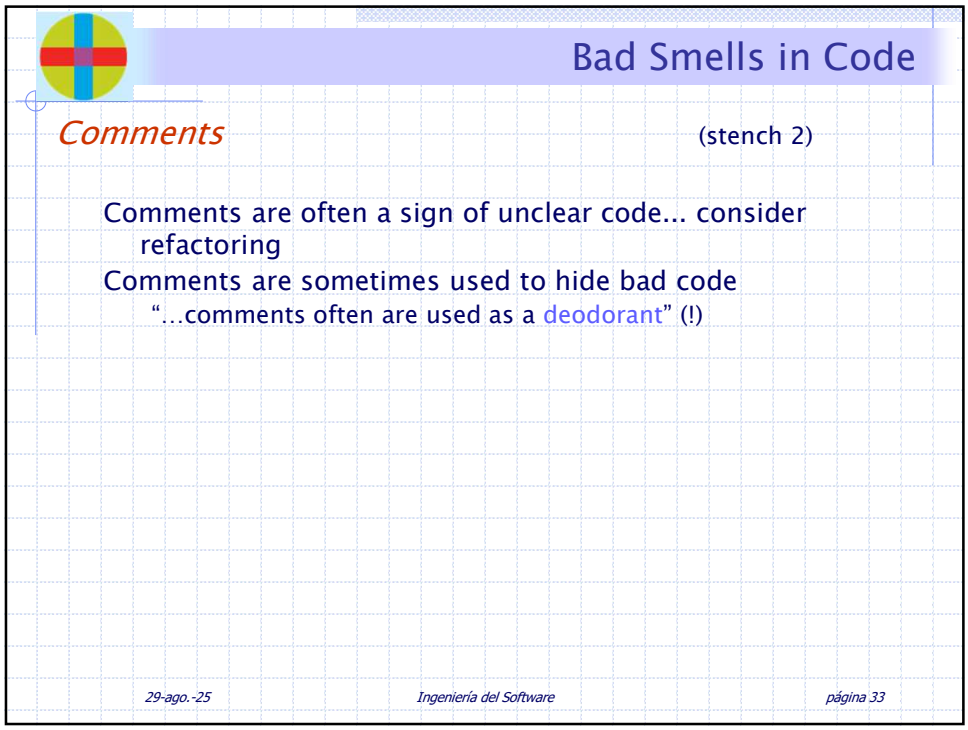
30


Bad Smells in Code
Incomplete Library Class
(stench 9)
Library builders have a tough job
The problem is that if the library is insufficient for your needs, it is usually impossible to modify a library class to do something that you would like it to do
• Introduce Foreign Method – if there are just a couple of methods that you wish the library class had
• Introduce Local Extension – if there is more extra behavior you need
29-ago.-25 Ingeniería del Software página 31

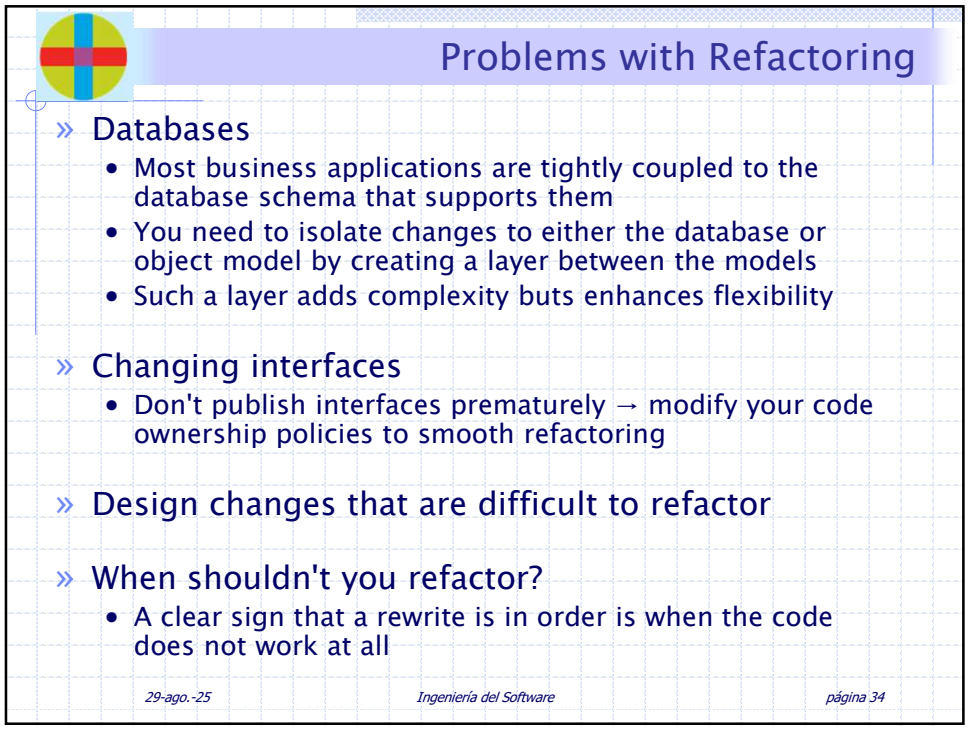
31


Bad Smells in Code
Data Class
(stench 1)
These are classes that have fields, getting and setting methods, and nothing else
Such methods are dumb data holders and are manipulated in far too much detail by other classes
• Encapsulate Field – if in a previous life the classes were public fields
• Encapsulate Collection – if you have collection fields not properly encapsulated
• Remove Setting Method – on fields that should not change
• Move Method – move uses of getters and setters by other classes into the data class
• Extract Method – if you can't move a whole method
29-ago.-25 Ingeniería del Software página 32

32


Bad Smells in Code
Comments
(stench 2)
Comments are often a sign of unclear code... consider refactoring
Comments are sometimes used to hide bad code
"...comments often are used as a deodorant" (!)
29-ago.-25
Ingeniería del Software
página 33

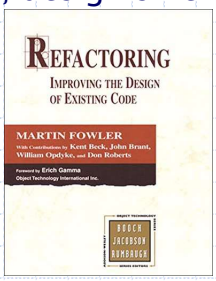
33

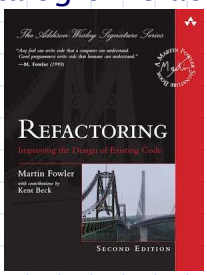

Problems with Refactoring
» Databases
• Most business applications are tightly coupled to the database schema that supports them
• You need to isolate changes to either the database or object model by creating a layer between the models
• Such a layer adds complexity buts enhances flexibility
» Changing interfaces
• Don't publish interfaces prematurely → modify your code ownership policies to smooth refactoring
» Design changes that are difficult to refactor
» When shouldn't you refactor?
• A clear sign that a rewrite is in order is when the code does not work at all
29-ago.-25
Ingeniería del Software
página 34

34

Further reading

- Fowler, design smells, and catalog of refactorings:





- The book (first and second edition):
  https://www.refactoring.com
- List of smells on the web:
  https://sourcemaking.com/refactoring/smells
  https://www.industriallogic.com/blog/smells-to-refactorings-cheatsheet

¿Preguntas?