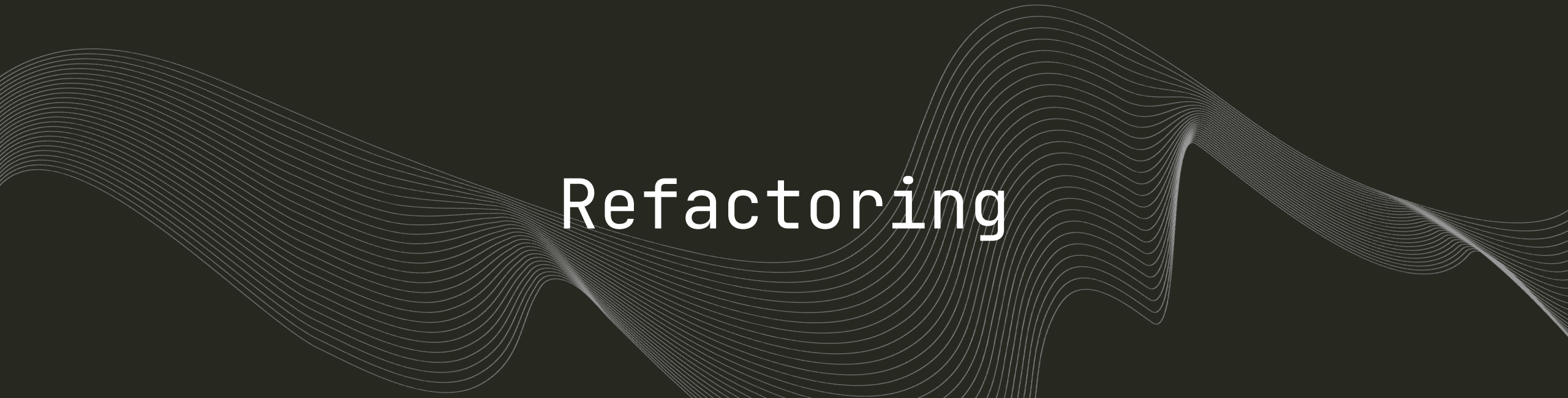


Refactoring

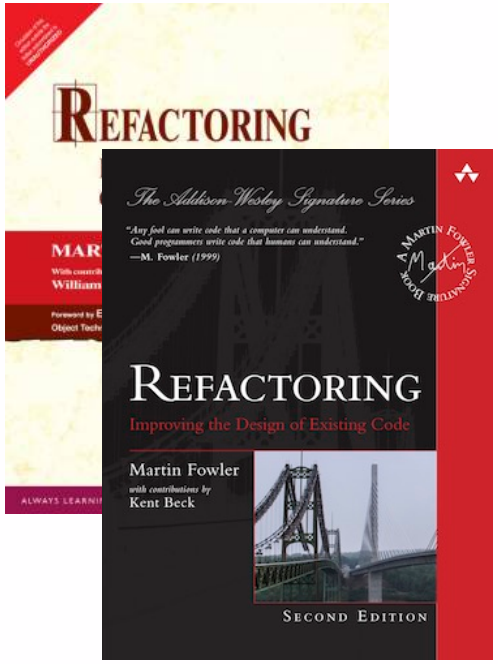
Software Evolution — L4T2

Dr. Vadim Zaytsev aka @grammarware, February/March 2021

UNIVERSITY OF TWENTE. | Formal Methods & Tools

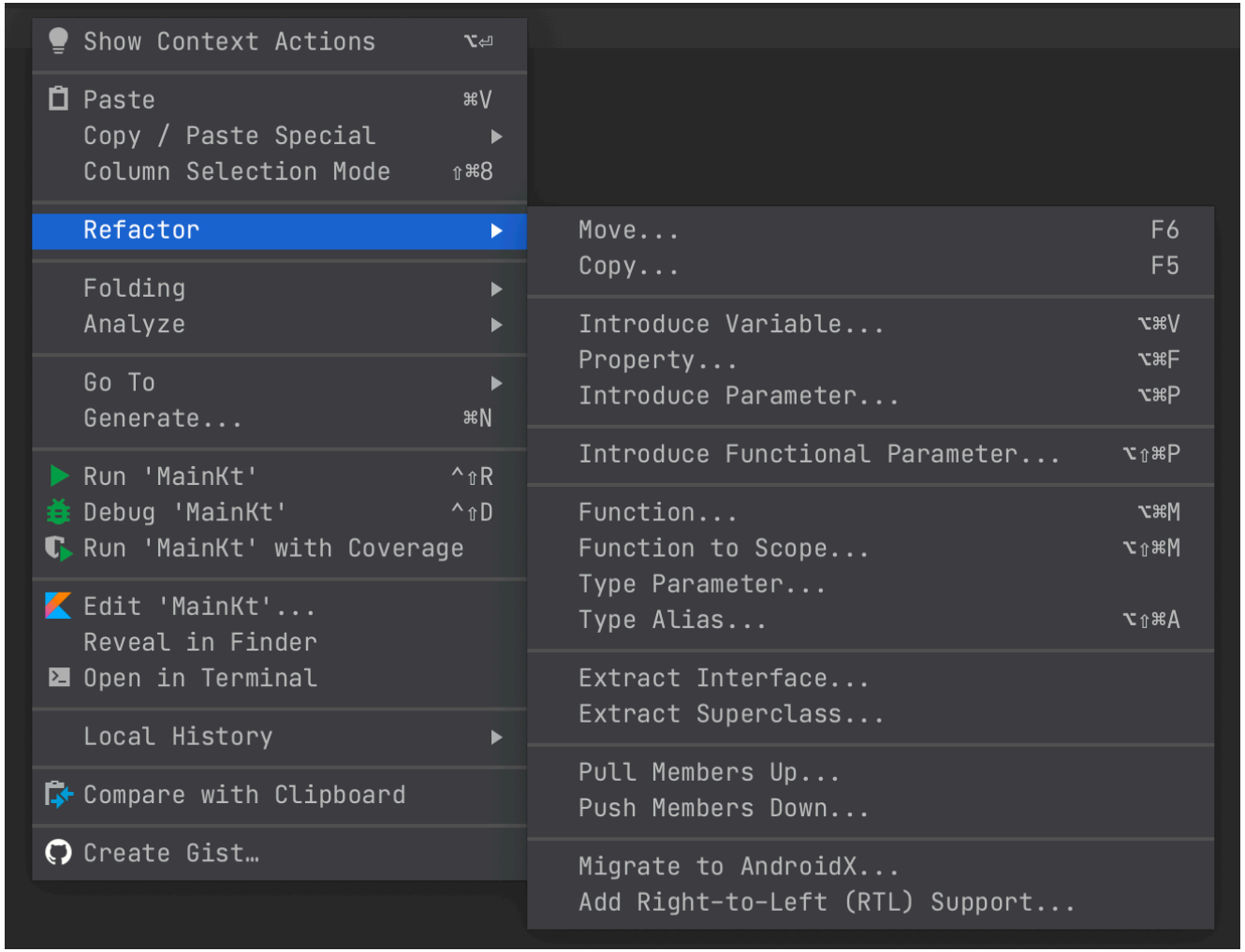
Refactoring is…

changing
the internals



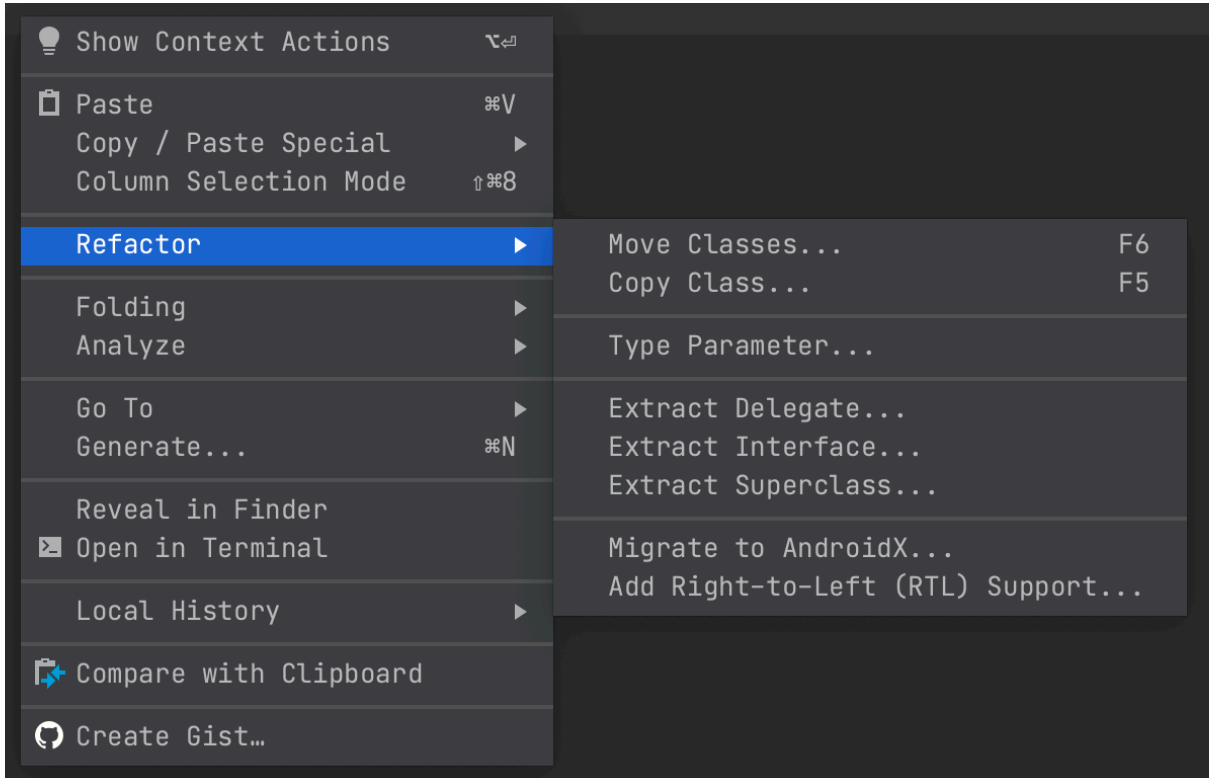
keeping
the externals

Refactoring in IDEs

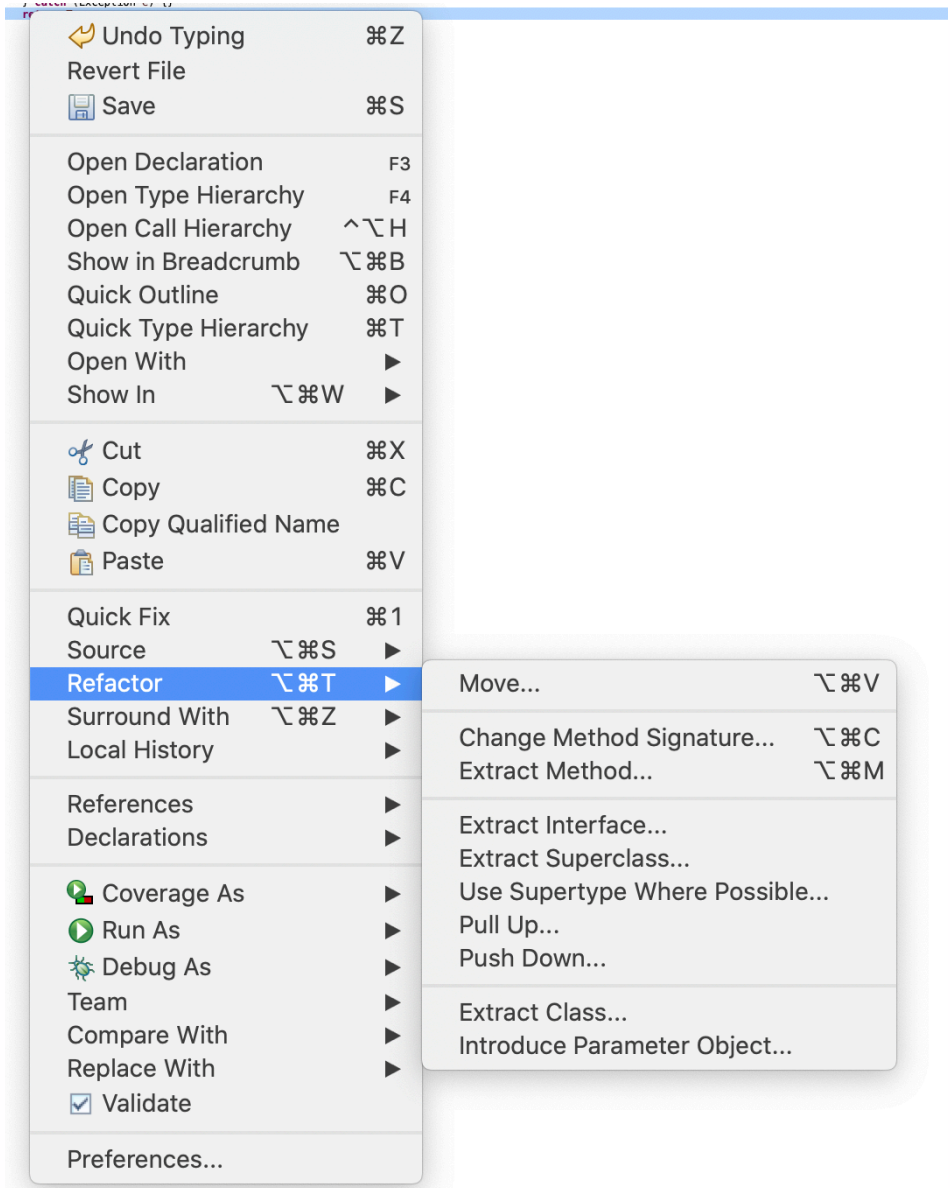


Kotlin in IntelliJ IDEA

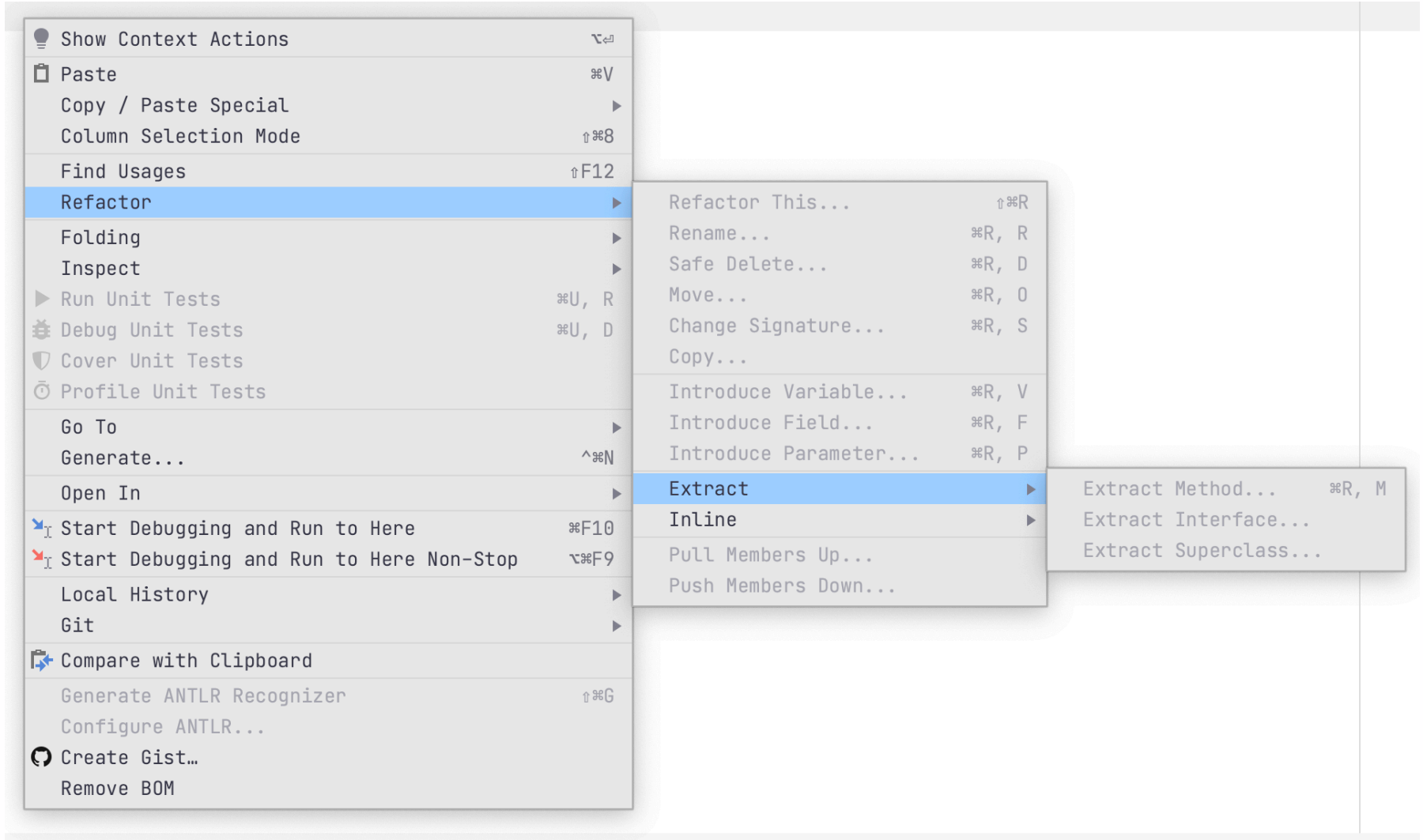
Refactoring in IDEs



Refactoring in IDEs



Refactoring in IDEs

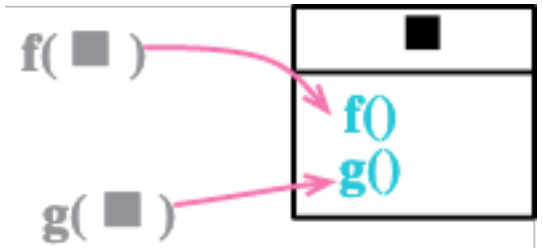


C# in Rider

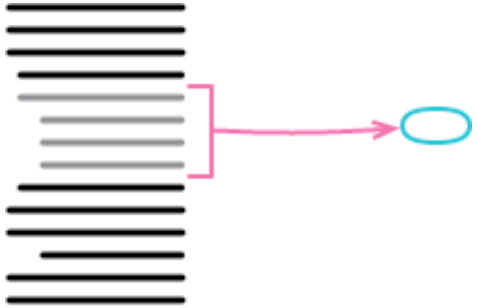
Refactoring in a Narrow Sense

Remove Dead Code

Combine Functions into Class

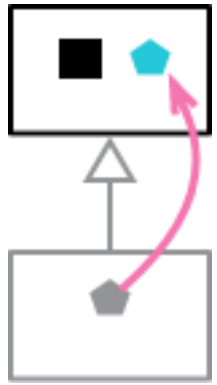


Extract Function



Replace Loop with Pipeline

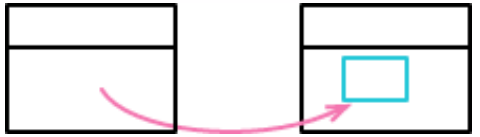
Collapse Hierarchy



. . .

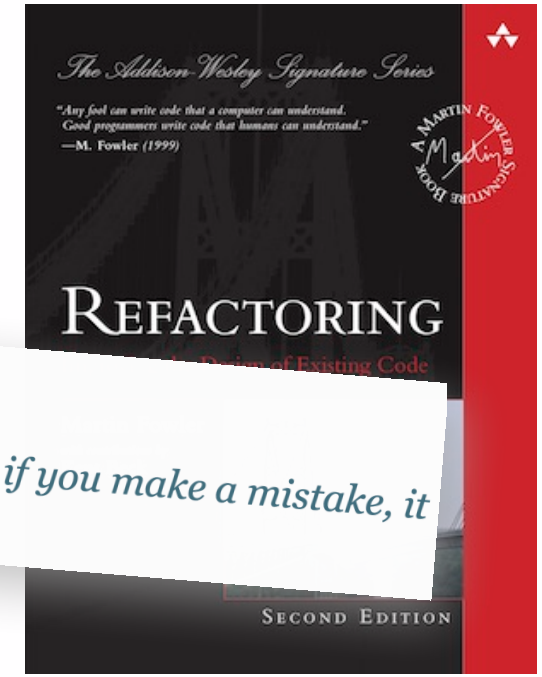
Inline Variable

Move Field



https://refactoring.com/catalog/

Refactoring in a Narrow Sense



Before you start refactoring, make sure you have a solid suite of tests.

Refactoring changes the programs in small steps, so if you make a mistake, it is easy to find where the bug is.

When you have to add a feature to a program but the code is not structured in a convenient way, first refactor the program to make it easy to add the feature, then add the feature.

If someone says their code was broken for a couple of days while they are refactoring, you can be pretty sure they were not refactoring.

Refactoring in a Broad Sense

- Change the internals
  - usually for better quality
- Keep the externals
  - for interoperability
- Is not limited to code!
  - grammars
  - data
  - . . .

Conclusion

- Refactorings: in IDE, narrow sense, broad sense
- Some refactorings lack any sense
- “Refactoring” without tests is just changing stuff
- Q&A Sessions @ Canvas
  - ⇒ v.zaytsev@utwente.nl
  - ⇒ https://discord.gg/n7VQAPNBPD