Extracted from:

Software Design X-Rays

Fix Technical Debt with Behavioral Code Analysis

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Software Design X-Rays

Fix Technical Debt with Behavioral Code Analysis

Adam Tornhill

edited by Adaobi Obi Tulton

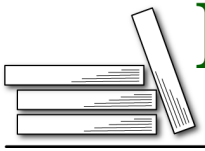
Software Design X-Rays

Fix Technical Debt with Behavioral Code Analysis

Adam Tornhill

The Pragmatic Bookshelf

Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Managing Editor: Brian MacDonald
Supervising Editor: Jacquelyn Carter
Development Editor: Adaobi Obi Tulton
Copy Editor: Candace Cunningham
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

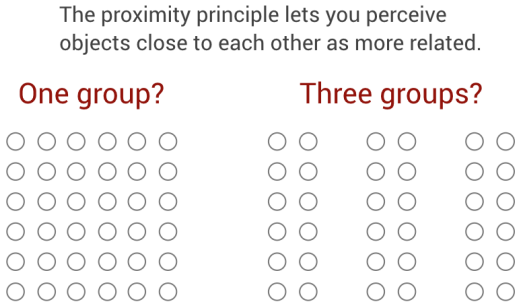
For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-272-5
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—March 2018

Use Your Perception

A century ago the movement of *Gestalt psychology* formed theories on how we make sense of all chaotic input from our sensory systems.[4] The *proximity principle* is a Gestalt theory that specifies that objects or shapes that are close to each other appear to form groups. This is why our brains sometimes perceive multiple, distinct parts as a whole, as the following figure illustrates.

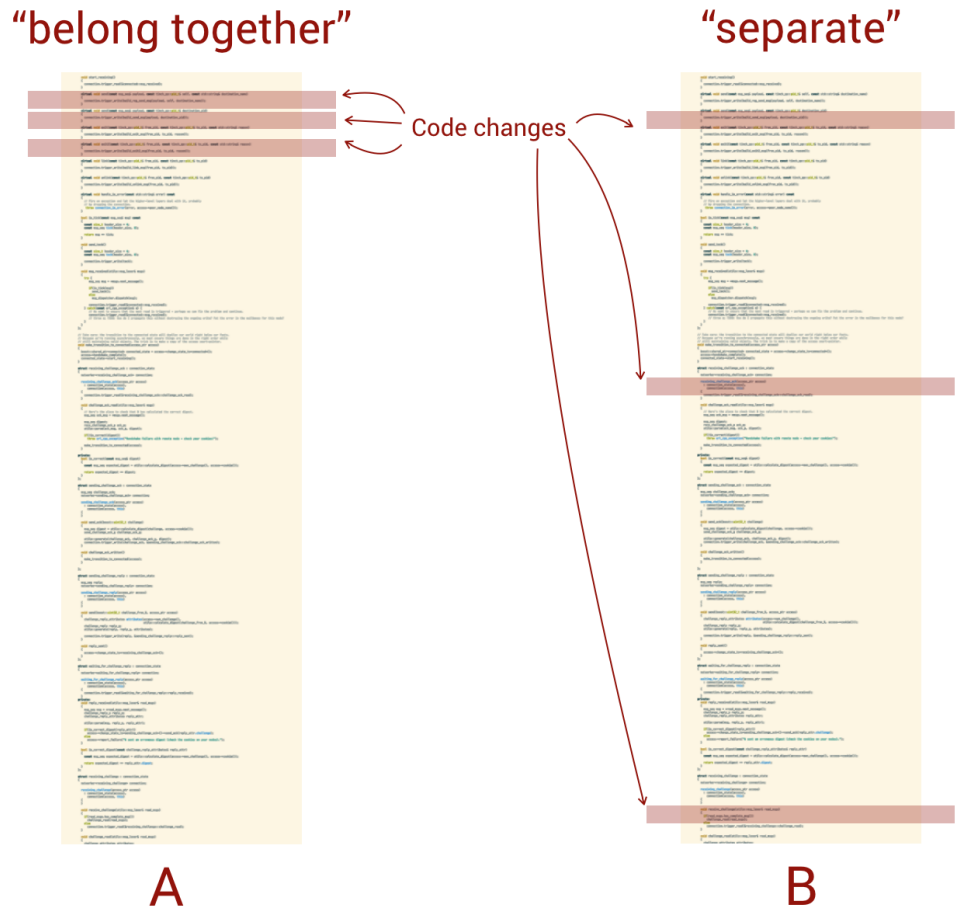


If we translate the proximity principle to software, it means we should favor a structure that guides our code-reading brain toward interpreting related parts of the source file as a group. Let's look at a specific example by considering the information carried by the changes we make to our code, shown in the figure on page 6.

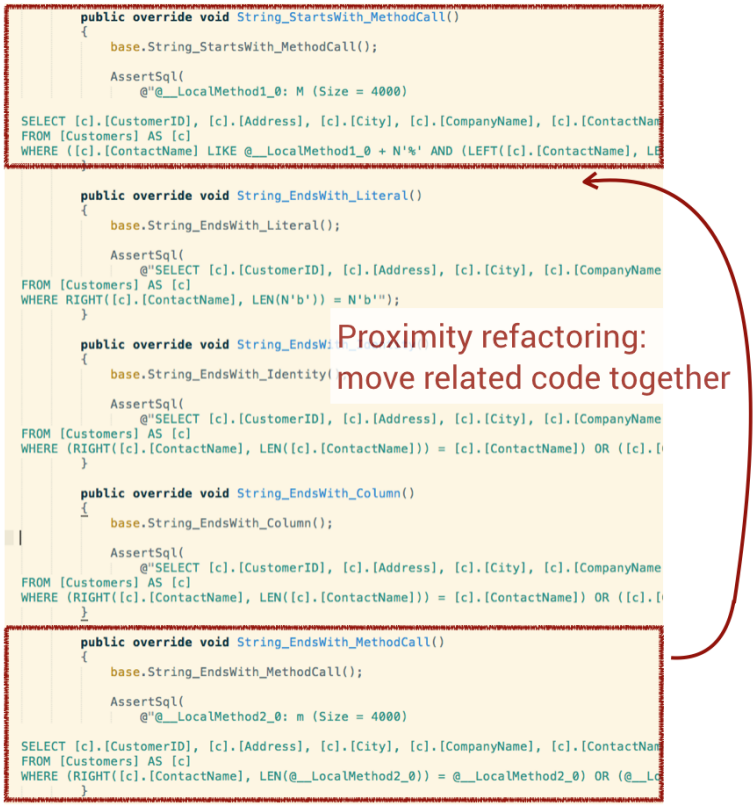
In the this figure, both case A and B show three hypothetical changes that form a single commit. However, there's a different effort behind them although the same amount of code gets changed. Remember that as developers we spend most of our time trying to understand existing code. With the proximity principle in mind, case A exhibits a change pattern that suggests a group of related functionality. This is in contrast to case B, where the parts that make up a concept are distributed, which means we initially—and falsely—perceive these as unrelated functions.

Now, let's return to the code duplication we identified in Entity Framework Core, where we found the methods `String_EndsWith_MethodCall` and `String_StartsWith_MethodCall` change together. If you look at the whole file you see that there are 50 lines of code between these two methods. More important, there are three other methods modeling different behavior interspersed between them. We improve this code, as the figure on page 7 illustrates, by moving methods that belong together close to each other.

4. https://en.wikipedia.org/wiki/Gestalt_psychology



The proximity principle is a much-underused refactoring technique that uses feedback from how our code evolves. By ordering our functions and methods according to our change patterns we communicate information that isn’t expressible in programming-language syntax. That information serves as a powerful guide to both the programmer and, more important, the code reader on which parts belong together and how we expect the code to grow.


public override void String_StartsWith_MethodCall()
{
base.String_StartsWith_MethodCall();
AssertSql(
@"@__LocalMethod1_0: M (Size = 4000)
SELECT [c].[CustomerID], [c].[Address], [c].[City], [c].[CompanyName], [c].[ContactNam
FROM [Customers] AS [c]
WHERE ([c].[ContactName] LIKE @__LocalMethod1_0 + N'%' AND (LEFT([c].[ContactName], LE
public override void String_EndsWith_Literal()
{
base.String_EndsWith_Literal();
AssertSql(
@"SELECT [c].[CustomerID], [c].[Address], [c].[City], [c].[CompanyName
FROM [Customers] AS [c]
WHERE RIGHT([c].[ContactName], LEN(N'b')) = N'b'");
public override void String_EndsW
{
base.String_EndsWith_Identity(
AssertSql(
@"SELECT [c].[CustomerID], [c].[Address], [c].[City], [c].[CompanyName
FROM [Customers] AS [c]
WHERE (RIGHT([c].[ContactName], LEN([c].[ContactName])) = [c].[ContactName]) OR ([c].[
public override void String_EndsWith_Column()
{
base.String_EndsWith_Column();
AssertSql(
@"SELECT [c].[CustomerID], [c].[Address], [c].[City], [c].[CompanyName
FROM [Customers] AS [c]
WHERE (RIGHT([c].[ContactName], LEN([c].[ContactName])) = [c].[ContactName]) OR ([c].[
public override void String_EndsWith_MethodCall()
{
base.String_EndsWith_MethodCall();
AssertSql(
@"@__LocalMethod2_0: m (Size = 4000)
SELECT [c].[CustomerID], [c].[Address], [c].[City], [c].[CompanyName], [c].[ContactNam
FROM [Customers] AS [c]
WHERE (RIGHT([c].[ContactName], LEN(@__LocalMethod2_0)) = @__LocalMethod2_0) OR (@__Lo
Proximity refactoring:
move related code together