**SOFTWARE ENGINEERING**

**HOMEWORK 4**

الطالبة : ريان ناطورة

الفىة :الخامسة

**خطوات انشاء المشروع :**

**انشاء مجلد جديد ونكتب التعليمات في terminal:**

dotnet new sln –n LoanApp

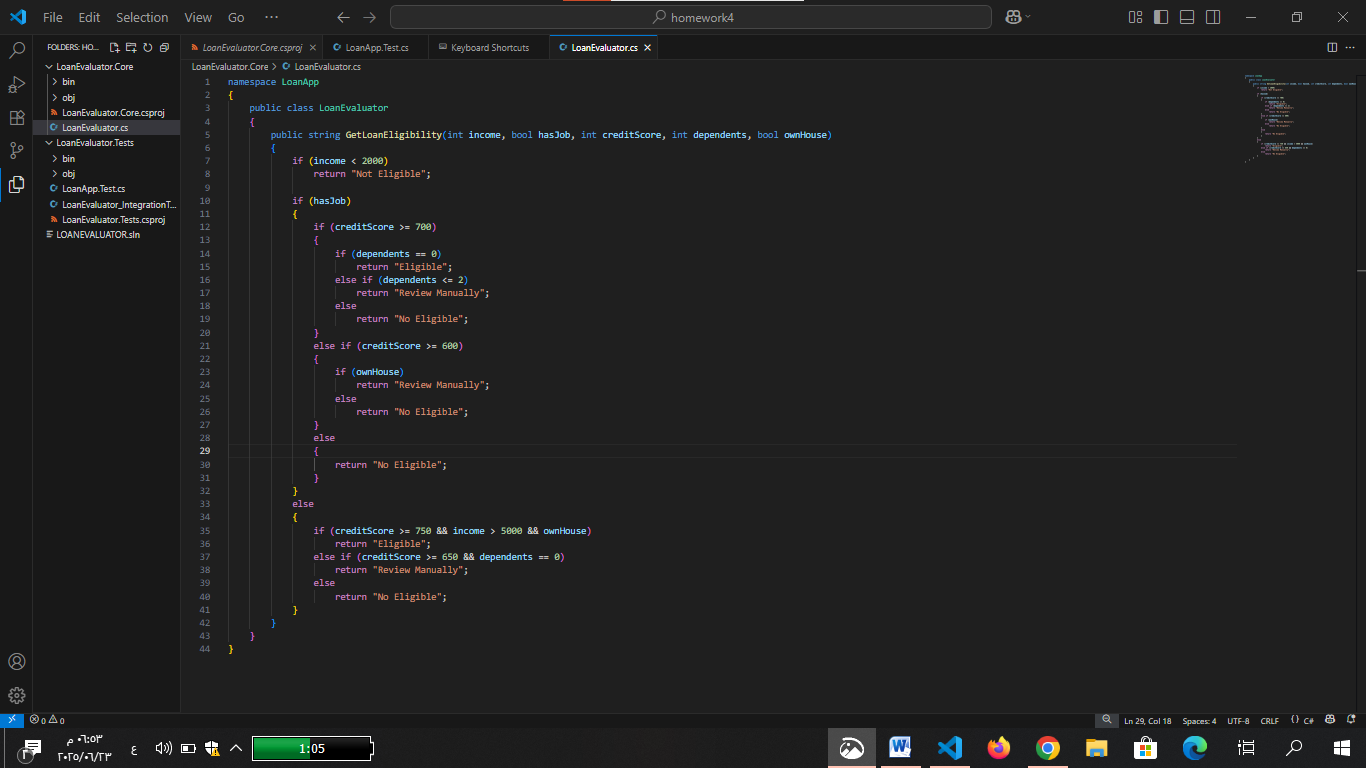
dotnet new classlib -n LoanApp.Core

dotnet new xunit -n LoanApp.Tests

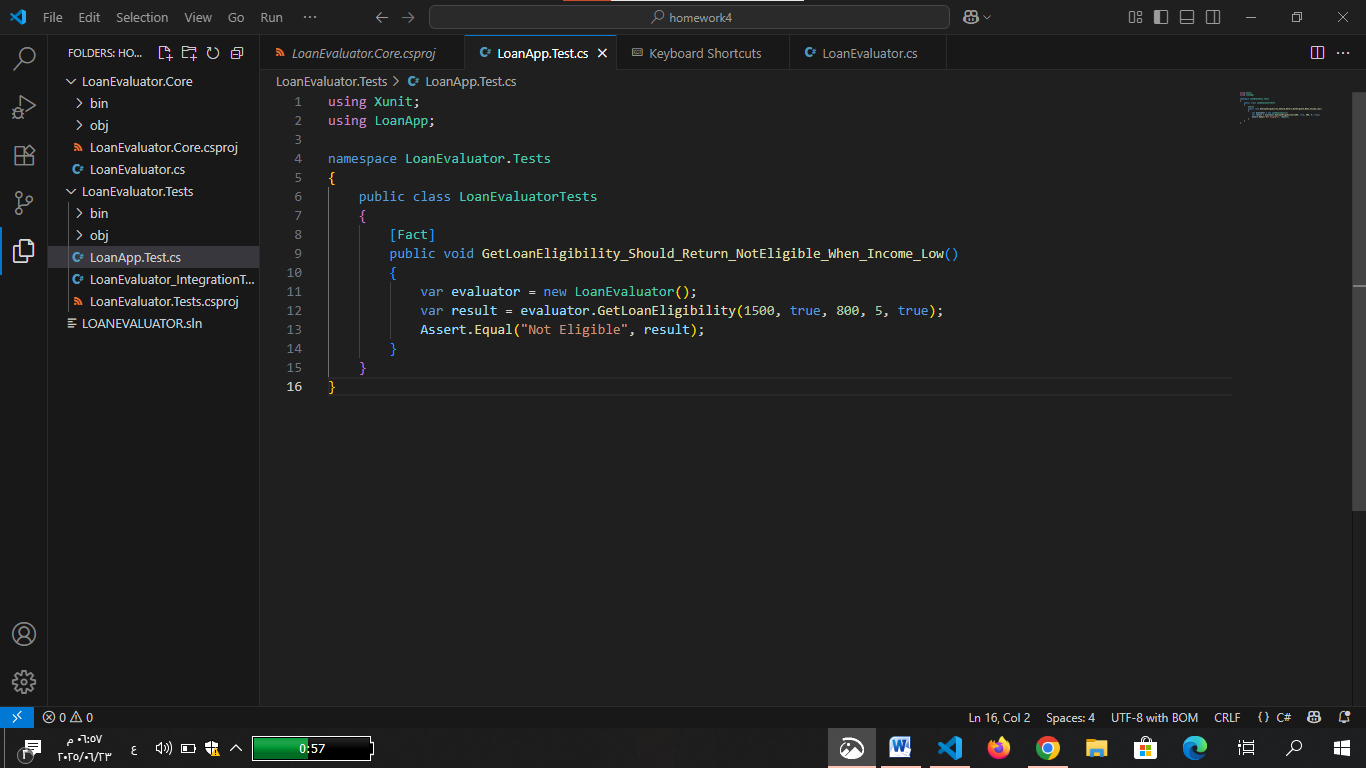
dotnet sin add LoanApp.Core/LoanApp.Core.csproj

dotnet add LoanApp.Tests/LoanApp.Tests.csproj reference LoanApp.Core.csproj

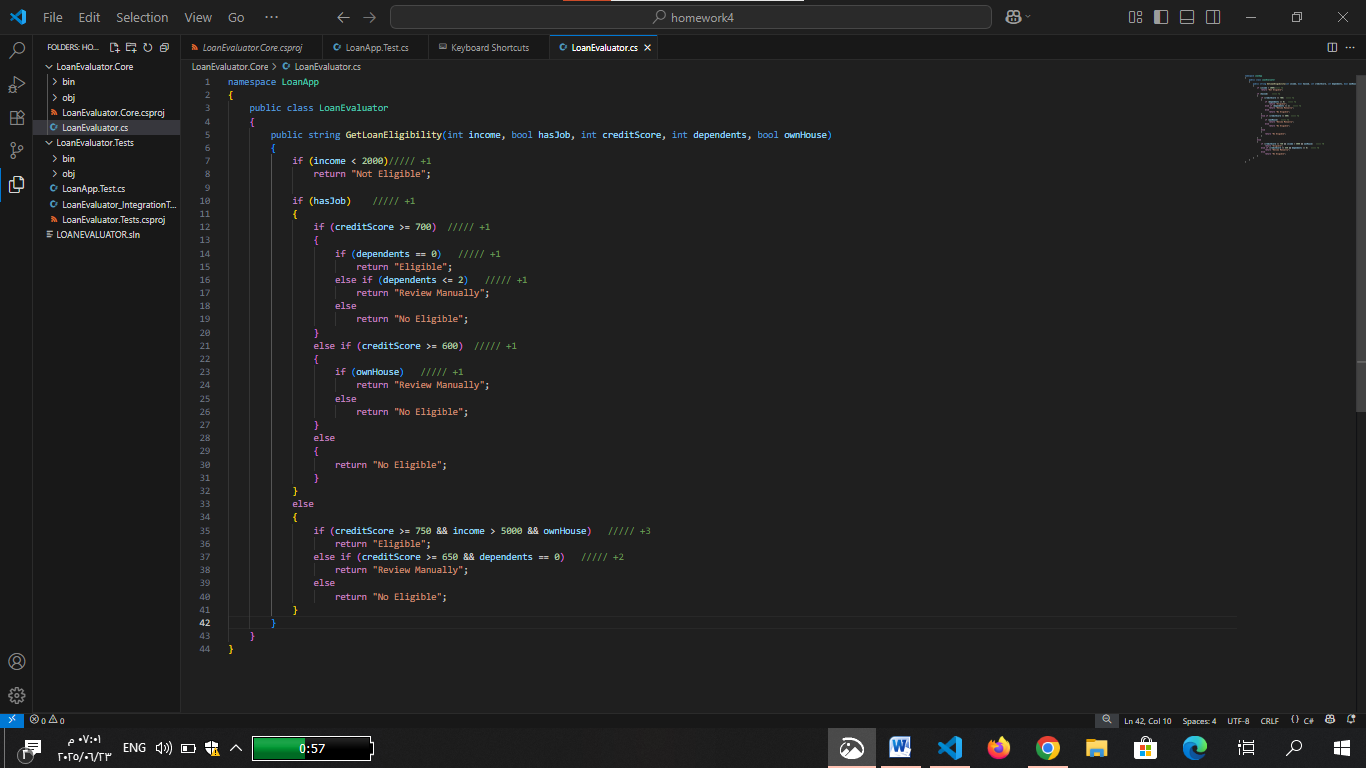
1. محتوى ملف LoanEvaluator.cs )كود المحاضرة):



1. محتوى ملف LoanApp.Test.cs:



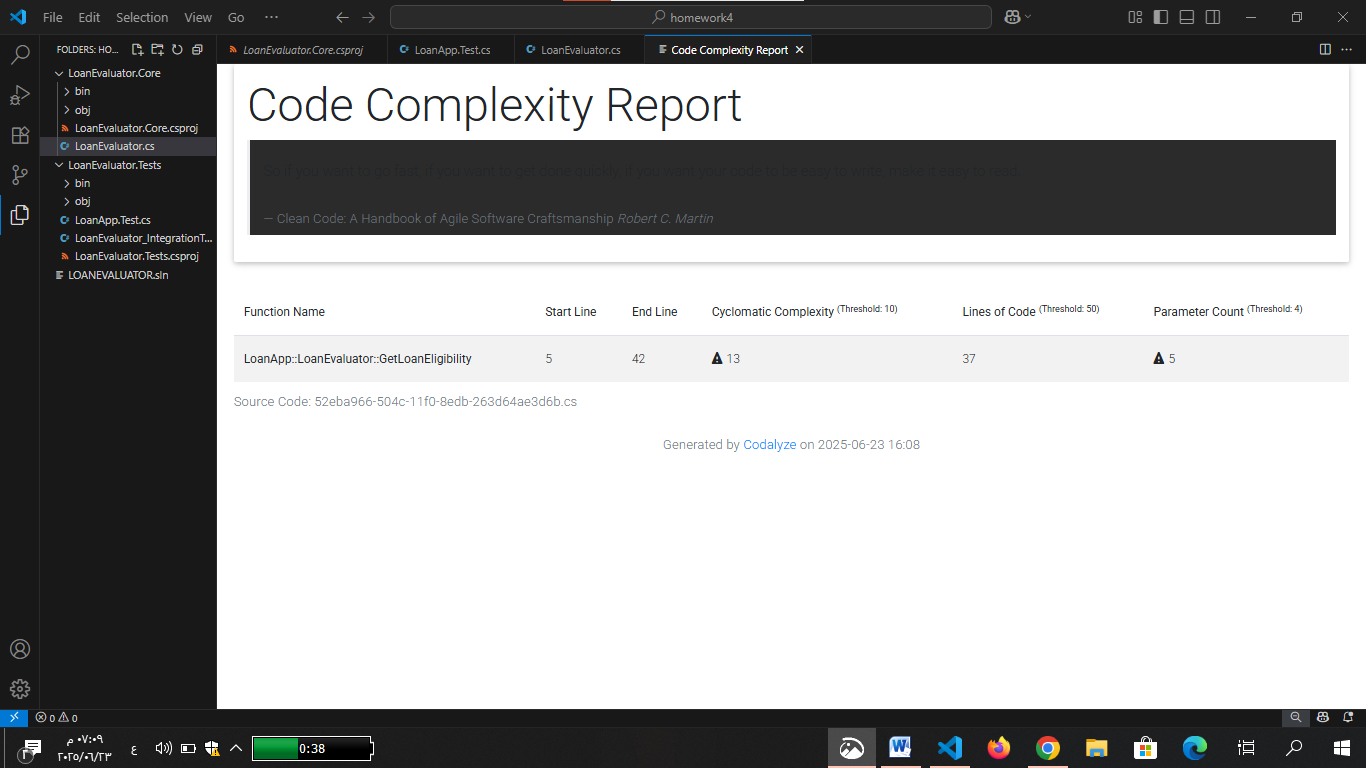
1. حساب التعقيد يدويا قبل التحسين Cyclomatic Complexity:



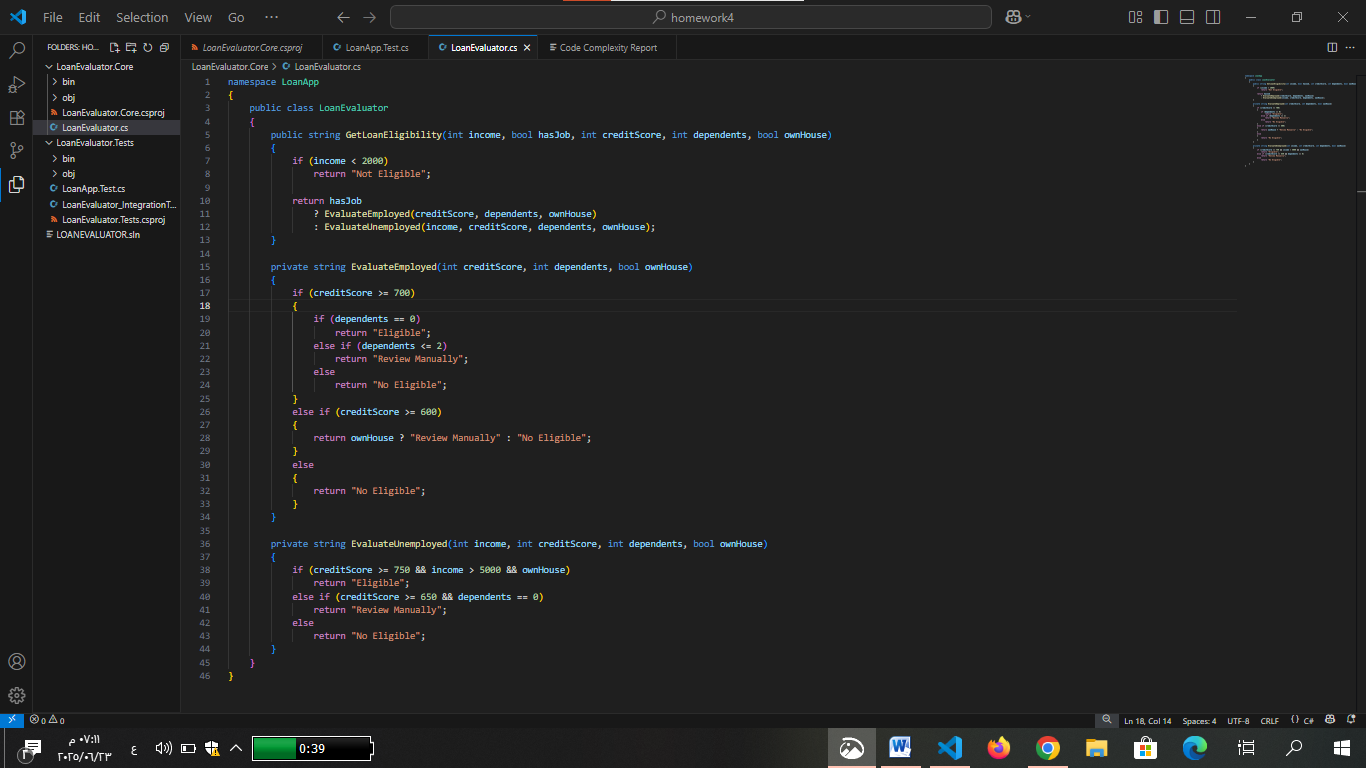
المجموع = 12 نقاط القرار +1(ثابت)

Cyclomatic Complexity =13

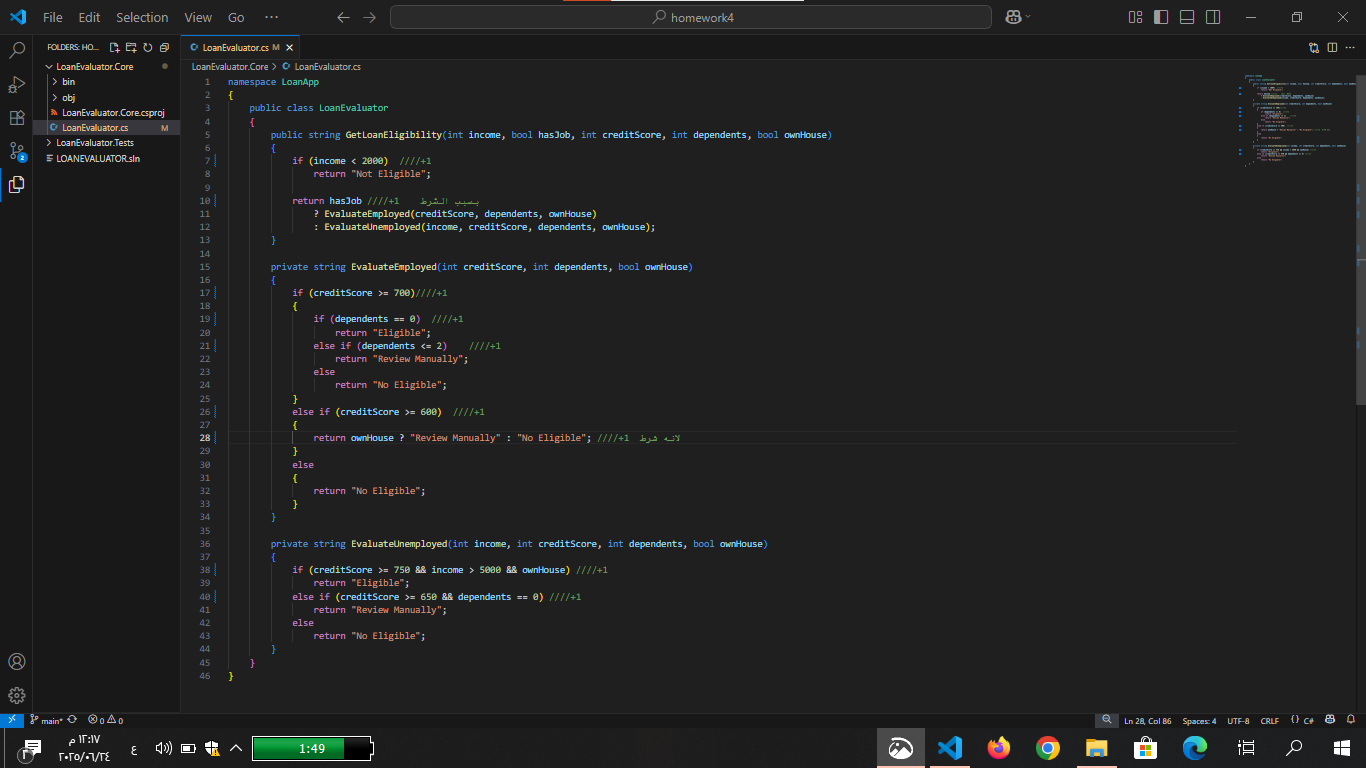
1. حساب التعقيد باستخدام أداة Codalyze:



1. الكود بعد التحسين Refectoring:



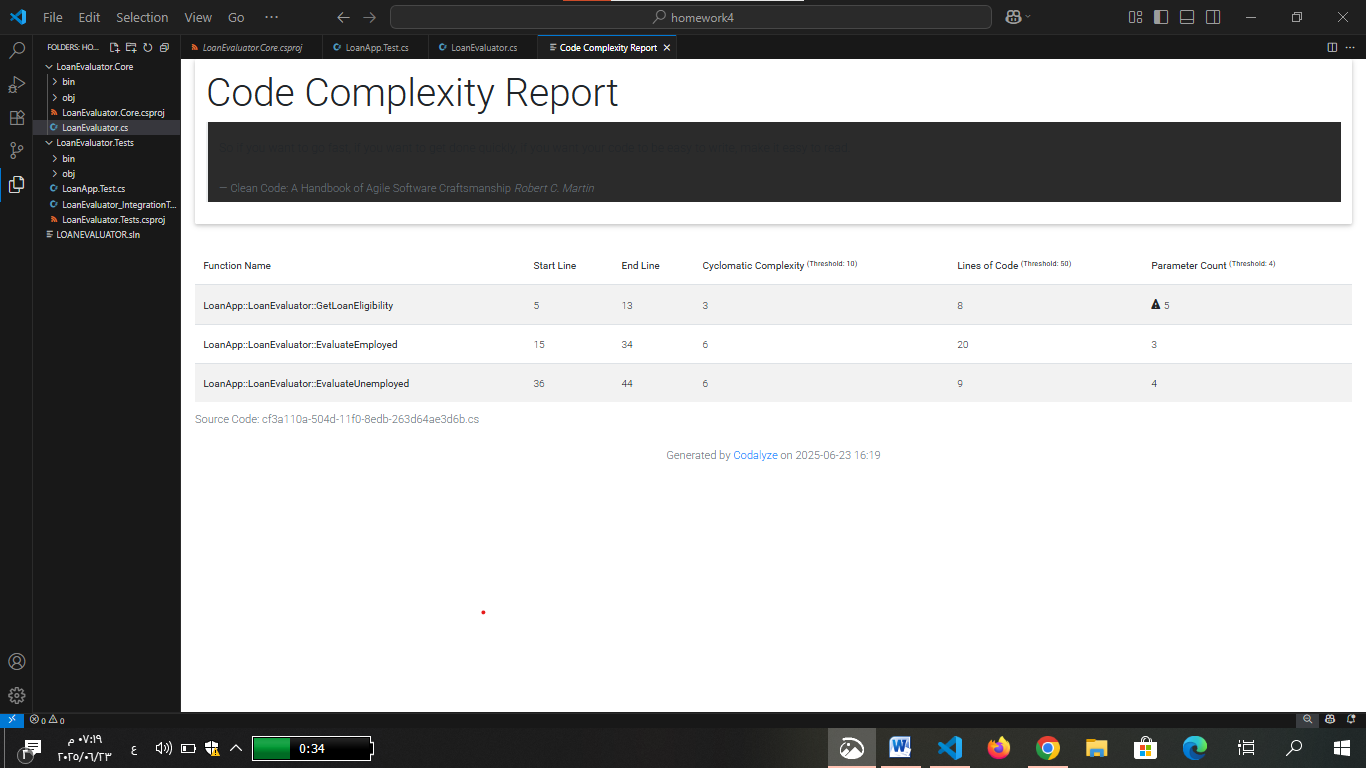
1. حساب التعقيد يدويا بعد التحسين Cyclomatic Complexity:



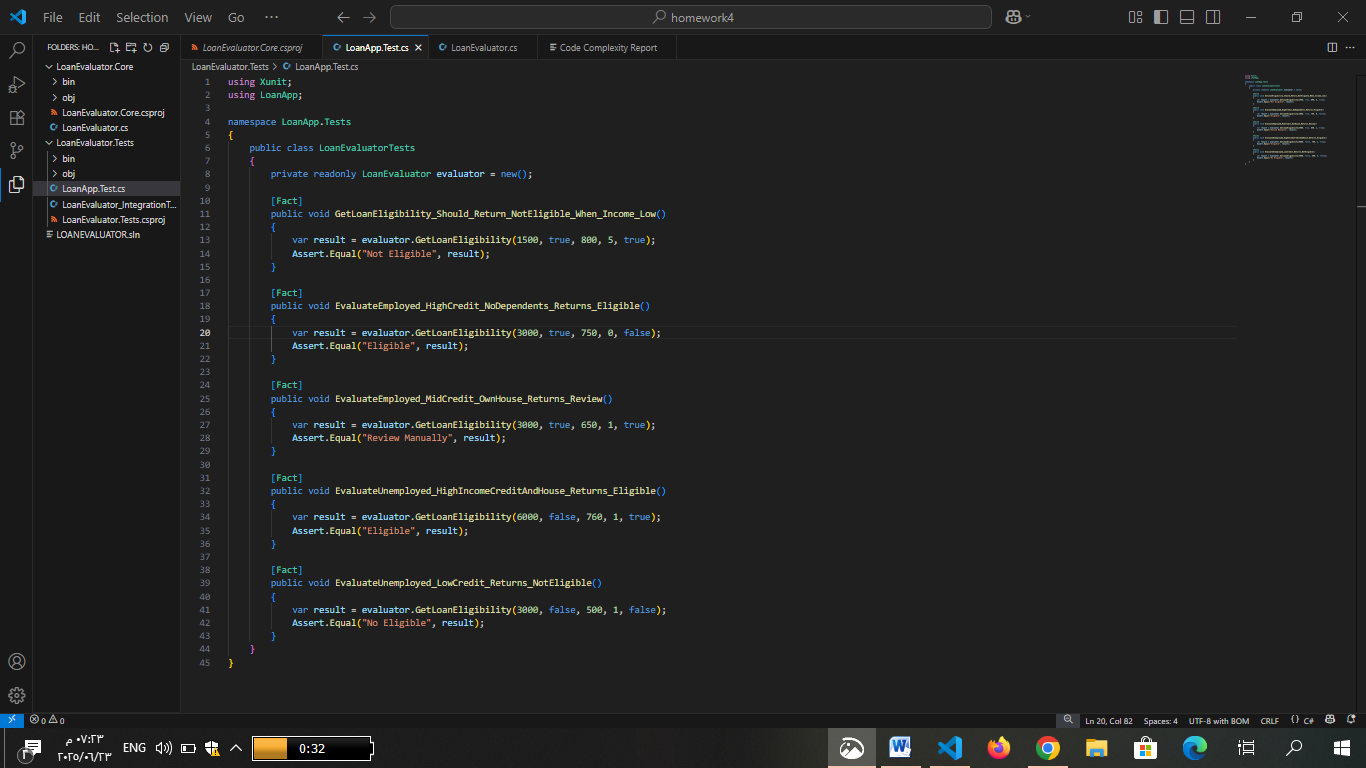
المجموع =9نقاط القرار +1(ثابت)

Cyclomatic Complexity =10

1. حساب التعقيد باستخدام أداة Codalyze بعد التحسين:



1. كود الاختبار لتغطيت جميع الحالات:



شرح الحالات :

1. GetLoanEligibility\_Should\_Return\_NotEligible\_When\_Income\_Low

تتحقق هذه الحالة من أن أي شخص دخله أقل من 2000 يُرفض طلبه مباشرة، بغض النظر عن الوظيفة أو الائتمان أو أي عامل آخر.

2. EvaluateEmployed\_HighCredit\_NoDependents\_Returns\_Eligible

تتحقق هذه الحالة من أن الموظف الذي لديه دخل جيد، ودرجة ائتمان عالية (750 أو أكثر)، ولا يعيل أحد، يعتبر مؤهلًا للحصول على قرض.

3. EvaluateEmployed\_MidCredit\_OwnHouse\_Returns\_Review

تتحقق هذه الحالة من أن الموظف الذي لديه درجة ائتمان متوسطة (بين 600 و699)، ويملك منزلًا، يتم تحويل طلبه إلى مراجعة يدوية.

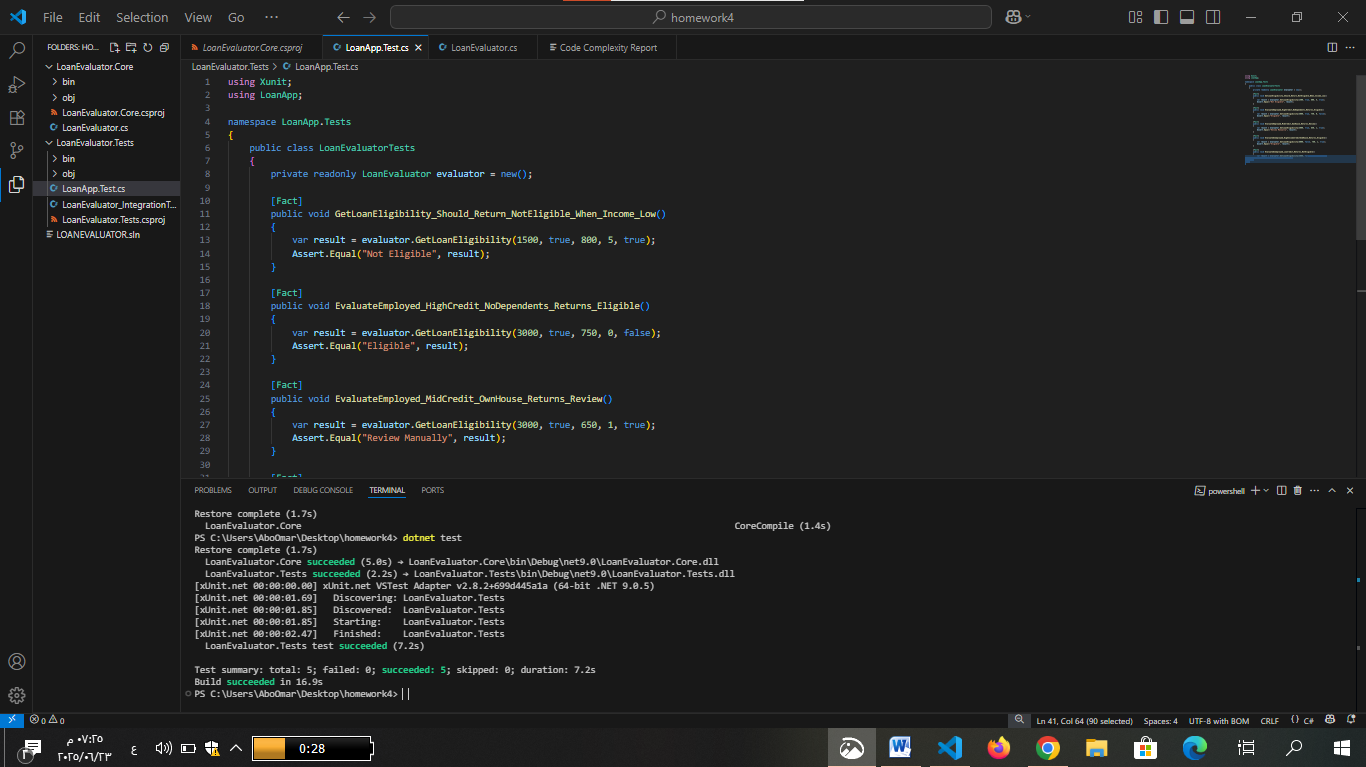
4. EvaluateUnemployed\_HighIncomeCreditAndHouse\_Returns\_Eligible

تتحقق هذه الحالة من أن الشخص غير الموظف، إذا كان دخله مرتفعًا (أكثر من 5000)، ودرجته الائتمانية عالية (750 أو أكثر)، ويملك منزلًا، يعتبر مؤهلًا.

5. EvaluateUnemployed\_LowCredit\_Returns\_NotEligible

تتحقق هذه الحالة من أن الشخص غير الموظف الذي لديه درجة ائتمان ضعيفة (أقل من 600) ولا يملك منزلًا يتم رفض طلبه بشكل مباشر.

الخرج:



مقارنة قبل وبعد عملية التحسين:

* قبل:
* تعقيد حلقي (Cyclomatic Complexity) عالي = 13
* صعب الفهم والصيانة
* يصعب اختبار الحالات الداخلية كلٌ على حدة
* دالة واحدة تحتوي كل منطق القرار

بعد:

* تقليل التعقيد الحلقي (Cyclomatic Complexity) إلى 3 في كل تابع فرعي بدلًا من 13 في تابع واحد.
* تبسيط منطق القرار عبر توزيع الشروط إلى توابع صغيرة متخصصة.
* تحسين قابلية القراءة والفهم بفضل فصل الحالات (موظف / غير موظف) في توابع مستقلة.
* سهولة اختبار كل حالة على حدة عبر كتابة اختبارات وحدة منفصلة لكل تابع فرعي.
* رفع قابلية الصيانة والتطوير، حيث يمكن تعديل منطق فئة معينة دون التأثير على الأخرى.
* إعادة استخدام التوابع الفرعية عند الحاجة دون تكرار الكود.
* تهيئة الكود ليكون أكثر مرونة للتوسع مستقبلاً مثل إضافة قواعد جديدة أو شروط خاصة دون تعقيد إضافي.