

Kauno Technologijos Universitetas

Informatikos Fakultetas

**T120B162 Programų sistemų testavimas**

Trečio laboratorinio darbo aprašymas

**Atliko:**

IFF-6/8 gr. stud.

Tadas Laurinaitis

Kaunas 2019

# Darbo užduotis

1. Code review for the selected on the 1st lab software:
   1. Review several classes,
   2. Check for possible issues locations,
   3. Find and use suitable code review checklists.
   4. Document possible places in the document (table: source file, class, method, line, unsatisfied checklist rule, comment).
2. Static code analysis for the selected on the 1st lab software:
   1. Select static code analysis tool: possible chooses are JTest, .NET Test, FindBugs, FxCop, etc.
   2. Execute code analysis tool on the selected software.
3. Design and implement one static code analysis rule.
   1. Execute code analysis tool on the selected software sing newly implemented rule.
   2. Document detected issues in the document (table: source file, class, method, line, unsatisfied checklist rule, comment).

# Programos kodo peržiūros eiga

Pirmiausia yra peržiūrimas programos kodas nenaudojant jokių įrankių. Šis procesas vadinamas kodo peržiūra. Tam, kad kodo peržiūra būtų nuosekli, vadovaujamasi šiais kodo peržiūros punktais: 1) Kodo formatavimas – ar kodas rašomas nuosekliai, ar tinkami kodo eilučių atitraukimai nuo kairiojo krašto, ar yra taikomos teisingos kodo rašymo taisyklės (pvz., naudojamas CamelCase); 2) Kodo architektūra – ar kodas yra išskaidytas į komponentus arba failus (pvz., stiliaus ir scenarijų kalbų failai nėra saugomi viename faile);

3) Kodo gerosios praktikos – ar nėra kietai įsiuto kodo, ar naudojamos konstantos vietoj įkoduotų bereikšmių skaičių (pvz., if (role == 1)), ar rašomi tikslingi komentarai, ar vengiama didelių if/else blokų, ar tinkamai išnaudojamos projekte naudojamų karkasų galimybės ir pan;

4) Kodo saugumas – ar kodas yra apsaugotas nuo SQL injekcijų, cross-side scripting, ar tie programos duomenys/rezultatai, kurie neturėtų būti matomi, nėra išvedami į konsolę/naršyklės langą, ar tikrinamos įvedimo laukų reikšmės (ar yra taikoma laukų validacija).

Taip pat tam, kad būtų išlaikomas nuoseklumas, programos kodo peržiūros metu rastos klaidos bus dokumentuojamos tokiomis lentelėmis:

|  |
| --- |
| Failo pavadinimas |
| Klasė |
| Metodas |
| Eilutė |
| Taisyklės neatitikimas |
| Komentaras |

# Programos Kodo peržiūra

Atlikus kodo peržiūrą, programoje buvo rąsta keletas klaidų ir/ar pastebėjimų. Jie buvo dokumentuojami lentelėse:

|  |
| --- |
| SimpleEnemyCreator.cs |
| SimpleEnemyCreator |
|  |
| 25 |
| Kodo gerosios praktikos |
| Įhardcode‘intos paveiksliuko dydžio reikšmės. Jeigu bus norima naudoti kitokias paveiksliuko dydžio reikšmes, reikės visose klasėse eiti ir keisti skaičius. Būtų protingiau naudoti konstantas/configuracinį failą. |

|  |
| --- |
| SimplePlayerCreator.cs |
| SimplePlayerCreator |
|  |
| 25 |
| Kodo gerosios praktikos |
| Įhardcode‘intos paveiksliuko dydžio reikšmės. Jeigu bus norima naudoti kitokias paveiksliuko dydžio reikšmes, reikės visose klasėse eiti ir keisti skaičius. Būtų protingiau naudoti konstantas/configuracinį failą. |

|  |
| --- |
| BombCreatorHandler.cs |
| BombCreatorHandler |
|  |
| 19-30 |
| Kodo gerosios praktikos |
| Įhardcode‘intos switch atvejų reikšmės. Iš string reikšmių kuriami objektai. Jeigu padaroma nors menkiausia rašybos klaida, objektų kūrimas veikia neteisingai. |

|  |
| --- |
| Backward.cs |
| Backward |
|  |
| 21, 26 |
| Kodo gerosios praktikos |
| Įhardcode‘intos y koordinatės padidinimo/pamažinimo reikšmės. Būtų tikslinga naudoti konstantas arba configuracinį failą. |

|  |
| --- |
| Forward.cs |
| Forward |
|  |
| 21, 26 |
| Kodo gerosios praktikos |
| Įhardcode‘intos y koordinatės padidinimo/pamažinimo reikšmės. Būtų tikslinga naudoti konstantas arba configuracinį failą. |

|  |
| --- |
| Left.cs |
| Left |
|  |
| 21, 26 |
| Kodo gerosios praktikos |
| Įhardcode‘intos x koordinatės padidinimo/pamažinimo reikšmės. Būtų tikslinga naudoti konstantas arba configuracinį failą. |

|  |
| --- |
| Right.cs |
| Right |
|  |
| 22, 27 |
| Kodo gerosios praktikos |
| Įhardcode‘intos x koordinatės padidinimo/pamažinimo reikšmės. Būtų tikslinga naudoti konstantas arba configuracinį failą. |

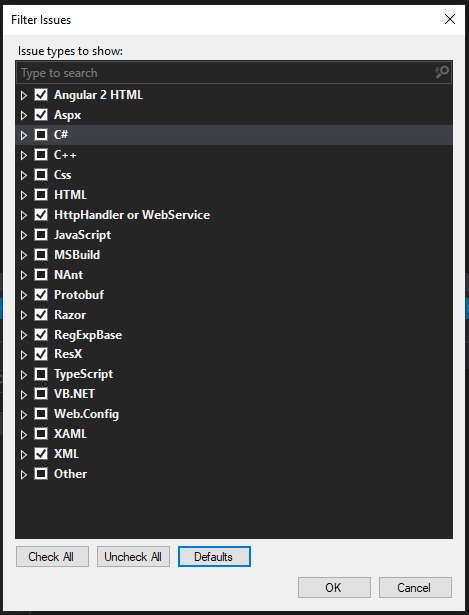
|  |
| --- |
| Crate.cs |
| Crate |
|  |
| 66-69 |
| Kodo saugumas |
| Kviečiant nerealizuotą Destroy metodą, niekas neįvyks. Reikėtų mesti NotImplementedException, kad nekiltų jokių neaiškumų. |

|  |
| --- |
| Player.cs |
| Player |
|  |
| 13-14 |
| Kodo gerosios praktikos |
| Įhardcode‘intos x ir y koordinatės reikšmės. Būtų tikslinga naudoti konstantas arba configuracinį failą. |

|  |
| --- |
| Player.cs |
| Player |
|  |
| 13-14 |
| Kodo gerosios praktikos |
| Įhardcode‘intos x ir y koordinatės reikšmės. Būtų tikslinga naudoti konstantas arba configuracinį failą. |

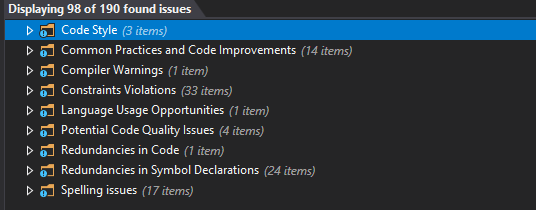
# Kodo statinė analizė

Kadangi pasirinktas projektas testavimui buvo padarytas su .NET Core ir didžioji dalis kodo parašyta su C#, todėl statinei kodo analizei buvo pasirinktas Resharper įrankio statinis kodo analizatorius. Šis įrankis yra labai naudingas, kadangi jį įsirašant yra galimybė taip pat įsirašyti įvairių papildomų funkcijų, tokių kaip unit test‘avimas, kodo padengimas, statinės kodo analizės įrankis, be to Reshaper turi labai gerą integraciją į Visual Studio. Įrankio rašomi pranešimai ir parodomos klaidos/problemos yra aiškiai suformuluotos, galima kodo, neatitikusio kažkokio reikalavimo, peržiūra. Naudoti filtrai, kad atmesti nenaudingas/nereikšmingas klaidas/problemas:



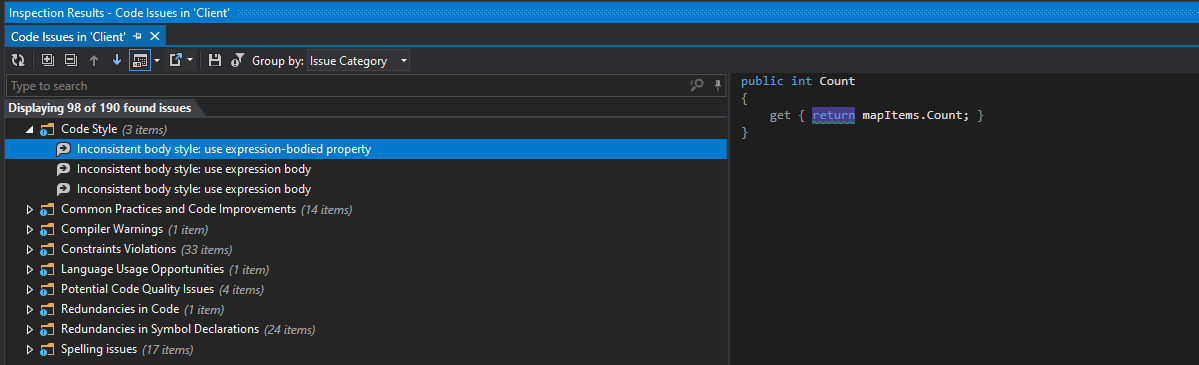
1 pav. Statinėje analizėje naudoti problemų filtrai

Įvykdžius kodo analizę ir pritaikius filtrus gauti rezultatai:



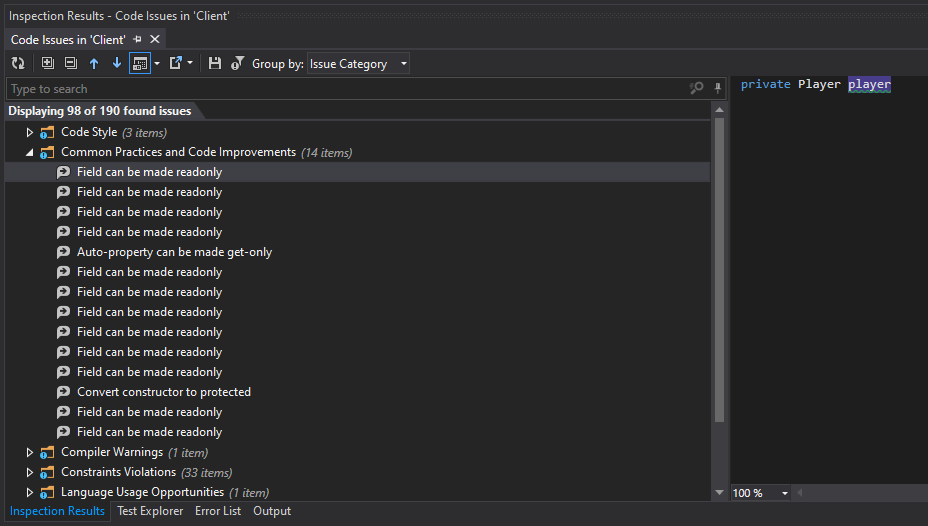
2 pav. Gautos visos klaidos/problemos po statinės analizės

Iš gautų rezultatų matome, kad daugiausiai klaidų buvo gauta „Constraints Violations“ ir „Redundancies in Symbol Declarations“ kategorijose.



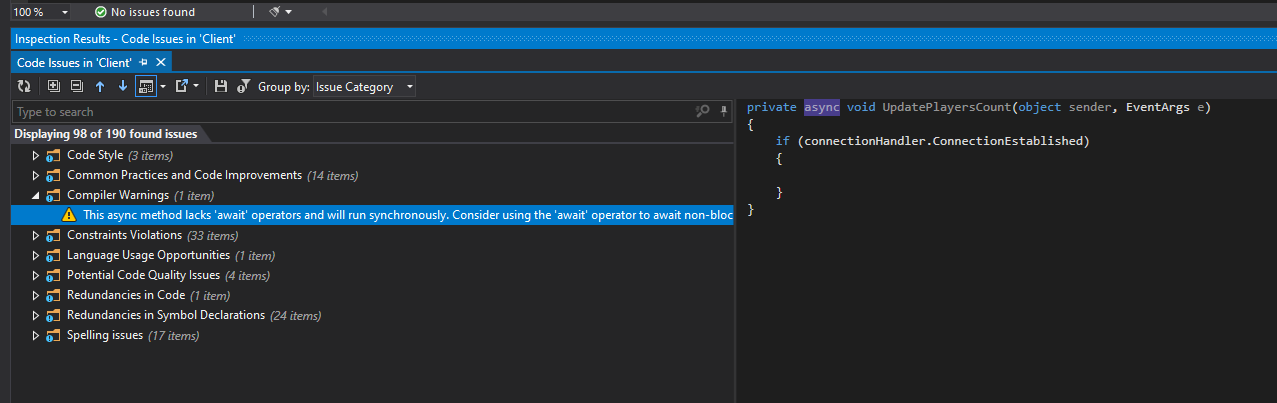
3 pav. Code Style problemų kategorija

Pirmojoje kategorijoje, kuri vadinasi „Code Style“ (Kodo stilius), buvo rastos tik 3 problemos dėl nevienodo stiliaus. Šios klaidos nėra labai svarbios ir neturi įtakos kodo vykdymui, tačiau naudojant labai nevienodą ir išsiskiriantį stilių, mažėja kodo skaitomumas.



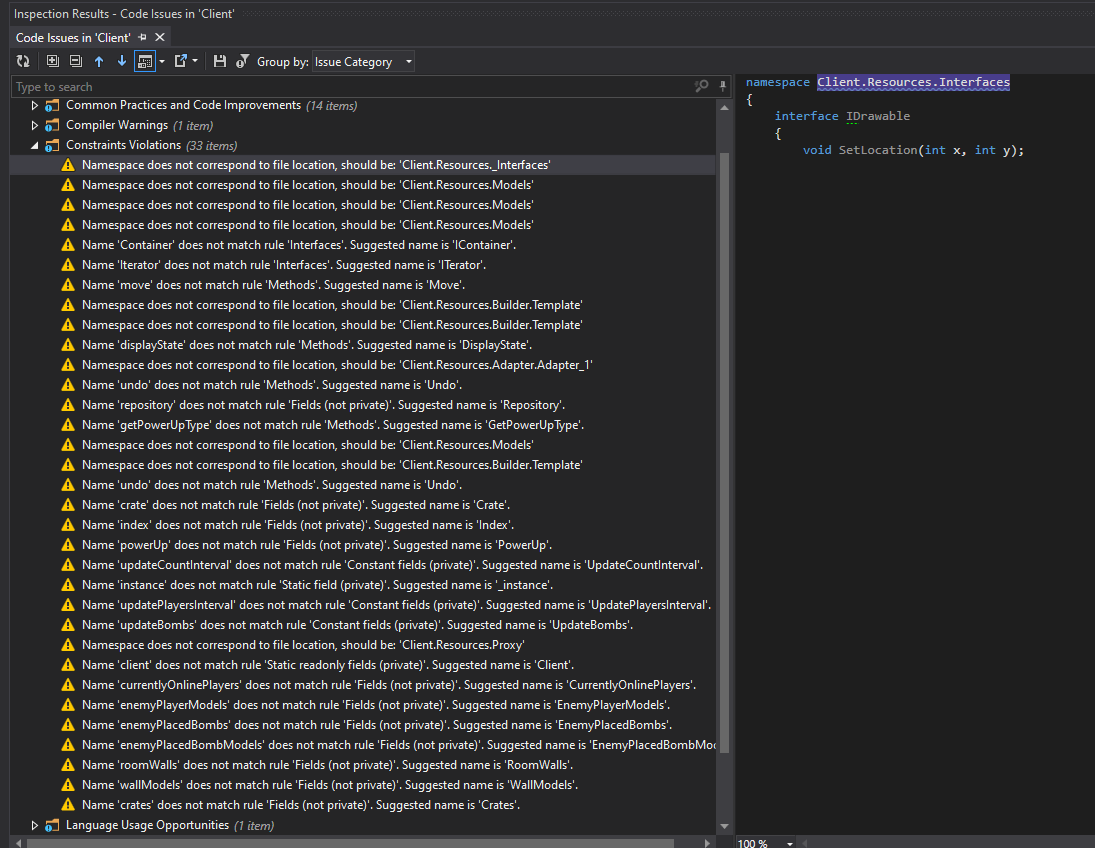
4 pav. Common practices and Code improvements problemų kategorija

Antrojoje kategorijoje, kuri vadinasi „Common practices and Code improvements“ galime matyti, kad įrankis daug kur siūlo pakeisti klasių laukus į readonly. Taip pat buvo pavienių atvejų, kai įrankis siūlė konvertuoti konstruktorių į protected arba pakeisti Auto-property į get-only.



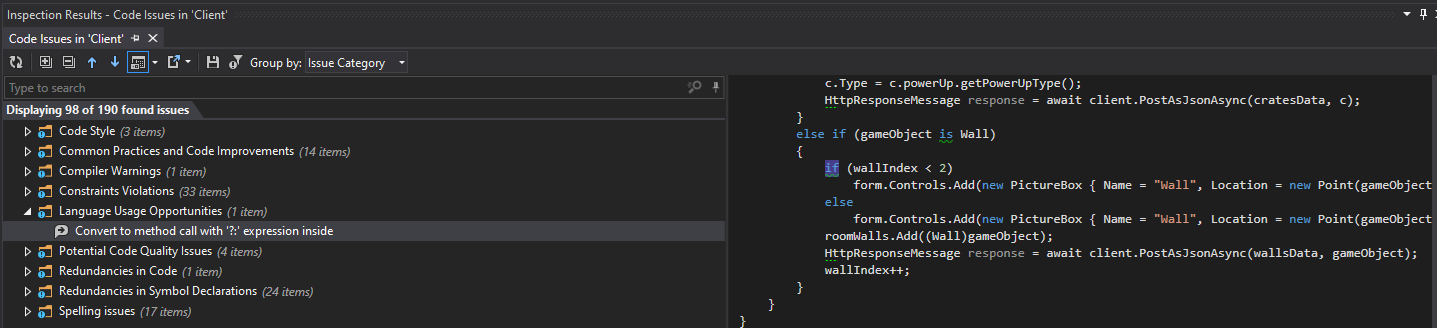
5 pav. Compilator warning problemų kategorija

Trečioje kategorijoje, kuri vadinasi „Compilator warnings“ galime pastebėti, kad įrankis rado tik vieną probleminį atvejį, tačiau šią klaidą rodo dėl to, kad tas metodas nėra pilnai realizuotas.

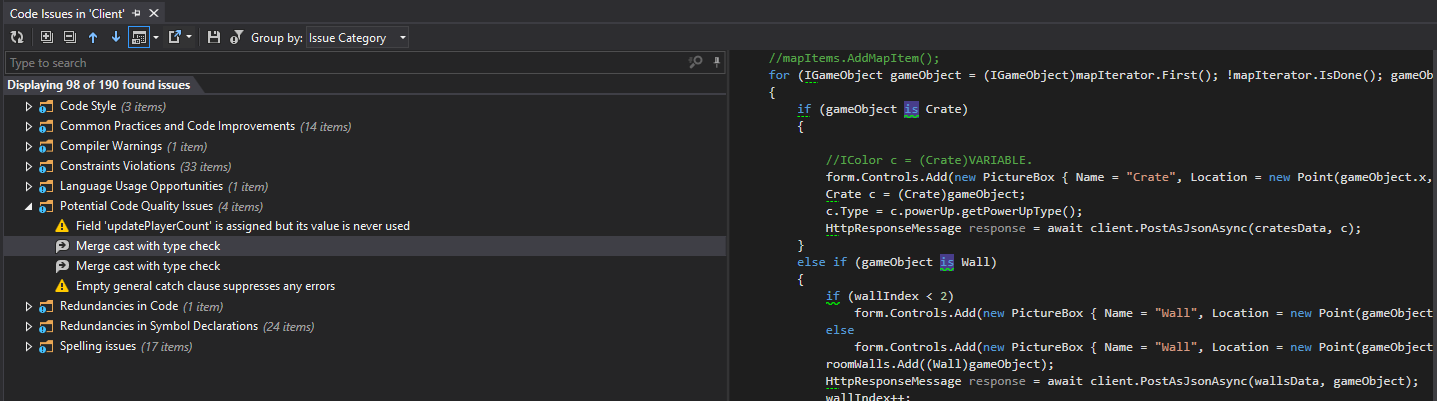


6 pav. Constraints violations problemų kategorija

Ketvirtą kategorijoją, kuri vadinasi „Constraints Violations“, daugiausiai sudaro vardų arba namespace‘ų netinkami pavadinimų formatai. Ši kategorija kodo veikimui reikšmės nedaro, tačiau gali pradėti kenkti navigacijai per kodą, jeigu atsiras dar daugiau netvarkingai užvadintų klasių, metodų ar aplankų.

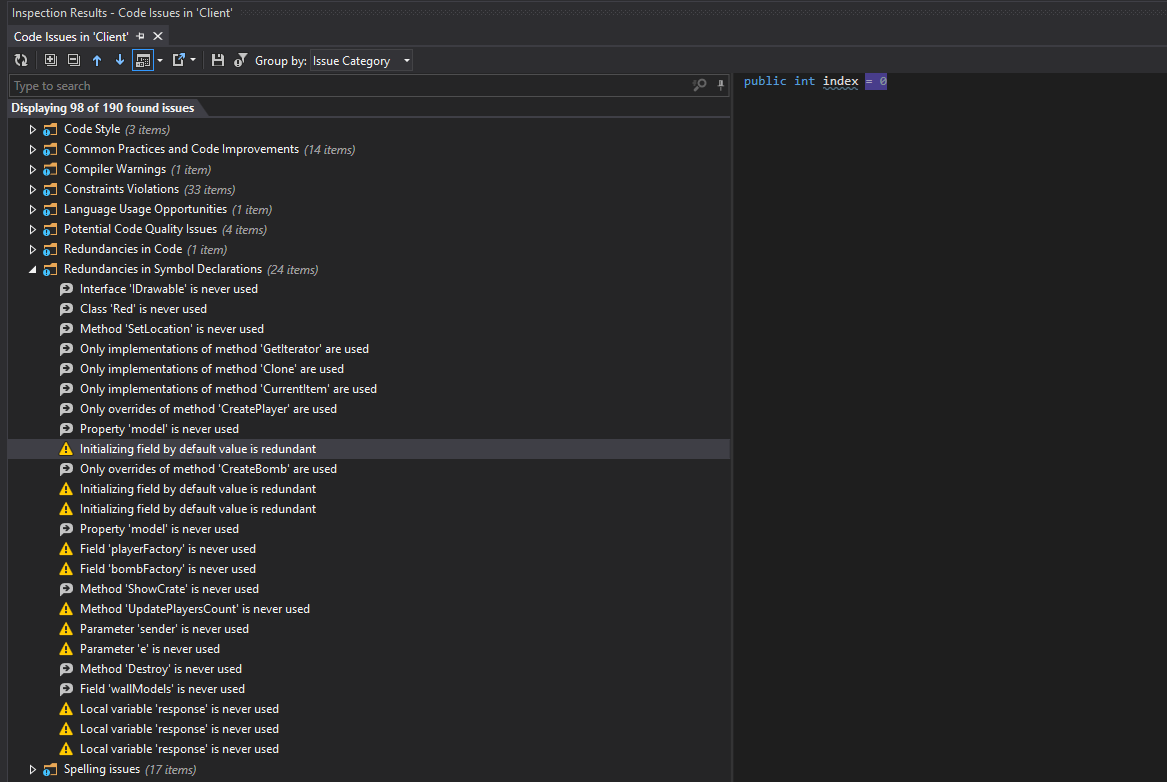


7 pav. Language usage opportunities kategorija



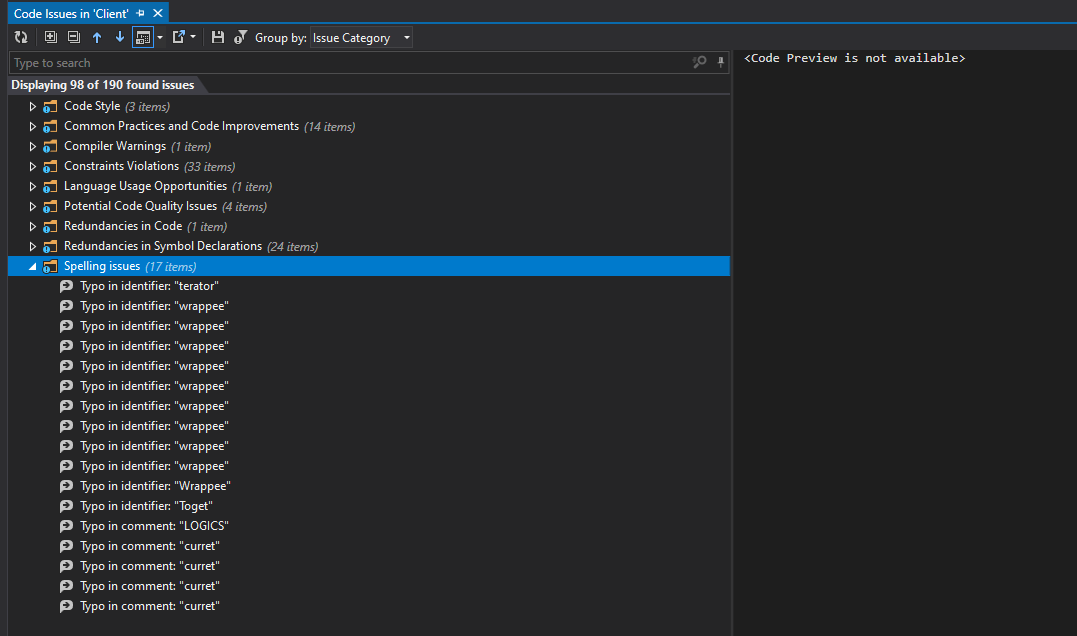
8 pav. Potential Code Quality Issues kategorija

Šeštą kategoriją kuri vadinasi „Code Quality Issues“ sudaro tik 4 problemos: Dvi iš jų siūlo tipo patikrinimą sulieti į cast‘inimą, o kitos dvi susijusios su dar nerealizuotomis sistemos dalimis.



9 pav. Potential Code Quality Issues kategorija

Aštuntą kategoriją, kuri vadinasi „Redundancies in Symbol Declarations“ sudaro 24 problemos. Daugiau negu pusė jų yra susijusios su dar nepanaudotais laukais, kintamaisiais, metodais ir klasėm. Kita pusė yra susijusi su pasenusių programavimo taisyklių naudojimu.



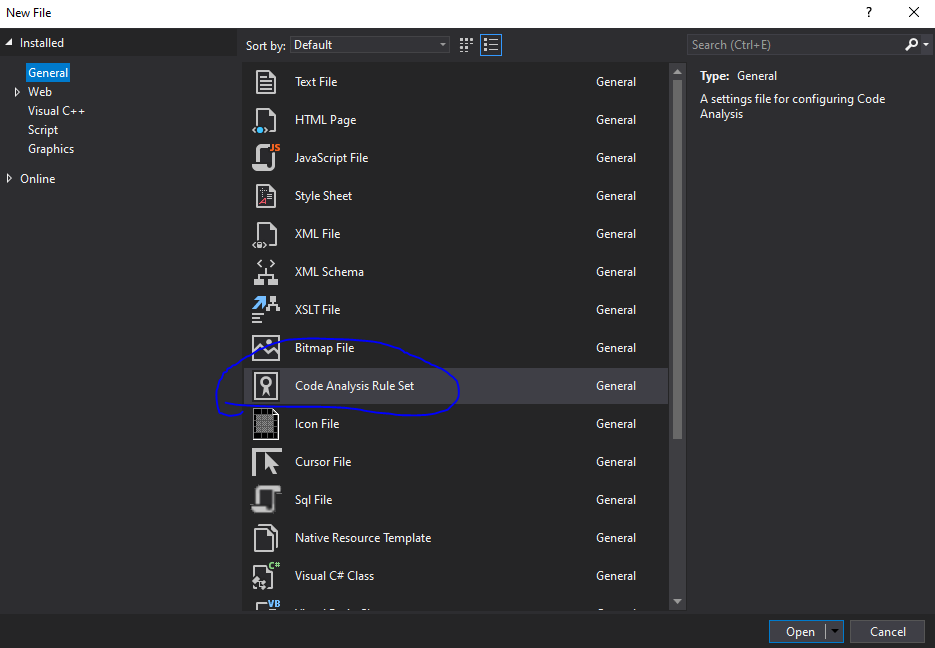
9 pav. Spelling issues kategorija

Šioje paskutinėje kategorijoje matome visas rašybos klaidas, rąstas programos kode. Nors šios problemos programos veikimui įtakos nedaro, gramatines klaidas reikia ištaisyti dėl bendros kodo tvarkos.

# Naujų taisyklių įtraukimas į statinės analizės įrankį

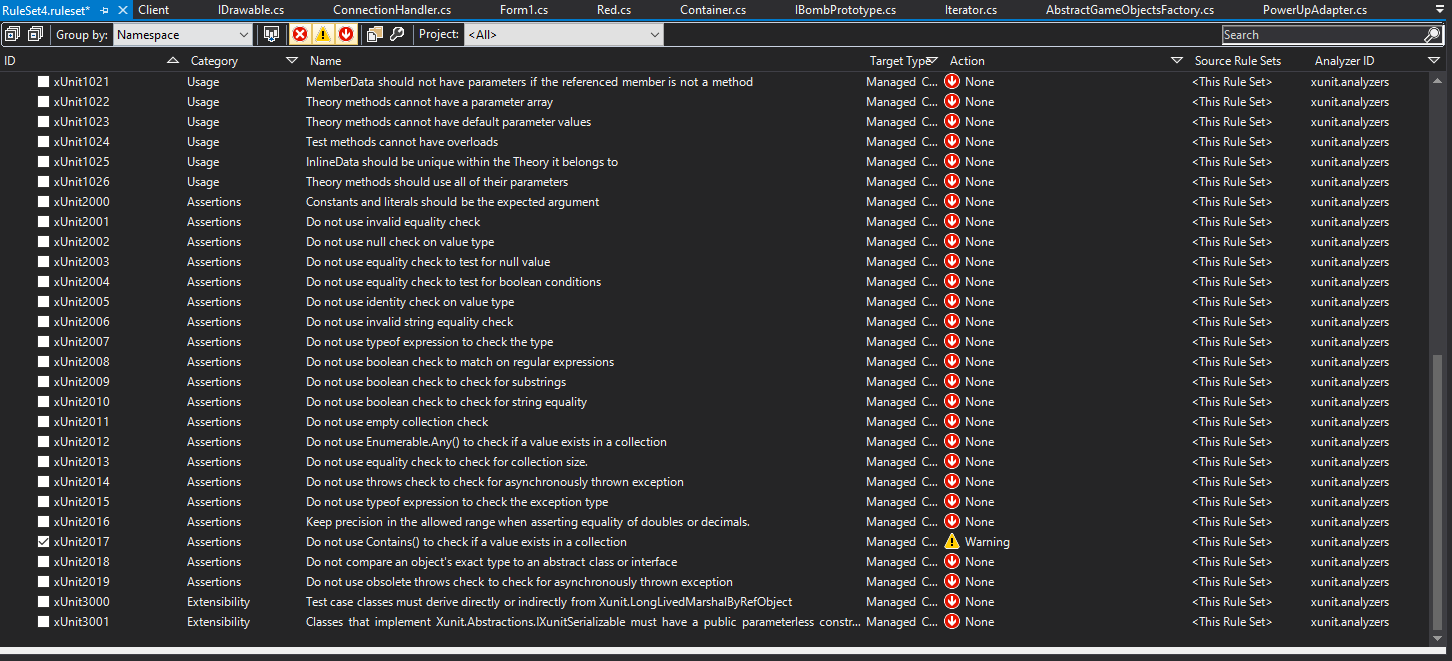
Nors Resharper įrankis turi pakankamai gerą statinę kodo analizę, tačiau jis neturi tokio gero statinės analizės taisyklių rinkinių sudarymo įrankio kaip FxCop, kurį šiai laboratorinio darbo daliai ir naudojau. FxCop įrankis nuo 2012 metų yra įrašomas kartu su visomis Visual Studio versijomis.

Pirmas žingsnis – sukurti naują taisyklių rinkinį:



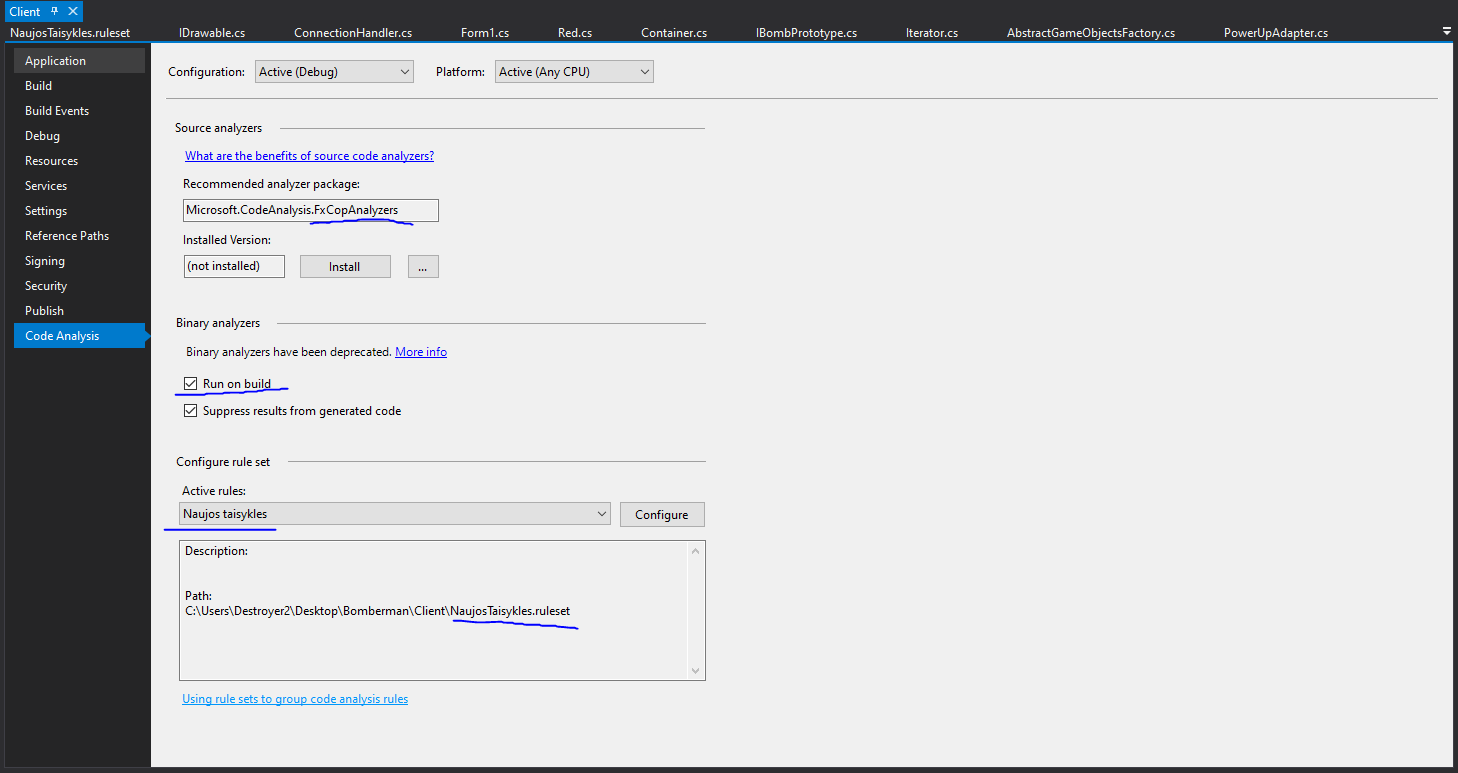
10 pav. Naujo kodo analizės taisyklių rinkinio sukūrimas

Antras žingsnis – pridėti norimas naujas taisykles į taisyklių rinkinį.

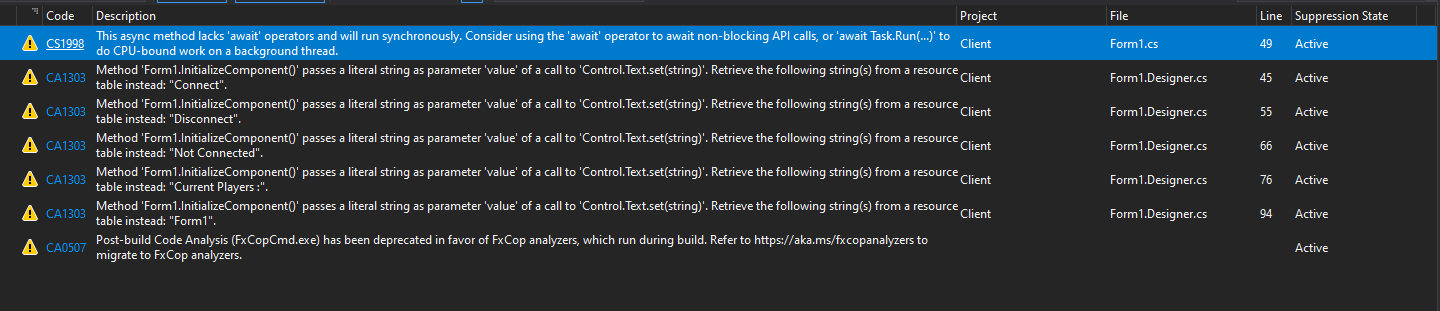


11 pav. Pasirenkamos norimos naujos taisyklės

Pasirinkus norimas taisykles, taisyklių rinkinys išsaugomas. Atidaromi projekto properties, pasirenkama “Code Analysis” skiltis. Skiltyje užžymime kad analizė būtų vykdoma kiekvieną kartą build’inant. Taip pat parenkamas naujai sukurtas taisyklių rinkinys.



12 pav. FxCop įrankio konfiguracija



13 pav. FxCop statinės analizės su naujomis taisyklėmis parinkimas.

Naujos taisyklės implementacija naujose FxCop versijose yra ganėtinai sudėtinga, dėl vienos priežasties: dėl ir taip labai plataus problemų ir klaidų spektro, gamintojas nusprenė kad kažkokių ranka kurtų taisyklių neprireiks, ko pasekoje yra visiškas dokumentacijos nebuvimas. Laboratorinio darbo metu bandžiau keliais skirtingais būdais, naudojantis senais (2011) pavyzdžiais sukurti naują taisyklę, tačiau iki galo padaryti kad veiktų neišėjo.

Taisyklės C# kodas:

using System;

using System.Collections.Generic;

using System.Text;

using Microsoft.Cci;

using Microsoft.FxCop.Sdk;

using Microsoft.FxCop.Sdk.Introspection;

public class DoNotKeepDBCredentialsInVariables : BaseMigrationIntrospectionRule

{

public DoNotKeepDBCredentialsInVariables() : base("DoNotKeepDBCredentialsInVariables")

{

}

public override ProblemCollection Check(Member member)

{

// Check Member Variables.

if (ContainsUserOrPasswordInName(member.Name.Name))

{

// Found an offending member variable.

base.Problems.Add(new Problem(base.GetResolution(member.Name.Name),

member.Name.Name));

}

// Look at local variables.

Method method = member as Method;

if (method == null)

return null;

LocalList list = null;

if (method.Instructions.Length > 0)

{

list = method.Instructions[0].Value as LocalList;

}

if (list != null)

{

for (int i = 0; i < list.Length; i++)

{

Local local = list[i];

if (ContainsUserOrPasswordInName(local.Name.Name))

{

// Found an offending local variable.

base.Problems.Add(new Problem(base.GetResolution(local.Name.Name),

local.Name.Name));

}

}

}

return base.Problems;

}

private bool ContainsUserOrPasswordInName(string variableName)

{

variableName = variableName.ToUpper();

if (variableName.Contains("USER") || variableName.Contains("PASSWORD"))

{

return true;

}

else

{

return false;

}

}

}

Taisyklės XML failas:

<?xml version="1.0" encoding="utf-8"?>

<RuleSet Name="Naujos taisykles" Description=" " ToolsVersion="16.0">

<Rules AnalyzerId="NaujaTaisykle" RuleNamespace="NewNewTaisykle">

<Rule TypeName="DoNotKeepDBCredentialsInVariables"

Category="VBMigration" CheckId="AA1001">

<Name>

Do not store security credentials in variables

</Name>

<Description>

Do not store security credentials in module level or local variables

</Description>

<Url>

http://www.thescarms.com/

</Url>

<Resolution>

Do not store security credentials in module level or local variables. Variables with the

word 'user' or 'password' in their name have been found. It is not secure to store

credential information in variables.

</Resolution>

<MessageLevel Certainty="75">

Warning

</MessageLevel>

<FixCategories>

NonBreaking

</FixCategories>

<Owner />

</Rule>

</Rules>

</RuleSet>

# Išvados

Laboratorinio darbo metu buvo ištirtas visas programos kodas, surastos probleminės vietos, potencialios klaidos, trūkumai ir stiliaus bei skaitomumo problematiškos vietos. Naudojant Resharper static analysis įrankį buvo atlikta kodo analizė, surasta virš 90 skirtingų problemų. Nors problemų kiekis pakankamai nemažas, tačiau didžioji dauguma jų nėra kritinės, o labiau kodo skaitomumą ir panaudamumą įtakojančios problemos. Taip pat buvo susipažinta su FxCop statinės analizės įrankiu, sukurtas naujas taisyklių rinkinys su naujomis taisyklėmis. Nors naudojantis FxCop iki galo naujos taisyklės implementuoti nepavyko, tačiau bandydamas tai padaryti pamačiau kaip tai maždaug darosi.

# Šaltiniai

<http://www.thescarms.com/dotnet/FxCop10.aspx>

<https://stackoverflow.com/questions/14425346/how-to-effectively-do-unit-test-and-static-code-analysis-on-net-codes>

<https://blogs.msdn.microsoft.com/codeanalysis/2010/03/26/how-to-write-custom-static-code-analysis-rules-and-integrate-them-into-visual-studio-2010/>

<https://docs.microsoft.com/en-us/visualstudio/code-quality/walkthrough-analyzing-managed-code-for-code-defects?view=vs-2019>

<https://docs.microsoft.com/en-us/visualstudio/code-quality/roslyn-analyzers-overview?view=vs-2019#roslyn-analyzers-vs-static-code-analysis>

<https://stackoverflow.com/questions/36982618/is-fxcop-dead-can-it-be-used-with-vs2015>

<https://docs.microsoft.com/en-us/visualstudio/code-quality/analyzer-rule-sets?view=vs-2019>

<https://docs.microsoft.com/en-us/visualstudio/code-quality/working-in-the-code-analysis-rule-set-editor?view=vs-2019>

<https://docs.microsoft.com/en-gb/visualstudio/code-quality/walkthrough-analyzing-managed-code-for-code-defects?view=vs-2019>