İSTANBUL TEKNİK ÜNİVERSİTESİ
BİLGİSAYAR VE BİLİŞİM FAKÜLTESİ

Kusurlu Kodların Belirlenmesi

Bitirme Ödevi

Murat Sekin
040040243

Bölüm : Bilgisayar Mühendisliği
Anabilim Dalı: Bilgisayar Bilimleri

Danışman : Doç. Dr. Feza BUZLUCA

Şubat 2016

1 PROJENİN TANIMI VE PLANI

Proje kapsamında nesne yönemli kaynak kodlar içerisindeki kusurlu kodların bulunması amacıyla bir yazılım geliştirilecektir. (Detecting bad codes in object-oriented programs) Böylece yazılım kalitesinin arttırılması amaçlanmaktadır.

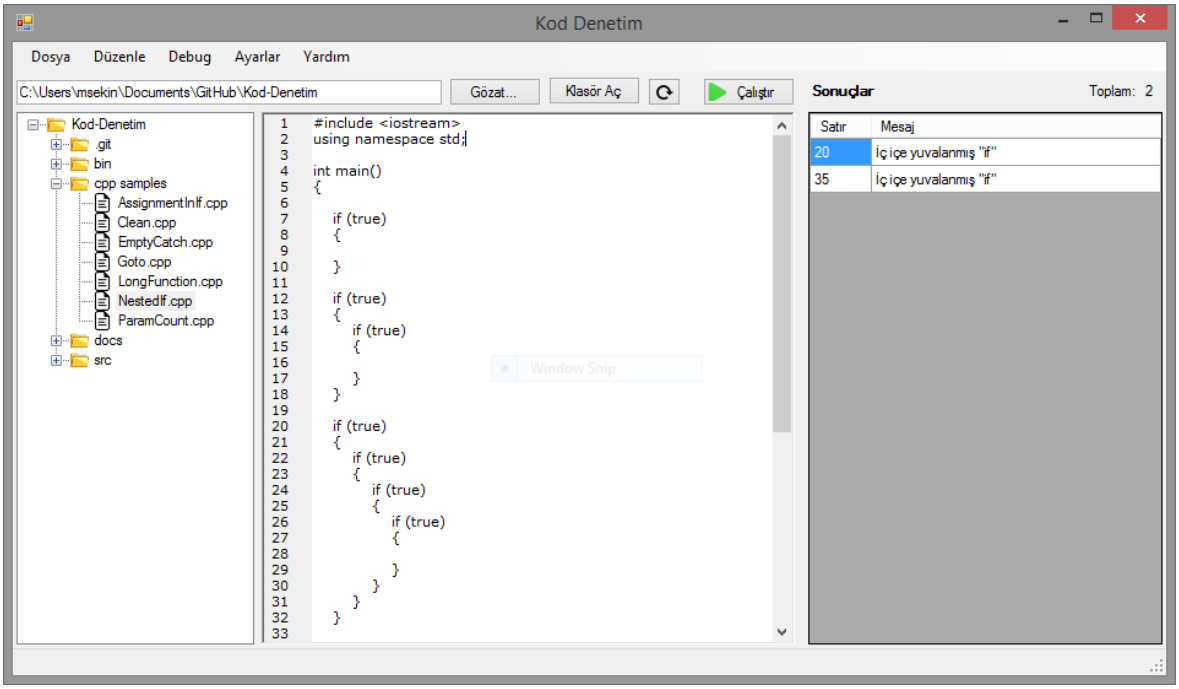
Kusurlu kodlar programların derlenmesini ve çalışmasını engellemez.*"Code smell*" olarak da bilinir. [1] Kötü uygulamalar ("Bad practices") bu kategoriye girer. Bu yönüyle syntax hataları ve buglardan ayrılır. Ancak programların işlevini olumsuz etkilemese de kaynak kodun okunmasını, anlaşılmasını, bakımını ve geliştirilmesini zorlaştırır. Maliyet (para, zaman) artışına neden olabilir. Günümüze kadar gelişitirilmiş yazılım standartlarına uyulmaması kusurlu kodların oluşmasının en önemli nedenlerinden birisidir. [2] Kusurlu kod örnekleri için EK 1 Kusurlu Kod Listesine bakılabilir.

Kusurlu kodların tespit edilmesi süreci statik kod analizi olarak adlandırılır. Statik kod analizi kaynak kodun derlenmeden otomasyon programları tarafından incelenmesidir. Günümüzde bu işlem için yazılmış hazır birçok yazılım mevcuttur.

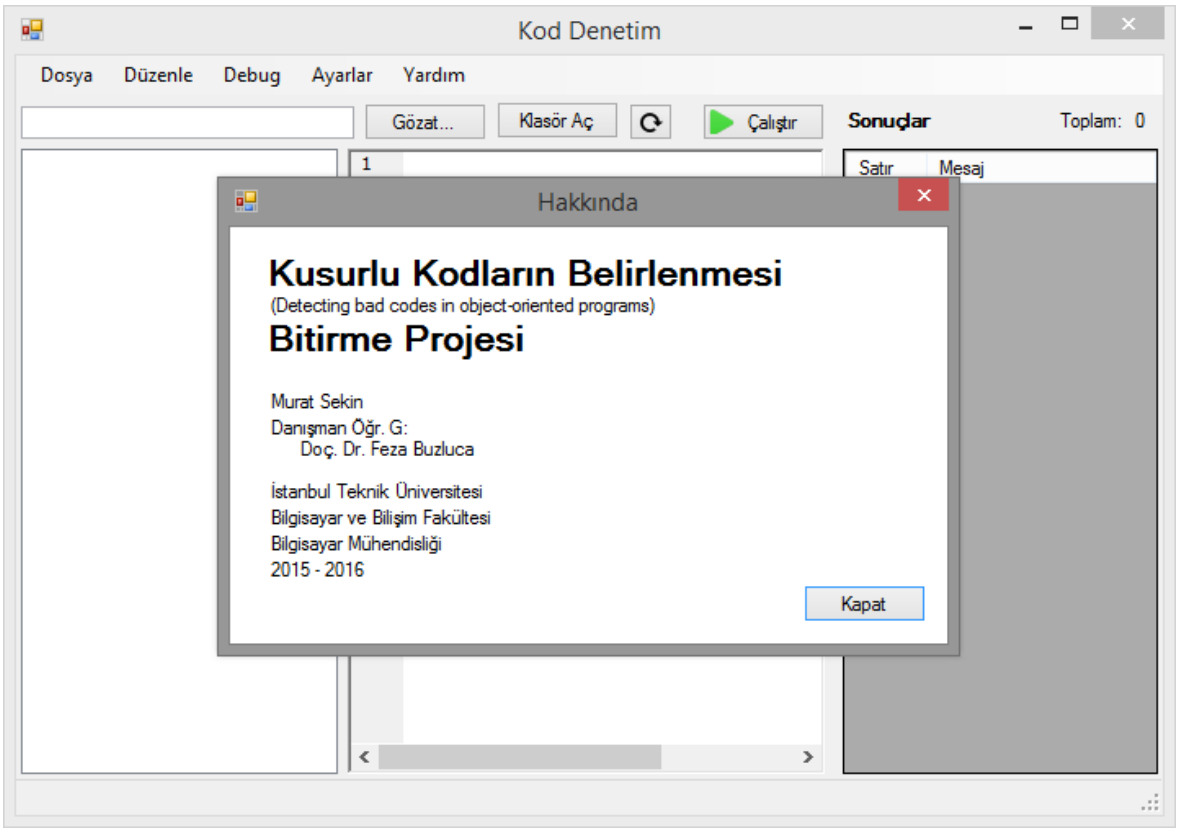
Hedef dil olarak C++ programlama dili kullanılmıştır. C++ programlama dili ile yazılmış kaynak kodlar analiz edilecek ve belirlenen kriterlere uymayan ve kusurlu kod olarak adlandırılan kısımlar tespit edilecektir. Hedef dil olarak C++'ın seçilmesinin nedeni orta seviye bir programlama dili olması ve bellek yönetimini programcılara bırakmasıdır. Bu da yazılım geliştiricilerin daha kolay hata yapmasına neden olmaktadır. Örneğin C#'da garbage collection sistemi bulunmaktadır.

2 TASARIM, GERÇEKLEME VE TEST

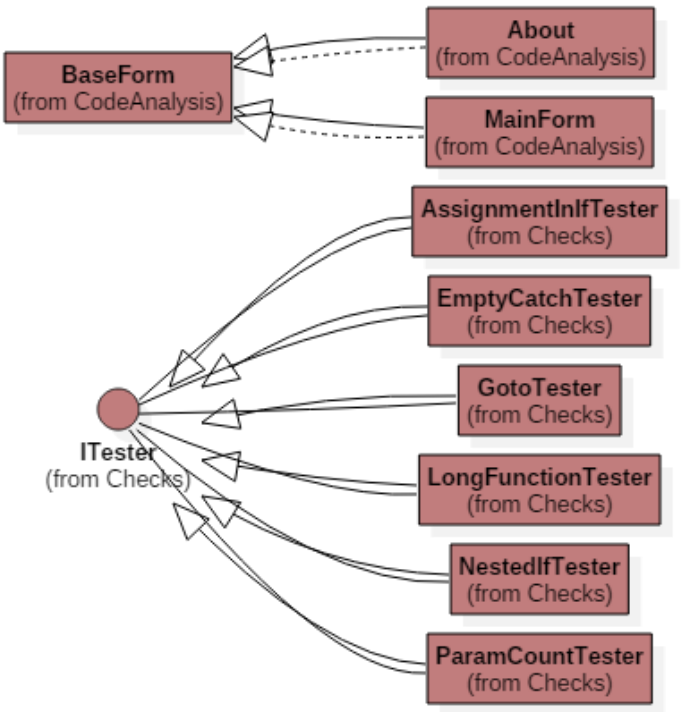
İlk olarak Kusurlu kodların araştırması yapılmıştır. Bu süreç sonunda EK 1 Kusurlu Kodlar Listesi oluşturulmuştur.
C++ kodunun analiz edilebilmesi için öncelikle parse edilip daha sonra anlamlandırılması gerekmektedir. Ancak bu dilin karmaşıklığı nedeniyle yeni bir parser geliştirmenin çok maliyetli olacağından dolayı hazır bir parser kullanılmıştır. ANTLR bir parser generatordır. Kendisine bir grammar verildiğinde bu grammare ait parserı oluşturur. [3] Kullandığımız grammarde C++ keywordleri ve kuralları vardır.
Üçüncü aşamada kodlamaya geçilmiştir. Nihai uygulama .NET Framework üzerinde çalışacaktır. Uygulama C# programlama dili kullanarak yazılmaktadır. Microsoft Visual Studio 2013 IDEsi kullanılmaktadır.



Şekil 1: Ana ekran



Şekil 2: Hakkında



Şekil 3: Type hierarch

3 KAYNAKLAR

[1] Wikipedia, Code Smell
https://en.wikipedia.org/wiki/Code_smell
[2] Feza Buzluca, "BLG 625 - Yazılım Tasarımı Kalitesi Ders Notları", 2014,
http://ninova.itu.edu.tr/tr/dersler/fen-bilimleri-enstitusu/2716/blg-625/ekkaynaklar?g403200
[3] ANTLR
http://www.antlr.org/, 2014

4 EKLER

EK 1. Kusurlu Kodlar Listesi

1. İsimlendirme kurallarına uyulmaması. (bkz. Naming convetions, Style guidlines)
2. Yorum ve açıklama yazılmaması. (bkz. Comments)
3. Büyük sınıflar. Bir sınıf bir nesneyi betimlemeli, sadece bir sorumluluğu olamalıdır. Farklı işler yapmamalıdır. (bkz. Single responsibility principle)
4. Veri saklayan sınıflar. Herhangi bir işi yoktur. Çok fazla alan (fields) içerir. (bkz. Data class)
5. Bir sınıfın işiyle ilgili çok az şey yapması. (bkz. Lazy class)
6. Sınıf içerisinde belli durumlarda kullanılan alanlar. (bkz. Temporary field)
7. Alt sınıf taban sınıfın veri ve methodlarının hepsini kullanmıyor. (bkz. Refused bequest)
8. Bir sınıfta değişiklik yapıldığında, sınıfın ilgisiz yerlerinde de değişiklik yapılması. (bkz. Divergent change)
9. Bir sınıfta değişiklik yapıldığında, birçok farklı sınıfta da değişiklik yapılması. (bkz. Shotgun surgery)
10. Bir sınıfın başka bir sınıfın üyelerini aşırı kullanması. (bkz. Feature envy)
11. Bir sınıfın tüm işlerini başka sınıflara yaptırması. (bkz. Middle man)
12. Message chains

```
A a;
a.b().c().d().e().f().g().h().i();
```

13. Inappropriate Intimacy.
14. Primitive Obsession.
15. Data Clumps.
16. Alternative Classes with Different Interfaces.
17. Parallel Inheritance Hierarchies.
18. Incomplete Library Class.
19. Speculative Generality.
20. Combinatorial Explosion.
21. Indecent Exposure.
22. Downcasting.
23. Cyclomatic complexity.
24. Üst sınıf, alt sınıfa bağlı.
25. Uzun fonksiyon. Her fonksiyon sadece tek bir iş yapmalıdır. Fonksiyonların kısa olması okuma ve anlaşılmayı kolaylaştırır.
26. Methodların fazla parametreye sahip olması.
27. Çok sayıda kod bloğunun iç içe yuvalanması.

```
if (true)
{
	if (true)
	{
		if (true)
		{
			if (true)
			{
				if (true)
```

```
                        {
                                // ...
                        }
                }
        }
    }
}
```

28. Hardcoded sayı ve kelimeler. (bkz. Magic numbers)
29. Kod tekrarı (bkz. Duplicated code)
30. Kullanılmayan değişkenler.
31. Kullanılmayan method parametreleri.
32. Kullanılmayan methodlar, sınıflar. (bkz. Dead code)
33. “goto" deyimini kullanmak.

```
void f()
{
label:
        // do something

        goto label;
}
```

34. == (eşit operatörü) yerine = (atama operatörünü) kullanmak

```
if (i = 0)
{
        // do something
}
```

35. “switch” deyiminin “break” keywordünü içermemesi.

```
switch (i)
{
        case 1:
                break;
        case 2:
                // do something
        default:
                break;
}
```

36. void pointer.

```
int n;
float f;

void* ptr;
ptr = &n; // valid
ptr = &f; // valid
```

37. Erişilemeyen kod.

```
int f()
{
        // do something
```

```
	return 1;

	int i;
}
```

38. Macro yerine inline function, enum, const kullanılmalı.

39. Boş kod blokları. (if, else, for, while, do/while, switch, try, catch, finally)

```
if (condition)
{
}

if (condition)
{
		// do something
}
else
{
}

for (int i = 0; i < length; i++)
{
}
```

40. Methodların içerisinde parametrelere tekrardan değer atamak.

```
void f(int i)
{
	// do something
	i = 0;
}
```

41. Yerel bir değişkenin referansını döndürmek.

```
int& f()
{
	int i;
	return i;
}
```

42. Yanlış “;” (noktalı virgül) kullanımı

```
if (true);
	DoSomething();

while (true);
	DoSomething();

for (size_t i = 0; i < length; i++);
	DoSomething();

int i;;
DoSomething();;
;
```

43. Referans veya işaretçilerin tip dönüşümleri.

```
class A
{
	/* ... */
};

class B
{
	/* ... */
};

A * a = new A;
B * b = reinterpret_cast<B*>(a);
```

```
class A
{
	/* ... */
};

class B : A
{
	/* ... */
};

A a;
B& b = dynamic_cast<B&>(a);
```

44. Dinamik belleklerin serbest bırakılmaması, dosyaların kapatılmaması, vb.

```
int* i;
//do something
//delete i;

fstream file;
file.open("test.txt");
//do something
//file.close();
```

45. Exceptions (istisnai durum) yutulmamalı. Sadece işlenebilecekse yakalanmalı.
46. Spaghetti code.
47. Satırların uzun olması. Ekrana sığmaması. Okumayı zorlaştırır.
48. Değişkenler, fonksiyonlar, sınıflar anlamlı isimlere sahip olmalı.
49. Sonsuz döngü.