Yazılımda Kopyaların (Klonların) Bulunması (*Clone Detection*)

Klon Tanımı:

- Aynı veya belli bir kritere göre benzer olan kod parçaları yazılım klonları olarak adlandırılır.
- Bu bölümde ele alınan örnek çalışmada sadece kod klonları değil, **tasarım klonları** da (sınıflar ve ilişkilerinin benzerlikleri) ele alınmıştır.
- Klonlar tıpa tıp aynı olabileceği gibi kopyalandıkları yerde değiştirilmiş de olabilirler.
- Klonlar aynı proje (program, dosya) içinde yer alabileceği gibi farklı projeler arasında da bulunabilirler.

Klon Tipleri:

- **Tip I:** Boşluklar ve biçim dışında birbirinin aynısı olan kod parçalarıdır.
- **Tip II:** Sadece değişken ve fonksiyon isimleri gibi işlevsel olmayan sözcüklerin değiştirildiği kod parçalarıdır.
- **Tip III:** Ufak çapta ekleme çıkarma gibi anlamsal veya yapısal değişikliklerin yapıldığı kod parçalarıdır.
- **Tip IV:** Yapısal olarak farklı olmalarına rağmen anlamsal olarak aynı işi yapan kod parçalarıdır.

Tip I	Tip II	Tip III	Tip IV

int a, b;
b = 20;
a = b + 1;
	int a, b;
	b = 20;
	a = b + 1;
		int a, b;
		b = 20;
		a = b + 1;
			int recursive(**int** n)
			{
			if (n == 0) {
			return 1;}
			else {
			return n * f(n - 1);
			}
			}

int a, b;
b = 20;
a = b + 1;
	int x, y;
	x = 20;
	y = x + 1;
		int a, b;
		b = 20;
		int c = 0;
		c = a + 5;
		a = b + 1;
			int sequential(**int** n)
			{
			int c = 2;
			int R = 1;
			for(; c <=n; c++)
			R = R * c;
			return R;
			}

- **Tip V: Tasarım klonları** (Örnek çalışmada önerilmiştir.)
  Önceki klon tipleri kodlama ile ilgilidir. Bu tür ise tasarımla ilgilidir.
  Kodlar farklı, fakat tasarım modeli aynıdır.

Klonların nedenleri (kaynakları):

- **Tekrar kullanma (*reusability*):** Program geliştirenler başka (veya aynı) projede çalışan kod parçalarını yeniden kullanmak için kopyalarlar.
- **Eskiyi koruyarak uyarlama:** Programın bazı kısımları yeni koşullara göre (yeni bir cihaz veya işletim sistemi) güncellenirken eski kısımlar yok edilmeyebilir.
- **Takım içindeki gereksiz tekrarlar:** Program geliştirme takımındakiler birbirlerinden habersiz aynı işleri yapan kod parçaları yazabilirler.
- **Problem uzayına özgü ortak yapılar:** Yazılım takımının çalıştığı problem uzayında (ağ yazılımı, otomasyon yazılımı, grafik yazılımı) sık kullanılan yapılar her projede tekrar oluşturulmaktadır.
- **Tasarım kalıplarının kullanımı:** GoF gibi kalıpların kullanılması.
- **Güvenilirlik ve yedekleme amacıyla tekrarlar:** İnsan hayatını etkileyen kritik yazılımlarda (örneğin demiryolu kontrol sistemleri) iki ekip tarafından yazılan ve aynı işi yapan iki program aynı anda çalıştırılır.
- **Paralel çalışmayı sağlamak:** İki benzer kod benzer işleri aynı anda yapabilirler.
- **Lisans ihlalleri (aşırma):** Program veya tasarım fikri çalınmış ve güncellenmiştir.
- **Rastlantısal:** İki kod veya tasarım parçaları rastlantısal olarak benzerdir.

Klonların bulmanın yararları:

Tüm klonlar zararlı değildir.

Bazıları bilinçli olarak yaratılmaktadır. Örneğin bankacılık uygulamalarında yedekleme için veya paralel çalışmayı sağlamak için.

Yine de klonların bulunması, nedenlerinin incelenmesi ve gerekli görülenlerin yok edilmesi yazılım kalitesinin artmasını sağlamaktadır.

- Kütüphane oluşturma: Tekrar eden birimler kütüphanelerde toplanır.
  - Bakım maliyeti düşer.
  - Tekrar eden veya düzeltilmesi unutulan hatalar önlenir.
- Yazılımın anlaşılırlığının artması.
  - Kolonların yok edilmesine gerek duyulmasa bile nedenlerini incelemek ve varlıklarının farkında olmak takımın yazılım hakkındaki bilgisini arttırır.
- Problem uzayına (yazılım takımına) ait yaygın kalıpların ortaya çıkartılması ve gerek varsa iyileştirilmesi.
- Sık tekrarlanan hatalar (tasarım kusurları) varsa bunların giderilmesi
- Lisans ihlallerinin ortaya çıkartılması

Tasarım Klonların Bulunması

Örnek Çalışma:

Umut Tekin, Feza Buzluca "A Graph Mining Approach For Detecting Identical Design Structures in Object-Oriented Design Models", Science of Computer Programming, Elsevier, Vol. 95, Part 4, December 2014, pp. 406 - 425.

Çalışmanın dayandığı düşünce:

- Nesneye dayalı yazılımlarda sınıfların tek başına kopyalanması düşük bir olasılıktır.

  Bu tür yazılımlarda birden fazla sınıf işbirliği yaparak bir hizmeti (*service-oriented*) yerine getirirler.

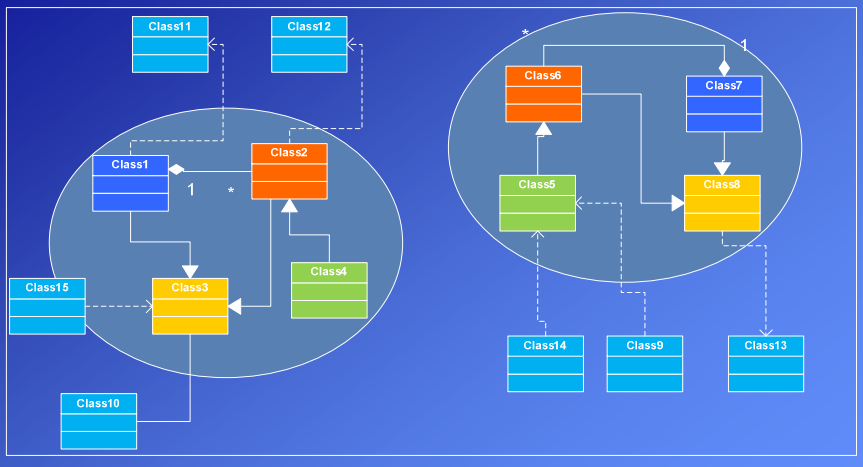
  Bu nedenle kopyalama sırasında birden fazla sınıf, tasarım fikri ile birlikte kopyalanır.
- Kaynak kod kopyaladıktan sonra çeşitli nedenlerle değiştirilir (düzeltme, geliştirme, kopyayı gizleme vb.).

  Ancak tasarım fikri (sınıf yapıları ve aralarındaki ilişkiler) çoğunlukla aynı kalır.

Örnek çalışmada kod tekrarları değil, nesneye dayalı tasarım yapılarının tekrarları araştırılmıştır.

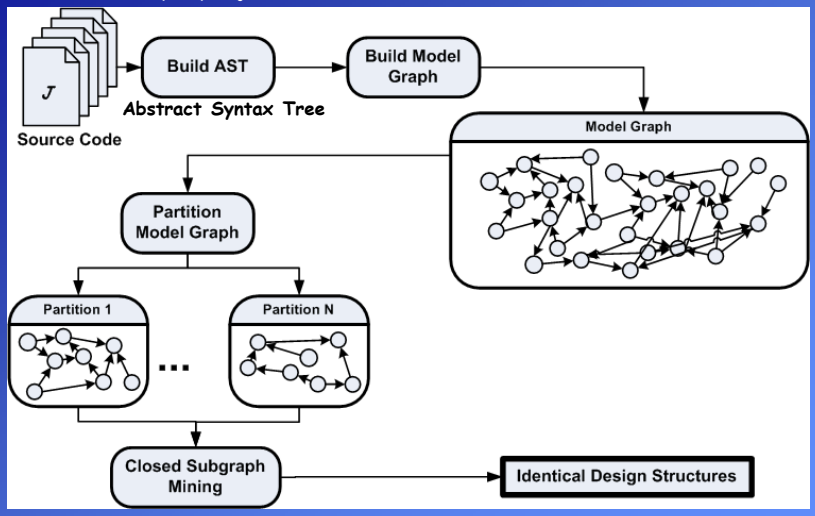
Tasarım tekrarları aynı projenin içinde olabildiği gibi iki farklı proje arasında da olabilir.

Örnek tasarım klonu:



Yöntemin genel yapısı:

Aynı proje içinde klon arama



AST: Abstract Syntax Tree

Yöntemin genel yapısı:

RL-Man ve NW-Man yöntemin uygulandığı iki örnek endüstriyel yazılım projesidir.

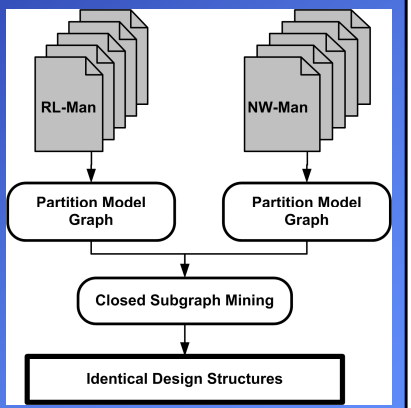
Tek projede olduğu gibi önce tasarım modeli grafları oluşturulur.

Graflar bölünür (parçalara ayrılır).

Grafların parçaları ortak bir kümede toplanır.

Bu parçalar içinde benzerlikler aranır.

Farklı iki proje içinde klon arama



Yazılımın tasarım modeli grafının oluşturulması:

Klonları belirleyebilmek için ilk adımda yazılımın tasarımı basit, yönlü ve etiketli bir graf şeklinde ifade edilir. **Tasarım → Graf (yönlü ve etiketli)**

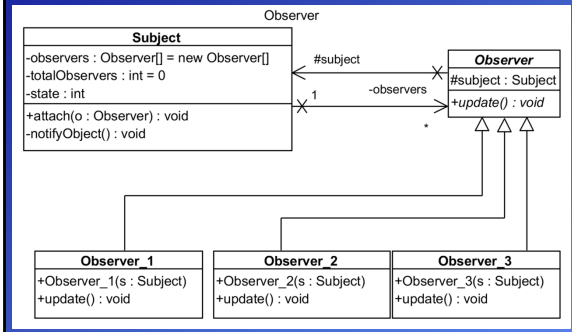
Tasarım grafında düğümler sınıfları, kenarlar da sınıflar arasındaki ilişkileri temsil eder. **Düğüm: Sınıf , Kenar : Sınıflar arası ilişki**

Etiketler ise düğümlerin ve ilişkilerin türünü gösterir.

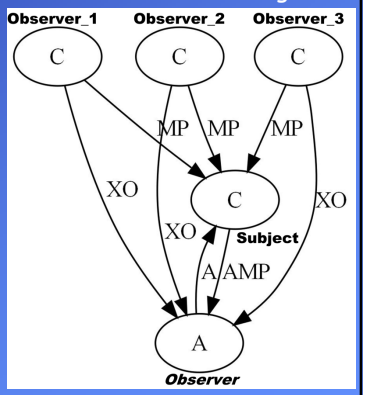
Vertex Label	Düğümler: Sınıf Tipleri (*Entity Type*)
C	Class
I	Interface
A	Abstract Class
T	Template Class
Edge Label	**Kenarlar: İlişkiler (*Relation Type (Edges are directed from A to B)*)**
X	Class A extends Class B
I	Class A implements Class B
A	Class A has the field type of Class B
T	Class A uses Class B in a generic type declaration
L	Class A has a method that defines a local variable with the type of Class B
P	Class A has a method that has an input parameter with the type of Class B
R	Class A has a method with the return type of Class B
M	Class A has a method that calls a method of Class B
F	Class A directly accesses fields of Class B without method calls
C	Class A creates objects of Class B
O	Class A overrides methods of Class B

Örnek: "Observer" GoF Kalıbının tasarım modeli grafı:

"Observer" UML Sınıf diyagramı



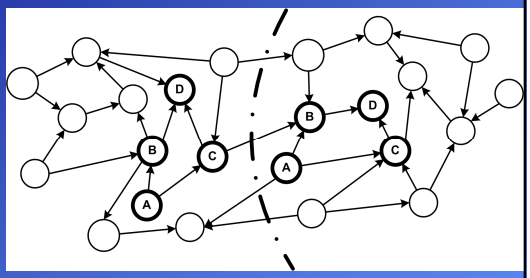
"Observer" tasarım grafı



Graf bölme (*Partitioning*) :

İki nedenden dolayı yazılımın tasarım grafı parçalara ayrılır (bölünür).

1. Yöntemin hızını arttırmak
2. Aynı yazılımın içinde de graflar arası karşılaştırma yaparak klon bulmak



Bölme sonucu graftaki bazı kenarlar (tasarımdaki ilişkiler) yok olur.

Kaybedilen kenarların klonları bulmayı engellemeyecek zayıf (önemli olmayan) ilişkiler olması istenir.

Nesneye dayalı tasarımda türetim/kalıtım ilişkisi tasarımın yapısını etkilediği için önemlidir.

Bu nedenle bölme işleminde **ağırlıklı graflar** kullanılmış ve kalıtımla ilgili "implements (I)" ve "extends (X)" ilişkilerine diğer ilişkilere göre daha büyük ağırlık verilmiştir.

Yapılan araştırmalar sonucu I ve X ilişkilerine ağırlık değeri olarak 2x|E|/|V| değerinin, diğer ilişkilere ise 1 değerinin atanmasının uygun olduğu görülmüştür.
|E|: Graftaki kenar sayısı, |V|: Graftaki düğüm sayısı

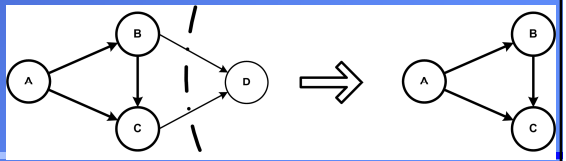
Graf bölme (*Partitioning*) (devamı):

Model grafının bölünmesinde parça sayısı (bir parçadaki düğüm/kenar sayısı) yöntemin başarımını etkilemektedir.

- Graf az sayıda parçaya bölünürse parçalar büyük olur (çok sayıda düğüm)
  - \+ Bölme sırasında az kenar kaybolur.
  - \+ Daha büyük klonlar (eş yapılar) bulunabilir.
  - \- Bazı klonlar aynı parçada kaldığı için gözden kaçabilir.
  - \- Yöntemin bellek tüketimi ve çalışma süresi artar.
- Graf çok sayıda parçaya bölünürse parçalar küçük olur.
  - \+ Yöntemin bellek tüketimi ve çalışma süresi kısalır.
  - \+ Klonların farklı parçalara düşme olasılığı artar.
  - \- Bölme sırasında daha çok kenar kaybolur.
  - \- Bazı klonlar bölündüğü için daha küçük klonlar (eş yapılar) bulunur.

Yapılan çalışmada bir parçada 20 düğüm olacak şekilde grafın bölünmesinin uygun olduğu görülmüştür.

Yandaki örnekte D düğümü klonun bir elemanı olduğu halde bölme sonucu başka parçada kalırsa bulunan klonun bir elemanı olduğu anlaşılamayacaktır.



Eş yapıdaki (*isomorphic*) alt grafların (*sub-graph*) bulunması

Model grafının bölünmesinden sonra oluşan parçalar arasında eş yapıdaki alt graflar aranır.

Eş yapıdaki (*isomorphic*) graflarda düğümler, kenarlar ve etiketler birebir aynıdır.

Bu işlem için CloseGraph algoritmasından yararlanılmıştır.

X. Yan and J. Han, CloseGraph: mining closed frequent graph patterns, in Proceedings of the ninth ACM SIGKDD International Conference on Knowledge discovery and data mining, 2003, pp. 286-295.

CloseGraph algoritmasının ParSeMiS kütüphanesindeki programı kullanılmıştır .

ParSeMiS: The Parallel and Sequential Mining Suite
http://www2.informatik.unierlangen.de/EN/research/ParSeMiS/

Sonuçlar Aynı proje içindeki aramalar:

Test ortamı: 2-GHz 4-CPU, 64-bit Linux, 32 GB RAM.

En az üç düğüm içeren yapılar dikkate alınmıştır. SMG: Software Model Graph

	Yari	Zest	JUnit	E-Quality	JFreeChart	ArgoUML
Number of Vertices in SMG	69	144	271	281	614	1571
Number of Edges in SMG	148	390	1,046	939	2,409	6,509
Line of Code (LOC)	4,727	14,432	15,247	21,662	56,925	117,903
Number of Graph Partitions	3	8	16	16	32	96
Max. Vertex Count in Partitions	23	21	18	18	20	17
Number of Identical Structures	15	37	61	39	307	745
Running Time in Seconds	0.8	0.75	76.56	17.13	1102.78	15126.92

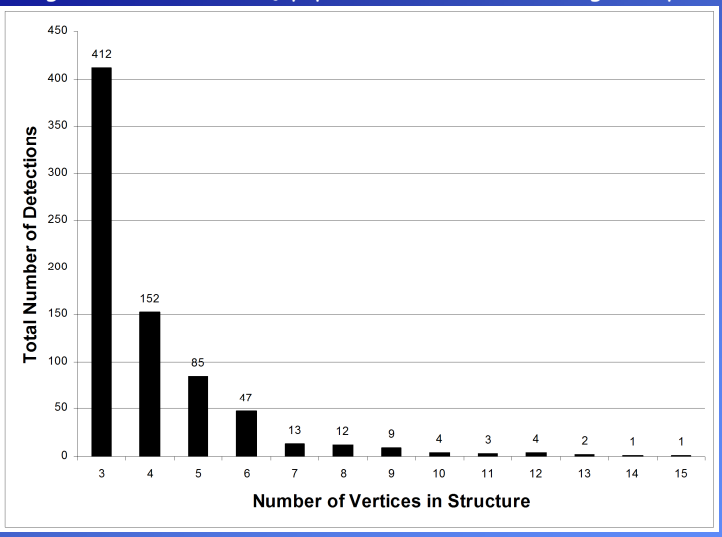
Bulunan eş yapıların tekrarlanma sayısı:

Frequency Value	Yari	Zest	JUnit	E-Quality	JFreeChart	ArgoUML
2	14	29	38	27	207	374
3	1	7	14	9	58	174
4	NA	1	5	3	26	92
5	NA	0	3	0	6	41
6	NA	0	1	0	4	27
7	NA	0	0	0	3	15
8	NA	0	0	0	2	10
9	NA	NA	0	0	1	7
10	NA	NA	0	0	0	4
11	NA	NA	0	0	0	1

ArgoUML üzerindeki çalışma yaklaşık 253 dakika sürmüş ve 25GB bellek kullanılmıştır.

Sonuçlar Aynı proje içindeki aramalar (devamı):

ArgoUML'de bulunan eş yapılardaki (klonlardaki) düğüm sayısı:



Çok sayıda düğüm içeren yapılar daha ilginç ve önemlidir.

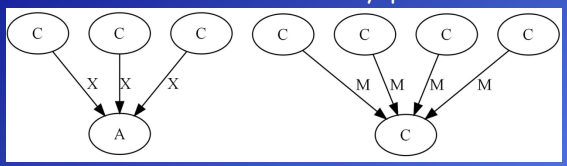
Az sayıda düğüm içeren yapılar genellikle nesneye dayalı programlamanın kullanıldığı her projede rastlanan standart yapılarıdır.

Büyük yapıları (düğüm sayısı beşten çok) özellikle incelemek gerekir.

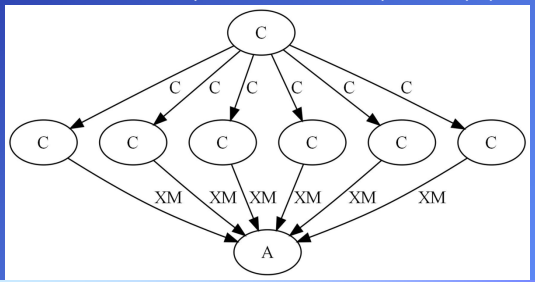
Sonuçlar Aynı proje içindeki aramalar (devamı):

Bulunan bazı eş yapılara (klonlar) örnekler:

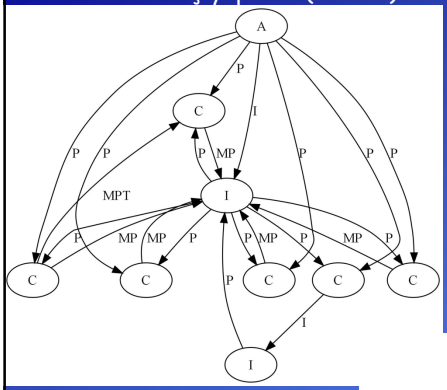
Sık rastlanan standart tasarım yapıları:



GoF fabrika (*factory*) kalıbına benzeyen bir yapı:

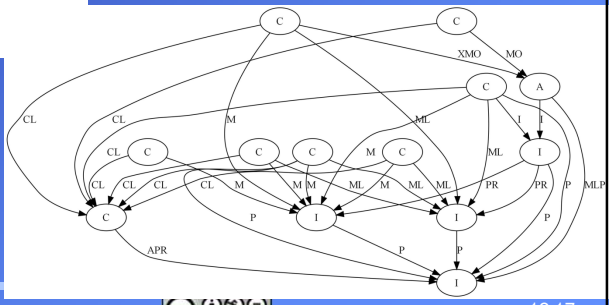


Bulunan bazı eş yapılara (klonlar) örnekler:



GoF Visitor kalıbı

JFreeChart projesinde bulunan karmaşık bir klon:



Sonuçlar Farklı projeler arasındaki aramalar:

- Aynı firma tarafından gerçekleştirilen iki endüstriyel proje incelenmiştir.
- RL-Man (*Radio Link Management*): iki yıl önce tamamlanmış ve sahada kullanılmaktadır.
- NW-Man (*Management software for new generation high-speed network security equipment*): Proje devam etmektedir.
- İki projenin yazılım mimarı aynı kişidir.
- Programlama takımlarında hem ortak hem de farklı elemanlar vardır.
- Projelerin hem kendi içindeki hem de aralarındaki klonlar belirlenmiştir.
- Proje elemanları ile birlikte çalışılıp bulunan klonların kaynak kodlar incelenmiş ve klonların nedenleri belirlenmiştir.

	RL-Man	NW-Man	Combined Set
Number of Vertices in SMG	332	547	879
Number of Edges in SMG	1446	2503	3949
Number of Graph Partitions	16	32	48
Maximum Vertex Count in the Partitions	23	20	23
Number of Identical Structures	34	55	107
Running Time in Seconds	34.23	1187.10	1402.34

Ortak graf kümesinde bulunan 107 eş yapı incelendiğinde iki proje arasında toplam 26 klon (düğüm sayısı üç ve daha fazla) belirlenmiştir.

Sonuçlar Farklı projeler arasındaki aramalar (devamı):

İki proje arasında bulunan 26 klonun kaynak kodları proje çalışanları ile birlikte incelenmiştir.

- Klonların 11 tanesinin nesneye dayalı sistemlerde yaygın olarak kullanılan yapılardır.
- Yedi tanesi tasarım kalıbıdır: Dört *factory*, iki *observer*, bir *visitor*.
- Geri kalan sekiz adet klonun önceki RL-Man projesinden yeni NW-Man projesine kopyalanan kodlar nedeniyle oluştuğu anlaşılmıştır.

Bu sekiz klonun kaynak kodları ayrıntılı olarak incelenmiştir.

- Yazılım takımları bu klonların bazılarının farkında olmakla beraber üç tane klonun kimse farkında değildir (Takım değiştiği için unutulmuş olabilir).
- Kaynak kodlar incelendiğinde şu anda çalışan RL-Man projesindeki klonlardan birinin kodlarında iyileştirmeler (2 adet) ve düzeltmeler (10 adet) yapıldığı ancak bunların NW-Man projesindeki yapıya uygulanmadığı anlaşılmıştır.
  Bu modülleri geliştiren kişi bir süre önce firmadan ayrılmış.
- Benzer bir durum diğer kopyalanan kodlarda da oluşmuştur.
  Bu klonlarda yeni projede düzeltmeler yapılmış ancak bunların eski projeden kopyalandığı unutulup düzeltmeler eski projeye de uygulanmamıştır.
  Bu modülleri geliştiren kişi hala projede çalıştığı halde bu durum gözden kaçmıştır.