|  |  |  |
| --- | --- | --- |
| Karadeniz_Teknik_Üniversitesi.gif | **T.C.**  **KARADENİZ TEKNİK ÜNİVERSİTESİ**  **OF TEKNOLOJİ FAKÜLTESİ**  **YAZILIM MÜHENDİSLİĞİ**  **İŞYERİ UYGULAMASI RAPORU** | Karadeniz_Teknik_Üniversitesi.gif |

|  |  |  |
| --- | --- | --- |
| **SONUÇ RAPORU**  **AD SOYAD**  **NO** |  | images.jpg |

|  |  |  |
| --- | --- | --- |
| **ÖĞRENCİ** | **Adı Soyadı** |  |
| **Öğrenci Numarası** |  |
| **T.C. Kimlik No** |  |
| **Bölümü** | Yazılım Mühendisliği |
| **Yarıyılı** | 20.. - 20.. Eğitim ve Öğretim ……. Yarıyılı |
| **İşyeri Eğitimi Süresi** | .../…/20.. - …/…/20.. ( ……. Gün) |
| **DENETÇİ** | **Adı Soyadı** |  |
| **Unvanı** |  |
| **Bölümü** | Yazılım Mühendisliği |
| **Adresi** | KTÜ Of Teknoloji Fakültesi |
| **İli** | Of/TRABZON |
| **İŞYERİ**  **YETKİLİSİ** | **Adı Soyadı** |  |
| **İşyeri Yetkilisi - Unvanı** |  |
| **İşyeri Bölümü** |  |
| **İşyeri/Firma Adı** |  |
| **İşyeri Adresi** |  |
| **İşyeri İli** | TRABZON |

|  |  |  |
| --- | --- | --- |
| **İŞYERİ EĞİTİMİ KOMİSYON BAŞKANI** | **DENETÇİ ÖĞRETİM ÜYESİ** | **İŞYERİ YETKİLİSİ** |
| Dr. Eyüp GEDİKLİ |  |  |
| / /20.. | / /20.. | / /20.. |

|  |  |
| --- | --- |
| Karadeniz_Teknik_Üniversitesi.gif | **T.C.**  **KARADENİZ TEKNİK ÜNİVERSİTESİ**  **OF TEKNOLOJİ FAKÜLTESİ**  **YAZILIM MÜHENDİSLİĞİ BÖLÜMÜ**  **İŞYERİ UYGULAMASI HAFTALIK RAPOR** |
| Tarih | / / 20.. |
| Sayfa No |  |

|  |
| --- |
| **YAPILAN İŞ:** Eğitim seminerleri, HAVELSAN, çalıştığım bölüm ve proje tanıtımı |

|  |  |  |
| --- | --- | --- |
| **DENETÇİ** | | **İŞYERİ YETKİLİSİ** |
|  | |  |
| / /20.. |  | / /20.. |

**İş Sağlığı ve Güvenliği**

İSG kısaltmasıyla iş hayatında yoğun olarak kullanılan iş sağlığı ve güvenliği tanımını yapacak olursak. "işyerinde işin yürütülmesi sırasında çeşitli nedenlerden kaynaklanan sağlığa zarar verebilecek koşullardan korunmak amacıyla yapılan sistemli ve bilimsel çalışmalardır" diyebiliriz.

İş sağlığı ve güvenliği - İSG olgusu, Avrupa Birliği içerisinde, 1980’li yıllardan itibaren ağırlıkla ele alınmaya başlanmıştır. Özellikle, 1989 yılında çıkarılan 89/391/EEC sayılı İş Sağlığı ve Güvenliği Direktifi, iş sağlığı ve güvenliği alanında çerçeve direktif olarak kabul edilmiş ve daha sonra bu çerçeve direktife dayanarak çok sayıda bireysel direktif çıkarılmıştır. Ülkemizde 2012 yılında yürürlüğe giren 6331 sayılı İş Sağlığı ve Güvenliği Kanunu da Avrupa Birliği uyum süresince hayatımıza girmiştir.

İş sağlığı ve güvenliğinin çalışma amaçlarını sıralayacak olursak;

* **Çalışanları Güvenliğini Sağlamak:** İş sağlığı ve güvenliği çalışmalarının ana amacını oluşturur. Çalışanları iş kazaları ve meslek hastalıklarına karsı koruyarak ruh ve beden bütünlüklerinin sağlanması amaçlanmaktadır.
* **Üretim Güvenliğini Sağlamak:** Bir işyerinde üretim güvenliğinin sağlanması beraberinde verimin artması sonucunu doğuracağından özellikle ekonomik açıdan önemlidir.
* **İşletme Güvenliğini Sağlamak:** İşyerinde alınacak tedbirlerle, iş kazalarından veya güvensiz ve sağlıksız çalışma ortamından dolayı doğabilecek makine arızaları ve devre dışı kalmaları, patlama olayları, yangın gibi işletmeyi tehlikeye düşürebilecek durumlar ortadan kaldırılacağından işletme güvenliği sağlanmış olur.

Amaçlarından da anlaşılabileceği gibi iş sağlığı ve güvenliği çok yönlü bir çalışmayı gerektirir. Çalışmalar sırasında da tıp, fizik, kimya, matematik, istatistik, iktisat, sosyoloji, psikoloji ve ergonomi gibi bilimlerden yararlanılır. Çalışma alanı ise, tüm iş kollarını kapsar. Doğal olarak da her iş kolunda bazı farklılıklar gösterir. İş sağlığı ve iş güvenliği konusunda yapılan tüm çalışmalara rağmen, iş kazaları ve meslek hastalıklarının istenilen düzeye indirgenemediği tespit edilmiştir. Meslek hastalıkları yeterli düzeyde tespit edilememekte ancak iş kazaları büyük oranda tespit edilebilmektedir. Meydana gelen iş kazaları sonucu kesikler, yaralanmalar, elektrik çarpması, zehirlenmeler, cilt hastalıkları, kırıklar, organ kayıpları, ölümler ve toplu ölümler şeklinde karşımıza çıkmaktadır. İş sağlığı ve iş güvenliğini tek cümleyle özetleyecek olursak; “insan merkezli yaklaşımın adıdır” demek uygun olacaktır.

**HAVELSAN**

HAVELSAN 1982 yılında ülkemizin teknolojik bağımsızlık mücadelesinde, milletimizin bağışlarıyla kurulan ve milli savunma sanayinin kilometre taşları arasında sayılan bir yapıdır.

TSKGV şirketi olan HAVELSAN, kuruluş değerlerini asla unutmadan, geçmişinden aldığı güç, geleceğe olan güven ve ülkesine olan bağlılığı ile 1800’ü aşkın personeli ile yüksek katma değere sahip, teknoloji-yoğun ve görev-kritik ürün, çözüm ve sistemleri hayata geçiriyor.

|  |  |
| --- | --- |
| Karadeniz_Teknik_Üniversitesi.gif | **T.C.**  **KARADENİZ TEKNİK ÜNİVERSİTESİ**  **OF TEKNOLOJİ FAKÜLTESİ**  **YAZILIM MÜHENDİSLİĞİ BÖLÜMÜ**  **İŞYERİ UYGULAMASI HAFTALIK RAPOR** |
| Tarih | / / 20.. |
| Sayfa No |  |

HAVELSAN, kuruluşundan bugüne geçen 37 yıl içinde sadece Hava Kuvvetlerimize bakım hizmeti veren bir küçük işletme durumundan, kamu ve özel sektöre savunma ve bilişim alanında hizmet sağlayan dünya pazarında da söz sahibi olma adımını sağlam şekilde atan bir şirket haline gelmiştir.

Türkiye ve ABD’deki toplam altı yerleşkesi ile bir misyon şirketi olarak faaliyet gösteriyor. Ulusal ve uluslararası pazarlarda birçok referans projeye imza atan HAVELSAN, yenilikçi ve sürdürülebilir stratejileri ile belirlediği faaliyet alanlarında güvenilir, sürdürülebilir, yüksek teknoloji ürün ve hizmet sağlayıcısı konumunda.

Yazılım yoğun sistemler konusunda faaliyet gösteren HAVELSAN:

* Komuta Kontrol ve Savunma Teknolojileri
* Eğitim ve Simülasyon Teknolojileri
* Bilgi ve İletişim Teknolojileri
* Ülke Güvenliği ve Siber Güvenlik Çözümleri

alanlarında, geniş bir yelpazede özgün ürün ve sistemlerden oluşan çözümlere sahip. HAVELSAN, bu çözümlerini, hem Türk Silahlı Kuvvetleri (TSK)’nin hem de uluslararası müşterilerinin hizmetine sunuyor.

**HAVELSAN KKST**

Komuta Kontrol ve Savunma Teknolojileri (KKST) faaliyetlerini Genel Müdür Yardımcılığı bünyesinde İstanbul ve Ankara’daki Merkez binalarında sürdürmektedir. KKST Genel Müdür Yardımcılığı;

* Havadan Erken İhbar ve Deniz Karakol Uçaklarının,
* Su üstü ve Su altı Deniz Platformlarının,
* Stratejik, Taktik ve Operasyon el Seviyede Karada Konuşlu Karargâh ve Birliklerin

görev sistemi yazımlarını, savaş yönetim sistemlerini ve komuta kontrol ihtiyaçlarını karşılamaktadır. Bu sistemlerin üretim ve entegrasyonu alanında liderliğini devam ettirmektedir.

KKST Genel Müdür Yardımcılığı, ana faaliyet alanını teşkil etmek üzere temel olarak aşağıdaki başlıklarda faaliyet göstermektedir

* Komuta Kontrol Bilgi Sistemleri
* Havadan Erken İhbar ve Havadan Komuta Kontrol Sistemleri
* Su üstü Komuta Kontrol ve Savaş Yönetim Sistemleri
* Su altı Komuta Kontrol ve Savaş Yönetim Sistemleri
* Platform Veri Dağıtım Sistemleri
* Taktik Data Link Sistemleri
* Performansa Dayalı Lojistik

KKST bölümünde DOOB Tactical projesine dahil oldum. Projeye dahil olan mühendisler;

* Levent Tanın. Yüksek Bilgisayar Mühendisi. Doob Ürün Takim Lideri
* Muzaffer Öztürk. Bilgisayar Mühendisi
* Büşra Karabulut. Bilgisayar Mühendisi

|  |  |  |
| --- | --- | --- |
| **DENETÇİ** | | **İŞYERİ YETKİLİSİ** |
|  | |  |
| / /20.. |  | / /20.. |

|  |  |
| --- | --- |
| Karadeniz_Teknik_Üniversitesi.gif | **T.C.**  **KARADENİZ TEKNİK ÜNİVERSİTESİ**  **OF TEKNOLOJİ FAKÜLTESİ**  **YAZILIM MÜHENDİSLİĞİ BÖLÜMÜ**  **İŞYERİ UYGULAMASI HAFTALIK RAPOR** |
| Tarih | / / 20.. |
| Sayfa No |  |

|  |  |  |
| --- | --- | --- |
| **DENETÇİ** | | **İŞYERİ YETKİLİSİ** |
|  | |  |
| / /20.. |  | / /20.. |

**DOOB Projesi**

HAVELSAN 30 yıldır Komuta Kontrol, Simülasyon, Eğitim ve Bilgi Sistemlerini Türk Silahlı Kuvvetleri ve tüm dünyada kullanılmak üzere geliştirmektedir. DOOB ürün ailesi, HAVELSAN’ın engin saha ve mühendislik tecrübesinin sonucunda geliştirilmiş ürünlerdir. DOOB ürün ailesi, Komuta ve Kontrol Bilgi Sistemlerinin temel niteliklerini içeren ve stratejik, operasyon el ve taktik komuta seviyelerinde kullanılmak üzere tasarlanmış üç farklı uygulamadan oluşur. DOOB uygulamalarının temel amacı tüm komuta seviyelerindeki durumsal farkındalığı arttırmaktır.

DOOB ürün ailesi kendi arasında entegre bir şekilde çalışır. Bu yüzden DOOB Tactical ve DOOB Mobile tarafından yayınlanan bütün bilgiler DOOB tarafından izlenebilir.

DOOB uygulaması farklı uluslararası ve milli standartlar vasıtasıyla farklı kaynaklardan toplandığı bilgileri entegre edip, Cari Harekât Takibi Modülü aracılığıyla bu bilgileri birleştirilmiş bir ara yüz sunarak durumsal farkındalığı arttırır. Bunun yanında, farklı fonksiyonel alanlarda hiyerarşik ve eş zamanlı planlama ve bu planların icra safhalarının farklı fonksiyon alanlarına uygun olarak takibini de destekler. DOOB milli ve NATO ihtiyaçlarını desteklediği gibi özel gereksinimlere kolayca uyarlanabilir ve genişletilebilir. DOOB ürün ailesi, HAVELSAN tarafından geliştirilmiş bir CBS çözümü olan TMAP ile desteklenmektedir.

DOOB web tabanlı bir uygulamadır ve modern web tarayıcısı haricinde, istemci makinesine herhangi bir kurulum gerektirmeden çalıştırılabilir. Kullanım uzmanları, DOOB ürünlerinin tasarım ve geliştirilmesi safhalarında yer aldıkları için her üç ürünün de kullanımı modern web uygulamalarına benzer, kolay ve hızlıdır. DOOB modüler bir uygulamadır ve farklı modülleri ayrı olarak kurulabilir. Bu nedenle kurulumu, bakımı ve yönetimi kolaydır.

DOOB Tactical uygulaması komuta yerlerinde taktik seviye komutanlar tarafından kullanılmak üzere geliştirilmiştir. DOOB Tactical ile taktik kullanıcılar DOOB tarafından üretilen planların basit halini alabilirler. Bununla birlikte kendi planlarını yapabilir ve cari harekâtı takip edebilirler. Taktik kullanıcılar semboloji standartlarına uygun olarak genel durumsal farkındalığa bilgi girişi yaparak katkı sağlayabilir ve bu bilgiyi DOOB ve DOOB Mobile ile paylaşabilirler. DOOB Tactical, DOOB Mobile ile entegre olduğu için kendi birliğine ait DOOB Mobile kullanıcılarını izleyebilir.

SmartOps olarak da bilinen DOOB Mobile uygulaması, erler için geliştirilmiş ve bu seviyedeki komuta kontrol faaliyetlerini gerçekleştirmeyi hedefleyen taktik seviye bir uygulamadır. Android işletim sistemi tabanlı mobil uygulamadır. Güvenli VPN üzerinden temel olarak konum ve kalp atışı bilgilerini paylaşır, kendi takımındaki diğer erlerin aynı şekilde durumlarını takip eder ve bağlı olduğu DOOB Tactical komutanının konumunu görebilir. Bunun yanında canlı video yayımı, mesajlaşma, ses, fotoğraf ve diğer doküman paylaşım özellikleri de mevcuttur.

DOOB ürün ailesi kendi arasında entegre bir şekilde çalışır. Bu yüzden DOOB Tactical ve DOOB Mobile tarafından yayınlanan bütün bilgiler DOOB tarafından izlenebilir.

DOOB web tabanlı ve dağıtık mimariye sahip bir uygulama olduğu için tek bir sunucu üzerinden birçok kullanıcıya destek verebileceği gibi dağıtık mimari sayesinde bütün bir orduyu destekleyebilir. Bu şekilde kolay bir şekilde kurulup, kolay bir şekilde yönetilebilir ve genişletilebilir. DOOB ürününün geliştirilmesinde modern ve açık kaynak kodlu web ve uygulama çatıları kullanılmış ve herhangi bir şekilde rahat veya paralı bir kütüphane kullanılmamıştır. DOOB kullanıcının aşina olduğu diğer web uygulamalarına yakın bir kullanım kolaylığı sağlar.

|  |  |
| --- | --- |
| Karadeniz_Teknik_Üniversitesi.gif | **T.C.**  **KARADENİZ TEKNİK ÜNİVERSİTESİ**  **OF TEKNOLOJİ FAKÜLTESİ**  **YAZILIM MÜHENDİSLİĞİ BÖLÜMÜ**  **İŞYERİ UYGULAMASI HAFTALIK RAPOR** |
| Tarih | / / 20.. |
| Sayfa No |  |

**DOOB Tactical Projesi**

DOOB Tactical uygulaması komuta yerlerinde taktik seviye komutanlar tarafından kullanılmak üzere geliştirilmiştir. DOOB Tactical ile taktik kullanıcılar DOOB tarafından üretilen planların basit halini alabilirler. Bununla birlikte kendi planlarını yapabilir ve cari harekâtı takip edebilirler. Taktik kullanıcılar semboloji standartlarına uygun olarak genel durumsal farkındalığa bilgi girişi yaparak katkı sağlayabilir ve bu bilgiyi DOOB ve DOOB Mobile ile paylaşabilirler. DOOB Tactical, DOOB Mobile ile entegre olduğu için kendi birliğine ait DOOB Mobile kullanıcılarını izleyebilir.

Askeri birimler tablete kurulan yazılım ile haberleşir. Uygulama açıldığında karşınıza dünya haritası gelir. Kategorilere ayrılmış şekiller ile harita üzerinde düşman birimleri, kendinize birimlerinizi ve harekât alanları çizebilirsiniz. Aynı haberleşme ortamına bağlı cihazlar birbirlerini otomatik şekilde eklerler. Açılan rapor türlerine göre reel time olarak o anda açık olan tabletlere bilgiler göndermektedir. Verilerin gönderiminde NATO standarttı olan VMF kullanılmaktadır.

Proje hakkında yeterli sözel bilgi edinildi. Geliştirme yapılabilmesi için kuruma özel ağ olan intranetin açılması gerekmektedir. İntranetin açılmasını beklerken yazılımın iç tasarımını ilgilendiren alanlarda bilgilerimi tazelemek için aşağıda verilen bilgileri araştırmaya başladım.

**Esnemezlik ( Rijidite )** Bir yazılım sistemi zaman içinde değişen veya yeni gündeme gelen gereksinimleri karşılayabilmelidir. Eğer bir değişim ya da gelişim yapılması isteniyorsa ve sistem bu duruma aşırı direnç gösteriyorsa bu rijit bir sistemdir. İyi bir tasarımda değişime karşı olan direncin düşük olması istenir.

**Kırılganlık ( Fragility )** Amaç sistemi her zaman olabildiğince esnek tasarlamak ve muhtemel değişikliklere daha az direnç gösteren dolayısıyla daha az kırılgan yazılımlar geliştirebilmektir. Aksi takdirde yapılacak bir değişiklik sistemde domino etkisi yaratıp zincirleme şekilde birden çok şeyin değişmesini gerektirebilir.

**Yeniden Kullanılabilirlik** Yazılım sistemleri modüler şekilde tasarlanmalıdır. Bir projede kullanılmış olan herhangi bir bileşenin farklı bir projeye taşınıp orada da kullanılabilmesi olarak bilinir.

**Yapışkanlık ( Cohesion )** Bir kavramın tek bir yerden ulaşabilir olması. Alakalılık kavramıdır. Modül içerisindeki etkileşim derecesidir. Sınıflar arası ilişkinin gevşek bağlı olması istenirken, sınıflar içerisindeki yordamların ve veri alanlarının yapışkan/bağlantılı olması istenir. Bir sınıf Single Responsibility Principle çerçevesinde yalnızca tek bir görevi gerçekleştirmelidir. Eğer bir sınıf içerisinde birden fazla değişik iş yapan yordamlar ya da bağımsız veri alanları bulunuyorsa, bu iş grupları farklı sınıflar halinde ayrıştırılmalıdır. Bir sınıf içerisindeki yordamlar ortak veri alanlarını kullanmıyorlarsa ya da yaptıkları iş bakımından tamamen ayrık iseler, onları aynı kümenin içinde barındırmanın bir anlamı da bulunmamakta. Yapışkanlık derecesi düşük sınıflar modülerliği ve yazılım bakımını tehdit eden bir unsurdur. Yüksek kohezyon yazılım sisteminin esnekliğini arttırır, bakım ve yeniden kullanılabilirliği kolaylaştırır. Kohezyonun 7 türü vardır ve 1’den 7’ye doğru iyileşmektedir:

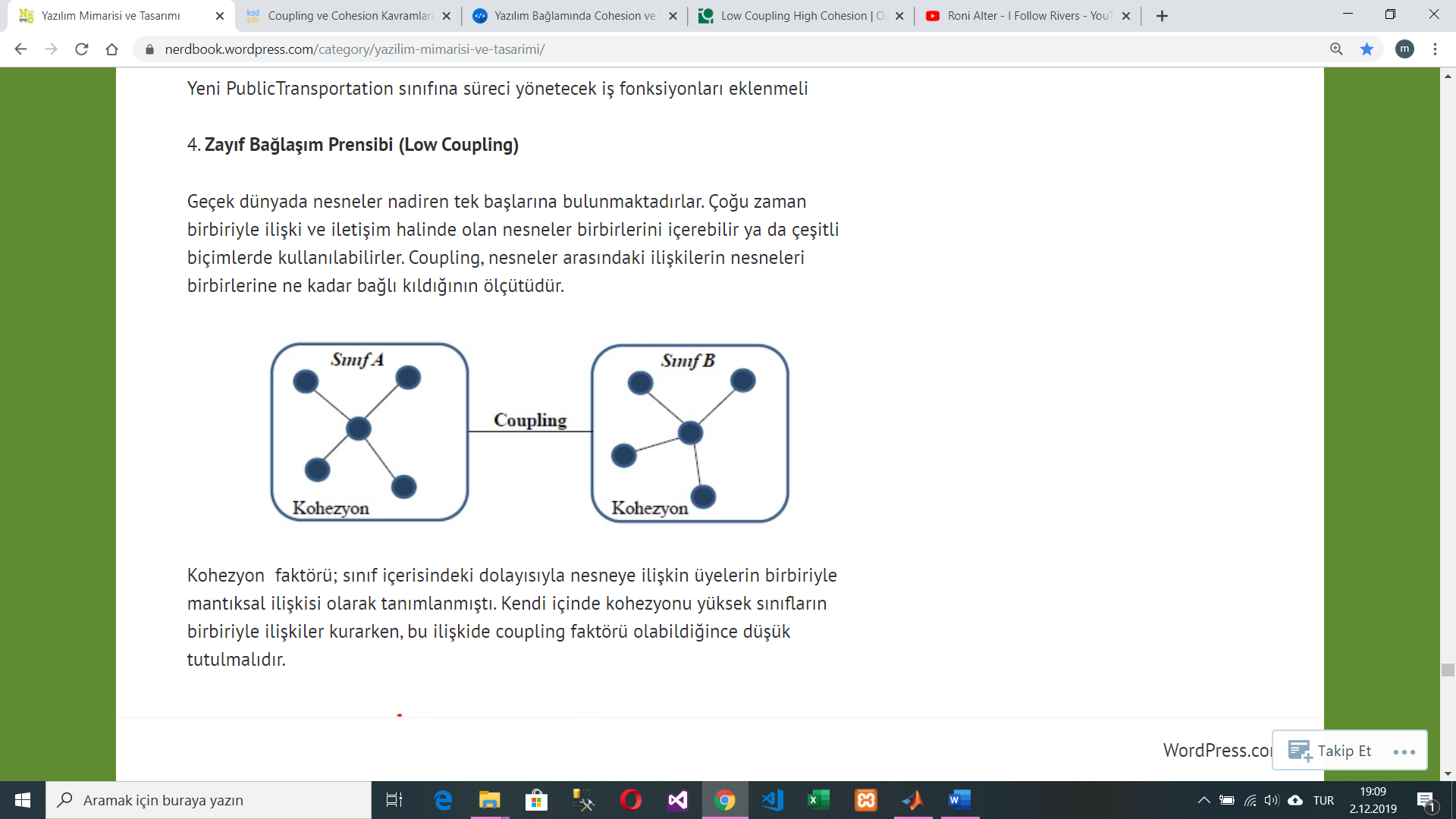
1. **Rastlantısal ( Coin Cidental )** En kötü kohezyon tipidir. Modüldeki öğelerin rastgele seçilmesi – birbirleri ile alakalı olmayan işlemleri içerme durumudur. Bu daha çok Utils, Helpers gibi isimlendirilen modüllerde görülmektedir**. Bakımı ve yeniden kullanımı zordur. Her bir alakasız kısım ayrı bir modül olarak ayrılmalıdır.**

|  |  |  |
| --- | --- | --- |
| **DENETÇİ** | | **İŞYERİ YETKİLİSİ** |
|  | |  |
| / /20.. |  | / /20.. |

|  |  |
| --- | --- |
| Karadeniz_Teknik_Üniversitesi.gif | **T.C.**  **KARADENİZ TEKNİK ÜNİVERSİTESİ**  **OF TEKNOLOJİ FAKÜLTESİ**  **YAZILIM MÜHENDİSLİĞİ BÖLÜMÜ**  **İŞYERİ UYGULAMASI HAFTALIK RAPOR** |
| Tarih | / / 20.. |
| Sayfa No |  |

1. **Mantıksal ( Logical ):** İf/switch yapısı ile birden fazla ve alakasız işlemin aynı modülde olmasıdır. Örneğin bütün veri tabanı işlemlerinin ya da bütün input girişlerine ilişkin işlemlerin aynı modüle konulması. Bakımı ve yeniden kullanımı zordur. Ekleme ve genişletme uzun uğraş ve yeni kodlar gerektirir.
2. **Geçici – Zamana Bağlı ( Temporal ):** Birden fazla işlemin birbiri ardına yapılması durumudur. Bazı durumlarda zorunludur. Yeniden kullanımı zordur.
3. **İşleme Bağlı ( Procedural ):** Birbiri ile ilişkili işlemlerin aynı modül içerisinde olmasıdır. Örnek olarak dosyanın varlığının kontrolü, dosyanın açılması, yazılacak içeriğin oluşturulması, verinin yazılması, dosyanın kapatılması gibi işlemlerden oluşan bir modül düşünülebilir. Modülde birden fazla işlem yapıldığından yeniden kullanımı zordur.
4. **İletişimsel ( Communication ):** Aynı veri üzerinden birden fazla kod bloğunun birden fazla işlem yapmasıdır. Örneğin bir nesnemiz var ve bu nesneye ilişkin bütün her şey bir arada. Kontroller, hesaplamalar, güncellemeler vb. Yeniden kullanımı zordur.
5. **Fonksiyonel ( Functional ):** Bir modül içinde tek bir işlem yapılmasıdır. Yeniden kullanılabilir. Bakım ve geliştirilmesi kolaydır.
6. **Bilgiye Dayalı ( Informational ): Tek bir veri üzerinde işlem yapar. Encapsulation yapılabilir. Yeniden kullanılabilir. Bakımı ve geliştirmesi kolaydır.**

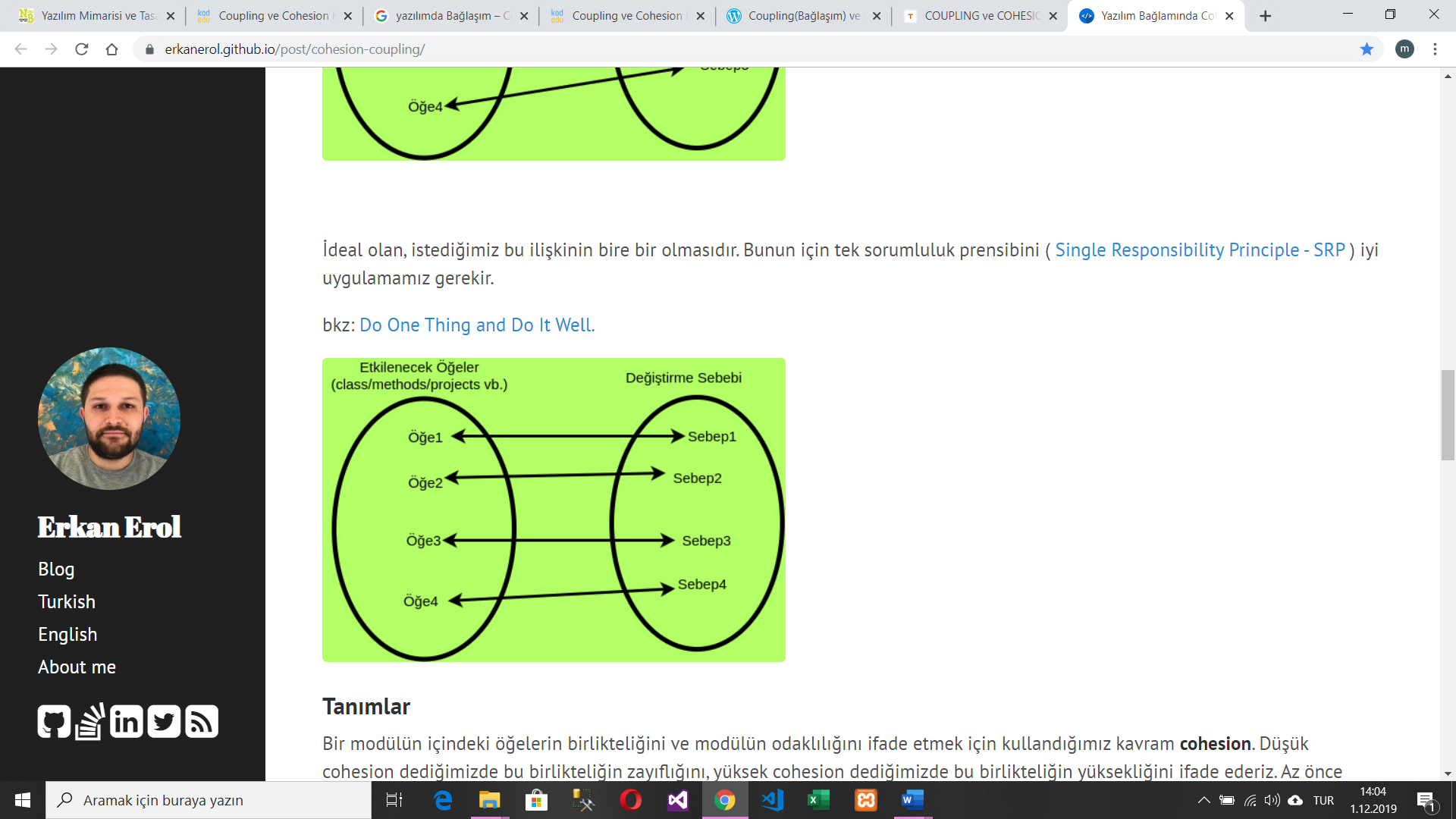
**Bağlaşım ( Coupling )** Birden fazla sınıf/nesne arasındaki ilişkiyi temsil etmektedir. Burada iki alt kavram daha çıkmaktadır. Gevşek bağlılık ve sıkı bağlılık. Nesne yönelik programlama dillerinde bir sınıfın diğerine erişmesi/içermesi Kalıtım ya da bir sınıfın diğer bir sınıfa bir referans ( bağ ) ile bağlanması yöntemleriyle gerçekleşiyor. Sıkı sıkıya bağlılık kavramında; İlişki içerisinde bulunan bir sınıf üzerinde yapılacak değişiklik, diğer bağlantılı sınıfları da etkileyebiliyor. Bu da modülerliği ve bakım maliyetlerini etkileyen bir unsur. Bu sebeple Yazılım Mühendisliğinde nesneler arasındaki ilişkinin sıkı olmaması yani gevşek bağlı olması istenir. Yani bir sınıf üzerinde yapılan değişikliğin diğer sınıflara sıçramaması istenir.

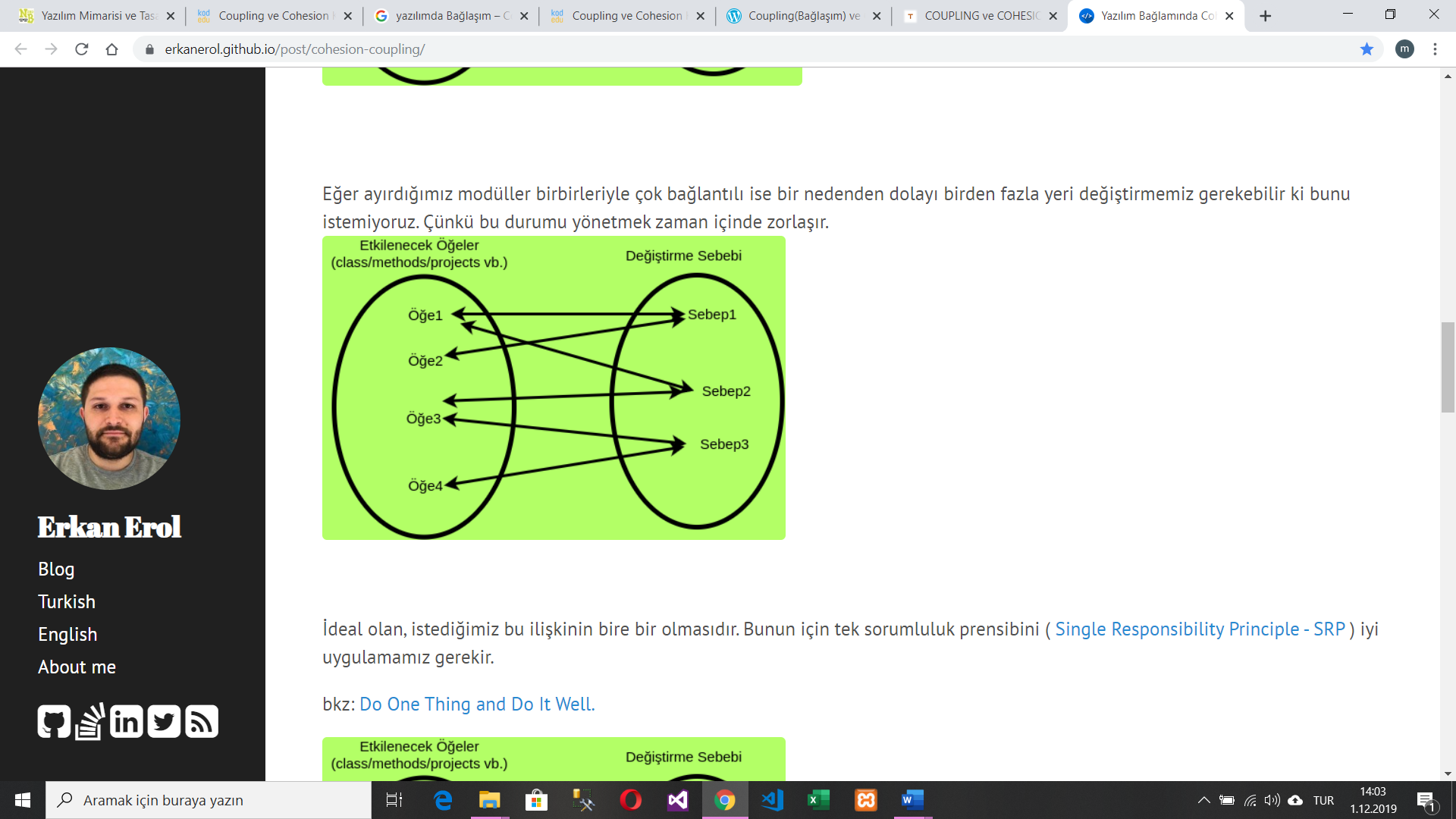


Coupling faktörünün 5 seviyesi vardır:

1. **Nil Coupling:** Teorik olarak en düşük dolayısıyla en iyi coupling düzeyidir. Zira bu seviyede bağımlılık söz konusu olmamaktadır. Diğer sınıflarla hiçbir ilgili olmayan, tek başlarına kullanılan sınıflar bu duruma örnek teşkil etmektedir. Dikkat edilecek husus boş sınıf olmamaları gerektiğidir. Sınıf yapmak için sınıf oluşturamayız.
2. **Export Coupling:** Herhangi bir sınıf başka bir sınıfa ortak bir arayüzle bağlıysa aralarında export coupling oluşur. Birçok durumda ulaşılmaya çalışılan ideal seviye export coupling seviyesidir. Bu seviyeden sonraki seviyeler yaratılmak istenen low coupling ilkesine zarar vermeye başlar.
3. **Overt Coupling:** Bir sınıf, başka bir sınıfa ilişkin üyeleri belli bir izin dahilinde kullanıyorsa aralarında overts coupling söz konusudur. Bazı sınıflarda yazılması zorunludur.
4. **Covert Coupling:** Bir sınıf, başka bir sınıfa herhangi bir izin vermeden arkadaşlık kurması durumudur.
5. **Surreptitious (Gizlice) Coupling**: Bir sınıf, başka bir sınıfın içsel detaylarının tümünü biliyorsa ve bunları kullanarak işlem gerçekleştiriyorsa bu sınıfların arasında surreptitious coupling oluşur. Tasarım açısından tehlikelidir. Bağımlılık, prensip gereğince az olması gerekirken bu seviyedeki coupling’de çok fazladır.

Özet olarak eğer ayırdığımız modüller birbirleriyle çok bağlantılı ise bir nedenden dolayı birden fazla yeri değiştirmemiz gerekebilir ki bunu istememekteyiz. Çünkü bu durumu yönetmek zaman içinde zorlaşacak ve maliyeti arttıracaktır. İdeal olan, istediğimiz bu ilişkinin bire bir olmasıdır. Bunun için tek sorumluluk prensibini iyi uygulamamız gerekir.





**=>**

SOLID (Single responsibility, Open-closed, Liskov substitution, Interface segregation ve Dependency inversion) yazılım tasarım prensipleri için kullanılan bir kısaltmadır. Yazılım yaparken SOLID uygulandığı taktirde bakımı ve geliştirilmesi kolay yazılım sistemleri oluşturmak mümkündür.

* **S – SRP – Single Responsibility Principle =** Tek sorumluluk prensibi (Single-Responsibility Principle) amacı; bir sınıf, interface, fonksiyon vs. sadece tek bir sorumluluk yerine getirmelidir. Bu sorumluluk sadece kendi ile ilgili işleri yapmalıdır. Bir sınıf, interface, fonksiyon vs. birden fazla sorumluluk yerine getirmeye çalıştığı zaman aşırı büyürve karmaşıklaşır.
* **O – OCP –** **Open/Close Principle =** OCP ye göre programlar geliştirmeye açık olmalıdır, yani programı oluşturan modüller yeni davranış biçimlerini sergileyecek şekilde genişletilebilmelidirler. Bir modüle yeni bir davranış biçimi kazandırılarak düşünülen değişiklik sağlanır. Bu durumu, (sistemler gelişime açık, ancak değişime kapalı olmalıdır) cümlesiyle ilk kez Bertrand Meyer formüle etmeye çalışmıştır. Buradaki sistemler ifadesinden kasıt; sınıf, modül veya kütüphane olabilir. Sistemleri mevcut kodları üzerinde değişiklikler yaparak değiştirmeye çalışmak genelde kaosa yol açar. Değişimi sadece yeni kodlar ekleyerek yapabilmek amaçlanmaktadır. Aksi takdirde bu prensip ihlal edilmiş olunur.
* **L – LSP – Liskov SubstitutionPrinciple =** Temel sınıfta, temel sınıftan türetilen bazı sınıflara ait özel fonksiyon vs bulunmamalıdır. Yani temel sınıfta, türetilmiş sınıflardan hiç biri için özel bir şey yazılmış olmamlıdır. Böylece base class ile türetilen sınıflar yer değiştirdiğinde aynı davranışı sergileyecektir.
* **I – ISP – Interface Segregation Principle =** Bir sınıf bir arayüzü uyguluyor ise, o sınıf o arayüzde tanımlı bütün metotları kullanması gerektiğini önerir. Eğer işlemler kendi içlerinde de ayrılıyor ise base interface kullanarak işlemler dağıtılır. Örneğin veri tabanı işlemleri gerçekleştiren bir uygulama yazıyoruz. Veri tabanı olarak accsess ve sql server kullanıyoruz. Access veri tabanı stored prosedurleri desteklemezken sql server desteklemekte. Veri tabanı oldukları için ortak işlemleride bulunmakta. Ortak işlemler için bir interface yazılır. Veri tabanlarının kendilerine özel işlemleri için ayrı arayüzler oluşturulur.
* **D – DIP – Dependency Inversion Principle =** Bu prensibe göre yazdığımız program soyutlama üzerine dayandırılmalıdır, katılaştırma üzerine değil. Tasarladığımız kodun bağımlılığı implementation ( uygulama ) Class’lara değil, Interface’lere olması gerekir. Somut bir sınıfa bağlı olunmamalıdır. Değişkenliğin olabileceği hiçbir yerde somut bir sınıf referansı ya da işaretçisi tutulmalmalıdır. Hiçbir sınıf bir somut sınıftan türetilmemelidir. Temel sınıfının uyguladığı bir metodu ezen bir metot bulunmamalıdır.

**Maven**

DOOB tactical projesinde proje yönetim aracı olarak maven kullanılmaktadır. Maven, ilk versiyonu 30 Mart 2002’de çıkan proje bağımlılık yönetim aracıdır. Hatta direk kendisine “Proje Yöneticisi” denilebilir. Ant’a bir alternatif olarak gösterilen aynı Ant gibi bir JDT(Java Development Tool) ‘dur. Bağımlılık yönetiminin yanında modüler bazda proje geliştirme imkânı, plugin desteğiyle proje yönetimini daha güçlü kılma ve benzeri imkanlar da sunar.

Maven, proje geliştirirken proje içerisinde bir standart oluşturmamızı, geliştirme sürecini basitleştirmemizi, dokümantasyonumuzu etkili bir şekilde oluşturmamızı, projemizdeki kütüphane bağımlılığını ve IDE bağımlılığını ortadan kaldırmamızı sağlayan bir araçtır. Bizi internette sayfa sayfa dolaşıp ihtiyacımız olan library(kütüphane)’nin ihtiyacımız olan versiyonunu arama zahmetinden kurtaran bir çatıdır. Yani projeniz örneğin log4j.jar ve Spring.jar gibi kütüphanelere ihtiyaç duyuyorsa; bu ihtiyacı ilgili jar dosyalarını bularak ilgili yerlere kopyalamak mavenin sorumluluğundadır. Pom.xml (maven in proje ayar dosyası) içerisinde log4j nin şu versiyonu ve Spring’ın şu versiyonu kullanmak istiyorum demeniz yeterlidir. Kullanabilmemiz için çeşit çeşit konfigürasyonlar yapmamıza gerek yoktur. Böylece Projelerinizi istediğiniz yere kolayca taşımanıza + kütüphanelerin yönetilmesine imkân sağlar.

Örneğin Spring Core kütüphanesini projemize dahil edebilmek için aşağıdaki XML kodunu pom.xml dosyamıza eklememiz gerekiyor.

<**dependency**>

<**groupId**>org.springframework</**groupId**>

<**artifactId**>spring-core</**artifactId**>

<**version**>4.3.2.RELEASE</**version**>

<**/dependency**>

Kısaca etiketlerin özelliklerine değinmek gerekirse;

* **<dependency>** : Bir bağımlılık temsil eder. İçerisinde bağımlılığa ait özellikleri barındırır.
* **<groupId>** : Bağımlılığın hangi organizasyona ait olduğunu ifade eder. Burada org.springframework aslında birçok kütüphane barındıran bir organizasyondur. Java dünyasında groupId genellikle projenin web adresi ile alakalıdır. Eğer projenizin web adresi “blogprojem.com” ise groupId “com.blogprojem” olabilir. Bu bir şart değil, sadece genel bir yazılı olmayan kuraldır.
* **<artifactId>** : Belirtilen organizasyon içinde hangi kütüphanenin getirileceğini ifade eder.
* **<version>** : Anlaşılacağı üzere hangi versiyonun getirileceğini ifade eder.

**Spring Framework**

Spring framework, Java tabanlı enterprise uygulamalar için kapsamlı bir programlama ve konfigürasyon altyapı desteği sunar. Yazılımcı bu altyapı desteği sayesinde ciddi bir iş yükünden kurtulur, business logic dediğimiz, uygulamamızda verinin yorumlandığı ve iş kurallarının uygulandığı katmana odaklanma şansı bulur. Spring, kendine has özelliği olan DependencyInjection (bağımlılıkların enjekte edilmesi) metodu ile nesneler arası bağlar, yapılandırma dosyaları üzerinden otomatik olarak gerçekleştirilir. Örneğin;  A ve B sınıflarımız olsun. A sınıfı içerisinde B sınıfının bir nesnesi olsun. Yani B sınıfı, A sınıfına bağlanmış oldu. Bu bağımlılık, A sınıfından bir nesne oluştururken Spring tarafından göz önünde bulundurulur.  Daha sonra A sınıfından bir nesne oluşturmak istediğimizde Spring, otomatik olarak B sınıfından da bir nesne oluşturarak, A sınıfından oluşturduğu nesneye enjekte eder.  Bu örnekte olduğu gibi Spring, DependencyInjection metodu ile bağımlılıkları kullanıcı yerine kontrol eder. Tüm modüllerini kullanma zorunluluğu yoktur. Sadece gerekli modüller kullanılarak uygulama geliştirebilir.

**Aşağıda basit bir spring örneği ver. Bunu iş yerinde yap.**

**Caching ve Spring Cache**

Caching genellikle verileri ve nesneleri ram’e saklayıp bir dış veri kaynağının okunması gereken toplam sayıyı düşürerek, very tabanı güdümlü web sitelerini hızlandırmak amaçıyla kullanılır.

Spring Cache annotasyonlar üzerinden bilgileri cache’lemenize olanak sağlayan bir mekanizma. Spring’in bir parçası. Tek kısıtı, annotasyonlarınızı kullandığınız Bean’lerin Spring tarafından yönetilmesi gerekiyor. En güzel yanı ise alt tarafta istediğiniz cache kütüphanesini(Ehcache, Memcache, Guava, Redis vs.) kullanabilirsiniz. İleride bununla ilgili daha detaylı bir yazı yazmayı düşünüyorum zaten.

Annotasyon

@Cacheable İlgili metodun sonucu ayarlanan cache kütüphanesine kaydedilir. Eğer yeni bir talep gelirse ve aynı ‘key’ değeri kullanıldıysa yeni sonuç doğruca cacheten okunur. Default olarak metodun parametreleri ‘key’ olarak kullanılmaya çalışılır. Tabi burada parametrelerin alt taraftaki cache kütüphanesi tarafından desteklenen veri tiplerinden olması gerekiyor.

@CachePut @Cacheable’da olduğu gibi metod sonucunu ilgili cache kütüphanesine kaydeder. Fakat bir sonraki çağırımda yine method çalışır.

@CacheEvict Bir verinin cache’lenmesinde en önemli konu, cache’teki verinin güncelliğinin korunmasıdır. Bu annotasyon cache’inizin gerektiğinde güncellenmesine olanak sağlar. İlgili metod çağırıldığında ‘key’ değerindeki veri cache’ten silinir.

@EnableCaching

Bu daha çok ayarlarla ilgili. Ayarlarınızı java sınıfı üzerinden yapıyorsanız, ilgili ayar sınıfınızın tepesine bu annotasyonu koyduğunuzda cache’iniz aktif hale gelir. Fakat sadece aktif etmeniz yetmez bit CacheManager tipinde Bean yaratmanız ve hangi Cache kütüphanesinin yaratılacağını belirtmeniz gerekir.