User Interface Design

The user interface:

- User interface, programı kullanması beklenen kişilerin deneyimlerine uygun olmalıdır.
- Kullanıcılar programı genelde interface'e göre değerlendirirler.
- Kötü dizayn edilmiş bir interface, kullanıclıların büyük hatalar yapmasına sebep olabilir.

Human factors in interface design:

- **Limited short-term memory**: İnsanlar 7tane bilgiyi "kısa süreli hafıza"larında tutabilirler. Bundan daha fazla bilgi vermek kullanıcıyı hata yapmaya iter.
- **People make mistakes**: Kullanıcı hata yaptığında verilen hata mesajları uygunsuzsa (sert, kaba, vs.) bu mesajlar kullanıcıyı daha fazla stres altına sokarak daha çok hata yapmasına sebep olabilir.
- **People are different**: İnsanların yetenekleri farklıdır. Designerlar, interface'i kendi yeteneklerine göre tasarlamamalıdır.

UI design principles:

- Bir interface; kullanıcıların ihtiyaç, yetenek ve deneyimlerine göre tasarlanmalıdır.
- Bir interface tasarlanırken kullanıcıların fiziksel ve mental limitleri gözönünde tutulmalı ve kullanıclıarın hata yapabileceği dikkate alınmalıdır.

Şu şekilde seçilmelidir.

- **Consistency***:* Interface'in göürünümü belli bir tutarlılık göstermelidir. Menuler, butonlar aynı formatta olmalıdır.
- **Minimal surprise**: Benzer özellikte işlemler uygulanıyorsa, bu işlemlerin gerçekleştirimi, yazı formatı butonların yerleri benzer özellik göstermelidir.
- **Recoverabilit***y*: Sistem kullanıcı hatalarına karşı dayanıklı olmalı, yaptığı hatayı geri alabilmeli kullanıcı. (örn. Undo commandi yapılmalı)
- **User guidance**: Help sistemleri, on-line yardım ve manualler bulunmalıdır.
- **User diversity**: Farklı tipteki kullanıcılar gözönüne alınmalıdır. (örn. Büyük font olayı konulmalı)

Design issues in Uis:

Iki sorun vardır: "Sisteme girilmesi gereken bilgiler kullanıcıdan nasıl alınacak?" ve "Sistemden kullanıcıya sunulacak bilgiler nasıl gösterilecek?".

Interaction styles:

- **Direct manipulation** : Hızlı ve öğrenmesi kolay,sisteme implement etmek zor. Pc oyunları bu kategoridedir.
- **Menu selection** : Kullanıcının hata yapma olasılığını çok azaltır. Kolay implement edilir. Deneyimli kullanıclar için yavaş olabilir. Çok menü karmaşa yaratır.
- **Form fill-in** : Kolay bilgi girişi, kontrolü kolay. Çok fazla yer kaplıyor.Kullanıcının ihtiyaçlarını tam karşılayamayabiliyor. Stok kontrol sistemleri örnek verilebilir.
- **Command language** : Süper bişi. Esnek.Öğrenmesi zor ve hata ayıklamak problemli. İşletim sistemleri yazıyosan süper.
- **Natural language** : Her kullanıcıya ulaşabilirsin. Ama biraz zor yazarsın ve güvenilir olmaz.

Web-based interfaces:

Interfaceler de web-based olur. Butonlar, fieldlar, radio butonlalr bulunabilir. (ne gereksiz bir başlıkmış bu yav...)

Information presentation:

Bilgi kullanıcıya olduğu gibi ya da bazı formatlarda gösterilebilir. Örneğin direk text olarak gösterilebilirken veriler bir grafik gösterimine dönüştürülebilir.

- **Static information**: Program çalıştıktan sonra değişmeyecek olan verilerdir. Text ya da numeric olabilirler.
- **Dynamic information:** Kullanıcının görmesi geren bu türdeki verilerdeki her değişiklik kullanıcıya verilmelidir. Text ya da numeric olabilirler.

Information display factors:

- Kullanıcı kesin, net bir veri ile mi ilgileniyor yoksa veriler arsındaki ilişkiyle mi?
- Verinin değeri ne kadar hızla değişiyor? Her değişiklik anında kullanıcıya gösterilecek mi?
- Kullanıcı değişen değerler karşısında bir eylem gerçekleştirecek mi? (az sonraaa!!! gibi oldu)
- Direct manipulation interface'i var mı?
- Veri text mi numeric mi? Kullanıcı relative veriyle ilgileniyor mu ?

Analogue or digital presentation?

- **Digital presentation**: Az yer kaplar. Net değerler gösterilebilir.
- **Analogue presentation**: Veri hakkında yüzeysel bilgi almak kolaydır. Relative değerleri göstermek mümkündür.Beklenmedik (exceptional) verileri görmek daha kolaydır.

Data visualisation:

Buraya ne yazsam bilemedim....

Color displays:

- Renkler ayrı bir boyut kadar interface'e. Görünümü güzelleştirir, veriler ayırt edilmesini ve rahatça okunmasını sağlar. (günde bir avuç yerseniz.)
- Beklenmedik durumları belirtmekte kullanılabilir.
- Çok fazla(gereksiz) renk kullanılmamalı (palyaçoya dönmesin program)

Colour use guidelines:

- Renk kullanımını kontrol altında tutun.
- Sistem durumlarındaki değşimleri belirtmek için renk değişimlerini kullanın.

Error messages:

- Uygun olmayan, yetersiz hata mesajları kullanıcıyı programdan uzaklaştır.
- Hata mesajları; kibar,tutarlı, yapıcı, kısa ve öz olmalıdır.
- Kullanıcı profiline göre hazırlanmalıdır.

User documentation:

- Online yardım sisteminin yanısıra döküman olarak da yardım klavuzu olmalıdır.
- Dökümasyon deneyimliden - deneyimsiz kullanıcıya kadar geniş bir kullanıcı aralığına göre olmalıdır.

Document types:

- **Functional description**: Sistemin genel olarak ne yaptığını belirtir.
- **Introductory manual**: Sisteme yinelik küçük bir giriş yapar.
- **System reference manual**: Sistemin tüm özelliklerinden, neyin ne iş yaptığını belirtir.
- **System installation manual**: Sistemin nasıl kurulacağını anlatır.
- **System administrator's manual**: Sistemin nasıl yönetileceğini anlatır.

The UI design process:

UI desing kullanıcı ve designer arasında tekrarlı, dönüşümlü(iterative) bir süreçtir. 3 ana aktivitesi vardır.

- **User analysis**: Kullanıcıların sistemde neler yapacağının belirlenmesi.
- **System prototyping**: Deneme amaçlı farklı prototatipler üretilmelidir.
- **Interface evaluation**: Elde edilen prototiplerin kullanıcılarla beraber denenmesi.

User analysis

- Kullanıcıların sistemden beklentilerini bilmezsek, doğru tasarımı yapamayız.
- Analiz sonuçları kullanıcıların ve designerların anlayacağı bir şekilde belirlenmelidir.

Analysis techniques:

- **Task analysis:** Bir task'in tamamlanması için izlenmesi gereken adımları belirler.
- **Interviewing and questionnaires:** Kullanıcıya yapatığı iş hakkında sorular sormak. Ucu açık, kullanıcının tamamlayabileceği sorular sormak. Kullanıcı kendisine önemli gözüken şeyleri de söylemelidir. Sadece bizim öğrenmek istediklerimiz olmaz. Kullanıcılar toplu olarak bilgi alışverişinde bulunmaldır.
- **Ethnography:** Kullanıcıyı iş yaparken gözlemlemek. Bir gözlemci kullanıcıları izler onlara metinsel bir not almadan sorular sorar. Interviewing ile öğrenilemeyecek şeyler bu şekilde öğrenilebilir. Çünkü kullanıcılar bazı şeyleri ifade etmekte zorlanabilir.

User interface prototyping:

- Kullanıcıya interface ile ilgili deneyim kazandırmayı amaçlar.
- Böyle bir deney yapılmazsa interface'in kullanılabilirliği ölçülemez.
- Önce kağıt üzerinde tasarımlar yapılabilir. Uygun görülürse software haline getirilir. Kağıt üzerinde skeçler hazırlanır. Sistemle etkileşim durumlarını tutmak için storyboardlar hazırlanır. Kullanıcının görüşünü almak için iyi bir yöntemdir.

User interface evaluation:

- Büyük çaplı bir değişim maliyetli zahmetli olur. O yüzden kullanışlılık durumuna göre düzenlemeler yapılmalıdır.

Usability attributes:

- **Learnability:** Kullanıcının yazılım üzerinde hakimiyet kurması ne kadar sürüyor?
- **Speed of operation:** Kullanıcının sistemi kullanma hızıyla, sistemin yanıt süresi ne kadar uyuşuyor?
- **Robustness:** Sistem, kullanıcı hatalarını ne kadar tolere edebiliyor.
- **Recoverability:** Sistem kullanıcı hatalarından ne kadar kolay arındırılıyor?
- **Adaptability:** Sistem belli biri şmodeline ne kadar rahat uyum sağlayabiliyor?

Simple evaluation techniques:

- Kullanıcıdan geri-bildirim almak için anketler.
- Sistemin kullanılırken alınmış video kaydı.
- Sistemin kullanıcı hatalarını ve diğer önemli durumları tutması (log oluşturmak denebilir).
- Online kullanıcı gerbildirimi için altyapı hazırlamak.

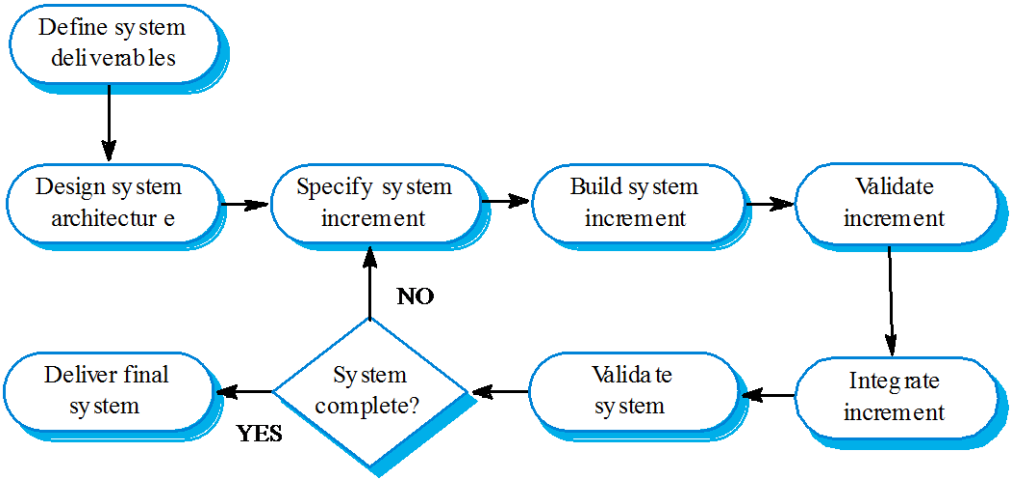
Rapid Software Development

İş alanlarındaki hızlı değişimlere aya uydurabilmek için rapid software development mantığı geliştirilmiştir.

- Değişen iş koşullarından dolayı belli bir denge bulunması imkansızdır.
- Değişen koşullardan dolayı waterfall geliştirim modeli yerine değişime olanak veren iterative specificaiton daha uygundur.

Characteristics of RAD processes:

- Specification, design ve implementation safhaları iç içedir.
- Sistem küçük parçalar halinde gerçekleştirilir. Kullanıcı her parçayı inceler sonraki parçalar için tekliflerde bulunur.



Advantages of incremental development:

- Kullanıcıya hızlı bir şekilde iletilen incrementler kullanıcı için önceliği yüksek olan yapıları barındırır.
- Kullanıcı sistemle iç içedir. Kendi isteklerine göre şekillendirme imkanları vardır.

Problems with incremental development:

- **Management problems**: Sürecin izlenmesi, hataların bulunması zordur çünkü iç içe geçmiş olan safhalar ve dökümantasyon azlığı bunu engeller.
- **Contractual problems**: Normal anlaşmalar specification'ı da içerebilir. Bu yüzen farklı anlaşma formatları kullanılmak zorunda kalınabilir.
- **Validation problems**: Specification olmadan sistemi hangi kriterlere göre test edeceğiz?
- **Maintenance problems**: Durmadan gelen istekleri yanıtlayabilmek için yapılan değişimler structural yapıyı bozabilir ileride değişim maliyetlerini arttırabilir.

Prototyping:

Bazı büyük sistemlerdeincremental development yapmak zordur özellikle aynı projede çalışan birde fazla takım varsa.

Prototyping yönteminde demene amaçlı o an üzerinde uğraşılan requirementları destekleyen küçük bir parçadır. Requirementlar sağlanıyorsa bu prototip atılır ve diğerine geçilir.

Agile methods:

- Desing'dan daha çok kodlamaya önem verilir.
- Iterative yaklaşıma dayanır.
- Çalışan bir ürünü çabuk üretmek ve bunun değişen ihtiyaçlarını çabucak karşılamak birincil hedefidir.
- Küçük ve orta seviyeli business sistemleri ve PC programları için ideal yöntemdir.

Principles of agile methods:

- **Customer involvement:** Müşteri geliştirim sürecinin bir parçasınıdır. Sistem requirementları belirlenmesinde ve gelişim sürecinde yol gösterici larak görev alırlar.
- **Incremental delivery:** Yazılım increment denen parçalar halinde gerçekleştirilir. Her incrementte yeni bir requirement gerçekleştirilir.
- **People not process:** Geliştirme ekibinin yetenekleri belirlenmelidir. Gruptaki kişilerin kendi tarzlarında kod yazmasına izin veilmelidir.
- **Embrace change:** Yazılımın değişebileceği yerleri belirleyip ona göre bir tasarım yapılmalıdır.
- **Maintain simplicity:** Mümkün olduğu kadar sade bir yazılım süreci sağlanmalıdır. Eğer sistemin complex yerleri basite indirgenmelidir.

Problems with agile methods:

- Process sürecinde bulunan müşterilerin ilgisini uzun süre projede tutmak zor olabilir.
- Takım üyeleri agile metodları özelliği olan sıkı, zor çalışma ortamına ayak uyduramayabilirler.
- Birden fazla stakeholders bulunuyorsa öncelik gerektiren değişimlerin yapılması zor olabilir.
- Yazılımı sadeleştirmek ekstra bir çalışma gerektirir.
- Diğer iterative yaklaşımlarda olduğu gibi anlaşma (contracts) koşulları sorun çıkarabilir.

Extreme programming

- En çok bilinen ve en kullanılan agile metottur.
- Birgün içinde bir yazılım birden fazla versiyonu çıkabilir.
- Üretilen incrementler 2 haftada bir müşterilere iletilir.
- Her parça için test yapılır ve testi geçmeyen parça müşteriye verilmez.

Extreme programming practices

- **Incremental planning*:*** Story kartlarına requirementlar yazılıyor. Zaman uygunluğu ve önceliğine göre bu requirementlar gerçekleştiriliyor. Requirementlar "task" denen parçalara bölünerek implement ediliyor.
- **Small Releases *:*** Fonksiyonellik bakımından en zayıf requirementdan başlanıyor. Diğer requirementlar bu ilk yapının üzerine geliştiriliyor.
- **Simple Design*:*** İhtiyaçları karşılayacak kadar kod. Daha fazla değil.
- **Test first development*:*** Bir functionality impement edilmeye başlamadan test birimi yazılır.
- **Refactoring*:*** Gelişen kod tekrar elden geçmelidir. Yapılan eklemeler, geliştirmeler sadeliği bozabilir, yeni eklemeler için engel olabilir.
- **Pair Programming *:*** Developerlar çiftler halinde çalışırlar. Biri kod yazarken diğeri yazılan kodu inceler, arkadaşına destek olur, sırtını sıvazlar.
- **Collective Ownership*:*** Bu developerlar kodun her alanında çalışırlar. Böylece belli bir bölgede belli bir grubun yoğunlaşması olmaz. Developerlar tüm projeyi sahiplenir. Herkes her şey hakkında bilgi sahibi olur.
- **Continuous Integration*:*** Her tamamlanan task asıl projeye eklenir ve her eklemeden sonra sistemin tüm unit testleri geçmesi gerekir.
- **Sustainable pace*:*** Çok sayıda mesai iyi değildir. Araştırmalar çok sayıda fazla mesai yapan kişilerde verim düşüklüğü olduğunu gösteriyor.
- **On-site Customer*:*** Müşteri temsilcisi, developerlar için her zaman ulaşılabilir olmaldıdır çünkü o da projenin bir elemanıdır. Kendisi proje requirementlarının gerçekleştiriminde yardımcı olur.

XP and agile principles

- Ufak ve sık yenilenen sistemler için uygundur.
- Customer involvement demek geliştirme takımıyla full-time müşteri ilişkisi demektir.
- Basit bir pair programming değil, tüm işi sahiplenen ve aşırı çalışmadan uzak duran prensiplerdir.
- Son ikisini anlamadım.

Requirements scenarios

- XP'de requirementlar scenario ya da user stories olarak adlandırılır.
- Bunlar kartlara yazılıdır. Developerlar bunları tasklere bölerler. Bu taskler yaklaşık olarak proje bitim süresini ve diğer harcamalar hakkında bilgi verir.
- Müşteri kendi değişen ihtiyaçlarına ya da önceliklerine göre story kartlarını güncelleyebilir.

XP and change:

- Geleneksel yazılım mühendisliğinde, kodu değişimlere açık olarak yazmak vardır. "Proje mutlaka değişecek o yüzden tasarımını ona göre yap" yaklaşımı kullanılır. Bu uzun bir zaman ve efor gerektirir.
- Xp ise bu işe sıcak bakmaz. Değişimlerin her zaman beklenildiği gibi olmayabileceğini, buna göre tasarım yapmaya değmeyeceğini düşünür.
- Eğer değişim olursa refactoring ile hallederiz.

Testing in XP:

- Önce test, sonra kod.
- Her senrayo için uygun test.
- Test development ve validation için kullanıcı (müşteri) desteği.
- Otomatikleştirilmiş test, sisteme her yeni bir birim eklendiğinde tüm sistemi test eder.

Test-first development

- İhtiyacı giderecek olan kod yazılmadan önce testi yazılır.
- Testler bir veri gibi değil de çalıştıralcak bir kod gibi yazılır böylece sistem testi geçerse otomatik olarak bir check döndürür.
- Tüm testler yeni bir yapı eklendiğinde tekrarlanır.

Pair programming

- XP'de programcılar çiftler halinde çalışıyolar
- Bu yazılan kodda bir ortaklık ve kişiler arasında bilgi akışı sağlıyor.
- Her satır birden fazla kişi tarafından kontrol edilmiş oluyor.
- Tüm takım kodu bildiği için refactoring için cesaret verici oluyor.
- Araştıramalar pair programming'in, iki ayrı kişi varmış gibi çalışmaya yakın olduğunu gösteriyor.

Verification and Validation:

Verification:

"Are we building the product right?" Yazılım belirlenen gereksinimleri karşılamalı. Bu control edilir.

Validation:

"Are we building the right product?": Belki yazılım , yazılım sürecinde belirlenen gereksinimleri karşılar ama bu gereksinimler doğru hazırlanmamıştır. Yani validation da "kullanıcının gerçekten istediği yazılımı mı yapıyoruz?" sorusu sorulur.

V&V Process:

Sadece kodlama işi bittikten sonra V&V ye başlanabilir.

V&V processin amacı yazılımın kullanıcının isteklerini karşılayıp karşılamadığını kontrol etmektir.

Asıl olarak iki temel amacı vardır.

- Yazılımdaki hataları(defect) bulmak.
- Yazılımın kullanıcı için işe yarar (useful) ve kullanılabilir (useable) olup olmadığını bulmak.

V&V Goals:

Yazılımın amacına uygun olduğuna emin olabilmek için V&V yapılır. Ama bu yazılımda hiç kusur olmaması anlamına gelmez.Yapılan yazılıma göre belli bir hata toleransı vardır.

V&V Confidance:

Temel olarak üç etmene bağlı olarak güvenilebilirlik değişir.

- **Sotware Function**: Yazılım çok hayati bir iş yapıyorsa hata kabul edilemez.
- **User Expectations**: Kullanıcının yazılımdan beklentileri düşük olabilir.
- **Marketing Enviromant**:Yazılımı bir an önce piyasaya sürmek hayati önem taşıyabilir. Bu sebepten V&V ye çok fazla zaman ayırılamayabilir.

Static and dynamic verification:

- **Software İnspections (statick verification)** *:* Yazılım çalıştırılmadan yapılan konrtollerdir. Tasarım dokümanları, kaynak kodları incelenebilir. Bazı toollar kullanılarak ya da elle de yapılabilir.
- **Software Testing(dynamic verification)** *:* Yazılımın çalıştırılıp denenmesiyle ilgilidir. Test verileri kullanılır ve elde edilen sonuçlar beklenen sonuçlarla karşılaştırılır.

Program Testing:

Hataların varlığı gösterilebilir ama yokluğu kanıtlanamaz.Hata bulunamaması bazen program testinin in başarısız olduğu, hataların yakalanamadığı anlamına gelebilir.

Eğer *static verificationla* birlikte kullanılmazsa tam olarak başarılı olamaz.

İki test şekli vardır;

- **Defect Testing**: Kusurları bulmaya yönelik hazırlanmış testlerdir.
- **Validation Testing**: Yazılımın gereksinimlerini karşılayıp karşılamadığını test eder.

Testing and Debugging:

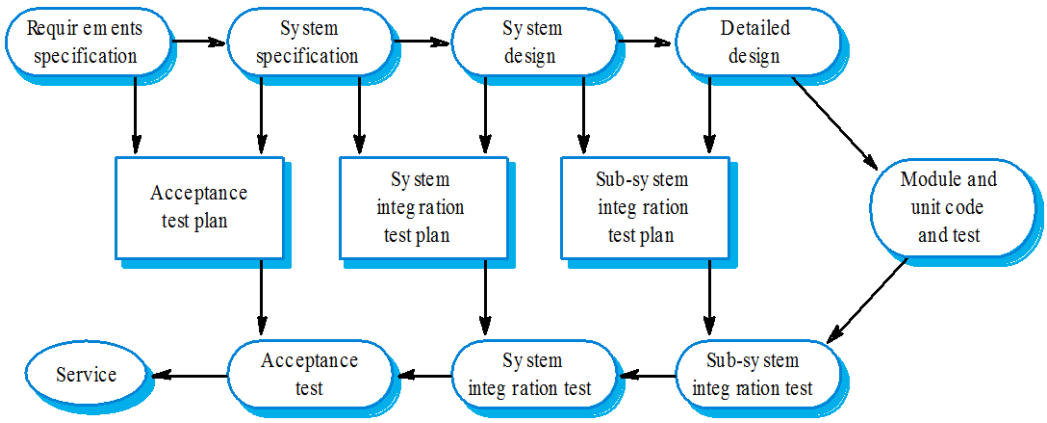
Test ve *debug* bir birinden farklı şeylerdir. V&V de hataların olup olmadığı, varsa neler olduğu anlaşılmaya çalışılırken *debuging* de olan hataların kaynağını bulunmaya ve bu hataları düzeltilmeye çalışılır.

Debug yapılabilmesi için hatayla ilgili bir fikre bir hipoteze sahip olmak gerekir. Bunun için de sistemi tanımak lazım.

V&V Planing:

Yazılım geliştirmenin erken safhalarından itibaren planlama başlamalıdır.

Plan statik ve dinamik doğrulama arasında denege kurmalıdır. Test planlama ürün için testler hazırlamaktan öte, testler için bir standart getirmektir.



Her tasarım aşamasına karşılık gelen bir de test aşaması var. Ama dizaynın ilk safhalarana, testin son safhaları karşılık geliyor. Yani küçük parçaların testinden en son gereksinimlerin testine doğru bir akış var.

- **Acceptance Test**: kullanılıcılarla birlikte yapılan test.

The Structure of a Test Plan:

Bir test planında aşağıdaki aşamalar olmalıdır;

- **The Testing Process**: Test surecinin genel olarak tanımlanmalı
- ***Requriments Traceability*:** Bütün gereksinimlerin nasıl eksiksiz olarak test edilebileceği belirlenmeli
- **Test Items**: Yazılım sürecinde test edilecek ürünler belirlenmeli
- **Testing Schedule:** Test sürecinin bir takvimi olmalıdır. Bu takvim genellikle yazılım geliştirme takvinde belirtilir.
- **Test Record Procedure**: Sadece test yapmak yeterli değildir. Yapılan testlerin sonuçları da kaydedilmelir. Bunun için bir format, bir sistematik belirlemek gerekir.
- **Hardware and Software Reqirements**: Yazılımın çalışması için gerekli olan donanım ve yazılımlar nelerdir.
- **Constraints**: Test sürecini etkilecek kısıtlamalar (eleman eksikliği gibi)

Software Inspection(denetleme):

Bir S*tatic Verification'dur*. Yani Software Inspection için kodun çalıştırılması gerekmez ve yazılım geliştirme süreci boyunca elde edilen tüm ürünlerin (dizayn dokumanları da dahil olmak üzere) genel olarak elden geçirilmesidir.
Hataların bulunmasında oldukça etkilidir(%60 gibi). Software Testing'de harcanacak zamanın azaltılmasını sağlar.

Inspection Success:

Test aşamasında oluşan bir hata başka hataların görünmesini engelleyebilir. Bütün hataların farkedilmesi için birçok kez kaynak kodun çalıştılması gerekecektir. Ama Software Inspection bir kez yapıldığında birçok hata farkedilebilir.

Ayrıca Inspection yaparken yazılımın domain'i ve genel yapısı bir kez daha gözden geçirildiği için çıkan hataların genel yapısı ve kaynağı bulunabilir.

Inspections and Testing:

Inspection ve Testing birbirlerine zıt kavramlar değillerdir. Birbirlerini tamamlar ve birlikte kullanılmaları gerekir.
Inspection'lar yazılımın analize göre yapılıp yapılmadığını belirleyebilir fakat yazılımın, kullanıcının asıl istediği yazılım olup olmadığını belirleyemez. Ayrıca Inspeciton'larda *nonfunctional requirment'*ların gerçekleşip gerçekleşmediğini belirleyemeyiz. Yani programı çalıştırmadan hızını, kullandığı belleği tahmin etmemiz imkansızdır.

Program Inspections:

Programdaki kusurları bulmak için yapılan sistematik gözden geçirmelerdir. Diğer gözden geçirmelerden temel farkı programın genel durumunu gözden geçirmek yerine kusur bulmaya odaklanmasıdır.
Program Inspeciton'un amacı hataları düzeltmek değil hataları bulmaktır.Bunlar anormal durumlara sebep verebilecek olan mantık hataları olabilir.
En azından 4 kişiden oluşan bir ekip tarafından yapılmalıdır.Bu ekip içinde belli görev dağılımları olmalıdır.

Inspection Pre-Conditions:

- Kesin bir tanımlama olmalıdır.
- Ekipteki elemanların program geliştirmedeki standartlar hakkında bilgisi olmalı
- Ekibin elinde Syntax olarak hatasız kodlar olmalı.
- Error Checklist olmalı
- Yönetim, projenin başında inspection'un getireceği maliyetleri kabul etmeli. Çünkü inspection oldukça maliyetli bir süreçtir.
- Yönetim inspection'u kimin hata yaptığını bulmaya yönelik bir araç olarak kullanmamalı.

Inspection Procedure:

- Inspeciton yapacak ekibe yazılımın tanıtımı yapılır.
- En güncel kodlar ve tasarım dokumaları ekibe dağıtılır.
- Inspection yapılır.
- Hatalar düzeltilir (programı yazanlar tarafından).
- Yeniden inspection yapmaya gerek duyulur veya duyulmaz.

Inspection Roles:

- **Author/owner**: Program geliştirmeden sorumlu kimse
- **Inspector**: Proramlarda ve dokumanlarda hatalar, uyuşmazlıklar bulur. Inspeciton ekibinin bakış açısı dışında kalan kısımları, programın genel hatlarını gözden geçirir
- **Reader**:En gariban budur. Dokumanları ve ya kodu ekibe okur, gösterir.
- **Scribe**: Ekibin bulduğu sonuçları kayıt altına alırç
- **Moderator**: Inspection sürecinden genel olarak sorumludur, sürecin sonuçlarını patrona raporlar.
- **Chief Moderator**: Inspection sürecindeki ilerlemelerden, checklist'in güncellenmesinden falan sorumludur.

Inspection Check List:

Programda hani hataların aranacağını gösteren listedir.
programlama diline bağımlıdır. Yapılan hatalar programlama diline göre farklılık gösterir. Mesela programlama dilinde tip konrolü yoksa checklist uzar.
örenek;

- **Data Faults***:* Kullanılan veri yapılarıyla ilgli hatlardır. Her değişken kullanılmadan önce tanımlanmış mı? gibi
- **Control Faults***:* Akış kontrollerinde yapılan hatalardır. Sonsuz döngü var mı?
- **Input/Output Faults**: Çıktılar için her koşulda değer dönüyormu var mı(null referance), beklenmeyen girdiler sorun oluyor mu?
- **Interface Faults***:* Her fonksiyon için parametrelerin sayısı, sırası, tipi falan doğru mu?
- **Storege Managment Faults***:* Her işi biten nesne bellekten atıldı mı?(de-allocate)
- **Exception Managment Faults**: Olabilecek her hata dikkate alındı mı?(sayı grilmesi gereken yere harf girince hata mesajı verecek mi?)

Automated Static Analysis:

Kaynak kodu okuyup muhtemel hataları gösteren programlardır.Inspection yaparken gözden kaçan noktaların aydınlatılması için faydalıdırlar. Inspection sürecine bir destek olabilirler ama alternatif olamazlar.
Çok miktarda bilgi içerirler.
Gereklilikleri programlama diline göre değişir. Bazi dillerde compilar hataları daha iyi denetleyebilir(java gibi), bazılarında denetim daha azdır. Denetim az olan dillerde Automated Static Analysis daha etkilidir.

Stage of Static Analysis:

- **Control Flow Analysis** : Birden fazle giriş ve çıkış noktası olan döngüleri, ulaşılamaz kodaları gösterir.
- **Data Use Analysis** : Değişkenleri kontrol eder. Birden fazla tanımlanan değişken var mı falan.
- **Interface Analysis** : Fonksiyonların tanımlamaları, parametreleri
- **Data Flow Analysis** : input ve outputlarda olabilecek anormallikleri gösterir.
- **Path Analysis** : Program dışına çıkan parhlere falan bakar

Software Testing

Testin temel olarak iki amacı vardır.

- Kullanıcıya ve geliştiriciye sistemin, gereksinimleri karşıladığını göstermek.
- Sistemdeki hataları, kusurları bulmak.

İki farklı test tipi vardır.

- **Component Testing**: Sistemin alt bileşenlerinin tek tek test edilmesi anlamına gelir.Bu testlerden o bileşeni geliştiren kişi sorumludur. Bu testler geliştiricinin yeteneğine ve deneyimine dayanır.
- **System Testing**: Componentlerin oluşturduğu alt sistemler veya tüm sistemin genel olarak test edilmesi anlamına gelir.Bu test genellikle geliştiricilerden farklı bir test ekibinin sorumluluğundadır. Sistem testleri sistem tanımlamalarına(specification) a bağımlı olarak yapılır ve yazılımın specification'lara uygunluğunu belirlemeye çalışır. Yani eğer system specification'da bir hata varsa bu testlerde belirlenemez.

Eğer sistemin hata toleransı çok az sa konusunda uzman test ekiplerinin kullanılmasında fayda vardır(maliyeti artırır) ama eğer hata toleransı daha fazlaysa yazılım geliştiricilerde kendi testlerini yapabilir.

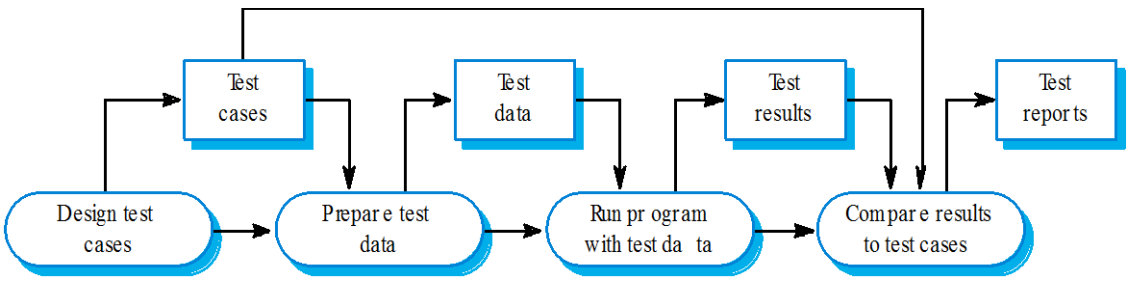
Defect Testing:

Programdaki hataları, beklenmeuyen davranış şekillerini bulmayı amaçlar.
Başarılı bir defect testing programın çakılmasını, patlamasını, abuk subuk şeyler yapmasını sağlar.
Defect testing'le hataların varlığı anlaşılabilir ama yokluğu kanıtlanamaz.

Testing Process Goals:

- **Validation Testing**: Sistemin geliştircinin ve müşterinin beklentilerini karşılayıp karşılamadığını test eder.Başarılı bir test sistemin istenen şekilde çalıştığını gösterir.(defect testing de bunun tersi)
- **Defect Testing**:Yukarıda anlatılan şeyler.
- **Success Testing**: Hata vermeyen gereksinimlerin uygun olarak çalıştığını gösterir.

Software testin Process :



Testing Policies:

Sistem testlerini seçerken izlenecek yolları belirlenir.
Geniş kapsamlı testlerin (*exhausive tests*) yapılması çok maliyetli ve zordur. Programın çalışması sırasında oluşabilecek her durum , her olası kombinasyon test edilir. Bunu karmaşık sistemlerde yapmak çok zor hatta bazen imkansızdır.
Exhausive testing yerine sistem kullanımından elde edilen deneyimlerle aşağıdaki gibi yaklaşımlar izlenebilir;

- Menülerle ulaşılabilen her işlev kontrol edilir.
- Aynı menü içerisindeki tüm fonksiyonların(yani birbiyle alakalı fonksiyonların) kombinasyonları test edilir.
- Veri girişi olan yerlerde beklenen verilerin yanında beklenmeyen veriler de testler yapılmalıdır.

System Testing:

Bileşenlerin oluşturduğu sistemleri veya alt sistemleri inceler bunun yanı sıra sisteme yapılacak ekelemeri (incement) de inceleyebilir.
İki aşaması vardır.

- **Integration Testing:** Component'ler bir araya getirilirken sorunlar oluşacak mı? Bu test edilir.
- **Release Testing:** Teslim edilecek sistemin tamamı test edilir.

Integration Testing:

Sistem geliştiriliken kullanılan bileşenler hazır bileşenler olabilir, daha önce yazılmış olabilir veya yeni geliştirilmiş olabilir. Bu bileşenler tektek düzgün çalışsa da bir araya geldiğinde sorunlar ortaya çıkabilir.
İşte integration testing bu componentlerin bir araya getirilmesi sırasında ortaya çıkabilecek sorunları test eder, bileşenler arayüzler vasıtasıyla düzgün bir iletişim kurabiliyor mu anlamaya çalışılır.
Hata oluştuğunda debug edilecek componenti belirler. Genellikle progrmlama kusurlarını bulmaya yöneliktir.
Integration iki farklı prensibe göre yapılabilir.

- **Top-Down Integration:** Bileşenlerin eklenebilmesi için bir iskelet hazırlanır ve bileşenler tek tek bu iskelete eklenir.
  Sistem tasarımındaki hataları bulmak daha kolaydır.
  Geliştirmenin ilk safhalarında kısıtlı sayıda denemeye imkan sağlar.
- **Bottom-Up Integration:** Bir iskelete gerek yoktur. Bileşenler birbirlerine sırayla eklenir.
  Testlerin yürütülmesi genellikle daha kolaydır.

Hataların yerini kolay bulabilmek için bileşenler tek tek eklenmeli ve ardından testler yapılmalıdır.Buna *Incremantal Integration Testing* denir.

- **Regression Testing**: Daha önce yapılan testlerin sistemin yeni halinde tekrar denenmesidir.

Testing Approaches

- **Architecture Testing**:Sistemin genel mimarisi incelenir. Top-Down la daha kolay yapılır.Fazladan kod yazmak gerekebilir.
- **System Demonstration**: Yazılımın ne tür fonksiyonlara sahip olacağı daha yazılım tamamlanmadan gösterilebilir.Top-Down'la daha rahattır.
- **Test imlementation**:Testlerin gerçekleştirilmesi. Bottom-Up daha rahat.
- **Test observation**:Test sonuçlarının incelenmesi, hatanın kaynağının tespit edilmesidir. Bottom-Up daha iyi.

Release Testing:

Geliştiricilerin programı teslim etmeden önce yazılıma olan güvenlerini artırmayı amaçlar.
Sistemin içerisine bakmadan yapılan testlerdir(black-box testing). Sistemin genel işlevlerini yerine getirip getirmediğini belirler.
System Spesification'a bağlıdır.
Test ekibinin sistemin iç yapısıyla ilgili bilgisi yoktur.

Testin GuideLines

Test ikibine hata bulmalarında yardımcı olacak yöntemlerdir.

- Sistemdeki bütün hata mesajlarını görmeye yönelik test verisi hazırlamak
- Aynı test verisini birkaç defa üst üste girmek.
- Sistemi abuk subuk çıktılar vermeye zorlamak, laftan anlamazsa sopayla girişmek ,azını burnunu kırmak.

Use Cases

- Use case'lere bakarak her use case gerçekleştirilmiş mi kontrol edilir.
- Use case'le ilgili SSD lere bakarak hazılanması gereken test veri ve elde edilmesi beklenen çıktı hakında bilgi edinilebilir.

Performance Testing:

Non-functional requirement'ları önemli olan sistemler için yapılır. Sistem üzerindeki yük, sistemin performansı kabuledilemiyecek bir seviyeye düşene kadar artırılır. Böylece sistemin kadırabileceği en fazla yük miktarı bulunabilir.
örn:winamp playlistine en fazla kaç şarkı ekliyebiliriz

Stress Testing:

Sistem ağır yük altındayken nasıl davranıyor bunu belilemek için yapılır. Veri yükü çok fazla olsa da sistem geridönülemez hatalar yapmamalıdır ya da tamamen iflas etmemelidir.
Örn: winamp playlistine 'a 500gblık şarkı koyarsak ne olur? Tamamen çökmemesi lazım.

Component(unit) Testing:

Sistemin gereksinimlerini karşılayıp karşılamadığına değil de her bileşenin doğru düzgün çalışıp çalışmadığına bakar. Kodlamadaki kusurları bulmaya çalışır. Bileşenler:

- Sınıfların fonksiyonları, metotları
- Sınıflar kendileri
- Tanımlanmış arayüzlerle erişilebilen karmaşık bileşenler olabilir.

Object Class Testing:

Her nesnenin baştan aşağı test edilmesidir.

- Sınıf içindeki tüm fonksiyonlar test edilir.
- Sınıfın tüm attribute'leri get ve set edilir.
- Her olası durum(state) için testler yapılır.

Inheritance kullanılması object class testing i zorlaştırı çünkü inherite edilen her sınıf için ayrı testler yapmak gerekir.

Interface Testing:

Bilşenlerin arayüzlerindeki hataları bulmayı amaçlar. Bileşenlerin birbirilerinin arayüzleri üzerinde yaptığı yanlış varsayımları bulmaya çalışır.

Interface Types:

4 tane arayüz şekli var.

- **Parameter Interfaces**: Bir prosedüreden diğereine veri aktarımı.
- **Shared Memory Interface**: Bir alt sistemin sağladı veri ve belleğin başka alt sistemler tarafından kullanılması.
- **Procedural Interface**:Alt sistemler diğer alt sistemlerin kullanabilmesi için bazı prosedürleri içerir(encapsulate).
- **Massage Passing Interface**: Alt sistemler birbirlerinden servis isteyebilir ve geriye servisin sonucu döndürülür. Clint-Server yaklaşımı gibi.

Inteface Errors:

- **Interface Misusa:** Bir prosedür çağırırken parametreleri yanlış sırada göndermek gibi hatalı kullanımlar.
- **Interface Misunderstanding**: Çağıralan bir presedürün ne iş yaptığını yanlış anlamak.
- **Timing Errors**: Servis isteyen ve servis istenen alt sistemlerin birbirinden farklı hızlarda çalışması.

Interface Testing Guidelines:

Arayüzleri test etmek için bazı öneriler:

- Parametreleri alt ve üst sınırlarda yollamak.Verilen iki integer derğeri toplayan bir component çağırılacaksa.Sınırda iki değer verirsek taşma(ower-flow) olabilir.
- Pointer olan parametreleri null göndermek.
- Hatalı davranışlara sebep olacabilecek test bilgileri yollamak.
- Message Passing sistemlere aşırı yüklemeyle servis veren sistemi zorlamak.
- Ortak bellek kullanan sistemlerde bileşenleri farklı kombinasyonlarda çalıştırmak. Mesela bir fonksiyonun değiştirdi veri diğer fonksiyonun çalışmasını bozuyor mu?

Test Case Design:

Sistemi test etmek için girilecek veriler ve beklenen çıktıların hazırlanmasıdır. Veriler hazırlanırken kusur bulmayı ve sistemin doğru çalışıp çalışmadığını anlamayı hedefler. 3 temel yaklaşım vardır.

- **Requrements Based Testing**:
- **Partition Testing**:
- **Structural Testing**:

Requrements Based Testing:

Kusur bulmayı amaçlamaktan çok sistemin gereksinimleri yerine getirdiğini göstermeyi amaçlar.
Bunun yapılabilmesi için use caselerin, gereksinimlerin düzgün şekilde belirlenmiş olması gerekir.

Partition Testing:

Girdi ve çıktılar genellikle içerisindeki verilerin ortak özelliklere sahip olduğu sınıflaradan oluşur(işte integer sayılar -32.768 ile 32.767 arasındadır). Burdan yola çıkarak her değer için test yapılmaz. Böylece test verileri kısaltılmış olur.
Partition Testing oluşturulacak test verisinin belli bölümlere ayrılarak hazırlanmasını amaçlar.
Örneğin eğer verilen bir sayıyı array'de arayan bir fonksiyonu test edeceksek;

- Ön kabul: array boş değildir.
- Son durum : aradığımız sayı array'deyse sırası değilse, sayı bulunamadı mesajı döndürülür.

Bu durumda hazırlayacağımız test verisi için;

- Hazırladığımız test verilerinin bir kısmında ön koşul gerçekleşmemli.
- Array'in sadece 1 elemana sahip olduğu durumlar olmalı.
- Bir kısmında aranan sayı arrayde bulunmalı,bir kısmında bulunmamalı.
- Arrayde birden fazla olan sayılar denenmeli
- Array'ın başından, sonundan, ortasından sayılar seçilerek aranmalı gibi

Structural Testing:

Açık kutu, cam kutu, beyaz kutu falan gibi saçma sapan isimler de verilmiş bu yapısal teste.
Structural test yapabilmek için programın yapısı hakkında bilgi sahibi olmalıyız ve kaynak kod elimizde bulunmalı. Bu test doğrulamadan çok hata bulmaya yöneliktir.
Structural Testin asıl amacı her olası durumu programın kaynak kodu üzerinde görebilmek, her deyimi çalıştırmaktır.

- **Path Testing**: Her deyimin çalışması için kod üzerindeki bütün yolların kullanılması gerekir. Yani bir *if-then-else* yapısında hem *true* hem de *false* girdiler denenerek bütün deyimler çalıştırılmalı.

Software Evolution

Softaware Change:

Yazılımlarda değişiklik yapılması kaçınılmazdır. Bu değişikliklerin sebepleri;

- Gerekesinimlerin değişmesi, yeni gereksinimler eklenmesi,
- Hataların onarılması
- İş dünyasının sürekli değişmesi
- Sisteme yeni bilgisayarlar, yeni alet edevat eklenmesi(mesela el bilgisayarları)
- Donanımdaki değişiklere uygun olarak performansın ve güvenirliğin artılmak istenmesi olabilir.

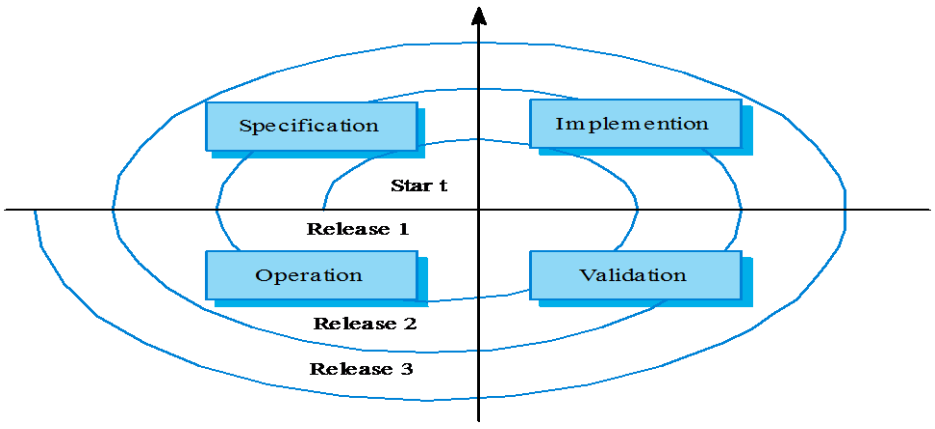
Mevcut yazılımların değiştirilmesi firmalar için bir derttir.

Importance of Software Evolution:

Halen kullanımda olan yazılımlar için çok fazla yatırım yapılmıştır ve bu yazılımlar bazen firmalar için hayati önem taşırlar. Bu yazılımların değerlerini koruyabilmeleri için değişen iş koşullarına göre yenilenmeleri gerekmektedir.
Yazılım bütçelerinin büyük kısmı yeni bir yazılım üretmekten çok var olan yazılımları değiştirmek için ayrılır.

Spiral Model of Evolution:

Tanımlama, gerçekleştirme , sınama ve kullanıma sunma aşamaları sürekli olarak tekrarlanır.



Program Evolution Dynamics:

Sistem üzerindeki değişikliklerle ilgili yapılan araştırmalardır.
Lehman ve Belady geliştirilen yazılımlar üzerinde çalışmalar yapmışlar ve bazı genel sonuçlar çıkarmışlardır.
Bu çıkarımlar kesin olmasa da genel olarak doğrudurlar. Büyük çaptaki yazılımlar bu çıkarımlarla örtüşür.

Lehman's Laws:

- **Continuing Change:** Sistemlerin bakımı, değiştirimesi(maintanence) kaçınılmazdır.
- **Increasing Comlexity:** Değişiklikler, eklemeler yapıldıkça sistem karmaşıklaşır.
- **Large Program Evolution:** Bir Sistemdeki değişiklikler birbirlerine benzer.Yani her yeni eklenti arasındaki süre, eklentilerin büyüklüğü, hata bildirim sayısı genel de birbirine yakındır.
- **Organisational Stability**: Programların gelişme hızı ayrılan kaynağa bakmaksızın sabittir. Yani siz ne kadar büyük bir ekip kurarsanız kurun belli bir geliştirme hızının üzerine çıkamazsınız çünkü bu sefer de ekip üyelerinin iletişimi zayıflar.
- **Conservation of Familiarity**: Programın ömrü boyunca her sürümdeki gelişme miktarı yaklaşık olarak birbirinin aynıdır.
- **Continuing Growth**: Sistemlerin sağladığı kullanıcı memnuniyeti sürekli artmalıdır.
- **Declining Qualty**: Sistemin kalitesi çalıştığı çevredeki gelişmelere uymadığı sürece kabul görmez.
- **Feedback System**: Yazılımı geliştirmek için kullanıcıdan geri bildirimler alınmak zorundadır.

Software Maintenance:

Yazlım kullanıma girdikten sonra yapılan değişikliler *maintenance* oluyor. Maintenance'de genel olarak sistem mimarisinde büyük değişiklikler yapmaz. Var olan bileşenler değiştirilir ya da sisteme yeni bileşenler eklenir.

Maintenance is Inevitable:

Yazılım geliştirilirken gereksinimlerde değişiklikler olmuş olabilir, sistem kullanıma girdikten sonra çevresiyle sıkı bir ilişkiye girdiği için uygulama alanını değiştirebilir bu da gereksinimleri değiştirebilir.
Yazılımlar kullanışlı kalabilmek için çevrelerindeki değişikliklere uyum sağlamak zorundadır.

Types Of Maintenance:

3 maintanence şekli vardır.

- **Maintenance to Repair:**
- **Maintenance to Adapt**: Yazılımın yeni işletim sistemlerine, yeni donanımlara ya da başka yazılımlara adaptasyonunu hedefleyen maintenance'lar.
- **Maintenance to Add**: Yazılıma yeni gereksinimler doğrultusunda yeni işlevler kazandırmak için yapılan maintenance'lar.

Maintenance için harcanan kaynaklarda en büyük paya Maintenance to add0 (%65) sahiptir. Diğerleri: maintenance to adapt(%18),maintenance to repair(%17)

Maintenance Costs:

- Genellikle bakım masrafları yazılım geliştirme masraflarından fazladır(oranı sistemden sisteme değişir).
- Bakım masrafları hem teknik hem de teknik olmayan sebeplere göre değişebilir (programı yazan elemanın işten ayrılması ve geri de programın yapısını bilen kimse kalamaması gibi).
- Yapılan her bakım daha sonraki bakımları zorlaştırır çünkü sistemin mimarisini zedeler.
- Eski yazalımların geliştirilmesi daha masraflı olur(eski işlemcilerin, işlerim sistemlerinin, programlama dillerinin kullanılmış olması yüzünden).

Development/Maintenance Costs:

Yazım geliştirilmesi esnasında çakallık yapıp masrafları kısmak için sistemin yapısallığına önem gösterilmezse, gerekli dokumanlar çıkarılmazsa ileride bakım masrafları çok daha fazla artar çok daha pis patlar.
Yani sistem geliştirirken çok özenilirse, para ve zaman harcanırsa ileride bakım için fazladan haracanması muhtemel zaman ve para korunmuş olur.

Maintenance Cost Factors:

Maliyeti etkileyen faktörler 4 başlıkta toplanabilir.

- **Team Stability**: Eğer programı geliştiren ekip bakımla da ilgilenirse masraflar azalır çünkü adamlar zaten sistemi iyi tanıyo.
- **Contractual Responsibilty:** Eğer yazılımı geliştiren ekip, kurum sistemin bakımından sorumlu değilse tasarımı ilerideki olması muhtemel değişiklikleri düşünmeden yapabilirler. Bu da bakım masraflarını artırır.
- **Staf Skills:** Genellikle bakım yapan çalışanlar çalışma alanı hakkında çok deneyimli değillerdir. Bunun sebebi bakım sürecinin ikinci sınıf, tatsız bir iş gibi gözükmesi ve çalışmaya yeni başlamış çömlerin başına yıkılmasıdır. Bu da bakım masraflarını artırabilir(gerçi çömlere daha az maaş verirler ama) çünkü deneyimsiz çalışanın yeni bir programlama dili öğrenmesi gerekebilir, sistemi anlaması zor olabilir.
- **Programing Age and Structure**: Eski yazılımları anlamak ve değiştirmek daha zordur çünkü bu tür yazılımlar modern yazılım geliştirme teknikleriyle yazılmamış olabilir, Etkinliği artırmak için yapısal bir şekilde tasarlanmamış olabilir, artık pek kullanılmayan bir dille yazılmış olabilir.

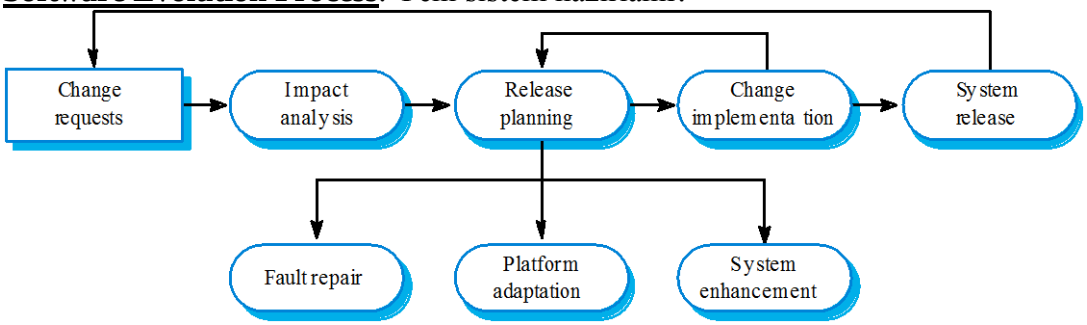
Evolution Process:

Evelution process 3 kritere bağlıdır.

- Bakım yapılacak yazılımın türüne
- Kullanılan geliştirme süreçleri.
- Geliştirme işine burnunu sokan elemanların deneyimleri ve becerileri.

Change Identification and Evolution:

- **Change Identification Process**: Yapılması istenen değişikliklerin bulunduğu change proposal'lar hazırlanır. Bu proposal'lar programdaki hataları, eklenmesi istenen fonksiyonları falan gösteriri.
- **Software Evolution Process**: Yeni sistem hazırlanır.



Yukarıdaki safhalar programın ömrü boyunca sürekli yinelenir.

Urgent Change Request:

Bazen acil değişiklik istekleri gelebilir. Bu durumda yazılım geliştirme süreçlerinin hepsi uygulanmayabilir. Bu acil değişiklik isteklerinin sebepleri;

- Hayati bir sistem hatasını düzeltmek
- Yazılımın kullanıldığı cevreyle ilgili çok önemli bir değişiklik
- İş dünyasıyla ilgili anında tepki verilmesi gereken bir gelişme

Emergency Repair:

Önce değişiklik istediği gelir, kaynak kod incelenir , değişiklikler yapılır ve sistem teslim edilir. Analiz, tasarım ,test gibi aşamalar yok.

System Re-Engineering:

Bazen mevcut sistemin geliştirilmesi çok zor olabilir.Eski sistemleri anlamak ve değiştirmek özellikle zordur çünkü performansı artırmak için yapısallıktan ve kodun anlaşılabilirliğinden feragat edilmiş olabilir. Yapılan değişikliklerle sistemin mimarisi bozulmuş olabilir.
Bu gibi durumlarda sistemin yeniden inşa edilmesi düşünülebilir. Bu sayade eski sistemi fonksiyonlarını değiştirmeden daha güncel bir programlama diline çevirebilir, yapısallığını artırabilir yada sisteme tasarım dökümanları ekleyebiliriz.

Advantages of Re-Engineering:

- **Reduced Risk**: Sistemi yeniden yazmak riskli bir iştir. Geliştirme sırasında çokça problem çıkabilir. Mesela oluşturulan sistem hatalı olabilir, gecikebilir.
- **Reduced Cost**: Maliyetler sistemin yeniden yazılmasından çok daha düşüktür.

Reengineering process activities:

- **Source code translation**: ilk önce Kaynak kod yeni bir programlama diline çevrilir.
- **Reverse engineering**: Program analiz edilir ve içinde bilgiler toplanır.
- **Program structure improvement**: Programda yapısal değişiklikler yapılır.
- **Program modularisation**:Birbiriyle alakalı bileşenler bir araya toplanır.
- **Data reengineering**: Programdaki değişikliklere göre veriler de değiştirilir.

Re-Engineering Cost Faactors:

- Re-engineering uygulanacak programın kalitesi
- Re-engineering için gereken toolların bulunup bulunmaması.
- İşlem sonrasında değiştirilmesi gereken veri miktarı.
- Bu işlemi yapacak uzman çalışanların verlığı-yokluğu.

Configuration Management

Yazılımlar değiştikçe yeni verisyonları üretilir. Yazılımların neden değiştiğini zaten biliyoruz ama yine de yazmış.

- Başka işletim sistemlerine, donanımlara uyum sağlamak
- Yeni özellikler eklemek
- Değişik kullanıcı isteklerine cevap vermek için

Not: Nedense repair'den bahsetmemiş hiç. Acaba hata düzeltilmesi yeni versiyon sayılmıyor mu?

Confuguration Management(CM) sistemin gelişimini yönetmeyi amaçlar. Buna ihtiyaç duyulmasının sebebi.

- Değişiklik yapmak bir takım işidir, düzenlenmesi gerekir.
- CM harcanan parayı ve zamanı kontrol etmeye yarar.

Bazen bir sistemin birden fazla versiyonu kullanımda olabilir, aynı anda birden fazla versiyonu geliştiriliyor olabilir. Bu durumda işlerin karışmaması için CM gerekir.Mesela sistemin yanlış bir versiyonu değiştirilebilir, kullanıcıya yanlış bir sistem verilebilir, yada kaynak kod iyice birbine girebilir, değişikliğin nerde yapıldığı karıştırılabilir.

Kısaca CM gerekli değişikliklerin ne şekilde yapılacağını ve kaydedileceğini belirten standartlar getirir. Bu yüzden qualty management 'ın (kalite yönetiminin)bir parçası olarak da kabul edilir.

CM Standarts:

CM belirli standartlara dayanmak zorundadır. Bu standartlar bileşenlerin nasıl tanımalanacağını , değişikliklerin nasıl yapılacağını, yeni sistemlerin nasıl organize edileceğini belirlemelidir.
Bu standartlar kurum dışındaki standartlara da dayanabilir(IEEE gibi).
Standartlar tanımlanırken genelde waterfall modeli esas alınmıştır, evolutionary devolopment için yeni standartlar tanımlanması gerekebilir.

Concurrent(iç içe geçmiş) Development and Testing:

Yeni sistem bileşenlerinin teslim edilmesine karar verildiği zaman bu bileşenlerden sistemin yeni versiyonları yapılır. Bu aşamada bileşenlerin derlenmesi ve birbirlerine bağlanması gerekir.
Bu yeni sistem daha önceden tanımlanmış test verileriyle test edilir.
Test süresinde hatalar tespit edilirse bunlar kayıt altına alınıp geliştirim ekibine bildirilir. Hatalar onarıldığında işlem tekrarlanır. Hata yoksa sistem teslim edilir.

Frequent System Building:

Incremental Development için farklı bir CM yaklaşımı geliştirilmiştir. Bu yaklaşıma göre nerdeyse her gün sistem component'lar bir araya getirilerek oluşturulur. Bileşenlerin geliştirilmesinde bir teslim saati belirlenir. Bu süre dolduğunda geliştirme işi tam olarak bitmemişse bile ana işlevler yerine getirilmelidir ki test edilebilsin. Böylece:

- Hatalı bileşenler bütün sistemi iflas ettirir. Böylece bileşenleri geliştirenler hata yapmamak konusunda daha fazla baskı altına alınmış olunur ("breaking the build")
- Bileşenlerin etkileşimiyle ilgili sorunlar geliştirme sürecinin erken safhalarında ortaya çıkar.
- Bileşen geliştiricileri hatalı bileşenleri sistem testine sokmak istemeyecekleri için (rezil olmamak için) unit testing daha dikkatli yapılmış olur böylece system testing de daha az hata çıkar.
- Sistemin hergün inşa edilmesi daha sağlam, ciddi bir CM yapılmasını gerektirir.Çünkü ortaya çıkarılan hataların ve yapılan düzeltmelerin kaydını tutmak gerekir ayrıca bu yöntemle çok sayıda versiyon ortaya çıkar.

Confuguration Management Planing

Yazılım geliştirirken ortaya çıkan her ürünün yönetilmesi lazım.Bu ürünler:

- Analiz dökümanları
- Tasarım dökümanları
- Program
- Test verileri
- Kullanıcı katoloğu

Olabilir. Geniş bir yazılım sistemi için çok fazla döküman ortaya çıkar. Bu sebepten bütün bu dökümanların belli bir sistematiğe göre yönetilmesi gerekir bu sistematik de CM'dir. CM planın belli parçalara bölünür.

- Hangi dökümanlar CM de yer alacak, bunu açıklayan bir tablo hazırlanır.
- CM sürecinden kimin sorumlu olduğu belirtilir.
- Değişiklik ve versiyon yönetimi için uyulacak standartlar belirlenir.
- CM için hangi tool'lar ne şekilde kullanılacak bu belirtilir.
- CM veri tabanının yapısı, bu veri tabanıyla ve içereceği verilerle ilgili bilgiler tanımlanır.

Confuguration Item Specification:

Büyük yazılım sistemlerinde çok sayıda kaynak kod, tasarım dökümanı, test verisi gibi bilgiler vardır ve genellikle bu bilgiler

- Farklı farklı kişiler tarafından oluşturulmuş olabilir
- Belirgin isimlere sahip olmayabilirler.
- Yazılım geliştirme süresince değişikliğe uğramış olabilir.

Bu karışıklığı önlemek ve istenen veriye çabuk ulaşabilmek için CM içerisindeki nesneler için bir isimlendirme (identification) şeması hazılamak gerekir. İsimlendirme yaparken hiearşik bir yapı kullanmak esnek bir çözümdür.
(müzik/mp3/yabancı/blackmores_night gibi)

The Configuration Database:

CM'le ve CM'in içerdiği birimlerle ilgili tüm veriler bir veritabanınd tutulur. Yalnız bu veritabanıyla ilgili bazı özellikler olması gerekir.
Veri tabanı:

- Sorgulama yapmaya uygun olmalıdır (kimin elinde sistemin hangi versiyonu var, hangi versiyonun gereksinimleri nelerdir gibi)
- Geliştiren sistemle bağının olması iyi olur yani projede yapılan eklentiler değişiklikler veri tabanına otomatik olarak aktarılabilir.
- CASE tool'larla birebir bağlantılı olabilir.
- Ya da kullanması kolay, ucuz ve esnek olması için kendi başına çalışabilir (otomatik değil, babadan kalma yöntemle güncellenir)

Change Management:

Şöyle şeyler yapılır:

- Bir değişiklik istek formu hazırlanır
- Değişiklik isteği incelenir
- Uygun bir değişiklik isteğiyse süreç devam eder
- Ne gibi değişikliler yapılmalı incelenir.
- Değişikliğin maliyeti hesaplanır
- Değişikliğin yapılıp yapılmamasına karar verilir. Yapılacaksa devam edilir.
- Değişiklik sistemin kalitesini düşürmeyecek şekilde yapılır, eğer kalite düşüyorsa düzeltmeler yapılmaya devam edilir.
- Yeni versiyon oluşturulur

Change Request Form:

Bu form:

- Yapılması istenilen değişikliği,
- Değikliği kimin istediğini,
- Değişiklik isteğinin sebebini,
- Değişikliğin aciliyetini(isteği yapana göre) belirtmelidir.
- Değişikliğin istendiği tarih
- Değişikliğin beklenen maliyeti, harcanacak zaman tahmini
- Değişikliğin ne şekilde yapılacağı
- Bu değişiklikten etkilenmesi beklenen bileşenler gibi şeyleri içerir.

Change Control Board:

Hangi değişikliğin yapılacağına karar verecek olan guruba *change control* board denir. Bu gurup karar verirken işin teknik kısmına bakmaz. Değişikliğin ekonomik, stratejik etkilerine bakar. Bu gurupta karar verebilmek için yetkili insanlar, müşteri temsilci falan olmalıdır. Bazen teknik bilgi vemek için mühendisler de olabilir.

Derivation History:

Yapılan değişikileri kayıt altında tutmak gerekir.Bu kayıtlarda

- Yapılan değişikliği
- Değişikliği kimin yaptığı
- Değişikliğin sebebi
- Değişikliğin yapılmasıyla etkilenen kısımlar gibi bilgileri içermelirdir.

Eğer standart bir gösterim tanımlanırsa bu kayıtlar kaynak kod içinde de bulunabilir hatta abartıp kaynak koda bu bilgileri otomatik olarak koyacak toolar kullanılabilir.

Version and Release Management:

Sistem büyüdükçe ortaya çıkan versiyonlar karmaşıklaşabilir. Bunu engellemek için hangi versiyonun ne içerdiğini belirten bir şema oluştulur, yeni bir versiyon oluşturulurken bir plana göre davranılır, versiyon konrtol sisteminin doğru düzgün çalıştığına emin olunur ve dağıtılacak release ler planlanır.

- **Versiyon**: Sistemin, diğer örneklerinden belli farkları bulunan örneği.
- **Variant**: Bir verisyonla işleyiş olarak aynı olan ama yine de küçük farkları olan sistem örenekleri(mesela sistemin farklı bir yazılım veya donanımla birlikte çalışması için değiştirilmiş hali)
- **Release**: Her versiyon piyasaya sürülüp, kullanıcıya dağıtılmaz. Release ler dağıtılır.

Version Identification:

Sistemin versiyonlarına, bu versiyonları rahat takip edebilmek için karmaşayı engelleyecek şekilde, belli bir sisteme göre isimler verilebilir.
İki şekli vardır.

- **Version Numbering**: Anlamlı olmayan bir numaralama sistemidir. *V 1.4.5* gibi birşey olur. Eğer *V 1.4.5* üzerinde de bir değişkil yaparak iki farklı versiyon oluştutulacaksa bunlardan biri *V 1.4.5.1* diğeri *V 1.4.5.2* diye adlandırılır. (Coloborationlardaki mesaj isimlerine benziyo)
- **Attribute Base Identification**: Tarih, değişikliği yapan gibi özelliklerin versiyon isimlendirmede kullanılmasıdır. Bu şekilde her isim unic olmuş olmaz ama versiyonların ismine göre sorgu yapabilmeyi sağlar.

Release(sürüm) Management:

Yeni sürüm çıkartma kararından, bu sürümün oluşturulması ve dağıtılmasından *relaese managerler* sorumludur.
Release managerlar her kullanıcının elinde her sürümü varmış gibi hareket edemezler. Hangi sürüm hangi sürümü gerektiriyor, eski sürümleri kullanan kullanıcıların durumu yeni sürümle ne olacak bunları düşünmeleri gerekir.

Bir release sadece çalışabilir kod değildir. İçinde;

- Kullanıcnın yapması gereken ayarları gösteren dosyalar(sound setup gibi)
- Gerekebilecek veri dosyaları.(Windowstaki örnek resimler gibi, hazır masaüstü arka planı gibi şeyler heralde)
- Bir install programı
- Sistami tanıtan belgeler(beni oku)
- Release le ilgişkili diğer ürünler(sql server mesela)

Release Decision Making:

Yeni bir sürüm çıkarma kararını vermek önemli bir iştir çünkü sürüm çıkarmak ve dağıtmak masraflır. Bu kararı verirken;

- Pazarlama stratejileri
- Sistemin teknik kalitesi
- Kullanıcının beklentileri gibi şeyler göz önüne alınmalıdır.

Release Creation:

Yeni sürüm için gereken tüm verileri ve belgeleri bir araya toplamakgerekir.
Farklı donanım ve yazılımlar için kurulum ve ayar programları gerekebilir.
Sürüm hazırlanırken hangi veriler ne şekilde kullanıldı bunları kayıt altına almak gerekir ki gerektiğinde aynı sürüm bir kez daha basılabilsin.