Giriş

Yazılım sektörü yıllardan beri kan kaybediyor. Ama artık taze kan bulundu ve hastalığın tedavisi kolaylaştı. Çözüm çevik süreçler!

Günümüze kadar uzanan süreçte yazılım sektöründe yapılan projeler nereye varacağı belli bile olmayan büyük maceralar haline gelmiştir. Bunun başlıca sebebi kullanılan yazılım yöntemlerinin gereksinimlere cevap verecek yapıda olmamasıdır. Çevik süreçler bu sorunu çözecek nitelikte. Bu bölümde

- yazılım yaparken hedefin ne olduğunu,
- çevikliğin ve çevik sürecin ne olduğunu,
- çevik manifesto ve prensiplerinin ne anlama geldiğini,
- çevik sürecin diğer yazılım metotlarına kıyasla hangi farklılıkları beraberinde getirdiğini,
- hangi çevik süreç türlerinin mevcut olduğunu,
- bir çevik süreç olan Extreme Programming in ne olduğunu,
- Extreme Programming in hangi değer, prensip ve teknikler üzerine kurulu olduğunu

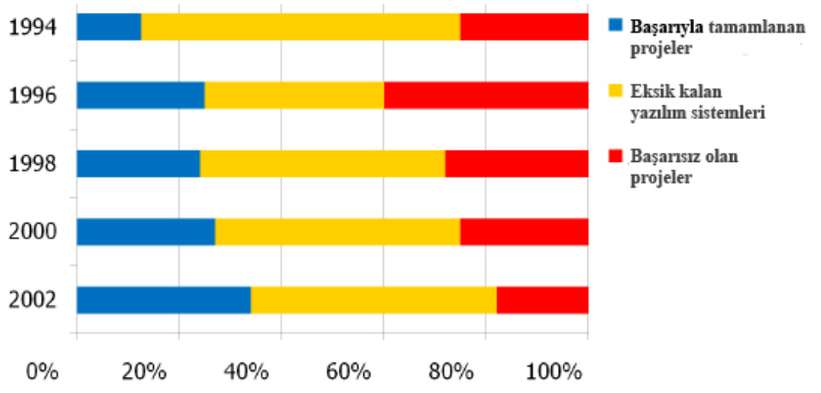
yakından inceleyeceğiz.

Hedef

Ana hedef, müşteri gereksinimlerini tatmin eden, sabit bir bütçe ve belirli bir zaman diliminde, hatalardan arındırılmış ve müşterinin piyasadaki rekabet etme yeteneğini kuvvetlendirecek bir yazılım sistemi geliştirmektir. Yazılım ve proje geliştirme süreçleri bu hedefe ulaşmak için kullanılan metotları ihtiva eder.

Hedefin net bir şekilde tarifinin yapılabilmesine rağmen, günümüzde uygulanan birçok projenin başarıyla tamamlanamadığını ya da oluşturulan yazılım sistemlerinin müşteri gereksinimlerini tatmin etmediğini görmekteyiz. Bunun sebepleri:

- Geleneksel yazılım metotları proje başlangıcında müşterinin tüm gereksinimlerini tespit eder veya tespit ettiğini düşünür ve akabinde implemente eder. Yazılım süreci zaman içinde değişikliğe uğrayan müşteri gereksinimlerine ayak uyduracak şekilde organize edilmemiştir. Müşteri tarafından istenen değişiklikler hoş karşılanmaz. Bu sebepten dolayı geliştirilen yazılım sisteminin müşteri gereksinimlerini %100 tatmin etme şansı yok gibidir.
- Müşteri istekleri doğrultusunda yapılmak zorunda kalınan değişiklikler proje maliyetlerini yükseltir, çünkü bu beraberinde basit olmayan yapısal değişiklikler getirebilir.
- Proje yöneticilerinin programcılardan insan üstü beklentileri, projenin büyük bir bölümünün sorumluluk ve riskini omuzlarında taşıyan bu bireylerin fazla mesai yapmalarına ve ruhen ve bedenen yorulmalarına sebep vermektedir. Motivasyonu düşük olan programcıların yaratıcı yönleri azalmakta ve kodun kalitesi buna orantılı olarak düşmektedir. Bu programlardaki hata oranının artması anlamına gelmektedir.
- Projelerde takım çalışması ve bireyler arası iletişimin kuvvetlendirilmesi desteklenmez. Her programcı yazılımını yaptığı program parçasından sorumlu olduğu için projeden ayrılan programcılar proje için risk haline dönüşür.
- Proje başlangıcında müşteri gereksinimleri en son detayına kadar tespit edilir. Ayrıca yazılım sistemi için teknik mimari ve uygulanacak tasarım belirlenir. Bunlar proje başlangıcında çok zaman kaybedilmesine neden olur. Ayrıca bir zaman sonra tasarımın implemente edilen gereksinimlere cevap vermez hale geldiği ve yapılan her değişiklik sonucunda tasarım çıkmaza girdiği görülür.
- Oluşturulan ve uygulanan proje planı bir zaman sonra geçerliliğini yitirir. Plana mutlaka uyulması gerektiği düşüncesi, plan dışına çıkılmasını önlemek için fazla mesai yapılmasını zorunlu kılar.



Resim 1.1 Standish Group tarafından yapılan Chaos Study istatikstikleri

Bu ve bunun gibi sıralayabileceğimiz bir çok nedenden dolayı projelerin büyük bir bölümü başarısızlığa mahkum olmaktadır. Bahsedilen sorunları ortadan kaldırmak mümkün olabilmelidir. Bunun cevabını çevik süreçler verir.

Çevik Süreç

Türk Dil Kurumu'nun online sözlüğünde çevik kelimesi şu şekilde tanımlanmaktadır:

```
Kolaylık ve çabuklukla davranan, tetik, atik
```

Çevik süreçler yazılım sektöründe kullanılan mevcut geleneksel yöntemlere (örneğin Rational Unified Process – RUP ya da V-Model) alternatif olarak geliştirilmiş modern ve bürokrasiye mesafeli yazılım yöntemlerini ihtiva ederler. Çevik yazılım (Agile Development) bir yandan bir değer sistemini, diğer yandan da somut yazılım metotlarını içerir. Çevik yazılıma yazılım sektöründe yeni bir filozofi akımı ya da yeni bir yazılım metamodeli olarak bakabiliriz. Bu yeni filozofinin somut örnekleri arasında Extreme Programming, Scrum ve Lean Development bulunmaktadır. Bu kitabın konusu Extreme Programming (XP) dir.

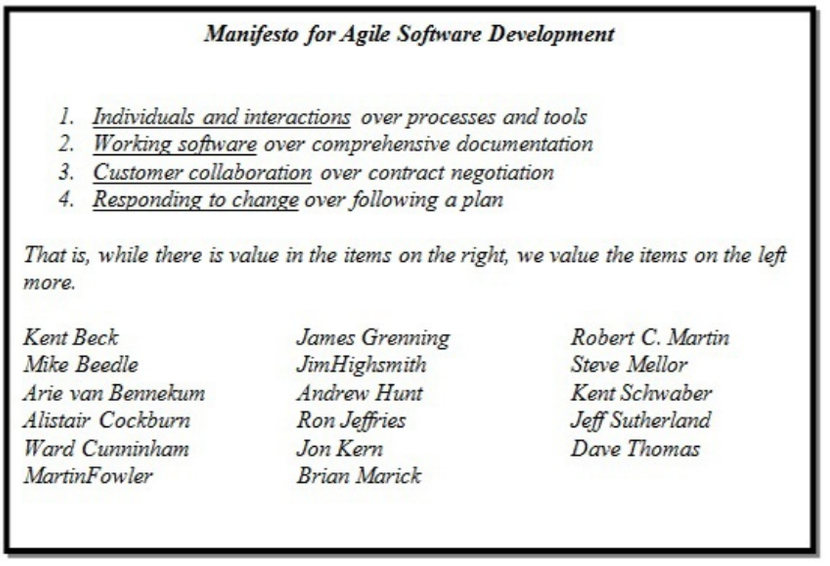
Çevik Sürecin Geçmişi

2000 senesinde Kent Beck ve arkadaşlarının yer aldığı bir toplantı düzenlendi. Kent Beck ve arkadaşları belli bir süredir çevik yazılım metotlarını projelerde uyguluyorlardı ve fikir alışverişi için böyle bir toplantının faydalı olacağını düşündüler. Smaltalk gibi nesneye yönelik modern programlama dilleri sayesinde iteratif yazılım metotları geliştirilmiş ve değişik ekipler tarafından uygulanmaktaydı. Smalktalk ile başlayan bu yeni akım doksanlı yılların ortalarına doğru iyice kuvvetlenmişti. Çevik (Agile) kelimesi ilk kez bu toplantıda değişik iteratif yazılım metotlarını bir çatı altında toplamak için kullanıldı.

Toplantıya katılanlar tarafından bir manifesto (Agile Manifesto) hazırlandı. Yeni bir yazılım filozofisi doğmuştu. Çevik sürecin mimarları olan katılımcılar kısa bir zaman sonra Agile Alliance organizasyonunu kurdular. Bu organizasyon ile çevik süreç ve çevik yazılımın desteklenmesi, geliştirilmesi ve uygulanması hedef alındı. Konuyla ilgilenen kurum, kuruluş ve şahıslar için bu organizasyon merkezi bir buluşma noktasıdır. Agile Alliance organizasyonu ilgilenenlere çevik süreç ve çevik yazılım hakkında geniş bir literatür sunmakta ve her yıl bu konuda konferans düzenleyerek, katılımcıları bilgiledirmektedir.

Çevik Manifesto (Agile Manifesto)

2000 senesinde Kent Beck ve 16 arkadaşı tarafında aşağıda yer alan çevik manifesto oluşturulmuştur.



Manifesto for Agile Software Development

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck	James Grenning	Robert C. Martin
Mike Beedle	JimHighsmith	Steve Mellor
Arie van Bennekum	Andrew Hunt	Kent Schwaber
Alistair Cockburn	Ron Jeffries	Jeff Sutherland
Ward Cunninham	Jon Kern	Dave Thomas
MartinFowler	Brian Marick	

Türkçesi:

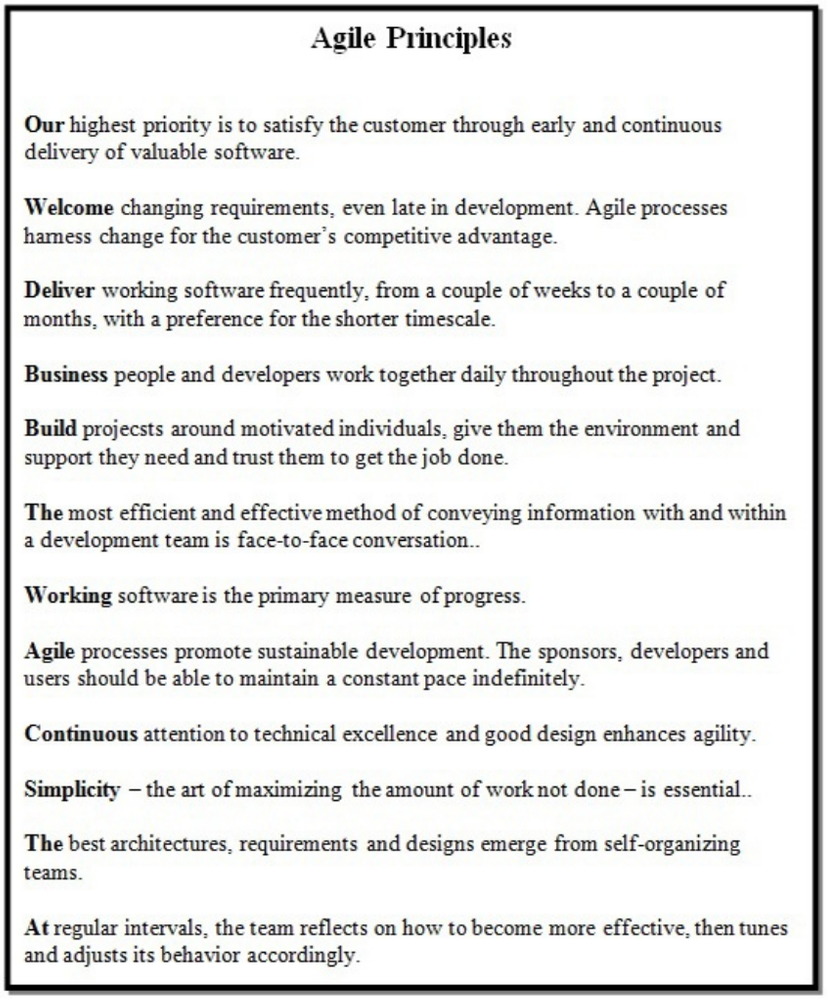
1. Kişiler ve iletişim süreç ve araçlardan önce gelir.
2. Çalışır durumda olan program detaylı dokümantasyondan daha önceliklidir.
3. Müşteri ile beraber çalışmak sözleşmelerden ve anlaşmalardan daha önceliklidir.
4. Değişikliklere ayak uydurmak bir planı takip etmekten daha önemlidir.

Sağ bölümde yer alanlar (altı çizili olmayanlar) değer taşımakla beraber, sol bölümde altı çizili olan değerler bizim için daha kıymetlidir.

Bu manifesto ile çevik sürecin bir değer sistemine sahip olması ve geleneksel yazılım metotlarından elde edilen tecrübeler doğrultusunda daha pratik ve çevik bir yapıda olması gerektiği dile getirilmiştir.

Çevik Prensipler (Agile Principles)

Çevik manifestoda yer alan ifadeler on iki prensip ile somut hale getirilmekte ve açıklanmaktadır. Bunlar:



1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

En önemli öncelik erken ve sürekli olarak kullanılabilir programlar oluşturarak, müşteriyi tatmin etmektir.

Bu prensip ile aslında programın müşteri tarafından talep edildiğini ve müşteriyi sadece istekleri doğrultusunda geliştirilmiş ve çalışır durumda olan bir programın tatmin edebileceği dile getirilmektedir. Peki müşteri nasıl tatmin edilebilir? Bunun için proje ekibinin proje başlangıcından itibaren ve sürekli olarak çalışır durumda olan programlar oluşturarak, müşteriye sunması ve görüşlerini alması gerekmektedir. Bu sayede müşteri inceleyebileceği ve çalışır durumda olan bir program aracılığıyla gereksinimlerinin ne oranda implementasyonla örtüştüğünü kontrol edebilir. Gereksinimlerin değişmesi ya da müşterinin istekleri dışında bir yazılım yapılması durumunda müşteri yazılım sürecine müdahale edebilir ve gerekli değişiklikleri talep edebilir. Bu durum proje sonunda müşteri gereksinimleri ile yüksek oranda örtüşen bir programın oluşmasını sağlar.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Yazılımın ilerleyen dönemlerinde gelse bile talep edilen değişiklikler hoş karşılanmalıdır. Çevik süreçler değişiklikleri müşterinin rekabetteki avantajını korumak ve sağlamak için kullanırlar.

Geleneksel yazılım metotlarının uygulandığı projelerde mimarinin ve planlamanın büyük bir bölümü implementasyon öncesi oluşturulur. Yazılım, hazırlanan planlardan sapmadan gerçekleştirilir. Bu müşteri tarafından talep edilen değişikliklerin göz ardı edilmesi anlamına gelmektedir. Böyle bir sürecin sonunda müşteri isteklerini tatmin etmeyen ve müşterinin piyasadaki rekabet kabiliyetini sınırlayan programlar

oluşur. Bunu engellemek için her zaman müşteriden gelen değişiklik taleplerinin implementasyon esnasında dikkate alınması gerekmektedir. Değişiklik ne zaman gelirse gelsin, implementasyon bu değişiklik doğrultunda adapte edilebilmelidir.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale.

Kısa sürelerde (birkaç haftadan, birkaç aya kadar sürebilen zaman dilimlerinde) çalışır programlar ortaya koy. Seçim zaman diliminin kısa tutulması yönünde olmalıdır.

Çevik süreçlerde programlar hem iteratif hem de inkrementel olarak geliştirilir. Bir iterasyon yazılım için gerekli tüm safhaları ihtiva eden belirli bir zaman biriminde, örneğin 2 hafta implementasyonun gerçekleştirildiği süreçtir. Program ayrıca inkrementel oluşturulur, çünkü programcı ekip her iterasyonda müşterinin seçtiği gereksinimlere konsantre olarak programı yavaş yavaş oluşturur. Her iterasyon ardından program müşteriye sunularak, görüşleri alınır.

4. Business people and developers work together daily throughout the project.

Müşteri ve programcılar proje süresince beraber çalışırlar.

Çevik projelerde müşteri ile proje ekibinin beraber çalışması doğaldır. Programdan olan beklentileri en iyi müşteri bilebileceği için sürekli müşteriden geribildirim alınması gerekmektedir. Bunun en kolay yolu müşteri ile proje ekibinin beraber çalışmasıdır.

Proje ekibi, özellikle programcılar müşteri tarafından dile getirilen gereksinimlerin (requirement) fizibilitesini (yapılabilirlik) araştırırken, her gereksinim için implementasyon zamanını tahmin ederler. Bu doğrultuda müşteri kendi tanımlamış olduğu gereksinimlere bir öncelik sırası vererek, hangi gereksinimlerin öncelikli olarak implemente edilmesi gerektiğini tayin eder. Programcılar bu konularda müşteriye yardımcı olarak, gereksinimlerin daha somut hale getirilmelerini sağlarlar. Bunlar genellikle birkaç cümleden oluşan kullanıcı hikayelerine (user story) dönüştürülür. Sürekli müşteri ve programcılar arasındaki diyalog yanlış anlaşılmaları ortadan kaldırır. Bu yüzden müşterinin her gün programcıların erişebileceği bir mesafede olması gerekmektedir.

Geleneksel yazılım metotlarında bunun böyle olmadığını görmekteyiz. Adeta programcı ekip ve müşteri arasına görünmez bir duvar örülür. Projenin ilerleyen safhalarında müşteri tarafından talep edilen değişiklikler olumlu karşılanmaz, çünkü bu proje başlangıcında yapılan anlaşmalara ters düşmektedir. Başlangıçta ne yapılmasına karar verildiyse, programcılar rahatsız edilmeden mevcut gereksinimlerin implementasyonuna odaklanabilmelidirler. Tabii böyle bir sistem gölgesinde oluşan programın müşteri gereksinimlerini ne ölçüde tatmin edebileceğini düşünebilirsiniz.

5. Build projecsts around motivated individuals, give them the environment and support they need and trust them to get the job done.

Projelerin motivasyonu yüksek bireyler tarafından yapılmasını sağla, onlara ihtiyaç duydukları ortamı ve desteği ver ve işi bitirebileceklerine inan.

Çevik projeler bireylerden oluşur ve her programcı kendi bireysel karakteri ile takımın bir parçasıdır. Ekip elemanlarının değişik karakterde olmaları doğaldır. Programcıların onlara duyulan güvenin hissettirilmesiyle motivasyonları artırılır. Onların sorumluluk almaları sağlanarak, öz güvenlerinin pekişmesi sağlanır. Programcılar birbirlerine yardım ederler. Onlar arasında kıdem farkı yoktur. Bilen bilmeyene öğreterek, kısa bir zamanda aynı teknik seviyeye gelmeleri sağlanmış olur.

6. The most efficient and effective method of conveying information with and within a development team is face-to-face conversation.

Bilgi alışverişinde en verimli ve efektif yöntem takım içinde yüz yüze konuşmaktır.

Çevik projelerde bilgi alışverişi yüz yüze gerçekleşir, çünkü bilginin transfer esnasında en az hasara uğradığı yöntem budur. Programcılar arasındaki kişisel konuşmalar güveni artırır ve yanlış anlaşılmaları ortadan kaldırır. Sorunlar hemen açıklığa kavuşturulabilir.

7. Working software is the primary measure of progress.

Çalışır durumda olan program ilerlemenin ana göstergesidir.

Almanca'da kullanılan bir deyim vardır: "torbada kedi satın almam!". Torbanın içini görmeden satın aldığımızda, beklentimiz haricinde (beklentimiz örneğin bir horoz olabilir) bir şeyle (örneğin kedi – deyimde değersiz anlamında kullanılıyor) karşılaşabiliriz. Bu durum bir program siparişi veren müşteri içinde geçerlidir. Müşteriye kısa aralıklarla çalışır durumda olan lakin henüz tamamlanmamış bir program sunulduğunda, müşteri, mevcut programın kendi gereksinimlerini tatmin edecek durumda olup, olmadığını kontrol edebilir. Program yalan söylemez. Bu yüzden müşteri çalışır durumda olan programı kullandıktan sonra torbanın içine bakmış ve torbanın içindeki şeyin kedi mi yoksa horoz mu olduğunu görebilmiştir.

8. Agile processes promote sustainable development. The sponsors, developers and users should be able to maintain a constant pace indefinitely.

Çevik süreçler etkili yazılım yöntemlerini destekler. Müşteri, programcılar ve kullanıcılar sabit bir tempoda beraber çalışabilmelidirler.

Çevik projelerde sabit bir çalışma temposunun oluşturulması büyük önem taşımaktadır. Programcılar arasında görev eşit bir şekilde paylaştırılır. Fazla mesai yapılması hoş karşılanmaz. Bu ayrıca çalışanların motivasyonu olumlu etkiler. Proje ilerledikçe iş azalır. Sonradan programcıları kötü sürprizler beklemez, çünkü ortada iyi test edilmiş ve çalışır durumda olan bir program vardır.

9. Continuous attention to technical excellence and good design enhances agility.

Devamlı teknik mükemmelliğe özen gösterilmesi ve iyi tasarım çevikliği kuvvetlendirir.

Çevik projelerde kalite beklentisi yüksektir. Yazılım temin edilen en iyi araç ve yetenekli programcılarla gerçekleştirilir. Bu süreçte programın tasarımı sürekli optimize edilir. Programcılar yeniden yapılandırma (refactoring) esnasında buldukları tasarım hatalarını hemen giderirler.

10. Simplicity – the art of maximizing the amount of work not done – is essential.

Sadelik (basitlik) esastır.

Mümkün olan en basit tarzda implementasyonu gerçekleştirmek, programın karmaşıklığını azaltır. Karmaşıklık oranı düşük olan bir uygulamanın bakımı ve geliştirilmesi kolaydır. Programcılar yazılım esnasında sadece kendilerinden o anda olan beklentiyi tatmin edecek kadar kod yazarlar. Bu gereksiz kodun oluşmasını engeller ve test edilebilir bir yapının ortaya çıkmasını sağlar.

11. The best architectures, requirements and designs emerge from self-organizing teams.

En iyi mimariler, gereksinimler ve tasarımlar kendi kendine organize olabilen takımlardan çıkar.

Çevik takımlar kendi başlarına organize olabilme özelliğine sahiptir. Böyle takımlar görevleri kendi aralarında eşitçe paylaşırlar. Takımlar ve bireyler arasındaki görüşmeler ve bilgi alışverişi sonucunda mimari ve tasarım gözden geçirilir ve optimize edilir. Ayrıca bireyler arası yapılan konuşmalar müşteri tarafından dile getirilen gereksinimlerin açıklığa kavuşturulmalarını ve daha iyi anlaşılmalarını sağlarlar.

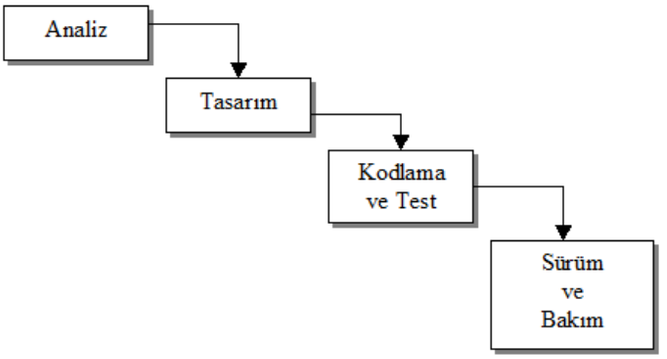
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Belirli zaman dilimlerinde takım daha nasıl verimli olabileceği konusunda kendini sorgular ve edindiği bilgiler doğrultusunda çalışma tarzını adapte eder.

Çevik takımlarda bilgi alışverişi ve devamlı öğrenme çok önemlidir. Belirli zaman aralıklarıyla takım çalışanları bir araya gelerek, fikir alışverişinde bulunurlar ve çalışma yöntemlerini sorgularlar. Edinilen tecrübeler doğrultusunda yazılım sürecinde adaptasyonlar gerekebilir. Nihai amaç en verimli çalışabilmek ve müşteri tarafından kabul görmüş bir program oluşturabilmektir.

Çevik Sürecin Farkı

Yazılım geliştirme süreci analiz, tasarım, kodlama, test, sürüm ve bakım gibi safhalardan oluşur. Geleneksel yazılım metotlarında bu safhalar şelale modelinde olduğu gibi doğrusal (linear) olarak işler. Her safha başlangıç noktasında bir önceki safhanın ürettiklerini bulur. Kendi bünyesindeki değişikler doğrultusunda teslim aldıklarını bir sonraki safhanın kullanabileceği şekilde dönüştürür.



Resim 1.2 Şelale modeli

Şelale modelinin özelliklerini şu şekilde sıralayabiliriz:

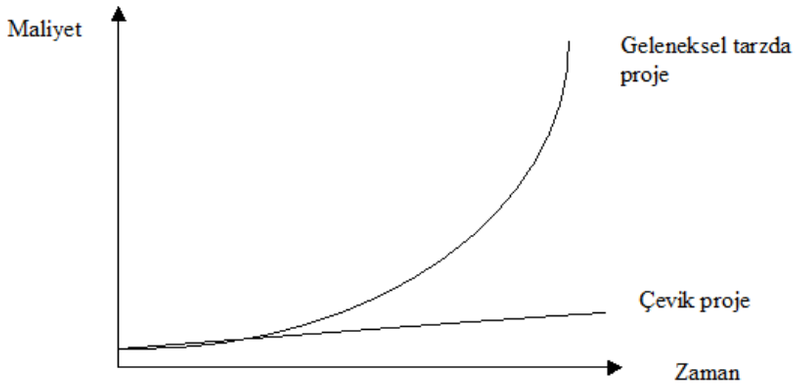
1. Şelalenin her basamağında yer alan aktiviteler eksiksiz olarak yerine getirilir. Bu bir sonraki basamağa geçmenin şartıdır.
2. Her safhanın sonunda bir doküman oluşturulur. Bu yüzden şelale modeli doküman güdümlüdür.
3. Yazılım süreci doğrusaldır, yani bir sonraki safhaya geçebilmek için bir önceki safhada yer alan aktivitelerin tamamlanmış olması gerekir.
4. Kullanıcı katılımı başlangıç safhasında mümkündür. Kullanıcı gereksinimleri bu safhada tespit edilir ve detaylandırılır. Daha sonra gelen tasarım ve kodlama safhalarında müşteri ve kullanıcılar ile diyaloğa girilmez.

Bu modelin beraberinde getirdiği problemleri şu şekilde sıralayabiliriz:

1. Safhaların birbirinden kesin olarak ayrı tutulmaları gerçekçi değildir. Projelerde safhalar arasındaki bu sınırlar yok olabilir.
2. Teoride safhalar birbirlerini takip edeler. Projelerde bunun bazen mümkün olmadığını ve önceki safhalara geri dönülmek zorunda kalındığını görebiliriz.
3. Safhalar arası geri bildirim yetersizdir. Model değişikliğe açık değildir.
4. Müşteri gereksinimlerinin proje öncesi detaylı olarak kağıt üzerinde oluşturulması ilerde sorun yaratabilir. Müşteri gereksinimleri değişikliğe uğrayabileceği için yazılım sisteminin de yapısal değişikliğe uğraması kaçınılmaz olabilir. Böyle bir durum maliyeti artırır, çünkü yeni ve değişen gereksinimleri implemente edebilmek için modelde yer alan safhaların birkaç kere uygulanması gerekebilir.
5. Sistemin kullanılır hale gelmesi uzun zaman alabilir.
6. Başlangıçta yapılan hataların tespiti çok uzun zaman alabilir. Bu hataların giderilmesi maliyeti yükseltir.
7. Modül implementasyonları için zaman tahminleri proje planlarını oluşturan yöneticiler tarafından yapılır. Teknik bilgiye sahip olmayan şahıslar tarafından yapılan bu tahminler çoğu zaman doğru

değildir. Bu durum proje planlama sürecini negatif etkiler.

Proje başlangıcında her detayı göz önünde bulundurmak mümkün olmadığı için şelale modeliyle geliştirilen yazılım sistemlerinin müşteri gereksinimlerini tam tatmin etmediğini görmekteyiz. Bunun önüne geçebilmek için projenin başlangıç safhasında analiz için çok zaman harcanır ve müşteri gereksinimleri en ince detayına kadar tespit edilir. Aslında proje başlangıcında oluşturulan dokümanlar geçerliliklerini yitirmiştirler, çünkü müşteri gereksinimleri piyasa ve rekabet koşulları gereği değişikliğe uğramış olabilir. Ne yazık ki şelale modeli bunları dikkate almaz ve müşterinin talep ettiği değişiklikleri aza indirmeye çalışır. Bunun bir sebebi de sonradan gelen değişiklik taleplerinin maliyeti yükseltmesidir, çünkü bu durumda şelale modelinde yer alan safhaların birkaç kere uygulanması gerekebilir.

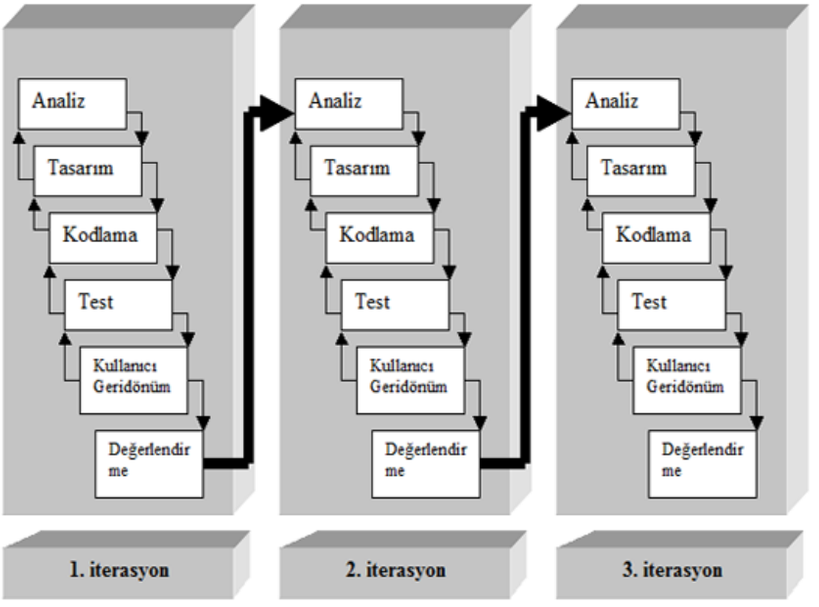


Resim 1.3 Geleneksel tarzda yapılan projelerde gerekli değişikliklerin yapılması maliyete zamanla yükseltir.

Bu çerçeveden bakıldığında proje sonunda oluşan program müşterinin güncel gereksinimlerini tatmin etmez durumdadır. Program daha çok müşterinin proje başlangıcında sahip olduğu gereksinimleri tatmin edecek şekilde tasarlanmıştır. Projelerin birkaç sene boyunca sürebileceğini düşünürsek, aslında bu süreç sonunda oluşan program güncel değildir.

Çevik süreçlerde durum farklıdır. Çevik süreç değişimi kabul eder ve onunla yaşamayı kolaylaştırmak için yeni yazılım metotları sunar.

Çevik süreçlerde iterasyon bazında çalışmalar sürdürülür. Her iterasyon bir ile dört haftalık zaman dilimlerinden oluşur ve şelale modelinde yer alan safhaları ihtiva eder. Aslında her iterasyona bir mini şelale modeli ihtiva ediyor diyebiliriz.



Resim 1.4 Çevik süreç iterasyon modeli

Çevik süreç modelinin avantajları şöyledir:

1. Çevik projelerde müşteri gereksinimleri (requirement) ve bu gereksinimlerin karşılığı olan uygulama paralel olarak gelişir. Müşteri projenin gidişatına her zaman müdahale etme yetkisine sahiptir. Bu uzun süren projelerde önem taşımaktadır, çünkü çoğu zaman proje başlangıcında müşteri kendi gereksinimlerini tam olarak bilmeyebilir. Zaman içinde oluşan ilk prototipler müşterinin kafasında nasıl bir sistem istediği hakkında daha net bir resmin oluşmasını sağlar. Projedeki geri bildirim mekanizmalarıyla müşteri gereksinimleri her zaman değişikliğe uğrayabilir.
2. Bu modelde geri bildirim merkezi bir rol oynamaktadır. Çeşitli safhalarda sağlanan geri bildirim ile projenin hangi durumda olduğu saptanır. Programcılar hazırladıkları testler aracılığıyla geri bildirim sağlayarak, implemente ettikleri komponentlerin hangi durumda olduklarını tespit ederler. Sürekli entegrasyon yapılarak programın hangi durumda olduğu geri bildirim olarak elde edilir. Müşteriye kısa aralıklarla programın yeni sürümü sunularak, geri bildirim sağlanır.
3. Proje başlangıcında detaylı dokümantasyon ve tasarım oluşturulmaz. Programcılar test güdümlü (Test Driven Developlement) çalışır ve oluşan tasarımı her yeni testle gözden geçirirler. Eğer tasarım yetersiz kalırsa, gerekli tasarım değişikliklerini ilerideki testlerin çıkmaza girmesini engellemek için uygularlar.
4. Her iterasyon başlangıcında müşteri tarafından dile getirilen gereksinimler analiz edilir ve implemente edilecek olanlar seçilir. Müşteri her gereksinim için bir öncelik sırası belirler. Öncelik sırası yüksek olan gereksinimler öncelikli olarak implemente edilir. Her iterasyon sonunda gerekli değerlendirmeler yapılarak, oluşan problemler tartışılır ve tekrar meydana gelmelerini engellemek için gerekli önlemler alınır.
5. Test güdümlü çalışıldığı için kod kalitesi çok yüksek olur. Gün boyunca programcılar kendilerine değişik bir programcıyı takım arkadaşı olarak seçerek (pair programming), implementasyonu gerçekleştirirler. Kısa bir zaman sonra programcılar arasındaki teknik bilgi aynı seviyeye ve her programcı programın herhangi bir bölümünde çalışacak hale gelir. Bu şekilde bir programcının kod hakkında bilgi monopolüne sahip olması engellenir.
6. Programcılar aktif olarak proje planlamasında yer alırlar. Onlar gereksinimlerin tespitinde müşteriye yardımcı olurlar ve zaman tahminlerinde bulunarak, proje planlaması için gerekli verilerin oluşturulmasını sağlarlar. Bu programcılara belirli bir sorumluluk yükler. Kendisine güvenildiğini bilen ve sorumluluk sahibi bir programcının öz güveni ve motivasyonu artar.
7. Çevik projelerde iyi bir çalışma ortamının ve temposunun oluşturulması fazla mesai yapılmasını engeller. Fazla mesai yapılmayacak diye bir kural yoktur. Lakin fazla mesai bir kural haline

gelmemelidir. Bu durum tüm ekibin motivasyonunu negatif etkiler. Hatalar genelde isteksizce yer alınan fazla mesailerde meydana gelir. Proje çalışanları sekiz saat olan ve fazla mesai yapılmayan iş günlerinde daha verimli olurlar.

8. Müşteriye kısa aralıklarla çalışabileceği bir sürüm sunulur. Program tamamlanmamış olsa bile, müşteri hazır bölümleri kullanarak, yaptığı yatırımın hızlı şekilde geri dönmesini sağlanır.
9. Programcılar ve müşteri arasında devamlı iletişim vardır. Programcılar soru ve sorunları müşteri ile paylaşarak, kısa sürede çözüm üretebilirler.
10. Müşterinin piyasadaki değişikliklere ve bununla birlikte rekabet ortamına ayak uydurabilmesi önemlidir. Çevik süreç hızlı reaksiyon göstererek, bu değişikliklere ayak uydurulmasını sağlar. Rekabete ayak uydurabilmek için hızlı reaksiyon gösterebilmek hayatı bir önem taşımaktadır.

Çevik Süreç Türleri

Zaman içinde bir metamodel olarak kabul edebileceğimiz çevik süreci implemente eden değişik çevik süreç türleri oluşmuştur. Bunların çoğu çevik manifestodan sonra oluşmuştur. Ama bazıları çevik manifesto öncesinde mevcuttu ve kullanılmaktaydı.

Önemli çevik süreç türlerini şunlardır:

- ***Scrum*** - Scrum seksenli yıllarda Kent Schwaber ve Jeff Sutherland tarafından geliştirilmiş bir çevik süreçtir. Scrum Rugby oyununda kullanılan bir terimdir. Oyuncular kısa bir süre için bir araya gelerek, bir sonraki oyun hamlesi hakkında fikir alışverişinde bulunurlar, yani kısa bir toplantı yaparlar. Scrum daha çok proje yönetim metotlarına konsantre olmaktadır. Yazılımın nasıl yapılması gerektiği hakkında detay ihtiva etmez. Birçok projede Scrum Extreme Programming (XP) ile kombine edilir.
- ***XP*** - Extreme Programming (XP) doksanlı yılların sonunda Kent Beck, Ron Jeffries ve Ward Cunningham tarafından Chrysler için yapılan bir proje sonrasında oluşmuş bir çevik süreçtir. XP Scrum dan esinlenilerek geliştirilmiş bir çevik süreçtir. XP Scrum ın aksine daha çok yazılım metotlarına konsantre olmaktadır. Bu sebepten dolayı Scrum ve XP bir projede kombine edilebilir.
- ***IXP*** - XP den doğan IXP nin (Industrial XP) amacı XP yi geliştirmek ve XP de yer alan metot ve tekniklerin daha büyük organizasyonlar için adapte etmektir.
- ***FDD*** - Jeff DeLuca tarafından doksanlı yılların sonunda geliştirilmiş bir çevik süreç türüdür (FDD = Feature Driven Development). FDD yazılım özelliği (feature, function) güdümlü çalışır. Sisteme yeni bir özellik kazandırılmadan önce detaylı bir tasarım çalışması yapılarak bu özelliği kapsayan mimarik yapı oluşturulur. Bu yüzden FDD daha çok tasarım odaklı işleyen bir çevik süreçtir.

Extreme Programming (XP)

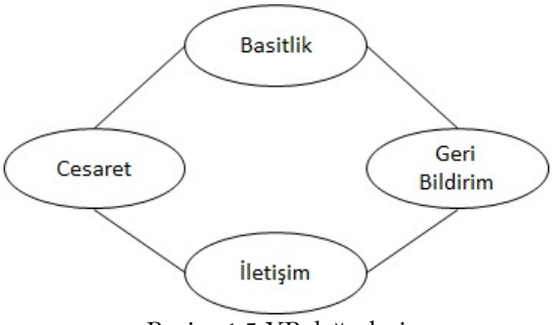
En popüler çevik süreçlerden (Agile Process) birisi XP olarak bilinen Extreme Programming dir. Kent Beck ve arkadaşları tarafından 1996 yılında Chrysler firmasında yapılan bir proje bünyesinde oluşan XP ihtiva ettiği basit ama bir o kadar etkili yöntemlerle yazılım sektöründe yeni bir rüzgarın esmesini sağlamıştır.

XP ile oluşturulan çevik süreçte müşteri ve gereksinimleri merkezi bir rol oynamaktadır. Yazılım esnasında XP ile tam belirli olmayan ve çabuk değişikliğe uğrayan müşteri gereksinimlerine ayak uydurulabilir. Bu konvansiyonel yazılım metotlarda mümkün değildir, çünkü proje öncesi müşteri gereksinimleri en son detayına kadar kağıda dökülmüştür. Oluşan bir dokümantasyon baz alınarak, yazılım gerçekleştirilir. Proje ilerledikçe müşteri tarafından yapılması istenen değişikliklerin maliyeti çok yüksek olacaktır, çünkü mevcut yapı (tasarım – design) istenilen değişiklerin yapılmasını engelleyebilir ya da yeni bir yapılanmaya gidilmesi gerekebilir. XP, kullanıldığı projelerde formalite ve bürokrasinin mümkün en az seviyeye çekilmesine önem verir. Çevik olabilmek için az yükle yola çıkılması gerekmektedir. Bu yüzden proje öncesi geniş çapta tasarım ve dokümantasyon oluşturulmasına izin

verilmez.

XP Değerleri

XP dört değer üzerine kuruludur: Basitlik (Simplicity), İletişim (Communication), Geri Bildirim (Feedback) ve Cesaret (Courage).



Resim 1.5 XP değerleri

XP nin özü bu dört değer içinde yatmaktadır. Bu değerler yaşandığı taktirde XP öğrenimi ve kullanımı kolaylaşır. Bu değerlerin geçerlilik bulmadığı ortamlarda XP nin uygulandığı söylenemez. XP ile verimli bir çevik süreç oluşturabilmek için bu değerlerin hepsinin kabul görmesi ve uygulanması gerekmektedir.

Bu değerlerin ne anlama geldiğini yakından inceliyeyim. XP basit yöntemler aracılığıyla sonuca ulaşmak ister, çünkü sadece bu şekilde hızlı ve düşük maliyetli projeler gerçekleştirilebilir. Bunun yanı sıra basit çözümlerle oluşturulan programın bakımı ve geliştirilmesi kolaydır. Basit çözümler kolay anlatılır ve adapte edilir. Bu zaman kazanılması anlamına gelmektedir.

Yazılımda en önemli konulardan birisi kalite kontrolüdür. XP projelerinde kalite kontrolü geri bildirim üzerinden sağlanır. Programcılar yazdıkları testlerden geri bildirim alarak kaliteyi sağlarlar. Kısa zamanlarla yeni sürüm oluşturularak müşteri ve kullanıcılardan geri bildirim aracılığıyla programın gereksinimleri tatmin edip, etmediği kontrol edilir. Yazılım esnasında sürekli entegrasyon yapılarak, programın en son durumu hakkında geri bildirim sağlanır. XP nin uygulanabilmesi için değişik katmanlarda geri bildirim mekanizmalarının oluşturulması gerekmektedir. İnsanlar için su ne ise, XP için geri bildirim odur.

Tüm proje çalışanlarının sürekli olarak aralarında iletişim kurmaları gerekmektedir. Bireyler arası yüz yüze görüşmeler büyük önem teşkil etmektedir. Sadece bu sayede sağlıklı bilgi transferi gerçekleşebilir. Böylece yanlış anlaşılmalar ve bilinmeyenler ortadan kaldırılır. Eğer takım içinde iletişim ve interaksiyon güçlü ise, dokümantasyon oluşturma ve kullanma gereksiz hale gelebilir. Bu zaman kaybını önler. Dokümantasyon yazılımı ve kullanımı başka sebeplerden dolayı gerekli olabilir, ama projenin başarısı için dokümantasyon öncelikli rol oynamamaktadır.

Basit çözümler, geri bildirim ve iletişim için cesaret gereklidir. Bu saydığımız değerler bireyler arası interaksiyonu ve iletişimi artıracağı için bireylerin kendi iç dünyalarını terk edip, takımın bir parçası olmalarını kolaylaştırırlar. Bu kişisel gelişmeyi sağlar ve temelinde kişisel cesaret yatar.

XP Prensipleri

XP değerlerinden yola çıkarak on beş XP prensibi oluşturulmuştur. Bunlar:

1. Rapid Feedback

Hızlı geri bildirim

Sık ve hızlı geri bildirim edinmek, projenin gidişatını olumlu etkiler. Geri bildirim sayesinde yanlış anlaşılmalar ve hatalar ortadan kaldırılır.

2. Assume Simplicity

Basitliği tercih etmek

Basit çözümler kolay implemente edilir ve kısa zamanda oluşturulur. Bu geri bildirimin de hızlı bir şekilde gerçekleşmesini sağlar. Basit çözümlerin kavranması ve anlatılması daha kolaydır. XP programcılardan o anki gereksinimi tatmin etmek için basit çözümü bekler. Programcı gelecekte oluşabilecek eklemeleri ve değişiklikleri düşünmemeli, sadece ve sadece kendisinden o an için bekleneni en basit haliyle implemente etmelidir.

3. Incremental Change

Inkrementel değişiklik

Basit çözümler uygulasak bile, yazılım sistemleri zaman içinde karmaşık bir yapıya dönüşebilir. Yapılan en ufak bir değişiklik bile sistemin düşünmediğimiz bir bölümü üzerinde hata oluşmasına sebep verebilir. Oluşabilecek bu hataları kontrol altında tutabilmek için değişikliklerin ufak çapta olması gerekmektedir. Büyük değişiklikler beraberinde büyük sorunları getirebilir. Bu sebepten dolayı değişikliklerin ufak çapta ve sıklıkla yapılması gerekmektedir.

4. Embracing Change

Değişimi istemek

İlerleyebilmek için kendimize bir yön tayin etmemiz ve yeniliklere açık olmamız gerekiyor. Yeniliklere açık olmak cesaret gerektirir. Bilinmeyenle uğraşmak, rahatsız edici olabilir, ama başarıyı elde edebilmek için değişimi istemek gerekir.

5. Quality Work

Kaliteli iş

XP projelerinde kaliteli işin ortaya konabileceği bir ortamın oluşturulması gerekmektedir. Hiçbir programcı hatalı program yazmak istemez. Çalışma ortamınında etkisiyle yüksek kalitede yazılım yapmak hem programcının öz güvenini artırır hem de müşteriyi tatmin edici ürünlerin ortaya konmasını sağlar.

6. Teach Learning

Öğrenmeyi öğret

XP programcı takımlarında tertipçilik ve kıdem farkı yoktur. Tecrübeli programcılar bilgilerini daha az tecrübeli programcılarla paylaşarak, hem bilginin çoğalmasını sağlarlar hem de takım arkadaşları ile teknik olarak aynı seviyeye gelirler. Programcılara komutlar vererek iş yaptırmak yerine, kendiliğinden bazı şeyleri öğrenerek, görevlerini yerine getirmeleri sağlanmalıdır.

7. Small Initial Investment

Az başlangıç yatırımı

XP en modern ve pahalı araç gereçlerle projeye başlanmasını beklemez. Başlangıç giderleri ne kadar düşük tutulabilirse, projenin iptali durumunda kayıplar o oranda az olacaktır. Başlangıçta tüm takımın dar bir finansman korsası giymesi sağlanarak, proje için daha önemli görevlere odaklanmaları sağlanır. Bunu bir örnekle açıklayabiliriz. Proje başlangıcında en son Oracle veri tabanı ve kullanıcı araçları (Toad gibi) satın alınarak, tüm ekibe bu veri tabanı ve araçları nasıl kullanacaklarına dair eğitim verilebilir. Bu çok masraflı ve bir o kadar da gereksiz bir şeydir. Projeye açık kaynak ve ücretsiz olan HSQL ya da

PostgreSQL veri tabanı ile başlanabilir. Programcı ekip böylece ne bir hafta tanımadıkları bir ürün için harcamış, ne de büyük masraflar yapılarak şu an için gerek olmayan bir altyapı komponenti satın alınmış olur. Gerekli araçlar zamanı geldiğinde edinilmelidir.

8. Play to win

Kazanmak için oyna

XP takımları kazanmak için oynar. Her zaman gözlerinin önünde nihai sonuç vardır: programı tamamlamak ve müşteriye teslim etmek. XP programcı takıma tünelin sonundaki ışığı görmek için gerekli tüm imkanları sunar.

9. Concrete Experiments

Somut denemeler

Verdiğimiz kararların sonuçlarını kontrol edebilmek için denemeler yaparız, çünkü alınan kararlar her zaman doğru olmayabilir. Bir kontrol mekanizmasına ihtiyacımız olduğu belli. Bu da somut denemeler aracılığıyla nerede olduğumuzu tespit etmekten geçer. Bu somut denemeler yazılım sistemleri içinde geçerlidir. Örneğin testler hazırlayarak, oluşturduğumuz mimari ve tasarımı kontrol ederiz.

10. Open, honest Communication

Açık ve samimi iletişim

Projenin başarılı olabilmesi için bireyler arasında açık ve samimi türde bir iletişim olması gerekmektedir. Birçok projede bu böyle değildir. Çoğu zaman bireylerin korkuları, deneyimsiz olmaları ya da kendilerini çok beğenmeleri ve diğerlerini kendilerinden alt safhada görmeleri, açık ve samimi bir iletişim ortamının oluşmasını engeller.

11. Work with people's instincs, not against them

Takımın içgüdülerini kullan, onlara karşı koyma

Bireysel içgüdü yanı sıra, bireylerin oluşturduğu takımların da içgüdüsü vardır. Eğer takım bir şeylerin doğru gitmediği hissine sahipse ve bunu dile getiriyorsa, o zaman bir şeyler yolunda gitmiyor demektir. Takımın içgüdüsüne kulak verilmelidir. Bunun göz ardı edilmesi, proje için olumsuz sonuçlar doğurabilir.

12. Accepted Responsibility

Sorumluluk üstlenmek

Sorumluluk birilerine verilmemeli, bireyler kendileri sorumluluk üstlenmeliler. Eğer bir bireye ya da bir takıma yapılması zor bir projenin sorumluluğu yüklenirse, bu birey ya da takım için motivasyonun düşmesini ve kaybetme korkusunun pekişmesini hızlandırır. Eğer bireyler ya da takımlar kendi sorumluluklarını kendileri seçerlerse, hem yaptıkları işte kendilerini iyi hissederler, hem de yüksek motivasyon ile üstlendikleri işi başarıyla tamamlarlar.

13. Local Adaptations

Sürecin ortam şartlarına adapte edilmesi

Büyük bir ihtimalle her takımın XP yi Kent Beck'in anlattığı tarzda harfiyen uygulaması mümkün değildir. Atalarımızın da dediği gibi her yiğidin yoğurt yiyiş tarzı başkadır. Amaç XP yi harfiyen uygulamak değildir, amaç kısa bir zamanda projeyi başarılı bir sonuca ulaştırmaktır. Eğer proje XP de yapılacak değişiklerle başarıya ulaşacaksa, o zaman süreç üzerinde bu değişiklikler yapılmalı ve uygulanmalıdır. Bunda bir sakınca yoktur.

14. Travel light

Az yükle yolculuk yapmak

Projede hızlı ilerleyebilmek için fazla bir yükle yola çıkılmaması gerekmektedir. Beraber çalışmayı kolaylaştırmak için kullanımı kolay araç ve gereçler seçilmelidir. Formalitelerden uzak durulmalıdır.

15. Honest Measurement

Doğru ölçüm

Proje gidişatını kontrol edebilmek için değişik türde ölçümlerin yapılması gerekmektedir. Örneğin hazırlanan birim testleri ile sınıfların işlevleri kontrol edilir. Yapılan ölçümler doğru ve samimi yapıldığı taktirde kontrol mekanizması olarak kullanılan ölçümlerin bir anlamı vardır. Programcılar tarafından samimi ve doğru yapılmayan ölçümler projenin gidişatını olumsuz yönde etkiler.

XP Teknikleri (XP Practices)

Dört XP değer ve on beş XP prensibi on dört XP tekniği ile desteklenmektedir. XP teknikleri programcıların XP değer ve prensiplerini uygulamada yardımcı olur. Kent Beck tarafından hazırlanan ilk XP versiyonunda on iki teknik yer almaktaydı. Diğer çevik süreçlerin de etkisiyle Standup-Meeting ler ve retrospektif toplantılar XP teknikleri arasına katıldı.

Bunlar:

1. On-site Customer

Programcıya yakın müşteri olarak tercüme edilebilir.

XP projeleri müşteri gereksinimlerine odaklı ilerler. Bu yüzden müşteri ve sistem kullanıcılarının projeye dahil edilmeleri gerekmektedir. Müşteri gereksinimlerini ekibe bildirir. Programcıların implementasyonu gerçekleştirebilmesi için müşteri tarafından dile getirilen gereksinimleri anlamaları gerekmektedir. Yanlış anlaşılmaları ve hataları gidermek için programcıların müşteri ve sistem kullanıcıları ile diyalog halinde olabilmesi gerekmektedir. Bu sebepten dolayı müşteri veya sistem kullanıcılarının programcıların erişebileceği bir uzaklıkta olmaları gerekir. Tipik XP projelerinde müşteri ve programcılar aynı odada beraber çalışırlar. Müşteri ekibin sorularını zaman kaybı olmadan cevaplar ve projenin ilerlemesine katkıda bulunur.

2. Standup-Meeting

Ayakta toplantı

Proje çalışanları her gün 15 dakikayı aşmayan ve ayakta yapılan toplantılarda bir araya gelirler. Bu toplantının amacı, projenin gidişatı hakkında bilgi alışverişinde bulunmaktır.

3. Planning Game

Planlama oyunu

XP projeleri iteratif ve inkrementel yol alır. Bir sonraki iterasyonda yapılması gereken işleri planlama oyununda görüşülür ve sürüm ve iterasyonun içeriği tespit edilir. Planlama oyununa müşteri, kullanıcılar ve programcılar katılır. Müşteri ve kullanıcılar daha önce kullanıcı hikayesine (user story) dönüştürdükleri isteklerine öncelik sırası verirler. Programcılar her kullanıcı hikayesi için gerekli zamanı tahmin ederler. Kullanıcı hikayelerinin öncelik sırası bu tahmine bağımlı olarak değişebilir. Planlama oyunlarında sürüm ve iterasyon planları oluşur.

4. Short Releases

Kısa aralıklarla yeni sürüm

XP projelerinde yeni implemente edilen ve değişikliğe uğrayan komponentler yeni sürümler oluşturularak müşteri ve kullanıcının beğenisine sunulur. Bu sayede hem müşteriler çalışır durumda olan programdan faydalanabilir hem de yeni sürümü inceleyerek, gereksinimleri ile örtüşüp, örtüşmediğini kontrol edebilirler. Eğer yeni sürüm müşteriyi tatmin edecek durumda değilse, gereksinimler değişikliğe uğrayabilir. Bu değişiklikler bir sonraki iterasyonda göz önünde bulundurularak, müşteri istekleri ile yüksek derecede örtüşen bir programın oluşturulması sağlanır.

5. Retrospective

Geriye bakış

Proje çalışanları düzenli aralıklarla geriye bakarak, meydana gelen sorunları gözden geçirirler. Buradaki amaç gelecekte bu sorunların tekrarını önlemektir. Geriye bakış bir ile altı aylık zaman birimleri için tüm proje çalışanları ya da seçilen bireyler tarafından yapılır. Geriye bakış toplantıları yarım gün ile üç gün arasında sürebilir.

6. Metaphor

Mecaz

XP projelerinde hazırlanan program için bir veya birden fazla, programın nasıl bir işlevi olacağını ekibin gözünde canlandırmalarını sağlayacak mecazi isim, öğe ya da resimler kullanılır. Bunlar proje çalışanlarının ortak bir payda da buluşarak, ne yapılması gerektiği hakkında bir fikir sahibi olmalarını kolaylaştırır. Örneğin bir shop sistemi yazılımı yapılacak. Burada metaphor olarak alışveriş sepeti kullanılabilir. Alışveriş sepetini duyan her programcının aklında bir shop sisteminin programlanması gerektiği fikri doğar.

7. Collective Ownership

Ortak sorumluluk

XP projelerinde programcılar ortak sorumluluk taşırlar. Bu her kod parçasının herhangi bir programcı tarafından gerekli durumlarda değiştirilebileceği anlamına gelir. Böylece yapılması gereken işler aksamaz, çünkü belli kod bölümlerinden belli programcılar sorumlu değildir. Aksine her programcı programın her bölümü üzerinde çalışma hakkına sahiptir. Bir programcının işe gelmemesi durumunda, başka bir programcı kolaylıkla onun görevlerini üstlenebilir.

8. Continuous Integration

Sürekli entegrasyon

Sistem değişiklikleri ve yeni komponentler hemen sisteme entegre edilerek test edilir. Sürekli entegrasyon sayesinde yapılan tüm değişiklikler her programcının sistem üzerinde yapılan değişiklikleri görmesini sağlar. Ayrıca sistem entegrasyonu için gerekli zaman azaltılır, çünkü oluşabilecek hatalar erken teşhis edilerek, ortadan kaldırılır.

9. Coding Standards

Kod standartları

Programcılar tarafından aynı kalitede kod yazılımı yapılabilmesi için kod yazarken kullanılacak kuralların oluşturulması gerekmektedir. Kodun nasıl formatlanacağı, sınıfların, metot isimlerinin ve değişkenlerin nasıl isimlendirileceği kod standartlarında yer alır.

10. Sustainable Pace

Kalıcı tempo

XP projelerinde programcılar haftalık belirli mesai saatlerini aşmazlar. Gereğinden fazla çalıştırılan ve yorulan bir programcıdan verimli iş yapması beklenemez. Programcıların motivasyonun ve çalışma enerjilerinin yüksek olması için günde sekiz saatten fazla çalışmalarına izin verilmemelidir. Bazen fazla mesai saatlerine ihtiyaç olabilir. Eğer durum devamlı böyle ise, bu proje gidişatında bazı olumsuzlukların göstergesi olabilir.

11. Testing

Test etmek

Oluşturulan programların kalite kontrolünden geçmesi gerekmektedir. Bu yazılım esnasında oluşturulan testlerle yapılır. Programcılar komponentler için birim testleri hazırlar. Sınıf bazında yapılan bu testlerle komponentlerin işlevleri kontrol edilir. Müşteri gereksinimlerini test etmek için onay/kabul testleri hazırlanır. Komponentlerin entegrasyonunu test etmek için entegrasyon testleri hazırlanır.

12. Simple Design

Sade tasarım

Programcılar üstlendikleri görevleri (task) en basit haliyle implemente ederler. Bu programın basit bir yapıda kalmasını ve ilerde değiştirilebilir ve genişletilebilir olmasını sağlar. Sade bir tasarım yazılım sisteminin karmaşık bir yapıda olmasını önler. Bunun yanı sıra basit tasarımlar daha kolay ve daha hızlı implemente edilebilir. Basit bir implementasyonu anlamak ve anlatmak daha kolaydır.

13. Refactoring

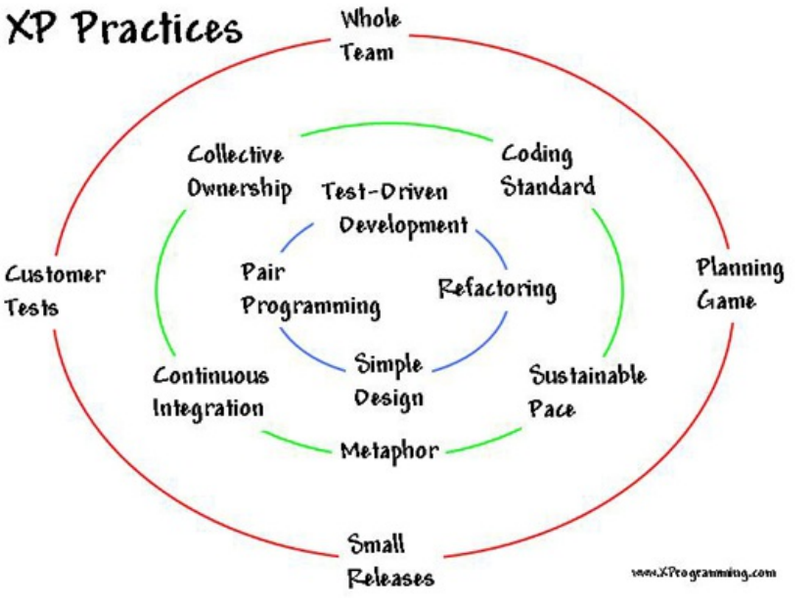
Yeniden yapılandırma

Tasarım hataları yazılım sisteminin daha ilerde tamir edilemeyecek bir hale dönüşmesine sebep verebilir. Bu yüzden bu hatalar hemen giderilir. Bu yeniden yapılandırma işlemine refactoring ismi verilir. Hazırlanan birim testleri ile yapılan değişikliklerin yan etkileri kontrol edilir. Bu açıdan bakıldığında birim testi olmayan bir sistem üzerinde yeniden yapılandırma işlemi hemen hemen mümkün değildir, çünkü değişikliklerin doğurduğu yan etkileri tespit etme mekanizması bulunmamaktadır.

14. Pair Programming

Eşli programlama

XP projelerinde iki programcı aynı bilgisayarda çalışır. Bu sayede programcıların kısa bir zaman içinde aynı seviyeye gelmesi sağlanır. Ayrıca bu kalitenin yükselmesini sağlar.



Resim 1.6 XP teknikleri

XP Rolleri

Bir çevik projede ekip çalışanlarının sorumluluk alanlarını tanımlamak için roller tayin edilir. Her rol beraberinde bazı sorumluluklar ve tanımlanmış haklar getirir. Bu roller sabit değildir. Ekip içinde değişik kişilere değişik roller verilebilir ve daha sonra rol değişikliği yapılabilir. Proje gereksinimleri doğrultusunda yeni rollerin oluşturulması mümkündür.

Müşteri (Customer)

Projenin var olma sebebi müşteridir. Müşteri ihtiyaç duyduğu ve gereksinimlerine cevap verebilecek bir yazılım sistemi için yatırım yapan kişidir. XP projelerinde şu atasözümüz geçerlidir: "parayı veren düdüğü çalar!" ;-).

Proje bünyesinde ne programlanması gerektiğini müşteri tayin eder. Müşteri yapılması gerekenleri kullanıcı hikayeleri (user story) oluşturarak ifade eder. Programcılar müşteriye bu süreçte yardımcı olurlar. Ama kullanıcı hikayelerinin oluşturulma sorumluluğu büyük ölçüde müşteriye aittir. Her kullanıcı hikayesi yazılım sisteminin bir özelliğini tanımlar. Programcıların implementasyonu gerçekleştirebilmeleri için kullanıcı hikayesini anlayabilmeleri gerekmektedir. Sadece bu durumda implementasyon süreci için bir tahminde bulunabilirler. Ayrıca oluşturulan kullanıcı hikayelerinin test edilebilir yapıda olması gerekmektedir.

Müşteri çalışma alanı (domain knowledge) hakkında bilgiye sahip olan kişidir. Programcılar müşteriye karşılaştıkları sorunları çözmek için sorular sorabilirler. Bu soruların cevabını en iyi verebilecek şahıs müşteridir.

Hangi kullanıcı hikayelerinin implemente edileceğine müşteri karar verir. Bu konuda müşteriye herhangi bir sınırlama getirilmez. Müşteri seçer ve programcılar implemente eder.

İmplemente edilen kullanıcı hikayelerini kontrol etmek amacıyla müşteri onay/kabul testleri tanımlar. Bu testler programcı ya da testçi tarafından implemente edilir. Onay/kabul testleri kullanıcı hikayesini doğru implemente edilip, edilmediğini kontrol edici bir mekanizmadır.

Programcı

Sistem analizi, tasarım, test ve implementasyon programcılar tarafından yapılır.

Müşteri tarafından hazırlanan kullanıcı hikayelerinin implementasyon süresi programcılar tarafından tahmin edilir. Bu programcıların XP projelerinde proje planlama sürecine dahil edildikleri anlamına gelmektedir. Geleneksel projelerde bu tahminleri teknik bilgiye sahip olmayan yöneticiler yapmak zorundadır. Bu yüzden bu tahminler genelde gerçekleri yansıtmaz.

Her programcı test güdümlü ve bir takım arkadaşıyla beraber çalışır. Pair programming olarak bilinen, iki programcının birlikte yazılım yapması, kısa zamanda kod hakkındaki bilginin tüm programcılar tarafından paylaşılmasını kolaylaştırır. Ayrıca pair programming programcılar arasında iletişimi ve takım içinde çalışabilme özelliğini artırır. Test güdümlü çalışmak bakımı ve geliştirilmesi kolay kodun oluşmasını sağlar. XP projelerinde programcılar herhangi bir satır kod yazmadan önce gerekli test sınıflarını oluşturarak implementasyona başlarlar.

Programcılar oluşturdukları testler yardımıyla her gün bir veya birden fazla şekilde modül entegrasyonu gerçekleştirirler. Sürekli entegrasyon programcılar tarafından oluşturulan modülleri entegre eden bir süreçtir. Her programcı bu süreçten geri bildirim sağlayarak, kendi yaptıklarının ne derecede sisteme entegre olduğunu ölçebilir.

Proje Menajeri

Proje menajeri müşteri ve programcıları bir araya getirir. Onların beraber çalışabilecekleri ortamların oluşmasını sağlar. XP proje menajeri tek başına proje planlamasından sorumlu değildir. Programcılara görev atamaz, onların kendi başlarına seçim yaparak, sorumluluk almalarını kolaylaştırır. Toplantı ve diğer buluşmaları koordine eder, takımın karşılaştığı sorunları ortadan kaldırmak için gerekli olanları yapar.

Koç

Çevik süreci tanıyan ve nasıl uygulanması gerektiğini bilen experdir. Koçun görevi proje başlangıcında çevik takımı oluşturmak ya da bir araya getirmek ve onlara belirli bir süre rehberlik yapmaktır. Koç projede sorun çıktığı zaman ya da takım XP yöntemlerinin dışına çıktığında müdahale eder. Zaman zaman koç implementasyonda aktif olarak rol alır. Örneğin birim testlerin nasıl doğru bir yapıda oluşturulabileceğini diğer programcılara gösterebilir.

Testçi

Müşteri tarafından oluşturulan onay/kabul testlerini implemente eden programcıdır. Aynı zamanda birim ve entegrasyon testlerinin implementasyonunda takım arkadaşlarına yardımcı olur.

Haklar ve Sorumluluklar

Proje çalışanları çoğu zaman içgüdüsel korku hissedebilirler. Bunun en büyük sebebi belirsizliktir. Ne ve nasıl yapılması gerektiği bilinmediği taktirde ekip içinde huzursuzluk doğar. Bu projenin başarısını negatif etkiler.

Projenin başarılı olabilmesi için çalışanların hissettikleri korkunun aza indirilmesi ya da yok edilmesi gerekmektedir. XP bu konuda proje çalışanlarına bir takım hakların tanınması gerektiğini belirtir. Hangi rollerin hangi haklara sahip olduğunu yakından inceleyelim.

Müşteri Hakları

Müşterinin sahip olduğu haklar şu şekilde özetlenebilir:

- Müşteri bütçe ve zaman planlaması yapabilmek için neyin yapılabilir olduğunu ve hangi zaman biriminde yapılabileceğini bilme hakkına sahiptir.
- Müşteri fikir değiştirerek, gereksinimler üzerinde değişiklik yapma ve yeni gereksinimlerin

implementasyonunu talep etme hakkına sahiptir.
- Müşteri programcılar tarafından sağlanabilecek en yüksek verimi ve değeri elde etme hakkına sahiptir.
- Müşteri projede somut ilerlemeyi görme hakkına sahiptir. Bu kısa aralıklarla oluşturulan yeni sürüm ve onay/kabul testlerine olumlu cevap veren yeni implementasyonlarla sağlanır.
- Müşteri bir sonraki sürümde implemente edilecek kullanıcı hikayelerini seçme hakkına sahiptir.

Programcı Hakları

Programcının sahip olduğu haklar şu şekilde özetlenebilir:

- Programcı takım arkadaşlarına soru sorma ve cevap alma hakkına sahiptir.
- Programcı kullanıcı hikayeleri için implementasyon zaman tahmini yapma hakkına sahiptir. Programcı tahminler üzerinde değişiklik yapma hakkında sahiptir.
- Programcı, kendisine görev verilmesi yerine sorumluluk alma hakkında sahiptir.
- Programcının her zaman yüksek kalitede iş çıkarma hakkı vardır.
- Programcının hangi öncelik sırasına göre neyi yapması gerektiğini bilme hakkı vardır.

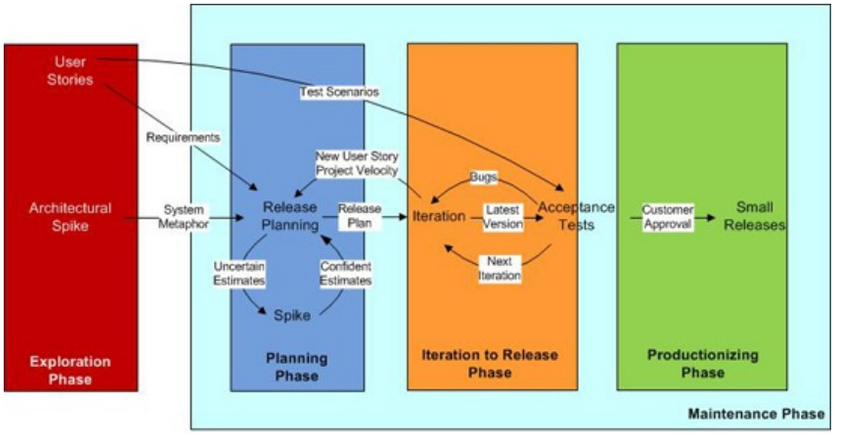
Süreç İşleyişi

XP projelerinde projenin gidişatını genel olarak şu şekilde özetleyebiliriz:

- Müşteri gereksinimleri ihtiva eden kullanıcı hikayelerini oluşturur. Bunlara öncelik sırası atar. Programcılar her kullanıcı hikayesi için implementasyon zamanını tahmin ederler
- Müşteri programcılarla beraber iterasyon(1-2 hafta) ve sürüm (1-2 ay) planını hazırlar. Her sürüm birden fazla iterasyon ihtiva eder ve müşteri tarafından kullanılacak özellikte çalışır bir sistemdir.
- Müşteri ilk iterasyon (1 hafta) için gerekli kullanıcı hikayelerini programcı tahminleri de dikkate alarak seçer.
- Programcılar iterasyon için seçilen kullanıcı hikayelerini implemente ederler. Kafalarda oluşan sorular müşteriye danışılarak cevaplandırılır.
- İterasyon sonunda programcılar müşteriye çalışır bir sistem sunarlar. Müşteri sistemi değerlendirerek programcılara geri bildirim sağlar.
- Edinilen tecrübeler ışığında bir sonraki iterasyon planlanır. Eğer müşteri yeni gereksinimlerin implemente edilmesini isterse, bunlar için tekrar kullanıcı hikayeleri oluşturulur ve tahminler yapılır. Eğer yeni kullanıcı hikayeleri yoksa, mevcut kullanıcı hikaye listesinden en yüksek öncelik sırasına sahip olanlar seçilir ve bir sonraki iterasyondan implemente edilir.
- İlk sürüm sonunda oluşan yazılım sistemi müşteri tarafından kullanıma alınır. Başka sürümler planlandıysa bir sonraki iterasyonla devam edilir.

XP Proje Safhaları

Bir XP projesi değişik safhalardan oluşur. Her safha, bünyesinde kendine has aktiviteler ihtiva eder. Resim 1.7 de bir XP projesinde olması gereken safhalar yer almaktadır.



Resim 1.7 XP proje sahfaları ve aktiviteler
(http://www.agilemodeling.com/essays/agileModelingXPLifecycle.htm)

XP projesi şu safhalardan oluşur:

- ***Keşif safhası (Exploration Phase)*** - Projenin başlangıcında keşif safhasını oluşturan aktiviteler yer alır. Bu safhada müşteri kullanıcı hikayelerini (user story) oluşturur. Programcılar teknik altyapı için gerekli deney (spike) ve araştırmayı yaparlar.
- ***Planlama Safhası (Planning Phase)*** - Keşif safhasını planlama safhası takip eder. Bu safhada müşteri programcılar yardımıyla iterasyon ve sürüm planlarını oluşturur. İterasyon planlaması için oluşturulan kullanıcı hikayelerinin implementasyon süresi programcılar tarafından tahmin edilir. Müşteri kullanıcı hikayelerine öncelik sırası vererek, iterasyonlarda hangi kullanıcı hikayelerinin öncelikli olarak implemente edilmeleri gerektiğini tespit eder. Programcılar tarafından herhangi bir kullanıcı hikayesinin implementasyon süresi tahmin edilemezse, programcılar spike solution olarak bilinen basit bir çözüm implemente ederek, kullanıcı hikayesinin gerçek implementasyonu için gerekli zamanı tahmin etmeye çalışırlar.
- ***İterasyon ve Sürüm Safhası (Iterations to Release Phase)*** - Kullanıcı hikayelerinin implementasyonu iterasyon ve sürüm safhasında gerçekleşir. Bir iterasyon bünyesinde implemente edilmesi gereken kullanıcı hikayeleri müşteri tarafından belirlenir. İmplementasyonunun işlevini kontrol etmek için müşteri tarafından onay/kabul testleri belirlenir. Bu testler programcılar ya da testçiler tarafından implemente edilir. Her iterasyon sonunda müşteriye çalışır bir yazılım sistemi sunulur. Bu şekilde müşterinin sistem hakkındaki görüşleri alınır (geri bildirim). İterasyon son bulduktan sonra çalışma hızını tahmin etmek için bir önceki iterasyonda elde edilen tecrübeler kullanılır ve iterasyon planı bu değerler doğrultusunda gözden geçirilir. Bir önceki iterasyonda oluşan hatalar bir sonraki iterasyonda gözden geçirilmek ve giderilmek üzere planlanır.
- ***Bakım Safhası (Maintenance Phase)*** - Bu programın bakımının ve geliştirilmesinin yapıldığı safhadır. Bu safhada kullanıcılar için eğitim seminerleri hazırlanır ve küçük çapta eklemeler ve sistem hatalarının giderilmesi için işlemler yapılır. Müşterinin istekleri doğrultusunda bir sonraki büyük sürüm için çalışmalara başlanır. Bu durumda tekrar kesif safhasına geri dönülmesi ve oradan işe başlanması gerekmektedir.