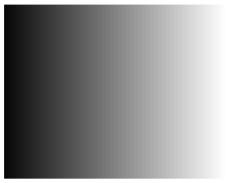


Çevik Süreç
Agile Process

KurumsalJava.com

Özcan Acar
Bilgisayar Mühendisi
http://www.ozcanacar.com

Bu makale Özcan Acar tarafından yazılmış olan Extreme Programming isimli kitaptan alıntıdır. Extreme Programming ve Çevik Süreçler hakkında genel bilgiyi Özcan Acar tarafından KurumsalJava.com'da yazılmış makalelerden edinebilirsiniz.

Okunması Tavsiye Edilen Diğer Makaleler

- Etreme Programming Nedir?
  http://www.kurumsaljava.com/2008/11/21/extreme-programming-nedir/
- Test Güdümlü Yazılım – Test Driven Development (TDD)
  http://www.kurumsaljava.com/2008/11/26/test-gudumlu-yazilim-test-driven-development-tdd/
- Sürekli Entegrasyon – Continuous Integration
  http://www.kurumsaljava.com/2008/11/26/surekli-entegrasyon-continuous-integration

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

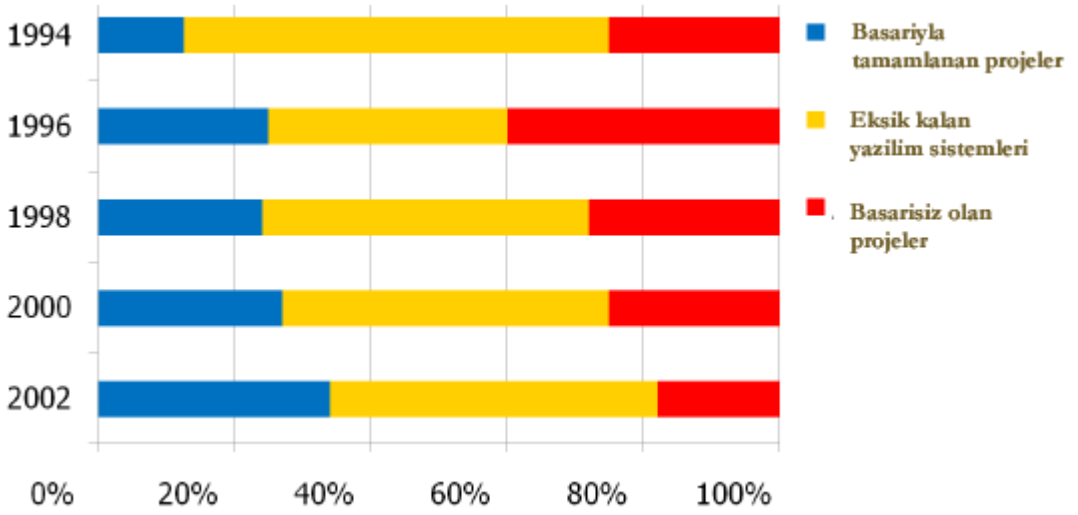
Ana hedef, müşteri gereksinimlerini tatmin eden, sabit bir bütçe ve belirli bir zaman diliminde, hatalardan arındırılmış ve müşterinin piyasadaki rekabet etme yeteneğini kuvvetlendirecek bir yazılım sistemi geliştirmektir. Yazılım ve proje geliştirme süreçleri bu hedefe ulaşmak için kullanılan metotları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

1. 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.
2. Müşteri istekleri doğrultusunda yapılmak zorunda kalınan değişiklikler proje maliyetlerini yükseltir, çünkü bu beraberinde basit olmayan yapısal değişiklikler getirebilir.
3. 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.

4. 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.
5. 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 sebep verir. Ayrıca bir zaman sonra tasarımın, implemente edilen gereksinimlere cevap vermez hale geldiği ve yapılan her değişiklik sonucunda tasarım çıkmaza girdiği görülür.
6. 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 istatistikleri

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 Kurumunun web sözlüğünde[1] çevik kelimesi şu şekilde tanımlanmaktadır:

Kolaylık ve çabuklukla davranan, tetik, atik

[1] Bakınız: http://www.tdk.gov.tr/TR/SozBul.aspx?F6E10F8892433CFFAAF6AA849816B2EF05A79F75456518CA

Çevik süreçler yazılım sektöründe kullanılan mevcut geleneksel yöntemlere (örneğin Waterfall[2] Model ya da V-Model[3]) 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 meta modeli olarak bakabiliriz. Bu yeni filozofinin somut örnekleri arasında Extreme Programming, Scrum ve Lean Development bulunmaktadır.

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 alış verişi için böyle bir toplantının faydalı olacağını düşündüler. Smaltalk gibi nesneye yönelik modern programlama dilleri yanı sıra iteratif (tekrarlamalı) 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[4] 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.

[2] Bakınız: http://en.wikipedia.org/wiki/Waterfall_model
[3] Bakınız: http://en.wikipedia.org/wiki/V-Model_%28software_development%29
[4] Bakınız: http://www.agilealliance.org

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 Cunningham	*Jon Kern*	*Dave Thomas*

Türkçesi:

1. Kişiler ve iletişim süreç ve araçlardan önce gelir
2. Çalışır durumda olan program detaylı dokümentasyondan 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:

Agile Principles

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

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

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

Business people and developers work together daily throughout the project.

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

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

Working software is the primary measure of progress.

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

Continuous attention to technical excellence and good design enhances agility.

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

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

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

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.

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.

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 (tekrarlanan) hemde inkrementel (artışlı) olarak geliştirilir. Bir iterasyon (tekrarlama) 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.

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 geri beslenme

(feedback) alınması gerekmektedir. Bunun en kolay yolu müşteri ile proje ekibinin beraber çalışmasıdır.

Özellikle programcılar, müşteri tarafından dile getirilen gereksinimlerin (requirement) fizibilizesini (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.

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.

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 alış verişi 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.

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 takdim edildiğinde, 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.

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.

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 iyileştirilir. Programcılar refactoring (yeniden düzenleme) esnasında buldukları tasarım hatalarını hemen giderirler.

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 programın 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.

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 alış verişi sonucunda mimari ve tasarım gözden geçirilir ve iyileştirilir. Ayrıca bireyler arası yapılan konuşmalar müşteri tarafından dile getirilen gereksinimlerin açıklığa kavuşturulmalarını ve daha iyi anlaşılmalarını sağlarlar.

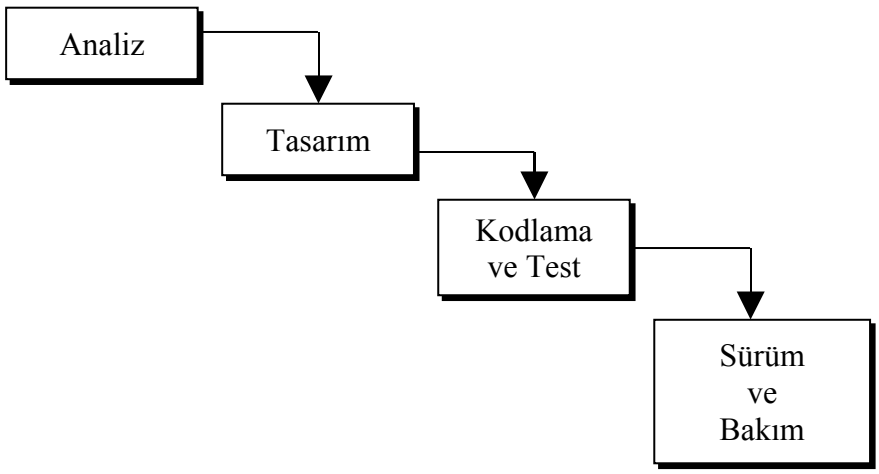
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

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

Çevik takımlarda bilgi alış verişi ve devamlı öğrenme çok önemlidir. Belirli zaman aralıklarıyla takım çalışanları bir araya gelerek, fikir alış verişinde 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 linear (doğrusal) olarak işler. Her safha başlangıç noktasında bir önceki safhanın ürettiklerini bulur. Kendi bünyesindeki değişikler doğrultusunda teslim aldıklarını bir sonraki safhanın kullanabileceği şekle dönüştürür (transformation).



Resim 1.2 Şelale modeli

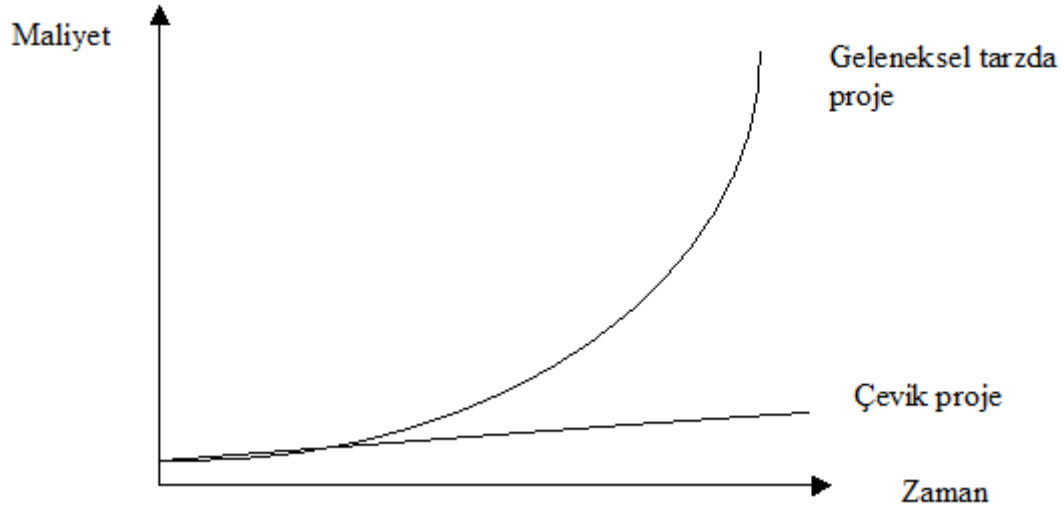
Şelale modelinin özelliklerini şu şekilde sıralayabiliriz:

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 lineardir, 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 beslenme (feedback) 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 başlangıcıyla oluşturulan dokümanlar eskimiş hale gelmiştir, 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.

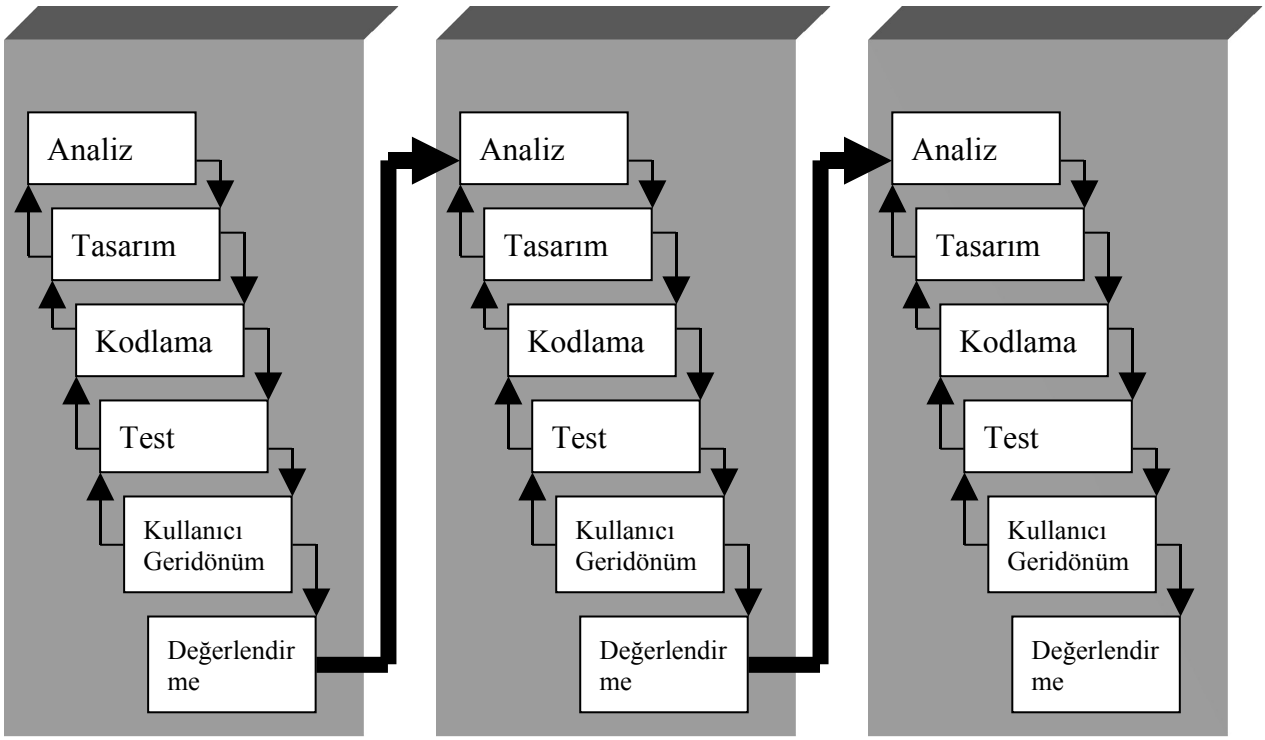


Resim 1.3 Geleneksel tarzda yapılan projelerde gerekli değişikliklerin yapılması maliyete zamanla yükseltir.

Bu çerçeveden bakıldığında proje sonunda oluşan program müşterinin aktüel 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 müşteri gereksinimlerini tatmin etmesi imkansızdır.

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 program 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 beslenme (feedback) mekanizmalarıyla müşteri gereksinimleri her zaman değişikliğe uğrayabilir.
2. Bu modelde geri beslenme merkezi bir rol oynamaktadır. Çeşitli safhalarda sağlanan geri beslenme ile projenin hangi durumda olduğu saptanır. Programcılar hazırladıkları testler aracılığıyla geri beslenme sağlayarak, implemente ettikleri komponentlerin hangi durumda olduklarını tespit ederler. Sürekli entegrasyon yapılarak programın hangi durumda olduğu geri beslenme olarak elde edilir. Müşteriye kısa aralıklarla programın yeni sürümü sunularak, geri beslenme sağlanır.
3. Proje başlangıcında detaylı dokümantasyon ve tasarım oluşturulmaz. Programcılar test güdümlü (Test Driven Developlement - TDD) 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 gelir 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 öncesi de 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 alış verişinde 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[5]) amacı XP yi geliştirmek ve XP'de yeralan 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[6]). 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.

[5] Bakınız: http://industrialxp.org
[6] Bakınız: http://www.featuredrivendevelopment.coms

Extreme Programming (XP)

Extreme Programming hakkında detaylı bilgiyi http://www.kurumsaljava.com/2008/11/21/extreme-programming-nedir/ adresinde yer alan makaleden edinebilirsiniz.