Büyük bir yazılım krizi oldu yazılıma büyük bir talep vardı ancak üretim bu talebi karşılayamıyordu aynı zamanda yazılımlar projelerinde büyük oranda başarısızlık vardı. Piyasa bizden bilgisayar bilimlerindeki o disiplinler yerine mühendislere, akademisyenlere, yetiştirdiğimiz öğrencilere yazılım yaşam döngüsü temel çekirdeklerine hâkim olsun istiyordu. Yazılım geliştirmenin, yazılımın verimsiz ve düşük kalitede olması, zamanında tamamlanmaması gibi zorluklarıyla bahşetmek için yazılım geliştirmeyi sistematik hale getirmeyi hedefleyen çeşitli süreç modelleri ortaya çıktı. Bu modellerin temel hedefi; proje başarısı için, yazılım geliştirme yaşam döngüsü (“Software Development Lif Cycle”) boyunca izlenmesi önerilen mühendislik süreçlerini tanımlamaktadır. Bir yazılım ürününün ihtiyacının ortaya çıkmasından kullanımdan kalkmasına kadar geçen döneme yazılım yaşam döngüsü modeli denir. Eğer yazılım yaşam döngü modelinde adımları takip edip yazılım projelerimizi yaparsak yazılım krizinde çıkacak hataları minimuma indirdiğimiz için mağduriyetler azalacaktır. Yazılım yaşam döngüsü modelleri sırasıyla şunlardır:

* **Gelişigüzel Model:** Adından da anlaşılacağı gibi, gelişigüzel geliştirmede belirlenmiş bir model ya da yöntem kullanılmaz. Tam anlamıyla bir model olduğunu söylemekte doğru değildir. Yalnızca geliştiren kişiye bağımlı olarak yapılır ve bu yüzden izlenebilirliği, bakım yapılabilirliği oldukça zordur, daha da ötesinde geliştiren kişinin bile aradan belirli bir zaman süreci geçtikten sonra anlamayacağı ve değiştirme güçlüğü yaşanılan ortamlardaki üretim tarzıdır.
* **Barok Modeli:** Barok modeli 70’li yılların ortalarından başlanarak kullanılmaya başlanmıştır. Yaşam döngüsü çekirdek süreçlerinin doğrusal bir biçimde gerçekleştirilmesini öngörür. Adımlar arası ilişkilerin tanımlı olmadığı bir yöntemdir. Günümüzde pek kullanılmayan bir yöntemdir. Bunun en büyük nedeni ise "Belgeleme" adımının bu modelde ayrı bir adım gibi ele alınıp yazılımın geliştirilmesi ve testinin ardından yapılmasıdır. Günümüzde kullanılan modellere aykırı bir durumdur. Günümüzde tercih edilen modellerde "Belgeleme" geliştirilen yazılımın ürünü olarak kabul edilmektedir. Ayrıca "Gerçekleştirme" adımını daha fazla ağırlık veren bir modeldir.
* **Çağlayan Modeli:** Şelale modeli, yazılım projelerinde uygulanan faaliyetlerin ardışık süreçler halinde icra edildiği, yazılım mühendisliğinin en eski ve temel modelidir. Günümüzde hükümetler ve büyük şirketler tarafından her türlü projenin yönetim standardı olarak kabul görmektedir**.** Yazılım tanımlarında belirsizlik yok ya da az belirsizlik varsa ve yazılım üretiminin çok uzun zaman alması beklenmiyor ise bir süreç modeli olarak önerilmektedir. Çağlayan yaşam döngüsü modeli aşamaları şunlardır: Gereksinimler, tasarı, uygulama, test ve bakım. Bir adımın tamamlanmasından sonra diğerine geçilir. Eksiklikler veya hatalar fark edilirse önceki adımlara geri dönülür. Adımları geride bıraktıkça, ilerleyen aşamalarda karşılaşılan hataların düzeltilmesi üstel olarak zorlaşmaktadır. Son ürünün eldesi uzun süreceğinden müşteri sabırlı olmalıdır. Birçok müşteri, gereksinimlerini eksiksiz ve kesin belirtmekte zorlanmaktadır. Rutin projelerde uygudur. Her sürecin sonunda dokümantasyondan oluşan bir çıktı elde edilir ve her çıktı kendini takip eden sürecin girdisini oluşturur. Çıktı-girdi ilişkisi nedeniyle yaşanacak olası değişimlerin projeye maliyeti yüksek olacağı için süreçler arasında atlama veya büyük çapta geri dönüşlere izin verilmez. Şelale modelinin uygulandığı projelerde süreç bitişleri net bir şekilde tanımlanmıştır ve her süreç değişiminde katı incelemeler, yoğun dokümantasyon ve yönetim onayı gibi kontrollerle proje yoğun bir denetim altında icra edilir. Bu nedenle uygulanması katı bir disiplin gerektirir. Sistem ve yazılım gereksinimleri, gerekli her şeyin baştan bilindiği ve değişiklik olmayacağı kabullenmeler kabul edilerek en başta belirlenir. Bu nedenle belirsizlik içermeyen tam anlaşılmış ve değişme ihtimali olmayan projeler için daha uygundur. Çünkü Şelale Modelinin en büyük zayıflığını bu kabullenmeler oluşturmaktadır. Müşteri, gereksinimleri tanımlanırken projenin içindedir, analiz, tasarım ve kodlama süreçlerinde görünmez, test sürecinde yeniden ortaya çıkar. Şelale modelinde en çok planlama, zaman çizelgeleri, hedef tarihler, bütçe ve bütün sistemin bir defada kullanıma alınması konuları üzerinde durulur. Şelale modelinin temel amacı proje süresince değişikliklere izin verilmeyerek, kapsam, zaman ve kaynaklar gibi faktörleri baştan sabitlemek ve büyük bir risk faktörü olan değişimi etkisiz kılmaktır. Şelale modelinin ortaya çıktığı zaman ve şartlar değerlendirildiğinde, iş süreçleri bilgi sistemlerine günümüzde olduğu kadar bağımlı değildi ve projelerin aceleyle bitirilmesi gerekmiyordu. Teknolojik gelişme ve iş yaşamındaki değişim günümüzde olduğu kadar hızlı değildi; bu yüzden projeler günümüzde olduğu kadar hızlı bir tempoyla ilerlemiyordu. Teknoloji ve kaynak kısıtlarından dolayı bilgi sistemleri donanım merkezliydi ve yazılımın ana maksadı, depolama kapasitesi ve işlemci gücü gibi kısıtlı donanımsal kaynakları en optimum şekilde kullanabilmekti.
* **V Süreç Modeli:** V-model şelale modelinin gelişmiş hali olarak düşünülebilecek bir yazılım geliştirme süreci sunar. Doğrusal bir yönde ilerlemek yerine, süreç adımları kodlama evresinden sonra yukarıya doğru eğim alır ve tipik V şeklini oluşturur. V-Model geliştirme yaşam çevriminin her bir evresi arasındaki ilişkileri gösterir. Yatay ve dikey açılar zaman veya projenin tamamlana bilirliğini ve soyut seviyeyi gösterir. V’nin sol tarafı üretim sağ tarafı ise sınama işlevleri ile ilgilidir. Sırası ile sol tarafta modül, alt sistem, sistem, sistem tanımları, gereksinimler bulunur sağ tarafta ise sırasıyla sınanmış modül, sınanmış alt sistem, sınanmış sistem, bitmiş sistem, sistem bulunur. Model aynı zamanda sınama işlemlerinde hata bulma durumunda nereye dönüleceğini de belirtmektedir. Sağ tarafta yapılan sınama işlemlerinde bulunan bir hata durumunda, yatay olarak karşısına gelen sol taraf işlevlerine dönülmektedir. V süreç modelinin temel çıktıları: Kullanıcı modeli, mimari model ve gerçekleştirim model olarak adlandırılan üç alt modeldir.
  + Kullanıcı modeli: Geliştirme sürecinin kullanıcı ile olan ilişkileri tanımlanmakta ve sistemin nasıl kabul edileceğine ilişkin sınama belirtimleri ve planları ortaya çıkarılmaktadır.
  + Mimar modeli**:** Sistem tasarımı ve oluşacak alt sistem ile tüm sistemin sınama işlemlerine ilişkin işlevler.
  + Gerçekleştirme modeli**:** Yazılım modüllerinin kodlanması ve sınanmasına ilişkin fonksiyonlar.

Belirsizlikler az iş tanımlarının belirgin olduğu BT (bilgi teknolojileri) projeleri için V süreç modeli , uygun bir model olarak belirtilmektedir. Model kullanıcının projeye katkısını artırmaktadır.BT projesinin iki aşamalı ihale edilir. İlk ihale kullanıcı modeli hedeflenerek , iş analizi ve kabul sınamalarının tanımları yapılmaktadır. İkinci ihalede ise ilkinde elde edilmiş olan kullanıcı modeli tasarlanıp gerçekleşmektedir.

* **Helezonik Model:** Helezonik modelin üzerinde durduğu en temel konular risk analizi ve prototip üretmedir. Her döngü öncesinde içerisinde bulunan bölümün risk analizi yapılır. Her döngü sonunda yeniden planlama yapılarak kısıt, ister ve hedefler yeniden hesaplanır. Bu model, önceden geliştirilmiş yazılım bileşenlerinin yeniden kullanıldığı projeler için çok uygundur. Helezonik modelin en büyük getirisi risk analizinin detaylı ele alınmasıdır. Bu nedenle zaman ve maliyet daha kolay hesaplanabilir. Özellikle güvenlik yazılımlarının oluşumunda bu model kullanılmaktadır.4 sektörden oluşur bunlar sırasıyla planlama, risk değerlendirme ve azaltma, geliştirme ve doğrulama, kullanıcı değerlendirmesidir. Açıklamak gerekirse aşamanın başarımı için somut hedefler belirlenmesine planlama denir. Riskler adresleyerek azaltıcı eylemler gerçekleştirilmesine risk değerlendirme ve azaltma denir. Genel modeller içinden geliştirme için bir model seçilmesine geliştirme ve doğrulama denir. Ara ürün ile ilgili olarak kullanıcı tarafından yapılan sınama ve değerlendirmelere kullanıcı değerlendirmesi denir. Tanımlama ve tasarım gibi sabit aşamaları yoktur; her döngü ihtiyaca göre seçilir. Her döngü bir fazı temsil eder. Yinelemeli artımsal bir artış vardır. Prototip yaklaşımı vardır. Helezonik yaşam döngüsünün avantajları vardır. Kullanıcıya katkısı üretim süreci boyunca ara ürün üretme ve üretilen ara ürünün kullanıcı tarafından sınanması temeline dayanır ve yazılımı kullanacak personelin sürece erken katılması ileride oluşabilecek istenmeyen durumları engeller. Yönetici bakışına göre gerek proje sahibi gerekse yüklenici tarafındaki yöneticiler, çalışan yazılımlarla proje boyunca karşılaştıkları için daha kolay izleme ve hak ediş planlaması yapılır. Yazılım geliştiriciye göre yazılımın kodlanması ve sınanması daha erken başlar.
* **Artımsal Geliştirme Süreç Modeli:** Artırımsal model bir takvime bağlı olarak yazılımı kesim kesip geliştirip teslim etmeye dayanır. Her bir yeni kesim öncekinin üstüne bazı ek işlevlerin eklenmesini öngörür. Artırımsal model yazılım geliştirmenin kısıtlı sayıda çalışanla işin yapılmasını sağlama gibi bir üstünlüğü vardır. Üretilen her yazılım sürümü birbirini kapsayacak ve giderek artan sayıda işlev içerecek şekilde geliştirilir. Uzun zaman alabilecek ve sistemin eksik işlevlikle çalışabileceği türdeki projeler bu modele uygun olabilir. Bir taraftan kullanım, diğer taraftan üretim yapılır. Artırımsal geliştirme süreç modeli birinci adımda genel gereksinimler belirlenir, ikinci adımda gereksinimleri artırımlara bölme, üçüncü adımda sistem mimarisini tanımlama, dördüncü adımda sistem artırılmasının yapılması, beşinci adımda artırılımın onaylanması, altıncı adımda artılımın birleştirilmesi, yedinci adımda sistemin onaylanmasıdır. Eğer sistem bitmemiş ise dördüncü adıma geri dönülür.

Her teslimle birlikte müşteriye görünen bir değer döndüğünden, sistemin işlevselliği erken aşamalarda ortaya çıkar. Erken teslimler, sonraki teslimler için gereksinimleri çıkarmada prototip vazifesi görür. Projenin tümden batması riskini azaltır. Öncelikli gereksinimleri karşılayan sistem işlevleri daha çok test edilir.

* **Kodla ve Düzelt Yaşam-Döngü Modeli:** Çok uzun programlar için kullanılabilir. İlk etapta yazılım ürününün ilk sürümü geliştirilir. Yazılım ürünü gerçekleştirilir bu siste en son istenilen şekle gelinceye kadar devam geliştirilir. Bakım safhası vardır ama çok zordur. Çünkü sisteme ait belgeleme yoktur. Ayrıca emeklilik safhası vardır. İşlem basamakları sırasıyla ilk adım ilk versiyonun gerçekleştirme, ikinci adım müşteri gereksinimleri karşılana kadar düzenle, üçüncü adımda teslim sonrası bakımdır bu adımdan gerekli görüldüğü taktirde ikinci adıma geri dönülür. Dördüncü ve son adımda ise emekliliktir. Yazılım geliştirmenin en kolay yoludur ancak en pahalısıdır. Kodlamadan sonra bir yazılım ürünündeki değişikliklerin maliyetini düşünürsek, bu maliyet çok yüksektir. Ayrıca, şartname ve tasarım dokümanı olmaksızın, ürünün bakımını yapmak fazlasıyla zordur. Bireysel geliştiriciler için uygundur, takım için değildir. Bazı avantajları vardır bunlar şöyledir: Herhangi bir planlamaya ihtiyaç duyulmaz, çok küçük projelerde ya da kısa ömürlü prototiplerde uygulanabilir, program aşamaları çabuk geçilir​​ ,uzman görüşüne ihtiyaç düşüktür herkes bu modeli kullanabilir​​.Bazı da dezavantajları vardır onlarda şöyledir: Kontrollü değildir, kaynak planlaması yoktur,​​ bitiş süresi belli değildir, hataların bulunması ve doğrulaması zordur, kodları düzeltmek maliyetli olabilir​​ ,kodlar kullanıcının ihtiyacını karşılamayabilir, kodlar sonradan değiştirmek için planlanmadığından esnek değildir, değiştirilmesi zordur​​.
* **Çevik Yazılım Geliştirme :**2001 yılında Kent Beck ve 16arkadaşı tarafından çevik manifesto oluşturuldu. Yazılım geliştirme amacıyla üretilen bu modelleme biçimi, kapsadığı değerler, prensipler ve pratikler sayesinde geleneksel modelleme metotlarına göre yazılımlara daha esnek ve kullanışlı biçimde uygulanabilir. Bu manifestoya göre:
  + Bireyler ve aralarındaki etkileşim, kullanılan araç ve süreçlerden daha önemli önceliktir.
  + Çalışan yazılım, detaylı belgelerden daha önemli önceliktir.
  + Müşteri ile iş birliği, sözleşmedeki kesin kurallardan daha önemli önceliktir.
  + Değişikliklere uyum sağlayabilmek, mevcut planı takip etmekten daha önemli önceliktir.

Çevik Yazılım Geliştirme Prensipleri

1. İlk öncelik, sürekli kaliteli yazılım teslimatıyla müşteri memnuniyetini sağlamaktır.
2. Proje ne kadar ilerlemiş olursa olsun değişiklikler kabul edilir. Çevik yazılım süreçleri değişikleri müşteri avantajına dönüştürür.
3. Mümkün olduğunca kısa zaman aralıklarıyla çalışan, kaliteli yazılım teslimatı yapılır.
4. Analistler, uzmanlar, yazılımcılar, testçiler vs. tüm ekip elemanları bire bir iletişim halinde günlük olarak birlikte çalışır.
5. İyi projeler motivasyonu yüksek bireyler etrafında kurulur. Ekip elemanlarına gerekli destek verilmeli, ihtiyaçları karşılanarak proje ile ilgili tam güvenilmelidir.
6. Ekip içerisinde kaliteli bilgi akısı için yüz yüze iletişim önemlidir.
7. Çalışan yazılım projenin ilk gelişim ölçütüdür.
8. Çevik süreçler mümkün olduğunca sabit hızlı sürdürülebilir gelişmeye önem verir.
9. Sağlam teknik alt yapı ve tasarım çevikliği arttırır.
10. Basitlik önemlidir.
11. En iyi mimariler, gereksinimler ve tasarımlar kendini organize edebilen ekipler tarafından yaratılır.
12. Düzenli aralıklarla ekip kendi yöntemlerini gözden geçirerek verimliliği artırmak için gerekli iyileştirmeleri yapar.

En yaygın kullanılan çevik metodolojiler Extreme Programming (XP), SCRUM, Agile Unified Process, Feature-Driven Development (FDD), Test-Driven Development (TDD)LEAN Development, Dynamic System Development Methodology (DSDM), Microsoft Solution Framework (MSF) olarak bilinen çevik metodolojiler vardır.

**EXTREME PROGRAMMING:** 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 rüzgarınesmesini sağlamıştır. XP kolay, grup içi iletişime önem veren, geri dönüşlerin daha fazla olmasına imkân sağlayan bir yazılım geliştirme yöntemidir. Extreme Programming' in dört temel değeri vardır: İletişim, Basitlik, Geri Bildirim, Cesaret. Projelere baktığımızda insanların birbiriyle tam olarak anlaşamaması nedeniyle önemli problemlerin olduğunu görürüz. Projenin başarıya ulaşabilmesi için ekibin iyi iletişimi olmalı. XP iletişim eksikliğini ortadan kaldırmayı amaçlar. XP’de iletişim yüz yüze olmalıdır. Yazılım ekibi ile yazılım kullanıcıları arasında iyi bir iletişim olması önemlidir. Basitlik XP’nin temel değeridir. Basitlik zorunlu işlerin yapılmasıdır. Karmaşıklık XP’nin mantığına aykırıdır. XP esnek ve basit bir sistem gerçekleştirmeye çalışır. Geri bildirim ortaya çıkabilecek hataları ortadan kaldırılır. Müşteri proje grubunun üyesidir. Müşteri yazılım ekibiyle sık sık buluşarak ilerde olabilecek büyük anlaşmazlıklar daha önceden giderilir. En zoru cesarettir. Başarısızlıktan korkmak projenin hızının düşmesine neden olur. Yaptığınız işi müşteri memnun kalmazsa yeniden yazmalısınız. Yazılım geliştirmede kolaylığı ve esnekliği sağlamak için, XP 12 farklı pratiği ön görür; planlama oyunu, ekipte müşteri, önce test, basit tasarım, çiftli programlama, sürekli entegrasyon, kısa aralıklı sürümler, yeniden yapılandırma, ortak kod sahiplenme, metafor, kodlama standardı, haftada 40 saat.

**SCRUM:**1990’ların ortalarında geliştirilen proje yönetimi ve planlama ile ilgili yöntemlere odaklı olan mühendislik detayları içermeyen bir modeldir. Yazılımı küçük birimlere bölerek, yinelemeli olarak geliştirmeyi öngörür. Bir yinelemenin tanımlanması 30 günden fazla sürmemekte ve günlük 15 dakikalık toplantılarla sürekli iş takibi yapılır. Karmaşık ortamlarda adım adım yazılım geliştiren küçük ekipler için uygundur. Gereksinimleri kolaylıkla tanımlanamadığı ve kaotik durumların beklendiği projeler için en uygun metodolojidir. Yüksek seviyede belirsizlik arz eden projelerde son yaygın biçimde uygulandığı ve başarılı sonuçlar ürettiği bildirilmiştir. Scrumda üç temel kavram vardır bunlar sırasıyla roller, toplantılar, bileşenler/araçlar. Projenin başlangıç adımı olarak yazılım gereksinimlerinin ürün sahibi tarafından ürün gereksinim dokümanına yazılmasını düşünebiliriz. Bu dokümanın sahibi ürün sahibidir ve gereksinimleri önceliklerine göre sıralar. Ürün sahibi bu dokümana yazılım geliştirme süresince eklemeler ve çıkarmalar yapıp öncelikleri değiştirme hakkına sahiptir. Böylece ürün sahibi değişen ihtiyaçlarına uygun olarak bir yazılıma sahip olma şansını yakalamış olur. Gereksinimler belirlendikten sonra yazılım geliştirme takımı Sprint planlama toplantısında bir sonraki Sprint'de geliştirilmek üzere ürün gereksinim dokümanından ürün sahibinin belirlediği yüksek öncelikli gereksinimleri seçerek Sprint dokümanına aktarırlar. Bu toplantıya Scrum yöneticisi, ürün sahibi ve takım üyeleri katılırlar ve Sprint süresi en az 2 en fazla 4 hafta olarak belirlenir. Sprint planlama toplantılarını Scrum yöneticisi yönetir. Scrum yöneticisinin asıl görevi Scrumun temel prensiplerinin projeye uygulanmasını, bu prensiplerin takım üyelerince doğru şekilde anlaşılmasını sağlamaktır. En önemli görevi ise Sprint süresince takımı dışardan gelebilecek etkilere karşı korumak ve takımın ihtiyaçlarını karşılamaktır. Scrum her Sprint'in sonunda mutlaka ürün sahibine kullanabileceği bir yazılım sağlamayı hedefler, bundan dolayı planlanan Sprint süresi (2-4 hafta) asla uzatılmaz. Fakat eğer bir gereksinim belirlenen Sprint süresi içerisinde gerçekleştirilemeyecekse bir sonraki Sprint'e aktarılabilir. Ve aynı şekilde eğer Sprint süresi bitmeden Sprint dokümanındaki gereksinimlerin hepsi tamamlanmışsa ürün gereksinim dokümanından yeni gereksinimler Sprint dokümanına aktarılabilir. Sprint planlama toplantısında belirlenen gereksinimler takım üyelerince küçük görevlere bölünerek takım üyelerine geliştirilmek üzere atanır. Scrum takımı geleneksel yazılım geliştirme süreçlerinden farklı olarak kesin rollere sahip değildir. Scrum takımındaki bütün üyeler çapraz görevlerde yer alabilirler, böylece kodun tek bir kişiye bağımlılığı riski ortadan kaldırılmış olur. Sprint dokümanının sahibi bu sefer ürün sahibi değil yazılım geliştirme takımıdır, dolayısıyla bu dokümana ürün sahibi değil takım üyeleri katkıda bulunurlar. Sprint dokümanına aktarılan gereksinimlerin tahmini geliştirme süresi saat bazında takım üyelerince belirlenir ve Sprint boyunca sürekli olarak tahmini bu zamanlar güncellenerek Sprint kalan zaman grafikleri oluşturulur. Böylece Sprint süresince ürün sahibi ve scrum yöneticisi Sprint'in genel gidişi hakkında bilgi sahibi olur, aynı zamanda takım elemanları da kalan iş sürelerini ve harcadıkları zamanı takip edebilirler. Scrum'un belki de verimliliği artıran en önemli kavramlarından biri de günlük Sprint toplantılarıdır. Bu toplantılar her gün belirli saatlerde farklı bir takım üyesinin liderliğinde ayak üstü yapılır ve en fazla 15 dakika sürer. Bu toplantılarda her takım üyesi şu 3 soruya cevap verir; Dün ne yaptım? Bugün ne yapacağım? Önümde olan engeller ve karşılaştığım sorunlar neler? bu toplantılara herkesin zamanında ve davet edilmeden katılması ve uzun sürmemesi çok önemlidir. Bu toplantılar sayesinde takım üyelerinin her biri diğer üyelerin nelerle uğraştığını öğrenme fırsatını edinirler ve çalışacakları işleri diğerleriyle paylaştıkları için işlerine daha iyi konsantre olabilirler. Her Sprint'in bitiminde ortaya konulan ürün hakkında geri besleme alabilmek için yazılımla alakalı her türlü kişiye (Ürün sahibi, pazarlama, diğer takımlar vs.) açık Sprint gözden geçirme toplantısı yapılır. Bu toplantının amacı yazılımın ürün sahibinin gereksinimlerine uygun olarak geliştirildiğinden emin olmaktır. Bu sayede müşterinin gereksinimleri bir şekilde yanlış anlaşılmış ise bu fark edilir ve bir sonraki Sprint'de bu hataların önüne geçilir. Bu adımlar ürün sahibinin ürün gereksinim dokümanına yazdığı, zaman içinde geliştirip, değiştirdiği gereksinimler bitene kadar tekrarlanır.

ERHAN KIYAK

180601027

BAKIRÇAY ÜNİ/BİL. MÜH.