Software Development Lifecycle (SDLC)

Introduction

**Software Development Lifecycle (SDLC)** is a systematic process to be followed for a software project. In other words, it is a structured way to create and develop software.

The main purpose of why corporations implement SDLC is to produce software with the highest quality and lowest cost in the shortest time.

There are various phases within SDLC, and each phase has its own different process and activity. This helps the development team to design, create and deliver a high-quality product.

Every phase in a life cycle of software development needs to be deliverable from the previous phase. Requirements are converted into design, design into development and development into testing, after testing it is given to the client.

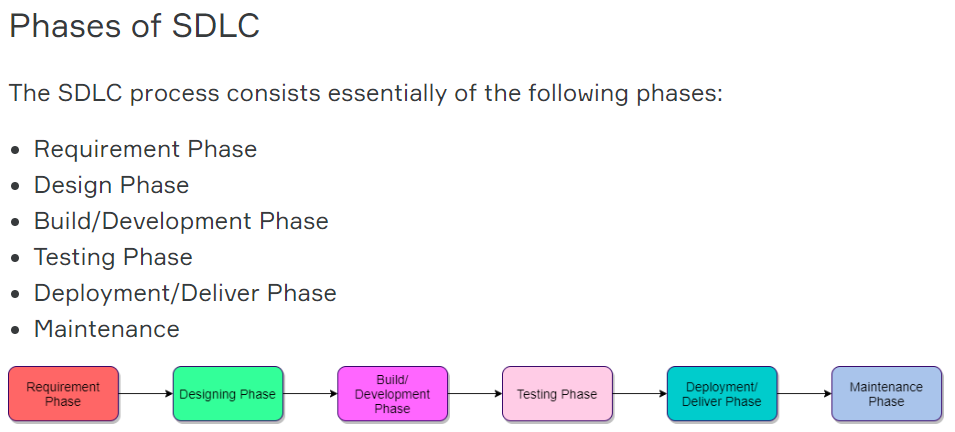
Yazılım Geliştirme Yaşam Döngüsü (SDLC), bir yazılım projesi için izlenecek sistematik bir süreçtir. Başka bir deyişle, yazılım oluşturmak ve geliştirmek için yapılandırılmış bir yoldur.

Şirketlerin neden SDLC uyguladıklarının temel amacı, en kısa sürede en yüksek kalitede ve en düşük maliyetli yazılım üretmektir.

SDLC içinde çeşitli fazlar vardır ve her fazın kendi farklı süreci ve aktivitesi vardır. Bu, geliştirme ekibinin yüksek kaliteli bir ürün tasarlamasına, yaratmasına ve teslim etmesine yardımcı olur.

Yazılım geliştirme yaşam döngüsünün her aşaması, bir önceki aşamadan teslim edilebilir olmalıdır. Gereksinimler tasarıma, tasarıma geliştirme ve geliştirmeye teste dönüştürülür, testten sonra müşteriye verilir.

SDLC Aşamaları



Phase-1: Requirement Phase

The requirement is the first and the most critical phase of SDLC for both the developing team and the project manager. During this phase, the client specifies requirements, specifications, expectations and any other special requirement related to the product or software. The business manager or project manager gathers all of this information and also prerequisites.

All the information gathered from this phase is crucial to developing the product as per the customer requirements.

To develop the software system we should have a clear understanding of the desired product/software. To achieve this we need to continuous communication with customers to gather all requirements.

Once all the information gathered, the next step is to clearly define and document the product requirements and get them approved by the customer or the market analysts.

This is done through an **SRS (Software Requirement Specification)** document. It consists of all the necessary requirements to be designed and developed during the project life cycle.

**SRS (Software Requirement Specification)** "SRS is a detailed description of a software system to be developed with requirements. The SRS is developed based on the agreement between customers and contractors. It may include the use cases of how a user is going to interact with a software system. "

Aşama-1: Gereksinim Aşama

Gereklilik, hem geliştirme ekibi hem de proje yöneticisi için SDLC'nin ilk ve en kritik aşamasıdır. Bu aşamada müşteri, ürün veya yazılımla ilgili gereksinimleri, özellikleri, beklentileri ve diğer özel gereksinimleri belirtir. İşletme yöneticisi veya proje yöneticisi tüm bu bilgileri ve önkoşulları toplar.

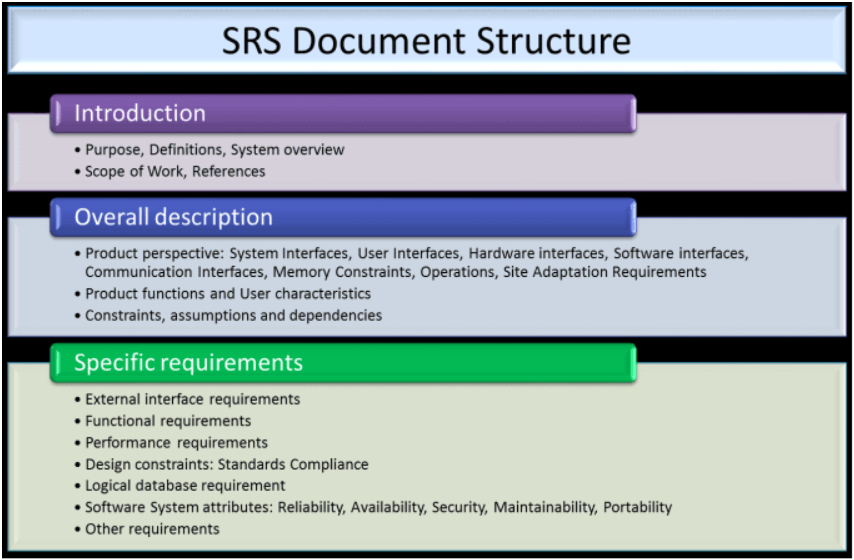
Bu aşamadan toplanan tüm bilgiler, ürünün müşteri gereksinimlerine göre geliştirilmesi için çok önemlidir.

Yazılım sistemini geliştirmek için, istenen ürün / yazılım hakkında net bir anlayışa sahip olmalıyız. Bunu başarmak için tüm gereksinimleri karşılamak üzere müşterilerle sürekli iletişim kurmamız gerekir.

Tüm bilgiler toplandıktan sonra, bir sonraki adım ürün gereksinimlerini açıkça tanımlamak ve belgelemek ve müşteri veya piyasa analistleri tarafından onaylanmasını sağlamaktır.

Bu bir **SRS (Software Requirement Specification)** belgesi ile yapılır. Proje yaşam döngüsü boyunca tasarlanması ve geliştirilmesi gereken tüm gereksinimlerden oluşur.

**SRS (Software Requirement Specification)** "SRS, gereksinimlerle birlikte geliştirilecek bir yazılım sisteminin ayrıntılı bir tanımıdır. SRS, müşteriler ve yükleniciler arasındaki anlaşmaya dayanarak geliştirilmiştir, bir yazılım sistemi. "



### Phase-2: Design Phase

In this phase, the requirement gathered in the SRS document is used as an input and software architecture that is used for implementing system development is derived.

This is the high priority phase in a system's development life cycle because the **logical designing** of the system is converted into **physical designing**. The output of the requirement phase is a list of things that are required and the design phase gives the way to accomplish these requirements. The decision of all required essential tools such as **programming language** like Java, .NET, PHP; **database** like Oracle, MySQL; **a combination of hardware and software** to provide a platform on which software can run without any problem is taken in this phase.

There are several tools and techniques used for describing system design, such as Flowchart, Data flow diagram (DFD), Data dictionary, Structured English, Decision table, and Decision tree.

Aşama-2: Tasarım Aşaması

Bu aşamada, SRS belgesinde toplanan gereksinim girdi olarak kullanılır ve sistem geliştirmeyi uygulamak için kullanılan yazılım mimarisi elde edilir.

Sistemin mantıksal tasarımı fiziksel tasarıma dönüştürüldüğünden, bu sistemin geliştirme yaşam döngüsünün yüksek öncelikli aşamasıdır. Gereksinim aşamasının çıktısı, gerekli olan şeylerin bir listesidir ve tasarım aşaması, bu gereksinimleri yerine getirmenin yolunu sunar. Java, .NET, PHP gibi programlama dili gibi gerekli tüm araçların kararı; Oracle, MySQL gibi veritabanı; yazılımın sorunsuz bir şekilde çalışabileceği bir platform sağlamak için donanım ve yazılımın bir kombinasyonu bu aşamada alınır.

Akış çizelgesi, Veri akış diyagramı (DFD), Veri sözlüğü, Yapısal İngilizce, Karar tablosu ve Karar ağacı gibi sistem tasarımını tanımlamak için kullanılan çeşitli araçlar ve teknikler vardır.

### Phase-3: Build/Development Phase

After the successful completion of the requirement and design phase, the next step is to implement the design into the development of a software system.

This phase is also known as **coding phase**.

Developers start to build the entire system by writing code using the chosen programming language.

Work/task is divided into small units or modules, and coding starts by the team of developers according to the design and the requirements of the client to produce the desired result.

Coding Phase is the longest phase of the SDLC process, and it requires a more focused approach for the developer.

Aşama-3: Yapım / Geliştirme Aşaması

Gereksinim ve tasarım aşamasının başarıyla tamamlanmasından sonra, bir sonraki adım tasarımı bir yazılım sisteminin geliştirilmesine uygulamaktır.

Bu aşama aynı zamanda kodlama aşaması olarak da bilinir.

Geliştiriciler, seçilen programlama dilini kullanarak kod yazarak tüm sistemi oluşturmaya başlar.

İş / görev küçük birimlere veya modüllere ayrılır ve kodlama, geliştiricilerin ekibi tarafından, müşterinin istenen sonucu üretmek için tasarımına ve gereksinimlerine göre başlar.

Kodlama Aşaması, SDLC sürecinin en uzun aşamasıdır ve geliştirici için daha odaklı bir yaklaşım gerektirir.

### Phase-4: Testing Phase

Once the software is complete, it is the time for the testing phase.

This phase is where you focus on investigation and discovery. The testing team starts testing the functionality of the entire system. This is done to verify that the software works and gives the result as per the requirements addressed in the requirement phase or not.

The development team makes a **test plan** to start the test. This test plan includes all types of essential testing such as integration testing, unit testing, acceptance testing, and system testing.

If there is a bug/defect detected in the software, or it is not working as expected. The testing team gives detailed information to the development team about the issue. If the defect is valid or worth fixing, it will be fixed and the development team replaces it with the new one. It also needs to be verified.

4. Aşama: Test Aşaması

Yazılım tamamlandıktan sonra test aşamasının zamanı gelmiştir.

Bu aşama, araştırmaya ve keşfetmeye odaklandığınız yerdir. Test ekibi tüm sistemin işlevselliğini test etmeye başlar. Bu, yazılımın çalıştığını ve gereksinim aşamasında ele alınan gereksinimlere göre sonuç verdiğini doğrulamak için yapılır.

Geliştirme ekibi testi başlatmak için bir test planı yapar. Bu test planı, entegrasyon testi, birim testi, kabul testi ve sistem testi gibi tüm temel testleri içerir.

Yazılımda bir hata / hata tespit edildiğinde veya beklendiği gibi çalışmıyorsa. Test ekibi, geliştirme ekibine konuyla ilgili ayrıntılı bilgi verir. Kusur geçerliyse veya düzeltilmeye değerse, düzeltilecek ve geliştirme ekibi yenisiyle değiştirilecektir. Ayrıca doğrulanması gerekir.

### Phase-5: Deployment/Deliver Phase

When software testing is completed with a satisfying result and there are no remaining issues in the working of the software, it is delivered to the customer.

As soon as customers receive the product, they are recommended first to do the beta testing. In beta testing, customers can require any changes which are not present in the software but mentioned in the requirement document to make it more user-friendly.

Besides this, if any type of defect is encountered while a customer using the software, the development team will be informed to fix this problem. If it is a critical defect, the development team solves it in a short time. Otherwise, it will wait for the next version.

After the solution of all types of bugs and changes, the software finally deployed to the end-user.

Aşama-5: Dağıtım / Dağıtım Aşaması

Yazılım testi tatmin edici bir sonuçla tamamlandığında ve yazılımın çalışmasında herhangi bir sorun kalmadığında, müşteriye teslim edilir.

Müşteriler ürünü alır almaz, öncelikle beta testini yapmaları önerilir. Beta testinde müşteriler, yazılımda bulunmayan ancak gereksinim belgesinde belirtilen değişiklikleri daha kullanıcı dostu hale getirmek için değişiklik isteyebilir.

Bunun yanı sıra, yazılımı kullanan bir müşteri ile herhangi bir kusurla karşılaşılırsa, geliştirme ekibine bu sorunu çözmesi bildirilecektir. Kritik bir kusursa, geliştirme ekibi kısa sürede çözer. Aksi takdirde, bir sonraki sürümü bekleyecektir.

Her türlü hatanın ve değişikliğin çözümünden sonra, yazılım son kullanıcıya dağıtıldı.

### Phase-6: Maintenance Phase

The **last** phase of the process SDLC is the maintenance phase where the process continues until the software's life cycle comes to an end. When a customer starts using software, actual problems start to show up. At that time, there's a need to solve these problems.

Maintenance Phase also includes making changes in hardware and software to maintain its operational effectiveness like to improve its performance, enhance security features and address customer's requirements.

Aşama-6: Bakım Aşaması

Proses SDLC'sinin son aşaması, yazılımın yaşam döngüsü sona erene kadar sürecin devam ettiği bakım aşamasıdır. Bir müşteri yazılımı kullanmaya başladığında, gerçek sorunlar ortaya çıkmaya başlar. O zaman, bu sorunları çözme ihtiyacı var.

Bakım Aşaması ayrıca, performansını artırmak, güvenlik özelliklerini geliştirmek ve müşteri gereksinimlerini karşılamak gibi operasyonel etkinliğini korumak için donanım ve yazılımda değişiklik yapılmasını da içerir.

SDLC Models

Introduction

A SDLC model describes the types of activities performed in a software development project at each stage, and how the activities relate logically and chronologically to each other.

There are many different SDLC models, each of which requires different approaches to testing.

Bir SDLC modeli, her aşamada bir yazılım geliştirme projesinde gerçekleştirilen faaliyet türlerini ve faaliyetlerin mantıksal ve kronolojik olarak birbiriyle nasıl ilişkili olduğunu açıklar.

Her biri test için farklı yaklaşımlar gerektiren birçok farklı SDLC modeli vardır.

### Software Development and Software Testing

It is an important part of a tester's role to be familiar with the common SDLC models so that appropriate test activities can take place. In any SDLC model, there are several characteristics of good testing:

* For every development activity, there is a corresponding test activity
* Each test level has test objectives specific to that level
* Test analysis and design for a given test level begin during the corresponding development activity
* Testers participate in discussions to define and refine requirements and design, and they are involved in reviewing work products as soon as drafts are available

No matter which SDLC model is chosen, test activities should start in the early stages of the life cycle, adhering to the testing principle of early testing.

Uygun test aktivitelerinin yapılabilmesi için ortak SDLC modellerine aşina olmak test uzmanının rolünün önemli bir parçasıdır. Herhangi bir SDLC modelinde, iyi testin birkaç özelliği vardır:

▪ Her geliştirme faaliyeti için karşılık gelen bir test faaliyeti vardır

▪ Her test seviyesinin o seviyeye özgü test hedefleri vardır

▪ Belirli bir test seviyesi için test analizi ve tasarımı, ilgili geliştirme faaliyeti sırasında başlar

▪ Testçiler, gereksinimleri ve tasarımı tanımlamak ve hassaslaştırmak için tartışmalara katılırlar ve taslaklar hazır olur olmaz iş ürünlerinin incelenmesine katılırlar.

Hangi SDLC modeli seçilirse seçilsin, test aktiviteleri yaşam döngüsünün erken aşamalarında başlamalı ve erken testlerin test ilkesine bağlı kalacaktır.

### Verification & Validation

In every development life cycle, a part of testing is focused on **verification** testing, and another part is focused on **validation** testing.

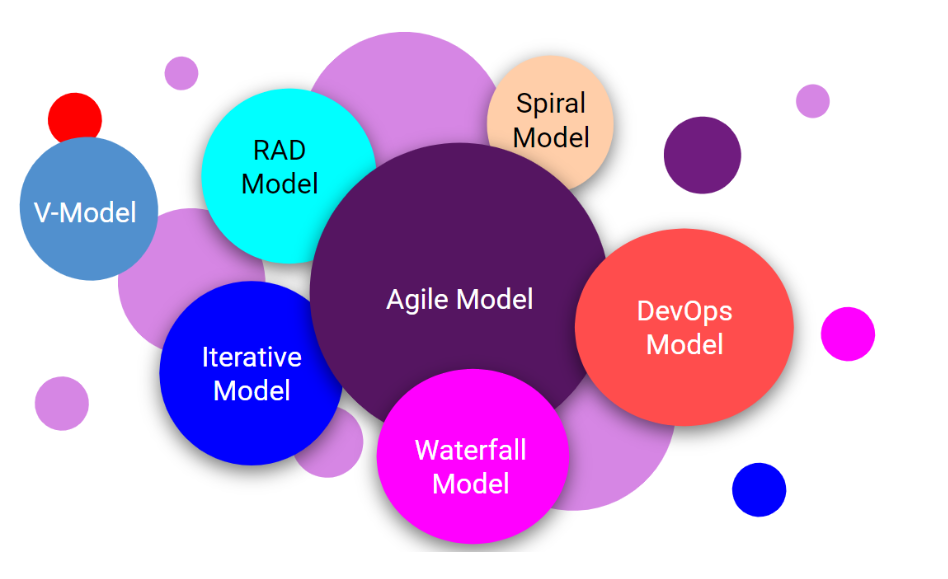
Verification is concerned with evaluating a work product, component, or system to determine whether it meets the requirements set. Verification focuses on the question **Is the deliverable built according to the specification?**

Validation is concerned with evaluating a work product, component, or system to determine whether it meets the user needs and requirements. Validation focuses on the question **Is the deliverable fit for purpose, and does it provide a solution to the problem?**

Her geliştirme yaşam döngüsünde, testin bir kısmı doğrulama testine, bir kısmı da onaylama testine odaklanır.

Doğrulama, bir iş ürününün, bileşenin veya sistemin, belirlenen gereksinimleri karşılayıp karşılamadığını belirlemek için değerlendirilmesi ile ilgilidir. Doğrulama şu soruya odaklanır: Teslim edilebilir şartnameye göre mi inşa edilmiştir ?

Onaylama, bir iş ürününün, bileşenin veya sistemin, kullanıcının ihtiyaç ve gereksinimlerini karşılayıp karşılamadığını belirlemek için değerlendirilmesi ile ilgilidir. Onaylama şu soruya odaklanır: Teslim edilebilir amaca uygun mu ve soruna bir çözüm sunuyor mu?



### Waterfall Model

**The Waterfall Model** is the earliest approach and a widely accepted SDLC model that was used for software development to ensure the success of some type of projects.

This model is very simple to understand and use. The model formed the transition between the phases of the software development process like a waterfall pouring.

It illustrates the process of software development in a **linear sequential flow**. This means that each phase must be completed before the beginning of the next phase. Each phase finds what the previous phase produces at the starting point and there is no overlapping in the phases.

Şelale Modeli, bazı tür projelerin başarısını sağlamak için yazılım geliştirme için kullanılan en eski yaklaşım ve yaygın olarak kabul edilen bir SDLC modelidir.

Bu modeli anlamak ve kullanmak çok basittir. Model, şelale dökülmesi gibi yazılım geliştirme sürecinin aşamaları arasındaki geçişi oluşturdu.

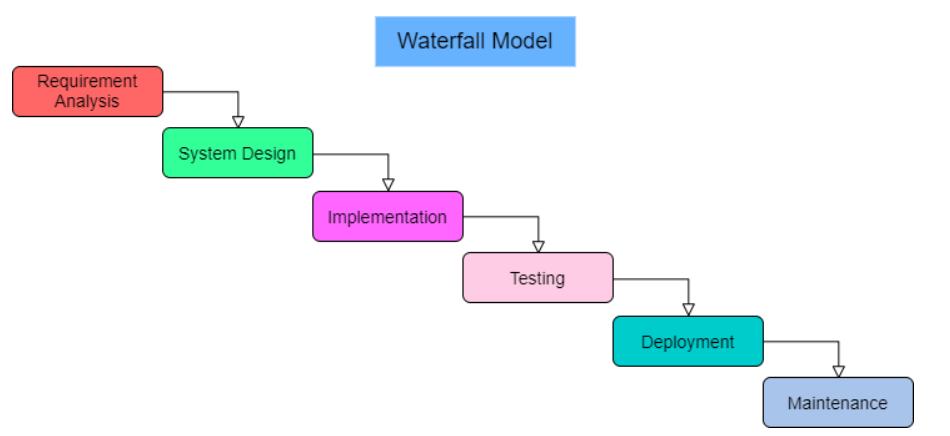
Doğrusal bir ardışık akışta yazılım geliştirme sürecini gösterir. Bu, her aşamanın bir sonraki aşamanın başlangıcından önce tamamlanması gerektiği anlamına gelir. Her faz, bir önceki fazın başlangıç noktasında ürettiklerini bulur ve fazlarda üst üste binme yoktur.

### Design

In **The Waterfall** approach, the entire software development process is divided into separate phases. Usually, In this model, the result of one phase acts sequentially as the input for the next phase.

The sequential phases in the Waterfall model are;

* **Requirement Gathering and analysis** − During this process, all possible system requirements to be created are collected and recorded in a requirement specification document.
* **System Design** − In this phase, the requirements from the first phase will be reviewed, and the device design will be prepared. This system design helps determine the specifications of the hardware and system and helps define the overall system architecture.
* **Implementation** − The system is first developed with inputs from the system design into small programs called units, which are integrated into the next phase. Each unit is developed and tested for their functionality, known as Unit Testing.
* **Integration and Testing** − After testing each unit, all the units built during the implementation process are incorporated into a system. Any flaws and deficiencies are checked for the post-integration of the entire system.
* **Deployment of system** − Once testing is done, the product is deployed in the customer environment or released into the market.
* **Maintenance** − Some issues arise in the client environment. Patches are released to fix those issues. Some improved versions are released to enhance the product. Maintenance is done in the consumer environment to make such improvements.



Şelale yaklaşımında, tüm yazılım geliştirme süreci ayrı aşamalara ayrılmıştır. Genellikle, bu modelde, bir fazın sonucu bir sonraki fazın girişi olarak sırayla hareket eder.

Şelale modelindeki sıralı aşamalar;

• **Gereksinim Toplama ve analiz** - Bu işlem sırasında, oluşturulacak tüm olası sistem gereksinimleri bir gereksinim spesifikasyon belgesinde toplanır ve kaydedilir.

• **Sistem Tasarımı** - Bu aşamada, ilk aşamadaki gereksinimler gözden geçirilecek ve cihaz tasarımı hazırlanacaktır. Bu sistem tasarımı, donanım ve sistem özelliklerinin belirlenmesine ve genel sistem mimarisinin tanımlanmasına yardımcı olur.

• **Uygulama** - Sistem ilk olarak sistem tasarımından, bir sonraki aşamaya entegre edilen üniteler adı verilen küçük programlara girdilerle geliştirilir. Her birim, Birim Testi olarak bilinen işlevleri için geliştirilmiş ve test edilmiştir.

• **Entegrasyon ve Test** - Her üniteyi test ettikten sonra, uygulama işlemi sırasında oluşturulan tüm üniteler bir sisteme dahil edilir. Herhangi bir kusur ve eksiklik, tüm sistemin sonradan entegrasyonu için kontrol edilir.

• **Sistemin dağıtımı** - Test yapıldıktan sonra, ürün müşteri ortamında dağıtılır veya piyasaya sunulur.

• **Bakım** - İstemci ortamında bazı sorunlar ortaya çıkar. Bu sorunları gidermek için yamalar yayınlandı. Ürünü geliştirmek için bazı geliştirilmiş versiyonlar yayınlandı. Bu gibi iyileştirmeler için tüketici ortamında bakım yapılmaktadır.

### Application

Each software developed is different and requires an appropriate SDLC approach based on both internal and external factors to be followed. Many situations where the most effective use of the Waterfall model are;

* The requirements are documented very well.
* The definition of a product is stable.
* Technology is comprehensible.
* Requirements are not ambiguous.
* Ample resources are available with the required expertise to support the product.
* The project is short.

Geliştirilen her yazılım farklıdır ve hem iç hem de dış faktörlere dayanan uygun bir SDLC yaklaşımı gerektirir. Şelale modelinin en etkili kullanımının olduğu birçok durum;

• Gereklilikler çok iyi belgelenmiştir.

• Bir ürünün tanımı kararlıdır.

• Teknoloji anlaşılabilir.

• Gereksinimler belirsiz değil.

• Ürünü desteklemek için gerekli uzmanlığa sahip geniş kaynaklar mevcuttur.

• Proje kısadır.

### Advantages

Some of the major advantages of the Waterfall Model are as follows;

* It allows control and departmentalization.
* A schedule may be set with deadlines for each stage of development, and a product may proceed one by one through phases of the development process model.
* Every development phase proceeds in strict order.
* Easy and simple to use and understand.
* Easy to manage because of model rigidity. There are specific deliverables and a review process for each phase.
* Phases are processed one at a time and completed.
* Works well for smaller projects where requirements are very well understood.
* Phases clearly defined.
* The coding and testing steps are very short, as the requirements and design are clearly defined during the analysis and design phases.
* The number of errors during the test phase is very small.
* Tasks are easy to arrange.
* There is good documentation of the process and results.

Şelale Modelinin en önemli avantajlarından bazıları şunlardır;

• Kontrol ve departmanlaşmaya izin verir.

• Her geliştirme aşaması için son tarihler içeren bir program oluşturulabilir ve bir ürün, geliştirme süreci modelinin aşamaları boyunca birer birer ilerleyebilir.

• Her geliştirme aşaması katı bir şekilde ilerler.

• Kolay ve kullanımı ve anlaşılması kolaydır.

• Model sertliği nedeniyle yönetimi kolaydır. Her aşama için spesifik çıktılar ve bir inceleme süreci vardır.

• Aşamalar birer birer işlenir ve tamamlanır.

• Gereksinimlerin çok iyi anlaşıldığı küçük projeler için iyi çalışır.

• Aşamalar açıkça tanımlanmıştır.

• Analiz ve tasarım aşamalarında gereksinimler ve tasarım açıkça tanımlandığından kodlama ve test adımları çok kısadır.

• Test aşamasındaki hata sayısı çok azdır.

• Görevleri düzenlemek kolaydır.

• Sürecin ve sonuçların iyi bir dökümantasyonu vardır.

### Disadvantages

Some of the major disadvantages of the Waterfall Model are as follows;

* It does not allow a great deal of reflection or revision.
* Once an application is in the phase of testing, it is very hard to go back and alter something that was not well-documented.
* No working software is produced until late during the life cycle.
* There is a high amount of risk and uncertainty.
* The time-loss due to the upper phase errors is quite high.
* It is not a good model for complex and object-oriented projects.
* It is a poor model for long and ongoing projects.
* It is not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty are high with this process model.
* The progress in phases can be difficult to measure.
* The product has to wait until the end of all the phases.
* It can not adapt to changing requirements.
* Scope adjustment over the life cycle can end a project.

Şelale Modelinin bazı önemli dezavantajları şunlardır;

• Çok fazla düşünmeye veya gözden geçirmeye izin vermez.

• Bir uygulama test aşamasındayken, iyi belgelenmemiş bir şeyi değiştirmek ve geri dönmek çok zordur.

• Yaşam döngüsü boyunca geç saatlere kadar hiçbir çalışma yazılımı üretilmez.

• Yüksek miktarda risk ve belirsizlik vardır.

• Üst faz hatalarından kaynaklanan zaman kaybı oldukça yüksektir.

• Karmaşık ve nesneye yönelik projeler için iyi bir model değildir.

• Uzun ve devam eden projeler için kötü bir modeldir.

• Gereksinimlerin orta ila yüksek değişim riski altında olduğu projeler için uygun değildir. Dolayısıyla, bu süreç modeli ile risk ve belirsizlik yüksektir.

• Fazlardaki ilerlemeyi ölçmek zor olabilir.

• Ürün, tüm aşamaların sonuna kadar beklemek zorundadır.

• Değişen gereksinimlere uyum sağlayamaz.

• Yaşam döngüsü boyunca kapsam ayarlaması bir projeyi sonlandırabilir.

## DevOps Model

### What is DevOps

The term DevOps is the short form of **Development** and **Operations**.

It focuses on collaboration between developers and other roles.

DevOps is a practice that allows a single team to manage the entire application development life cycle, that is, development, testing, deployment, operations.

DevOps aims to shorten the system’s development life cycle while delivering features, fixes, and updates frequently in close alignment with business objectives.

DevOps is an evolution of the Agile Model of software development.

As the Agile model addressed the gap between clients and developers, DevOps addressed the gap between Developers and Operations.

The development team will submit the application to the operations team for implementation.

The operations team will monitor the application and provide relevant feedback to developers.

DevOps terimi Geliştirme ve Operasyonların kısa biçimidir.

Geliştiriciler ve diğer roller arasındaki işbirliğine odaklanmaktadır.

DevOps, tek bir ekibin uygulama geliştirme yaşam döngüsünün tamamını, yani geliştirme, test etme, dağıtma, işlemleri yönetmesine izin veren bir uygulamadır.

DevOps özellikleri, düzeltmeleri ve güncellemeleri sık sık iş hedefleriyle uyumlu bir şekilde sunarken sistemin geliştirme yaşam döngüsünü kısaltmayı amaçlamaktadır.

DevOps, Agile yazılım geliştirme modelinin bir evrimidir.

Agile modeli müşteriler ve geliştiriciler arasındaki boşluğu ele alırken, DevOps Geliştiriciler ve Operasyonlar arasındaki boşluğu ele aldı.

Geliştirme ekibi başvuruyu uygulama için operasyon ekibine sunacaktır.

Operasyon ekibi uygulamayı izleyecek ve geliştiricilere ilgili geri bildirimleri sağlayacaktır.

### DevOps Phases

According to DevOps practices, the workflow in software development and delivery is divided into 8 phases.

**Plan** - Business owners and software development team discuss project goals and create a plan.

**Code** - Programmers then design and code the application and use tools like Git to store application code.

**Build** - Build tools like Maven and Gradle, take code from different repositories and combine them to build the complete application.

**Test** - Application is tested using automation testing tools like Selenium and Junit to ensure software quality.

**Integrate** - When testing is complete, new features are integrated automatically to the already existing codebase.

**Deploy** - Application is packaged after release and deployed from the development server to the production server.

**Operate** - Once the software is deployed, the operations team performs activities such as configuring servers and provisioning them with the required resources.

**Monitor** - Monitoring allows IT organizations to identify specific issues of specific releases and understand the impact on end-users.

DevOps uygulamalarına göre, yazılım geliştirme ve dağıtımındaki iş akışı 8 aşamaya ayrılmıştır.

**Plan** - İşletme sahipleri ve yazılım geliştirme ekibi proje hedeflerini tartışır ve bir plan oluşturur.

**Code** - Programcılar daha sonra uygulamayı tasarlayıp kodlar ve uygulama kodunu saklamak için Git gibi araçları kullanır.

**Build** - Maven ve Gradle gibi araçlar oluşturun, farklı depolardan kod alın ve tam uygulamayı oluşturmak için bunları birleştirin.

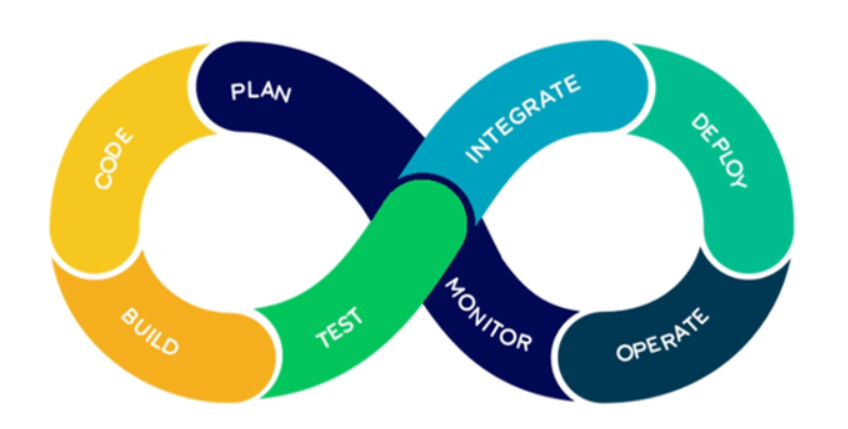
**Test** - Uygulama, yazılım kalitesini sağlamak için Selenium ve Junit gibi otomasyon test araçları kullanılarak test edilir.

**Integrate** - Test tamamlandığında, yeni özellikler mevcut kod tabanına otomatik olarak entegre edilir.

**Deploy** - Uygulama, yayınlandıktan sonra paketlenir ve geliştirme sunucusundan üretim sunucusuna dağıtılır.

**Operate** - Yazılım dağıtıldıktan sonra, operasyon ekibi sunucuları yapılandırma ve gerekli kaynaklarla sağlama gibi faaliyetleri gerçekleştirir.

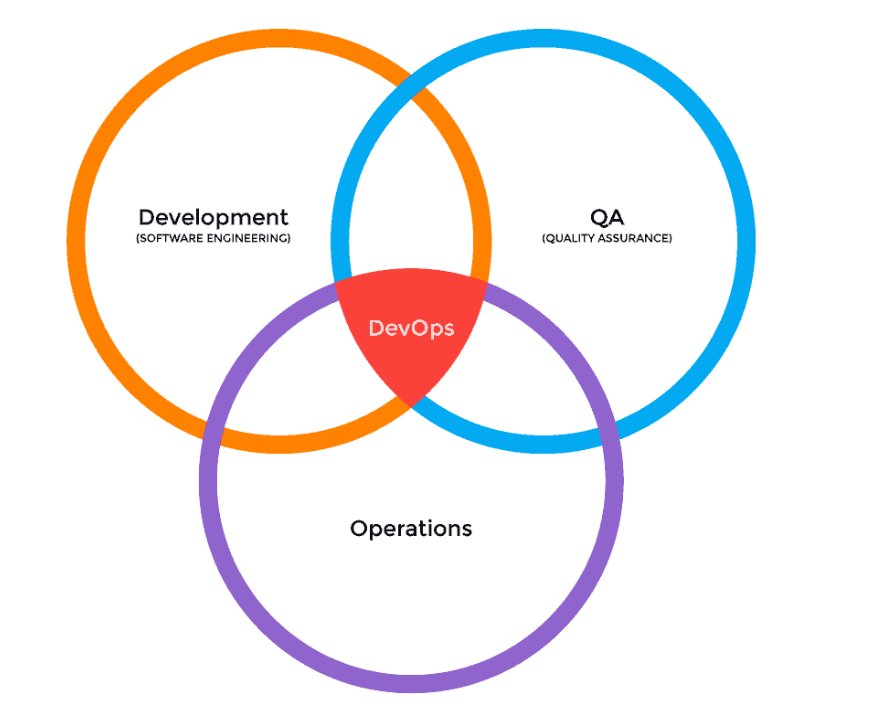
**Monitor** - İzleme, BT kuruluşlarının belirli sürümlerin belirli sorunlarını belirlemelerini ve son kullanıcılar üzerindeki etkisini anlamalarını sağlar.



### DevOps Model

DevOps is a culture that promotes collaboration between Development and Operations teams. This allows deploying code to production faster and in an automated way. It helps to increases an organization's speed to deliver applications and services.

DevOps is the practice of operations and development engineers participating together in the entire service lifecycle, from design through the development process to production support.



DevOps Modeli

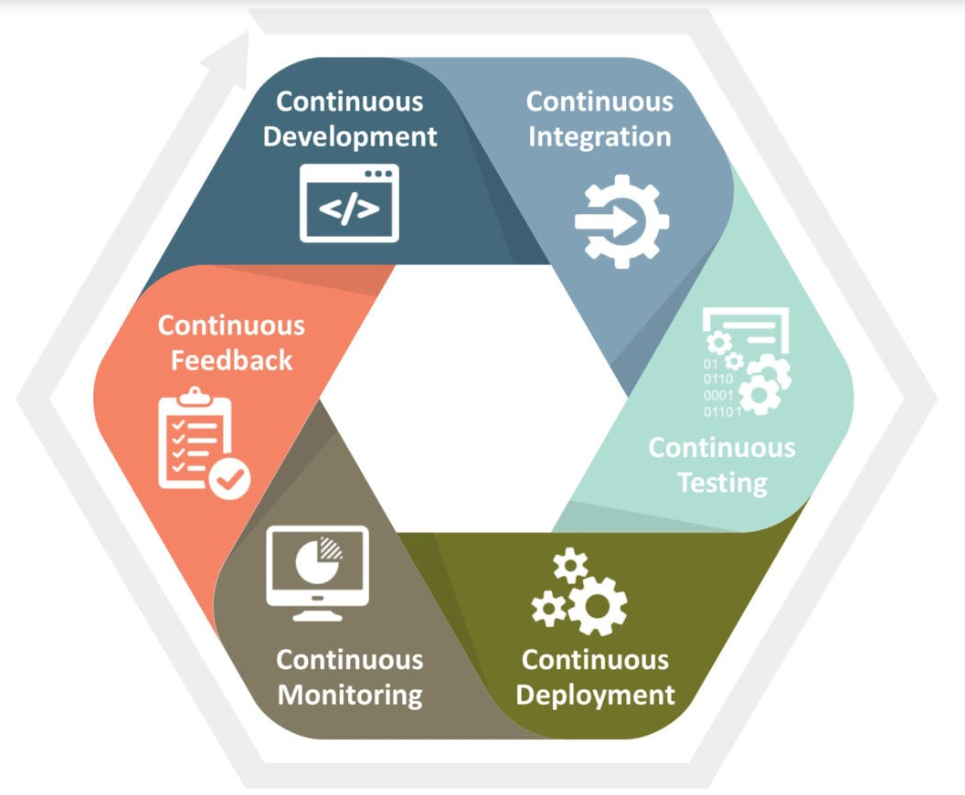
DevOps, inovasyonu ve daha kaliteli ve daha güvenilir yazılım ürünlerinin dağıtımını hızlandırmak için Geliştiriciler ve Operasyon ekipleri arasında iletişim, işbirliği, entegrasyona odaklanan yeni bir SDLC modelidir.

DevOps, Geliştirme ve Operasyon ekipleri arasındaki işbirliğini destekleyen bir kültürdür. Bu, kodun üretime daha hızlı ve otomatik bir şekilde dağıtılmasını sağlar. Bir kuruluşun uygulama ve hizmet sunma hızını artırmaya yardımcı olur.

DevOps, tasarımdan geliştirme sürecine ve üretim desteğine kadar tüm hizmet yaşam döngüsüne birlikte katılan operasyon ve geliştirme mühendislerinin uygulamasıdır.

### DevOps Processes

DevOps Model consists of various stages such as continuous development, continuous integration, continuous testing, continuous deployment, continuous monitoring, and continuous feedback.



**Continuous Development**

This is the phase that involves **planning** and **coding** of the software. The vision of the project is decided during the planning phase and the developers begin developing the code for the application. There are no DevOps tools that are required for planning, but there are a number of tools for maintaining the code.

**Continuous Testing**

This is the stage where the developed software is continuously tested for bugs. For Continuous Testing, automation testing tools like Selenium, TestNG, JUnit, etc are used.

**Continuous Integration**

This stage is the **heart** of the entire DevOps life cycle. It is a software development practice in which the developers require to commit changes to the source code more frequently.

This may be on a daily or a weekly basis. Every commit is then built and this allows early detection of problems if they are present. Building code not only involves compilation but it also includes code review, unit testing, integration testing, and packaging.

The code supporting new functionality is continuously integrated with the existing code. Since there is continuous development of software, the updated code needs to be integrated continuously as well as smoothly with the systems to reflect changes to the end-users.

**Jenkins** is a very popular tool used in this phase.

**Continuous Deployment**

This is the stage where the code is deployed to the production servers.

It is also important to ensure that the code is correctly deployed on all the servers.

**Continuous Monitoring**

This is a very crucial stage of the DevOps life cycle, where it is continuously monitored the performance of the application.

Here vital information about the use of the software is recorded.

This information is processed to recognize the proper functionality of the application.

The system errors such as low memory, server not reachable, etc. are resolved in this phase.

The root cause of any issue is determined in this phase.

It maintains the security and availability of the services.

### DevOps Processes

DevOps Modeli, sürekli geliştirme, sürekli entegrasyon, sürekli test, sürekli dağıtım, sürekli izleme ve sürekli geri bildirim gibi çeşitli aşamalardan oluşur.

Sürekli gelişim - **Continuous Development**

Bu, yazılımın planlanmasını ve kodlanmasını içeren aşamadır. Projenin vizyonuna planlama aşamasında karar verilir ve geliştiriciler uygulama için kod geliştirmeye başlar. Planlama için gerekli DevOps araçları yoktur, ancak kodu korumak için bir dizi araç vardır.

Sürekli Test - **Continuous Testing**

Bu, geliştirilen yazılımın hatalar için sürekli olarak test edildiği aşamadır. Sürekli Test için Selenyum, TestNG, JUnit, vb. Otomasyon test araçları kullanılır.

Sürekli Entegrasyon - **Continuous Integration**

Bu aşama, tüm DevOps yaşam döngüsünün kalbidir. Geliştiricilerin kaynak kodunda daha sık değişiklik yapmalarını gerektiren bir yazılım geliştirme uygulamasıdır.

Bu günlük veya haftalık olarak olabilir. Her taahhüt daha sonra oluşturulur ve bu, eğer varsa problemlerin erken tespit edilmesine izin verir. Building code sadece derleme değil, aynı zamanda kod inceleme, birim testi, entegrasyon testi ve paketlemeyi de içerir.

Yeni işlevselliği destekleyen kod, mevcut kodla sürekli olarak entegredir. Yazılımın sürekli gelişimi olduğundan, güncellenen kodun sürekli olarak ve son kullanıcılardaki değişiklikleri yansıtacak şekilde sistemlerle sorunsuz bir şekilde entegre edilmesi gerekir.

Jenkins bu aşamada kullanılan çok popüler bir araçtır.

Sürekli Dağıtım - **Continuous Deployment**

Bu, kodun üretim sunucularına dağıtıldığı aşamadır.

Kodun tüm sunuculara doğru bir şekilde dağıtıldığından emin olmak da önemlidir.

Sürekli izleme - **Continuous Monitoring**

Bu, uygulamanın performansını sürekli olarak izlendiği DevOps yaşam döngüsünün çok önemli bir aşamasıdır.

Burada yazılımın kullanımı hakkında hayati bilgiler kaydedilir.

Bu bilgiler, uygulamanın uygun işlevselliğini tanımak için işlenir.

Düşük bellek, sunucuya erişilemiyor vb. Sistem hataları bu aşamada çözülür.

Herhangi bir sorunun temel nedeni bu aşamada belirlenir.

Hizmetlerin güvenliğini ve kullanılabilirliğini korur.

### DevOps Principles

Some of the DevOps principles are;

* Create production-like systems for development and testing environment
* Deployments need to iterative and frequent. Ensure a reliable and repeatable process
* Continuously monitor and validate operational quality characteristics
* Amplify feedback loop
* Geliştirme ve test ortamı için üretime benzer sistemler oluşturun
* Dağıtımların yinelemeli ve sık sık yapılması gerekir. Güvenilir ve tekrarlanabilir bir süreç sağlamak
* Operasyonel kalite özelliklerini sürekli olarak izleyin ve doğrulayın
* Geri bildirim döngüsünü güçlendirin

### Advantages

Some of the DevOps advantages are;

* Time taken to create and deliver software is reduced
* The complexity of maintaining an application is reduced
* Improved collaboration between developers and operations team
* Continuous integration and delivery ensure faster time to market
* Yazılım oluşturmak ve sunmak için gereken süre azalır
* Bir uygulamayı korumanın karmaşıklığı azalır
* Geliştiriciler ve operasyon ekibi arasında iyileştirilmiş işbirliği
* Sürekli entegrasyon ve teslimat, pazara daha kısa sürede ulaşılmasını sağlar

### Disadvantages

Some of the DevOps disadvantages are;

* Technology investments in the automation tools required for DevOps are **costly** and will take a great deal of time to identify and implement.
* It needs a **specialist** who can cover each stage of the software delivery pipeline rather than investing in a smaller number of full-stack developers.
* It is easy to add new features, but more features may not always better, even if they can be implemented efficiently.

• DevOps için gerekli otomasyon araçlarına yapılan teknoloji yatırımları maliyetlidir ve tanımlanması ve uygulanması çok zaman alacaktır.

• Daha az sayıda tam yığın geliştiriciye yatırım yapmak yerine yazılım dağıtım hattının her aşamasını kapsayabilecek bir uzmana ihtiyaç duyar.

• Yeni özellikler eklemek kolaydır, ancak daha fazla özellik verimli bir şekilde uygulanabilse bile her zaman daha iyi olmayabilir.