# Софтуерни проекти

## Проект

Проектът е начинание, което се характеризира с начало и край на изпълнението си (т.е. е ограничено във времето) и цели създаването на определен резултат. Тъй като всеки проект е уникален, създаденият от него резултат също е единствен по рода си. По време на изпълнението на проекта се сформира екип от участници в него, на които се назначават определени роли.

## Софтуерен проект

Софтуерният проект цели създаването на специфична софтуерна система или софтуерно приложение. Това може да бъде софтуер, предназначен за употреба от конкретен поръчител (custom software), или софтуер, който да бъде продаван на широк кръг от потенциални купувачи (commercial software).

## Софтуерен процес

Това е структура на дейностите по разработката на софтуерна система. В основата на всеки софтуерен проект лежи подбран за целта софтуерен процес, който определя всички извършвани дейности, ролите на участниците в проекта и временните и крайни резултати, които се очакват от изпълнението на проекта.

## Софтуерен продукт

Софтуерният продукт е софтуерна система или приложение, получено като краен резултат от даден софтуерен проект.

## Заинтересовани лица

Заинтересовани лица (stakeholders) се наричат всички лица, имащи някакво отношение към разработвания софтуерен продукт. Това са клиентите/купувачите, потребителите, експертите в областта, юристите, свързани с проекта и др.

# Изисквания в софтуерен проект

## Видове изисквания

### Бизнес изисквания

Бизнес изискванията към една софтуерна система описват с термини от бизнеса какво трябва да бъде създадено или получено в резултат на проекта.

*Пример:* Да бъде разработено софтуерна система, която позволява поддръжката на онлайн банкиране, гарантираща пълна сигурност на данните и дейностите, извършвани с тях, според държавната нормативна уредба и вътрешния правилник на банката.

### Изисквания към процеса

Това са изисквания, които налагат ограничения върху процеса на софтуерната разработка, например спазването на определен модел на разработка или документирането на разработката по специфичен начин.

*Пример:* Процесът на разработка на системата трябва да следва еволюционния модел и да осигурява подробна документация за извършените технически промени в системата при всеки следващ еволюционен цикъл.

### Изисквания към продукта

Това са всички изисквания към софтуерния продукт, който трябва да бъде създаден.

#### Функционални изисквания

Това са изисквания към функционалността (поведението) на софтуерния продукт.

*Пример:* При избиране на опцията „Банкови сметки“ от главното меню на системата, трябва да се отвори екран със списък на всички активни банкови сметки на потребителя.

#### Нефункционални изисквания

Нефункционалните изисквания са ограничения върху разработката на продукта (архитектура, платформа, езици за програмиране, документиране на кода, преизползване на компоненти) или върху самия продукт (производителност, надежджност, сигурност).

*Пример:* Системата трябва да бъде реализирана с обектно-ориентиран език за програмиране.

*Пример:* Сървърът на системата трябва да може да обслужва едновременно 200 клиента без нарушение на качеството на услугата.

#### Изисквания на предметната област

Това са изисквания, наложени върху продукта от външни фактори, като например международни стандарти, държавни норми или фирмени правила.

*Пример:* Цветовата гама и шрифта на всички екрани на системата трябва да бъдат съобразени с описаните в държавната нормативна уредба критерии.

## Спецификация на изискванията

В началото на всеки софтуерен проект е нужно да се съберат и формулират изискванията към него и към продукта, който ще бъде разработен. Има много различни начини за събиране на изискванията – интервюта, анкети, допитвания до експерти по предметната област и др. Често се случва изискванията на различните заинтересовани лица да си противоречат – в такъв случай е необходимо да се постигне някакъв компромис между двете заинтересовани лица.

След събирането на изискванията те трябва да се формулират изрично в документ, наречен спецификация на изискванията (Software Requirements Specification – SRS). За всяко изискване е важно:

* да е формулирано ясно и точно;
* да не противоречи на останалите;
* да не се припокрива с някое от останалите;
* да не описва повече от един критерий към продукта или процеса;
* да подлежи на проверка дали е било правилно покрито.

След създаването на спецификацията е важно тя да бъде валидирана (проверена) за съответствие с горните критерии, както и да бъде потвърдено, че в нея са описани всички изисквания към процеса и продукта. Неправилното специфициране на изискванията е много сериозен риск за един софтуерен проект и затова е наложително евентуалните проблеми в спецификацията да бъдат идентифицирани и отстранени максимално рано в изпълнението на проекта.

На базата на спецификацията на изискванията се извършва проектирането на архитектурата на продукта, имплементацията му и проектирането на тестовете за него. Именно затова това е първата стъпка от изпълнението на всеки софтуерен проект.

# Софтуерни архитектури

В пректиката на софтуерното инженерство са се наложили множество шаблони за софтуерни архитектури, които са подходящо решение за определен клас проблеми/нужди. Всеки от тези шаблони има както предимства, така и недостатъци, и при проектирането на софтуерен продукт трябва да се прецени кой шаблон е най-подходящ за конкретния случай.

## Клиент-сървър архитектура (Client-Server)

Това е разпределена архитектура, при която системата е разделена на един или повече сървъри и множество клиенти, следователно е нехомогенна. Сървърът предоставя определени услуги на клиентите. Клиентите инициират връзката със сървъра. В по-сложни системи е възможно едно звено едновременно да бъде клиент за едни звена и сървър за други.

## Трислойна архитектура (Three-Tier)

Системата е разделена на три слоя – за данни, бизнес и презентационен. Всеки слой е изолиран от реализацията на останалите. По този начин се минимизира ефектът от промени в реализацията на някой от слоевете.

## Многослойна архитектура (n-Tier)

Подобно на трислойната архитектура, системата е разделена на множество слоеве с цел локализиране на отговорността (separation of concerns) и промените.

## Peer-to-Peer архитектура

Това също е разпределена софтуерна архитектура, но при нея отделните звена не се разграничават на обслужващи и обслужвани. Всяко звено може да обслужва други звена или да бъде обслужвано от тях (едновременно или в различни моменти от време). Това прави системата хомогенна.

## Event-Driven архитектура

При тази архитектура отделните компоненти се обвързват чрез събития, за които се абонират. При настъпването на събитие, абонираните за него компоненти го обработват по специфичен за тях начин. Докато в стандартните клиент-сървър системи действията се инициират от страна на клиентите, в event-driven системите действията се инициират от компонентите, предоставящи събитията/услугите.

## Архитектура с обмяна на съобщения (Message Passing)

Системите с обмяна на съобщения представляват мрежи от обработващи звена, свързани с опашки/канали за съобщения. Обработващите звена може да отсяват съобщенията, да ги преразпределят по избрани канали към други звена, или да трансформират съобщенията в нови и да ги предават по-нататък по мрежата.

## Архитектура с общо хранилище (Shared Repository)

В центъра на тази архитектура стои общо хранилище за данни. Отделните компоненти на системата взаимодействат помежду си непряко – през данните в хранилището.

## Архитектура с разпределяне на товара (Load Balancing)

Когато софтуерната система трябва да осигурява висока производителност и/или надеждност е възможно да се използват повече от едно звена, изпълняващи еднакви функции, като заявките към тях се разпределят от специално звено (load balancer), така че натоварването на всяко звено да е съизмеримо и производителността на системата да бъде оптимална. В допълнение, при проблем в работата на едно от дублираните взена е възможно останалите да поемат неговата част от операциите без нарушаване работата на цялата система.

## Архитектура, ориентирана към услуги (Service-Oriented Architecture, SOA)

При тази архитектура системата предоставя предварително описан набор от услуги за употреба от външни системи. Най-често тези услуги се предоставят през HTTP, с помощта на обменяне на XML или JSON съобщения. Много популярен е SOAP протокола за обмяна на съобщения. Заедно с WSDL протокола за описване на предоставяните от уеб услуга действия и UDDI протокола за откриване на уеб услуги, те формират най-широко използваната платформа за предоставяне и използване на уеб услуги в Интернет.

# Планиране на софтуерния процес

В началото на изпълнението на всеки софтуерен проект е важно да се състави план на процеса – дейностите, които ще бъдат извършени, и резултатите, които ще бъдат получени от тях. Има няколко често разпространени модела на софтуерни процеси, всеки със своите предимства и недостатъци.

## Ad-hoc модел

При ad-hoc разработката не се прави специфичен план на дейностите. Решенията, свързани с проектирането, разработката и тестването се вземат на място. Този модел работи за непрофесионални или малки проекти, но в крайна сметка създава големи рискове пред изпълнението на проекта и не се препоръчва.

## Модел на водопада (Waterfall Model)

При модела на водопада софтуерния процес се разделя на 5 части:

* извличане и специфициране на изисквания;
* проектиране на системата;
* разработка на системата;
* тестване на системата;
* внедряване/предаване на системата.

Специфичното за модела на водопада е, че след приключването на всеки етап от проекта не е възможно връщане обратно към него (откъде идва и названието му). В началото на проекта се създава план за дейностите във всеки от етапите. Моделът на водопада не се справя добре с промени в изискванията в по-нататъчните етапи на софтуерния процес.

## Спираловиден модел (Spiral Model)

Спираловидният модел може да се разглежда като прилагане на модела на водопада няколко пъти. Всеки цикъл от процеса започва с извличане/обновяване на изискванията и продължава с проектиране, разработка и тестване на софтуера. Това продължава до достигане на завършен вид на продукта. Цикличният модел минимизира риска от провал на проекта поради промяна в изискванията по средата на изпълнението му.

## Прототипен модел (Prototype Model)

При прототипния модел проекта започва със специфициране на изисквания и бърза разработка на прототип на системата с цел откриване на потенциални проблеми и рискове, както и доказване на нейната приложимост. Има два вида прототипи:

* throwaway прототипи – те не се използват при разработване на самия продукт;
* еволюционни прототипи – реалният продукт се доразвива от самите тях.

## Итеративен модел (Iterative Model)

Итеративният модел е комбинация на спираловидния и прототипния модели – разделя процеса на отделни итерации, всяка от които започва със специфициране на изисквания и завършва с тестване на създадения еволюционен прототип. В края на всяка итерация трябва да е получена работеща версия на продукта, която да предоставя част от функционалността на крайния продукт. Възможно е на всяка итерация да се добавя нов напълно завършен модул от системата, или пък да се развива функционалността на съществуващ от преди модул.

## Agile разработка

Традиционните модели на софтуерни процеси са свързани с генерирането на огромни количества документация, която е необходимо за подробно следене на дейностите. Това води до много допълнителен труд за създаването и поддръжката на тези документи, което оскъпява проектите и удължава времето за изпълнението им.

Agile подходите целят минимизирането на излишната документация, без да се нарушава качеството на разработвания софтуерен продукт.

Има много различни парадигми, които спадат към Agile разработката. Няколко общи черти са:

* Отдава се голямо внимание на пряката комуникация с клиента и другите заинтересовани лица;
* Не се залага на прекалена употреба на инструменти и документи;
* Цели се разбиването на проекта на кратки (няколкоседмични) цикли (спринтове), в края на които да има работещ а версия на продукта, която да бъде одобрена от клиента;
* Предвидени са начини за идентифициране и бърза реакция при промяна в изискванията;
* Минимизират се страничните дейности, които нямат пряко отношение към създаването на работещ продукт.