# Тема 26. Съвременни софтуерни технологии

1. Софтуерен продукт и процес
    - Модел на софтуерен процес - хващаме 3 и ги описваме
2. Управление на софтуерен проект и ресурсите.
    - Участници в софтуерния процес -
3. Функционални и нефункционални изисквания
4. Анализ и проектиране на софтуерните изисквания
5. Проектиране на софтуера
    a. Обектно-ориентиран дизайн
    b. Езици за описание.
        - UML
6. Управление на качеството на процеса на създаване на софтуер - верификация и валидация на софтуера. Тестване на софтуера. Управление на процеса на тестване
7. Съвременни софтуерни технологии - по 1-2 изречения за всяко
    a. Гъвкави (agile) софтуерни технологии.
    b. Extreme Programming (XP)
    c. Test Driven Development
    d. Feature driven development
    e. SCRUM

# 1. Софтуерен продукт и процес

**def. Софтуерен процес** наричаме структуриран набор от дейности, необходими за разработване на софтуерна система в срок и с високо качество по предвидим начин и с предвидими характеристики;

**def. Модел на софтуерен процес** представлява опростено(абстрактно) описание на софтуерен процес, представен от определена гледна точка

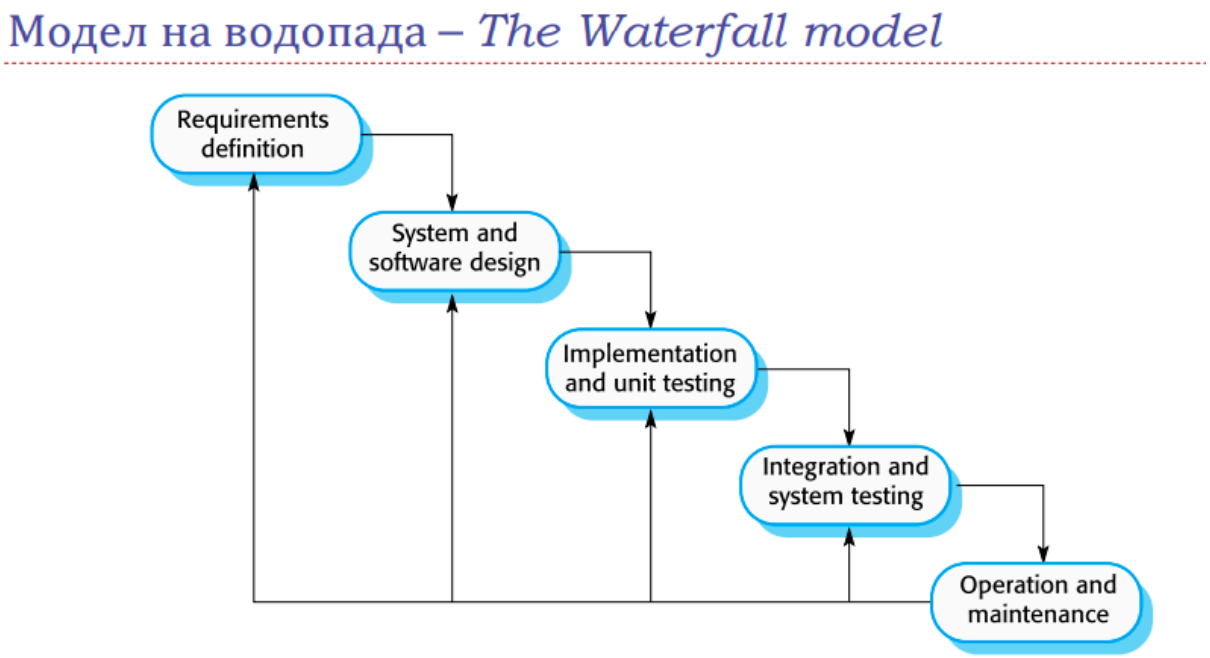
напр. такива модели са каскадния(Waterfall), модела на бързата разработка(Rapid Application Development model), фазовите(еволюционни модели като напр. инкременталния и постъпковия), прототипния и спираловидния, Personal software process

**Забл.** Всеки процес на разработка на каквото и да е вкл. 5 компонента:

- Анализ на изискванията
- Проектиране - решаваме как да се направи
- Имплементиране
- Тестване(Валидация, Верификация)
- Поддръжка

Тези елементи са в съответствие с Waterfall модела.

- **Waterfall** модел(Модел на водопада)




  - Ясно разграничен процес, който е лесен за разбиране

    - Всяка дейност трябва да бъде напълно завършена, преди да се премине към следващата
    - Ясно са дефинирани входовете и изходите на дейностите, както и интерфейсите между отделните стъпки
    - Ясно са дефинирани ролите на разработчиците на софтуер

  Waterfall моделът е подходящ, когато:
    - изискванията са осъзнати и ясно формулирани в началото;
    - проектите са ясно организирани
    - проектите са повторяеми и/или големи, за които времето и бюджетът не са критични.

  Не би бил подходящ в случаите, когато:
    - е трудно за потребителя да формулира всичките си изисквания в началото
    - биха се менили изискванията на клиента
    - разделянето на проекта няма как да се стане по гъвкав начин

  Той е предназначен по-скоро за статично дефинирани проекти - като тези от областта на хардуера, откъдето е произлязъл.

- **Инкрементален**(постъпков) модел
    - Системата не се доставя като едно цяло, а вместо това процесът на разработка и доставянето са разделени на стъпки, като всяка стъпка доставя само част от цялата функционалност
    - На идентифицираните потребителски изисквания се присвояват приоритети и тези с по-висок се реализират в първите стъпки
    - След като се започне разработката на една стъпка, изискванията не се променят
- **Итеративен** модел
    - В самото начало доставя цялостната софтуерна система, макар и част от функционалността да е в примитивна форма
    - При всяка следваща итерация не се добавя, а само се усъвършенства съществуващата
    - Обикновено върви ръка за ръка с инкременталния, в противен случай се случва зацикляне
- **Прототипен** модел
    - Създаване на основните потребителски интерфейси, без да има някакво значително кодиране
    - Разработване на съкратена версия на системата, която изпълнява ограничено подмножество от функции
    - Използване на съществуваща система или компоненти от система, за да се демонстрират някои функции, които ще бъдат включени
    - Прилага се в проекти, където не са достатъчно ясни изискванията на потребителите и дизайнът на софтуерната система
    - Използва се както самостоятелно, така в комбинация с други модели на процеси

Изборът на конкретен модел зависи основно от организационната среда и същността на приложението.

# 2. Управление на софтуерен проект и ресурсите

Управлението на софтуерните проекти е от огромна важност за спазването на **бюджетните и времевите ограничения**, както и за удовлетворяването на **изискванията относно качеството на продукта, заедно с изискванията към организацията на работата**. Така основните дейности, свързани с управлението на софтуерния проект, са **управление на риска** и **управление на човешките ресурси**.

Тъй като софтуерният процес включва всички дейности от разработката на софтуер, дейностите от по-високо ниво като спецификация, разработка, валидация и развитие, са част от него. В зависимост от избрания  модел, следните участници са въвлечени в софтуерния процес:

- бизнес анализатори
- софтуерни архитекти
- отбор от разработчици
- тестови инженери
- крайните потребители на системата

Всички заинтересовани лица; различните модели на софтуерни процеси предполагат различно участие на различните заинтересовани лица, напр. при гъвкавите методологии има представител на клиента

# 3. Функционални и нефункционални изисквания

3.1. **Функционални** изисквания - описват функционалностите на системата, т.е. услугите, които системата трябва да предоставя, начинът по който системата трябва да реагира на конкретни входни данни и поведението й в конкретни ситуации; отговарят на въпроса “какво”
   - напр. потребителите на системата трябва да могат да сортират резултатите от търсене във възходящ ред

3.2. **Нефункционални** изисквания(**качествени**) - ограничения върху услугите или функционалността на системата като времеви ограничения, ограничения върху процеса на разработване и използваните стандарти и пр.; отговарят на въпроса “как”
   - напр. заявка към базата данни трябва да бъде изпълнена за не повече от 10 ms.

Нефункционалните изисквания могат да бъдат по-критични от функционалните и от тях може да зависи пригодността на цялата система.

# 4. Анализ и проектиране на софтуерните изисквания

Включва набор от дейности, които предполагат точно систематично определяне и извличане на изискванията от всички заинтересовани лица; резултат би трябвало да бъде множество от добре формулирани изисквания, форматирани по определен стандарт.

- ФИ - с use case сценарии, описание на потока на изпълнение

  Може да сложим примерна схема и табличките за описание на изискванията(предусловие, постусловия, поток на изпълнение, резултат, вкл. актьори)

- Качествените изисквания(НФИ) влияят в/у избора на софтуерната архитектура

# 5. Проектиране на софтуера

Най-съществения резултат от дейностите по проектиране е софтуерната архитектура, т.е. съвкупност от структури, показващи различните софтуерни елементи, външно видимите им свойства и връзките между тях. Структура още наричаме изглед, т.е. конкретно представяне на софтуерната система и представлява начин на по-високо ниво заинтересованите лица да получат разбиране за СА на системата.
Например, такива са декомпозиция на модулите, употреба на модулите, структура на процесите

- **Модулни структури** - елементите в модулните структури са модули - единици работа за изпълнение; те предлагат поглед, ориентиран към реализацията на системата, без значение какво става по време на изпълнение
  - Връзките между модулите са от вида “X използва Y”, но ако има нужда от по-детайлно описание, може връзките да са насочени към конкретен интерфейс или ресурс на модула
- **Структура на процесите**
  - елементите са компоненти, които се проявяват по време на изпълнение(т.е. основни изчислителни процеси) и средствата за комуникация м/у прицеите
- **Структура на внедряването** - показва как софтуера се разполага в/у хардуера и комуникационното оборудване
  - Елементите са процеси, хардуерни устройства и комуникационни канали
  - Представлява интерес при разпр. системи, позволява да се разберат особеностите относно бързодействието, интегритета на данните, надеждността и сигурността

## 5.1. Обектно-ориентиран дизайн и моделиране на системата

Представя системата като група от обекти, които си взаимодействат и имат изградена структура, чрез което се постига конкретна цел;
Именно елементите са обектите, а конекторите са интерфейсите между тях.

**def.** **Обект** - атомична единица(същност), която притежава състояние(стойностите на всичките му атрибути), има дефинирано поведение(под формата на параметризирани методи) и може да се идентифицира уникално
**def.** **Клас** - шаблонът по който се създават инстанции, т.е. обектите

**Забл.** Класовете ни дават възможност да скрием вътрешното представяне или състояние на обектите. Важни характеристики на този тип проектиране са абстракция, енкапсулация, наследяване и полиморфизъм.

**Забл.** Стига се до ОО дизайн след ОО анализ, т.е. дефинирането на проблема и изискванията в термините на обекти.

## 5.2. Езици за описание

Езиците за описание са Domain Specific езици, използвани в областта на софтуерните архитектури, въвеждащи ниво на формализъм и предоставящи синтаксис за характеризирането на софтуерните архитектури, отново за да стандартизират процеса по комуникиране на софтуерната архитектура на по-високо ниво. Използването на тези езици изисква поддръжка от инструменти за анализиране, визуализиране, анализ и т.н., които са специфични за всеки език.

След 2000-та година втората версия на UML се разширява така че да обхваща елементи на тези езици на описание.

### 5.2.1. UML

UML е ОО език за моделиране, който служи за уточняване, визуализиране, конструиране и документиране на елементите на софтуерните системи. Също се използва за бизнес моделиране и моделиране на други системи, които не са софтуерни.

UML диаграмите представят решенията след ОО анализа и проектирането.
Според йерархията им, някои видове са:

- диаграми на поведението - Activity, State Machine Diagrams, Use Case Diagrams, Interaction Diagrams и др.
- структурни диаграми - Class Diagrams, Object Diagrams, Deployment Diagrams и др.

Например, за изгледите, които разгледахме по-горе, можем да използваме за моделиране следните UML диаграми:

- class диаграми и package диаграми - модулни
- sequence, activity, state charts - структури на процесите
- deployment диаграма - за физическия(на внедряването) изглед

# 6. Управление на качеството на процеса на създаване на софтуер - верификация и валидация на софтуера. Тестване на софтуера. Управление на процеса на тестване.

- **Верификация** на софтуера vs **Валидация** на софтуера
  Валидацията е процеса по проверка дали спецификацията покрива изискванията на клиентите, докато верификацията(дали разработчиците са си свършили работата) представлява процеса по проверка дали софтуерният продукт покрива тези изисквания.
  **Забл.** Алфа-бета **тестване** - валидация

- В този смисъл, **тестването** е форма на **верификация** за установяване на съответствието на софтуера с изискванията(макар че има и други методи на верификация, напр. формална верификация)
  - Функционално тестване - тества се функционалността на цялостната система
  - Потребителско тестване - тества ползата и използваемостта на продукта от крайните потребители
  - Тестване на производителността - напр. тестване на бързодействието на системата
  - Тестване на сигурността - тестване на степента на защита на потребителската информация

- **Управление на процеса на тестване**
  1. Анализ на изискванията за тестване
  2. Планиране на тестването - какви данни ще ползваме, какво ще тестваме, как ще го тестваме
  3. Изпълнение на тестването
  4. Анализ на резултатите
  5. Заключение

# 7. Съвременни софтуерни технологии

## 7.1. Гъвкави (agile) софтуерни технологии. Extreme Programming (XP)

- **Agile** методологиите вземат предвид изискванията на пазара за бързо доставяне на софтуерни продукти. При Agile фокусът е върху предоставянето на функционалностите бързо, в отговор на променящите се изисквания.

  Всички Agile методологии се основават на инкременталната разработка и доставяне на софтуера. При инкременталната разработка се започва с приоритизирането на функционалностите, които са най-важни, дефинират се детайлите на тази функционалност, след което се реализира, тества, интегрира, тества се след интеграция цялостно системата и евентуално се доставя, при успех.

  Отличително за agile е съвместната работа и комуникация между разработчиците и клиентите с цел предоставяне и приоритизиране на нови системни изисквания и оценяването на всеки инкремент.

- Extreme Programming е тясно свързано с agile методологиите. Основните му принципи са:
  - **Инкрементално планиране/потребителски истории(Feature Driven Development - )** - няма дефинирана крайна точка на разработка; вместо това, изискванията за всеки инкремент се установяват след дискусии(потребителски истории) с представители на клиентите, както и техния приоритет; това се случва итеративно
  - **Малки releases** - версиите на системата се доставят често и инкрементално добавят функционалности, не твърде много на брой и не твърде големи
  - **TDD**(Test-driven development) - първо се пишат тестовете за функционалностите, след което съответният код. Това допринася за определяне на поведението на кода и гарантира, че по всяко време, т.е. след всяка промяна може да се тества написаното.
  - **Continuous Integration** - веднага щом работата по дадена задача се приключи, се интегрира в системата и се създава нова версия на системата, което се последва от автоматично тестване на цялостната система, което е необходимо да завърши с успех, преди новата версия да бъде приета
  - **Рефакториране** - подобряване на структурата, четимостта, ефикасността и сигурността на системата
  - **Простота на дизайна** - проектирането е дотолкова, че да може да се удовлетворят текущите изисквания
  - **Pair programming** - програмиране по двойки с цел оценка и предоставяне на взаимопомощ при разработката
  - **Collective ownership** - разработчиците работят по разнообразни части от системата, така че знанията и експертизата им са по-равномерно разпределени

- **On-site customer** - представител на клиентите е наличен през процеса на разработка и има отговорността да предоставя системните изисквания на екипа за имплементиране

## 7.2. SCRUM

Scrum е гъвкав метод, който предоставя рамка за agile организация на проектите и планиране

**Ключови практики**

- **Product backlog** - TODO списък със задачи, които предстои да бъдат имплементирани; този списък се преглежда и обновява преди всеки sprint
- **Timeboxed sprints** - фиксирани периоди от време(напр. 2 - 4 седмици), през които се случва реализирането на функционалностите от backlog списъка
- **Самоорганизиращи се екипи** - екипи, които работят и дискутират проблеми, вземат решение чрез консенсус