**1.Софтвер(дефиниција, карактеристики, видови)**

**Софтверот се:**

1) инструкции (компјутерска програма) кои кога се извршуваат ги обезбедуваат посакуваните особини, функции и перформанси;

2) податочни структури кои овозможуваат програмите адекватно да ги манипулираат информациите;

3) описни информации, како копија на хартија или виртуелна форма кои го опишуваат работењето и употребата на програмите.

**Различни аспекти кон софтверот:**

-Од аспект на софтверските инженери софтверот е составен од програми, документи и податоци потребни за да се развие софтверскиот систем

-Од аспект на корисникот софтверот е само продукт кој ги задоволува или не ги задоволува нивните барања и им ја олеснува работата

**Софтверот го испорачува највредното нешто во денешно време ИНФОРМАЦИЈА**

**Карактеристики на софтверот**

-Софтверот се јавува и како финален продукт и како алатка за развој на други продукти.

-Софтверот се развива, тој не се произведува ν Софтверот не се троши (абе), но застарува

-Најголемиот дел од софтверот се прави по нарачка ν Софтверот може да биде екстремно комплексен па поради тоа и тежок за разбирање и следење што остава многу голем простор за човечки грешки.

-Развојот на комплексен софтвер кој ќе биде без грешки (или барем тие ќе бидат сведени на прифатливо ниско ниво) е неизводливо доколку на целиот процес не се пристапи на правилен и организиран начин.

**Софтвер - домени на примена**

**Системски софтвер**

♦ (компајлери, едитори) обработува комплексни но одредени информациски структури

♦ (оперативни системи, драјвери) обработува претежно неодредени информации

Во обата случаја се карактеризира со висока инеракција/зависност од хардверот, повеќе корисници, паралелни операции и распоредување, споделување на ресурси, комплексни податочни структури, надворешни интерфејси

Real-time software ν надгледува процеси и појави од реалниот свет: се појавуваат – треба да бидат обработени во реално време. Треба да дава одредени гаранции за времетраењето на обработките.

**Апликативен софтвер**

♦ индивидуални програми кои решаваат одредени бизнис потреби (инвентари, платни списоци, сметководство)

**Инженерски и научен софтвер**

♦ CAM, CAD, симулации, комплексни нумерички пресметки, ...

**Вградлив (embedded) софтвер**

♦ вграден во продукти или системи со цел да обезбеди контролна функција за системот и/или крајниот корисник ( мобилни телефони, MP3 players, routers, микробранови печки, автомобили

**Софтвер за PC - масовен софтвер (Product-line)**

♦ дизајниран да обезбеди одредени специфични функционалности за различни корисници; фокусиран на ν специфичен пазар (магацинско работење) ν масовен пазар (word processing, spreadsheets, computer graphics, multimedia, entertainment) ν Web и мобилни апликации

♦ веб апликации на кои им се пристапува преку прегледувач (browser)

♦ апликации кои се поставени на мобилни уреди

**Софтвер за вештачка интелигенција**

♦ роботика, експертни системи, препознавање на облици (слика, говор)

**Legacy software**

-Стар „наследен“ софтвер (10+ години)

♦ понекогаш од одредени причини засегнатите сакаат да го одржат во функција

♦ долготраен, менуван и дополнуван

♦ на него се потпираат некои од основните бизнис функции во одредена организација - и критичен за бизнисот - затоа е „незаменлив“

За жал, кај ваквите системи често се среќава и:

♦ слаб квалитет

♦ нефлексибилен дизајн

♦ неразбирлив код

♦ слаба или никаква документација

♦ слабо одржувана историја на измени

**Нови предизвици во развој на софтвер**

-Open world computing

-Netsourcing

-Open-source

**Структура на продажната цена на софтверот**

Вообичаено се дели на три дела:

1. up-front fee - се наплаќа на почетокот како лиценца (право) на користење на софтверот

2. maintenance – се наплаќа годишно за одржување (15-25% од 1.) – patches, upgrades, …

3. инсталација, интеграција, обука, ..

**Модели на лиценцирање**

-Subscription license – се купува правото на користење на одредено време

♦ подолг период 3 години – плаќање на рати

♦ покус период 1 месец (хостирање) ν usage license (Oracle)

-Site license

-On-demand usage-based license

-Software as a service ν “Razor and blade” model – основната апликација се нуди бесплатно, а се наплаќа за останатите делови на продуктот (Adobe)

-Service model – основната апликација се дава бесплатно а се заработува на услуги како инсталација, конфигурација, одржување, обука, ... (Red Hat)

-Advertising model – корисниците не плаќаат за употребата на софтверот, но за сметка на тоа мораат да гледаат реклами додека го користат (Google, Microsoft Live)

**Видови на софтвер**   
-Proprietary software( close source software)

-Free software (open source software)

**Видови софтвер (достапност)**

-Freeware (бесплатен)

-Shareware (бесплатен во основана / ограничена форма или за одредено време)

-Open source (бесплатен со достапен изворен код – може да се менува)

-Commercial ♦ Се продава за комерцијални цели ♦ Proprietary но може и free (Red Hat, Sun, …)

**2.Процес на развој на софтвер**

**Инженерство-**Примена на науката во решавање на социјалните проблеми.

**Што е работа на инженерот? - Дизајн**

Клучниот збор е ДИЗАЈН

Тоа е чекорот пред физичкото конструирање на решението

Опфаќа формулирање на идеите, скицирање на моделите, градење на прототипови и тестирање на дизајните. Главната цел е да се креира решение кое ќе може да се реализира. Ако решението не може да се реализира тогаш тоа е само теоретско решение и повеќе не е инженерство туку теорија.

**Софтверско инженерство-**Креирање на издржлив дизајн за остварливо софтверско решение кое треба да решава реален проблем.

Софтверските инженери употребуваат систематски и организиран пристап кон нивната работа и користат соодветни алати и техники зависни од проблемот кој треба да се реши, наметнатите ограничувања и расположливите ресурси.

**Софтверски инженери наспроти компјутерските научници.**

Софтверските инженери се разликуваат од компјутерските научници

Додека интересот на научниците кон субјектите и појавите е примарно теоретски, а секундарно практичен, интересот на инженерите е чисто практичен.

Додека научниците го откриваат и изучуваат постоечкото (фундаментални природни појави и закони) инженерите креираат работи кои никогаш не постоеле.

Софтверското инженерство акцентот го става на дизајнирањето а не на имплементацијата.

**Софтверски процес-** множество активности чија цел е развој или еволуција на софтвер.

Генерички активности кои се појавуваат во сите софтверски процеси се:

♦ Спецификација – што треба да прави системот и кои се неговите ограничувања

♦ Развој – продукција на софтверски систем

♦ Валидација – проверка дека софтверот е навистина она што купувачот барал

♦ Еволуција – промена на софтверот како резултат на променетите барања

Релевантни **концепти и техники кои се користат при реализацијата на големи софтверски проекти** и го опфаќаат целиот процес на развој на проект:

1. Анализа и спецификација на барања

2. Дизајн и архитектура

3. Кодирање и интеграција

4. Тестирање и отстранување на грешки

5. Документирање

6. Инсталација и одржување

**Модел на софтверски процес**

Поедноставена претстава на софтверскиот процес, дадена од одредена перспектива на процесот

Примери за перспектива на процесот се:

♦ Workflow perspective - sequence of activities;

♦ Data-flow perspective - information flow;

♦ Role/action perspective - who does what.

**Методи за развој на софтвер**

Структуриран пристап кон развојот на софтвер

Опис на моделот

Правила

Препораки

Водење во процесот

**Предизвици пред софтверскиот инженер**

Хетерогеност

♦ Примена на техники кои ќе овозможат развој на софтвер кој ќе може да се справи со хетерогени платформи и околини

Испорака

♦ Примена на техники кои ќе овозможат побрза испорака на софтверот

Верба

♦ Развој на техники кои ќе демонстрираат дека на софтверот може да му се верува од страна на корисниците

**Софтверско инженерство - Дефиниција (според IEEE)**

Софтверското инженерство претставува систематски, дисциплиниран и квантифициран пристап кон градењето (и одржувањето) на софтвер.

**Развој на софтвер**

Софтверското инженерство претставува структурен пристап кон развојот на софтвер.

Процесот на развивање на софтвер е итеративен процес на учење чиј резултат е „софтверски капитал“ кој содржи здобиено, рафинирано и организирано знаење.

**Софтверски процес**

Процес- збирка од активности, акции и задачи кои се изведуваат кога треба да се изведе нешто

Активност- тежнее да постигне погенерална цел без разлика на доменот, видот и големината на проектот.

Акција- множество задачи кои продуцираат главен работек артефакт.

Задача- се фокусира на потесна, добро дефинирана цел.

**Процесна рамка-** поставува основа на комплетниот процес на развој на софтвер идентификувајќи мал број на рамковни активности кои се применливи на сите видови на софтверски процеси.

**Генеричка процесна рамка за развој на софтвер**

Комуникација

Планирање

Моделирање

Конструкција

Испорака

**Umbrella Activities**

Дополнителни константни активности кои треба да обезбедат успешно и непречено одвивање на процесот:

♦ Tracking and control - Следење и контрола на софтверскиот проект

♦ Risk management - Менаџирање на ризици

♦ SQA (Software quality assurance) – Активности за да се обезбеди квалитет на софтверот

♦ Technical reviews - Формални технички прегледи (ревизии)

♦ Measurement – дефинирање и колекција на процесни и продуктни мерки

♦ Configuration management (менаџирање на ефектите на промена)

♦ Reusability management (критериуми за повторна употреба)

♦ Подготовка и продукција на работни материјали (модели, документи, записници, формулари, ...)

**Граничници(Milestones)**

Процесот на развој на софтвер е поделен на добро дефинирани задачи наречени фази. Секоја фаза завршува со граничник. Секој граничник има јасна цел која може лесно да се провери.

**Пракса**

1. Разбери го проблемот -комуникација и анализа

2. Планирај го решавањето- моделирање и дизајн

3. Спроведи го планот- кодирање

4. Провери ги резултатите - тестирање

**Генерички пристап кон фазите во софтверското инженерство**

**(Дефиниција)**- – Се фокусира на Што?

♦ Идентификација на потребните информации за системот кој треба да се направи – барања, очекувано однесување, ограничувања

♦ Системско или инженерство на информации, планирање на софтверски проект, анализа на барање

1. собирање на информации

2.планирање на проект

3.анализа на барања

**Развој** - Се фокусира на Како?

Како ќе бидат имплементирани функционалностите на системот, како ќе биде изведувано тестирањето

1. Дизајнирање на софтверот

2. Кодирање

3. Тестирање

**Поддршка** - Се фокусира на промени

♦ корекција на грешки, адаптација, подобрувања, нови барања

♦ повторно враќање на чекорите на дефиниција и развој но во контекст на постоечки систем. 4 типа на промени: 1) Корекција (на грешки)

2) Адаптација (на нов систем, околина, ...)

3) Подобрување (дополнителна функционалност надвор од иницијалните спецификации)

4) Превентивно (софтверот се „расипува“ од постојните промени (закрпи)). Промени кои треба да овозможат полесни идни корекции, адаптации и подобрувања (software reengineering).

**Фази во развојот на софтвер**

-Размена на информации помеѓу корисникот и „развивачот“ на софтверот -Оваа фаза завршува со документ наречен листа на барања

-Комплетен опис на проблемот -Листата на барања се преформулира во форма посоодветна и разбирлива за „развивачите“.Спецификациите дефинираат што точно софтверот треба да работи од перспектива на развивачот.

-Разложување на конкретни детални програмски задачи - Целта е да се дадат добро дефинирани програмерски задачи на индивидуалните програмери ν Граничникот на оваа фаза се нарекува документ за дизајн на системот

Програмирање - Во оваа фаза документот за дизајн се трансформира во извршен код (програма)-Граничник –програма.

Тестирање - Тестирањето е процес на проверка дали софтверот го прави она што треба да го прави.Граничник-тестиран софтвер.

Измени на програмата - Вообичаено клиентот наоѓа грешки и недоследности, бара измени, ... ν За да се поправат, потребни се измени во кодот а некогаш и во дизајнот на софтверот ν Променетиот софтвер треба одново да се тестира.

**Процесен модел**

-Пропишани процесни модели

-Агилни процесни модели

**Генерални принципи**

Принцип - Значајна законитост или претпоставка потребна во мисловниот систем

1. The Reason It All Exists **Причина за постоењето** - Софтверот постои за да обезбеди корисност на неговите корисници

2. KISS (Keep it simple, stupid!) **All design should be as simple as possible**, but no simpler.

3. Maintain the Vision **Одржувај ја визијата** - Јасната визија е суштествена за успехот на кој било софтверски проект

4. What You Produce, Others Will Consume **Она што ќе произведете, останатите ќе консумираат** - Секогаш специфицирајте, дизајнирајте и имплементирајте знаејќи дека некој друг ќе треба да разбере што сте направиле и зошто, за да може да го спроведе својот дел на активности

5. Be Open to the Future **Бидете отворени кон иднината** - don’t code yourself into a corner, запрашајте се „што ако...“ и бидете подготвени за одговорите

6. Plan Ahead for Reuse **Планирајте однапред за да биде повторно употребено** - ги намалува трошоците и ја зголемува вредноста и на повторно употребливите компоненти и на системот кој ги употребува

7. Think! **Размислувајте! -** Чиста и комплетна замисла пред секоја акција скоро секогаш дава подобри резултати

**Софтверски митови**

Менаџерски митови

♦ Постојат голем број на книги полни со препораки и стандарди за градење на софтверски системи

♦ Ги имаме најсовремените компјутери и алатки за развој на софтвер

♦ Ако доцниме со проектот, секогаш можеме да вработиме уште програмери за да постигнеме (Mongolian horde concept)

♦ Можеме дел од софтверот да пренесеме на друг (outsourcing)

Клиенти

♦Основните барања се доволни за да се почне изработката на софтвер, деталите може да се дополнат подоцна ♦Што ако барањата постојано се менуваат, софтверот е флексибилен и секогаш може да се промени/надгради

Практиканти (програмери)

♦Штом ќе проработи програмата која ја пишуваме нашата работа е завршена

♦Додека нe проработи програмата нема начин да се провери нејзиниот квалитет

♦Единствената работа која треба да ја испорача успешен проект е програма која работи

♦Креирањето на комплетна и обемна документација само не забавува

**3.Модели на процеси за развој на софтвер**

**Модел на процес-** стратегија за развој која ги опфаќа нивоата на процесот, методите и алатките како и генеричките фази во развојот на софтвер.

**Тек на процес**- опишува како се поставени рамковните активности, акции и задачи , и какви се нивните меѓусебни релации во рамките на процесот.

**Линеарен процесен тек** - ги извршува рамковните активности во редоследен секвенцијален редослед.

**Итеративен процесен тек** ги повторува една или повеќе од активностите пред да премине на следна.

**Еволутивен поцесен тек** ги извршува активностите во повторувачки циркуларен тек, испорачувајќи покомплетна верзија на софтверот со секое поминување низ циклусот.

**Паралелен процесен тек** извршува една или повеќе активности паралелно (симултано) со другите активности.

**Групи на задачи**

Секоја акција поврзана со некоја од рамковните активности на софтверското инженерство може да биде претставенa со одредена група на задачи. Групите на задачи се прилагодуваат кон специфичните потреби на конкретниот софтверски проект и карактеристиките на проектниот тим.

**За едноставен проект** -активноста комуникација може да се сведе на:

1. Контакт по телефон со засегнатиот

2. Дискусија околу барањата и прибелешки

3. Организација на прибелешките во куса пишан текст за барањата

4. Испраќање на текстот по електронска пошта до засегнатата страна на одобрување

**За посложени проекти** со повеќе засегнати страни активноста комуникација може да вклучува повеќе конкретни акции како:

♦ inception - почеток, зачнување

♦ elicitation - „изнудување“, прибирање на барања

♦ elaboration - елаборација (обработка)

♦ negotiation - преговарање

♦ specification - спецификација, и

♦ validation - валидација, проверка

**Шаблони** - Задачите во рамките на акција може да се изберат соодветно на специфичните потреби според типот на проектот, клиентот, тимот.

-Pattern Name

-Forces- околина во која се сретнува,

-Type (Stage(дефинира проблем поврзан со рамковна или процесн активност),

Task(дефинира проблем поврзан со акција или работна задача на софтверското инженерство),

Phase(дефинира секвенца на процесни активности кои се појавуваат во рамките на процесот),

-Initial context - – ги опишува предусловите за примена на шаблонот,

-Solution – Опишува како успешно да се имплементира шаблонот,

-Resulting Context – Ги опишува условите кои ќе важат по успешната примена на шаблонот,

-Related Patterns – листа на други процесни шаблони кои се поврзани со него,

-Known Uses and Examples.

**4.Класични(пропишани) модели за развој на софтвер**

**Пропишани процесни модели-** секој процесен модел пропишува тек на процесот- начинот на кој процесните елементи се во релација едни со други.

**Пропишани (плански водени)**

♦ waterfall (водопад)

♦ инкрементални

♦ еволуциски

♦ конкурентни

**Специјални**

♦ компонентно ориентирани

♦ формални методи

♦ аспектно ориентирани

**Unified process**

**Агилни**

**Линеарниот секвенцијален модел** наречен уште и класичен или **модел на водопад** (**waterfall model**) сугерира систематски секвенцијален пристап кој ги вклучува следниве активности:

♦ **Системско/информационо инженерство и моделирање** (бидејќи софтверот е дел од поголем систем)

♦ **Анализа на барања на софтверот**

Софтверскиот аналитичар мора да го разбере информацискиот домен на софтверот, како потребните функции, однесување, перформанси и интерфејси

♦ **Дизајн**

се фокусира на 4 атрибути на програмата:

1. податочни структури

2. софтверски архитектури

3. претстава на интерфејси

4. процедурални (алгоритамски) детали

♦ **Генерирање на код**

Преведување на дизајнот во машински читлива форма

♦ **Тестирање**

Се проверува интерната логика на софтверот(сите функции) и екстерната логика(дали за одреден влез се добива бараниот излез)

♦ **Поддршка**

– промени: поправка на грешки, прилагодување на нова околина или функционални и перформансни промени барани од клиентот

Класичен waterfall модел-   
Комуникација -> Планирање -> Моделирање -> Конструкција -> Испорака

Прифатлив пристап кога моделот е добро дефиниран

Сугерира систематски секвенцијален пристап

Најстар, најчесто користен и најкритикуван

**Предности** на waterfall моделот

-Времето потрошено во раните фази се исплати во подоцнежните. Поисплатливо е да се промени дизајнот кога има грешка отколку да се потроши време на имплементација на лош дизајн.

-Подобро е една фаза да се заврши 100% пред да се помине на друга.

-Генерирање и одржување прецизна документација

-Големите компании работат според овој модел

**Недостатоци** на waterfall моделот

-Реалните проекти ретко го следат строгиот секвенцијален модел

-Тешко е клиентот да ги постави сите свои барања на почеток

-Клиентот мора да биде трпелив бидејќи работна верзија на програмата ќе се добие многу подоцна во текот на проектот

-Blocking states-некои членови од тимот мора да чекаат други да завршат за да продолжат со својата работа(ниска продуктивност)

**Критика** на waterfall моделот

-Додека системот се имплементира барањата се промениле

-Корисниците не можат прецизно да ги дефинираат барањата на почеток

-Не нуди можност за менаџерска контрола

-Голема одвоеност на членовите на тимот

**Инкрементален модел**

Комбинација на линеарниот секвенцијален модел со итеративната филозофија на развојот на прототипови

Секоја секвенца (која се повторува) продуцира испорачлив инкремент на апликацијата. Во секоја итерација се додаваат нови можности.

За разлика од прототипови испорачува употреблив продукт.

**Еволуциски модел**

Еволуциските модели го гледаат развојот на софтверот како систем за кој однапред се знае дека ќе мора да се менува. Често пати е важно да се издаде делумно оперативна (која задоволува само дел од првичните барања) верзија на софтверот, а подоцна да се надгради.

**Модел на прототипови**

се концентрираат на деловите видливи за клиентот (input/output), целта не е долготрајна одржливост и добар дизајн (само фасада). Треба да се фрли!

Предности-Помага подобро да се разбере што треба да се изгради

Се применува кога има   
нејасно специфицирани барања (се идентификуваат областите каде е неопходна понатамошна попрецизна дефиниција)

♦ Клиентот има легитимен проблем, но не знаат да го искажат (дефинираат)

Проблем- најчесто се за една употреба и со строго одредена намена и не треба да еволуираат кон целосен систем

**Спирален модел**

Еволуциски модел кој ги комбинира итеративната природа на прототип пристапот и систематскиот аспект на линеарниот секвенцијален модел.

Софтверот се развива со серија на инкрементални изданија. Во првите фази е само на хартија или како прототип, додека во подоцнежните фази се оформува кон кончен продукт. Воведува попрецизна контрола и менаџирање на ризици.

**WINWIN спирален модел –** сличен на класичниот спирален модел. Комуникацијата со клиентот е претставена како преговарање и компромиси за цената, барањата и роковите.

**Паралелен модел**

Може да се претстави шематски со серија од технички активности, задачи и ним асоцирани состојби. Секоја од активностите може да егзистира во различна состојба. Сите активности може да бидат истовремено активни но секоја во различна состојба

Паралелниот развоен модел вообичаено се користи при развојот на клиент-сервер апликации

каде конкурентниот процес дефинира активности во две димензии

♦ системска димензија со активности design, assembly, use

♦ компонентна димензија со активности design, realization

Конкурентноста се постига на два начина:

1. Системските и компонентните активност се појавуваат симултано и можат да се моделираат со моделот ориентиран на состојби

2. Типична клиент-сервер апликации се реализира со употреба на многу компоненти од кои повеќето можат да се дизајнираат и реализираат паралелно

**Инкрементален развој**  
Поделба на развојот на фази и нивен распоред со што различни делови од системот се развиваат во различно време и со различен интензитет и се интегрираат како што се завршени

**Итеративен развој**

Се планира време за ревизија и унапредување на делови од системот (не подразбира задолжителен инкрементален развој) ♦ Продуктот се испорачува на (дел од) клиентите и се ревидира по повратните информации од нив  
Се обединуваат во Rational Unified Process(RUP)

**Компонентно базиран развој**

Commercial off-the-shelf (COTS) софтверски компоненти, понудени како продукти кои обезбедуваат одредена функционалност со прецизно дефинирани интерфејси кои може да бидат интегрирани во софтвер кој се гради

Component-based Development (CBD) моделот вклучува голем дел од карактеристиките на спиралниот модел – еволуциски, итеративен пристап кон развојот на софтвер.

CBD ги гради апликациите користејќи готови претходно креирани (и тестирани) софтверски компоненти (класи).

Процес сличен на спиралниот, со следниве чекори:

1. Пребарување низ расположливите компоненти за компоненти погодни за апликацискиот домен и нивна проценка

2. Проценка на интегралибилноста на компонентите

3. Креирање на софтверска архитектура во која се интегрираат компонентите

4. Интеграција на компонентите во архитектурата

5. Спроведување на темелно тестирање за да се осигури соодветно функционирање

Software reusability

Инженерскиот процес започнува со идентификација на кандидат компоненти и пребарување во базата на расположливи компоненти развиени во претходни проекти дали постои готова компонента која може да ги задоволи барањата. Ако постои истата се користи. Ако не, се развива нова или адаптира некоја постоечка.

CBD моделот води кон повторна употреба на софтвер (software reuse) што овозможува некои значајни предности како: 70% намалено време на развој, 84% намалување во цената на проектот и индекс на продуктивност од 26.2.

**Модели на формални методи**

Вклучува збир активности кои водат кон формална математичка спецификација на компјутерскиот софтвер. Ваквата спецификација овозможува да се специфицира, развие и верификува компјутерски систем применувајќи ригорозна математичка нотација.

CleanRoom Software Engineering е процес за развој на софтвер развиен во IBM со цел да се произведе софтвер со предвидливо ниво на доверливост (сигурност)

Се фокусира на спречувањето на дефекти, наместо нивно откривање и исправање

Во пракса поретко се применува (освен за развој на критични системи)

Скапа и долготрајна метода

Потребно специфично предзнаење за нејзина примена

Несоодветна за комуникација на дизајнот со клиентот.

**Аспектно-ориентиран развој на софтвер**   
Нова технологија за развој на софтвер кој бара нова модуларизација на софтверските системи со цел да се изолираат секундарните и функциите за поддршка од бизнис логиката на главната програма. AOSD овозможува различните грижи (concerns) да бидат изразени/разгледувани одвоено/изолирано и автоматски да бидат обединети во единствен работен систем.

**Rapid Application Development (RAD**) е моделот со инкрементални и итеративни компоненти во развојот на софтвер со нагласено исклучително кусо време на развој. Забрзана варијанта на линеарниот секвенцијален модел со масивна употреба на компонентно ориентирана конструкција на софтверот.

**Карактеритики:**

Рано развивање на прототипови (со користење на алатки за брз развој на прототипови) и нивна евалуација од страна на клиентот

Масовна употреба на готови компоненти

Помалку формални ревизии и комуникација помеѓу тимовите

Минимално планирање

JAD (joint application development)

Корисниците се вклучени во сите чекори на развојот

**Применлив кога:**

♦ барањата се добро разбрани

♦ опсегот на проектот е релативно ограничен (претежно за инфрмациони системи)

♦ поделба на функционалностите за паралелно да се развиваат од одвоени тимови

♦ на располагање се доволно човечки ресурси

♦ на располагање се софистицирани алатки за автоматизација на процесите

♦ посветеност од страна на развивачите и клиентите за брз развој (реагирање)

**Недостатоци**

♦ За големи проекти потребни се големи човечки ресурси

♦ Програмери и клиенти посветени на RAD моделот

♦ Не е применлив на поширок круг на апликации

♦ Не применлив кога технолошките ризици се високи (нови технологии, софтвер, интероперабилност)

**Unified Process**

Со употреба на Unified Modeling Language (UML) се дефинираат компонентите од кои ќе се изгради системот како и интерфејсот кој ќе ги поврзува компонентите. Со комбинација на итеративен и инкрементален развој UP ги дефинира функциите на системот со примена на пристап базиран на сценарија (од корисничка гледна точка).

**Rational Unified Process (RUP)** – нуди итеративна флексибилна рамка за развој на софтвер со можност на прилагодување кон потребите на софтверскиот тим и клиентот

RUP претставува имплементација на спирален модел.

RUP фази:

**-Inception –** зачнување

**-Elaboration-** проектот почнува да добива форма

**-Construction-** На основа на архитектонските модели, барањата и дизајните се развиваат (или земаат готови) компонентите за да се добие оперативен систем

**-Transition-** Доцните фази на конструкција и испорака на корисникот

**-Production-** Употреба на софтверот од страна на корисникот

Надгледување,

Поддршка,

реагирање на извештаите на грешки и барања за промени,

веројатно паралелно течат другите фази од RUP за следниот инкремент.

**Personal Process Models- Персонални**

Најдобар софтверски процес е оној кој е најблизок до клиентот.

Ја нагласува потребата од мерења на продуктот и неговиот квалитет, со цел да се преземат мерки за адаптација на процесот.

PSP дефинира 5 рамковни активности (во сите се спроведува мерење)

1.Planning

♦ проценка на големина и потребни ресурси, идентификација на работни задачи

2. High-level design

♦ спецификација на дизајнот на компонентите (прототипови по потреба)

3. High-level design review

♦ формални методи за верификација на дизајнот

4. Development

♦ ревизија на дизајнот на компонентите, генерирање, ревизија, преведување и тестирање на кодот

5. Postmortem

♦ користејќи ги собраните мерки и метрики се одредува ефективноста на процесот (да се разберат типовите на грешки кои се појавуваат и да се преземат активности и адаптација на процесот за нивно намалување)

**Team Software Process (TSP)**

Да се изгради „само-управувачки“ тим кој се организира самиот за да продуцира софтвер со висок квалитет

Цели:

♦ креирање на тимови кои ќе ја планираат и следат својата работа (3-20 инженери)

♦ Обука на менаџерите да ги водат и мотивираат своите тимови

♦ забрзување на подобрувањето на софтверскиот процес (CMM Level 5 е вообичаено и очекувано)

♦ обезбедување насоки за подобрување

♦ Промовирање на изучувањето тимски вештини на индустриско ниво на универзитетите

**Process technology tools**

Алатки кои помагаат во моделирање и спроведувањето на процесот на развој на софтвер.

Планирање, проценка, надгледување и контрола на активностите, акциите и задачите во моделот на процесот.

Листи на работи кои треба да бидат сработени, материјали кои треба да бидат испорачани, проверки кои треба да бидат спроведени.

**Техники на 4та генерација**

Вклучуваат различни софтверски алатки на кои заедничко им е што овозможуваат на софтверскиот инженер да специфицира одредени карактеристики на софтверот на високо ниво. Алатката потоа автоматски генерира изворен код на основа на зададената спецификација.

4GT парадигмата се фокусира на употреба на специјални јазици или графичка нотација за опис на проблемот.

Предности: забрзан развој и зголемена продуктивност

Ограничувања: применлив на релативно мали проекти и проекти од специфичен тесен проблемски домен, неефикасност на произведениот код, отежнато одржување ако се примани кај големи проекти

**5. Агилни методи**

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

**Агилниот развој на софтвер** претставува концептуална рамка од софтверското инженерство која промовира итерации при развојот на софтвер низ целиот животен век на проектот

Секоја итерација (2-4 недели) ги вклучува сите фази од развојот на софтвер: планирање, анализа на барања, дизајн, кодирање, тестирање и документирање

Целта е да се има расположлив софтвер за испорака (кој работи и без грешки) на крајот на секоја итерација

Агилните методи продуцираат многу мал број на документи во споредба со другите поради што се критикувани како недисциплинирани

Му даваат многу помало значење на проектирањето, анализата и документацијата и се концентрираат што побргу да произведат употреблив прототип.

Агилниот пристап може да се примени на било кој софтверски процес сè додека на проектниот тим му е дозволено да ги забрза задачите и го спроведе планирањето на начинот со кој се елиминираат несуштинските работни продукти

Трошоците на промена се зголемуваат брзо како што проект напредува кон завршување, колку што порано е направена промената толку е поефтина

**Клучни претпоставки** за сите агилни процеси

1. Тешко е однапред да се предвиди кои барања или приоритети на клиентите ќе се променат и кои не

2. За многу видови на софтвер, дизајнерските и изведбени активности се испреплетени (изведбата се користи за да се провери дизајнот)

3. Анализата, дизајнот и тестирањето не се толку предвидливи од перспектива на планирање колку што би сакале За справување со непредвидливоста агилниот процес мора постојано и самиот да се адаптира

Инкременталната адаптација бара повратни информации од клиентот врз основа на оценка на испорачаниот софтверски инкремент (извршни прототипови) во куси временски периоди

**Принципи на агилност**

1.Највисок приоритет е да се задоволат клиентите преку рана и континуирана испорака на употреблив софтвер

2. Добредојдени се и промени на барањата подоцна во развојот, на промената се гледа како на зголемување на конкурентната предност на корисникот

3. Честа испорака на употреблив софтвер со преферирање на куси периоди на испорака (пр. секои 2 до 4 недели) 4. Деловните луѓе и програмерите мора да работат заедно секојдневно во текот на проектот

5. Засновање на проектот околу мотивирани индивидуи, со обезбедување на околината и поддршката која им е потребна, и со верба дека ќе се заврши работата

6. Директната лице-в-лице комуникација е најефективен метод за пренесување на информации во рамките на тимот за развој

7.Извршниот софтвер е примараната мерка со која се проценува напредокот

8. Агилните процеси промовираат одржлив развој - програмерите и корисниците би можеле да го продолжат развојот на неодредено време

9. Континуираната грижа за техничка совршеност и добар дизајн ја подобрува агилноста

10. Едноставност (дефинирана како она што не е направено – бидејќи не морало да биде) е од суштинско значење

11. Најдобрите архитектури, барања и дизајн произлегуваат од само-организирачките тимови

12. Во редовни интервали тимовите согледуваат како да станат поефикасни и го прилагодуваат своето однесување во согласност со тоа

**Човечки фактори**

Карактеристики кои треба да ги поседуваат членовите на агилниот развоен тим:

Компетентност (talent, software skills, process)

Заедничка цел (to deliver a working software increment on time)

Соработка (team members with one another and all other stakeholders)

Способност за носење одлуки (the team is given autonomy - decision-making authority for both technical and project issues)

Способност за решавање непрецизно дефинирани проблеми (непрецизност, чести промени)

Заемна доверба и почитување

Само-организација

**Екстремно програмирање**

(Extreme Programming – XP) препорачува практики за менаџерите и програмерите кои стимулираат одредени вредности при развојот на софтвер (software engineering best practices taken to "extreme" levels)

**Цели:** -Обид да се консолидираат хуманоста и продуктивноста

-Механизам за социјални промени

-Препораки за подобрување

-Стил на развој на софтвер

-Дисциплина при развојот на софтвер

**Вредности:**

1. Communication ♦ informal (verbal) collaboration
2. Simplicity ♦ design for immediate needs only
3. Feedback ♦ from the implemented software itself, from the customer, from other software team members
4. Courage (discipline) ♦ design for now – change it later if needed
5. Respect

**Клучни активности:**

-Планирање

-Дизајнирање

-Кодирање

-Тестирање

**Ограничувања на екстремното програмирање:**

Променливост на барањата

Конфликтни потреби на клиентите

Со барањата претставени само како кориснички приказни и тест за прфатловoст е тешко да се избегнат пропустите и недоследностите

Недостаток на формален дизајн

**Критика на екстремното програмирање:**

Неформално барање на промени

Различни корисници може да имаат различни барања

User stories и acceptance tests се единствени формални исказа за барања

Недостиг на формален дизајн.

**Industrial XP**  
2005- Органска еволуција на ХР  
Вклучува повеќе менаџмент. Ја проширува улогата на клиентот

**Нови практики:**

Оценка на подготвеност

Проектна заедница

Проценка на оправданост на проект

Менаџирање водено до тестови

Ретроспективи

Непрекинато учење

**Модификации:**

SDD (Story-Driven Development) се инсистира приказната за тест на прифатливост да биде напишана пред да се пишува кодот

DDD (Domain-Driven Design) Унапредување на системската метафора – креирање на модел на домен

**Cowboy coding**

- програмира кој што сака без план и визија.

**software management Anti-pattern** – на изглед изгледа како да има предности но овие предности се надвладеани од консеквенците и ризикот на ваквиот пристап

**Адаптивен развој на софтвер**

-За комплексни системи

-Почива на соработка и самоорганизација во тимот

-Колаборација

♦ Незлонамерно критикување

♦ Помагање без гнев

♦ Без забошотување

♦ Поседување квалификации кои можат да помогнат

♦ Разгледување на проблемите на начин кои води кон ефективна акција

**DSDM**

**Dynamic Systems Development Model**

♦ Incremental prototyping

♦ Pareto принцип – 80% од апликацијата може да се испорача за 20% од времето потребно за да се изработи комплетно (100%)

**Crystal**

Фамилија од методи (означени со различни бои) за различна големина/комплексност на проекти

Уважува дека различни проекти имаат потреба од различни полиси, конвенции и методологии

**Feature Driven Development** –функција која може да е имплементирана за 2 недели или помалку.

**6.Инженерство на барања**

Requirements engineering (**инженерство на барања**/потреби) обезбедува соодветни механизми кои помагаат да се разбере што клиентот точно сака,

**анализирање** на потребите,

**проценка** на изводливост,

**преговарање** околу разумно решение,

**спецификација** на одредено (разумно) решение,

**валидација** на спецификацијата и

**менаџирање** на барањата при нивната трансформација во „оперативен“ систем.

Може да се подели на 7 чекори:

**1.Зачеток**  
се добиваат првичните сознанија за проблемот, природата на решението, луѓето кои го бараат решението и начинот на комуникација со нив

**2.Согледување**

проблеми при дефинирање на опсегот - неправилни граници во системот, детали кои збунуваат наместо да појаснат

Проблеми на разбирање - клиентите/корисниците ни самите не се сигурни што точно сакаат, ограничено разбирање на можностите и ограничувањата на компјутерскиот систем, ограничено разбирање на доменот на проблемот, проблеми во комуникацијата со системскиот инженер, испуштање на „очевидните“ информации,специфицирање на конфликтни барања или барања кои се непрецизни

Проблеми поради променливост – барањата се менуваат во тек на времето

**3.Елаборација**

Информациите прибрани во текот на зачетокот и согледувањето се прошируваат, прочистуваат.

**4.Преговарање**

Разрешување на конфликти низ процес од преговори во кои некои барања се елиминираат, комбинираат, менуваат  
**5.Спецификација**

Може да биде : ♦пишан документ ♦множество графички модели ♦формален математички модел ♦збирка кориснички сценарија ♦прототип ♦комбинација од сите

**6.Валидација**

Проверка на квалитетот на сработеното. Техничка ревизија од страна на тим од софтверски инженери, клиенти, крајни корисници и други засегнати.

**7.Менаџирање**

Менаџирање на барања (requirements management) претставува збир од активности кои помагаат на проектниот тим да ги идентификува, контролира и следи промените на барањата во текот на изведбата на проектот.

Типот на секое од барањата може да биде:

F = functional requirement,

D = data requirement,

B = behavioral requirement,

I = interface requirement,

P = output requirement.

**Traceability Tables**

По идентификацијата на барањата се прават т.н. traceability tables (табели за следење) во кои јасно се означуваат врските (релациите, зависноста) на секое од барањата со еден или повеќе од аспектите на системот или околината.

**Requirements Analysis and Negotiation**

Разрешувањето на недоследностите во барањата се изведува со преговори

Се рангираат барањата по важност

Се прави груба проценка на потребниот напор за да се задоволи секое од нив

Се проценува цената и времето потребно за реализација на проектот

Врз база на овие проценки некои од иницијалните барања се елиминираат, се комбинираат или се менуваат

**Requirements Specification**

Системската спецификација ги дефинира функциите и перформансите на компјутерскиот систем, како и информациите (податоци и контрола) кои се влез и излез од системот.

**Модел на системот** се гради (на хартија / нацрт) со цел да се евалуираат системските компоненти и нивните меѓурелации, како барањата и ограничувањата се вклопуваат во оваа слика и да се процени „естетиката“ на системот што и да се подразбира под тоа.

**Requirements Validation**  
Се испитува спецификацијата дали ги задоволува критериумите и стандардите поставени за проектот Примарниот механизам е технички преглед

♦ воочување на можни грешки, недоследности, контрадикторности, непрецизности

♦ истакнување на делови каде е потребно дообјаснување

♦ воочување на можни конфликтни или нереални барања

**Инженерство на софтверски барања**

Претставува систематска употреба на докажани принципи, техники, јазици, и алатки за ефективна анализа, документирање и еволуција на корисничките барања и спецификацијата на софтверскиот систем кој треба да ги задоволи тие барања. Во него активно учествуваат и клиентот и софтверскиот инженер.

**Принципи при спецификација**

Одвоеност на функционалноста од имплементацијата

Да се развие модел за посакуваното однесување на системот како одѕиви на стимули од околината Воспоставување на контекстот во кој ќе работи софтверот преку специфицирање на начинот на кој системските компоненти интер-реагираат со софтверот

Креирање на когнитивен модел наместо дизајн или имплементација (како системот го спознаваат корисниците) Спецификацијата може да биде и некомплетна и непрецизна

Воспоставување на структура на содржината на спецификацијата која овозможува лесни промени

**Репрезентација**

Форматот и содржината треба да бидат релевантни за проблемот

Информациите презентирани во спецификацијата треба да бидат вгнездени

Дијаграмите и останатите начини на означување треба да се контролирани и конзистентни (стандардизирани) Репрезентацијата треба да биде погодна за промени

**Репрезентација на софтверски барања**

Спецификација – кулминација на анализата

**IEEE standard No. 830**-1984 – стандарден формат за репрезентација на софтверските барања

♦ Introduction (цели, контекст)

♦ Information description (детален опис на проблемот, содржина, тек и структура на информациите, хардверски, софтверски и кориснички интерфејси)

♦ Functional Description (за секоја функција која е потребна за решавање на проблемот, наративен опис, ограничувања, перформанси, дијаграми, врска со останатите елементи)

♦ Behavioral Description (функционирање на софтверот како последица на надворешни настани и интерна состојба)

♦ Validation Criteria (како да се провери успешно реализиран софтвер, дефинирање на тестови

♦ Bibliography and Appendix (друга документација, референци, стандарди, ...)

**Ревизија на спецификација на софтверски барања**

Се изведува и од клиентот и од развивачот

Обид да се провери дали спецификацијата е комплетна, конзистентна и прецизна

Појаснување на изразите како: нешто, малку, понекогаш, вообичаено, претежно, ...

По изведената ревизија, спецификацијата се потпишува и станува дел од договорот помеѓу клиентот и развивачот

**FAST**

**Facilitated Application Specification Techniques (FAST)** – формирање заеднички тим од клиенти и развивачи на софтвер кои работат заедно за да го идентификуваат проблемот, да понудат елементи на решението, да преговараат околу различните пристапи и да специфицираат прелиминарно множество на барања.

**Состанок**

Оправданост на проектот

Презентирање на индивидуалните листи (за секоја област)

Листите се комбинираат

Дискусија – консензус листа

Поделба на подтимови кои ќе развијат мини- спецификации

♦ презентација на мини-спецификациите кои можат да откријат нови објекти, сервиси, ... и дискусија

♦ листа на отворени прашања

Критериуми за валидација

Делегирање на пишувањето на комплетна драфт спецификација на база на сите материјали од FAST состанокот

**Quality function deployment (QFD**) е техника за менаџирање на квалитет која ги преведува потребите на корисниците во технички барања на софтверот. Се концентрира на согледување што е значајно за корисникот и испорака на овие барања низ инженерскиот процес.

QFD идентификува три типа на барања:

♦ Normal requirements (Нормални барања – како што се поставени од клиентот)

♦ Expected requirements (Подразбирани барања – ненагласени фундаментални барања)

♦ Exciting requirements (Можности кои не се побарани, но од кои клиентот би бил особено задоволен)