Софтверско инженерство, теорија

Карактеристики на софтвер

Што спаѓа во софвтер?

- Инструкции кои кога се извршуваат ги обезбедуваат посакуваните особини, функции и перформанси
- Податочни структури кои овозможуваат програмите адекватно да ги манипулираат информациите
- Описни информации, како копија на хартија или виртуелна форма кои го опишуваат работењето и употребата на програмите

Карактеристики на софтверот:

- Софтверот не се произведува, тој се развива
  - Квалитетот се постига преку добар дизајн (но нема производство на ништо материјално)
  - Активностите зависат од луѓе но релациите помеѓу луѓето се различни
  - Трошоците се концентрирани во инженерството
- Софтверот не се троши (абе), но се расипува (се влошува)
- Најголем дел од софтверот се развива по нарачка
- Софтверот може да биде екстремно комплексен па поради тоа и тежок за разбирање и следење што остава многу голем простор за човечки грешки
- Расвојот на комплексен софтвер кој ќе биде без грешки 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
- Real-time software
  - Надгледува процеси и појави од реалниот свет – треба да бидат обработени податоците во реално време
- Апликативен софтвер – индивидуални програми кои решаваат одредени бизнис потреби
- 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 payers, routers, микробранови печки, автомобили
- Софтвер за PC – масовен софтвер (Product-line)
    - Дизајниран да обезбеди одредени специфични функционалности за различни корисници. Фокусиран на
        - Специфичен пазар
        - Масовен пазар
- Web и мобилни апликации
    - Веб апликации на кои им се пристапува преку прегледувач
    - Апликации кои се поставени на мобилни уреди
- Софтвер за вештачка интелигенција
    - Роботика, експертни системи, препознавање на облици
- Legacy software
    - Стар “наследен” софтвер (10+ години)
    - Понекогаш од одредени причини засегнатите сакаат да го одржат во функција
    - Долготраен, менуван и дополнуван

Трендови

- Web апликации и сервиси
- Мобилни апликации
- Cloud computing
- Product line software

Web апликации

- Мрежно интензивни
- Конкурентност
- Непредвидливо оптоварување
- Перформанси
- Достапност
- Податочно водени
- Зависни од содржината
- Постојана еволуција
- Итност
- Сигурност
- Естетика

Структура на продажната цена на софтверот

- Up-front fee – се наплаќа на почетокот како лиценца (право) на користење на софтверот
- Maintenance – се наплаќа годишно за одржување -patches, upgrades
- Инсталација, интеграција, обука

Модели на лиценцирање

- Free OpenSource софтвер upfront = 0
- Subscription license – се купува правото на користење на одредено време
- Usage license (oracle)
- Software as a service
- Razor and blade model – основната апликација се нуди бесплатно а се наплаќа за останатите делови од продуктот (Adobe)
- Service model – основната апликација се дава бесплатно а се заработува на услуги како инсталација, конфигурација, одржување, обука...
- Advertising model – корисниците не плаќаат за употребата на софтверот но за сметка на тоа мораат да гледаат реклами додека го користат

Видови на софтвер

- Proprietary software
  - Non-free software or closed-source software
- Free software (open-source software)
  - Source code must be made available
  - Free software licenses

Видови на софтвер (достапност)

- Freeware (бесплатен)
- Shareware (бесплатен во 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
- Abandonware
- Adware

Процес на развој на софтвер

Софтверско инженерство

- Професија чии припадници креираат (развиваат) и одржуваат софтверски апликации применувајќи техники од областите на:
  - Компјутерските науки
  - Проектниот менаџмент
  - Инженерството
  - Доменот на апликацијата
  - И други области
- Инженерство претставува примена на научни и математички принципи во развој на економични решенија на технички проблеми, креирање на продукти, средства и структури кои се корисни за луѓето. Инженерите употребуваат разум, имагинација и расудување при примена на науката, технологијата, математиката и практичните искуства. Резултатот е дизајн, производство или функционирање на корисни објекти или процеси. Професионалните практичари на инженерството се наречени инженери.
- Најопшта дефиниција за инженерство = примена на науката во решавање на социјалните проблеми

Што е софвтер?

- Софтверот е програма (нематеријален ентитет) која овозможува на машината (физичката компонента – најчесто компјутерот) да извршува точно одредена задача
- Како нематеријална содржина на меморијата софтверот во принцип може да се менува без промени на статичката парадигма на хардверот
- Најчесто софтверот е во вид на алгоритам кој се преведува во секвенца на машински инструкции

Софтверско инженерство

- Се занимава со теориите, методите и алатките за професионален развој на софтвер
- Одржувањето на софтверот често чини повеќе од неговиот развој
- Софтверското инженерство се занимава со економичен развој на софтвер
- Инженерска дисциплина која се занимава со сите аспекти на развојот на софтвер
- **Креирање на издржлив дизајн за остварливо софтверско решение кое треба да решава реален проблем**

- **Дизајн, имплементација и одржување** на софтверски решенија
- **Системски, дисциплиниран и квантифициран пристап кон градењето и одржувањето на софтвер**

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.

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 – проверска дека софтверот е навистина она што купувачот барал
  - Еволуција – промена на софтверот како резултат на променетите барања

Техники и концепти во софверското инженерство

- Анализа и спецификација на барања
- Дизајн и архитектура
- Кодирање и интеграција
- Тестирање и отстранување на грешки
- Документирање
- Инсталација и одржување

Модел на софтверски процес – поедноставена престава на софтверскиот процес, дадена од одредена перспектива на процесот

Software development lifecycle – процесот на развој на софтвер се нарекува и животен циклус на развојот на софтверкиот проект

Софтверското инженерство како слоевита технологија

- Подлогата претставува фокусирање на **квалитетот**

- Темелот на софтверското инженерство е **процесот**
  - ГИ поврзува технолошките слоеви и обезбедува рационален и навремен развој на софтвер
  - Дефинира рамка за клучните области на процесот (key process areas – KPAs) кои мора да бидат воспоставени за ефикасно применување на технологиите на софтверското инженерство
  - KPA претставуваат: основа за менаџерска контрола на софтверскиот проект, го дефинираат контекстот во кој се применуваат техничките методи, се креираат работни материјали, се поставуваат граничници, се контролира квалитетот и се управува со измените
- **Методите** на софтверското инженерство обезбедуваат техничко водство во развојот на софвер (комуникација, анализа на барања, дизајнирање, програмирање, тестирање, поддршка)
- **Алатките** на софтверското инженерство обезбедуваат автоматска или полуавтоматска поддршка на процесот и методите

Софтверски процес

- Процес – збирка од активности, акции и задачи кои се изведуваат кога треба да се изведе нешто
- Активноста тежнее да постигне погенерална цел (комуникација со засегнати странки) независно од апликативниот домен, видот и големината на проектот
- Акција (на пр. Архитектонско дизајнирање) опфаќа множество задачи кои продуцираат главен работен артефакт (модел на архитектурата)
- Задача (задолжение) се фокусира на потесна но добро дефинирана цел (Unit test) и продуцира видлив резултат
- Кај СИ процесот е многу флексибилен и адаптибилен

Процесна рамка

- Поставува основа на комплетниот процес на развој на софтвер идентификувајќи мал број рамковни активности кои се применливи на сите видови софтверски процеси
- Дополнително вклучува и одредени општи т.н. umbrella activities кои се применуваат низ целиот софтверски процес

Генеричка процесна рамка за развој на софтвер

- Комуникација (Communication)
  - Клиенти, барања
- Планирање (Planning)
  - Мапа како да се стигне до целта – software project plan

- Моделирање (Modelling)
    - Помага подобро да се разбере проблемот при неговото решавање
- Конструкција (Construction)
    - Генерирање на код
- Испорака (Deployment)
    - Испорака на клиент (evaluation and feedback)

Umbrella activities – независни од останатите рамковни активности и се одвиваат континуирано за целото времетраење 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)

- Поделба на добро дефинирани задачи наречени фази
- Секоја фаза завршува со граничник (milestone)
- За секој граничник постојат јасно дефинирани цели кои може лесно да се проверат

Пракса

- Разбери го проблемот
    - Комуникација и анализа
- Планирај го решавањето
    - Моделирање и дизајн
- Спроведи го планот
    - Кодирање
- Провери ги резултатите
    - Тестирање

Генерички пристап кон развој на фазите во софтверското инженерство

- Дефиниција – се фокусира на ШТО?
  - Собирање на информации
  - Планирање на проект
  - Анализа на барања
- Развој – се фокусира на КАКО?
  - Дизајнирање на софтверот
  - Кодирање
  - Тестирање
- Поддршка – се фокусира на ПРОМЕНИ
  - Корекција (на грешки)
  - Адаптација (на нов систем, околина)
  - Подобрување (дополнителна функционалност надвор од иницијалните спецификации)
  - Превенирање (промени кои треба да овозможат полесни идни корекции, адаптации и подобрувања)
  - Дополнителни активности – техничка помош, телефонски help-desk, web site за поддршка

Фази во развојот на софтвер

- Размена на информации помеѓу корисникот и развивачот на софтверот
- Комплетен опис на проблемот
- Разложување на конкретни детални програмски задачи
- Програмирање
- Тестирање
- Измени на програмата

Размена на информации

- Дискусија со крајниот корисник
- Истакнување на своите барања од страна на корисникот
- Фазата завршува со листа на барања (requirements document)

Комплетен опис на проблемот

- Листата на барања се преформулира во форма посоодветна и разбирлива за развивачите
- Соодветниот документ се нарекува спецификација

- Спецификациите дефинираат што точно софтверот треба да работи од перспектива на развивачот, а не како ќе биде постигната бараната функционалност

Разложување на системот на компоненти

- Брз основа на спецификацијата на системот се разложува на модули и секој од овие модули одделно се опишува
- Секој модул комуницира со останатите модули преку соодветен интерфејс
- Целта е да се дадат добро дефинирани програмерски задачи на индивидуалните програмери
- Граничникот на оваа фаза се нарекува документ за дизајн на системот

Програмирање

- Во оваа фаза документот за дизајн се трансформира во извршен код (програма)
- Граничникот на оваа фаза е извршна програма која може да се извршува на соодвениот систем

Тестирање

- Тестирањето е процес на проверка дали софтверот го прави она што треба да го прави
- Дали софтверот одговара во потполност на барањата
- Граничникот од оваа фаза е тестиран софтвер

Промени

- Поправање на грешки и недоследности за кои се потребни измени во кодот а некогаш и во дизајнот на софтверот

Процесни модели

- Пропишани процесни модели
- Агилни процесни модели

Модели на процеси за развој на софтвер

Генеричка проценс рамка

- Комуницирање
- Планирање
- Моделирање
- Конструкција
- Употреба

Паралелни активности:

- следење и контрола
- Менаџирање и ризици
- Проверка на квалитет
- Формални технички прегледи
- Measurement
- Менаџирање на ефектите на промена
- Менаџирање за повторна употреба
- Подготовка и продукција на документи

Модели на процеси за развој на софтвер

- За да ги реши актуелните проблеми, софтверскиот инженер (или тим) мора да примени стратегија за развој која ги опфаќа нивоата на процесот, методите и алатките како и генеричките фази во развојот на софтвер. Оваа стратегија е наречена **модел на процес** или **парадигма на софтверското инженерство**
- Изборот на моделот зависи од природата на проектот или апликацијата, методите и алатките кои ќе се користат како и резултатот кој се очекува

Тек на процес

- Опишува како се поставени рамковните активности, акции и задачи и ги моделира 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 процесен 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 со другите активности

Групи задачи

- Секоја акција поврзана со некоја од рамковните активности на софтверското инженерство може да биде претставена со одредена група на задачи
- Малите проекти не бараат групи на задачи кои се сложени и детални како задачите кај комплексните тим-ориентирани проекти
- Групите на задачи се прилагодуваат кон специфичните потреби на конкретниот софтверски проект и карактеристиките на проектниот тим

Capability Maturity Model

- Level 1: Initial – ad hoc & chaotic
- Level 2: Repeatable – основно менаџирање на проект за следење на трошоци, рокови и функционалност. Предвидлив успех за проекти слични на веќе изработените + сè од Level 1
- Level 3: Defined – Софтверскиот процес и за менаџментот и за инженерските активности е документиран, стандардизиран и интегриран на ниво на целата организација + сè од Level 2
- Level 4: Managed – Се вршат детални мерења на софтверскиот процес и квалитетот на софтверот + сè од Level 3
- Level 5: Optimizing – Континуирано подобрување на процесот е овозможено со помош на квантитативни повратни информации од процесот и тестирање на иновативни идеи и технологии + сè од Level 4

Класични (пропишани) методи за развој на софтвер

Work flow – пропишан тек на процесот, односно начин на кој процесните елементи се во релација едни со други

Поделба на процесните модели

- Пропишани
  - Waterfall (водопад)
  - Инкрементални
  - Еволуциски
  - Конкурентни
- Специјални
  - Компонентно ориентирани
  - Формални методи
  - Аспектно ориентирани
- Unified process
- Агилни
  - XP (екстремно програмирање)
  - Industrial XP
  - Adaptive Software Development (ASD)
  - Scrum
  - Dynamic Systems Development Method (DSDM)
  - Crystal
  - Feature Drive Development (FDD)
  - Lean Software Development (LSD)
  - Agile Modeling (AM)
  - Agile Unified Process (AUP)

Модели на софтверски процеси

- Модел на водопад (класичен животен циклус – старомоден, но разумен пристап кога барањата се добро разбрани)
- Инкрементални модели (испорачува софтвер во мали употребливи парчиња при што секое парче се заснова на веќе претходно направените парчиња)
- Еволутивни модели
  - Модел на прототипови (разумен иницијален чекор, кога клиентот има легитимна потреба, но деталите не се добро познати/дефинирани, развивачот треба да се спротивстави на притисокот на инвеститорот грубиот прототип да се прошири во продукциски систем)

    - Спирален модел (ги комбинира итеративниот пристап на прототипови со контролираниот систематски аспект на водопад моделот)
- Паралелен развоен модел (симултан инженеринг – овозможува софтверските тимови да ги претставуваат итеративните и паралелни елементи од кој било процесен модел)
- Специјализирани процесни модели
    - Компонентен развоен модел (варијација на спиралниот развоен модел во кој апликации се градат од готови софтверски компоненти наречени класи)
    - Формални методи (ригорозна математичка нотација која се користи за да се специфицира, дизајнира и потврди/верификува компјутерски-базиран систем)
    - Aspect-Oriented развој на софтвер (аспект-ориентирано програмирање – обезбедува процесот на дефинриање, специфицирање, дизајнирање и градење на софтверски аспекти како кориснички интерфејси, сигурност и управување со меморијата кои влијаат на многу делови на системот кој се развива)

Линеарен секвенцијален модел

- Линеарниот секвенцијален модел наречен уште и класичен или модел на водопад 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
    - Дизајн
    - Генерирање на код
    - Тестирање
    - Поддршка
- Фази во линеарниот секвенцијален модел
    - Софтверскиот аналитичар мора да го разбере информацискиот домен ан софтверот, како потребните функции, однесување, перформанси и интерфејси. Барањата се документираат и се прегледуваат со клиентот.
    - Дизајнот е повеќечекорен процес кој се фокусира на 4 атрибути на програмата:
        - Податочни структури
        - Софтверски архитектури
        - Претстава на интерфејси
        - Процедурални (алгоритамски) детали
    - Дизајнот ги преведува барањата во репрезентација на софтверот преку која ќе може да се провери неговиот квалитет уште пред да почне кодирањето

    - Генерирање на код – преведување на дизајнот во машински читлива (извршна) форма
    - Тестирање – се концентрира на проверка на интерната логика на софтверот (да се проверат сите функции) и екстерната логика (дали за секој дефиниран влез се добива бараниот излез
    - Поддршка – промени: поправка на грешки, прилагодување на нова околина или функционални и перформансни промени барани од клиентот
- Предности на waterfall моделот
    - Времето потрошено во раните фази повеќекратно се отплатува во подоцнежните фази. Доколку има грешка во дизајнот полесно е да се промени дизајнот отколку да се потрши многу време на имплементација на лош дизајн
    - Секоја од фазите да се заврши 100% за да се осигури нејзина исправност пред да се проследи на следната фаза
    - Генерирање и одржување прецизна и опсежна документација
    - Големите компании и владини контрактори работат според овој модел
- Недостатоци на waterfall моделот
    - Реалните проекти ретко кога го следат строгиот секвенцијален модел
    - Често е тешко клиентот да ги зададе сите свои барања на почетокот на проектот
    - Клиентот мора да биде трпелив бидејќи работна верзија на програмата ќе добие многу доцна во текот на проектот
    - Blocking states – некои членови од тимот чекаат другите да си ја завршат работата за тие да продолжат (ниска продуктивност)

Инкрементален модел

- Барањата се релативно добро познати, но постои потреба од ограничено множество на функционалности да биде обезбедено што побргу, а потоа да се прочистат и прошират функционалностите
- 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 прототипот, инкременталниот модел испорачува употреблив продукт (она што работи работи, но не е имплементирано сè што е планирано)

Еволуциски процесни модели

- Еволуциските модели го гледаат развојот на софтверот како систем за кој однапред се знае дека ќе мора да се менува
- Модел на прототипови

    - Requirements gathering
    - Quick design
    - Prototype construction
    - Prototype evaluation
    - Се концентрираат на деловите видливи за клиентот. Целта не е долготрајна одржливост и добар дизајн. Треба да се фрли!
- Предности на развој со прототипови
    - Помага подобро да се разбере што треба да се изгради
    - Кога се применува?
        - Нејасно специфицирани барања
        - Клиентот има легитимен проблем, но не знае да го искаже (дефинира)
    - Брз (неоптимален) дизајн кој се фокусира на аспектите видливи за корисникот
- Проблеми при развој на прототипови
    - Притисок од страна на засегнатите страни да еволуираат кон целосен систем
    - Несоодветни компромиси од страна на програмерот

Спирален модел

- Еволуциски модел кој ги комбинира итеративната природа на прототип пристапот со контролираниот и систематски 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 конечен продукт
- Воведува попрецизна контрола и менаџирање на ризици
- Дефинира (3-6) рамковни активности (task regions):
    - Costumer communication
    - Planning
    - Risk analysis
    - Engineering
    - Construction and release
    - Costumer evaluation
- WINWIN спирален модел
    - Многу сличен на стандардниот спирален модел во кој комуникацијата со клиентот е претставена како преговарање и наоѓање на компромиси помеѓу цената, барањата и роковите

Паралелен модел

- Паралелниот развоен модел (Concurrent Development Model) може да се претстави шематски со серија од технички активности, задачи и ним

асоцирани состојби. Секоја од активностите може да егзистира во различна состојба
- Дефинирани се серија од настани кои можат да го иницираат преминот од една во друга состојба
- Сите активности може да бидат истовремено активни но секоја во различна состојба
- Паралелниот развоен модел вообичаено се користи при развој на клиент-сервер апликации (кои обично може да се претстават како збир на фунцкионални компоненти) каде конкурентниот процес дефинира активности во две димензии:
    - Системска димензија со активности design, assembly, use
    - Компонентни димензија со активности design, realization
    - Системските и компонентните активности се појавуваат симултано и можат да се моделираат со моделот ориентиран на состојби
    - Типична клиент-сервер апликација се реализира со употреба на многу компоненти од кои повеќето можат да се дизајнираат и реализираат паралелно

Инкрементален и итеративен развој

- Waterfall model – big bang integration
- Incremental development – scheduling and staging strategy
    - 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
- Iterative development – rework scheduling strategy
    - Се планира време за ревизија и унапредување на делови од системот
    - Продуктот се испорачува на (дел од) клиентите и се ревидира по повратните информации од нив
- Обединети 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 развој на софтвер. CBD ги гради апликациите користејќи готови претходно креирани (и тестирани) софтверски компоненти (класи)
- Чекори на компонентно базиран развој
    - Пребарување низ расположливите компоненти за компоненти погодни за апликацискиот домен и нивна проценка

- Проценка на интеграбилноста на компонентите
- Креирање на софтверска архитектура во која се интегрираат компонентите
- Интеграција на компонентите во архитектурата
- Спровердување на темелно тестирање за да се осигури соодветно функционирање

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.
- Нејзина варијација е т.н. clean room software engineering
- Формалните методи нудат механизам за согледување и остранување на бројни проблеми кои тешко се откриваат ако се користи некоја од другите софтверски парадигми

CleanRoom Software Engineering

- Процес за развј 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 во развој на критични системи)
- Скапа и долготрајна метода
- Потребно специфично предзнаење за нејзина примена
- Несоодветна за комуникација на дизајнот со клиентот

Аспектно-ориентирано програмирање

- Aspect-oriented programming (AOP) е програмска парадигма која ги изолира споредните и услужни фунцкии од бизнис логиката на главната програма
- Има за цел да се зголеми модуларност со тоа што се овозможува поделба на грижите, формирајќи ја основата за аспектно-ориентиран развој на софтвер.
- Нова технологија за развој на софтвер која бара нова модуларизација на софтверските системи со цел да се изолираат секундарните и функциите за поддршка од бизнис логиката на главната програма

RAD – Rapid Application Development

- Модел со инкрементални и итеративни компоненти во развојот на софтвер со нагласено исклучително кусо време на развој. Забрзана варијанта на линеарниот секвенцијален модел со масивна употребна на конпонентно ориентирана конструкција на софтверот
- RAD моделот ги има следниве фази
    - Business modeling
    - Data modeling
    - Process modeling
    - Application generation
    - Testing and turnover
- Карактеристики
    - Рано развивање 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
    - 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 развивачките 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
    - Неприменлив кога технолошките ризици се високи

Unified Software Development 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. Потоа се развиваат функции кои соодветствуваат на архитектонската рамка на системот која ја опишува формата која ќе ја поприми софтверот

RUP – Rational Unified Process

- Нуди итеративна флексибилна рамка за развој на софтвер со можност на прилагодување кон потребите на софтверскиот тим и клиентот
- 6 принципи на RUP
  - Adapt the process
  - Balance stakeholder priorities
  - Collaborate across teams
  - Demonstrate value iteratively
  - Elevate the level of abstraction
  - Focus continuously on quality
- RUP фази
  - Inception
  - Elaboration
  - Construction
  - Transition
  - Production
- Inception phase
  - Зачнување – почетна (иницијална) фаза
    - Business case modeling
    - Rough architecture
    - Plan
  - Elaboration phase
    - Проектот почнува да добива форма
    - Ја проширува архитектурата од различни погледи:
      - Use case model
      - Requirements model
      - Design model
      - Implementation model
      - Deployment model
    - Ревизија на план, ре-евалуација на ризици
  - Construction phase
    - На основа на архитектонските модели, барањата и дизајните се развиваат компонентите за да се добие оперативен систем
    - Дизајнирање и спроведување на unit тестови
    - Интеграција на компонентите и интеграциско тестирање
  - Transition phase
    - Трансфер на софтверот кон корисникот за тестирање
    - Подготовка на документација за поддршка
    - Инкрементот станува употреблив софтвер
  - Production phase
    - Употребна 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

- Ја нагласува потребата од мерења на продуктот и неговиот квалитет, со цел да се преземат мерки за адаптација на процесот
- Дисциплиниран, metrics-based пристап
- 5 рамковни активности (во сите се спроведува мерење)
    - Planning
        - Проценка на големина и потребни ресурси, идентификација на работни задачи
    - High-level design
        - Спецификација на дизајнот на компонентите (прототипови по потреба)
    - High-level design review
        - Формални методи за верификација на дизајнот
    - Development
        - Ревизија на дизајнот на компонентите, генерирање, ревизија, преведување и тестирање на кодот
    - 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 самоуправувачки тим кој се организира самиот за да продуцира софтвер со висок квалитет
- Цели:
    - Креирање на тимови кои ќе ја планираат и следат својата работа
    - Обука на менаџерите да ги водат и мотивираат своите тимови
    - Забрзување на подобрувањето на софтверскиот процес
    - Обезбедување насоки за подобрување
    - Промовирање на изучувањето тимски вештини на индустриско ниво на универзитетите

Агилни методи за развој на софтвер

- **Individuals and interactions over processes and tools**
- **Working software over comprehensive documentation**
- **Costumer collaboration over contract negotiation**
- **Responding to change over following a plan**

Агилност

- Промени
- Флуидност
- Time-to-market
- Луѓе (со различни персонални и професионални особености)

Agile software development

- Претставува концептуална рамка од софтверското инженерство која промовира итерации при развојот на софтвер низ целиот животен век на проектот
- Минимизирање на ризикот со развој на софтверот за многу кусо време
- 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 на крајот на секоја итерација
- На крајот на секоја итерација тимот ги ре-евалуира проектните приоритети

Принципи на агилност

- Највисок приоритет е да се задоволат клиентите преку рана и континуирана испорака на употреблив софтвер
- Добредојдени се и промени на барањата подоцна во развојот, на промената се гледа како на зголемување на конкурентната предност на корисникот
- Честа испорака на употреблив софтвер со преферирање на куси периоди на испорака (пр. Секои 2 до 4 недели)
- Деловните луѓе и програмерите мора да работат заедно секојдневно во текот на проектот
- Засновање на проектот околу мотивирани индивидуи, со обезбедување на околината и поддршката која им е потребна, и со верба дека ќе се заврши работата

- Директната лице-в-лице комуникација е најефективен метод за пренесување на информации во рамките на тимот за развој
- Извршниот софтвер е примарната мерка со која се проценува напредокот
- Агилните процеси промовираат одржлив развој – програмерите и корисниците би можеле да го продолжат развојот на неодредено време
- Континуираната грижа за техничка совршеност и добар дизајн ја подобрува агилноста
- Едноставност (дефинирана како она што не е направено – бидејќи не морало да биде) е од суштинско значење
- Најдобрите архитектури, барања и дизајн произлегуваат од само-организирачките тимови
- Во редовни интервали тимовите согледуваат како да станат поефикасни и го прилагодуваат своето однесување во согласност со тоа

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)
- Способноста за решавање непрецизно дефинирани проблеми (непрецизност, чести промени)
- Заемна доверба и почитување
- Само-организација
    - Тимот се самоорганизира за работата која треба да ја заврши
    - Тимот го организира процесот за најдобро да се прилагоди на условите на околината
    - Распоредот на работа се организира за да се испочитува рокот за испорака

Екстремно програмирање – XP

- Препорачува практики за менаџерите и програмерите кои стимулираат одредени вредности при развој 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
- XP вредности:

    - Communication
        - Informal collaboration
    - Simplicity
        - Design for immediate needs only
    - Feedback
        - From the implemented software itself, from the costumer, from other software team members
    - Courage (discipline)
        - Design for now- change it later if needed
    - Respect
- Клучни активности (XP Process)
    - Planning
        - User stories, value, cost, project velocity
    - Design
        - CRC cards and spike solutions, continuous design – refactoring
    - Coding
        - Develop unit test before the code, pair programming, continuous integration
    - Testing
        - Automated testing, XP acceptance tests – derived from user stories
- XP Пракси
    - Fine scale feedback
        - Pair programming
        - Planning Game
        - Test Driven Development
        - Whole Team
    - Continuous process
        - Continuous integration
        - Design improvement
        - Small releases
    - Shared understanding
        - Coding standard
        - Collective Code ownership
        - Simple design
        - System Metaphor
    - Programmer welfare
        - Sustainable Pace
- Ограничувања на XP
    - Променливост на барањата – лесно може да доведе за изместување на фокусот кој може да предизвика потреба од промени во веќе завршената работа според тогаш актуелните барања
    - Конфликтни потреби на клиентите – за многу проекти со повеќе засегнати страни може да биде тешко да се одговори на потребите на сите

    - Со барањата претставени само како кориснички приказни и тест за прифатливост е тешко да се избегнат пропустите и недоследностите
    - Недостаток на формален дизајн (кај сложените системи може да е неопходен формален архитектонски дизајн за да се обезбеди квалитетен и одржлив продукт)

Industrial XP

- 2005 – органска еволуција на XP
- Засновано на минимализмост, ориентираноста кон корисникот и воден од тестови развој за XP
- Вклучува повеќе менаџмент
- Ја проширува улогата на клиентот
- Ги унапредува техничките практики. Некои нови практики:
    - Оценка на подготвеност (readiness assessment)
    - Проектна заедница (project community)
        - Тим заедница: правници, контролори... Пропишан начин на комуникација со нив
    - Проценка на оправданост на проект (project chartering)
    - Менаџирање водено до тестови (test-driven management)
        - Поставување на измерливи цели за проценка на напредокот на проектот
    - Ретроспективи (retrospectives)
        - Специјализирани ревизии по секој инкремент – (issues, events and lessons learned)
    - Непрекинато учење (continuous learning)
- Модификации
    - SDD (Story-Driven Development)
        - Се инстистира 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
    - Работењето во парови е проширено на менаџерите и другите засегнати
    - Iterative usability (usability design that evolves)
    - Модификации на практиките и редефинирање на ролји и одговорности за да се прилагодат кон посериозни и поголеми проекти за големи организации)
- Критики на XP
    - Requirement volatility
        - Неформално барање на промени

    - Conflicting customer needs
        - Различни корисници може да имаат различни барања
    - Requirements are expressed informally
        - User stories и acceptance tests се единствените формални искази за барања
    - Lack of formal design

Cowboy coding

- Не е агилна метода. Се програмира кој што сака без план и визија
- Software management Anti-pattern

Adaptive Software Development

- 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 кој води кон ефективна акција
- Чекори:
    - Speculation - Adaptive cycle planning, mission statement, project constraints, basic requirements, time-boxes release plan
    - Collaboration - Requirement gathering, JAD, mini-specs
    - Learning – components implemented/tested, focus groups for feedback, formal reviews. Postmortems
    - Release – software increment, adjustments for subsequent cycles

Scrum

- Чекори:
    - Requirements
    - Analysis
    - Design
    - Evolution
    - Delivery
- Process pattern – sprint
- Scrum 15-minute daily meeting
    - What did you do since last Scrum meeting?
    - Do you have any obstacles?
    - What will you do before next meeting?
- Backlog – prioritized product features desired by the customer

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
- Чекори:
    - Feasibility study
    - Business study
    - Functional model iteration
    - Design and build iteration
    - Implementation

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
- Crystal Clear
- Crystal Yellow
- Crystal Orange
- Crystal Orange Web
- Crystal Red
- Crystal Maroon
- Crystal Diamond
- Crystal Sapphire

Feature Driven Development

- A feature is a client-valued function that can be implemented in two weeks or less
- Чекори
    - Develop an overall model – more shape than content
    - Build a features list – a list of features grouped into sets and subject areas
    - Plan by feature – a development plan. Class owners. Feature Set Owners
    - Design by Feature – a design package (sequences)
    - Completed client-value function

Инженерство на барања (Requirements Engineering)

Инженерство на барања

- Збир на активности и техники кои треба да водат кон разбирање на барањата
- Главната активност која почнува за време на комуникацијата со клиентот и продолжува за време на моделирањето и треба да води кон разбирање на барањата
- Адаптација според процесот, проектот, продуктот, луѓето
- Може да бидат изразени како;
    - Сценарија за употреба
    - Листи на функционалности и карактеристики
    - Модели или спецификација
- 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 спецификација и менаџирање на барањата при нивната трансформација во “оперативен” систем
- Излезот од процесот на инженерство на барања треба да биде спецификација на компјутерски систем кој соодветно ќе ги задоволи потребите на клиентот

Requirements engineering процесот може да се подели на 7 чекори:

- Зачеток (inception)
- Согледување (requirements elicitation)
- Елаборација (elaboration)
- Преговарање (negotiation)
    - Анализа (requirements analysis and negotiation)
- Спецификација (requirements specification)
    - Моделирање (system modeling)
- Валидација (requirements validation)
- Менаџирање (management)

Inception

- Идентификација на нова бизнис потреба, нов потенцијален пазар или нов сервис
- Дефинирање на бизнис модел за идеја, првичниа студија за изводливост и негов првичен опис
- Првични сознанија за проблемот, природата на решението, луѓето кои го бараат решението и начинот на комуникација со нив

Requirements Elicitation

- Да се прашаат (клиентот, крајните корисници, останатите засегнати):
  - Што треба да се постигне?
  - Како системот треба да се вклопи во деловните потреби?
  - Како ќе се користи при секојдневната работа?
- Зошто согледувањето на барањата е тешко?
  - Problems of scope (проблеми при дефинирање на опсегот) – неправилни граници во системот, детали кои збунуваат наместо да појаснат)
  - Problems of understanding (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
  - Problem of volatility (проблеми поради променливост) – барањата се менуваат со тек на времето

Elaboration

- Елаборирање на информациите прибрани во тек на зачетокот и согледувањето и нивно:
  - Проширување
  - Прочистување
- Дефинирање на различните аспекти на функциите, однесувањето и податоците
- Креирање и анализа на кориснички сценарија за интеракција на крајните кросници со системот

Negotiation

- Се случува клиентите да бараат повеќе од она што може да биде постигнато или дури да постават различни (контрадикторни) барања
- Разрешување на овие конфликти низ процес од преговори во кои некои барања се елиминираат, комбинираат, менуваат

Specification

- Спецификацијата може да биде:
  - Пишан документ
  - Множество графички модели

    - Формален математички модел
    - Збирка кориснички сценарија
    - Прототип
    - Комбинација од горенаведените

Validation

- Проверка на квалитетот на сработеното (продуцираните материјали) за:
    - Двосмисленост
    - Нејасност
    - Неконзистентност
    - Пропусти
    - Грешки
- Нивно поправање
- Техничка ревизија од страна на тим од софтверски инженери, клиенти, крајни корисници и други засегнати страни

Requirements management

- Менаџирање на барањата во текот на нивното формулирање и менување кое ќе постои и за време на изработка и експлоатација на системот
- Следење на измените

Чекори при прибирањето на барања:

- Проценка на бизнис и технолошката изводливост на предложениот систем (feasibility study)
- Согледување на луѓето кои можат да помогнат во спецификацијата на барањата и да се разбере нивната организациона поставеност
- Дефинирање на технолошкото опкружување во кое системот треба да егзистира
- Идентификација на ограничувања во доменот (карактеристики на деловното опкружување специфични за апликацискиот домен)
- Дефинирање на еден или повеќе методи за прибирање на барања (интервјуа, фокусни групи, состаноци со тимови)
- Учество на повеќе страни во дефинирањето на барањата од различни точки на гледиште идентификувајќи ги принципите на секое евидентирано барање
- Идентификација на нејасните (непрецизни) барања како кандидати за испитување со конструкција на прототипови
- Креирање на кориснички сценарија со цел клиентите/корисниците подобро да ги идентификуваат клучните барања

Во процесот на прибирање на барања се “Произведува”:

- Извештај за потребите и изводливоста (feasibility study)
- Извештај ограничен на делокругот на системот или продуктот
- Листа на клиенти, корисници и останати инволвирани во прибирањето/дефинирањето на барањата
- Опис на техничко-технолошкото опкружување на системот
- Листа на барања (организирани според функција) и соодветните ограничувања на доменот на секое од нив
- Множество на кориснички сценарија кои проникнуваат во употребата на системот или продуктот во различни услови
- Прототипови развиени поради подобро дефинирање на барањата

При анализата (analysis and negotiation), барањата:

- Се категоризираат и се организираат во сродни групи
- Се испитува релацијата на секое барање со останатите
- Се проверуваат нивната конзистентност, недостатоци и недореченост
- Се рангираат по важност според барањата на клиентот/корисникот
- Во овој процес на анализа на барања се ппоставуваат и се одговара на прашања како:
    - Дали секое од барањата е конзистентно со глобалната цел на проектот?
    - Дали барањата се специфицирани на соодветно ниво на апстракција?
    - Дали барањето е неопходно или претставува дополнителна функција која не е есенцијална за реализација на системот?
    - Дали секое од барањата е соодветно ограничено и прецизно?
    - Дали секое барање има познат извор (кој го задал)?
    - Дали некое од барањата е во конфликт со некое друго?
    - Дали секое од барањата може да се реализира во зададената техничка околина во која ќе егзистира системот?
    - Дали секое од барањата може да се провери (тестира) откако ќе биде имплементирано?
- Разрешувањето на недоследностите во барањата се изведува со преговори
- Се рангираат барањата по важност
- Се прави група 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 кои се влез и излез од системот
- System modeling – модел на системот се гради (на хартија/нацрт) со цел да се евалуираат системските компоненти и нивните меѓурелации и како барањата и ограничувањата се вклопуваат во оваа слика

Requirements validation

- Квалитетот на она што е направено во процесот на инженеринг на барања се проценува во оваа фаза
- 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
- Валидацијата може формално да се изведе со проверка на секое барања во однос на колекција од прашања во форма на checklist

Менаџирање на барањата

- Претставува збир од активности кои помагаат на проекниот тим да ги идентификува, контролира и следи промените на барањата во текот на изведбата на проектот
- Секое барање се идентификува и означува со единствен идентификатор
    - <requirement type><requirement #?>
    - 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

- Некои од можните traceability tables се:
  - Features traceability table
  - Source traceability table
  - Dependency traceability table
  - Subsystem traceability table
  - Interface traceability table
- Вообичаено се одржуваат како дел од базата на барања

Репрезентација на софтверски барања

- IEEE стандард No. 830 – 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 фунцкија 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...)

FAST – Facilitated Application Specification Techniques

- 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
- Претежно користен за информациони системи

Collaborative Request Gathering

- Состанок
  - Оправданост на проектот
  - Презентирање на индивидуалните листи
  - Листите се комбинираат
  - Дискусија – консензус листа

    - Поделба на подтимови кои ќе развијат мини-спецификации
    - Критериуми за валидација
    - Делегираењ на пишувањето на комплетна драфт спецификација на база на сите материјали од FAST состанокот
- Правила на игра
- Агенда
- Олеснувач
- Механизам за дефиниција
- Целта е да се идентификува проблемот, да се предложат елементи на решението, да се разгледаат различните пристапи, да се специфицира прелиминарно множество на барања во атмосфера која е поволна за решавање на проблемот
- Пред состанокот – дистрибуција на “product request” документ, секој учесник прави листа на свои забелешки

Quality Function Deployment

- 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 инженерскит процес
- QFD идентификува три типа на барања:
    - Normal requirement (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)

Анализа и моделирање

Пристапи:

- Структурна анализа
- Објектно-ориентирана анализа

Модел е креиран за да биде основа на:

- Анализа за разбирање на информациите кои ги процесира системот, функциите со кои го прави тоа и однесувањето на системот со што процесот на прибирање и анализа на барањата го прави полесен и посистематски
- Ревизија, одредување на комплетност, конзистентност и прецизност на спецификацијата
- Дизајн обезбедувајќи му на дизајнерот основна репрезентација на софтверот кој треба да се прецлика во имплементациски контекст

Анализа

- Анализа е активност во која се гради модел
- Во неа се креираат и анализираат податочни, фунцкионални, модели на однесување и други модели кои треба да ја претстават суштината на она што треба да се изгради
- Се употребува комбинација од текстии и дијаграмска (графичка) претстава за да се опишат барањата за податоци, функционалност и однесување на начин кој е едноставен за разбирање и што е уште поважно овозможува проверка на коректноста, комплетноста и конзистентноста

Моделирање

- Податочно – ги дефинира податочните објекти, атрибутите и релациите
- ФУнкционално – изразува како податоците ќе се трансформираат во системот
- Моделирање на однесување – влијанието на настаните на системот

Анализа на барања

- Произведување модели за:
- Scenario-based models
- Data models
- Class-oriented models

- Flow-oriented models
- Behavioral models

Анализата на барања

- Резултира со спецификација за оперативните карактеристики на софтверот (function, data, behavior)
- Да ги воочи интерфејсите на софтверот со останатите елементи на системот
- Да ги воспостави ограничувањата кои софтверот мора да ги задоволи
- Да обезбеди репрезентација на
    - Информациите
    - Функциите
    - Однесувањето
    - Кои треба да се преточат во
        - Податочен дизајн
        - Архитектурен дизајн
        - Дизајн на кориснички интерфејс
        - Дизајн на компоненти

Чекори во анализа на барања:

- Препознавање на проблемот
- Проценка на проблемот и создавање решение
    - Дефинирање на сите ектерно видливи податочни објекти
    - Дефинирање и елаборирање на сите функции на софтверот
    - Разбираењ на однесувањето на софтверот во контекст на настаните кои влијаат на системот
    - Дефинирање на карактеристиките на интерфејсот кон системот
    - Откривање на дополнителни ограничувачки фактори во дизајнот
- Моделирање
- Спецификација
- Ревизија

Елементи на моделот за анализа

- Опишува што сака клиентот
- Создава основа за креирање на софтверкиот дизајн
- Се дефинира множество на барања кои ќе може да се проверат откако софтверот ќе биде направен

Структурна анализа – податоците и процесите кои ги трансформираат податоците ги разгледува како посебни ентитети

Објектно-ориентирана анализа – дефинирање на класи и начинот на кои колаборираат за да го решат проблемот

Елементи на анализа:

- Scenario-based models – use cases, user stories
- Class models – class diagrams, collaboration diagrams
- Behavioral models – state diagrams, sequence diagrams
- Flow models – DFDs data models

Scenario-Based modeling

- Прелиминарен Use Case
  - Начинот на кој актерот го користи системот за да постигне одредена цел
- Рафинирање на прелиминарниот Use Case
  - Дали актерот на одредено место може да преземе и други акции
  - Дали актерот може да се сретне со грешка на одредено место, каква и што при тоа?
  - Дали е можно актерот да се сретне со поинакво однесување и кои би можеле да бидат тие
- Составување формален Use Case
- Придружни UML модели (Activity, Swim line Diagrams)
- Примарни сценарија
  - Секвенцијалната репрезентација не разгледува алтернативни интеракции (наративната може и да содржи)
- Секундарни сценарија
  - Може ли да преземе други акции
  - Може ли да наиде на грешка
  - Може ли да наиде на поинакво однесување
- Use-case exception – исклучок опишува ситуација кое предизвикува системот да се однесува поинаку

Формален Use-case:

- Use case
- Iteration
- Primary actor
- Goal in context
- Preconditions
- Trigger
- Scenario
- Exceptions
- Priority
- When available frequency of use
- Channel to actor
- Secondary actors
- Open issues

Графички репрезентации

- Use case се фокусира на функционалните барања и барањата за однесување на системот – најчесто несоодветен за нефункционалните барања
- UML модели (дијаграми)

UML модели (дијаграми)

- Activity diagram
- Swimlane diagrams
- State machine diagram
- Interaction / Communication diagram
- Sequence / Timing diagrams

Data modeling

- Се користи кога софтверот има потреба од креирање, манипулација на комплексни податочни структури или интерфејс кон база на податоци, вообичаено се прави податочен модел
    - Entity-relationship diagram (ERD) ги опишува сите податочни објекти кои се внесуваат, чуваат, трансформираат и продуцираат во апликацијата
- Податочни објекти (data objects)
    - Предмет, екстерен ентитет, појава, настан, улога, организациона единица, ...
- Атрибути
    - Ги дефинираат особините на податочниот објект: именуваат, опишуваат, референцираат
- Релации
- Одговара на прашања во врска со податоците кои се обработуваат:
    - Кои се примарните податочни објекти кои ќе се процесираат?
    - Кој е составот на секој од обие објекти и кои се атрибутите кои го карактеризираат?
    - Каде се наоѓа објектот?
    - Која е релацијата на секој од објектите со останатите објекти?
    - Која е релацијата помеѓу објектите и процесите кои ги трансформираат
    - Се користи ERD (Entity relationship diagram) за полесно одговарање

ERD – entity relationship diagram

- Графичка нотација која овозможува идентификација на податочните објекти и нивните релации

- Ги разгледува чистите податоци и нивните релации, независно од процесите кои ги трансформираат
- Податочните модели се состојат од:
  - **Податочни објекти**
  - **Атрибути** кои ги опушуваат објектите
  - **Релации** кои ги поврзуваат податочните објекти меѓусебно
- Особини на податочен објект:
  - Единствен идентификатор
  - Улога
  - Опишан преку атрибутите
  - Тој е репрезентацијана било која композитна информација
  - Податочните објекти се во релација едни со други
- Типични објекти:
  - External entities (ентитет кој продуцира или консумира информации)
  - Things (извештај, приказ)
  - Occurrences or events (телефонски повик, аларм)
  - Roles (продавач)
  - Organization units (сметководство)
  - Places (магацин)
  - Structures (датотека)
- Атрибути:
  - Ги дефинираат својствата на објектот и имаат една од трите карактеристики:
    - Ја именуваат инстанцата на податочниот објект
    - Ја опишуваат инстанцата на податочниот објект
    - Референцираат кон друга инстанца во друга табела

Class-Based modeling

- Ги опишува
  - **Објектите** со кои системот ќе се манипулира
  - **Операциите** (методи, сервиси) кои ќе бидат применувани на објектите
  - **Релациите** (некои хиерархиски) помеѓу објектите
    - **Is-part-of**
    - **Has-knowledge-of**
    - **Depends-upon**
  - **Колаборациите** помеѓу класите
- **Идентификација и анализа на класите**
- **Специфицирање на атрибути**
  - Ја опишуваат класата, ја дефинираат и појаснуваат
  - Зависат од примената на класата
- **Дефинирање на операциите** (методите) – 4 вида
  - Операции со кои се манипулираат податоците
  - Операции кои прават некаква пресметка

    - Операции кои проверуваат состојба на објект
    - Операции кои го надгледуваат објектот за појава на контролирачки настан

- **CRC моделирање** (Class-Responsibility-Collaborator)
    - Index cards
        - Class name
        - Class responsibilities – anything the class knows or does
        - Collaborators – други потребни класи за да ги дадат потребните информации за класата да може да одговори на своите одговорности
- Асоцијации и зависности
- Пакети
- Категории на класи:
    - Entity classes – екстрахирани директно од проблемот, вообичаено работи кои се перзистентни за време на извршување на апликацијата, вообичаено се сместуваат во бази на податоци
    - Boundary classes (гранични класи) – за креирање на интефејси, содржат информации кои му се битни на корисникот, менаџираат со начинот на кој нетитетните објекти се презентираат на корисниците
    - Controller classes (контролер класи) – менаџирање на одредена работа од почеток до крај
        - Креирање и пормена на ентитетни објекти
        - Инстанцирање на гранични објекти при нивно примање информации од ентитетните објекти
        - Комплексни комуникации помеѓу множества на објекти
        - Валидација на податоците кои се проследуваат помеѓу објектите или корисникот и објектите
- Одговорнисти на класите – водилки за идентификација на одговорностите
    - Интелифенцијата на системот да биде дистрибуирана низ класите така што најдобро ќе може да одговори на предизвиците
    - Одговорностите да бидат што погенерално поставени
    - Информациите и однесувањет придружено со нив треба да биде во иста класа (енкапсулација)
    - Информациите за едно нешто треба да бидат локализирани во енда класа, а не дистрибуирани низ побеќе
    - Одговорностите треба да бидат делени помеѓу повеќе класи каде што е можно

Структурна анализа

- Разгледување на податоците и процесите кои ги трансформираат ка ко посебни ендитите
    - Податочни објекти кои се трансформираат додека се движат низ системот
    - Како апликацијата се однесува како последива на надворешните дејствија
    - Збирка на странсформации кои се вршат на податоците

Functional models

- Најмалку 3 генерички функции: влез, процесирање, излез

Behavioral models

- Стимул/одѕив карактеристики

Моделирање

Елементи на моделот за анализа

- Data dictionary – описи на сите објекти кои се користат во моделот
- Entity relationship diagram (ERD) – data modeling activity – ги опишува релациите помеѓу објектите
    - Атрибутите на секој објект се опишани во DOD (data object description)
- Data flow diagram (DFD) – со две цели:
    - Како податоците се трансформираат при нивното движење низ системот
    - Да ги опишие функциите кои ги трансформираат податоците
        - Описите на функциите од DFD се презентираат во PSPEC (process specification)
- State transition diagram (STD) – го опишува однесувањето на системот како последица на надворешни настани
    - Претставува различни модалитети на однесување (состојби) и начинот на кој се преминува од една во друга состојба
    - Дополнителни детали за контролните аспекти се содржани во CSPEC (control specification)

Релации

- Податочните објекти може да бидат поврзани со други објекти на различни начини
- Релацијата го претставува видот на поврзаност
- Врската во објект/релација паровите е двонасочна
- Кардиналност на релација

- Го дефинира максималниот број на објекти кои може да учествуваат во релација. Искажува колку појави на објектот X се поврзани со колку појави на објектот
    - 1:1
    - 1:N
    - M:N

Модалност на релација

- 0 доколку нема експлицитна потреба на релацијата да постои или истата е опциона
- 1 ако појавата на релацијата е задолжителна

Графичка претстава

- Објектите се претставуваат со правоаголници
    - Атрибутите во вид на табела за објектот или во елипси под објектот
- Релациите се именувани линиии кои ги поврзуваат објектите
- Кардинаалноста и модалноста се прикажуваат до излезот на линијата од објектот

Data Flow Modeling (Функционално моделирање)

- Информациите се трансформираат при нивниот тек низ системот
- Data Flow Diagram (DFD) е графичка репрезентација на текот и на информациите низ системот и трансформациите над нив
- Потребно е и data dictionary (подетално ги опишува податоците кои патуваат низ системот и process specification (PSEC) (ги специфицира деталите за процесирањето на податоците во еден процесен елемент