Kolegji AAB

CILËSI. LIDERSHIP. SUKSES!

Lënda:

Inxhinieria Softuerike

Pikat kyqe

Kapitulli 5-9

MSc.Elida Hadro

Kapitujt për kollokviumin e dytë

5. Modelimi i sistemit
6. Dizajni Arkitektural
7. Dizajni dhe Implementimi
8. Testimi i softuerit
9. Evolucioni i softuerit

Data e Kollokviumit të dytë: ***16 Janar 2024***

Kapitulli 5. Modelimi i Sistemit

Modelimi i sistemit

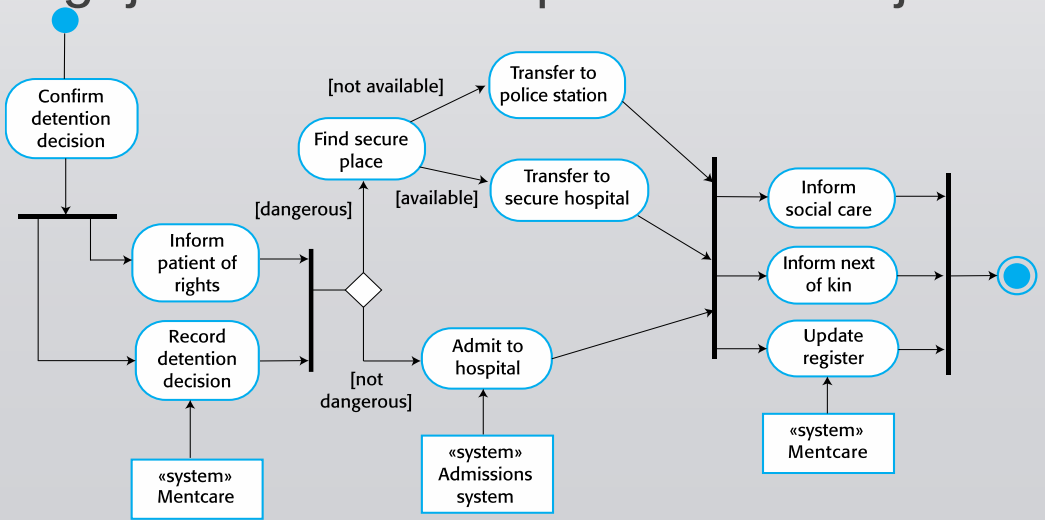
- Modelimi i sistemit është procesi i zhvillimit të modeleve abstrakte të një sistemi, ku secili model paraqet një pamje ose perspektivë të ndryshme të atij sistemi.
- Modelimi i sistemit ndihmon analistin të kuptojë funksionalitetin e sistemit, ndërsa modelet e sistemit përdoren kryesisht për të komunikuar me klientët.
- Modelimi i sistemit mundëson përfaqësimin e një sistemi duke përdorur shënime grafike dhe pothuajse gjithmonë bazohet në gjuhën e unifikuar të modelimit (UML).

Prespektivat e sistemit

- Mund të zhvillohen modele të ndryshme për të përfaqësuar sistemin nga këndvështrime të ndryshme. Për shembull:
  1. Një perspektivë e jashtme, ku bëhet modelimi i kontekstit ose mjedisit të sistemit.
  2. Një perspektivë ndërveprimi, ku bëhet modelimi i ndërveprimeve midis një sistemi dhe mjedisit të tij, ose midis përbërësve të një sistemi.
  3. Një perspektivë strukturore, ku bëhet modelimi i organizimit të një sistemi ose strukturës së të dhënave që përpunohen nga sistemi.
  4. Një perspektivë e sjelljes, ku bëhet modelimi i sjelljes dinamike të sistemit dhe mënyrën se si ai reagon ndaj ngjarjeve.

Modelet e kontestit

- **Modelet e kontekstit** tregojnë se si një sistem që po modelohet pozicionohet në një mjedis me sisteme dhe procese të tjera. Pra, bazohen në prespektivën e jashtme të sistemit.
- Kufijtë e sistemit vendosen për të përcaktuar se çfarë ka brenda dhe jashtë sistemit. Ato tregojnë sisteme të tjera, që përdoren ose varen nga sistemi që po zhvillohet.
- **Diagramet e aktiviteteve të UML**, të cilat tregojnë aktivitetet e përfshira në një proces ose në përpunimin e të dhënave.

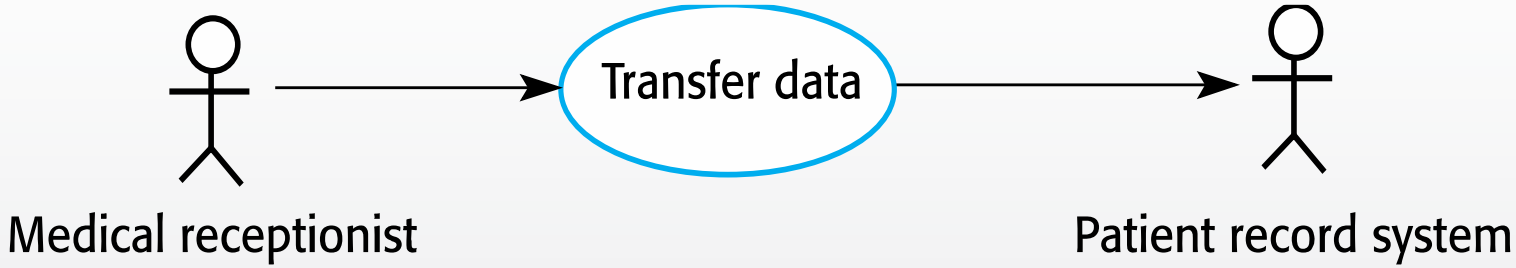


Modelet e ndërveprimit

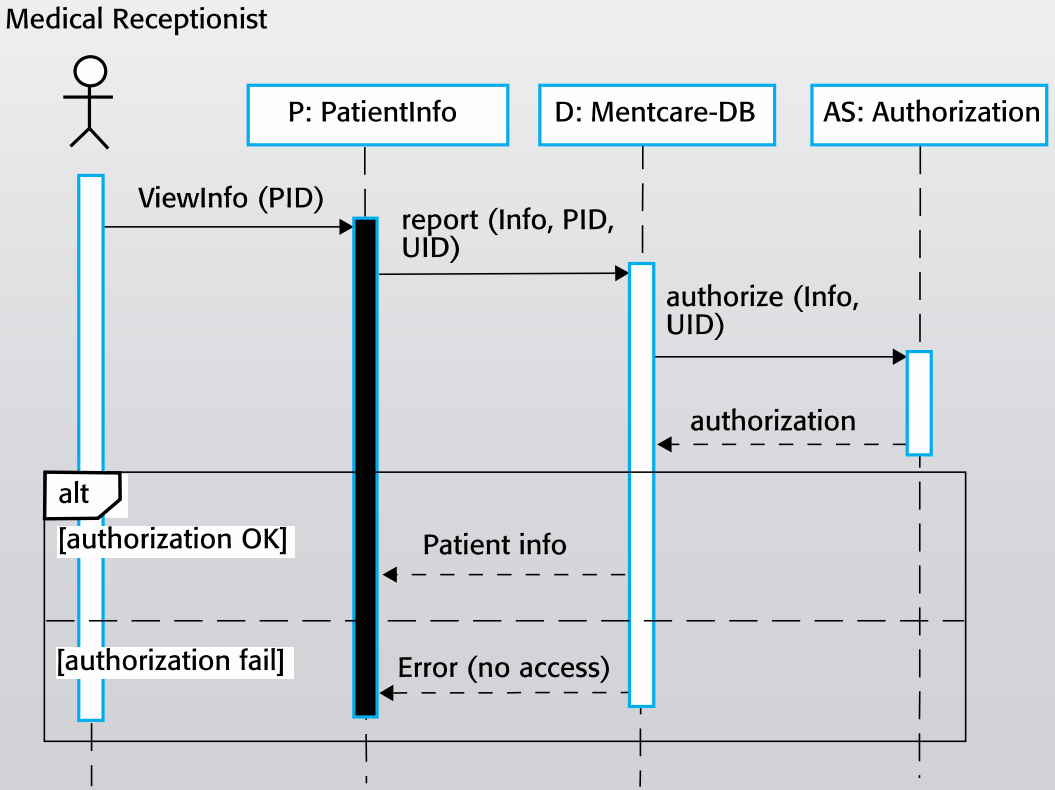
- **Modeli i ndërveprimit,** ku bëhet modelimi i ndërveprimeve midis një sistemi dhe mjedisit të tij, ose midis përbërësve të një sistemi.
- Të gjitha sistemet përfshijnë ndërveprim të njonjë lloji:
  - Modelimi i ndërveprimit të përdoruesit është i rëndësishëm pasi ndihmon në identifikimin e kërkesave të përdoruesit.
  - Modelimi i ndërveprimit sistem-me-sistem nxjerr në pah problemet e komunikimit që mund të lindin.
  - Modelimi i ndërveprimit të komponentëve na ndihmon të kuptojmë nëse një strukturë e propozuar e sistemit mund të jep performancën dhe besueshmërinë e kërkuar të sistemit.
- Këtu mund të përdoren *diagramet e rasteve të përdorimit* dhe *diagramet e sekuencës*.

Modelet e ndërveprimit /2

Diagramet Use-Case

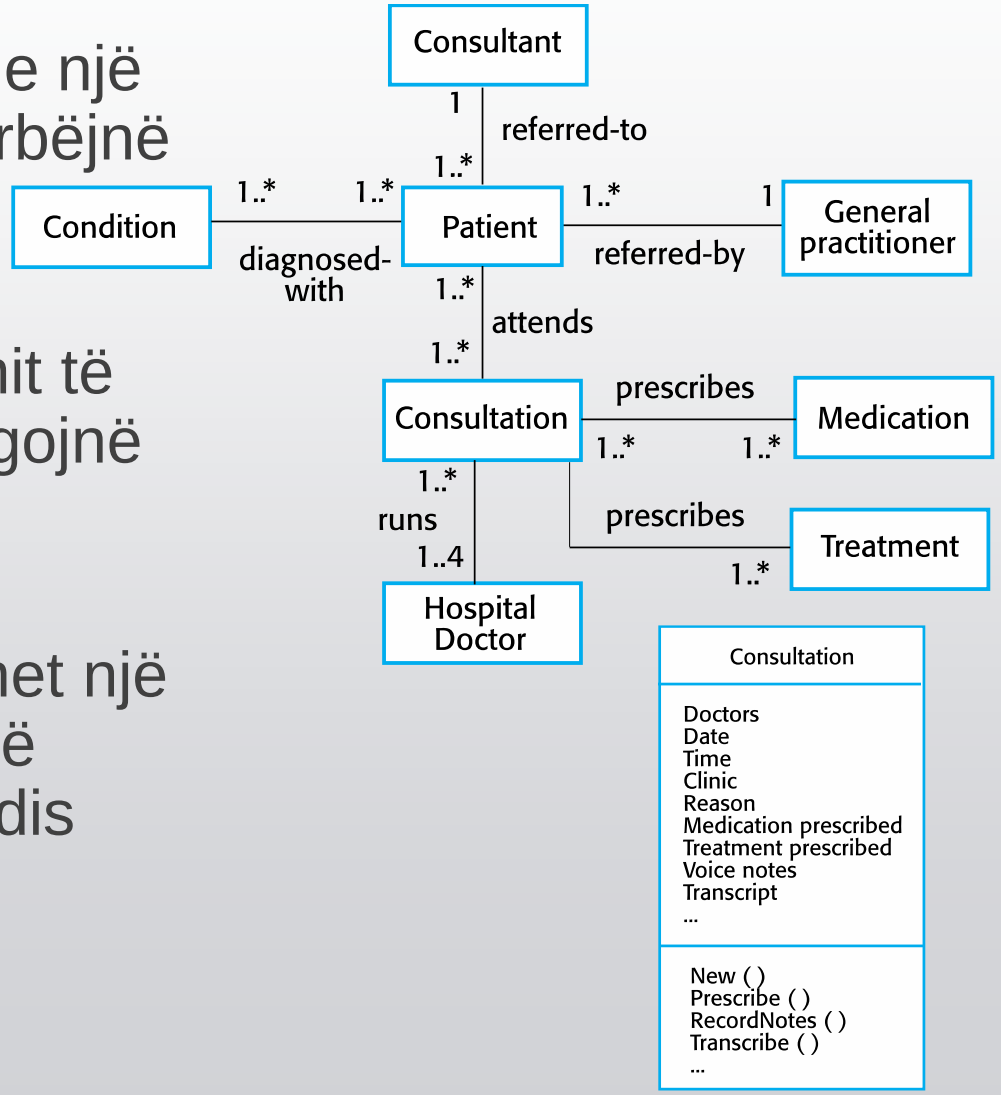


Diagramet e sekuencës



Modelet strukturore

- **Modelet strukturore** shfaqin organizimin e një sistemi për të kuptuar komponentet që përbëjnë atë sistem dhe marrëdhënieve të tyre.
- Modelet strukturore mund të jenë modele statike, të cilat tregojnë strukturën e dizajnit të sistemit, ose modele dinamike, të cilat tregojnë organizimin e sistemit kur ai është duke u ekzekutuar.
- *Diagramet e klasave* përdoren kur zhvillohet një model sistemi i orientuar nga objekti, për të treguar klasat në një sistem dhe lidhjet midis këtyre klasave.

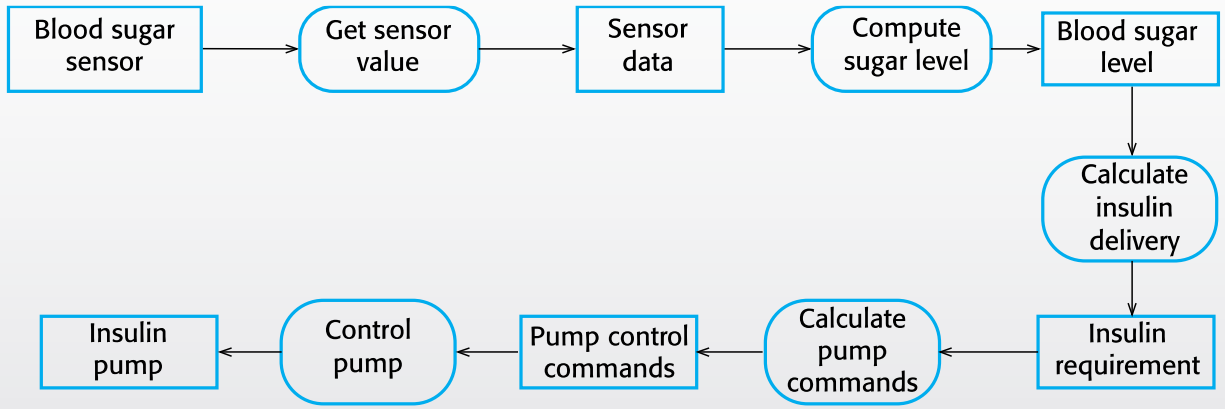


Modelet e sjelljes

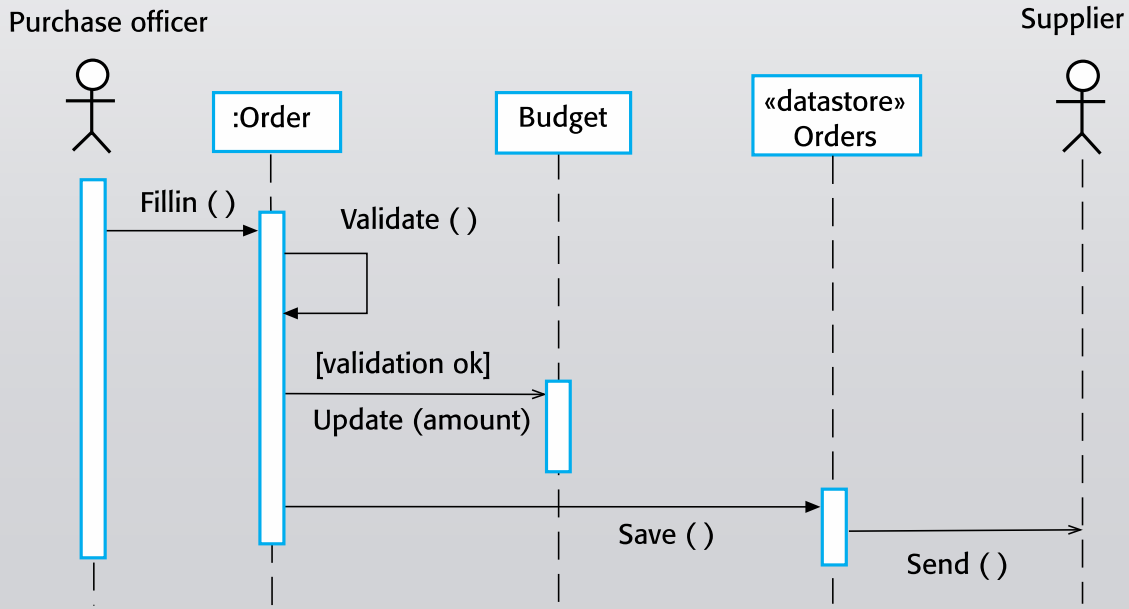
- **Modelet e sjelljes** përdoren për të përshkruar sjelljen dinamike të një sistemi ekzekutues. Kjo sjellje mund të modelohet nga këndvështrimi i të dhënave të përpunuara nga sistemi, ose nga ngjarjet që stimulojnë përgjigjet nga një sistem.
- **Modelimi i drejtuar në të dhëna**
  - Shumë sisteme biznesi janë sisteme të përpunimit të të dhënave që drejtohen kryesisht nga të dhënat.
  - **DFD** (Data Flow Diagramet) që prezantohen në formë të diagrameve të aktivitetit ose me diagramet e sekuencës.
- **Modelimi i drejtuar në ngjarje**
  - Sistemet e kohës reale shpesh drejtohen nga ngjarjet, me përpunim minimal të të dhënave.
  - (StateCharts) diagramet e gjendjes.

Modelet e sjelljes /2

Diagramet DFD



Diagramet e sekuencës



MDE dhe MDA

- **Inxhinieria e drejtuar nga modeli (MDE)** është një qasje ndaj zhvillimit të softuerit, ku modelet dhe jo programet janë rezultatet kryesore të procesit të zhvillimit.
  - Përkrahësit e MDE argumentojnë se kjo rrit nivelin e abstraksionit në inxhinierinë e softuerit, në mënyrë që inxhinierët të mos kenë më nevojë të shqetësohen me detajet e gjuhës së programimit ose specifikat e platformave të ekzekutimit.
- **Arkitektura e drejtuar nga modeli (MDA)** është paraardhësi i inxhinierisë më të përgjithshme të drejtuar nga modeli (MDE).
- **MDA** është një qasje e fokusuar në model për dizajnimin dhe implementimin e softuerit, që përdor një nëngrup modelesh UML për të përshkruar një sistem, ndërsa **MDE** merret me të gjitha aspektet e procesit të inxhinierisë së softuerit.

Kapitulli 6. Dizajni Arkitektural

Dizajni arkitektural

- **Dizajni arkitektural** ka të bëjë me të kuptuarit se si duhet të organizohet një sistem softuerik dhe dizajnimin e strukturës së përgjithshme të atij sistemi.
- Dizajni arkitektural është lidhja kritike midis dizajnit dhe inxhinierisë së kërkesave, pasi identifikon komponentet kryesore strukturore në një sistem dhe marrëdhëniet midis tyre.
- Rezultati i procesit të dizajnimit arkitektural është një model arkitektural që përshkruan se si sistemi është i organizuar si një grup komponentësh komunikues.

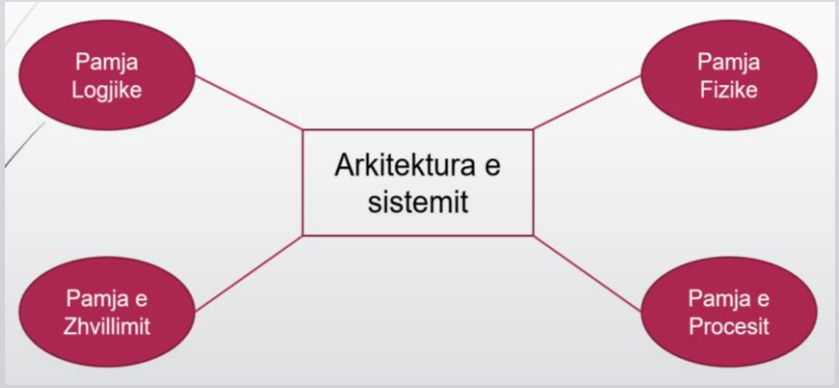
Pamjet arkitekturale

- **Çdo model arkitekturor tregon vetëm një pamje ose perspektivë të sistemit.**
  - Është e pamundur të paraqitet i gjithë informacioni për arkitekturën e një sistemi në një diagram të vetëm.
  - Mund të tregojë se si një sistem zbërthehet në module, si ndërveprojnë proceset e kohës së funksionimit ose mënyrat e ndryshme, në të cilat komponentët e sistemit shpërndahen në një rrjet.
  - Prandaj, si për dizajnin ashtu edhe për dokumentacionin, duhet të paraqiten pamje të shumta të arkitekturës së softuerit.

Modeli i pamjes 4 + 1 i arkitekturës së softuerit

1. Një **pamje logjike**, e cila tregon abstraksionet kryesore në sistem si objekte ose klasa objektesh.
2. Një **pamje e procesit**, e cila tregon se si është i përbërë sistemi nga procese ndërvepruese në kohën e ekzekutimit
3. Një **pamje e zhvillimit**, e cila tregon se si zbërthehet softueri për zhvillim.
4. Një **pamje fizike**, e cila tregon harduerin e sistemit dhe mënyrën se si komponentët e softuerit shpërndahen nëpër procesorët në sistem.

(+1) Këto pamje lidhen përmes rasteve të përdorimit (use case) ose të skenarëve.



Përfaqësimi i pikëpamjeve arkitekturore

- **Gjuha e Unifikuar e Modelimit (UML)** është një shënim i përshtatshëm për **përshkrimin** dhe **dokumentimin** e arkitekturave të sistemit.
  - Mirëpo, UML nuk përfshin abstraksione të përshtatshme për **përshkrimin** e sistemit të nivelit të lartë.
  - Por, UML ka më shumë vlerë kur **dokumentoni** një arkitekturë në detaje ose përdorni zhvillimin e drejtuar nga modeli.
- Propozohet përdorimi i gjuhëve më të specializuar të përshkrimit arkitekturor **(ADL – Architectural Description Language)**.
  - Për shkak se ADL-të janë gjuhë të specializuara, specialistët e domenit dhe aplikacioneve e kanë të vështirë të kuptojnë dhe përdorin ato.

Modelet arkitekturore

- **Një model arkitektural** është një përshkrim i stilizuar i praktikës së mirë të dizajnimit, i cili është provuar dhe testuar në mjedise të ndryshme.
- Modelet duhet të përfshijnë informacion se kur janë dhe kur nuk janë të dobishëm.
- Modelet mund të paraqiten duke përdorur përshkrime tabelare dhe grafike.

Modeli MVC (Model – View – Controller)

- Sistemi është i strukturuar në tre komponente logjike që ndërveprojnë me njëri-tjetrin.
  - **Komponenti Model** menaxhon të dhënat e sistemit dhe operacionet e lidhura me ato të dhëna.
  - **Komponenti View** përcakton dhe menaxhon se si të dhënat i paraqiten përdoruesit.
  - **Komponenti Controller** menaxhon ndërveprimin e përdoruesit dhe ia kalon këto ndërveprime *Pamjes* dhe *Modelit*.
- **Përdoret** kur ka mënyra të shumta për të paraqitur dhe ndërvepruar me të dhënat, gjithashtu kur kërkesat e ardhshme për ndërveprimin dhe paraqitjen e të dhënave janë të panjohura.

Modeli arkitekturor i shtresuar

- Organizon sistemin në shtresa, me funksionalitete të lidhura me secilën shtresë.
- Një shtresë ofron shërbime për shtresën mbi të, kështu që shtresat e nivelit më të ulët përfaqësojnë shërbimet bazë që mund të përdoren në të gjithë sistemin.
- **Përdoret** gjatë ndërtimit të objekteve të reja mbi sistemet ekzistuese; kur zhvillimi shpërndahet në disa ekipe dhe secilin ekip është përgjegjës për një shtresë funksionaliteti; kur ka një kërkesë për siguri në shumë nivele.

Modeli i repozitorit

- Në **modelin e repozitorëve (depove**) të gjitha të dhënat në një sistem menaxhohen në një depo qendrore, që është e aksesueshme për të gjithë komponentët e sistemit.
- Komponentët nuk ndërveprojnë drejtpërdrejt, vetëm përmes depove.
- Ju duhet ta **përdorni** këtë model kur keni një sistem, në të cilin gjenerohen vëllime të mëdha informacioni, që duhet të ruhen për një kohë të gjatë. Mund ta përdorni gjithashtu, në sistemet e drejtuara nga të dhënat.

Modeli klient/server

- Në një **arkitekturë klient-server**:
  - Sistemi paraqitet si një grup shërbimesh, me çdo shërbim të ofruar nga një **server** i veçantë.
  - **Klientët** janë përdorues të këtyre shërbimeve dhe aksesojnë serverët për t'i përdorur ato.
- **Përdoret** kur të dhënat në një bazë të dhënash të përbashkët duhet të aksesohen nga vende të ndryshme. Për shkak se serverët mund të riprodhohen, mund të përdoren gjithashtu, kur ngarkesa në një sistem është e ndryshueshme.

Arkitekturat e aplikacionit

- Një **arkitekturë e përgjithshme e aplikacionit** është një arkitekturë për një lloj të sistemit softuerik, që mund të konfigurohet dhe përshtatet për të krijuar një sistem që plotëson kërkesat specifike.
- **Përdorimi i arkitekturës së aplikacionit:**
  - Si pikënisje për dizajnin arkitekturor;
  - Si një listë kontrolli për dizajnimin (check-list);
  - Si një mënyrë e organizimit të punës së ekipit të zhvillimit;
  - Si një mjet për të vlerësuar komponentët për ripërdorim;
  - Si një fjalor për të komunikuar rreth llojeve të aplikacioneve.

Llojet e arkitekturës së aplikacionit

- Dy arkitektura të përgjithshme të aplikacionit shumë të përdorura janë sistemet e përpunimit të transaksioneve dhe sistemet e përpunimit të gjuhës.
  - **Sistemet e përpunimit të transaksioneve:** Përpunon kërkesat e përdoruesve për informacion nga një bazë të dhënash ose kërkesat për përditësimin e bazës së të dhënave.
    - *P.sh. Sistemet e tregtisë elektronike; Sistemet e rezervimit.*
  - **Sistemet e përpunimit të gjuhës:** Sistemet e përpunimit të gjuhës përkthejnë një gjuhë, në një paraqitje alternative të asaj gjuhe, si dhe për gjuhët e programimit, mund të ekzekutojnë gjithashtu kodin që rezulton.
    - *P.sh. Kompiluesit; Përkthyes komandues*.

Kapitulli 7. Dizajni dhe Implementimi

Dizajni dhe implementimi

- **Dizajni dhe implementimi** i softuerit është faza në procesin e inxhinierisë së softuerit ku zhvillohet një sistem softuerik i ekzekutueshëm.
- Aktivitetet e dizajnimit dhe implementimit të softuerit janë gjithmonë të ndërlidhura.
  - Dizajni i softuerit është një aktivitet krijues në të cilin identifikoni komponentët e softuerit dhe marrëdhëniet e tyre, bazuar në kërkesat e klientit.
  - Implementimi është procesi i realizimit të dizajnit si program.

Dizajni i Orientuar nga Objekti përmes UML

- **Proceset e strukturuara të dizajnimit të orientuara nga objekti** përfshijnë zhvillimin e modeleve të ndryshme të sistemit.
- Ekzistojnë shumë procese të ndryshme të dizajnimit të orientuara nga objekti, por varen nga organizata që përdor procesin.
- **Aktivitetet e zakonshme** në këto procese (Object Oriented) përfshijnë:
  1. Përcaktimi i kontekstit dhe ndërveprimet;
  2. Dizajnimi i arkitekturës së sistemit;
  3. Identifikimi i objekteve kryesore të sistemit;
  4. Zhvillimi i modeleve të dizajnimit;
  5. Specifikimi i ndërfaqeve të objekteve.

Aktivitetet e procesit të dizajnimit OO

Faza 1. Konteksti i sistemit dhe ndërveprimet

- **Të kuptuarit e marrëdhënieve** ndërmjet softuerit që po dizajnohet dhe mjedisit të tij të jashtëm është thelbësor për të vendosur se si të sigurohet funksionaliteti i kërkuar i sistemit dhe si të strukturohet sistemi për të komunikuar me mjedisin e tij.
- **Kuptimi i kontekstit** ju lejon gjithashtu të vendosni kufijtë e sistemit. Vendosja e kufijve të sistemit ju ndihmon të vendosni se cilat veçori zbatohen në sistemin që po projektohet dhe cilat veçori janë në sistemet e tjera të lidhura.

Aktivitetet e procesit të dizajnimit OO

Faza 2. Dizajni arkitektural

- Pasi të kuptohen ndërveprimet ndërmjet sistemit dhe mjedisit të tij në fazën e parë, këto informacione mund të përdoren për **dizajnimin e arkitekturës së sistemit**.
- Identifikohen **komponentët kryesorë** që përbëjnë sistemin dhe ndërveprimet e tyre dhe më pas mund organizohen komponentët duke përdorur një model arkitekuror, (p.sh. një model me shtresa ose klient-server).

Aktivitetet e procesit të dizajnimit OO

Faza 3. Identifikimi i klasës së objektit

- Identifikimi i klasave të objekteve është shpesh një pjesë e vështirë e dizajnit të orientuar drejt objektit.
- Nuk ka *'formulë magjike*' për identifikimin e objektit. Ai mbështetet në aftësitë, përvojën dhe njohuritë e dizajnerëve të sistemit.
- Identifikimi i objektit është një proces përsëritës. Nuk ka gjasa që ta merrni siç duhet herën e parë.
- Në këtë fazë të procesit të dizajnimit, duhet të keni disa ide rreth objekteve thelbësore në sistemin që po dizajnoni.

Aktivitetet e procesit të dizajnimit OO

Faza 4. Modelet e Dizajnit

- Modelet e dizajnit tregojnë objektet dhe klasat e objekteve dhe marrëdhëniet mes tyre.
- Kur përdoret UML për të zhvilluar një dizajn, duhet të zhvillohen dy lloje të modeleve të dizajnit:
  - **Modelet strukturore** përshkruajnë strukturën statike të sistemit në terma të klasave dhe marrëdhënieve të objekteve.
  - **Modelet dinamike** përshkruajnë ndërveprimet dinamike ndërmjet objekteve.

Aktivitetet e procesit të dizajnimit OO

Faza 5. Specifikimi i ndërfaqes

- **Ndërfaqet e objekteve** duhet të specifikohen në mënyrë që objektet dhe komponentët e tjerë të mund të dizajnohen paralelisht.
- **Objektet** mund të kenë disa ndërfaqe, secila prej të cilave është një këndvështrim mbi metodat që ofron.
- Mund të përdoren diagramet e klasave për specifikimin e ndërfaqes.

Çështje të implementimit

- Fokusi këtu nuk është vetëm te programimi, megjithëse, por edhe në çështje të tjera të implementimit:

1. **Ripërdorimi**: Shumica e softuerëve modernë janë ndërtuar duke ripërdorur komponentët ose sistemet ekzistuese. Kur jeni duke zhvilluar softuer, duhet të përdorni sa më shumë që të jetë e mundur kodin ekzistues.
2. **Menaxhimi i konfigurimit**: Gjatë procesit të zhvillimit, ju duhet të mbani gjurmët e shumë versioneve të ndryshme të secilit komponent të softuerit në një sistem të menaxhimit të konfigurimit.
3. **Zhvillimi host-target:** Softueri i prodhimit zakonisht nuk ekzekutohet në të njëjtin kompjuter si mjedisi i zhvillimit të softuerit. Përkundrazi, ju e zhvilloni atë në një kompjuter (host sistemi) dhe e ekzekutoni atë në një kompjuter të veçantë (sistemi i targetuar).

Zhvillimi me kod burimor të hapur

- **Zhvillimi me kod të hapur** (Open Source) është një qasje për zhvillimin e softuerit, në të cilën publikohet kodi burimor i një sistemi softuerik dhe vullnetarët ftohen të marrin pjesë në procesin e zhvillimit.
- Një parim themelor i zhvillimit me burim të hapur është që kodi burimor duhet të jetë i **disponueshëm i lirë dhe pa pagesë**, por kjo nuk do të thotë se kushdo mund të bëjë si të dojë me atë kod.
  - Ligjërisht, zhvilluesi i kodit (qoftë një kompani ose një individ) e zotëron ende kodin. Ata mund të vendosin kufizime se si të përdoret, duke përfshirë kushte ligjërisht të detyrueshme në licencë të softuerit me burim të hapur.

Licencimi me burim të hapur

- **Licenca GNU e Përgjithshme Publike (GPL)**
  - Kjo është një licencë e ashtuquajtur "*reciproke*" që do të thotë se nëse përdorni softuer me burim të hapur që është i licencuar sipas licencës GPL, atëherë duhet ta bëni atë softuer me burim të hapur.
- **Licenca GNU e Përgjithshme Publike e Vogël (LGPL)**
  - Është një variant i licencës GPL, ku mund të shkruani komponentë që lidhen me kodin me burim të hapur, pa pasur nevojë të publikoni burimin e këtyre komponentëve.
- **Licenca Standarde e Shpërndarjes Berkley (BSD).**
  - Kjo është një licencë *jo reciproke*, që do të thotë se nuk jeni i detyruar të ripublikoni çdo ndryshim ose modifikim të bërë në kodin me burim të hapur. Ju mund ta përfshini kodin (me burim të hapur) në sistemet e pronësuara që shiten.

Kapitulli 8. Testimi i Softuerit

Testimi i softuerit

- **Testimi** mund të zbulojë **praninë e gabimeve**,por **JO mungesën** e tyre.
- Testimi është pjesë e një procesi verifikimi dhe validimit më të përgjithshëm.
- Proceset e verifikimit dhe të validimit kanë të bëjnë me kontrollimin e softueri që po zhvillohet, se i plotëson specifikimet e tij dhe ofron funksionalitetin e pritur nga njerëzit që paguajnë për softuerin.
- **Verifikimi i softuerit** është procesi i kontrollit që softueri i plotëson kërkesat e deklaruara funksionale dhe jofunksionale.
- **Validimi i softuerit** është një proces më i përgjithshëm. Qëllimi i validimit të softuerit është të sigurojë që softueri të përmbushë pritshmëritë e klientit.

Fazat e testimit

- Në mënyrë tipike, një sistem softuerik komercial duhet të kalojë nëpër tre faza të testimit:
  1. **Testimi i zhvillimit**, ku sistemi testohet gjatë zhvillimit për të zbuluar gabime dhe defekte.
  2. **Testimi i lëshimit**, ku një ekip i veçantë testimi teston një version të plotë të sistemit, përpara se të lëshohet për përdoruesit.
  3. **Testimi i përdoruesit**, ku përdoruesit ose përdoruesit e mundshëm të një sistemi testojnë sistemin në mjedisin e tyre.

1. Testimi i zhvillimit

- **Testimi i zhvillimit** përfshin të gjitha aktivitetet e testimit që kryhen nga ekipi që zhvillon sistemin. Janë **tre faza** në testim të zhvillimit:
  - **Testimi i njësive**, ku testohen njësi programore individuale ose klasa objektesh. Testimi i njësisë duhet të fokusohet në testimin e funksionalitetit të objekteve.
  - **Testimi i komponentëve**, ku disa njësi individuale janë integruar për të krijuar komponentë të përbërë. Testimi i komponentëve duhet të fokusohet në testimin e ndërfaqeve të komponentëve.
  - **Testimi i sistemit**, ku disa ose të gjithë komponentët në një sistem janë të integruara dhe sistemi testohet në tërësi. Testimi i sistemit duhet të fokusohet në testimin e ndërveprimeve të komponentëve përbërëse të sistemit.

2. Testimi i lëshimit

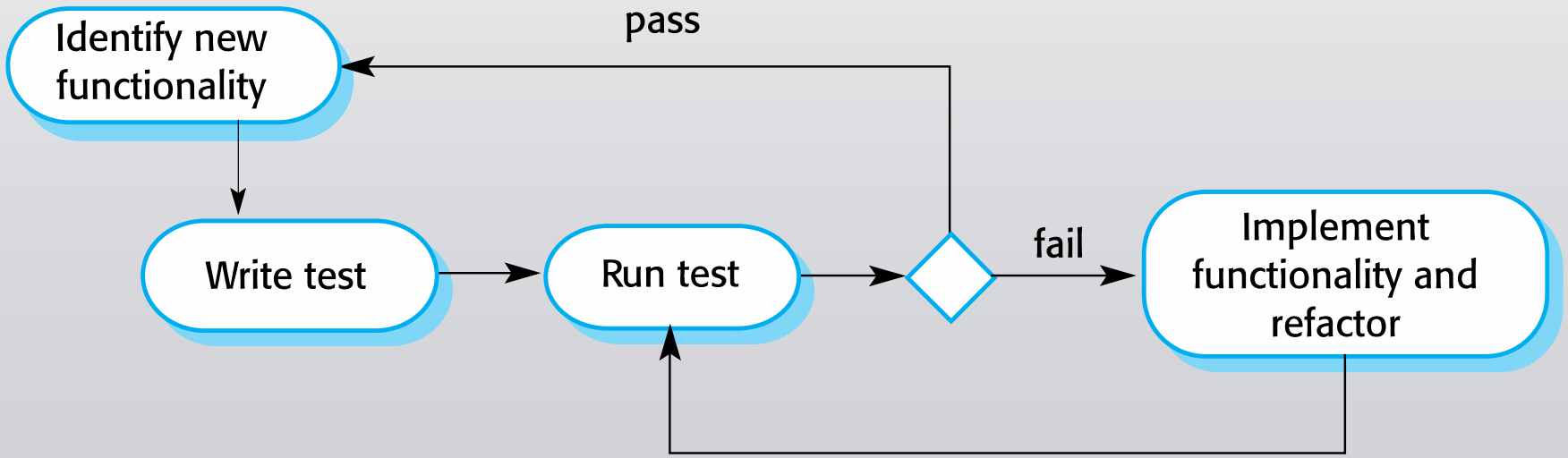
- **Testimi i lëshimit** është procesi i testimit të një versioni të veçantë të një sistemi, që do të përdoret nga përdoruesit real (të synuar).
- **Qëllimi kryesor** i procesit të testimit të lëshimit është të bindë prodhuesin e sistemit se ai është mjaft i mirë për përdorim.
- Prandaj, testimi i lëshimit duhet të tregojë se sistemi ofron funksionalitetin, performancën dhe besueshmërinë e tij të specifikuar dhe se nuk dështon gjatë përdorimit normal.

3. Testimi i lëshimit

- **Testimi i përdoruesit ose klientit** është një fazë në procesin e testimit, në të cilin përdoruesit ose klientët ofrojnë të dhëna dhe këshilla për testimin e sistemit.
- Testimi i përdoruesit është thelbësor, edhe kur është kryer testimi gjithëpërfshirës i sistemit dhe lëshimit.
  - Arsyeja për këtë është se ndikimet nga mjedisi i punës i përdoruesit kanë një efekt të madh në besueshmërinë, performancën, përdorshmërinë dhe qëndrueshmërinë e një sistemi. Këto nuk mund të përsëriten në një mjedis testimi.

Zhvillimi i drejtuar në testim

- **Zhvillimi i drejtuar nga testi (TDD)** është një praktikë e zhvillimit të softuerit që thekson shkrimin e testeve përpara se të shkruani kodin aktual.
- Ai ndjek një proces ciklik të shkrimit të një testi të dështuar, shkrimit të kodit minimal për të kaluar testin dhe më pas rifaktorimit të kodit.



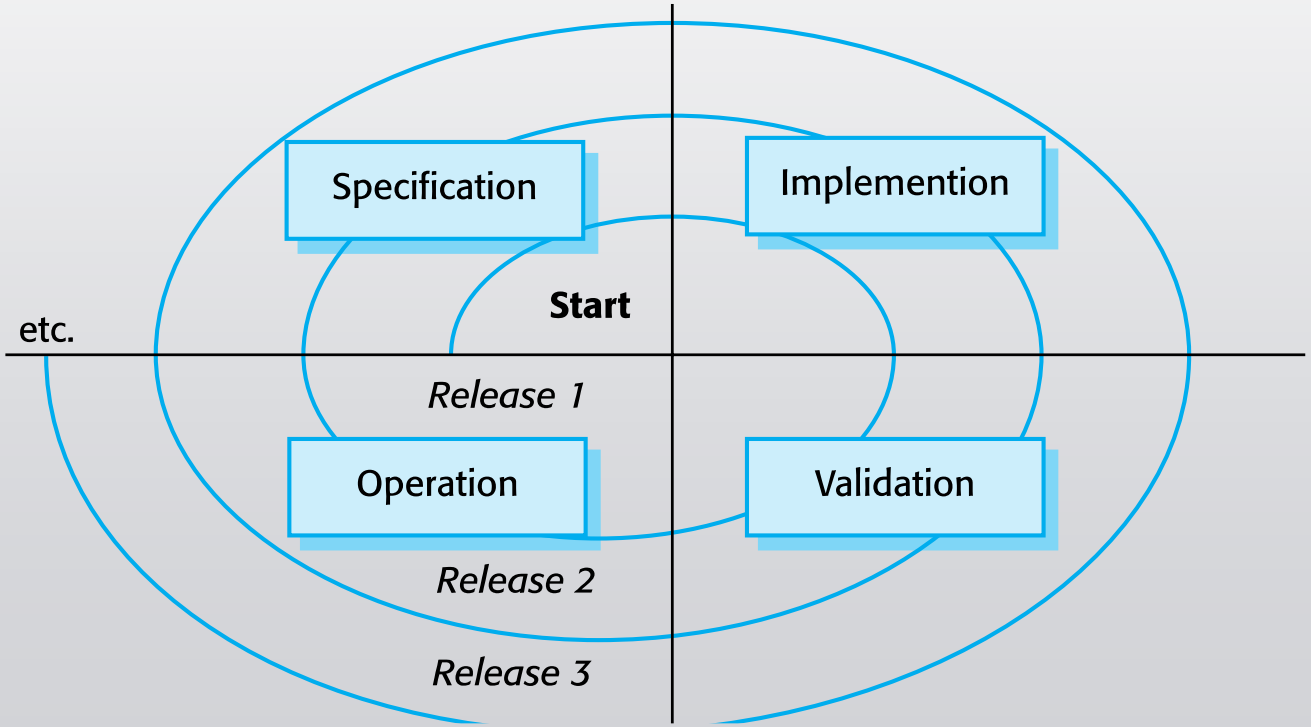
Benefitet e TDD

- **Mbulimi i kodit**: Çdo segment kodi që shkruani ka të paktën një test të lidhur, kështu që i gjithë kodi i shkruar ka të paktën një test.
- **Testimi i regresionit**: Një grup testesh regresioni zhvillohet gradualisht gjatë zhvillimit të një programi.
- **Korrigjimi i thjeshtuar**: Kur një test dështon, duhet të jetë e qartë se ku qëndron problemi. Kodi i sapo shkruar duhet të kontrollohet dhe modifikohet.
- **Dokumentacioni i sistemit**: Vetë testet janë një formë dokumentacioni që përshkruajnë se çfarë duhet të bëjë kodi.

Kapitulli 9. Evolucioni i Softuerit

Evolucioni i softuerit

- **Zhvillimi dhe evolucioni i softuerit** mund të mendohet si një proces i integruar përsëritës, që mund të përfaqësohet duke përdorur një model spiral.



Metodat e shkathëta dhe evolucioni

- **Metodat e shkathëta** bazohen në zhvillimin inkremental, kështu që kalimi nga faza e zhvillimit në evolucion është i pandërprerë, sepse këto janë të ndërlidhura mes vete.
  - Evolucioni është thjesht një vazhdim e procesit të zhvillimit bazuar në lëshimet e shpeshta të sistemit.
- **Teknikat e shkathëta** si: Testimi i automatizuar i regresionit dhe zhvillimi i drejtuar në testim janë veçanërisht të vlefshëm, kur bëhen ndryshime në një sistem.
- Ndryshimet mund të shprehen si tregime të përdoruesve (use case).

Problemet e dorëzimit të sistemit nga faza e zhvillimit në fazën e evolucionit

1. Aty ku ekipi i zhvillimit ka përdorur një qasje të shkathët, por ekipi i evolucionit nuk është i njohur me metodat e shkathëta dhe preferon një qasje të bazuar në plan.
   - Ekipi i evolucionit mund të presë dokumentacion të detajuar, për të mbështetur evolucionin dhe kjo nuk prodhohet në procese të shkathëta.
2. Aty ku është përdorur një qasje e bazuar në plan për zhvillim, por ekipi i evolucionit preferon të përdorë metoda të shkathëta.
   - Ekipi i evolucionit mund të duhet të fillojë nga e para, duke zhvilluar teste të automatizuara dhe kodi në sistem mund të mos jetë rifaktoruar dhe thjeshtuar, siç pritet në zhvillimin e shkathët.

Sistemet e trashëguara

- Sistemet e trashëguara janë sisteme më të vjetra që mbështeten në gjuhë dhe teknologji, që nuk përdoren më për zhvillimin e sistemeve të reja.
- Sistemet e trashëguara nuk janë vetëm sisteme softuerike, por janë sisteme më të gjera socio-teknike që përfshijnë harduerin, softuerin, bibliotekat dhe softuerët e tjerë mbështetës dhe proceset e biznesit.

Zëvendësimi i sistemeve të trashëguara

- **Zëvendësimi i sistemit të trashëguar** është i rrezikshëm dhe i shtrenjtë, kështu që shumë biznese vazhdojnë t'i përdorin këto sisteme.
  - Nëse një sistem i vjetër funksionon në mënyrë efektive, kostot e zëvendësimit mund të tejkalojnë kursimet që vijnë nga kostot e reduktuara të mbështetjes së një sistemi të ri.
- Zëvendësimi i sistemit është i rrezikshëm për disa arsye:
  - Mungesa e specifikimit të plotë të sistemit.
  - Integrim i ngushtë i sistemit dhe proceseve të biznesit.
  - Rregullat e biznesit pa dokumente të përfshira në sistemin e trashëguar.
  - Zhvillimi i softuerit të ri mund të jetë me vonesë dhe/ose mbi buxhet.

Ndryshimi i sistemeve të trashëguara

- **Sistemet e trashëguara** janë të shtrenjta për t'u ndryshuar për disa arsye:
  - Nuk ka stil të qëndrueshëm të programimit.
  - Përdorimi i gjuhëve të vjetëruara të programimit, me pak njerëz të disponueshëm me këto aftësi gjuhësore.
  - Dokumentacioni i pamjaftueshëm i sistemit.
  - Degradimi i strukturës së sistemit.
  - Optimizimi i programit mund t'i bëjë ato të vështira për t'u kuptuar.
  - Gabime të të dhënave, dyfishim dhe mospërputhje mes tyre.

Menaxhimi i sistemeve të trashëguara

- Shumica e organizatave kanë një buxhet të kufizuar për mirëmbajtjen dhe përmirësimin e sistemeve të trashëgimisë. Ata duhet të vendosin se si të marrin kthimin më të mirë të investimit të tyre.
- Organizatat që mbështeten në **sistemet e trashëgimisë** duhet të zgjedhin një strategji për evolucionin e këtyre sistemeve.
  1. Shkëputeni plotësisht sistemin
  2. Lëreni sistemin të pandryshuar dhe vazhdoni me mirëmbajtjen e rregullt
  3. Riinxhinieroni sistemin për të përmirësuar mirëmbajtjen e tij
  4. Zëvendësoni të gjithë ose një pjesë të sistemit me një sistem të ri
- Strategjia e zgjedhur duhet të varet nga cilësia e sistemit dhe vlera e tij e biznesit.

Mirëmbajtja e softuerit

- **Mirëmbajtja e softuerit** është një proces gjeneral i modifikimit të një programi pasi të jetë vënë në përdorim.
- Mirëmbajtja normalisht nuk përfshin ndryshime të mëdha në arkitekturën e sistemit.
- Ndryshimet implementohen duke modifikuar komponentët ekzistues dhe duke shtuar komponentë të rinj në sistem.
- **Llojet e mirëmbajtjes** së softuerit:
  - Riparimet e defekteve (Mirëmbajtje korrigjuese)
  - Përshtatja mjedisore (Mirëmbajtje adaptive)
  - Shtimi dhe modifikimi i funksionalitetit (Mirëmbajtje perfekte)

Mirëmbajtja e softuerit

- **Mirëmbajtja e softuerit** është një proces gjeneral i modifikimit të një programi pasi të jetë vënë në përdorim.
- Mirëmbajtja normalisht nuk përfshin ndryshime të mëdha në arkitekturën e sistemit.
- Ndryshimet implementohen duke modifikuar komponentët ekzistues dhe duke shtuar komponentë të rinj në sistem.
- **Llojet e mirëmbajtjes** së softuerit:
  - Riparimet e defekteve (Mirëmbajtje korrigjuese)
  - Përshtatja mjedisore (Mirëmbajtje adaptive)
  - Shtimi dhe modifikimi i funksionalitetit (Mirëmbajtje perfekte)

Kostoja e mirëmbajtjes e softuerit

- Zakonisht është më e shtrenjtë të shtosh veçori të reja në një sistem gjatë mirëmbajtjes sesa të shtosh të njëjtat veçori gjatë zhvillimit.
- Disa nga arsyet pse ndodh kjo:
  - Ekip i ri (mirëmbajtjes) duhet të kuptojë programet që po mirëmbahen.
  - Ndarja e mirëmbajtjes dhe zhvillimit do të thotë se nuk ka asnjë nxitje për ekipin e zhvillimit për të shkruar softuer të mirëmbajtur.
  - Puna e mirëmbajtjes së programit është e papëlqyeshme
    - Stafi i mirëmbajtjes është shpesh i papërvojë dhe ka njohuri të kufizuara për fushën.
  - Me vjetërim të programeve, struktura e tyre degradohet dhe ato bëhen më të vështira për t'u ndryshuar.

Ri-inxhinierimi i softuerit

- Ristrukturimi ose rishkrimi i një pjese ose të gjithë një sistemi të trashëguar, pa ndryshuar funksionalitetin e tij.
- Zbatohet kur disa, por jo të gjitha nën-sistemet e një sistemi më të madh kërkojnë mirëmbajtje të shpeshtë.
- **Riinxhinierimi** përfshin shtimin e përpjekjeve për t'i bërë ato më të lehta për t'u mirëmbajtur.
  - Sistemi mund të përfshijë ridokumentimin e sistemit, rifaktorimin e arkitekturës së sistemit, përkthimin e programeve në një gjuhë programimi moderne, ose modifikimin dhe përditësimin e strukturës së të dhënave të sistemit.

Aktivitetet e procesit të ri-inxhinierimit

1. **Përkthimi i kodit burimor**
   - Konvertimi i kodit në një gjuhë të re programuese.
2. **Inxhinierimi i kundërt**
   - Analizimi i programit për ta kuptuar dhe për të njerrë dokumentimin;
3. **Përmirësimi i strukturës së programit**
   - Ristrukturimi automatik i programit, për të kuptuar dhe lexuar më lehtë;
4. **Modularizimi i programit**
   - Riorganizimi i strukturës së programit, duke larguar tepricat;
5. **Riinxhinierimi i të dhënave**
   - Pastrimi dhe ristrukturimi i të dhënave të sistemit.

Rifaktorimi

- **Rifaktorimi** është procesi i përmirësimit të një programi, për të ngadalësuar degradimin e tij gjatë implementit të ndryshimeve.
- Mund të mendojmë rifaktorimin si ***"mirëmbajtje parandaluese***", që redukton problemet e ndryshimit të ardhshëm.
- **Rifaktorimi** përfshin modifikimin e një programi për të përmirësuar strukturën e tij, për të zvogëluar kompleksitetin ose për ta bërë më të lehtë për t'u kuptuar.
- Rifaktorimi është një pjesë e metodave të shkathëta, sepse këto metoda bazohen në ndryshim.

Dallimi mes riinxhinierimit dhe rifaktorimit

- **Ri-inxhinierimi** bëhet pasi një sistem është mirëmbajtur për disa kohë dhe kostot e mirëmbajtjes po rriten.
  - Ju përdorni mjete të automatizuara për të përpunuar dhe ri-inxhinieruar një sistem të trashëguar, për të krijuar një sistem të ri që është më i mirëmbajtur.
- **Rifaktorimi** është një proces i vazhdueshëm i përmirësimit, gjatë gjithë procesit të zhvillimit dhe evolucionit.
  - Ai synon të shmangë degradimin e strukturës dhe kodit që rrit kostot dhe vështirësitë e mirëmbajtjes së një sistemi.