# ŪDENSKRITUMA MODELIS

# (The waterfall model)

Modelis ietver vairākus posmus, kuri tiek izpildīti secīgi. Tātad izstrādes process līdzīgi ūdenim kaskādē “krīt” no viena posma uz otru, tā radies šāds nosaukums. Šis modelis ir derīgs vairākām programmatūras sistēmu klasēm. Ūdenskrituma modelis attēlo programmatūras dzīves ciklu un paredz pieciem posmiem atbilstošu darbību izpildi.

1. posms. Prasību analīze un definīcijas. Šo posmu var sadalīt divos apakšposmos: prasību analīze un specifikāciju definēšana jeb noteikšana. Pirmajā apakšposmā jānoskaidro un jādefinē tās prasības, kuras sistēmā jāievēro, jāformulē sistēmas izstrādes mērķis. Kā galveno jautājumu sīki jāizskata lietotāja saskarne ar sistēmu (lietotāja interfeiss). Jāpievērš uzmanība tādām prasībām kā informācijas apstrādes laiks, kļūdu varbūtība, sistēmas reakcija uz nepareizām lietotāja darbībām. Prasību analīzes gaitā obligāti jākonsultējas ar lietotājiem, lai izstrādātu tieši viņiem vajadzīgu sistēmu. Šā apakšposma izpildes rezultātus dokumentē, parasti sastādot “Prasību dokumentu”. Specifikāciju definēšanas apakšposmā izveido dokumentu “Prasību specifikācija” ar realizējamo funkciju precīzu aprakstu.

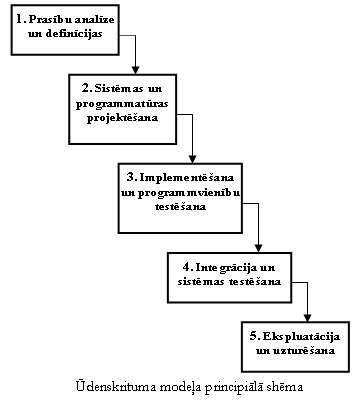
2. posms. Sistēmas un programmatūras projektēšana. Šajā posmā izveido sistēmas kopējo arhitektūru, kā arī izstrādā algoritmus atbilstoši specifikācijām. Projektēšanas posms iedalāms divos apakšposmos: arhitektūras projektēšana un detalizētā jeb komponentu projektēšana. Pirmajā apakšposmā izstrādā sistēmas struktūru, sadalot sistēmu nelielos blokos tā, lai katru no tiem varētu realizēt vienā vai vairākās programmās (vai moduļos). Katra bloka realizēšana tiek uzdota vienam izstrādātājam vai izstrādātāju grupai. Tad formulē prasības katram modulim, t.i. realizējamās funkcijas, lielumu, izpildes laiku, kā arī izstrādā katra sistēmas komponenta projektējumu. Jāizveido arī testi sistēmas pārbaudei (testēšanai) un šī darba izpildes grafiks. Projektēšanas posms arī ir jādokumentē.

3. posms. Implementēšana un programmvienību testēšana. Posms ietver divus apakšposmus: kodēšanu un programmvienību testēšanu. Kodēšana ir algoritmu realizēšana kādā programmēšanas valodā. Šim nolūkam parasti izmanto augsta līmeņa programmēšanas valodas un strukturētās programmēšanas metodes. Otrajā apakšposmā izpilda programmvienību pārbaudi jeb testēšanu, izmantojot iepriekš sastādīto testēšanas plānu un kontroles testus. Šīs darbības mērķis ir pārbaudīt, kā katra programmvienība izpilda savas funkcijas un atbilst specifikācijai. Apakšposma rezultējošais dokuments ir pārskats par programmvienību (moduļu) testēšanas rezultātiem.

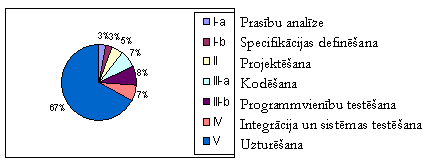
4. posms. Integrācija un sistēmas testēšana. Kad visi programmas moduļi ir pārbaudīti (testēti) un ir konstatēts, ka tie veic savas funkcijas pareizi, atsevišķie moduļi tiek apvienoti vienā sistēmā. Šim procesam seko sistēmas atkļūdošana. Pēc tam notiek sistēmas testēšana, izmantojot iepriekš sastādīto testēšanas plānu. Posmā noformē divus dokumentus: testēšanas kopsavilkuma pārskatu un lietotāja ceļvedi.

Kad četru posmu darbi ir izpildīti, sistēmu var nodot klientam vai piedāvāt pircējiem.

5. posms. Ekspluatācija un uzturēšana. Tas ir visilgākais posms programmatūras dzīves ciklā. Tas sākas ar sistēmas instalāciju un nodošanu ekspluatācijā. Sistēmas ekspluatācijas laikā var atklāties kļūdas, kas netika konstatētas un izlabotas iepriekšējos posmos. Klients parasti arī prasa izdarīt kādas izmaiņas sistēmā, jo viņš savulaik nevarēja precīzi noformulēt visas prasības. Dažreiz sistēma arī jāpielāgo citiem darbības apstākļiem (jāadaptē sistēma). Visas šīs darbības izpilda uzturēšanas posmā. Tātad sistēmas uzturēšana ir svarīga problēma, it īpaši, ja funkcionē vairākas sistēmas kopijas. Piemēram, ja lietotājs kādā organizācijā ir konstatējis kļūdu, tad šī kļūda jāizlabo visās kopijās.



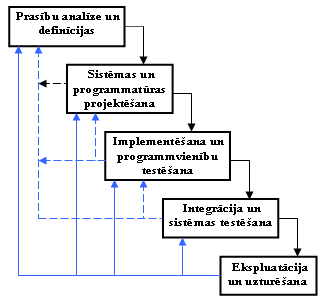
# Programmatūras izstrādes procesa laika patēriņa sadales diagramma



# PRECIZĒTS ŪDENSKRITUMA MODELIS

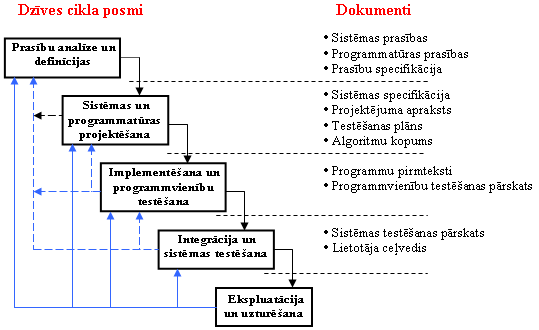
Ūdenskrituma modeli, kura shēmu Jūs redzējāt, nosauc arī par lineāro modeli. Patiesībā programmatūras izstrādes process nav lineārs, jo jebkurā izstrādes posmā var būt konstatētas kļūdas. Tādā gadījumā jāatgriežas uz iepriekšējo vai pat uz kādu no vēl agrākajiem posmiem. Tas bieži notiek, ja iepriekšējie posmi nebija pietiekami labi un precīzi izstrādāti vai klients bija nepareizi formulējis kādu prasību (vai vispār to aizmirsis). Var būt arī gadījumi, kad sistēmas darbība neatbilst lietotāja prasībām un sistēma funkcionē pavisam ne tā, kā paredzējis lietotājs, tad izstrāde jāsāk no jauna.

Tātad programmatūras izstrādes process satur veicamo darbību iteratīvu secību, kā Jūs redzat šajā attēlā. Tā ir ūdenskrituma modeļa precizētā shēma. Šo modeli dažreiz nosauc par iteratīvo ūdenskrituma modeli.



# DOKUMENTĀCIJA

Katra ūdenskrituma modeļa posmā tiek izstrādāti vairāki dokumenti. Dokumentu izveidi reglamentē Latvijas valsts standarti. Nepieciešamu dokumentu kopumu saskaņo ar klientu (programmatūras sistēmas pasūtītāju). Attēlā ir parādīts, kādi dokumenti jāsagatavo katrā dzīves cikla posmā.



# PĒTNIECISKĀS PROGRAMMĒŠANAS MODELIS

# (Exploratory programming)

Modelis paredz šādu pieeju programmatūras izstrādei: vispirms ātrā tempā jāizstrādā darbspējīga sistēma, kuru pēc tam pakāpeniski modificē, lai īstenotu vajadzīgās prasības. Šo pieeju parasti lieto, kad lietotāji nevar formulēt savas prasības.

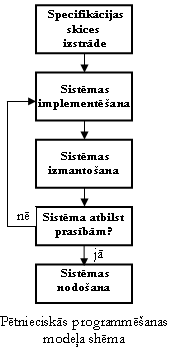
Programmatūras izstrāde ar pētnieciskās programmēšanas modeļa lietošanu ietver četrus posmus. 1. posms. Specifikācijas skices izstrāde. Šajā posmā izstrādā prasību specifikācijas skici, iekļaujot tajā tās prasības, kuras varēja noskaidrot pārrunās ar klientu.

2. posms. Sistēmas implementēšana. Posmā balstoties uz specifikācijas skici izveido sistēmas sākumversiju un piedāvā to ekspluatācijai. Kad šo posmu izpilda atkārtoti, sistēma tiek modificēta (uzlabota) atbilstoši jaunām lietotāju prasībām. Šāda sistēmas uzlabošana un precizēšana tiks izpildīta vairākas reizes, kamēr visas lietotāju prasības būs ievērotas.

3. posms. Sistēmas izmantošana. Lietotāji strādā ar sistēmu. Darba gaitā viņiem rodas jaunas vēlmes un/vai viņi prasa mainīt kaut ko sistēmā.

4. posms. Sistēmas novērtēšana un nodošana. Jā lietotājiem ir kādas prasības vai vēlmes, tad jāatgriežas uz otro posmu. Sistēmas nodošana notiek tad, kad lietotāji pilnīgi apmierināti ar programmatūras sistēmas darbību, t.i. visas viņu prasības ir īstenotas.

Lai sekmīgi lietotu šo modeli, jāizmanto tehnika, kas ļauj ātri izveidot un vairākkārt uzlabot sistēmu, jo vajadzīgās izmaiņas jāveic un jādemonstrē iespējami ātrāk. Tam nepieciešami šādi resursi: augsta līmeņa programmēšanas valodas, kā LISP vai Prolog; ražīga aparatūra; integrētie programmatūras rīki.



Pētnieciskā programmēšana ir labs līdzeklis, ja ir grūti vai neiespējami izstrādāt detalizētu sistēmas specifikāciju.

# PĒTNIECISKĀS PROGRAMMĒŠANAS MODEĻA PROBLĒMAS

Pētniecisko programmēšanu reti izmanto lielu un ilgi darbojošos sistēmu izstrādei. Tam ir trīs iemesli:

1. Programmatūras izstrādes process ir grūti kontrolējams, jo pētnieciskās programmēšanas gaitā, ievērojot to, ka sistēma pastāvīgi tiek mainīta, ir dārgi un neefektīvi sagatavot daudz dokumentu. Savukārt, pārvaldība parasti balstās uz izstrādes laikā noformētiem dokumentiem. ar kuriem varētu iepazīties. Tas ir galvenais iemesls, kāpēc šo pieeju neizmanto plaši.

2. Sistēmas parasti slikti strukturētas. Pētnieciskās programmēšanas tendence ir iegūt rezultātus sistēmās, kuru struktūra nav pietiekami labi definēta, jo projektēšanas posms pilnā apjomā šeit nav paredzēts. Turpmākām izmaiņām ir tendence bojāt sistēmas struktūru. Apkalpot un uzturēt tādu sistēmu ir grūti, un tas dārgi maksā, it īpaši, ja apkalpojošais personāls nav piedalījies sistēmas izstrādē, kas parasti ir raksturīgi lielām sistēmām.

3. Speciālie kadri bieži ir vajadzīgi, jo nav skaidrs, kā efektīvi izmantot pētnieciskajā programmēšanā programmētāju un citu izstrādātāju meistarību, kvalifikāciju un pieredzi.

Šīs problēmas nenozīmē, ka pētnieciskā programmēšana jānoraida. Tā labi noder:

• relatīvi mazu sistēmu izstrādei;

• sistēmu izstrādei ar īso programmatūras dzīves ciklu;

• apakšsistēmu izstrādei, kad nevar definēt sistēmas specifikācijas (piem., mākslīgā intelekta sistēmas).

# PROTOTIPĒŠANAS MODELIS

# (Prototyping)

Modeli lieto, ja lietotāji nespēj formulēt savas prasības un definēt, kā sistēmai jāfunkcionē. Šādos gadījumos ir divas pieejas: pētnieciskā programmēšana un prototipēšana (izstrāde pēc prototipa). Abi risinājumi nodrošina ātru izstrādi un darbspējīgu sistēmu īstenošanu.

Prototipēšanas mērķis ir atklāt sistēmas prasības.

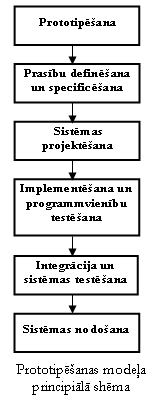
Programmatūras izstrāde, lietojot prototipēšanas modeli, ietver divus posmus:

1. Prototipēšana. Šajā posma balstoties uz specifikācijas skici izstrādā sistēmas prototipu un piedāvā to ekspluatācijai. Lietotāji, strādājot ar sistēmas prototipu, izsaka savas prasības un vēlmes, kuras konstatē izstrādātāji. Tā viņi var noskaidrot sistēmas prasības.

2. Sistēmas izstrāde. Šajā posma notiek sistēmas izveide no jauna, parasti lietojot ūdenskrituma modeli, kaut gan var izmantot arī citu modeli, piemēram, soļmodeli vai sistēmas komplektēšanu no atkārtotās lietošanas komponentiem.

Attēlā Jūs redzat prototipēšanas modeļa principiālo shēmu, kurā prototipēšanai seko ūdenskrituma modeļa posmi.

Tagad var aplūkot prototipēšanas modeļa precizētu shēmu



# PROTOTIPĒŠANAS MODEĻA SHĒMA

Programmatūras izstrādes process, lietojot prototipēšanas modelis, ir atspoguļots attēlā.

Prototipu izstrādā, pamatojoties uz sistēmas specifikācijas skici. Tad notiek prototipa izmantošana un novērtēšana. Ja lietotāji prasa kaut ko mainīt prototipā, lai izvēloties labāko variantu, izstrādātāji to izpilda (atgriezeniskā saite attēlā). Kad visas prasības ir noskaidrotas, seko sistēmas prasību definēšana un specificēšana, sistēmas projektēšana, implementēšana un testēšana. Ja kādu no prototipa komponentiem var iekļaut sistēmā bez izmaiņām, tad tas arī tiek darīts (attēlā tas parādīts ar raustīto līniju).

Prototipēšanas modeļa lietošanai ir vairākas pozitīvās īpašības:

1) var ātrāk identificēt un likvidēt nesapratni starp programmatūras izstrādātājiem un klientiem, ja sistēmas funkcijas ir jau demonstrētas;

2) var noteikt lietotāja servisa funkcijas sistēmas prototipa ekspluatācijas laikā;

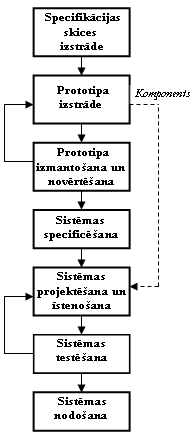
3) grūti realizējamo vai nenoteikto lietotāja servisu var identificēt un precizēt;

4) izstrādātājs var definēt nenoteiktās un/vai pretrunīgās prasības, ja prototips ir izstrādāts.

5) prototips ir pamats specifikāciju definēšanai, lai izstrādātu augstas kvalitātes sistēmu.

Prototipēšanas modelim ir arī trūkums – prototipa izveide palielina programmatūras izstrādes izmaksas.

Viens no galveniem jautājumiem prototipēšanas modeļa izmantošanā : Kā izstrādāt prototipu?



# PROTOTIPA IZSTRĀDE

Prototipa izstrāde ietver četrus soļus un paredz šādu darbību izpildi:

1. Formulēt prototipēšanas mērķi. Prototipēšanas mērķis varētu būt izstrādāt sistēmu ar kādu no sekojošiem uzdevumiem:

- lai precizētu lietotāja interfeisu;

- lai apstiprinātu sistēmas funkcionālās prasības;

- lai demonstrētu sistēmas lietojuma iespējas utt.

Prototipam nav jāatbilst vairākiem mērķiem vienlaicīgi. Ja prototipēšanas mērķis netiks skaidri formulēts, lietotāji nesapratīs, ka tas ir prototips, un uzskatīs, ka prototipam jānodrošina visas priekšrocības, ko varētu sniegt tikai gatava sistēma pēc tās pilnīgas pabeigšanas.

2. Izlemt, kādas funkcijas jāiekļauj prototipā un kādas nefunkcionālas prasības jāprototipē. Šeit ir divas iespējas:

- iekļaut prototipā visas paredzētās funkcijas, bet nepilna apjomā;

- iekļaut prototipā tikai atsevišķās funkcijas.

Prototipa ekspluatācija var arī palīdzēt atcelt vai atvieglot nefunkcionālās prasības, piemēram sistēmas reakcijas prasību.

3. Izstrādāt un īstenot prototipu. Prototipa īstenošanā jālieto tie paši rīki un standarti, kas plānoti izmantošanai, sistēmu implementējot. Prototipā var, piemēram, ignorēt kļūdu apstrādi utt.

4. Novērtēt prototipu. Tā ir svarīgā prototipēšanas fāze. Prototipēšanas mērķis ir jāizmanto, lai sastādītu novērtēšanas plānu. Pēc šā plāna būs jānoskaidro lietotāju prasības un sistēmas specifikācija. Ja kādas prototipa funkcijas neapmierina lietotājus un/vai izstrādātāji nevar atklāt prasības, tad prototips būs jāmodificē.

# SOĻMODELIS (The incremental model)

Modelis paredz, ka programmatūras izstrāde un tās nodošana klientam tiek veikta pakāpeniski pa soļiem.

Programmatūras izstrāde, lietotjot soļmodeli ietver trīs pamatposmus.

1. Sistēmas prasību definēšana un specificēšana. Šajā posmā definē un specificē prasības sistēmai.

2. Sistēmas arhitektūras projektēšana. Posmā projektē sistēmas arhitektūru un sadala to neatkarīgās sastāvdaļās (increments), kuras var veidot un nodot pa soļiem.

3. Sistēmas sastāvdaļas specificēšana, izveidošana, testēšana un nodošana. Posmā specificē, izstrādā, pārbauda un nodod kārtēju sistēmas sastāvdaļu, integrējot to ar iepriekšējām. Šo posmu atkārto vairākas reizes, kamēr sistēma tiks pilnīgi izstrādāta (atgriezeniskā saite attēlā).

Tātad programmatūras izstrāde, balstoties uz soļmodeli, nodrošina lietotājiem iespēju lietot, pārbaudīt un novērtēt katru sistēmas sastāvdaļu programmatūras izveides laikā. Tā ir šī modeļa pozitīva īpašība.

Soļmodeli ir lietderīgi izmantot, situācijās, kad:

- nav pietiekošu darbinieku programmatūras sistēmas izstrādei pilnā apjomā noteiktajā termiņā. Šai gadījumā sistēmas pirmversiju (pirmo sastāvdaļu) var izveidot neliela izstrādātāju komanda līdz tam laikam, kamēr tiks pieņemti jauni līdzstrādnieki;

- ir plānots novērtēt programmatūras projekta tehniskos riskus, piemēram, ja būs jālieto jauna aparatūra vai sistēmas programmatūra.

-[]



# FORMĀLĀS TRANSFORMĀCIJAS MODELIS

# (Formal transformation)

Formālās transformācijas modelis paredz sistēmas formālās specifikācijas izstrādi, kura pēc tam tiek transformēta programmā ar speciālo rīku palīdzību.

Formālā specifikācija ir specifikācija, kas uzrakstītā tādā valodā, kuras vārdnīca, sintakse un semantika ir formāli definēta, t.i., tā balstās uz matemātiku.

Formālās transformācijas modelis ietver trīs fāzes:

1. Formālās specifikācijas izstrāde. Šajā fāzē balstoties uz definētām prasībām izveido formālo specifikāciju.

2. Transformāciju izpilde. Šajā fāzē ar datorizēto rīku palīdzību veic transformāciju secību, lai pārveidotu specifikāciju sistēmas pirmkodā. Informācija par izpildāmiem pārveidojumiem un to gaitā pieņemtiem lēmumiem tiek ievietota speciālā failā, kas nosaukts par formālās izstrādes ierakstu (FII). Transformāciju secības izpildes rezultātā tiek veidots programmatūras sistēmas pirmkods.

3. Sistēmas testēšana un nodošana.

Formālās specifikācijas izstrādes pozitīvās īpašības:

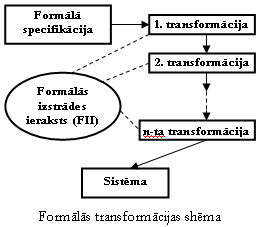
• formālās specifikācijas izstrāde nodrošina prasību un projektējuma saprotamību un samazina kļūdu skaitu. Tā ir labas projektēšanas bāze;

• formālā specifikācija balstās uz matemātiku, līdz ar to FS var analizēt, lietojot matemātiskās metodes. Piemēram, var pierādīt specifikācijas nepretrunīgumu un pabeigtību, kā arī pierādīt, ka implementācija atbilst specifikācijai;

• formālo specifikāciju var izmantot par pamatu programmatūras verifikācijai, piemēram lietot kā ceļvedi testu izveidei komponentu testēšanai;

• principā FS var veidot automātiski, ja izveidoti rīki, kuri palīdz izstrādāt, saprast un atkļūdot FS.

Diemžēl formālās transformācijas modelis netiek plaši lietots programmatūras izstrādē.



# FORMĀLĀS TRANSFORMĀCIJAS MODEĻA PROBLĒMAS

Formālās specifikācijas (FS) tehnika netiek plaši lietota programmatūras izstrādē vairāku iemeslu dēļ:

1. Formālā modeļa (un specifikācijas) izstrāde prasa vairāk laika un lielākas izmaksas, salīdzinot ar citiem programmatūras izstrādes modeļiem. Grūti demonstrēt, ka relatīvi lielas FS izveides izmaksas parasti izraisa kopējo izstrādes izmaksu samazināšanos.

2. Saskarne ar klientiem ir apgrūtināta, jo klienti nepārzina formālās specifikācijas tehniku un līdz ar to viņi nevar kontrolēt un novērtēt FS izveides procesu.

3. Daudzi programmatūras inženieri neprot veidot FS, t.i. speciāla mācīšanās ir nepieciešama. Formālās specifikācijas izstrādei labi jāzina diskrēta matemātika un loģika.

4. Programmatūras sistēmu dažas klases grūti specificēt, lietojot šo tehniku, it īpaši lietotāja interfeisa dialoga komponentus.

5. Trūkst metožu un rīku šīs tehnikas atbalstam, jo datorzinātnes pētījumi galvenokārt veltīti valodu attīstīšanai.

# SISTĒMAS KOMPLEKTĒŠANA NO ATKĀRTOTĀS LIETOŠANAS KOMPONENTIEM

# (System assembly from reusable components).

Šajā pieejā ir pieņemts, ka sistēmas tiek izveidotas no jau esošiem komponentiem. Tādējādi sistēmas izstrādes process realizē vienlaicīgi ne vien komponentu savākšanu, bet arī sistēmas izveidošanu. Programmatūras izstrādei ir paredzēts izmantot atkārtotās lietošanas komponentu (ALK) bibliotēku.

Programmatūras izstrāde ietver trīs fāzes:

1. Sistēmas specifikācijas izstrāde. Šajā fāzē balstoties uz ALK katalogu izstrādā sistēmas specifikāciju.

2. Sistēmas savākšana no ALK. Šajā fāzē izveido sistēmu no komponentiem, kas glabāti ALK bibliotēkā.

3. Sistēmas testēšana un nodošana. Fāzē testē izveidoto sistēmu un nodod klientam.

Sistēmas komplektēšana no ALK ļauj samazināt programmatūras izstrādes izmaksas, bet tās lietošanai ir nepieciešama ALK bibliotēka un katra komponenta detalizēta dokumentācija.

