

TEMATISKAIS PLĀNS

2. TEMATS

Kā izplānot un izveidot lietotājam atbilstošu programmētu risinājumu un uzraudzīt tā īstenošanu? (25 stundas)

Pilnveidot prasmi analizēt lietotāja vajadzības un definēt problēmu, kuru iespējams risināt, izstrādājot datorprogrammu. Veidot izpratni par programmatūras izstrādes plānošanas procesu, ietverot lietotāja prasību apkopošanu, risinājuma idejas formulēšanu un prasību specifikācijas izstrādi. Attīstīt prasmi strukturēti plānot programmatūras izstrādes procesu, izvērtēt iespējamās tehnoloģiskos risinājumus un izstrādāt vienkāršu lietotājam saprotamu sistēmas aprakstu. Pilnveidot spēju dokumentēt izstrādātā risinājuma funkcionalitāti, plānot tā īstenošanu un uzraudzīt projekta izstrādes gaitu, ievērojot sadarbības un drošas programmatūras izstrādes principus.

Tematā realizētie valsts pamatizglītības standarta sasniedzamie rezultāti

- T.V.2.4.1. Analizē vienkāršus ikdienas darba procesus un saskata tajos vai to daļās automatizācijas iespējas.
- T.V.2.4.2. Nosauc un raksturo ar piemēriem programmatūras dzīves cikla galvenos posmus – specifikēšanu, projektēšanu, izstrādi, testēšanu un uzturēšanu.
- T.O.2.4.4. Pēc parauga sastāda vienkāršotu programmatūras prasību specifikāciju atbilstoši uzdevumam, ievērojot mērķauditorijas vajadzības.
- T.O.2.4.5. Pēc parauga izstrādā vienkāršotu programmatūras projektējumu, t. sk. lietotāja saskarnes plānojumu, atbilstoši programmatūras prasību specifikācijai.
- T.O.2.4.3. Izstrādā programmatūru individuāli, ievērojot programmatūras izstrādes modeļa galvenos posmus un veic vienkāršotu izstrādes procesa dokumentēšanu.
- T.O.2.4.6. Veic programmatūras izstrādi un testēšanu atbilstoši izstrādātajai programmatūras prasību specifikācijai un projektējuma aprakstam.
- T.O.2.4.7. Izstrādā izveidotās programmatūras vienkāršotu lietotāja ceļvedi, prezentē un dalās ar izveidoto programmatūru, ievērojot mērķauditorijas specifiku.
- T.O.2.4.9. Lieto versiju pārvaldības rīku programmatūras risinājuma izstrādes procesā un sadarbojas ar citiem izstrādes dalībniekiem.
- T.O.2.3.2. Izmanto dažādu lietotņu dokumentu koplietošanas iespējas, nosakot atšķirīgiem lietotājiem piekļuves, rediģēšanas un komentēšanas tiesības.
- T.O.2.3.4. Sagatavo aptaujas vai anketēšanas formas, organizē datu iegūšanu un veic iegūto datu apkopošanu.
- T.O.3.1.5. Ievēro programmatūras licences nosacījumus, intelektuālā īpašuma un personas datu aizsardzības prasības programmatūras risinājuma izstrādē un izmantošanā.

2. TEMATS. Kā izplānot un izveidot lietotājam atbilstošu programmētu risinājumu un uzraudzīt tā īstenošanu?		25 stundas
1	Programmatūras prasību specifikācija	6
2	Lietotāja saskarnes plānošana un dizaina uzmetums Figma	2
3	Programmatūras projektējums	6
4	Programmatūras izstrāde, dzīves cikls, versiju kontrole	5
5	Programmatūras implementācija, integrācija un testēšana	6

Atbilstība valsts vispārējās vidējās izglītības standartam

Apakštēmas Nr.	Apakštēmas nosaukums	Standarta punkti
1	Programmatūras prasību specifikācija	T.O.2.4.1., T.O.2.4.4., T.O.2.3.4.
2	Lietotāja saskarnes plānošana un dizaina uzmetums Figma	T.O.2.4.5., T.O.2.4.4.
3	Programmatūras projektējums	T.O.2.4.2., T.O.2.4.5.
4	Programmatūras izstrāde, dzīves cikls, versiju kontrole	T.O.2.4.2., T.O.2.4.3., T.O.2.4.9., T.O.2.3.8.
5	Programmatūras implementācija, integrācija un testēšana	T.O.2.4.6., T.O.2.4.7., 3.1.5.

Sasniedzamie rezultāti

Zināšanas

- Programmatūras prasību specifikācija apraksta lietotāja vajadzības, sistēmas mērķi un vēlamu funkcionalitāti no lietotāja un pasūtītāja viedokļa. (T.Li.2.)
- Programmatūras izstrāde notiek secīgos posmos: prasību noteikšana, projektēšana, izstrāde, testēšana un uzturēšana. (T.Li.2.)
- Programmatūras projektējums palīdz plānot sistēmas uzbūvi, lietotāja saskarni un galvenās funkcijas pirms izstrādes sākšanas. (T.Li.2.)
- Lietotāja saskarnes plānojumam un dizaina uzmetumam jābūt saprotamam, pārskatāmam un atbilstošam lietotāja vajadzībām. (T.Li.3.)
- Informācijas strukturēšana, pārskatāms izkārtojums un piemērotu vizuālo elementu lietojums atvieglo risinājuma uztveri un lietošanu. (T.Li.3.)
- Lai izveidotu lietotājam atbilstošu risinājumu, ir jāizzina mērķauditorija, tās vajadzības, paradumi un lietošanas situācijas. (T.Li.3.)
- Labas komunikācijas prasmes, empātija un piemērotu prasību iegūšanas metožu izmantošana palīdz iegūt kvalitatīvu informāciju no lietotājiem. (T.Li.1.)
- Programmatūras izstrādē būtiska ir prasību, projektējuma un izmaiņu dokumentēšana, lai risinājumu varētu saprast, pilnveidot un uzturēt. (T.Li.2.)
- Versiju pārvaldība palīdz organizēt programmatūras izstrādes procesu, sekot izmaiņām un sadarboties ar citiem izstrādātājiem. (T.Li.2.)
- Programmatūras implementācija, integrācija un testēšana ļauj pārbaudīt, vai risinājums darbojas atbilstoši noteiktajām prasībām. (T.Li.2.)
- Autortiesības, licences, intelektuālā īpašuma un personas datu aizsardzības prasības jāievēro visā programmatūras izstrādes un lietošanas procesā. (T.Li.2.)

Prasmes

- Analizē vienkāršus ikdienas darba procesus un nosaka, kuras darbības iespējams automatizēt ar programmatūras palīdzību. (T.O.2.4.1.)
- Nosauc un skaidro programmatūras izstrādes galvenos dzīves cikla posmus (prasību noteikšana, projektēšana, izstrāde, testēšana, uzturēšana) un to nozīmi risinājuma izveidē. (T.O.2.4.2.)
- Atbilstoši uzdevumam izstrādā vienkāršotu programmatūras prasību specifikāciju, nosakot sistēmas mērķi, funkcijas un mērķauditorijas vajadzības. (T.O.2.4.4.)
- Veido vienkāršotu programmatūras projektējumu, t. sk. lietotāja saskarnes plānojumu un sistēmas struktūras uzmetumu, balstoties uz izstrādāto prasību specifikāciju. (T.O.2.4.5.)
- Izstrādā programmatūras risinājumu individuāli, ievērojot programmatūras izstrādes posmus un dokumentējot izstrādes procesu. (T.O.2.4.3.)
- Veic programmatūras implementāciju un testēšanu atbilstoši izstrādātajai programmatūras prasību specifikācijai un projektējumam. (T.O.2.4.6.)
- Izstrādā vienkāršotu lietotāja ceļvedi izveidotajai programmatūrai un prezentē izstrādāto risinājumu, skaidrojot tā funkcionalitāti un lietošanu. (T.O.2.4.7.)
- Izmanto versiju pārvaldības rīkus programmatūras izstrādes procesā un sadarbojas ar citiem izstrādes dalībniekiem. (T.O.2.4.9.)
- Izmanto dokumentu koplietošanas iespējas, nosakot dažādiem lietotājiem atšķirīgas piekļuves, rediģēšanas un komentēšanas tiesības koplietos dokumentos. (T.O.2.3.2.)
- Izveido aptaujas vai anketēšanas formas lietotāju vajadzību noskaidrošanai, organizē datu iegūšanu un apkopo iegūtos rezultātus. (T.O.2.3.4.)
- Programmatūras izstrādes procesā ievēro programmatūras licences nosacījumus, intelektuālā īpašuma un personas datu aizsardzības prasības. (3.1.5.)

Komplekss sniedzamais rezultāts

- Izveido digitāla risinājuma analīzi, nosaka iesaistītās personas un to vajadzības, analizē sistēmas lietošanas procesus un identificē automatizācijas iespējas. Izstrādā risinājuma funkcionālo aprakstu un pamato izvēlētajās tehnoloģijās. (T.V.1.1.2., T.O.2.4.1.)
- Risinājuma izstrādes gaitā izmanto piemērotus sadarbības un projekta vadības rīkus, izveido vienkāršotu programmatūras prasību specifikāciju, balstoties uz lietotāja vajadzībām, un prezentē izstrādāto risinājuma koncepciju. (T.V.1.2.1., T.V.2.3.8., T.O.2.4.4.)

Ieradumi

- Attīsta ieradumu cieņpilni komunicēt ar lietotājiem un citām iesaistītajām personām, uz klausot viņu vajadzības un viedokļus risinājuma izstrādes procesā. (Tikums – laipnība; vērtība – cilvēka cieņa)
- Attīsta ieradumu analizēt problēmas un strukturēti plānot to risināšanu, sadalot sarežģītu uzdevumu vairākos secīgos programmatūras izstrādes posmos. (Tikumi – gudrība, mērenība; vērtība – darbs)
- Attīsta ieradumu atbildīgi izmantot informācijas un komunikācijas tehnoloģijas, sadarbojoties ar citiem, izmantojot koplietošanas un projekta vadības rīkus programmatūras izstrādes procesā. (Tikums – gudrība; vērtība – darbs)

Temata saturs un tā realizācija

Temata apakšdaļa	Saturs
Programmatūras prasību specifikācija	Skolēni apgūst programmatūras prasību specifikācijas izstrādes principus, veidojot izpratni par to, kā programmatūras izstrādes procesā tiek noskaidrotas lietotāja vajadzības, formulētas sistēmas prasības un strukturēta informācija pirms programmatūras izstrādes uzsākšanas. Tēmas ietvaros skolēni iepazīstas ar funkcionālajām un nefunkcionālajām prasībām, prasību iegūšanas metodēm, iesaistītajām pusēm, lietošanas gadījumiem un vienkāršotiem sistēmas modeļiem, sasaistot prasību specifikāciju ar reāli izmantotām digitālajām sistēmām. Aktualizācijas posmā skolēni analizē, kāpēc programmatūras izstrādē rodas pārpratumi starp lietotāju, pasūtītāju un izstrādātāju, un spriež, kā dokumentācija palīdz šādus pārpratumus mazināt. Balstoties uz piemēriem no esošām sistēmām, piemēram, Kalkulators un Modinātājs, skolēni nonāk pie secinājuma, ka kvalitatīvi formulētas prasības palīdz precīzāk saprast sistēmas mērķi, nepieciešamo funkcionalitāti un izstrādes robežas. Apjēgšanas posmā skolēni iepazīstas ar programmatūras prasību specifikācijas jēdzienu, tās vietu programmatūras izstrādes ciklā un tipisko dokumenta struktūru. Skolēni apgūst atšķirību starp funkcionālajām un nefunkcionālajām prasībām, mēcās noteikt prasību kvalitātes pazīmes un atpazīt tipiskas kļūdas prasību formulējumos. Tiek veidota izpratne par prasību iegūšanas metodēm, prasību avotiem, iesaistītajām personām, mērķauditoriju un lietošanas gadījumiem, kā arī par diagrammu nozīmi sistēmas procesu un struktūras skaidrošanā. Praktiskajā darbībā skolēni soli pa solim izstrādā vienkāršotu programmatūras prasību specifikāciju vienai esošai sistēmai pēc izvēles — Kalkulatoram vai Modinātājam. Sākotnēji skolēni formulē funkcionālās un nefunkcionālās prasības, pēc tam nosaka iesaistītās personas un mērķauditoriju, apraksta lietošanas gadījumus un izveido prasību iegūšanas plānu. Turpinājumā skolēni sagatavo PPS dokumenta struktūras uzmetumu, vienkāršotu sistēmas modeli, viedokļu hierarhiju un darbību diagrammas, bet noslēgumā izstrādā strukturētu prasību sarakstu, pārbauda prasību kvalitāti un noformē vienotu gala dokumentu. Refleksijas posmā skolēni izvērtē, kā prasību specifikācija palīdz saprast sistēmu pirms tās izstrādes, kā diagrammas un lietošanas gadījumi palīdz pamanīt trūkstošu informāciju un kā precīza prasību formulēšana uzlabo dokumenta kvalitāti. Skolēni secina, ka programmatūras prasību specifikācija ir būtisks posms digitāla risinājuma izstrādē, jo tā palīdz sasaistīt lietotāja vajadzības ar sistēmas funkcionalitāti, atvieglo izstrādi un veido pamatu kvalitatīvam, lietotājam atbilstošam risinājumam.
Lietotāja saskarnes plānošana un dizaina uzmetums Figma	Skolēni apgūst lietotāja saskarnes plānošanas un vienkārša dizaina uzmetuma izveides pamatprincipus, veidojot izpratni par to, kā pirms programmēšanas tiek plānots programmas loga izkārtojums, ievades un izvades elementi un lietotāja darbību secība. Tēmas ietvaros skolēni iepazīstas ar pamata UI elementiem, to nozīmi programmas darbībā un vienkārša maketa izveidi Figma vidē, sasaistot vizuālo plānojumu ar turpmāko Python GUI izstrādi. Aktualizācijas posmā skolēni analizē, kāpēc programmas saskarni ir lietderīgi izplānot pirms koda rakstīšanas, un spriež, kā saskarnes makets palīdz saprast programmas loģiku. Balstoties uz piemēriem no ikdienā lietotām programmām, skolēni nonāk pie secinājuma, ka pārskatāms izkārtojums palīdz labāk saskatīt ievades laukus, darbības pogas un rezultāta attēlošanas vietu, kā arī atvieglo programmas izstrādi un testēšanu. Apjēgšanas posmā skolēni iepazīstas ar lietotāja saskarnes jēdzienu programmēšanā un ar to, kā vizuālie elementi pārvēršas programmas objektos. Skolēni apgūst galvenos UI elementus, piemēram, virsrakstu, ievades lauku, pogu un rezultāta lauku, kā arī sasaista tos ar Python GUI objektiem,

	piemēram, Label, Entry un Button. Tiek veidota izpratne par vienkāršiem saskarnes veidošanas principiem: elementu līdzināšanu, vienādām atstarpēm, loģisku secību un pārskatāmu informācijas izvietojumu. Vienlaikus skolēni iepazīstas ar Figma pamatrīkiem — Frame, Text, Rectangle un līdzināšanas iespējām. Praktiskajā darbībā skolēni soli pa solim izveido vienkāršas Python programmas loga maketu Figma vidē. Sākotnēji skolotājs demonstrē, kā no tukša darba laukuma izveidot programmas loga uzmetumu ar virsrakstu, diviem ievades laukiem, pogu un rezultāta attēlošanas vietu. Pēc tam skolēni patstāvīgi vai pāros izveido savu maketu uzdevumam “Skolēna gala vērtējuma aprēķins”, plānojot ievadi, darbību un rezultātu vienotā logā. Darba gaitā skolēni izvērtē elementu izvietojumu, nodrošina pārskatāmību un, pēc izvēles, papildina maketu ar papildu elementiem, piemēram, pogu “Notīrīt” vai kļūdas paziņojuma lauku. Tādējādi skolēni sasaista dizaina plānojumu ar nākamo programmēšanas posmu. Refleksijas posmā skolēni izvērtē, kā saskarnes makets palīdz saprast programmas uzbūvi un samazina neskaidrības pirms kodēšanas. Skolēni secina, ka jau maketa līmenī iespējams saprast, cik ievades lauki būs nepieciešami, kāda darbība notiks pēc pogas nospiešanas un kur tiks attēlots rezultāts. Tādējādi veidojas pamats apzinātai un strukturētai programmatūras izstrādei, kur lietotāja saskarnes plānošana palīdz sasaistīt vizuālo risinājumu ar programmas loģiku un turpmāko koda izveidi.
Programmatūras projektējums	Skolēni apgūst programmatūras projektējuma izstrādes pamatprincipus, veidojot izpratni par to, kā no iepriekš izstrādātas programmatūras prasību specifikācijas pāriet uz konkrētu risinājuma aprakstu pirms implementācijas. Tēmas ietvaros skolēni iepazīstas ar projektējuma nozīmi programmatūras izstrādes procesā, augsta līmeņa arhitektūras un detalizētā projektējuma jēdzieniem, kā arī nefunkcionālo prasību ietekmi uz sistēmas uzbūvi. Aktualizācijas posmā skolēni analizē atšķirību starp programmatūras prasību specifikāciju un projektējumu, spriežot, ko sistēmai nosaka prasības un ko apraksta projektējums. Balstoties uz piemēriem un iepriekš izstrādāto PPS, skolēni nonāk pie secinājuma, ka projektējums palīdz samazināt kļūdas, neskaidrības un pārbūves riskus, jo pirms programmēšanas ļauj saprast sistēmas komponentes, atbildības, procesus, datus un saistības starp tiem. Apjēgšanas posmā skolēni iepazīstas ar programmatūras projektējuma vietu izstrādes ciklā, projektējuma līmeņiem un arhitektūras pamatprincipiem. Skolēni apgūst, kā no PPS prasībām izriet kvalitātes atribūti, piemēram, drošība, veiktspēja, pieejamība un uzturamība, un kā šie atribūti ietekmē projektēšanas lēmumus. Tiek veidota izpratne par arhitektūras shēmām, komponentu atbildību sadali, sistēmas robežu noteikšanu, kā arī par dažādu diagrammu izmantošanu projektēšanā — aktivitāšu, secības, use case un datu modeļa diagrammām. Vienlaikus skolēni apgūst kļūdu apstrādes un drošības aspektu nozīmi projektējumā. Praktiskajā darbībā skolēni soli pa solim izstrādā izvēlētas sistēmas projektējumu, balstoties uz iepriekš izveidoto programmatūras prasību specifikāciju. Sākotnēji skolēni identificē galvenos kvalitātes atribūtus un pieņem pirmos arhitektūras lēmumus, nosakot sistēmas komponentes, slāņus un to atbildības. Turpinājumā skolēni izvēlas galvenos programmatūras procesus un veido to aprakstus, norādot gan lietotāja darbības, gan programmatūras iekšienē esošos procesus. Noslēgumā skolēni apraksta iespējamās datu aizsardzības risinājumus, sasaista projektējuma elementus ar PPS prasībām, analizē nefunkcionālo prasību ietekmi uz risinājumu un noformē strukturētu projektējuma dokumentu ar diagrammām, atsaucēm un atbildības tabulu. Refleksijas posmā skolēni izvērtē, kā projektējums palīdz pārvērst lietotāja vajadzības un prasības konkrētā, realizējamā risinājumā. Skolēni secina, ka arhitektūras shēmas un diagrammas palīdz pamanīt trūkstošus soļus, neprecīzas prasības un iespējamās problēmas vēl pirms programmēšanas uzsākšanas. Tādējādi veidojas pamats sistemātiskai programmatūras izstrādei, kur projektējums kalpo kā saikne starp prasību specifikāciju un turpmāko implementāciju, nodrošinot pārskatāmu, pamatotu un lietotāja vajadzībām atbilstošu risinājumu.

Programmatūras izstrāde, dzīves cikls, versiju kontrole	Skolēni apgūst programmatūras izstrādes procesa, programmatūras dzīves cikla un versiju kontroles pamatprincipus, veidojot izpratni par to, kā kvalitatīva programmatūra netiek radīta haotiski, bet gan secīgā, pārskatāmā un dokumentētā procesā. Tēmas ietvaros skolēni iepazīstas ar programmatūras izstrādes posmiem, tipiskajiem artefaktiem, komandas lomām, dažādiem dzīves cikla modeļiem un versiju kontroles izmantošanu, sasaistot teorētiskos jēdzienus ar praktisku darbu Git vidē. Aktualizācijas posmā skolēni analizē, kādi riski rodas, ja programmatūru izstrādā bez plāna, dokumentēšanas un vienotas darba kārtības, un spriež, kā strukturēts process palīdz mazināt kļūdas, kavējumus un nesaprašanos komandā. Balstoties uz vienkāršiem piemēriem, skolēni nonāk pie secinājuma, ka programmatūras dzīves cikls palīdz organizēt darbu, skaidri sadalīt atbildības un nodrošināt, ka risinājums tiek ne tikai izveidots, bet arī pārbaudīts, izvietots un uzturēts. Apjēgšanas posmā skolēni iepazīstas ar programmatūras dzīves cikla posmiem — prasību analīzi, projektēšanu, realizāciju, testēšanu, izvietošanu un uzturēšanu — un apgūst tipiskos artefaktus, kas rodas katrā posmā. Skolēni salīdzina dažādus dzīves cikla modeļus, piemēram, Waterfall, V-modeli, iteratīvo un inkrementālo pieeju, kā arī Agile pieeju. Tiek veidota izpratne par to, kā modeļa izvēli ietekmē prasību stabilitāte, riski, termiņi un nepieciešamība pēc biežas atgriezeniskās saites. Vienlaikus skolēni apgūst versiju kontroles jēdzienu, Git pamatprincipus un tipisku darba plūsmu ar repozitoriju, commit, branch, merge, pull request, code review un release. Praktiskajā darbībā skolēni soli pa solim pielieto versiju kontroles principus darbā ar reālu repozitoriju. Sākotnēji skolēni analizē vienkāršas sistēmas piemēru un apraksta, kā tajā izpaustos programmatūras dzīves cikla posmi un kādi riski tiktu mazināti ar strukturētu izstrādes procesu. Pēc tam skolēni praktiski izveido Git repozitoriju, sagatavo projekta struktūru, README un .gitignore, veic pirmos commit, izveido feature branch un apvieno izmaiņas galvenajā zarā. Turpinājumā skolēni pāros strādā ar pull request, veic vienkāršu code review, apzināti veido un risina merge konfliktu, kā arī iezīmē projekta versiju ar tag un sagatavo īsas release piezīmes. Tādējādi skolēni sasaista izstrādes procesu, sadarbību komandā un versiju kontroli vienotā praktiskā pieredzē. Refleksijas posmā skolēni izvērtē, kā programmatūras dzīves cikla izpratne un versiju kontroles izmantošana palīdz padarīt izstrādi pārskatāmu, drošu un uzturamu. Skolēni secina, ka skaidri definēti posmi, atbilstoši izvēlēts izstrādes modelis un pārdomāta darba plūsma Git vidē palīdz saglabāt izmaiņu vēsturi, atvieglot sadarbību, savlaicīgi pamanīt kļūdas un uzturēt kvalitatīvu programmatūras projektu. Tādējādi veidojas pamats profesionālai programmatūras izstrādes praksei, kur svarīga ir ne tikai koda rakstīšana, bet arī procesa organizācija, dokumentēšana un komandas sadarbība.
Programmatūras implementācija, integrācija un testēšana	Skolēni apgūst programmatūras implementācijas, sistēmu integrācijas un testēšanas pamatprincipus, veidojot izpratni par to, kā programmatūras risinājums no projektējuma un prasību specifikācijas pārtop strādājošā, pārbaudītā un lietojamā sistēmā. Tēmas ietvaros skolēni iepazīstas ar implementāciju kā programmas ieviešanu darbībā, kas ietver darba vides sagatavošanu un konfigurāciju, komponentu savienošanu, datu glabātuves sagatavošanu, palaišanu un sākotnējo pārbaudi. Vienlaikus skolēni apgūst sistēmu integrācijas un testēšanas nozīmi programmatūras kvalitātes nodrošināšanā. Aktualizācijas posmā skolēni analizē, kas vēl ir jāpaveic pēc koda uzrakstīšanas, lai sistēma būtu lietojama reālā vidē, un spriež, kādas problēmas var rasties, ja nav sakārtota konfigurācija, datu plūsma vai komponentu sadarbība. Balstoties uz piemēriem, skolēni secina, ka ar koda uzrakstīšanu vien nepietiek, jo programmatūras risinājumam jābūt sagatavotam darbībai konkrētā vidē, savienotam ar citām sistēmas daļām un pārbaudītam pirms nodošanas lietotājam.

	Apjēgšanas posmā skolēni iepazīstas ar implementācijas jēdzienu kā programmatūras ieviešanu, uzstādīšanu, konfigurēšanu un sagatavošanu darbībai dažādās vidēs. Skolēni apgūst atšķirību starp kodēšanu un implementāciju, izprot izstrādes, testēšanas un produkcijas vides nozīmi, kā arī iepazīstas ar konfigurācijas parametriem, atkarībām un minimāli strādājošas sistēmas principu. Vienlaikus skolēni apgūst sistēmu integrācijas pamatus, analizējot datu plūsmu starp lietotāja saskarni, programmas loģiku un datu glabātuvi, kā arī izprot validācijas, drošības un kļūdu apstrādes nozīmi. Tiek veidota izpratne arī par testēšanu implementācijas kontekstā, atšķirot smoke, vienību un integrācijas testus, kā arī definējot, kad sistēmu var uzskatīt par gatavu nodošanai. Praktiskajā darbībā skolēni soli pa solim realizē iepriekš sagatavotajā PPS definētās prasības un pārbauda to darbību praksē. Sākotnēji skolēni sagatavo projekta vidi (lokālu vai testēšanas vidi), konfigurāciju un atkarības, kā arī ievieš vismaz vienu strādājošu funkciju un pārlicinās par programmas darbību. Turpinājumā skolēni izveido vai pieslēdz datu glabātuvi, realizē datu nolasīšanu un saglabāšanu, ievieš validāciju un kļūdu apstrādi, sasaistot rezultātu ar PPS prasībām. Noslēgumā skolēni integrē vairākas funkcijas vienotā plūsmā, izstrādā smoke pārbaudes scenāriju, izveido atbilstības tabulu starp PPS prasībām, realizāciju un pārbaudi, kā arī sakārto projekta dokumentāciju un README. Tādējādi skolēni sasaista implementāciju, integrāciju un testēšanu vienotā, pārskatāmā izstrādes procesā. Refleksijas posmā skolēni izvērtē, kā implementācija, integrācija un testēšana palīdz pārliecināties, ka sistēma darbojas atbilstoši prasībām un ir gatava lietošanai. Skolēni secina, ka kvalitatīva programmatūras ieviešana nozīmē ne tikai funkciju uzrakstīšanu, bet arī korektu darba vides sagatavošanu, stabilu datu plūsmu, kļūdu apstrādi un atkārtojamo pārbaudi. Tādējādi veidojas pamats sistemātiskai programmatūras izstrādes praksei, kur katra realizētā prasība tiek ne tikai ieviesta kodā, bet arī integrēta, pārbaudīta un dokumentēta atbilstoši programmatūras prasību specifikācijai.
--	---