Autoprogramuotojo maketo reikalavimų analizė

2025-04-25



# Turinys

[Turinys 2](#_Toc198217472)

[Santrauka 3](#_Toc198217473)

[Įvadas 3](#_Toc198217474)

[Pagrindiniai naudotojų scenarijai (workflows) 5](#_Toc198217475)

[Funkciniai reikalavimai 8](#_Toc198217476)

[Funkciniai reikalavimai pagal darbo eigą 9](#_Toc198217477)

[Funkciniai reikalavimai pagal užduočių tipus 10](#_Toc198217478)

[Nefunkciniai reikalavimai 14](#_Toc198217479)

[Techniniai komponentai 16](#_Toc198217480)

[MVP apimtis 19](#_Toc198217481)

[Roadmap: tolesnės versijos 21](#_Toc198217482)

[Rizikos 23](#_Toc198217483)

# Santrauka

Ši reikalavimų analizė skirta AdoreIno platformos **1.3 modulio – autoprogramuotojo** – techninės užduoties suformavimui. Tai vienas kertinių platformos komponentų, skirtas automatizuoti smulkių IT palaikymo užduočių įgyvendinimą be programuotojo įsitraukimo, išlaikant sprendimo kokybę, kontrolę ir atsekamumą.

Remiantis išsamia rinkos problemą apibendrinančia analize, identifikuota, kad verslai ir IT komandos dažnai susiduria su šiomis kliūtimis:

* smulkių užduočių įgyvendinimo vėlavimu,
* perteklinėmis palaikymo sąnaudomis,
* programuotojų apkrova žemos vertės darbais,
* negebėjimu reaguoti į kritinius incidentus greitai ir struktūruotai,
* neefektyviu esamų sistemų palaikymu.

Autoprogramuotojas veikia kaip dirbtiniu intelektu grįstas automatinis vykdytojas, gebantis suprasti naudotojo užduotį (per chatbot ar vedlį), sugeneruoti atitinkamą programinį kodą, atlikti testavimą, pateikti peržiūrai ir, patvirtinus, įdiegti pokytį į sistemą. Tai nėra analizės įrankis – tai sprendimo įgyvendinimo modulis, kuris perkelia veiksmą iš „žmogaus pagalbos“ į „valdomą automatiką“.

Ši ataskaita:

* apibrėžia pagrindinius naudotojų scenarijus ir poreikius;
* išskiria funkcinius ir nefunkcinius reikalavimus;
* nubrėžia architektūrinius ir technologinius komponentus;
* įvardija MVP apimtį ir plėtros etapus;
* įvertina produkto rizikas ir jų valdymo logiką.

Tikslas – užtikrinti, kad 1.3 modulis būtų ne tik technologiškai įmanomas, bet ir realiai pritaikomas, kuriantis aiškią, apčiuopiamą vertę verslo naudotojams bei IT komandoms. Tai sprendimas, kuris padeda judėti greičiau, dirbti efektyviau ir mažinti žmogiškųjų resursų priklausomybę palaikymo procesuose.

# Įvadas

**1.1 Reikalavimų analizės paskirtis**

Šio dokumento tikslas – suformuluoti aiškius ir struktūruotus reikalavimus **AdoreIno** platformos **1.3 moduliui – autoprogramuotojui**. Tai funkcija, veikianti kaip dirbtiniu intelektu (DI) pagrįstas automatinis programuotojas, skirtas IT sistemų palaikymo užduočių įgyvendinimui. Ši analizė remiasi rinkos tyrimo išvadomis (nuo „Problemos apibendrinimas“ iki „Naudotojų analizė“) bei produkto vystymo strateginiais principais.

Autoprogramuotojo modulis nėra analizės ar sprendimų rekomendavimo sistema. Jis **veikia vykdomajame lygmenyje** – kai užduotis jau aiški, ji turi būti įgyvendinta efektyviai, automatiškai ir su žmogaus priežiūra tik esant poreikiui.

Ši specifikacija yra skirta:

* techninei komandai (programuotojams, AI inžinieriams),
* produkto vadovams,
* sistemų architektams ir integruotojams.

Dokumente nurodyti **funkciniai**, **nefunkciniai** bei **techniniai reikalavimai**, pagrįsti realiomis naudotojų problemomis, veiklos scenarijais ir siekiamu produkto vertės pasiūlymu.

**1.2 Produkto kontekstas**

AdoreIno platforma kuria integruotą sistemą IT palaikymo automatizavimui. Autoprogramuotojas yra vienas iš trijų pagrindinių modulių greta kodo ir turinio analizės, tačiau jo veikimo esmė – **techninių pakeitimų vykdymas kodu**, kai užduotis jau yra aiškiai suformuluota. Šis modulis:

* automatizuoja paprastus ir pasikartojančius techninius veiksmus,
* reaguoja į palaikymo užduotis greičiau nei klasikinis IT žmogus,
* suteikia programuotojams galimybę atsijungti nuo smulkių pataisymų ir fokusuotis į svarbiausius darbus.

Šio produkto naudotojai – IT tiekėjai, laisvai samdomi programuotojai, agentūros, įmonių IT komandos, taip pat verslo naudotojai, kuriems svarbu operatyviai inicijuoti nedidelius pakeitimus savo sistemose (pvz., formų korekcija, logikos atnaujinimas, tekstinių laukų valdymas).

**1.3 Tikslas**

Autoprogramuotojo modulio tikslas – sukurti dirbtiniu intelektu grįstą vykdomąją funkciją, kuri leistų automatizuotai įgyvendinti technines užduotis IT sistemų palaikymo kontekste. Modulis turi gebėti atlikti konkrečius, struktūrizuotai suformuluotus veiksmus – nuo kodo generavimo iki testavimo ir paleidimo – be tiesioginio žmogaus įsitraukimo, bet užtikrinant kontrolės ir patvirtinimo galimybę.

Funkcija turi būti skirta tiek paprastoms, tiek sudėtingoms sistemoms, tačiau ypatingas dėmesys skiriamas **paprastų IT sistemų palaikymui**, kur dažnos smulkios užduotys (pvz., formų korekcijos, tekstų logikos pakeitimai, UI elementų tvarkymas) šiuo metu vykdomos neefektyviai, per brangiai arba visai neatliekamos dėl IT resursų trūkumo.

Pagrindiniai tikslai:

* Automatizuoti smulkių programinių užduočių vykdymą palaikymo etape;
* Sumažinti žmogaus darbo poreikį ir programuotojų blaškymą;
* Užtikrinti greitą, ekonomiškai efektyvų ir saugų užduočių įgyvendinimą;
* **Suteikti automatinį techninį palaikymą paprastoms IT sistemoms**, kurioms trūksta specializuoto IT personalo ar palaikymo biudžeto.

Autoprogramuotojas turi veikti kaip savarankiškas vykdytojas – priimantis struktūrizuotą užduotį, generuojantis kodą, testuojantis, pateikiantis rezultatą peržiūrai ir, jei reikia, atliekantis pataisymus ar grąžinimą ankstesnei versijai.

**1.4 Ryšys su kitais moduliais**

Autoprogramuotojas glaudžiai susijęs su kitais AdoreIno platformos komponentais:

* **1.1 Kodo analitikas** – nustato problemos lokaciją ir techninį kontekstą.
* **1.2 Turinio analitikas** – įvardija, kur sistemoje trūksta turinio ar reikia atnaujinimo.
* **1.4 Turinio autokūrėjas** – generuoja tekstinius komponentus.

Autoprogramuotojas yra **vykdomasis branduolys**, kuris apjungia supratimą (iš kitų modulių) su realiu veiksmu – **kodo generacija, peržiūra, testavimu ir paleidimu**.

**1.5 Integracija su visa sistema**

Kad autoprogramuotojo funkcionalumas būtų realiai naudojamas praktiniuose IT palaikymo procesuose, būtina sklandi integracija su kitais platformos sluoksniais ir išorinėmis sistemomis:

* **Užduočių valdymo sistema** (pvz., Jira, ServiceNow): užduoties statusų keitimas, istorija, komentavimas.
* **Versijų valdymo įrankiai** (pvz., Git): automatinė šakų generacija, commit logika, „merge request“ integracija.
* **CI/CD pipeline**: galimybė paleisti testus ir diegimą automatizuotai pagal patvirtintą kodą.
* **Palaikomų sistemų API** (WordPress, Laravel, Node.js, kt.): automatinio diegimo ar failų rašymo galimybė.

Autoprogramuotojas turi veikti kaip **API-first komponentas**, kuris gali būti paleidžiamas UI sąsajoje (per vedlį ar chatbotą) arba kaip paslauga integracijoje su organizacijos IT infrastruktūra.

# Pagrindiniai naudotojų scenarijai (workflows)

Autoprogramuotojas – tai vykdomasis produktas, skirtas įgyvendinti jau apibrėžtas technines užduotis programiniu būdu, automatizuojant smulkių ir pasikartojančių darbų vykdymą IT sistemų palaikymo etape. Skirtingai nuo analizės ar planavimo modulių, autoprogramuotojas orientuotas į „veiksmą“ – realų, greitą ir standartizuotą kodo modifikavimą.

Šiame skyriuje aprašyti svarbiausi darbo scenarijai, kylantys iš rinkos tyrimo metu identifikuotų problemų, tokių kaip: užduočių įgyvendinimo delsimas, programuotojo neprieinamumas, aukštos palaikymo sąnaudos, per dažnas konteksto keitimas, žemos vertės veiklų blaškymas ar augimui netinkama struktūra.

**2.1 Greitas smulkaus funkcionalumo įgyvendinimas**

**Tikslas:** vartotojas nori atlikti smulkų sistemos pakeitimą, kurio techninė užduotis yra aiški (pvz., pakeisti tekstą, pridėti pasirinkimą, koreguoti formos lauką).

**Darbo eiga:**

1. Naudotojas perveda užduotį į autoprogramuotojo vedlį arba chatbot'ą.
2. Sistema klausia tikslių detalių (kuri forma, koks tekstas, kokios sąlygos).
3. Generuojamas kodas, kuris:
   * peržiūrimas su paaiškinimu,
   * gali būti koreguojamas arba iš karto taikomas.
4. Užduotis atliekama automatiškai, o naudotojas gauna ataskaitą ir rollback parinktį.

**Vertė:**

* Užduotis įgyvendinama per kelias minutes, o ne per dienas.
* Neapkraunamas programuotojas.
* Sumažėja smulkių pakeitimų sąnaudos.

**2.2 Automatinis atsakas į klientų palaikymo užklausas**

**Tikslas:** IT tiekėjas arba agentūra gauna kliento užklausą dėl nedidelio, bet konkretaus pataisymo (pvz., „neveikia forma“, „neatsidaro popup“), kurį reikia įgyvendinti nedelsiant.

**Darbo eiga:**

1. Užklausa importuojama iš bilietų sistemos arba įvedama rankiniu būdu.
2. Autoprogramuotojas analizuoja kontekstą ir parengia veiksmų planą.
3. Sistema sugeneruoja pataisymo kodą ir jį pritaiko testavimo aplinkoje.
4. Projektų vadovas peržiūri ir tvirtina (arba atmeta), tada pokytis paleidžiamas.

**Vertė:**

* Greita reakcija į klientą.
* Apsauga nuo „pavėluoto taisymo“ reputacinės žalos.
* Aiškus sprendimo pagrindimas be žmogiškos interpretacijos klaidų.

**2.3 Paruošimas diegimui ar „preview“ režimo įgyvendinimas**

**Tikslas:** prieš paleidžiant realų pakeitimą, verslo atstovas arba testuotojas nori peržiūrėti kaip atrodys pokytis.

**Darbo eiga:**

1. Naudotojas įveda pakeitimo poreikį (pvz., „šis laukas turi būti matomas tik darbo dienomis“).
2. Autoprogramuotojas generuoja kodą ir per „preview“ režimą rodo rezultatą su paaiškinimu.
3. Naudotojas peržiūri, patvirtina ir (jei reikia) pakoreguoja.
4. Po patvirtinimo – kodas įkeliamas į LIVE aplinką.

**Vertė:**

* Mažėja netikslumų.
* Užtikrinamas vizualinis ir loginis kokybės lygis.
* Vartotojas įgyja pasitikėjimą ir sprendimų kontrolę.

**2.4 Augančios IT agentūros darbo išskaidymas**

**Tikslas:** agentūra arba laisvai samdomas programuotojas aptarnauja daug smulkių klientų ir neturi pajėgumų įgyvendinti kiekvieno mikro pakeitimo rankiniu būdu.

**Darbo eiga:**

1. Užduotys surenkamos į eilę (pvz., per klientų portalą ar CRM integraciją).
2. Autoprogramuotojas analizuoja jas partiniu būdu ir generuoja kodo keitimus.
3. Kiekviena užduotis:
   * turi priskirtą rizikos klasę,
   * gali būti tvirtinama masiškai arba atskirai.
4. Agentūra deleguoja tik patvirtinimo žingsnius žmogui, visa kita – automatiškai.

**Vertė:**

* Palaikymo paslauga tampa mastelį didinanti.
* Mažėja perdegimas ir prastovos.
* Galima aptarnauti daugiau klientų be papildomo personalo.

**2.5 Reguliarūs svetainės palaikymo darbai**

**Tikslas:** atlikti dažnai pasikartojančius palaikymo veiksmus (pvz., savaitinis datos atnaujinimas, sezono banerio įjungimas) automatiškai.

**Darbo eiga:**

1. Užduotis iš anksto suplanuojama (pvz., „kas pirmadienį įkelk X“).
2. Autoprogramuotojas pagal grafiką:
   * sugeneruoja naują kodą,
   * pakeičia atitinkamą elementą sistemoje,
   * atlieka testavimą.
3. Automatiškai pateikiama ataskaita ar nuoroda peržiūrai.

**Vertė:**

* Sutaupoma laiko.
* Užduotys nepasimiršta.
* Net paprasti palaikymo darbai tampa struktūrizuoti ir patikimi.

**2.6 Bugo ar kritinio defekto šalinimas**

**Tikslas:** nedelsiant pašalinti klaidą (pvz., sugriuvęs šriftas, dingęs komponentas), kai programuotojas nepasiekiamas arba sistema neleidžia laukti.

**Darbo eiga:**

1. Vartotojas chatbot’ui aprašo problemą („po paskutinio atnaujinimo mygtukas pasislinko“).
2. Autoprogramuotojas analizuoja versijų skirtumus, struktūrą ir galimas priežastis.
3. Generuojamas kodo pataisymas, įkeliama į testavimo aplinką.
4. Jei numatyta – automatiškai taikoma LIVE, jei ne – laukia tvirtinimo.

**Vertė:**

* Sugrįžtama į veikiančią būseną per kelias minutes.
* Nebereikia laukti programuotojo.
* Sumažinama kritinių incidentų trukmė ir poveikis.

**Išvada:**  
Autoprogramuotojo scenarijai atspindi poreikį veikti greitai, ekonomiškai ir be žmogaus išteklių priklausomybės smulkiuose IT palaikymo procesuose. Modulis turi būti **ne sprendimų analitikas**, o **techninis vykdytojas**, gebantis reaguoti į sistemų poreikius – tiek planuotus, tiek netikėtus – automatiškai ir saugiai.

# Funkciniai reikalavimai

Autoprogramuotojo funkcionalumas turi apimti visą nedidelių techninių užduočių įgyvendinimo ciklą – nuo užklausos inicijavimo iki kodo generavimo, testavimo, paleidimo ir peržiūros. Modulis turi būti suprojektuotas kaip dirbtiniu intelektu pagrįstas vykdytojas, kurio rezultatai yra kontroliuojami, atsekami ir pritaikomi tiek paprastoms, tiek sudėtingoms sistemoms.

Funkciniai reikalavimai skirstomi į dvi grupes:

* **3.1 Pagal darbo eigą** – apima veiksmų seką nuo užduoties suformulavimo iki jos įvykdymo.
* **3.2 Pagal užduočių tipus** – detalizuoja reikalavimus konkretiems palaikymo atvejams (formos, tekstas, sąlyginė logika, stilistika ir pan.).

## Funkciniai reikalavimai pagal darbo eigą

| **Etapas** | **Reikalavimai** |
| --- | --- |

**1. Užklausos suformulavimas**

* Naudotojas turi galėti aprašyti užduotį per struktūrizuotą formą arba natūralia kalba per chatbot'ą.
* Sistema turi gebėti užduoti papildomus klausimus, kad tiksliai išsiaiškintų užduoties esmę.
* Užduotis gali būti importuojama iš išorinės sistemos (pvz., bilietų sistema ar Jira).

**2. Užduoties interpretavimas**

* DI turi interpretuoti užduotį ir paversti ją į programavimo veiksmų planą (pavyzdžiui: „redaguoti CSS failą X“, „pridėti lauką formoje Y“).
* Naudotojui turi būti pateiktas aiškus veiksmų sąrašas su priežastimis (angl. rationale).

**3. Kodo generavimas**

* Sistema turi sugeneruoti programavimo kodą pagal užduotį.
* Kodo blokai turi būti:  
  – paruošti įkėlimui,  
  – su paaiškinimais,  
  – atskirti į loginį vienetą (modulį, funkciją ar klasę).
* Generavimas turi palaikyti dažniausiai naudojamas kalbas ir sistemų tipus (pvz., PHP, JS, HTML, CSS, JSON, SQL, React/Blade/Smarty/Liquid ir pan.).

**4. Peržiūra ir tvirtinimas**

* Kodo generavimo rezultatą vartotojas gali peržiūrėti, koreguoti arba atmesti.
* Sistema turi siūlyti „preview“ režimą (prieš įrašant į LIVE).
* Tvirtinimo procesas turi būti valdomas:  
  – rankiniu būdu,  
  – automatiškai pagal saugumo parametrus,  
  – mišriu režimu.

**5. Testavimas**

* Sistema turi pasiūlyti automatinį testavimą:  
  – UI testai (pvz., ar rodomas komponentas),  
  – logikos testai (pvz., ar sąlygos veikia),  
  – HTML/CSS validacija.
* Testavimo rezultatai pateikiami naudotojui su rekomendacijomis.

**6. Diegimas ir atstatymas (rollback)**

* Jei tvirtinimas įvykdytas – kodas turi būti įdiegtas į tikslinę sistemą (per API arba copy-paste formatu).
* Privaloma rollback funkcija – galimybė grįžti į ankstesnę versiją vienu paspaudimu.
* Visi veiksmai turi būti atsekami (log’ai, kas, kada ir ką padarė).

**7. Užduočių valdymas**

* Sistema turi leisti:  
  – matyti visas atliktas ir suplanuotas užduotis,  
  – sekti statusą: „sukurta“, „peržiūrima“, „įdiegta“, „atmesta“,  
  – filtruoti užduotis pagal sistemą, naudotoją, tipą, datą.
* Galimybė deleguoti užduoties patvirtinimą kitam asmeniui (projekto vadovui ar klientui).

## Funkciniai reikalavimai pagal užduočių tipus

Autoprogramuotojas turi būti pritaikytas dažniausiai pasitaikančių palaikymo užduočių automatizavimui, kurie IT palaikymo realybėje sudaro didžiąją dalį pasikartojančių darbų. Kiekvienam užduočių tipui turi būti taikomi specifiniai funkcionalumai, testavimo logika ir saugos mechanizmai.

**3.2.1 UI ir tekstinių elementų korekcijos**

**Užduočių tipai:**

* Teksto pakeitimas (pvz., mygtuko pavadinimas, kontaktinė informacija).
* Blokų parodymas/slėpimas tam tikromis sąlygomis.
* Elemento pozicijos ar dydžio keitimas (pvz., išplėsti mygtuką).

**Funkcionalumas:**

* Identifikuoti tikslinį UI komponentą.
* Sugeneruoti HTML/CSS ar JS pakeitimus.
* Tikrinti, ar komponentas egzistuoja DOM’e.
* Tikrinti atsakymą mobiliuose ir desktop vaizduose (jei įmanoma).

**3.2.2 Formų ir laukų valdymas**

**Užduočių tipai:**

* Lauko pridėjimas / pašalinimas formoje.
* Validacijos logikos korekcijos (pvz., privalomas/nebūtinas).
* Sąlyginių laukų valdymas (pvz., „rodyti lauką tik jei pasirinkta X“).

**Funkcionalumas:**

* Sugeneruoti HTML struktūrą + backend duomenų priėmimo logiką.
* Atnaujinti JS validacijos skriptus.
* Ištestuoti ar forma sėkmingai siunčia duomenis.

**3.2.3 Stiliaus ir išvaizdos pakeitimai**

**Užduočių tipai:**

* Spalvų, šriftų, paraščių keitimas.
* Responsyvumo problemų taisymas (pvz., išlendantis tekstas).
* Vizualinio neatitikimo korekcijos.

**Funkcionalumas:**

* Pakeisti CSS/SCSS failus arba inline stilių.
* Sulyginti su esama dizaino sistema (jei integruota).
* Tikrinti rezultatus per preview renderį su screenshot palyginimu.

**3.2.4 Sąlyginių loginės logikos atnaujinimai**

**Užduočių tipai:**

* Rodyti/paslėpti funkciją pagal laiką, rolę ar įvykį.
* Keisti funkcionalumo aktyvavimo logiką (pvz., nuo 100 € taikyti papildomą veiksmą).
* Įdiegti „toggle“ tipo sprendimą.

**Funkcionalumas:**

* Generuoti loginį kodą JS/PHP/Node pagal naudojamą sistemą.
* Tikrinti, ar sąlygos veikia pagal pateiktą logiką (unit test pagrindu).
* Pateikti naudotojui pseudo „jeigu... tada...“ patvirtinimą.

**3.2.5 Backend ir duomenų struktūros modifikacijos (paprasto lygmens)**

**Užduočių tipai:**

* Duomenų modelio išplėtimas (pvz., pridėti lauką į duomenų lentelę).
* Pakeisti reikšmės logiką (pvz., statusas keičiamas pagal naują sąlygą).
* Sukurti/atnaujinti webhook ar API endpoint‘ą.

**Funkcionalumas:**

* Generuoti migracijas (jei sistema palaiko ORM).
* Atlikti schema validation testus.
* Pažymėti, kuriose vietose kodas liečia kitus modulius (priklausomybių įspėjimai).

**3.2.6 Laikini arba suplanuoti veiksmai**

**Užduočių tipai:**

* Įjungti tam tikrą funkciją tam tikram laikui.
* Atnaujinti komponentą tik konkrečiu laikotarpiu.
* Sezoniniai / reklaminiai pakeitimai.

**Funkcionalumas:**

* Palaikyti darbo planuoklį (angl. scheduler).
* Leisti iš anksto suplanuoti įvykdymo laiką.
* Užtikrinti automatinį „grįžimą“ į ankstesnę būseną po galiojimo.

**3.2.7 Techninės klaidos (bug) taisymas**

**Užduočių tipai:**

* Sugedęs šriftas, pertraukta funkcija, dingo elementas po atnaujinimo.
* Klaidingas mygtuko veiksmas.
* Problemos su renderinimu ar naršyklių suderinamumu.

**Funkcionalumas:**

* Analizuoti DOM/CSS/JS pokyčius tarp versijų.
* Sugeneruoti pataisą su rollback parinktimi.
* Pateikti sprendimą testavimo aplinkoje ir palyginimo rezultatą (prieš/po).

**3.2.8 Automatinis egzistuojančių svetainių palaikymas**

**Tikslas:** užtikrinti nuolatinį esamų (eksploatuojamų) sistemų palaikymą, atliekant pasikartojančias ar būtinas technines priežiūros užduotis automatiškai.

**Būdingi palaikymo darbai:**

* Taisyklingo turinio atvaizdavimo tikrinimas po atnaujinimų (pvz., ar nesugriuvo puslapio išdėstymas).
* Tinklalapio greičio parametrų ar SEO komponentų stebėjimas.
* Formų, mygtukų ar kitų UI elementų pasiekiamumo testavimas.
* Trečiųjų šalių integracijų (pvz., Stripe, Mailchimp, Google Maps) veikimo patikrinimas.
* Laikini arba sezoniškai kintantys komponentai (pvz., rodyti kalėdinę skiltį gruodį).
* Senų turinio blokų paslėpimas arba struktūrinis pertvarkymas (pvz., „naujienos 2022“ → archyvas).

**Funkcionalumas:**

* Periodinis automatinis sistemos veikimo stebėjimas (ar komponentai rodomi, veikia, nėra klaidų).
* Išankstinių šablonų palaikymas dažniems veiksmams (pvz., „pridėti sezono bloką“, „paslėpti akcijos banerį“).
* Aptikus trikdžius – automatinis siūlomos pataisos generavimas.
* Priežiūros log‘o pateikimas: kas atlikta, kada, ar buvo pokyčių.
* Galimybė konfigūruoti planuojamą palaikymo veiksmų grafiką (cron tipo funkcija).

**Vertė:**

* Sistemos išlieka „šviežios“ be žmogaus rankos.
* IT palaikymas pereina iš reaktyvaus į proaktyvų modelį.
* Minimalizuojama verslo priklausomybė nuo programuotojų smulkiems procesams.

**Išvada:**  
Autoprogramuotojas turi turėti „žinių biblioteką“ apie dažniausiai pasitaikančius užduočių tipus IT palaikymo procese ir mokėti automatiškai sugeneruoti jų įgyvendinimo kodą bei veiksmų seką. Tai nėra generatyvinė improvizacija – tai yra kontrolės, tikslumo ir greičio balansas, pagrįstas aiškiomis taisyklėmis ir rizikų valdymu.

# Nefunkciniai reikalavimai

Autoprogramuotojas – tai ne tik funkcijų rinkinys, bet ir įrankis, kuriuo naudotojai turi pasitikėti vykdydami techninius veiksmus. Kad sprendimas būtų realiai naudojamas tiek techninių, tiek netechninių naudotojų, būtina užtikrinti aukštus nefunkcinius kokybės, aiškumo, patikimumo ir sąveikos reikalavimus.

**4.1 Patikimumas ir saugumas**

* Kiekvienas generuojamas sprendimas turi būti **technologiškai saugus**, išlaikant sistemų stabilumą ir nelaužant kodo priklausomybių.
* Privaloma palaikyti **rollback (grąžinimo)** galimybę – kiekvienas įvykdytas pakeitimas turi turėti atstatymo variantą.
* Sistema negali įdiegti nepalaikomo ar pavojingo kodo be naudotojo įsikišimo.
* Privaloma **rizikos klasifikacija**: prieš diegiant pokytį turi būti įvertinta, ar veiksmas nepalies kritinių komponentų.

**4.2 Aiškumas ir suprantamumas**

* Naudotojas turi **matyti, ką sistema ketina padaryti**, dar prieš veiksmą – su galimybe peržiūrėti kodą, sprendimo logiką ir numatomą rezultatą.
* Kiekvienas sugeneruotas veiksmas turi turėti **paaiškinimą natūralia kalba** (pvz., „šis kodas pakeičia mygtuko tekstą tik darbo dienomis“).
* Sistema turi teikti suprantamus statusus, pvz.: „pakeitimas atliktas“, „pakeitimas laukia patvirtinimo“, „pakeitimas atmestas“.

**4.3 Vartotojo patirties paprastumas**

* Net ne techninis naudotojas (pvz., verslo savininkas) turi gebėti pateikti užduotį ir suprasti rezultatą.
* Chatbot’as ar vedlys turi **veikti kaip žmogaus kalbos vertėjas į techninį veiksmą** – užduodamas tikslius, bet neperkrautus klausimus.
* Naudotojas turi visada jaustis saugiai – kad sprendimas nebus įdiegtas be jo žinios ar galimybės atšaukti.

**4.4 Automatizacijos balansas su kontrole**

* Sistema turi automatiškai generuoti sprendimą, bet naudotojas privalo išlaikyti **kontrolės taškus**:  
  – ar vykdyti dabar,  
  – ar siųsti peržiūrai,  
  – ar atidėti.
* Galimi **automatizavimo lygiai**:  
  – tik generacija,  
  – generacija + diegimas į preview,  
  – pilnas automatizavimas tik paprastoms užduotims (jei įjungtas).
* Sistema turi palaikyti „saugų režimą“ – tik demonstracinis vykdymas be poveikio gyvai sistemai.

**4.5 Veikimo greitis ir reagavimo laikas**

* Užduoties interpretavimas, sprendimo plano generacija ir kodo pasiūlymas turi būti atlikti per **< 10 sek.** standartinėms užduotims.
* Diegimo procesas (preview ar live) – per **< 1 min.** (atsižvelgiant į išorinės sistemos API laiką).
* Sistemos apkrovos metu sprendimas turi informuoti apie numatomą vėlavimą.

**4.6 Integruojamumas**

* Visi funkcionalumai turi būti **API-first**, kad būtų galima naudoti:  
  – per UI (chatbot ar formą),  
  – per trečiųjų šalių sistemas (pvz., Jira, klientų portalą),  
  – per automatizavimo procesus (pvz., webhook’ai, cron job’ai).
* Kiekvienas sprendimas turi turėti **aiškų įvykdymo logą** (kas, kada, kokioje sistemoje ir su kokiu kodu įvykdyta).

**4.7 Auditas ir atsekamumas**

* Visi pokyčiai turi būti **fiksuojami su kontekstu**: kas inicijavo, ką pakeitė, koks buvo ankstesnis variantas.
* Kiekvienas įvykdytas sprendimas turi turėti **istoriją su difu** („kas pasikeitė“).
* Galimybė eksportuoti veiksmų žurnalą (audit trail) į PDF / JSON / CSV formatu.

**Išvada:**  
Autoprogramuotojas turi būti ne tik funkcionalus, bet ir pasitikėjimą keliantis produktas. Sistemos veikimas privalo būti **aiškus, atsekamas ir valdomas**, net kai užduotys vykdomos be tiesioginio žmogaus įsitraukimo. Tik tokios sąlygos leidžia naudotojams realiai deleguoti techninius veiksmus DI sistemai.

# Techniniai komponentai

Autoprogramuotojo modulis turi būti suprojektuotas kaip **API-first, modulinis, lengvai integruojamas komponentas**, gebantis valdyti užduotis nuo natūralios kalbos interpretavimo iki kodo generavimo, testavimo, vykdymo ir atsekimo. Sistemai būtinas lankstumas dirbti su skirtingomis programavimo kalbomis, sistemų architektūromis ir prieigos modeliais.

Pagrindinės komponentų grupės:

**5.1 Užduočių interpretavimo sluoksnis (chatbot/vedlys)**

**Paskirtis:** suprasti naudotojo pateiktą užklausą (natūralia kalba arba struktūruotu būdu) ir paversti ją į techninę specifikaciją.

**Komponento funkcijos:**

* Chatbot sąsaja (tekstinė ir (ar) GUI forma).
* Struktūrinis klausimų-vietų medis pagal užduočių tipus.
* Užduočių tipų atpažinimas pagal semantinę analizę.
* Užduoties validacija: ar galima vykdyti automatizuotai, ar reikalingas žmogaus įsikišimas.

**5.2 Kodavimo ir generavimo variklis (LLM pagrindu)**

**Paskirtis:** sugeneruoti programinį kodą pagal užduoties specifikaciją.

**Technologinis pagrindas:**

* LLM (pvz., OpenAI GPT-4, Claude, Mistral ar kitas pasirinktas modelis).
* Promptų tvarkyklė (prompt templates, tonas, kodavimo kalba, sistema).
* Kodo post-processing modulis:  
  – įtraukia komentarus,  
  – optimizuoja formatą,  
  – tikrina kodo sintezę.

**Kalbų palaikymas (MVP stadijoje):**

* HTML, CSS, JS, PHP (WordPress), Smarty, Liquid, Laravel Blade, Node.js, React.

**5.3 Testavimo ir validacijos sluoksnis**

**Paskirtis:** įsitikinti, kad sugeneruotas kodas veikia ir nekeičia kitų sisteminių komponentų.

**Funkcionalumas:**

* Kodo sintaksės validacija (linting, format checking).
* UI renderio testai (pvz., ar komponentas rodomas puslapyje).
* Loginiai testai (pvz., if/else logikos testavimas).
* Automatizuoti HTML/CSS struktūros ir vizualinio rezultato testai.
* Testų režimai: preview, staging, live.

**5.4 Diegimo mechanizmas (deployment handler)**

**Paskirtis:** valdyti, kaip ir kur bus įrašomas/taikomas sugeneruotas kodas.

**Režimai:**

* Diegimas į failų sistemą (per SFTP, Git ar kitą integraciją).
* API pagrindu (pvz., WordPress REST API, webhook’ai, Node.js endpoint’ai).
* Manual export (jei įdiegia naudotojas).

**Papildomos funkcijos:**

* Rollback valdymas (versijų istorija).
* „Preview first“ politika – viskas pirmiausia matoma testinėje aplinkoje.
* Deployment statusų logika („įkelta“, „patvirtinta“, „atmesta“, „grąžinta korekcijai“).

**5.5 Užduočių valdymo ir veiksmų istorijos komponentas**

**Paskirtis:** užtikrinti atsekamumą, leidimą deleguoti, ir sprendimų kokybės stebėseną.

**Funkcijos:**

* Užduoties žurnalas (kas, kada, ką padarė).
* Veiksmų istorija su dif’ais („kas pasikeitė“).
* Komentarų ir patvirtinimų logika.
* Naudotojų rolės: iniciatorius, tikrintojas, tvirtintojas.
* Eksportas į PDF/CSV/JSON.

**5.6 Integracijų API sluoksnis**

**Paskirtis:** leisti autoprogramuotoją naudoti per išorines sistemas ir/arba įmonės infrastruktūrą.

**Palaikomos integracijos (MVP stadijoje):**

* Bilietų sistemos (pvz., Jira, ClickUp) – užduočių importas/eksportas.
* CMS (pvz., WordPress, PrestaShop) – per REST API ar HTML blokų eksportą.
* Git versijų valdymas – automatinis branch kūrimas, commit’as su komentaru.
* CI/CD webhook’ai – paleidimas pagal įvykį ar užklausą.

**5.7 Saugumo ir leidimų valdymas**

**Paskirtis:** užtikrinti, kad tik tinkami naudotojai galėtų inicijuoti ar vykdyti veiksmus.

**Funkcionalumas:**

* Rolės ir leidimų sistema.
* Apsauga nuo žalingo kodo generavimo (LLM saugos filtrai).
* Išorinė autentifikacija (OAuth2, SSO).
* Duomenų srauto šifravimas, audit log’ai.

**Išvada:**  
Autoprogramuotojo architektūra turi būti grindžiama **atvirumo, kontrolės ir lankstumo principais**: naudotojas turi gebėti pasinaudoti sprendimu tiek per AI sąsają, tiek per technines integracijas, o kiekvienas automatiškai įvykdytas veiksmas turi būti atsekamas, validuotas ir – jei reikia – anuliuojamas.

# MVP apimtis

Minimalus veikiantis produktas (MVP) turi leisti naudotojui per paprastą sąsają inicijuoti IT palaikymo užduotį, gauti automatiškai sugeneruotą kodą, jį peržiūrėti, patvirtinti ir įgyvendinti sistemoje (per eksportą arba API). MVP versija turi būti orientuota į **paprastų sistemų** (WordPress, PrestaShop, Laravel, HTML svetainių) palaikymo automatizavimą.

**6.1 Įtraukiami komponentai MVP etape**

|  |  |
| --- | --- |
| **Komponentas** | **Apimtis MVP versijoje** |
| **Užduoties suformulavimas** | Chatbot/vedlys leidžia įvesti užduotį natūralia kalba arba per struktūrizuotą formą. |
| **Užduoties interpretacija** | DI atpažįsta tipinį veiksmą (pvz., teksto pakeitimas, lauko pridėjimas) ir pateikia veiksmų planą. |
| **Kodo generavimas** | GPT-4 arba analogiškas modelis generuoja kodą HTML, CSS, PHP, JS formatais. |
| **Peržiūra ir patvirtinimas** | Naudotojas gali peržiūrėti kodo rezultatą, matyti paaiškinimą ir pasirinkti „vykdyti“ / „atmesti“. |
| **Diegimo mechanizmas** | Eksportas copy–paste formatu (HTML, CSS, PHP); galimybė siųsti el. paštu arba įkelti į CMS rankiniu būdu. |
| **Rollback funkcija** | Kiekvienas pakeitimas išsaugomas; galimybė atšaukti pakeitimą rankiniu būdu. |
| **Užduočių sąrašas** | Vartotojas mato visų atliktų veiksmų žurnalą: statusas, data, kodas, naudotojas. |
| **Palaikomi užduočių tipai** | Teksto keitimas, formos lauko pridėjimas/pašalinimas, UI klaidų taisymas, CSS stiliaus keitimas, loginės sąlygos (pvz., „rodyti tik savaitgaliais“). |
| **Papildomi įrankiai** | Galimybė pridėti komentarą prie užduoties, matyti paskutinį veiksmą, pažymėti kaip „atlikta“ arba „neįgyvendinta“. |

**6.2 Neįtraukiami į MVP, bet numatomi vėliau**

|  |  |
| --- | --- |
| **Komponentas** | **Pastaba** |
| Integracija su bilietų sistemomis | Neįgyvendinama MVP etape, tik eksportas į CSV arba el. paštą. |
| Git/CI/CD integracija | Planuojama vėliau; MVP – tik rankinis eksportas. |
| Automatinis diegimas per API | MVP – tik copy-paste arba failo eksportas. |
| Testų automatizavimas | Tik bazinė sintaksės ir struktūros validacija; pilna testavimo sistema – vėlesniuose etapuose. |
| Rizikos įvertinimo logika | Rankinis patvirtinimas. DI tik pažymi užduoties tipą, bet nesiūlo sprendimo rizikos klasės. |
| Automatizuotas planinis palaikymas | MVP nepalaiko suplanuotų veiksmų (cron), tik ad hoc. |
| Versijų palyginimas (diff) | Istorija fiksuojama, bet nėra grafinio palyginimo. |

**6.3 MVP sėkmės kriterijai**

Kad MVP būtų laikomas sėkmingu, turi būti pasiekti šie rezultatai:

* Naudotojai gali savarankiškai suformuluoti tipinę užduotį be techninės pagalbos.
* ≥70 % sugeneruoto kodo sprendimų įgyvendinami be rankinių pataisymų.
* Užduoties ciklas (nuo iniciacijos iki įgyvendinimo) trunka < 5 min.
* Vartotojai supranta, ką sistema padarė ir jaučiasi saugiai priimdami ar atmesdami sprendimą.
* Projektų komanda gauna reikšmingą grįžtamąjį ryšį apie:  
  – naudingiausius užduočių tipus,  
  – sunkiausiai interpretuojamus atvejus,  
  – vartotojo pasitikėjimo ribas.

**Išvada:**  
MVP versija turi būti funkcionaliai siaura, bet vertiškai tiksli – ji turi išspręsti **konkrečią realią problemą: kaip greitai ir be programuotojo atlikti smulkią palaikymo užduotį**. Visi papildomi scenarijai ar giluminės integracijos nukeliamos į vėlesnius etapus.

# Roadmap: tolesnės versijos

Autoprogramuotojo modulis turi būti vystomas etapais – pradedant nuo MVP, kuris sprendžia aiškiai apibrėžtą problemą, ir toliau plečiamas pagal rinkos poreikius, grįžtamąjį ryšį ir sistemos integracijos brandą. Kiekviena vėlesnė versija turi išplėsti produkto funkcionalumą tiek **techniniu**, tiek **organizaciniu** lygmeniu – pereinant nuo įrankio prie **automatizuoto pagalbininko**.

**7.1 Plėtros kryptys pagal funkcines grupes**

**1. Užduočių interpretavimo plėtra**

* **Pažangesnis natūralios kalbos apdorojimas (NLP)** – naudotojas gali aprašyti užduotį laisvai, o sistema supranta kontekstą ir tipą.
* **Automatinis užduočių klasifikavimas** – sistema automatiškai priskiria prioritetą, riziką, priklausomybes.
* **Pagalbinės užklausos** – DI užduoda papildomus klausimus, jei informacija nepakankama.

**2. Kodo generavimo kokybės didinimas**

* **Modulinis kodo struktūravimas** – sistema generuoja kodą pagal projekto struktūros logiką (controllers, services, components).
* **Versijų palyginimas** – kiekvienas pakeitimas turi automatinį „diff“ vaizdą.
* **Sugeneruoto kodo mokymosi mechanizmas** – sistema mokosi iš naudotojų korekcijų.

**3. Testavimo ir kokybės kontrolės automatizavimas**

* **Unit ir functional testų generavimas** – kartu su kodu sugeneruojami testai.
* **Automatiniai UI testai** (render’io tikrinimas, CSS validacija).
* **Integracija su CI/CD sistemomis** (GitHub Actions, Jenkins, GitLab CI).

**4. Deployment automatizavimas**

* **Pilnas API diegimo procesas** – per REST ar GraphQL integracijas su CMS, PaaS ar e. komercijos platformomis.
* **Preview režimo valdymas** – naudotojas mato rezultatą be publikavimo.
* **Rollback mygtukas** – viena komanda leidžia grįžti į ankstesnę versiją su dokumentacija.

**5. Užduočių ir projektų valdymas**

* **Naudotojų rolės ir delegavimas** – projektuose dalyvauja keli asmenys su skirtingomis teisėmis (inicijavimas, patvirtinimas, vykdymas).
* **Palaikymo ataskaitos ir audit trail** – automatinės metinės/mėnesinės veiksmų suvestinės.
* **Klientų režimas agentūroms** – skirtingi projektai, klientų įgaliojimai ir white-label peržiūra.

**6. Automatinis palaikymas ir planavimas**

* **Planinės užduotys (cron tipo)** – sistema vykdo pasikartojančius veiksmus be rankinės iniciacijos.
* **„Watchdog“ režimas** – sistema stebi svetainę ir inicijuoja pataisas, jei aptinkama problema (pvz., sugriuvęs mygtukas).
* **Integracija su stebėjimo sistemomis** (pvz., uptime robot, Lighthouse score, HTML validatoriai).

**7.2 Technologinės integracijos ir infrastruktūra**

* **Git integracija** – automatinis šakų kūrimas, commit'ai su komentarais, pull request’ų generavimas.
* **Jira, ClickUp, Linear integracijos** – užduočių sinchronizavimas, būsenų atnaujinimas.
* **Webhook'ai ir API iškvietimai** – galimybė įtraukti autoprogramuotoją į vidines organizacijos sistemas.

**7.3 Laiko horizontas (rekomenduojamas)**

|  |  |  |
| --- | --- | --- |
| **Versija** | **Laikotarpis** | **Funkcinė apimtis** |
| **MVP (v1.0)** | 2025 Q3 | Tipinių užduočių generavimas, preview, eksportas, rankinis diegimas |
| **v1.1** | 2025 Q4 | Rizikos klasifikavimas, rollback, automatizuotas testavimas |
| **v1.2** | 2026 Q1 | API diegimas, planiniai veiksmai, naudotojų rolės, projektų valdymas |
| **v2.0** | 2026 Q2 | Agentūrų režimas, CI/CD, klientų sąsajos, automatinė diagnostika |
| **v2.1+** | 2026 H2+ | Mokymosi algoritmai, savireguliacija, adaptacija pagal sistemą |

**Išvada:**  
Autoprogramuotojas turi aiškią ir valdomą augimo trajektoriją: nuo tikslaus, vertę kuriančio MVP – iki išmanaus, verslui strategiškai svarbaus IT palaikymo modulio, gebančio savarankiškai inicijuoti, atlikti ir stebėti smulkius techninius veiksmus.

# Rizikos

Nors autoprogramuotojo sprendimas remiasi aiškiu poreikiu ir rinkos problema, jo įgyvendinimas susijęs su tam tikromis rizikomis. Šiame skyriuje aptariamos techninės, vartotojų elgsenos, architektūrinės bei produktinės rizikos ir jų valdymo gairės.

**8.1 Techninės rizikos**

**Rizika:** Netikslus ar klaidingas kodo generavimas dėl LLM elgsenos.  
**Poveikis:** Neteisingi arba nestabilūs pakeitimai, sisteminiai gedimai.  
**Valdymas:**

* Promptų validavimo ir testavimo sistema.
* Sintaksės ir semantikos patikra.
* Rollback mechanizmas privalomas visiems veiksmams.
* Įgyvendinti „preview-first“ diegimo režimą.

**Rizika:** Trūksta suderinamumo su realiomis sistemų versijomis ir struktūromis.  
**Poveikis:** Kodo įterpimas gali būti nesuderinamas su „custom“ sistemomis ar projektų ypatybėmis.  
**Valdymas:**

* Aiškus sistemų palaikymo sąrašas (MVP stadijoje).
* Plėsti palaikomų šablonų biblioteką.
* Užduoties interpretacijos metu – tikslinti platformą, versiją, struktūrą.

**8.2 Vartotojo elgsenos ir pasitikėjimo rizikos**

**Rizika:** Naudotojai nepasitiki, kad sistema atliks veiksmus „saugiai“.  
**Poveikis:** Vengimas naudoti sprendimą realiose situacijose.  
**Valdymas:**

* Paaiškinimo sistema kiekvienam veiksmui („ką ir kodėl darome“).
* Vizualinė peržiūra prieš vykdymą.
* Ribotos automatizacijos režimai MVP etape (tik su patvirtinimu).

**Rizika:** Nesuprantama, kaip naudoti arba ką tiksliai įvesti.  
**Poveikis:** Naudotojai klaidingai suformuluoja užduotis, rezultatai netikslūs.  
**Valdymas:**

* Interaktyvus vedlys arba chatbot’as su papildomais klausimais.
* Pavyzdžiai ir naudotojo pagalba (inline help, tooltip’ai).
* Užduočių šablonų galerija.

**8.3 Architektūrinės ir integracijų rizikos**

**Rizika:** Neapibrėžtas įdiegimo būdas – nuo HTML eksportų iki API.  
**Poveikis:** Sunkumai diegiant sprendimus realiose sistemose.  
**Valdymas:**

* MVP apimtyje naudoti copy–paste ir HTML eksportą.
* Tolesnėse versijose – tiesioginė API integracija su WordPress, PrestaShop, Laravel.

**Rizika:** Nepakankamas audito ir veiksmų atsekamumas.  
**Poveikis:** IT vadovai negali įrodyti, kas ką padarė; prarandamas pasitikėjimas.  
**Valdymas:**

* Privalomas veiksmų žurnalas (log).
* Versijų istorija ir rollback funkcija.
* Integracijos su Git ir projektų valdymo sistemomis.

**8.4 Produktinės ir komercinės rizikos**

**Rizika:** Produktas nepataiko į svarbiausią naudotojo problemą MVP etape.  
**Poveikis:** Mažas naudojimas, ribotas vertės pajautimas.  
**Valdymas:**

* MVP versijoje susitelkti į dažniausias smulkias užduotis (tekstas, forma, CSS).
* Pirmieji naudotojai – tik paprastos sistemų aplinkos (WordPress, PrestaShop).
* Reguliarus grįžtamojo ryšio rinkimas ir prioritizavimas pagal realius atvejus.

**Rizika:** Konkurentai greitai pasiūlo panašų produktą.  
**Poveikis:** Sumažėja inovatyvumo įspūdis.  
**Valdymas:**

* Fokusas į naudotojo patogumą, lokalizaciją, integracijų gilumą.
* Žingsnis į gilesnes integracijas su kitais AdoreIno moduliais (1.1, 1.4).

**Rizika:** Funkcionalumo išplėtimas tampa per brangus ar sudėtingas.  
**Poveikis:** Užstringa produkto vystymas.  
**Valdymas:**

* Modulinė architektūra.
* Etapinė plėtra pagal prioritetus.
* Fokusas į pagrindines 3–5 užduočių klases.

**Išvada:**  
Autoprogramuotojo modulio sėkmė priklauso nuo gebėjimo **subalansuoti automatizavimą su valdomumu, funkcionalumą su paprastumu ir universalumą su konteksto jautrumu**. Aiškus MVP rėmas, rizikų prevencija ir naudotojo pasitikėjimo stiprinimas turi būti nuosekliai integruoti į visus produkto vystymo etapus.