**SĄVOKOS (Automatinis testavimas)**

1 pamoka:

**Testavimas** – procesas, kurio metu tikrinama PĮ kokybė **esamuoju laiku/momentu**. Šio proceso pagrindinė užduotis yra patikrinti ar **PĮ atlieka jai** **keliamus reikalavimus** (requirements), ar šie reikalavimai buvo įgyvendinti, surasti programos klaidas, silpnas vietas, padėti gerinti kokybę.

**Verifikavimas** – tai veikla, kuria siekiama įsitikinti, kad tam tikra funkcija vykdoma korektiškai (atsako į klausimą, ar teisingai **veikia** kuriamas produktas?)

**Validavimas** – veikla, kuria siekiama įsitikinti, kad funkcija atitinka reikalavimus (atsako į klausimą, ar kuriamas teisingas produktas?)

**Kokybės užtikrinimas (Quality Assurance)** – **procesas**, kurio pagrindinė užduotis minimizuoti klaidų ir defektų kiekį. Apsakomas kūrimo ciklas, ar viskas veikia efektyviai, ar komanda dirba kokybiškai. **Atsakingumas už platesnį procesą**, kad visuose žingsniuose viskas būtų **atlikta kiek geriau ir optimaliau.**

**Kokybiška programa –** programa, kuri atitinka jai keltus reikalavimus, veikia be “crash” ir nedaro klaidų, stabili, joje veikia visas funkcionalumas, ji vykdo visus reikalingus skaičiavimus, rodo teisingus duomenis, bei nepažeidžia vartotojo teisių/neplatina duomenų.

**Reikalavimai** – tai **sąrašas sąlygų,** kurias turi **tenkinti kuriama programinė įranga.** Labai dažnai iš reikalavimų galima suprasti ką darys kuriama PĮ, kokias problemas spręs.

**Vartotojo reikalavimai** – ko vartotojas tikisi iš sistemos (lengvai skaitomas dokumentas).

**Sistemos reikalavimai** – dokumentas skirtas aprašyti kokie komponentai turėtų sudaryti sistemą bei kaip ji turėtų veikti.

**Programos specifikacija** – techninis dokumentas skirtas programuotojams.

**Infrastruktūra** – **pagrindas arba karkasas, kuris išlaiko sistemą.** IT ir kompiuterinėje srityje, infrastruktūra sudaryta iš fizinių ir virtualių resursų, kurie palaiko PĮ/sistemos veikimą, procesų vykdymą, duomenų laikymą, procesavimą ir jų analizę. Pvz.: hardware (techninė įranga), software (PĮ), routers, apps.

**Duombazė** - organizuotas (susistemintas, metodiškai sutvarkytas) **duomenų rinkinys**, kuriuo galima individualiai naudotis elektroniniu ar kitu būdu. Duomenų bazėje įrašai saugomi tokia tvarka (pavidalu), kad kompiuterinė programa gali jais naudotis ir atsakyti į užklausas.

**Error/Mistake** – tai klaida žmogaus veiksmuose, kuri iššaukia neteisingą rezultatą

**Defect/bug/fault/problem** – tai **klaida** sistemoje, kuri gali iššaukti **neteisingą sistemos veikimą**. Jei problema nėra surandama ir patenka pas klientą ji traktuojama kaip “failure”.

**Failure** – kai sistema neatlieka savo funkcijos, klaida matoma vartotojui.

**Produkcinė aplinka** – kuo naudojasi ir mato galutinis vartotojas. (Pvz. lrt puslapis).

**Neprodukcinė aplinka** (developer/testuotojo aplinka) – aplinka, kurią mato tik to puslapio tvarkytojai (testuotojai, programuotojai). Galutinis vartotojas šio puslapio nemato.

**Funkcinis testavimas** - programinės įrangos testavimo tipas, **patvirtinantis programinės įrangos sistemos veikimą atsižvelgiant į funkcinius reikalavimus/specifikacijas**. Funkcinių testų **tikslas - išbandyti kiekvieną programinės įrangos funkciją**, pateikiant atitinkamą įvestį, patikrinant išvestį pagal funkcinius reikalavimus. Atsako į klausimą – **AR programa veikia**?

Nuoroda: <https://lt.csstricks.net/8222639-what-is-functional-testing-types-and-examples-complete-tutorial>

* **Pagrindinės funkcijos** : **Pagrindinių programos funkcijų** testavimas
* **Pagrindinis naudojimas** : Tai apima pagrindinį sistemos tinkamumo naudoti testavimą. Jis patikrina, ar vartotojas gali laisvai naršyti ekranuose be jokių sunkumų.
* **Prieinamumas** : Patikrina sistemos prieinamumą vartotojui
* **Klaidos sąlygos** : testavimo metodų naudojimas klaidų sąlygoms patikrinti. Jis patikrina, ar rodomi tinkami klaidų pranešimai.

**Nefunkcinis testavimas** - programinės įrangos testavimo tipas, siekiant **patikrinti nefunkcinius programinės įrangos aspektus (našumą, tinkamumą naudoti, patikimumą ir kt.).** Jis sukurtas siekiant patikrinti sistemos pasirengimą pagal neveikiančius parametrus, kurių niekada neatlieka funkciniai bandymai.

1. **Saugumas**
2. **Patikimumas** (Kiek bet kuri programinės įrangos sistema nuolatos atlieka nurodytas funkcijas be gedimų. Tai išbandoma patikimumo testavimu)
3. **Išgyvenamumas** (Parametras patikrina, ar programinės įrangos sistema ir toliau veikia, ir atsistato sistemos gedimo atveju. Tai patikrina atkūrimo testavimas (regression))
4. **Prieinamumas**
5. **Naudojimas**
6. **Mastelis** (Šis terminas reiškia laipsnį, kuriuo bet kuri programinė įranga gali išplėsti savo apdorojimo pajėgumus, kad patenkintų paklausos padidėjimą. )
7. **Sąveika** (įrangos sąsajos su kitomis programinės įrangos sistemomis.)
8. **Efektyvumas** (Kiek bet kuri programinės įrangos sistema gali valdyti pajėgumą, kiekį ir atsako laiką.)
9. **Lankstumas** (programa gali dirbti skirtingose ​​aparatinės ir programinės įrangos konfigūracijose. Kaip ir minimalūs RAM, CPU reikalavimai.)
10. **Perkeliamumas** (Programinės įrangos lankstumas perkėlimui iš dabartinės aparatinės ar programinės įrangos aplinkos.)
11. **Pakartotinis naudojimas** (Tai reiškia programinės įrangos sistemos dalį, kurią galima konvertuoti naudoti kitoje programoje.)

Nuoroda: <https://lt.csstricks.net/8222489-what-is-non-functional-testing-types-with-example>

**Performance testing** – sukuriama **(ap)krova programai,** patikrinimas ar ji veikia greitai ar lėtai.

**Happy flow** – testavimas, kai/ar suvedami teisingi duomenys ir gaunamas pozityvus rezultatas. **Įprastas/tikėtinas scenarijus**, kurį įvykdžius nesukeliamos klaidos programoje.

**Negative case –** scenarijus, kuomet suvedami neteisingi duomenys ir susidaro klaida.

**Test Entry Criteria** – **dokumentacija su kriterijais, paruošiama prieš kuriant PĮ ir parodanti, kada galima pradėti PĮ kūrimą.** Su dokumentacija jau galima kurti planavimą, kurtis test cases (testavimo atvejus), išsigryninti funkcijas (krepšeliai, kalkuliacijos).

**Test Exit Criteria** – **kriterijai**, kurie parodo, kad **testavimas buvo pabaigtas arba nusako, kokius kriterijus reikia įvykdyti, kad sustabdyti testavimą**. Testuotojai pateikia informaciją, konstatuoja faktus, kad įvykdė kažkokius scenarijus, kokius testus, ir yra kokie rezultatai. Pagal rezultatus kiti darbuotojai sprendžia, ar reikia leisti PĮ ar ne.

**Regression test** **(atkūrimo testavimas)** – **testų** (funkcinių ir nefunkcinių) **perleidimas/pertestavimas iš naujo**, kad patikrinti, ar visas senas PĮ funkcionalumas išliko **padarius pakeitimą PĮ/kode**. Ar įvedus naujus pakitimus sistemoje PĮ neregresavo. (Pvz. pataisius bug‘ą).

**Test execution** – testavimas pagal sukurtus testavimo atvejus, defektų registracimas ir jų ištaisymas/pertestavimas.

**Test object** – produkto aprašymas

**Test scope** – kas bus testuojama.

**Out of scope** – ko testuoti neplanuojama.

**Testavimo scenarijus** – **aprašymas, kokios funkcijos bus testuojamos**. Tai **dokumentas**, **kuriame aprašomi testavimo atvejai** (testuojamai funkcijai), testavimo žingsniai, būtinos sąlygos (pre-conditions), testavimo duomenys, trukmė, papildoma info.

**Testavimo atvejis** – vienas iš būdų, **kaip reikia ištestuoti funkciją**. (happy flow/negative case). Pagrindinis **tikslas – praktiškai išbandyti kurią nors programos funkciją** ir **nustatyti, ar įgyvendintas funkcionalumas atitinka reikalavimus.**

**Compatibility (suderinamumas)** – scenarijų panaudojimas visose naršyklėse.

**SDLC** – programos **kūrimo** ciklas.

**STLC** – programos **testavimo** ciklas.

**Smoke testas** (Sanity, Confidence test) - tai happy flow tipo testas, kuriuo patikrinamas **kritinis programos funkcionalumas**. Patikrina būtent pagrindines puslapio dalis, pvz. Ar puslapis atsidaro. Tik po šio testo galima pradėti funkcinį (sisteminių komponentų) testavimą.

**Regression testing** – Tai toks testas, kurio pagrindinė paskirtis patikrinti, ar naujas kodas (pakeistas kodas) išlaiko ankstesnėje versijoje veikiantį funkcionalumą, testuojant visą funkcionalumą, ne tik pasikeitusias sritis bei modulius. Regresijos statysas dažnai yra Test Exit Criteria.

**Regression testing** – automatizavimas. Efektyviausias būdas kovai su regresijomis – turėti [automatinių testų](https://lt.wikipedia.org/wiki/Automatinis_testas) rinkinį, kuris detaliai patikrintų visos programos darbą. Toks testų rinkinys leidžia sumažinti keitimų (ypač nedidelių) riziką, neinvestuojant daug resursų visos programos darbo [testavimui](https://lt.wikipedia.org/w/index.php?title=Testavimas&action=edit&redlink=1).

Regresijų pasitaiko todėl, jog įvairios programos dalys dažnai yra labai glaudžiai tarpusavyje susiję, ir programuotojas, taisydamas ar plėsdamas vienos dalies darbą, gali neapsižiūrėjęs pažeisti su ja susijusios kitos dalies darbą.

Kaip kraštutinė priemonė naudojamas nenutrūkstamas testavimas, kuomet testų rinkinys veikia nuolat, automatiškai patikrindamas programą po kiekvieno keitimo.

**Testavimo lygiai:**

* **Unit Tests** (Modulių testavimas) Pirmas testas, kurį rašo programuotojai. Pasirašo tas, kuris pasirašė metodą.
* **Integration Tests (**Integracinis testavimas) – sąveika/komunikacija tarp dviejų servisų/modulių. Atlieka programuotojai/patyrę testuotai. Testuojama komponentų tarpusavio sąveika. Gali aprašyti tas kuris parašė metodus arba kitas. Pirma turi veikti du metodai, kad įvyktų integracija. Skaičiavimas – testas, ar vyksta integracija abiejų metodų.
* **System, UI Testing (**Sistemos testavimas) – tikrinamas visos sistemos veikimas. Atlieka testuotojai. GUI, komandinė eilutė, API (Application Programming Interface).
* **Acceptance test** (priėmimo testavimas) – formalus testavimas, tikrinamas pagal „acceptance criteria“, atlieka testuotojai, tikrinami nefunkciniai reikalavimai, atsako į klausimą – ar patogus, atitinka reikalavimus, teisingai veikia? Aukšto lygio testas, padengia alfa/beta testavimą.

**Boundary values (ribinis testavimas) –** testavimo metodas, kuris leidžia patikrinti, kaip sistemos laukai veikia ribinėse situacijose. Validacija – pvz nuo 10-50. Pirma patestuojama šiek tiek mažesnė reikšmė už minimum (ar klientas gali įvesti ko negalima). Ar tikrai galima įvesti mažiausią reikšmę (ar tikrai ribinis mažiausias skaičius yra 10 o ne koks 15). Tuomet patikrinama vidutinė reikšmė. Analogiškai su maximum.

**Field validation (lauko validacija, error handling**) –

**Decision table (Sprendimų lentelė/Matrica) –** sudaroma lentelė apimanti visus galimus veiksmus ir jų rezultatus.

Lentelė gerai, jei reikia kąnors automatizuoti (dėl aiškumo). Pagal lentelę galima išsigryninti visus scenarijus.

**White box testai** – unit testai, Programuotojas parašė kodą, žino ką parašė, ir tam kodui parašo kodą. Daro programuotojas. Struktūros testavimas. Testai kuriami po kodo analizės ir yra aiškus supratimas kaip veikia programa. Aiškus proceso eiliškumas ir aiškios priklausomybės.

**Black-box testing** – kai testuotojas tikrina aplikaciją, paduoda reikšmę ir nežino kas su ja bus, ir tikisi gauti kažkokį atsakymą. (nežino kaip veikia kodas. Žino kokie yra inputs outputs). Daro testuotojas/žmogus nedalycvaujantis programavimo procese (negali atlikti programuotojas) Funkcinis testavimas, imituojamas vartotojas. Žinoma - įėjimai ir išėjimai (input/output), programos kodas nežinomas, **testuojama naudojantis reikalavimais,** specifikacija, reikalinga veikianti sistema arba jos dalis.

**Alfa/beta testavimas** – galutinė versija, pateikiama daliai klientų arba užsakovui.

**Testavimo metodų svarba** – padeda sugalvoti konkrečius testavimo atvejus bet kokiai PĮ.

**Nefunkcinis testavimas** – dažniausiai daromas po to, kai atliekamas funkcinis testavimas. Matuojami tokie kriterijai kaip greitis, failų dydis, saugumas, patogumas.

**Performance testing** – Planuojama, kokia gali būt apkrova, pagal tuos planavimus sugeneruojami performance testai.

* **Stresinis testavimas (Stress)** testuoja sistemą su tokiais **duomenų kiekiais, kurie išeina už leistino kiekio ribų arba yra netoli maksimalaus leistino kiekio ribos.** Šio testavimo metu siekiama patikrinti, kas atsitiks, kai Sistema dirbs esant sąlygoms, kurioms ji nebuvo suprojektuotaL ar busprarastas funkcionalumas ir duomenys, kiek jų bus prarasta. (pvz. Sistema gali apdoroti 2mb duomenų, tuomet suvedama 2,5mb).
* **Load/Apkrovos** testavimo tikslas – patikrinti kaip sistema reaguoja į **dideles apkrovas jas didinant palaipsniui.** (Vartotojų apkrova – daugiau vartotojų, užklausų (20000 max))
* **Endurance/Ištvermės** testavimo tikslas – patikrinti kaip sistema reaguoja į **padidintas apkrovas per ilgą laiką.** (užkeliama apkrova ir ji laikoma ilgą laiką, žiūrima kaip sistema reaguoja).

**Security/Penetration testing (**saugumo, įsiskverbimo testai) – toks testas, kuris parodo sistemos **silpnas vietas, rizikas ir apsaugo nuo duomenų pagrobimo/atakų.** Tikslas – įvardinti visas įmanomas skyles ir silpnumus sistemoje, kas gali vėliau turėt pasėkmes kaip duomenų/informacijos praradimą, nutekinimą.

* **SQL** injekcijos ataka (Sequel kalba, skirta komunikuoti su duombaze (tekstine lentele su daug lauku, kurioje saugomi duomenys). Jeigu įvedimo laukas nėra apsaugotas, papraščiausia ataka SQL – parašius 1=1 (TRUE) į lauką.)
* **XSS** injekcijos ataka

**JIRA** – projekto valdymo sistema. (Dažnai būna pritaikyta Agile metodologijai.) Skirta projektams, darbams, defektams, kurta užduotims. **Svarbiausi laukai** – Summary, Description, Priority, Component.

**Bug reporting rules** –

* **Summary (santrauka)** – trumpas ir objektyvus problemos sprendimas. Vengti – apibendrinančių frazių (niekas neveikia), emocijų/pesimizmo „vėl neveikia“. Naudoti – vieno sakinio aprašymas, 10-15 žodžių, kiek įmanoma aiškiau aprašyti problemos esmę.
* **Component (komponentas)** – konkreti sistemos dalis (posistemė), kur rastas defektas. Vengti – blogai nustatyti komponentą. Naudoti – komandos patarimus, patirtį, sistemos žinias.
* **Priority (**prioritetas) –
* **Blocker** – kritinis funkcionalumas neveikia, nėra prieigos prie jo, programa nenaudojama, greitaveika degradavo N %
* **Critical** - sistema veikia, paveikta viena iš svarbiausių funkcijų, paveiktas kritinis funkcionalumas, tačiau yra „workaround“ apėjimas, duomenų praradimas (jei reik sumokėt su specifiniu mokėjimo metodu, ir jis neveikia, kompanija praranda pinigus).
* **Major –** paveiktas nekritinias funkcionalumas, nežymus greitaveikos degradavimas, blogai implementuotas funkcionalumas, kuris neblokuoja testavimo.
* **Minor –** blogas vartotojo sąsajos elementų išdėstymas, vartotojo sąsajos klaidos, viskas kas nepapuola į kitus minėtus priority.
* **Attachment –** screenshot (ekrano), failas su duomenimis, sistemos logas. Vengti daryti teksto nuotrauką
* **Description –**
* žingsniai kaip atkartoti/pamatyti problemą
* galimai kuriuo metu ji buvo pastebėta
* Expected result, kodėl?
* Actual result
* **Affects ir fix versions –** Affects – kurioje versijoje defektas **aptiktas** pirmą karta, Fix version – kurioje versijoje defektas **ištaisytas** kartą (kurioje bus padarytas pakitimas/atnaujinimas).
* **Aplinka (Environment) –** aprašyti kokioj aplinkoj buvo rasta klaida (Production ar developer)
* **Label –** kažkoks tag, kad galima būtų lengviau surasti užduotis.
* **Reporter/Asignee – kas užregistravo/kam skirta**

**Back-log** – vieta AGILE, kur yra išrašytos visos sukurtos užduotys, su kuriom vėliau komanda dirbs.

**Front-end –** tai grafinės svetainės/programėlės vartotojo sąsajos (UI) kūrimas naudojant HTML, CSS ir „JavaScript“, kad vartotojai galėtų peržiūrėti tą svetainę ir su ja bendrauti. Įeina vartotojo sąsaja (UI), Web aplikacija, mobile Aplikacija.

Frontend yra ta tinklapio dalis, kurią mato vartotojas ir į kurią tiesiogiai įsitraukia. Ji kuriama su HTML, CSS ir JavaScript.

**Back-end –** posistemė, kuri veikia už kliento matomo vaizdo. Tai duomenų bazės, kuriose laikomi puslapio duomenys, verslo logika, visi užkulisiai. Pvz.: mobilus įrenginys (daug komponentų, servisai, duombazė, API GATEAWAY).

**HTTP** – naudojamas komunikacijai su back-end. Tai pagrindinis metodas naudojamas pasauliniame tinkle, pasiekti svetainės informaciją. Pradinė protokolo paskirtis – pateikti standartinį būdą HTML puslapiams skelbti ir skaityti.

**HTML** – kalba, naudojama rašyti puslapiams, kad juos galėtų perskaityti vartotojas, taip pat, kad galėtų paspausti mygtukus ir būti nukreiptas į reikiamus puslapius/vietas. Pagrindinis HTML kalbos vienetas yra elementas.

Tai kompiuterinė žymėjimo kalba, naudojama pateikti turinį internete.

**HTTPS://** – saugus protokolas. Duomenys keliauja koduotai naudojant certifikatą, keliauja slaptai. (skaičių raidžių derinys).

**HTTP://** – paprastas protokolas. Kai jis naudojamas, visi duomenys kuriuos įvedam nešifruotai/nekoduotai keliauja internetu.

**HTTP** užklausų metodai –

* GET – dažniausia užklausa, reikalaujanti tam tikro resurso iš URL adreso
* POST – panašu į GET, bet siunčiama papildoma informacija, paprastai tai reikšmių ir raktų poros, nusakančios HTML formos laukų reikšmes.
* PUT-naudojama failams į server įkelti
* DELETE
* HEAD
* TRACE
* CONNECT

**Header (antraštė)** – papildomi duomenys. Rakto + Reikšmės derinys. Visi atributai – kaip papildomi duomenys, gali pasakyt ko tiksliau norim iš to adreso kai kreipiamės.

HTTP headers yra svarbi API užklausų ir atsakymų dalis, nes reprezentuoja metaduomenis susijusius su API užklausomis ir atsakymais.

**Path** – adresas į kurį reik kreiptis.

**URL (Uniform Resource Locator)** – trumpai, web address.

**Parameters (parametrai)** – viskas kas seka po web adreso.

**Request Body** – siunčiami duomenys. Kai padaroma bet kokia užklausa, ji gauna atsakymą. Gauna skaitine reikšmę, gauna ir atsakymo kodą.

**Status code** – statuso kodas, parodantis, ar užklausa įvykdyta teisingai ar ne.

* **2XX (Successful)** – užklausa sėkmingai įvykdyta
* **4XX (Client Error)** – užklausa klaidinga ir negali būti įvykdyta. Padaryta vartotojo klaida/suvesti blogi duomenys,...Pvz.: užklausti duomenys/resursai neegzistuoja, nėra suteiktos teisės. Pvz.: **404** – Error/Nerastas adresas. 403 – autorizacinė klaida (Forbidden: Access denied).
* **5XX (Server Error)** – serveris nesugebėjo apdoroti (greičiausiai teisingą) užklausą. Kažkas blogai su serveriu/serverio klaida. Pvz.: **502** Bad Gateway. Kažkas netaip su API Gateway (pas paslaugos tiekėją, iki jo pasikreipia, bet nuo jo neturi kur kreiptis).

**API** – Application programming interface. **Rinkinys metodų, padedantis komunikuoti su sistemos komponentais.**

API pagal prieinamumą būna:

* **Private** – naudojamas tik kompanijos/projekto viduje
* **Partner** – naudojamas tarp Verslo partnerių (pvz.: Uber ir Bolt)
* **Public** – viešai prieinami resursai.

**JSON** – JavaScript Object Notation. **Duomenų formatas**, nepriklausomas nuo programavimo kalbos.

[Atviro standarto](https://lt.wikipedia.org/w/index.php?title=Atviras_standartas&action=edit&redlink=1) formatas, perduodantis duomenų objektus, sudarytus iš atributo ir reikšmės porų, lengvai skaitomame tekste.

**Postman** - aplikacija, naudojama API testavimui. Testuoja HTTP užklausas, utilizuoja/naudoja GUI, per kuriuos gaunami įvairių tipų atsakymai, kurie vėliau validuojami.