

# INTERNETINIŲ SISTEMŲ TESTAVIMO TOBULINIMAS

Elžbieta Gulbinienė  
Janina Petrauskienė  
Grażina Tautvydienė  
Šiaulių kolegija  
Aušros al. 40, Šiauliai

## Anotacija

Straipsnyje analizuojami internetinių sistemų testavimo modeliai bei labiausiai paplitusios testavimo strategijos. Aptariama, dažniausiai taikoma, automatizuoto testavimo eiga ir būtinos specifikacijos. Sistemos funkcionalumui svarbus ir scenariinio testavimo būdas, todėl aptartas WebScriptRecorder įrankis, skirtas testo scenarijui sugeneruoti.

## Įvadas

Internetinė sistema – tai informacijos sistema, kurios technologiniam pagrindui naudojamos interneto palaikomos technologijos: sistemos architektūra, techninė įranga (tinklų specifikacijos, kompiuterinė įranga), programinė įranga (adresacijos, duomenų perdavimo internetu, nutolusių kompiuterių susijungimo, kiti protokoliai, vartotojo sąsaja), darbo principai, bei (nebūtinai) vieši interneto tinklo kanalai. Šiuolaikinėse internetinėse sistemose praktiškai visada yra naudojamas bent nedidelis skaičius su internetu susijusių technologijų. Bendruoju technologijų požiūriu interneto informacijos sistemas galima apibrėžti kaip sistemas, naudojančias visuotinai paplitusias interneto paslaugas: WWW (pasaulinį žiniatinklį), FTP (failų perdavimo protokolą), elektroninį paštą, naujienų grupes ir kt. Nuo tradicinių, lokalių (organizacijų mastu) sistemų interneto informacijos sistemos skiriasi savo fiziniu dydžiu bei integracijos su organizacijos veiklos sistema laipsniu.

Internetinės sistemos projektavimo etape definiuojami ir projektuojami šie elementai:

- techninė įrangos platforma: parenkamas kompiuterių tipas, tinklo galimybės, įvedimo bei išvedimo įrenginiai, kaupikliai;
- programinė įranga: programavimo kalba, programinės įrangos paketas, duomenų bazių valdymo sistema;
- išvedama informacija: ataskaitų bei sistemos išvaizdos projektavimas;
- įvedama informacija: formos, dokumentai, įvedamų duomenų tikrinimo procedūros;
- vartotojo sąsaja;
- modulinė architektūra: sudėtinės programinės įrangos dalys ir jų tarpusavio sąveika;
- testavimo planas, testavimo duomenys.

Vien tik inžinierinės paradigmos laikymosi rezultatas sukuria daug informacijos sistemų, kurios iš tiesų netenkina organizacijos joms keliamų reikalavimų. Viena klasikinių problemų - neįvertinami būsimųjų sistemos vartotojų poreikiai. Neretai koncentruojamasi ties tais procesais, kuriuos lengva automatizuoti, o ne tais, kuriuos reikėtų automatizuoti. Tokios problemos skatino kurti informacijos sistemų teoriją, leisiančią pažvelgti į internetinių sistemų problematiką iš aukštesnio lygio, apimant įvairiausias problemas, ieškant joms sprendimų, naujų galimybių bei jų realizavimo metodų, kurie išliktų aktualūs pakankamai ilgą laiką. Pagrindinė problema – neprofesionalus internetinių sistemų testavimo atlikimas, nesavalaikis klaidų radimas. Tai, geriausiu atveju, atlieka klientai arba patys programuotojai. Pastariesiems dažnai sunku pastebėti vartotojiškumo klaidas: operacijų panaudojimo nepatogumą, nenumatytų ar neteisingų veiksmų atveju išskylančius programinės įrangos defektus.

Testavimas - tai programos vykdymo procesas, kurio tikslas yra klaidų radimas. Tai procesas, kuris padeda nustatyti programos išbaigtumą, taisyklingumą, saugumą ir kokybę.

Viena pagrindinių testavimo taisyklių: kuo anksčiau surandamos klaidos, tuo pigiau jas ištaisyti bei didesnė tikimybė, kad klaidos bus ištaisytos teisingai. Nesurastos klaidos kaina didėja, kai internetinė sistema nėra tinkamai ištestuota ir klaidą randa užsakovai arba vartotojai (Grigaitis, 2008). Testavimo tikslas yra: atrasti sistemos defektus, ir įvertinti, ar sistema gali būti naudojama darbinėje situacijoje [7].

Šio straipsnio **tikslas** – išanalizuoti internetinių sistemų testavimo proceso teorinius aspektus bei atskleisti testavimo tobulinimo galimybes.

**Objektas** – internetinės sistemos testavimo modeliai.

**Uždaviniai:**

1. Apibūdinti labiausiai paplitusias programinės įrangos testavimo strategijas;

2. Apžvelgti testavimo būdus ir tipus;
3. Išanalizuoti testavimo modelius bei testavimo eigą;
4. Parengti rekomendacijas internetinių sistemų testavimui tobulinti.

**Metodai** – teorinės literatūros analizė, modeliavimas.

### Testavimo strategijos

Prieš testuojant programą, reikia pasirinkti testavimo strategiją. Labiausiai paplitusios testavimo strategijos yra: juodosios dėžės (Black-box), baltosios dėžės (White-box) ir pilkosios dėžės (Grey-box) (Ian Sommerville, 2001), [5].

*Baltosios dėžės* testavimas (White-Box) vadinamas logika paremtu testavimu, kuris leidžia išanalizuoti vidinę programos struktūrą. Nagrinėjant programos vidinę logiką, struktūrą, yra sudaromi testavimo duomenys. Šios strategijos tikslas yra sukurti analogą išsamiam įvesties duomenų testavimui juodosios dėžės požiūriu. Tiesa, yra tokia problema, kad unikalių logiškų testavimo kelių programoje gali būti astronomiškai didelis skaičius.

*Pilkosios dėžės* testavimo (Grey-Box) strategija vadinama geriausia ir protingiausia testavimo strategija. Apjungia juodosios ir baltosios dėžės testavimo elementus. Pilkosios dėžės testavimas apžvelgia programos rezultatą (iš galutinio vartotojo pozicijos), operacinę aplinką bei techninį konkrečios sistemos kontekstą. Šis būdas labai tinkamas efektyviam internetinės sistemos testavimui, nes internetinė sistema susideda iš įvairių komponentų, tiek programinės, tiek techninės įrangos. Komponentės turi būti testuojamos sistemos dizaino kontekste, norint įvertinti jų funkcionalumą ir suderinamumą.

*Juodosios dėžės* testavimas (Black-Box) - viena pagrindinių testavimo strategijų, kitaip dar vadinama duomenų įvedimo - išvedimo testavimu (Ian Sommerville, 2001), [5]. Šiuo būdu į programą yra žiūrima kaip į „juodąją dėžę“, t.y. visiškai nekreipiamas dėmesys į programos vidinę struktūrą. Programos testavimo atvejai yra paremti sistemos specifikacija, testo planavimas pradedamas programinės įrangos kūrimo procese. Taigi, vienintelis būdas patikrinti visas galimas klaidas yra įvesti visas galimas reikšmes. Tokiu atveju reikia tikrinti ne tik validžias reikšmes, bet visus galimus įvesties duomenis. Tai padaryti yra neįmanoma, nes įvesties duomenų skaičius yra begalinis. Kadangi toks išsamus testavimas nėra galimas, tai reikia su baigtiniu skaičiumi testavimo variantų rasti begalinį skaičių klaidų. Tam yra naudojama testavimo variantų (test-case-design) strategija. Testuotojas gali išbandyti tik ribotą kiekį visų galimų įvesties duomenų. Tad norisi pasirinkti būtent tuos variantus, kurie turi didžiausią tikimybę surasti kuo daugiau klaidų. Vienas iš būdų rasti reikalingus variantus, tai parinkti tokius testavimo variantus, kurie padengia sritis taip, kad sumažėja reikalingų testavimo variantų skaičius; t.y. pasako apie klaidų buvimą ar nebuvimą žemiau ir virš pasirinktos specifinės reikšmių aibės. Kitaip tariant, įvesties duomenų sritis yra suskirstoma į ekvivalentumo klases (equivalence classes). Daroma prielaida, kad jei vienas testavimo variantas iš tam tikros ekvivalentumo klasės randa klaidą, tai ir visi kiti ras tą pačią klaidą. Ir priešingai, jei testavimo variantas nerado klaidos, tai tikėtina, kad ir kiti variantai iš pasirinktos ekvivalentumo klasės neras klaidos.

Įvedamų duomenų sritis yra suskaidoma į validžias (valid) ekvivalentumo klases, t.y. teisingai įvedamus duomenis, ir nevalidžias (invalid) - visi kiti įvedimo duomenys (klaidingi). Daugiausia yra sudaroma nevalidžių, netikėtų įvedimo duomenų klasių.

Svarbiausia programos testavime yra efektyvių testavimo variantų sukūrimas. Taip stengiamasi sukurti labiau išbaigtus testus. Pati neefektyviausia testavimo metodika yra atsitiktinių įvesties duomenų testavimas, t.y. testuoti atsitiktinai pasirenkant kažkokį poabį visų galimų įvesties reikšmių. Testavimo variantai kuriami tam, kad testavimo duomenys būtų pasirenkami protingai. Geras testavimo variantas yra tas, kuris turi pagrįstą tikimybę rasti klaidą. Užrašant testavimo variantus, yra žinomi įvedimo duomenys ir tikėtinas rezultatas. Tai apsibrėžiama prieš atliekant testus [7]. Testavimo variantai, kurie tyrinėja reikšmių ribų sąlygas pateikia geresnį rezultatą. Ribų sąlygos yra tokios situacijos ant, virš ir žemiau įvedamų/išvedamų duomenų ekvivalentumo klasių ribų. Ribų reikšmių analizė skiriasi nuo ekvivalentumo klasių analizės tuo, kad ribų reikšmių atveju yra tikrinamos būtent ekvivalentumo klasių ribinės reikšmės (o ne bet kuris elementas iš tos aibės). Taip pat atsižvelgiama į išvedamos informacijos galimas reikšmes. Šis metodas reikalauja nemažai kūrybingumo. Jei testuojama kažkokia sritis, tai geriausia imti ribinę reikšmę, vieną žemiau srities ribos ir vieną aukščiau ribos. Imami variantai turi padengti apatinę ir viršutinę ekvivalentumo klasių aibės ribą.

### Testavimo būdai

Skirtingiems testų tipams taikomas skirtingas testavimo būdas – rankinis arba automatizuotas. Automatizuotus būdus dažniausiai taiko didelės kompanijos, kurios turi laiko ir pinigų testų automatizavimui. Sukūrus automatizavimo įrankius, vėliau juos galima taikyti kituose projektuose.

*Rankinis testavimas* (manual testing) - tai testavimo žingsniai, kuriuos testuotojas atlieka rankiniu būdu. Dažniausiai testavimas atliekamas rankiniu būdu, kai procesą automatizuoti yra labai sudėtinga, reikia daug laiko arba neįmanoma dėl techninių trukdžių (pavyzdžiui, nustatyti komponentės elgesį, kai dingęs tinklo ryšys).

*Automatizuotas testavimas* - tai programinės įrangos naudojimas testų vykdymo kontrolei, rezultatų palyginimui tarp esamų ir nuspėjamų rezultatų, testų išankstinių sąlygų nustatymams ir kitoms testų kontrolės ir rezultatų funkcijoms. Dažniausiai, testų automatizavimas – tai esamų rankinių (manual) procesų, kurie naudoja formalizuotus testavimo procesus, automatizavimas.

Testų automatizavimas dažniausiai taikomas regresijos testams (regression tests).

### Testavimo tipai

Testavimo procesas yra gana naujas, įvairioje literatūroje išskiriami skirtingi testų tipai.

*Funkcinis testavimas (Functional Testing)* – internetinės sistemos testavimas, norint įsitikinti, kad visos funkcijos veikia teisingai: paveikslukai rodomi, nuorodos veikia, meniu tvarkingas, tekstas rodomas taisyklingai ir t.t. Pagrindinis dėmesys kreipiamas į įvedamų ir išvedamų duomenų korektiškumą.

*Suderinamumo testavimas (Compatibility Testing)* – tikrinama internetinė sistema, norint įsitikinti, kad ji teisingai veikia su skirtingomis naršyklėmis, operacinėmis sistemos, kalbomis, duomenų bazėmis.

*Atlikimo testavimas (Performance Testing)* – šiuo testu nustatoma, kaip greitai sistema atlieka užduotis bei surandami užduočių atlikimo defektai.

*Apkrovos testavimas (Load Testing)* – tikrinama, kaip sistema veikia, esant didelei apkrovai (daug vartotojų vienu metu).

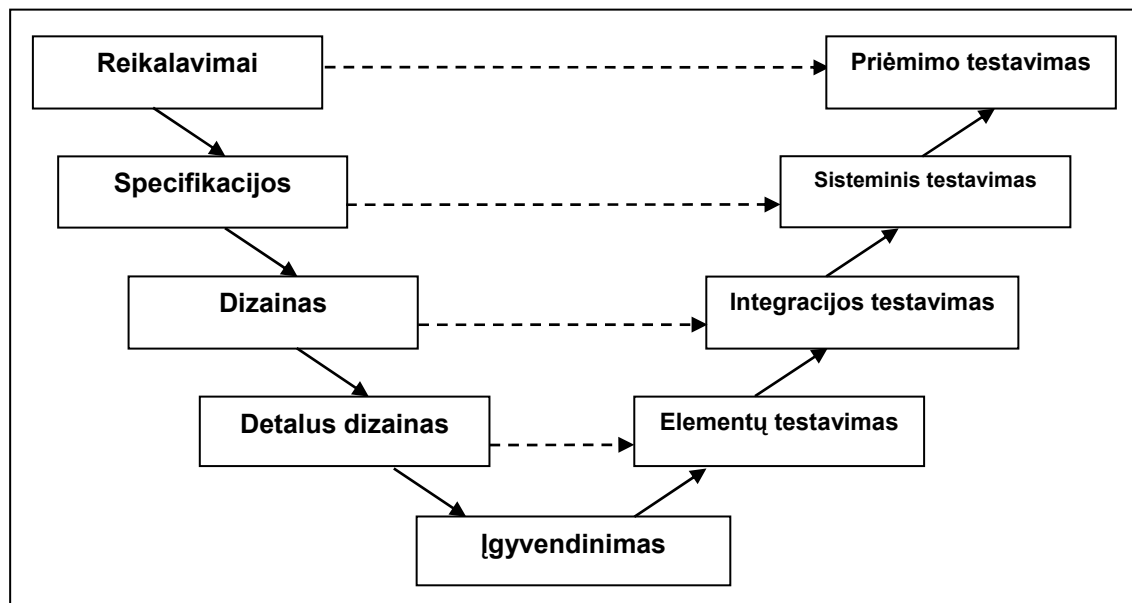
*Regresijos testavimas (Regression Testing)* – ieškoma, ar naujoje sistemos versijoje ištaisytos žinomos klaidos ir ar neatsirado naujų. Dažniausiai šis testavimo procesas yra automatizuojamas.

*Spaudimo testavimas (Stress Testing)* – tikrinama kaip sistema veikia esant didelei apkrovai (viršijančiai apibrėžtų limitų reikalavimus), tranzakcijų kiekiui, dideliame spaudime. Sistemos spaudimas dažnai lemia defektų pasirodymą. Sistema spaudžiama, kad klaidingai veiktų. Ji neturėtų katastrofiškai suklysti. Stresinis testavimas tikrina nepriimtina duomenų ar paslaugų praradimą. Ypatingai tinka paskirstytoms sistemoms, kurios gali eksponuoti žymų nuosmukį, kai tinklas perkrautas. Pageidaujamas rezultatas – neglobalus sistemos smukimas.

*Vartotojiškumo testavimas (Usability Testing)* – ieškoma vartotojiškumo klaidų: ar patogiu naudoti sistema, aiškios instrukcijos, vartotojui suprantami sistemos veiksmi.

### Testavimo eiga

Testavimas gali būti atliekamas po sistemos įdiegimo. Autorių kolektyvas [3] pateikia sistemos integracijos ir testavimo modelį („V-modelis“), kuriame parodomas sąryšis tarp vystymo ir testavimo fazių (žr.1 paveikslą).



1 pav. V-modelis

Pagal šį modelį sudaromi reikalavimai, specifikacijos bei dizainas sistemai. Tik toliau seka sistemos įdiegimas (implementation) ir testavimas.

Pirmiausiai atliekamas *elementų testavimas (unit testing)*. Pagal esamą detalaus dizaino prototipą testuojama minimali sistemos komponentė, modulis, ieškoma funkcinų ir struktūrinių klaidų. Šis testavimas atliekamas prieš visos sistemos komponentų integravimą (Grigaitis, 2008).

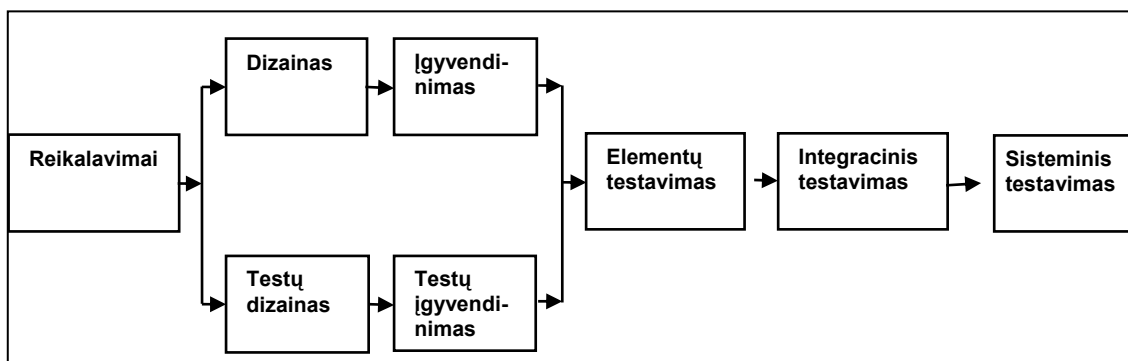
Sekantis žingsnis - sukurto sistemos dizaino pagalba atliekamas *integracijos arba modulių sujungimo testavimas (integration testing)*. Šiuo testavimu atskleidžiami sąsajos defektai ir integruotų modulių interakcija.

Pagal esamas specifikacijas atliekamas *sisteminis testavimas (system testing)*. Žiūrima, ar integruota sistema atitinka visus reikalavimus. Testuojama juodosios dėžės (black box) metodu, tokiu atveju nebūtina žinoti vidinio kodo dizaino ar logikos.

Ir galiausiai, remiantis visais reikalavimais, yra atliekamas *priėmimo testas (acceptance testing)*. Šiame etape dažniausiai užsakovas įvertina sukurtą sistemą. Priėmimo testą sudaro alfa ir beta versijos, pirmąją tikrina bet kokie vartotojai, antrąją - potencialūs vartotojai.

### Testavimo integravimas į internetinės sistemos kūrimą

Testavimas, atliekamas po sistemos įdiegimo (1 pav.) nėra efektyvus. Testavimas galėtų būti vykdomas lygiagrečiai su sistemos kūrimu. Pagal Hung Q. Nguyen, Bob Johnson, Michael Hackett, Robert Johnson (2003) testavimas gali būti atliekamas po sistemos sukūrimo, bet testų projektavimas ir įdiegimas atliekami lygiagrečiai sistemos kūrimui (žr.2 paveikslą).



2 pav. Lygiagretus sistemos kūrimas ir testavimas

Apibūdinsime sistemos testavimo proceso žingsnius. Projekto analitikas, susitikęs su užsakovu, turi apibrėžti būsimo sistemos funkcionalumą, vartojimo scenarijus, galutinį vartotoją. Surenkami pradiniai reikalavimai sistemai. Parengiami kokybės reikalavimų, scenarijų dokumentai.

Tuomet testuotojas, turėdamas analitiko sukurtus dokumentus ir testavimo plano šabloną, sudaro preliminarų testavimo planą:

- iškeliamas testavimo tikslas;
- nustatomi uždaviniai;
- apibrėžiami testavimo metodai bei testavimo aplinka;
- numatomas testavimo laikas ir funkcionalumo apimtys.

Toliau seka konkretaus plano sudarymas [6]. Testavimo planas apibūdina požiūrį į sistemos vystymą, integraciją, apribojimus ir priėmimo testavimą (acceptance testing). Turi būti nurodomi tokie reikalavimai (specifikacija):

- testavimo filosofija;
- testų tipai: pasirinkta kokie testai bus atliekami kuriamoje sistemoje;
- testavimo aplinka, įrankiai ir programinė įranga;
- išteklių ir grafikas, reikalingi sukurti/vykdyti testus;
- testavimo variantai, duomenys ir tikėtinų rezultatų specifikacija;
- paskirstomos testuotojų pareigos (jei daugiau nei vienas testuotojas).

Šis planas sudaromas visų projekte dalyvaujančių žmonių pagalba. Testavimo planas naudojamas atliekant tokius testavimus: sistemos, posistemės, programos sujungimo, modulių ir vartotojo priėmimo. Jis sudarytas iš konkrečių nurodymų įvairių tipų testavimui, kuris bus atliekamas per visą projekto gyvavimo ciklą.

Pagrindiniai projekto plano privalumai:

- padeda užsakovui suprasti kaip sistema bus testuojama;
- vadovaujantiems žmonėms padeda tinkamai paskirstyti laiką ir resursus testavimui;

- įvairios testuotojų grupės geriau suvokia kas yra iš jų reikalaujama. Prieš pradėdant testavimą, reikalingas vartotojo sąsajos prototipas. Sudarius planą, reikia nustatyti testavimo aplinką į kurią įeina:

- naudojami įrankiai, programinė įranga;
- tinklo reikalavimai;
- naudojama techninė įranga;
- saugumas (kokie vartotojai gali naudotis sukurta sistema, prie kokių funkcijų turi prieigą);
- dokumentacija.

Kiekvienas testuotojas, dirbantis su projektu, turi žinoti savo pareigas ir atsakomybę. Jei dirba daugiau nei vienas testuotojas, vienas yra vyresnysis, kuris atsakingas už kitų darbą. Darbo grafikas sudaromas iš anksto ir testuotojas turi būti pakankamai kompetentingas, kad atliktų savo pareigą.

Esant sudarytam vartotojo sąsajos prototipui, jį taip pat reikia ištestuoti. Pagal J.Nielsen, žinomą tinklalapių vartotojiškumo guru, ir grafinę vartotojo sąsają (Sanjay J. Koyani, Robert W. Bailey Ph.D, Janice R. Nall, 2004) yra labai svarbi. Kadangi šiais laikais tinklalapių sparčiai daugėja, tad vartotojai tapo ypač išrankūs. Bet kokia smulkmena, erzinti vartotoją, gali būti svarbus faktorius klientų praradimui. Taip pat labai svarbu, kad esamas funkcionalumas atitiktų scenarijuose numatytą funkcionalumą. Stengiamasi išvengti tokių klaidų, kaip:

- visiško išėjimo iš tinklalapio nuorodos pagalba. Vartotojas, paspaudęs nuorodą, visiškai išėina iš tinklalapio. Tai nėra patogu vartotojui (nes jam iš naujo reikia suvesti URL adresą ir grįžti į norimą puslapį) ir užsakovui (galimi klientų praradimo atvejai);
- konkretaus funkcionalumo nebuvimas. Pavyzdžiui, tituliname puslapyje parašyta, kad šiame tinklalapyje galima turėti keletą skirtingų sąskaitų to paties vartotojo vardu, tačiau pabandžius tai padaryti, sistema praneša klaidą;
- nuoseklumo (consistency) nebuvimas. Viename lange sukurta paieškos forma tam tikru principu, kitame lange – kitokiu. Šiuo atveju vartotojas gaišta laiką, norėdamas išsiaiškinti kaip naudotis sistema;
- nelogiškos nuorodos arba apskritai nuorodų trūkumas. Pvz., pasitaiko reklamų, kurios visai nesiderina su esamo tinklalapio kontekstu ir erzina vartotoją. Arba suradus straipsnį dominančia tema nėra nuorodų į panašaus pobūdžio straipsnius.

Įvedamų duomenų srities testavimas – pagrindinė testavimo dalis, tikrinant sistemos funkcionalumą. Šie duomenys yra iš anksto apibrėžti ir dokumentuojami. Testuojama pagal grafinę vartotojo sąsają, t.y. juodosios dėžės testavimo metodu. Išskiriami validūs ir nevalidūs variantai. Validūs variantai išsaugomi duomenų bazėje, o nevalidžių duomenų atveju turi būti pranešama apie klaidą. Testavimo variantų (test cases) sudarymui, reikia turėti keletą paruoštų dokumentų. Pirmiausia, reikalingas testavimo variantų sudarymo dokumento šablonas. Tuomet turi būti paruošti scenarijai. Scenarijus – tai konkretūs žingsniai, kuriuos reikia atlikti konkrečiam vartotojui, norint įvykdyti tam tikrą užduotį. Svarbu pažymėti, kad gali būti kelių tipų vartotojai, kurie atlieka skirtingas užduotis. Taip pat reikalingas vartotojo sąsajos prototipas. Tai – grafiškai pavaizduotas sistemos veikimo planas (gali būti įvairios formos, sąsajos). Į prototipą yra įtraukiamos įvedamų duomenų sritys. Geriausia, kai įvedamų duomenų sritys yra pavaizduojamos kartu su prototipu, tokiu atveju aiškiai matomi įvesties laukai ir jų sritys. Norint sudaryti efektyvesnes testavimo variantų grupes, naudojami scenarijai ir grafinis programinės įrangos modulio atvaizdavimas. Tai leidžia išdėstyti testavimo variantų grupių seką.

Sudarinėjami testavimo variantai yra dokumentuojami turimame šablone. Jame nurodyti reikiami įvesties duomenų laukai ir juos atitinkančios ekvivalentumo klasės. Pagal ekvivalentumo klases yra sudaromi testavimo variantai kiekvienam laukui. Pasirenkamos validžios klasės ir pagal jas sudaromas validus testavimo variantas. Jei validūs testavimo variantai yra korektiški, jie išsaugomi sistemoje. Tai patvirtina programinės įrangos korektišką veikimą ir leidžia dirbti turimu funkcionalumu. Tačiau jei validūs duomenys nėra išsaugomi, tuomet šias klaidas reikia ištaisyti pirmiausia, kad būtų galima testuoti susijusias formas.

*Scenarinis testavimas* tai toks testavimo būdas, kai testuojama pagal sudarytus scenarijus. Scenarinis testavimas atliekamas juodosios dėžės metodu (black-box testing), t.y. testuojama per vartotojo sąsają. Šiuo testavimo būdu patikrinamas sistemos funkcionalumas. Scenarijus sudaryti turi labiausiai patyrę ir geriausiai išmanantys sistemą darbuotojai. Norint sudaryti scenarijus testuojamai sistemai ar jos daliai, reikia žinoti sistemai keliamus reikalavimus. Juose turi būti nurodoma, koks funkcionalumas bus tikrinamas.

Pagal [1], idealaus scenarijaus charakteristika:

- testas yra paremtas siužetu pagal tai, kaip programa bus naudojama;
- siužetas turi motyvą;
- siužetas yra tikėtinas. Tai ne tik gali įvykti realiame pasaulyje, užsakovas manys, kad tai iš tikro įvyks;

- į siužetą įeina sudėtinės (kompleksinės) aplinkos, duomenų aibės;
- rezultatus yra lengva įvertinti. Tai svarbu visiems testams, bet ypatingai svarbu scenarijam testavimui, nes patys scenarijai yra sudėtingi.

Testavime didžiausias dėmesys skiriamas sistemos funkcionalumo patikrinimui. Lygiagrečiai su kitais testais, reikia planuoti ir sistemos saugumo, reagavimo į stresines situacijas, apkrovos testus.

*Apkrovos (load) testavimas* yra labai svarbus internetinei sistemai, kuria naudosis daug vartotojų vienu metu. Apkrovai patikrinti naudojami automatizuoti įrankiai. Pirmiausiai įrašomi testai tikrinamai sistemai. Norint patikrinti sistemos reakciją į apkrovą, yra imituojamas daugybės vartotojų naudojimas sistema vienu metu. Nurodoma, kokie testai bus paleisti, kiek vartotojų, kurį laikotarpį sistema bus apkrauta. Apkrovos testus galima atlikti su MS Visual Studio programa. Rezultatai yra pateikiami grafiškai, tokiu būdu patogų matyti, kaip sistema reaguoja į vis kitokį apkrovimą. Kai sistemai apkrova yra daug didesnė nei tikimasi, norint patikrinti sistemos reakciją į maksimalią apkrovą bei pažiūrėti jos stabilumą, yra atliekamas stresinis testavimas (stress testing). Tikimasi klaidingų sistemos rezultatų. Sunku pasakyti, kur yra riba tarp apkrovos ir stresinio testavimo, tačiau norint patikrinti sistemą šiais būdais, reikia žinoti sistemai keliamus reikalavimus.

*Apimties testavimas* (volume testing) atliekamas norint patikrinti duomenų apimtį. Tai gali būti duomenų bazės dydis arba sąsajos bylos dydis. Norint ištestuoti tam tikrą duomenų bazės dydį, reikia padidinti duomenų bazę iki reikiamo dydžio, tada žiūrėti kaip sistema veikia su tokiu duomenų kiekiu. Tai labai svarbu internetinei sistemai.

*Atlikimo testavimas* (performance testing) – vertinama, kaip veikia galutinis sistemos variantas. Šis būdas naudojamas, kai nėra detalios sistemos ar komponentų specifikacijos.

Taip pat reikia patikrinti sistemos *patikimumą, saugumą* (server). Galima naudoti įvairias automatizuotas saugumo testavimo programas.

Labai svarbus yra *pakartotinis testavimas*. Pakartotinio testavimo tikslas yra patikrinti, ar programa veikia korektiškai. Tikrinama, ar po pakeitimų neatsirado naujų klaidų, ar pataisytos jau žinomos klaidos. Pakartotinis testavimas atliekamas, turint dokumentuotus testavimo variantus ir scenarijus. Naudojama programinė įranga testų automatizavimui, pvz., MS Visual Studio programa. Pakartotinis testavimas turi būti atliekamas po kiekvienos programos naujos versijos sukūrimo.

Paskutiniu metu įrankiai, kurie padeda programuotojams kurti taikomąsias programas su grafinė vartotojo sąsaja, labai pagerino programuotojų darbo našumą. Tai padidino reikalavimus testuotojams, į kuriuos žiūrima kaip į programinių produktų pristatymo trukdžius. Testuotojų prašoma patikrinti vis daugiau ir daugiau kodo per vis mažesnę laiką. Testuojant rankiniu būdu (manual testing), sugaištama daug laiko, vienas iš būdų pagreitinti testavimą – testų automatizavimas. Bet ir tuomet, esant skirtingoms programų versijoms, naujos savybės turi būti vėl testuotos rankiniu būdu. Dabar yra tokių įrankių, kurie padeda testuotojams automatizuoti grafinės vartotojų sąsajos testavimą, o tai sumažina testavimo laiką bei kainą. Kiti testų automatizavimo įrankiai padeda padaryti atlikimo testus [8].

Internetinėje sistemoje įvesties duomenys yra gaunami įvykių dėka. Įvykiai - vartotojų atliekami veiksmai, tokie, kaip pelės judinimas ir mygtuko paspaudimas ar duomenų įvedimas klaviatūros pagalba. Testuoti tokias programas yra gana sudėtinga, nes, norint peržvelgti įvairias įvykių sekas ir kombinacijas, reikia atlikti daug darbo. Vien visų galimų įvykių kombinacijų identifikavimas gali būti tikras iššūkis, nes kai kurie veiksmai gali iššaukti daugialypius (pakartotinus/sudėtinius) įvykius.

Internetinių sistemų funkcinių testų automatizavime naudojamos dviejų įrankių specifikacijos: WebScriptRecorder ir WebScript. Web Script Recorder įrankis, skirtas atliekamo testo scenarijui (script) sugeneruoti. Tą testo scenarijų vėliau galima paleisti vykdyti su WebScript įrankiu.

Pagrindinės atliekamos įrankio funkcijos:

- testų scenarijų kodo sugeneravimas;
- elementų, neturinčių ID (identifikatorių), unikalios ID suradimas;
- atliekamų funkcijų sąrašo išvedimas, kurį vėliau naudos WebScript.

WebScript interpretuoja WebScriptRecorder sugeneruotą scenarijų. Tačiau vien testo įrašymo ir jo paleidimo neužtenka, norint nustatyti, ar funkcionalumas veikia teisingai. Reikia įrašyti ir palyginti įvestus duomenis. Galima integruoti ir papildomus įrankius, pvz., įrankį, kuris seka duomenų bazės pasikeitimus, momentines duomenų bazės nuotraukas tik įrašius ir paleidus testą. Tada galima palyginti rezultatus ir nustatyti, ar pakitęs funkcionalumas.

Atliekant testus, būtina rasti klaidas, tvarkingai jas dokumentuoti. Dokumentuojant rastą klaidą, turi būti nurodoma klaidos situacija, vieta, atlikti žingsniai (kartais

klaidos pasirodo, tik atlikus tam tikrų žingsnių seka), prioritetai kitų klaidų atžvilgiu ir kita informacija, kuri gali būti svarbi programuotojui.

Testuojant pagal sukurtus testavimo variantus, patogu tame pačiame dokumente žymėti, kokios klaidos rastos. Klaidas pagal jų svarbą galima skirstyti į:

- sistemos/taikomosios programos sugedimas – aukščiausio prioriteto,
- funkcionalumo klaidos – vidutinio prioriteto,
- susijusios su grafine vartotojo aplinka (spalvos, įvedimo laukų networka, gramatinės klaidos) – smulkios.

Įvairūs šaltiniai klaidas skirsto skirtingai. Svarbu pažymėti, kad kiekviename projekte klaidų prioritetas priklauso nuo to, kiek klaidos kenkia sistemos funkcionalumui, ar yra neatitikimų užsakovo reikalavimams. Pakartotinio testavimo metu patikrinama, ar ištaisytos visos, ar bent dalis buvusių klaidų. Dažnai tai atliekama automatizuotu būdu, pasitelkiant įvairius automatizavimui skirtus įrankius. Toks testavimas atliekamas po kiekvienos naujos programos versijos sukūrimo, norint patikrinti senas ir naujai atsiradusias klaidas.

Labai svarbu validuoti puslapius, t.y. užtikrinti, kad puslapio kodas atitinka W3C (World Wide Web Consortium), arba pasaulinio tinklo konsorciumo nustatytus standartus. Jei puslapiai yra parašyti validžiu kodu, tuomet jie sistemingai bendraus su naujausiomis naršyklėmis. Sutvarkius nestandartinį kodą, puslapiai tampa lankstūs ir lengviau tvarkomi. Tvarkingas kodas išsaugo gerą tinklo pralaidumą ir tai užtikrina greitesnį tinklalapio startavimą naršyklėje. Validuoti reikėtų ne tik sukurtus tinklalapius, bet ir modifikuotus. Tad geras įprotis yra validuoti ne tik sukurtus puslapius, bet ir modifikuotus. Kuriant naujus puslapius, rekomenduotina juos ištestuoti skirtingomis naujausiomis naršyklėmis:

- Internet Explorer 7.0 (Windows), Internet Explorer 6.0 (Windows);
- Internet Explorer 5.2 (Mac);
- Firefox 2.0 (Windows ir Mac);
- Safari (Mac);
- Netscape 8 (Windows ir Mac);
- Opera 9 (Windows ir Mac).

Dar be to, reikėtų patikrinti puslapius ir senesnėmis naršyklėmis, pavyzdžiui Netscape 4.7. Nebūtinai visi puslapio elementai turi atrodyti vienodai visose naršyklėse, tačiau svarbu, kad organizaciniai santykiai ir turinio hierarchija išliktų tokia pati visose naršyklėse, ir internetinė sistema būtų tinkama naudoti.

### Išvados

1. Egzistuojančios programinės įrangos testavimo strategijos tinka ir internetinei sistemai. Internetinei sistemai labai svarbi yra grafinė vartotojo sąsaja. Ypatingas testavimo dėmesys turi būti atkreiptas į *sistemos saugumą, funkcionalumą, apimtį ir perkrovą*.

2. Juodosios dėžės metodas taikomas testavimo variantų sudarymui, scenarijų tikrinimui. *Didelė dalis internetinės sistemos testavimo gali būti atlikta pagal juodosios dėžės strategiją*.

3. Testavimas, atliekamas po sistemos įdiegimo, nėra efektyvus. Rekomenduotina *testavimą vykdyti lygiagrečiai su sistemos kūrimu*.

### Literatūra

1. Cem Kaner. *An Introduction to Scenario testing*. Prieiga per internetą 2008-03-20: <<http://www.testingeducation.org/a/scenario2.pdf>>.
2. Grigaitis S., 2008, *Web sistemų testavimas*. Prieiga per internetą 2008-03-20: [http://www.rubyonrails.lt/files/saulius\\_grigaitis\\_bdd\\_testing\\_with\\_rspec\\_presentation\\_ruby\\_conf\\_lt\\_1\\_2008.pdf](http://www.rubyonrails.lt/files/saulius_grigaitis_bdd_testing_with_rspec_presentation_ruby_conf_lt_1_2008.pdf)
3. Hung Q. Nguyen, Bob Johnson, Michael Hackett, Robert Johnson, 2003, *Testing Applications on the Web: Test Planing for Mobile and Internet Based Systems*. Wiley Publishing, Inc.
4. Ian Sommerville. *Software engineering*. Addison Wesley. 2001. Prieiga per internetą 2008-03-15: <<http://www.scribd.com/doc/240298/Ian-Sommerville-Software-Engineering-2000>>.
5. *Integration and Testing*. Prieiga per internetą 2008-03-15: <[http://www.cs.rit.edu/~hpb/Scia/Bridge\\_2005/intro\\_se\\_workshop\\_materials/s6\\_TestingPhases.pdf](http://www.cs.rit.edu/~hpb/Scia/Bridge_2005/intro_se_workshop_materials/s6_TestingPhases.pdf)>.
6. Klariti, *The Template Store. Test Plan*. Prieiga per internetą 2008-03-20: <<http://www.klariti.com/templates/Test-Plan-Template.shtml>>.
7. *Objektiškai orientuotų sistemų testavimas: esama padėtis*. Prieiga per internetą 2008-03-20: <<http://kopustas.elen.ktu.lt/~rsei/PT/OOT.ppt>>

8. *Programinės įrangos testavimas*. Prieiga per internetą 2008-03-20:  
<<http://kopustas.elen.ktu.lt/~rsei/SE4/18.Testavimas.ppt>>
9. Sanjay J. Koyani, Robert W. Bailey Ph.D, Janice R. Nall, 2004, *Research-Based Web Design & Usability Guidelines*. GSA.

## IMPROVEMENT OF INTERNET SYSTEMS TESTING

The main goal of testing is to find program errors. It is a process, which helps to evaluate the quality of software or the web system. Testing ensures that most of the bugs (fixing all bugs is probably not possible) will be fixed and the product will be highly qualified.

Developing a testing model separately from the system development is not effective. It is best to do this process in parallel: when the process of the system development is being started, the testing process should be initiated as well.

This article analyzes existing testing techniques and strategies. The testing plan is suggested as a part of the web systems development process. The testing model is developed in accordance with the theory examined and practices done. The tool prototype for the automatization of the functional testing is also suggested.