



# Kauno technologijos universitetas

Informatikos fakultetas

T120B162 Programų sistemų testavimas

---

Ugnius Rinkevičius

Studentas

Martynas Kuliešius

Studentas

Nedas Liaudanskis

Studentas

---

Dėst. Lukas Arlauskas

# Turinys

<b>Lab1. Testavimo veiklų planavimas</b>	<b>3</b>
Darbo tikslas	4
Pasiruošimas	4
Informacija	4
Darbo užduotys	5
Komentarai	5
Darbo gynimas	5
Gynimui pasiruošimo klausimai	5
Nuorodos	5
Įvadas	6
Testavimo apimtis	6
Testavimo strategijos	6
Pradinės sąlygos	6
Testavimo prioritetai	7
Testavimo tikslai	7
Testavimo technikos	7
Rolės ir atsakomybės	8
Rezultatai	8
Testavimo aplinka	8
Testų scenarijai	8
Testų vykdymas	10
Testavimo rizikos	10
Testavimo atvejai Qase sistemoj	11

## Lab1. Testavimo veiklų planavimas

### Darbo tikslas

Sudaryti programinės įrangos testavimo planą.

### Pasiruošimas

Yra pasirinktas programinės įrangos projektas. Turimas ar bus sukurtas jo programinis kodas. Projektas gali būti kuriamas įvairiomis programavimo kalbomis naudojant įvairias technologijas.

### Informacija

Testavimo planas – dokumentas kuri sistematiškai apibrėžia kaip bus testuojamas programinė įranga. Egzistuoja įvairūs testavimo planų šablonai. Daugumoje jie visi aprašo panašius dalykus. Vienas iš šablonų yra IEEE 829-2008. Pagrindinės testavimo plano dalys yra:

1. Testavimo apimtis – Apibrėžiama programinės įrangos apimtis, apibūdinama kas ketinama testuoti.
2. Testavimo strategijos – nurodoma kokios testavimo strategijos bus taikomos, pagrindžiama kodėl.
3. Pradinės sąlygos – nurodoma, kas turi būti atlikta, kad būtų galima pradėti vykdyti testavimo veiklas.
4. Testavimo prioritetai – sudaromas testavimo veiklų programinių komponentų testavimui prioritetų sąrašas.
5. Testavimo tikslai – apibrėžiami testavimo tikslai nurodant kokius sistemos aspektus svarbu patikrinti..
6. Testavimo technikos – nurodoma kokios testavimo technikos bus taikomos apibrėžtiems tikslams pasiekti.
7. Rolės ir atsakomybės – apibrėžiama testavimo procese dalyvaujanti komanda ir jos atsakomybės.
8. Rezultatai – kokie yra laukiami testavimo rezultatai, kaip jie bus dokumentuoti, pateikti.
9. Testavimo aplinka – nurodoma kokia programinė ir techninė įranga reikalinga testavimo veikloms atlikti. Kokiose įrangos konfigūracijose reikės išbandyti testuojamą programinę įrangą.
10. Testų scenarijai – apibrėžiami testų scenarijai, reikalingi išsikeltiems tikslams pasiekti ir sukurti pagal pasirinktas testavimo technikas.
11. Testų valdymas – apibrėžiama, kaip bus valdomi testai, kur registruojami. Apibrėžiama kur ir kaip bus registruojami defektai, kaip stebimas testavimo procesas.
12. Testavimo tvarkaraštis – pateikiamas testavimo veiklų atlikimo tvarkaraštis.
13. Testavimo rizikos – numatoma kas galėtų sutrikdyti testavimo veiklų vykdymą, apibrėžiama kaip minimizuojama rizika.

## Darbo užduotys

1. Pasirinkti programinę įrangą testavimui, trumpai aprašyti jos funkcionalumą (panaudos atvejai, funkcijų sąrašas, architektūra)
2. Iškeliami testavimo tikslai, nusprendžiama kokios technikos bus jiems pasiekti taikomis.
3. Apibrėžiamos testavimo aplinkos.
4. Užpildomas testavimo plano dokumentas atsižvelgiant į pasirinktus testavimo tikslus ir technikas.
5. Sukuriami/suplanuojami testų scenarijai.
6. Suplanuoti testų scenarijai patalpinami pasirinktoje testų valdymo sistemoje (pvz. qase.io) arba užrašomi testų užrašymo kalba (pvz. Gherkin naudojant Cucumber įrankį).

## Komentarai

1. Sekančiuose darbuose visų suplanuotų testavimo veiklų neprivaloma atlikti.
2. Sekančiuose darbuose vykdomo tik nedidelė dalis suplanuotų testavimo veiklų.
3. Planą galima pateikti lietuvių ar anglų kalbomis.

## Darbo gynimas

1. Sukurtas testavimo plano dokumentas.
2. Sukurti testai suvesti į testų valdymo įrankį (jei reikia, testų valdymo įrankį reikia įsidięgti).
3. Testavimo plano dokumentas įkeltas į Moodle.
4. Sukurti testai eksportuoti iš testų valdymo įrankio ir įkelti kaip dokumentas į Moodle).
5. Pasiruošta atsakyti į įvairius su darbu susijusius klausimus.

## Gynimui pasiruošimo klausimai

1. Kokiu tikslu kuriamas testavimo planas?
2. Kaip bus atliekami suplanuoti testų scenarijai?
3. Kada geriausia pradėti vykdyti suplanuotas testavimo veiklas?
4. Kodėl kaž kurios testavimo veiklos buvo įtraukti į planą, kodėl nebuvo įtraukti?
5. Kodėl kaž kurie testavimo tikslai buvo įtraukti į planą, kodėl nebuvo įtraukti?

## Nuorodos

1. IEEE 829-2008. IEEE Standard for Software and System Test Documentation.
2. <https://specflow.org/learn/gherkin/>
3. <https://qase.io/>

# Įvadas

Šis dokumentas nurodo testavimo planą programinei įrangai „SignalR-Snake“.

## Testavimo apimtis

„SignalR-Snake“ programinės įrangos testavimui planuojame atlikti šiuos testus:

1. Frontend testavimas: Užtikrinti, kad vartotojo sąsaja ir GUI veiktų teisingai, įskaitant gyvatės judėjimo ir žaidimo atvaizdavimo valdiklius.
2. Backend testavimas: Patikrinkite realaus laiko ryšį naudojant SignalR, žaidėjų sesijų valdymą ir žaidimo logikos nuoseklumą.
3. Integracijos testavimas: Patikrinkite sklandžią kliento ir serverio sąveiką, ypač žaidžiant daugeliui žaidėjų.
4. Našumo testavimas: Išmatuokite žaidimo našumą skirtingomis tinklo sąlygomis ir su keliais žaidėjais.

## Testavimo strategijos

Testavimo strategija orientuota į svarbiausių sistemos funkcijų ir našumo tikrinimą. Naudosime:

1. Žaidimo logikai naudokite **vienetų testus**.
2. **Integracijos testai** SignalR ryšiui ir daugelio žaidėjų scenarijams.
3. **Rankiniai testai**: vartotojo sąsajos funkcionalumo ir žaidimo patirties.

## Pradinės sąlygos

Testavimo veiklai pradėti reikalingi šie veiksmai:

1. Visiškai paruoštas „SignalR-Snake“ prototipas.
2. Apibrėžti testavimo atvejai.
3. Veikiančios testavimo aplinkos ir reikalingi įrankiai.
4. Klaidų sprendimo tvarka.
5. Bandomoji aplinka sukonfigūruota su keliais klientais, kad būtų galima imituoti daugelio žaidėjų scenarijus.
6. Funkcinis tinklas, skirtas realaus laiko ryšio bandymams.

## Testavimo prioritetai

Testavimo veiklų prioritetai bus tokie:

1. Žaidimo logikos nuoseklumas.
2. Vartotojo sąsajos reakcija.
3. Realaus laiko ryšys su SignalR.
4. Kelių žaidėjų sinchronizavimas.
5. Našumas esant apkrovai.

# Testavimo tikslai

Programinės įrangos “fleiser/singlar-snake” testavimui, planuojame atlikti šiuos testus:

1. **Functional testai.** Vienetų testavimas, Client-Server komunikacijos testai, įrangos funkcijų testai.
2. **Performance testai.** Load testavimas - serverio apkrovimą simuliuojant, Latency testavimas - tikrinimas tarp kliento veiksmo ir serverio atsako.
3. **Usability testai.** UI ir UX testavimas
4. **Security testai.** Autentifikacija bei autorizacija.
5. **Stress testai.** Serverio perkrovimo testavimas, prisijungimo stabilumas
6. **Compatibility testai.** Skirtingų platformų bei versijų suderinamumas.
7. **Network testai.** Tinklo naudojimo matavimas. Paketų praradimo testavimas.
8. **Integration testai.** uhh
9. **Scalability testai.** Tikrinama ar galima sistemą praplėsti/ paruošti didesnėm apkrovom pridedant daugiau serverių arba paduodant kitą įrangą.
10. **Regression testai.** Atliekant pakeitimus patikrinti ar neatsirado naujų problemų
11. **Bug/Crash testai.** Tikrinti kas būna jeigu sistema užlūžta/ crashina.

# Testavimo technikos

Skirtingiems testams yra naudojamos skirtingos technologijos:

1. Unit testams bus ranka aprašomi testai didžiai daliai metodų bei funkcijų, kurios yra aprašytos programinėje įrangoje. Taip pat tam panaudosim “**NUnit**” arba “**xUnit**” pagalbinę programinę įrangą.
2. Performance testams papildomai galima naudoti TPL biblioteką arba “**Apache JMeter**” arba “**Gatling**” simuliuoti serverio apkrovą kuriant fiktyvius naudotojus. Latency testus galima atlikinėti pasitelkiant C# integruotais Stopwatch metodais.
3. UI ir UX testams galima naudoti “**Selenium**” kartu su “**NUnit/xUnit**” tikrinti funkcionalumą. Taip pat bus atliekamas rankinis testavimas, nes ne viską galima ištestuoti su papildomomis programomis
4. Autentifikaciją bei autorizaciją testuojant galima naudoti unit testus arba “**QWASP ZAP**” arba “**Burp Suite**”.
5. Stress testus galima simuliuoti pasitelkiant “**TPL**” arba “**Task.Delay()**” didinti serverio apkrovai.
6. Compatibility testus tikrinti platformų suderinamumui galime pasitelkti “**CI/CD pipelines**” per “**GitHub actions**” arba “**Azure DevOps**”
7. Network testavimui galima naudoti Wireshark įrankį analizuoti duomenų paketus.
8. Integration testams galima naudoti išorinius API kaip “**Moq**” tikrinti integraciją. **UHH**
9. Scalability testavimui galime aprašyti testus arba naudoti “**Azure Load Testing**” debesijos paslaugas.
10. Regression testavimą galima automatizuoti naudojant “**CI/CD**” įrankius kaip “**Azure DevOps**” arba “**Jenkins**”, tikrinti programos klaidas.

11. Bug/Crash testams galima aprašyti testus arba pasitelkti **“NUnit Assert.Throws”** išgauti klaidų pranešimus.
12. Rankinį testavimą atliekant bus tikrinami kompiliatoriaus pranešimai, kodo taisymo patarimai

## Rolės ir atsakomybės

Visus esamus ir būsimus darbus nutarėme pasidalinti tolygiai, todėl visų trijų studentų rolės ir atsakomybės bus padalintos taip pat tolygiai.

## Rezultatai

Po testavimo tikimasi, kad bus detalai aprašyta ir koreguota:

- Testavimo planas - papildomas laboratorinių darbų metu.
- Kodo keitimo ataskaita bei kiekvieno laboratorinio ataskaita.
- Testų tikėti rezultatai ir gauti rezultatai bus laboratorinių darbų atasakaitose.

## Testavimo aplinka

Testuojant „fleiser/singlar-snake“, kuriame naudojama C#, reikalinga ši programinė ir techninė įranga:

- **Operacinė sistema:** „Windows“, „Linux“ (su „.NET“ palaikymu), arba „macOS“. Testavimas turi būti atliktas keliose OS versijose, siekiant užtikrinti suderinamumą.
- **Procesorius:** „Intel i5“ ar aukštesnis arba analogiškas.
- **RAM:** Bent 8 GB, kad programa veiktų sklandžiai įvairiomis aplinkybėmis.
- **Programinė įranga:** „Visual Studio“ arba „JetBrains Rider“ kaip pagrindiniai IDE, „.NET Core SDK“, „NUnit“ arba „xUnit“ C# testavimo sistemoms, taip pat „Git“ versijų valdymui, Qase.io testų tvarkymui.
- **Techninė įranga:** Testavimas turėtų vykti įvairiose įrangos konfigūracijose: žemos, vidutinės ir aukštos klasės kompiuteriuose, kad būtų įvertintas našumas. Taip pat naudotinos virtualios mašinos ar konteineriai, siekiant patikrinti programos veikimą skirtingose aplinkose. „Selenium“ ar „Postman“ gali būti naudojami vartotojo sąsajos ir API testavimui.

## Testų scenarijai

Testavimo scenarijai kuriami atsižvelgiant į funkcinius ir nefunkcinius programos reikalavimus.

Pagrindiniai scenarijai:

**Funkciniai testai:** Patikrinti pagrindines funkcijas, tokias kaip:

- Gyvatės sukūrimas naudojant metodą **NewSnek()** ir tinkamas jos dalių generavimas (pagal failą „SnakeHub.cs“).
- Gyvatės valdymas, kaip ji juda ir kaip keičiasi jos pozicijos. Tai galima testuoti atliekant funkcijas **SendDir()** ir **MoveTimer\_Elapsed()** patikrinimus.
- Taškų skaičiavimas pagal surinktą maistą naudojant metodą **Score()**.
- Maisto generavimas ir patikrinimas, kaip gyvatė reaguoja į maisto surinkimą (Foods objektų sąveika).

**Našumo testai:** Testuoti, kaip greitai reaguoja „SignalR“ ryšiai, kai yra daug prijungtų naudotojų ir kaip tvarkomi keli gyvačių judėjimo atvejai.

**Kryžminės platformos testai:** Patikrinti, ar žaidimas veikia skirtingose naršyklėse ir operacinėse sistemose.

#### Testų scenarijus 01.0.1: Naujos gyvatės sukūrimas

*Aprašymas:* Testuojama gyvatės sukūrimo funkcija, kai naudotojas pasirenka vardą ir pradeda žaidimą.

*Pradiniai duomenys:* Pradinė būseną be sukurtos gyvatės, tuščias naudotojo vardas.

*Testavimo veiksmai:*

1. Naudotojas įveda vardą ir paspaudžia „Sukurti gyvatę“.
2. Gyvatė sukuriamas ir pridedama į bendrą gyvačių sąrašą.

**Testų atvejai:**

Naudotojo vardas	Laukiamas rezultatas	Komentaras
Player1	Gyvatė sukurta sėkmingai	Vartotojas Player1
Player2	Gyvatė sukurta sėkmingai	Naudotojas Player2 sukurtas
(tuščias laukas)	Gyvatė nesukurta, klaida	Pranešimas apie tuščią lauką

#### Testų scenarijus 01.0.2: Gyvatės judėjimas

*Aprašymas:* Testuojama gyvatės valdymo funkcija, kai naudotojas keičia gyvatės kryptį.

*Pradiniai duomenys:* Sukurta gyvatė pradiniam taške.

*Testavimo veiksmai:*

1. Naudotojas spausdina mygtukus, kurie keičia gyvatės kryptį (pvz., rodyklių klavišai).
2. Gyvatė turi judėti nurodyta kryptimi pagal naują kampą.

**Testų atvejai:**

Kryptis	Laukiamas rezultatas	Komentaras
0 laipsnių	Gyvatė juda į dešinę	Judėjimas į priekį
90 laipsnių	Gyvatė juda į viršų	Keičiama kryptis į viršų



180 laipsnių	Gyvatė juda į kairę	Atbulinis judėjimas
270 laipsnių	Gyvatė juda žemyn	Keičiama kryptis į apačią

## Testų vykdymas

Testai bus vykdomi remiantis parengtais testų scenarijais ir atvejais. Kiekvienas testas bus atliekamas pagal nustatytas instrukcijas, kad būtų patikrintas programos funkcionalumas ir užtikrintas reikalavimų atitikimas. Testuotojai bus atsakingi už tikslų scenarijų įgyvendinimą ir rezultatų stebėjimą.

### Testų rezultatų ataskaitos

Kiekvieno testo rezultatai bus užfiksuoti ir pateikti ataskaitų forma. Ataskaitose bus nurodoma, ar testas buvo sėkmingas, kokie rezultatai buvo gauti, bei kokie defektai ar problemos buvo nustatytos. Testų rezultatai bus dokumentuojami naudojant <https://qase.io/>.

## Testavimo rizikos

Pagrindinės rizikos, galinčios sutrikdyti testavimą:

- **Neužbaigti ar netikslūs reikalavimai:** Gali sukelti painiavą dėl tikslų ir testavimo veiklų. Šią riziką galima sumažinti anksti ir išsamiai analizuojant reikalavimus.
- **Techniniai nesklandumai:** Aparatinės įrangos gedimai, nesuderinamumas su „.NET“ ar IDE problemos gali trukdyti testavimui. Siekiant šios rizikos išvengti, būtina turėti atsarginius įrenginius ar parengtą virtualią aplinką.
- **Vėlavimai ruošiant aplinką:** Aplinkos nustatymo vėlavimai gali trukdyti testavimo pradžiai. Ši rizika mažinama tiksliai planuojant ir iš anksto nustatant prioritetus.
- **Testų duomenų trūkumas:** Jei testavimui neparengti reikiami duomenys arba duomenų kokybė yra prasta, testų rezultatai gali būti klaidingi ar nepakankami. *Rizikos mažinimas:* Laiku parengti ir išbandyti duomenų rinkinius, užtikrinti, kad jie atitinka tikrąsias darbo sąlygas.
- **Nepakankamas testų skaičius:** Jei nebus atlikta pakankamai testų arba bus testuojamos ne visos funkcijos ir scenarijai, gali likti nepastebėtos klaidos, dėl kurių programinė įranga nebus patikima arba nepatenkins vartotojų lūkesčių. Dėl per mažo testų kiekio gali būti praleisti kritiniai programos defektai, ypač rečiau naudojamose ar sudėtingesnėse funkcijose.

## Testavimo atvejai Qase sistemoj

# Testų Scenarijus 01.0.1: Gyvatės Kūrimas

-  Edit
-  Clone
-  Delete
-  AI Test Case

[General](#) Properties Runs History Defects Comments

**Description**

Šis testų scenarijus užtikrina, kad gyvatės kūrimo funkcija veiktų teisingai, kai vartotojas įveda savo vardą ir pradeda žaidimą.

**Pre-conditions**

Vartotojas turi turėti prieigą prie žaidimo sąsajos.

**Post-conditions**

Gyvate sėkmingai sukurta ir pridėta prie gyvačių sąrašo arba rodomas klaidos pranešimas.

**Steps**

Shown as table ☒

⌵	Action	Data	Expected result	Attachments
○ 1	Atidarykite žaidimo sąsają.	Tuščias gyvatėlės sąrašas.	•	
○ 2	Įveskite žaidėjo vardą „Sukurti gyvatę“ lauke.	Tuščias gyvatėlės sąrašas.	•	
○ 3	Paspauskite mygtuką „Sukurti gyvatę“.	Tuščias gyvatėlės sąrašas.	Gyvate sėkmingai sukurta ir pridėta prie gyvačių sąrašo arba rodomas klaidos pranešimas.	

## Testų Scenarijus 01.0.2: Gyvatės Judėjimas

 Edit Clone Delete AI Test Case[General](#)[Properties](#)[Runs](#)[History](#)[Defects](#)[Comments](#)

### Description

Šis testų scenarijus užtikrina, kad gyvatė judėtų teisingai, kai žaidėjas paspaudžia krypties klavišus.

### Pre-conditions

Gyvatė turi būti sėkmingai sukurta.

### Post-conditions





Gyvatė juda atitinkama kryptimi, kaip nurodyta.

### Steps

Shown as table ☒

⌵	Action	Data	Expected result	Attachments
○ 1	Paspauskite skirtingus krypties klavišus (aukštyn, žemyn, kairėn, dešinėn).	Paspaudymai	Gyvatė juda atitinkama kryptimi, kaip nurodyta.	

# Testų Scenarijus 01.0.3: Taškų Skaičiavimas

-  Edit
-  Clone
-  Delete
-  AI Test Case

[General](#) [Properties](#) [Runs](#) [History](#) [Defects](#) [Comments](#)

**Description**

Šis testų scenarijus tikrina, ar taškų skaičiavimo sistema veikia, kai gyvatė valgo maistą.

**Pre-conditions**

- Gyvatė turi būti sukurta.
- Maistas turi būti žaidimo lauke.

**Post-conditions**

Gyvatės ilgis ir taškai atnaujinami atsižvelgiant į suvartoto maisto kiekį.

**Steps**

Shown as table ☒

⌵	Action	Data	Expected result	Attachments
○ 1	Judinkite gyvatę link maisto.	Gyvatėlė yra gyvatėlių sąrašė. Maisto pozicijos.	Gyvatėlė juda link maisto.	
○ 2	Stebėkite, kaip gyvatė valgo maistą ir auga.	Gyvatėlė yra gyvatėlių sąrašė. Maisto pozicijos.	Gyvatės ilgis ir taškai atnaujinami atsižvelgiant į suvartoto maisto kiekį.	