Testare Sistemelor Software, Master IS, an 2, sem 2
An univ. 2023-2024
Curs: Florentin Ipate
Laborator: Radu Bobe

I. Obiective: Asimilarea principalelor concepte si tehnici de testare a sistemelor software, model checking si model-based testing.

II. Cerinte

Notarea se va face pe baza proiectului de laborator (50%) si proiectului de curs (50%).

Proiectul de laborator va fi individual si va fi prezentat la laborator conform programarii stabilite cu cadrul didactic de la laborator.

Proiectele de curs se vor efectua in echipe (dimensiunea echipei pentru fiecare tema este specificata mai jos) si vor fi prezentate la curs in saptamanile 6-12 conform programarii stabilite, astfel incat numarul maxim de studenti care prezinta in cadrul unui curs sa fie 10. Tema proiectului de curs va fi aleasa din lista de mai jos. Fiecare tema din lista poate fi aleasa de cel mult 3 echipe.

Prezentarea proiectului de curs va fi sub forma de slide-uri, insotite de demo-uri. La prezentare este necesara prezenta intregii echipe, fiecare student descriind principala sa contributie la proiect. Timpul alocat fiecarui student este de 10 minute. Atunci cand echipa considera ca nu toti membrii echipei au avut o contributie egala la realizarea proiectului, se va indica, in procente, contributia estimata a fiecaruia.

III. Proiect laborator

Sa se scrie un program in Java, precum si cerintele (specificatia) acestuia.

1. Pe baza cerintelor programului, sa se genereze date de test folosind a) *equivalence partitioning* b) *boundary value analysis* si c) *cause-effect graphing*. Sa se implementeze testele obtinute folosind JUnit.
2. Sa se stabileasca nivelul de acoperire realizat de **fiecare** dintre seturile de teste de la 1) a), b) si c) folosind unul dintre utilitarele de code coverage prezentate in [6]. Sa se compare si sa se comenteze rezultatele obtinute de cele trei seturi de teste.
3. Sa se transforme programul intr-un graf orientat si, pe baza acestuia, sa se gaseasca un set de teste care satisface criteriul *modified condition/decision coverage* (MC/DC).
4. Sa se scrie un mutant de ordinul 1 echivalent al programului.
5. Pentru unul dintre cazurile de testare de mai sus sa se scrie un mutant ne-echivalent care sa fie omorat de catre test si un mutant ne-echivalent care sa nu fie omorat de catre test.

Observatii:

- se poate folosi orice limbaj de programare care permite scrierea testelor unitare si folosirea unui tool de acoperire.
- se poate testa o metoda (functie, procedura) din program, insa trebuie mentionat clar acest lucru, descrisa functionalitatea acesteia, iar toate cerintele se vor rezolva pe baza aceleiasi metode.
- pe langa programul efectiv si testele implementate, se va realiza si o documentatie in care se vor descrie detaliat cum s-au obtinut rezultatele pentru fiecare din cerintele proiectului.

- Graful asociat programului va fi realizat cu un utilitar (de exemplu: Lucidchart, Draw.io, yEd, Microsoft Visio).

Referinte: [1-7]

IV. Teme proiecte curs

1. FSM based testing

- Prezentati problematica si principalele tehnici de FSM based testing: W-method, UIO, DS.
- Scrieti un program care implementeaza W-method si ilustrati aplicarea acesteia pe un exemplu.

Marime echipa: 3 studenti
Bibliografie: [8]

2. Search based testing pentru EFSM

- Prezentati metoda de generare de date de test prezentata in [9].
- Scrieti un program care implementeaza metoda de generare de date de test prezentata in [9] si ilustrati functionarea acesteia pe un exemplu.

Marime echipa: 2 studenti
Bibliografie: [9]

3. Automatic Evolutionary Test Suite Generation cu EvoSuite

- Prezentati utilitarul EvoSuite, care generează automat cazuri de testare cu aserțiuni pentru clase scris în cod Java folosind o abordare hibridă pentru generarea si optimizarea întregii suite de testare ce satisface un criteriu de acoperire.
- Ilustrati capabilitatile utilitarului folosind unul sau mai multe exemple create de echipa.

Marime echipa: 2 studenti
Bibliografie: [10,11]

4. Automatic Search-based Test Case Generation for Autonomous Systems cu AmbieGen

- Prezentati framework-ul AmbieGen, care generează automat cazuri de testare pentru sisteme autonome folosind o căutare evolutivă pentru a identifica cele mai critice scenarii pentru un sistem dat.
- Ilustrati capabilitatile utilitarului folosind unul sau mai multe studii de caz create de echipa.

Marime echipa: 3 studenti
Bibliografie: [12,13]

5. Testing Self-Driving Car Software Using Search-Based Procedural Content Generation cu AsFault

- Prezentati tool-ul AsFault, care generează automat teste virtuale pentru testarea sistematică a software-ului mașinilor cu conducere autonomă.
- Demonstrați capabilitățile utilitarului testând caracteristica de menținere a benzii a unui software de mașină cu auto-conducere, pentru care AsFault generează scenarii care o determină să iasă de pe șosea.

Marime echipa: 2 studenti
Bibliografie: [14,15]

6. **Model based testing cu JSXM**
   - Prezentati limbajul de modelare bazat pe stream X-machines precum si principalele facilitati oferite de utilitarul JSXM: animarea modelului, generarea testelor pe baza modelului de forma stream X-machine, implementarea testelor in JUnit.
   - Ilustrati aceste capabilitati folosind unul sau mai multe exemple create de echipa.

   Marime echipa: 3 studenti
   Bibliografie: [16-18]
   Observatie: pentru instalarea tool-ului se va consulta documentul „JSXM installation instructions".

7. **Model based testing for Cloud Services cu Broker@Cloud**
   - Prezentati limbajul de modelare bazat pe stream X-machines precum si principalele facilitati oferite de utilitarul Broker@Cloud verification and testing tool: verificare, test generation si test grounding.
   - Ilustrati aceste capabilitati folosind unul sau mai multe exemple create de echipa.

   Marime echipa: 3 studenti
   Bibliografie: [19,20]

8. **Model based testing cu GraphWalker**
   - Prezentati principalele facilitati oferite de utilitarul GraphWalker: generarea testelor din modelul masinilor cu stari finite (FSM) folosind algoritmi de traversare precum A* sau parcurgere random cu diverse criterii de acoperire (state, edge, requirement).
   - Ilustrati aceste capabilitati folosind unul sau mai multe exemple create de echipa.

   Marime echipa: 2 studenti
   Bibliografie: [21,22]

9. **Modelare si analiza cu Event-B, Rodin**
   - Prezentati capabilitatile de modelare si analiza ale limbajului Event-B: evenimente, rafinare, demonstrare automata a proprietatilor.
   - Ilustrati aceste capabilitati folosind unul sau mai multe exemple create de echipa.

   Marime echipa: 3 studenti
   Bibliografie: [23,24]

10. **Model checking cu Event-B, Rodin si Pro B**
    - Prezentati abordarea de model checking folosind Event-B, Rodin si Pro B descrisa in [25].
    - Ilustrati aceasta abordare folosind unul sau mai multe exemple create de echipa (diferite de exemplul din articolul [25]).

    Marime echipa: 3 studenti
    Bibliografie: [25,26]

11. **Model checking cu NuSMV**
    - Prezentati pe scurt logica LTL si logica CTL.
    - Ilustrati capabilitatile de model checking (verificare de proprietati LTL si CTL) folosind unul sau mai multe exemple create de echipa.

    Marime echipa: 3 studenti
    Bibliografie: [27,28]

12. **Modelare, simulare si verificare cu membrane systems si kPWorkbench**
    - Prezentati formalismul kP systems si capabilitatile de modelare si verificare formala a suitei kPWorkbench.
    - Ilustrati aceste capabilitati folosind unul sau mai multe exemple create de echipa.

    Marime echipa: 3 studenti
    Bibliografie: [29-31]

V. Bibliografie

[1]. Functional testing, material de curs.
[2]. Structural testing, material de curs.
[3]. Mutation testing, material de curs.
[4]. http://www.vogella.com/tutorials/JUnit/article.html
[5]. https://cs.gmu.edu/~offutt/mujava/
[6]. https://www.softwaretestinghelp.com/code-coverage-tools/
[7]. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=4755ebf92ea2bfcbd603ebf3ab14e852dd64e978
[8]. Aditya P. Mathur. Foundations of Software Testing, Pearson Education 2008. Chapter 6: Test Generation: FSM Models.
[9]. Raluca Lefticaru, Florentin Ipate: Automatic State-Based Test Generation Using Genetic Algorithms, in SYNASC, 188-195 (2007). http://www.ifsoft.ro/~florentin.ipate/publications/SYNASC%202007%20Automatic%20State-Based%20Test%20Generation%20Using%20Genetic%20Algorithms.pdf
[10]. https://www.evosuite.org/
[11]. Gordon Fraser, Andrea Arcuri: EvoSuite: automatic test suite generation for object-oriented software. SIGSOFT FSE 2011: 416-419 https://www.researchgate.net/profile/Andrea-Arcuri-3/publication/221560749_EvoSuite_Automatic_test_suite_generation_for_object-oriented_software/links/595cd3e50f7e9b3aefaddbd0/EvoSuite-Automatic-test-suite-generation-for-object-oriented-software.pdf
[12]. https://arxiv.org/abs/2301.01234
[13]. https://github.com/swat-lab-optimization/AmbieGen-tool
[14]. https://www.researchgate.net/profile/Alessio-Gambi/publication/335362266_AsFault_Testing_Self-Driving_Car_Software_Using_Search-Based_Procedural_Content_Generation/links/5d70bd4792851cacdb21a956/AsFault-Testing-Self-Driving-Car-Software-Using-Search-Based-Procedural-Content-Generation.pdf
[15]. https://github.com/alessiogambi/AsFault
[16]. Florentin Ipate, Mike Holcombe: Specification and testing using generalized machines: a presentation and a case study, Software Testing, Verification and Reliability, 8, 61-81 (1998) http://www.ifsoft.ro/~florentin.ipate/publications/STVR98%20-%20Specification%20and%20testing%20using%20generalised%20machines.pdf
[17]. http://www.jsxm.org/
[18]. Dimitris Dranidis, Konstantinos Bratanis, Florentin Ipate: JSXM: a tool for automated test generation, in SEFM, 352-366 (2012). http://www.ifsoft.ro/~florentin.ipate/publications/SEFM2012_CR.pdf
[19]. http://staffwww.dcs.shef.ac.uk/people/A.Simons/broker/

[20]. Lefticaru, R. and Simons, A.J.H. (2015) X-Machine Based Testing for Cloud Services. In: Ortiz, G. and Tran, C., (eds.) Advances in Service-Oriented and Cloud Computing. Workshops of ESOCC 2014, September 2-4, 2014, Manchester, UK. Lecture Notes in Computer Science, 508 . Springer , pp. 175-189. https://eprints.whiterose.ac.uk/98321/1/XMtestingforcloud.pdf
[21]. https://graphwalker.github.io/
[22]. Muhammad Nouman Zafar, Wasif Afzal, Eduard Enoiu, Athanasios Stratis, Aitor Arrieta, Goiuria Sagardui: Model-Based Testing in Practice: An Industrial Case Study using GraphWalker. ISEC 2021: 5:1-5:11 http://ebiltegia.mondragon.edu/xmlui/bitstream/handle/20.500.11984/5411/Model-Based%20Testing%20in%20Practice_An%20Industrial%20Case%20%20Study%20using%20GraphWalker.pdf?sequence=1&isAllowed=y
[23]. Jean-Raymond Abrial. Modeling in Event-B: System and Software Engineering.
[24]. http://www.event-b.org/
[25]. Adrian Turcanu, Talal Shaikh and Cristina Nicoleta Mazilu, On Model Checking of a Robotic Mechanism. https://scholars.direct/Articles/robotics/jra-4-018.pdf
[26]. https://www3.hhu.de/stups/prob/
[27]. https://www.cs.utexas.edu/users/moore/acl2/seminar/2010.05-19-krug/slides.pdf
[28]. https://nusmv.fbk.eu/
[29]. https://github.com/Kernel-P-Systems/kPWorkbench
[30]. R. Lefticaru, M. E. Bakir, S. Konur, M. Stannett, F. Ipate: Modelling and Validating an Engineering Application in Kernel P Systems, Int. Conf. on Membrane Computing 2017, LNCS, 183-195, 2017. http://www.ifsoft.ro/~florentin.ipate/publications/EngApl-CMC2017.pdf
[31]. S. Konur, L. Mierla, F. Ipate, M. Gheorghe: kpworkbench: A software suit for membrane systems, SoftwareX, 11: 100407, 2020. http://www.ifsoft.ro/~florentin.ipate/publications/SoftwarexKMIG.pdf