

Testarea Performanței Interfețelor Grafice

Olaru Laura-Elena

1. Abstract

Link istoric github:

<https://github.com/Laura-ElenaOlaru/ResearchProject/commits/main>

De cele mai multe ori, un programator caută un instrument de testare a interfeței grafice care poate oferi nu numai feedback legat de funcționarea corectă a aplicației, dar și legat de performanța aplicației, pentru a ști ce trebuie îmbunătățit, aplicațiile performante fiind mai valoroase în producție. În acest articol testez și compar rezultatele testării cu diferite instrumente și tehnologii și în final concluzionez care sunt cele mai eficiente și complete instrumente de testare a interfeței grafice în funcție de metricile de performanță rezultate în urma testării.

2. Clasificare

2.1 AMS Mathematical Subject Classification

68Nxx Software

68N19 Other programming techniques (object-oriented, sequential, concurrent, automatic, etc.)

2.2 ACM Computing Reviews Categories and Subject Descriptors

D. Software

D.2 SOFTWARE ENGINEERING

D.2.5 Testing and Debugging

3. Introducere

De fiecare dată când un programator proiectează o aplicație, în final are nevoie să știe dacă aplicația sa funcționează în mod corect și conform așteptărilor, pentru ca aceasta să intre în producție. Din moment ce aproape orice aplicație ce intră în producție prezintă o interfață grafică ce poate fi folosită de către utilizatori, este important ca acea interfață grafică să funcționeze corespunzător scopului în care a fost implementată, fără ca utilizatorii să întâmpine probleme atunci când o folosesc. Pentru ca acest aspect să fie îndeplinit, programatorii își pot da seama din timp dacă aplicația lor funcționează și nu prezintă erori la utilizare, prin intermediul testării interfeței grafice. Există diferite moduri prin care un programator poate testa interfața grafică și există și foarte multe instrumente de testare care oferă mai multe sau mai puține feature-uri. De cele mai multe ori, un programator caută acel instrument care poate oferi nu numai feedback legat de funcționarea corectă a aplicației, dar și legat de performanța aplicației, pentru a ști ce trebuie îmbunătățit, aplicațiile performante fiind mai valoroase în producție. Fiecare instrument de testare poate prezenta anumite metrice de performanță care pot ajuta în diferite moduri în procesul de perfecționare a aplicației pentru intrarea sa în producție. Din moment ce nu există niciun ghid care să indice și să compare instrumentele folosite în testarea interfețelor grafice în ceea ce privește aceste metrice, în acest articol voi prezenta care sunt instrumentele cele mai complete din punct de vedere a ceea ce oferă când vine vorba de testarea performanței unei interfețe grafice.

4. Descriere abordare originală

4.1 Date folosite

Aplicația testată (projyproject) este o aplicație de gestionare a utilizatorilor. Datele folosite pentru testarea performanței aplicației sunt interfețele grafice de pe mai multe tipuri de dispozitive: IOS, Android, Web. De asemenea, aceste interfețe grafice sunt formate din mai multe componente: ecrane diferite, împreună cu navigarea dintre ele, diferitele funcționalități care trebuie să funcționeze conform așteptărilor și funcționarea corectă a widget-urilor aplicației.

4.2 Experimente

Experimentele care se fac constau în măsurarea performanței pe diferitele dispozitive. Pentru setul de date artificial, aplicația este testată folosind `console.firebase.google.com` deoarece aplicația este implementată în Flutter, iar platforma oferă mai multe feature-uri de performanță. În ceea ce privește setul de date real, pentru a testa o interfață grafică din producție am folosit Robotic Process Automation (RPA), o tehnologie pe care am putut să o aplicăm cu tool-ul de la UiPath pentru a testa performanța aplicației YouTube.

4.3 Model matematic

Fiabilitatea software-ului [1] este definită ca probabilitatea funcționării software fără defecțiuni pentru o anumită perioadă de timp într-un mediu specificat, ceea ce ajută în calculul probabilității ca testele făcute pe o interfață grafică să funcționeze. Acesta este prezentată în model matematic după cum urmează.

$$R(t) = P(T > t) \quad (4.1)$$

unde T este variabila aleatoare desemnată ca timp de eșec, P este probabilitatea și t este durata de timp. Deoarece T este variabila aleatoare, are o funcție de distribuție și o funcție de densitate de probabilitate. Prin urmare, SR poate fi definit și ca

$$R(t) = \int_t^{\infty} f(s)ds \quad (4.2)$$

unde f este funcția de densitate de probabilitate. Probabilitatea de eșec este dată ca

$$F(t) = 1 - R(t) = \int_0^t f(s)ds \quad (4.3)$$

și rata de eșec (FR) este următoarea

$$FR = \frac{F(t + \Delta t) - F(t)}{\Delta t R(t)} \quad (4.4)$$

Rata hazardului (HR) este cazul limită al FR pe măsură ce intervalul de timp se apropie de zero și se calculează ca

$$HR(t) = \lim_{\Delta t \rightarrow 0} \frac{F(t + \Delta t) - F(t)}{\Delta t R(t)} = \frac{f(t)}{R(t)} \quad (4.5)$$

4.4 Compararea rezultatelor

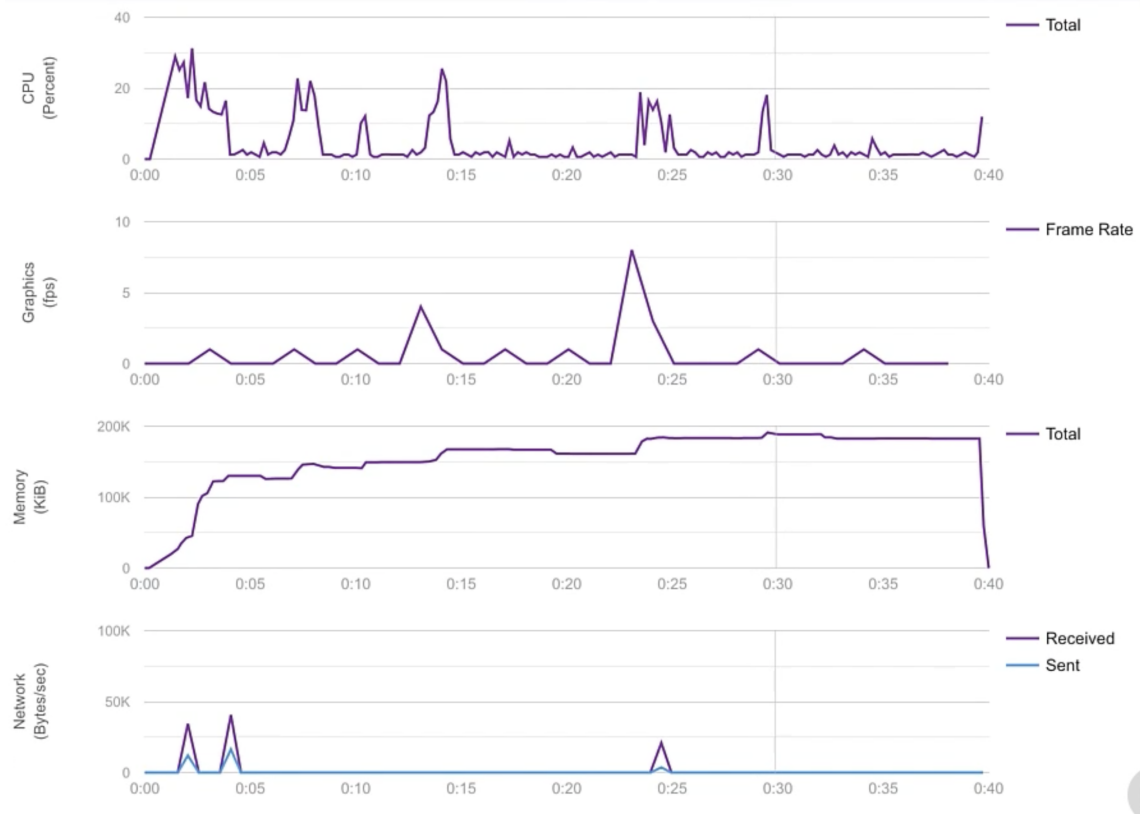
Există diferite metode de testare a interfețelor grafice și anume: Record and Replay (înregistrarea acțiunilor unui utilizator și repetarea lor), folosind Robotic Process Automation (există diferite moduri în care poate fi folosit pentru testare, printre care record actions, programatic sau folosind computer vision), prin integration testing în Flutter, prin testare automată sau prin script-uri, folosind diferite framework-uri (de exemplu Selenium). Fiecare prezintă rezultate diferite, dar și folosirea lor diferă din punct de vedere al eficienței atunci când este utilizată de programatori. De exemplu, prin folosirea de integration testing în Flutter, toate rezultatele din punct de vedere al performanței calculate de către Firebase sunt afișate la final, fără ca programatorul să programeze calcularea lor. De asemenea, când vine vorba de calculul performanței în urma testării interfeței grafice pe date reale, graficele rezultate sunt mult mai complete și mai sugestive, acesta fiind un mod foarte eficient de aflare a performanței comparativ cu alte metode care necesită programarea pas cu pas a calculării metricilor de performanță și afișarea lor.

5. Validare experimentală

5.1 Experiment pe un set de date artificial

Având un set de date artificial, singura metodă mai completă de testare a interfeței grafice ce poate fi folosită pentru măsurarea performanței este cea integrată din Flutter. Pentru a testa aplicația există 3 tipuri de testare în Flutter: unit testing, widget testing și integration testing, iar ultima dintre ele este cea care testează interfața grafică a aplicației. Am testat aplicația `projyproject` pe un set de date mai mic, flow-ul fiind cel de completare a field-urilor cu username și parolă, login în aplicație și verificare că s-a ajuns la home screen după logarea în aplicație. Testele au avut succes și un timp minim de 600ms. În timpul folosirii a integration testing sunt simulate acțiunile utilizatorului pentru a verifica dacă fiecare caz ajunge la rezultatul așteptat. De asemenea, Firebase afișează timpul rezultat în urma testării aplicației, dar și metricile de performanță folosind diferite grafice. Se poate observa eficiența acestei abordări în urma verificării consumului de memorie și de timp din metricile afișate: timpul măsurat în urma testării pentru display inițial și maximizat, diferite metrici ce țin de sincronizare, latență, thread-uri și upload. De asemenea, prezintă și grafice pentru performanța CPU (Central processing unit), grafice în fps (frames per second), memoria în KiB și rețeaua în bytes/secundă. Acest instrument ușurează cu mult măsurarea performanței, comparativ cu alte metode de testare a interfeței grafice.

Metrici de performanță folosind Firebase



5.2 Experiment pe date reale

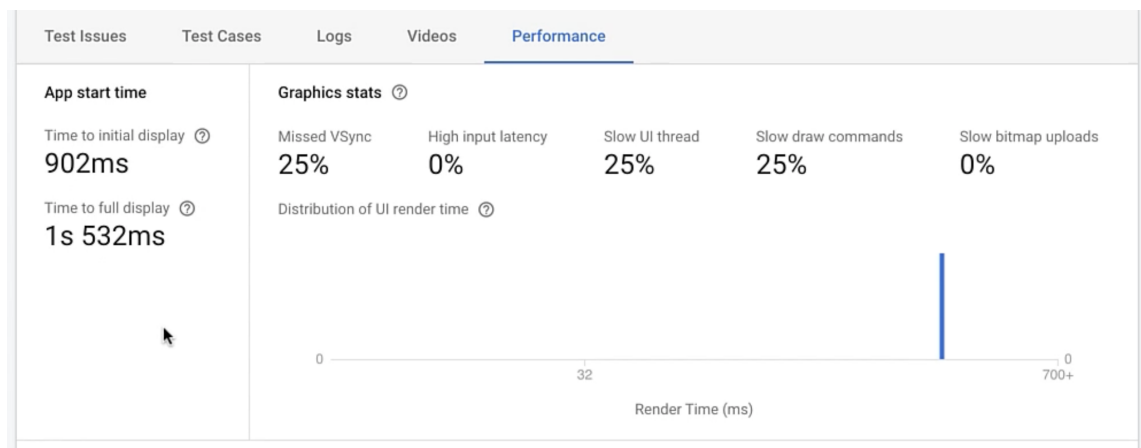
Am folosit un set de date real pentru testarea interfeței grafice a unui site web cunoscut și anume YouTube. Pentru a testa interfața grafică am folosit Robotic Process Automation (RPA), o tehnologie pe care am putut să o aplic cu tool-ul de la UiPath. Ceea ce face aplicația este să folosească boți pentru a deschide în mod automat YouTube, să caute o melodie și să o insereze într-un playlist. Tot ce face utilizatorul care rulează aplicația este să comunice numele playlist-ului și tipul, iar boții vor face toate operațiile pentru el. Dacă playlist-ul deja există, melodia căutată se adaugă în playlist. Altfel, se creează un playlist nou în care se adaugă melodia. În timpul efectuării operațiilor de către boți se măsoară performanța prin folosirea instrumentului Insights din UiPath, ce prezintă un set complet de metrice de performanță.

6. Rezultate și concluzii

6.1 Validarea rezultatelor

Validarea rezultatelor se face prin verificarea funcționării testelor conform flow-ului corect al aplicației și prin verificarea metricilor de performanță.

Figure 6.1: Validarea performanței folosind Firebase



6.2 Compararea cu abordări existente

Comparativ cu o metodă tradițională de Record and Replay folosită de obicei în industrie pentru a testa interfețele grafice, tool-ul de la UiPath oferă nu numai partea de programare a boților pentru repetiția acțiunilor, dar și partea de recunoaștere a elementelor de pe interfețele grafice folosind computer vision, împreună cu metoda de Record and Replay, fiind o metodă mult mai completă și mai intuitivă de testare a interfețelor grafice. Metricile sunt mult mai sugestive comparativ cu metoda de Record and Replay care arată doar măsurarea performanței de pe heap, statistica RPA arătând câte procese au fost automatizate, rata de succes, productivitatea boților și grafice pentru timpul mediu folosit în testare, metrici mult mai complete din punct de vedere al performanței.

6.3 Concluzii

Așadar, în urma comparațiilor și testărilor, cele mai complete instrumente de testare a performanței unei interfețe grafice sunt Firebase pentru un set artificial de date și UiPath Insights pentru un set real de date, deoarece oferă un pachet complet de metrici de performanță pe care programatorii le pot folosi pentru îmbunătățirea performanței aplicațiilor cu interfață grafică. În ceea ce privește alte direcții viitoare de cercetare, se poate aprofunda utilitatea acestor metrici în funcție de caz și găsirea soluțiilor ce rezolvă problemele de performanță ale aplicațiilor cu interfață grafică.

Figure 6.2: Metricile de performanță ale experților în domeniu folosind metoda de record and replay [2]

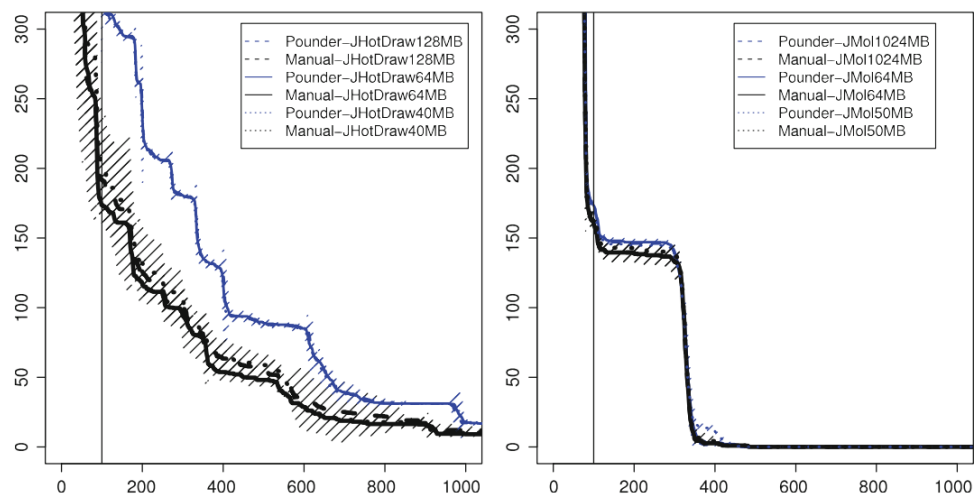


Fig. 8 Automated measurement of impact of heap size on perceptible performance

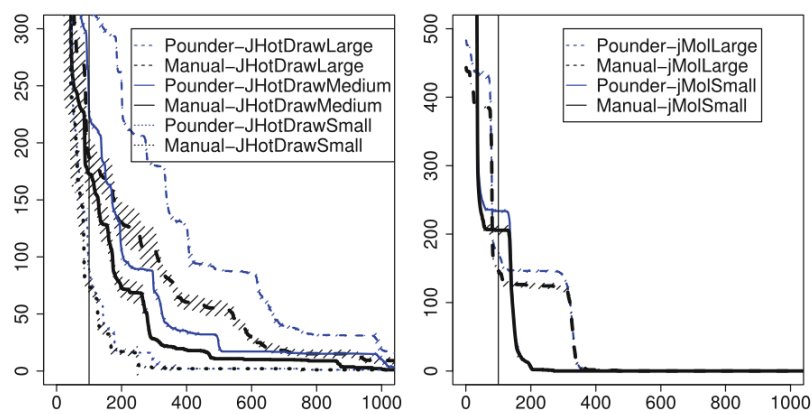
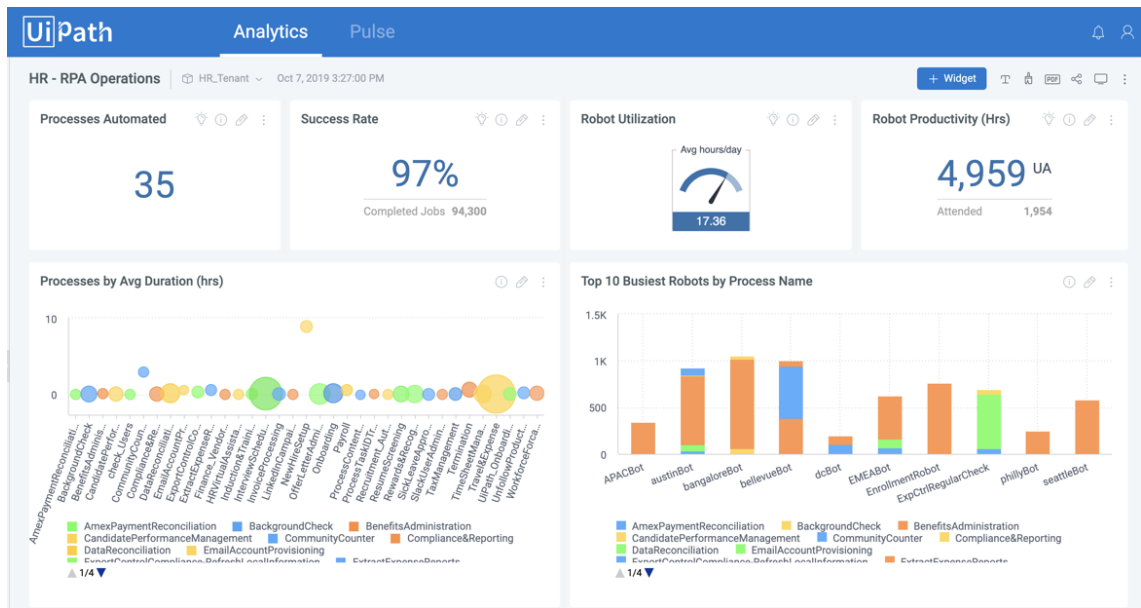


Fig. 9 Automated measurement of impact of input size on perceptible performance

Figure 6.3: Metricile de performanță folosind metoda RPA



7. Bibliography

- [1] Event-oriented, model-based gui testing and reliability assessment—approach and case study. volume 85 of *Advances in Computers*, pages 277–326. Elsevier, 2012.
- [2] Adamoli, Zaparanuks Andrea, Dmitrijs, Jovic Voytyuk, Hauswirth Milan, and Matthias. Automated gui performance testing. In *Software Quality Journal*, 2011.
- [3] Pekka Aho and Tanja Vos. Challenges in automated testing through graphical user interface. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 118–121, 2018.
- [4] Marina Cernat, Adelina-Nicoleta Staicu, and Alin Stefanescu. Improving ui test automation using robotic process automation. In *15th International Conference on Software Technologies*, pages 260–267, 01 2020.
- [5] Tsung-Hsiang Chang, Tom Yeh, and Robert C. Miller. Gui testing using computer vision. In *SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2010. Association for Computing Machinery.
- [6] InTMs Coimbra Morgado and Ana Paiva. Mobile gui testing. In *Software Quality Journal*, volume 26, 12 2018.
- [7] Cuixiong Hu and Iulian Neamtiiu. Automating gui testing for android applications. In *6th International Workshop on Automation of Software Test*, New York, NY, USA, 2011. Association for Computing Machinery.

- [8] Xue Qin, Hao Zhong, and Xiaoyin Wang. Testmig: Migrating gui test cases from ios to android. In *28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, New York, NY, USA, 2019. Association for Computing Machinery.
- [9] Thomas D. White, Gordon Fraser, and Guy J. Brown. Improving random gui testing with image-based widget detection. In *28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, New York, NY, USA. Association for Computing Machinery.
- [10] Solomiya Yatskiv, Iryna Voytyuk, Nataliia Yatskiv, Oksana Kushnir, Yuliia Trufanova, and Valentyna Panasyuk. Improved method of software automation testing based on the robotic process automation technology. In *2019 9th International Conference on Advanced Computer Information Technologies (ACIT)*, pages 293–296, 2019.