Primerjava zmogljivosti večplatformsko razvitih mobilnih aplikacij

Boris Ovčjak, Tatjana Welzer Družovec, Gregor Polančič, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Inštitut za informatiko, Smetanova 17, 2000 Maribor
boris.ovcjak@um.si; tatjana.welzer@um.si; gregor.polancic@um.si

Izvleček

Prispevek prikazuje rezultate primerjalne analize zmogljivosti večplatformsko razvite mobilne aplikacije za iOS, Android in Windows Phone. Analiza je namenjena ocenitvi primernosti uporabe izbranih razvojnih ogrodij. V ta namen sta bila izbrana dva razvojna pristopa (hibridni in večplatformsko prevedeni) ter pripadajoča ogrodja (Ionic in Xamarin). Implementirana vzorčna aplikacija je bila na podlagi petih testov zmogljivosti primerjana z aplikacijo, razvito z domorodnim pristopom. Rezultati testov so pokazali primernost uporabe večplatformsko prevedenega pristopa, saj se je ta na podlagi izbranega ogrodja domorodnemu pristopu precej približal, medtem ko se je hibridni pristop predvsem v strojno zahtevnejših opravilih izkazal za neučinkovitega.

Ključne besede: mobilne aplikacije, mobilne platforme, večplatformski razvoj, zmogljivost, hibridne aplikacije, večplatformsko prevedene aplikacije, Xamarin, Ionic.

Abstract

Performance comparison of cross-platform mobile applications

The article presents the results of a performance based comparative analysis of cross-platform developed mobile application for iOS, Android and Windows Phone. The analysis is intended to assess the appropriateness of use of the selected development frameworks. To this end, two development approaches (hybrid and cross-compiled) and associated frameworks (Ionic and Xamarin) were selected. The implemented sample application was compared with the native application on the basis of five performance tests. Test results have shown the suitability of cross-compiled approach, since the selected frameworks' performance came close to the native approach, while the hybrid approach proved ineffective especially in regard to in complex hardware tasks.

Keywords: mobile applications, mobile platforms, cross-platform development, performance, hybrid applications, cross-compiled applications, Xamarin, Ionic.

1 UVOD

Trg mobilnih naprav ter s tem povezanih mobilnih aplikacij je že nekaj časa v polnem razcvetu. Leta 2017 je predvideno 4,77 milijarde mobilnih uporabnikov (Statista, 2016b), pomembne rasti pa je deležen predvsem mobilni podatkovni promet, ki je leta 2015 zrasel za 74 odstotkov v primerjavi z letom 2014 (Cisco, 2016). Vsi ti podatki kažejo na razširjenost mobilnih naprav za poslovno in vsakodnevno uporabo. Tega se zavedajo tudi razvijalci, saj število aplikacij, ki so na razpolago v aplikacijskih trgovinah, nenehno raste. Tri izmed najbolj razširjenih platform, Android, iOS in Windows Phone, skupaj dosegajo že skoraj pet milijard aplikacij (Android 2,2 milijarde, iOS 2 milijardi in Windows Phone 0,67 milijarde) (Statista, 2016a). Prav porazdeljenost platform je za razvijalce eden večjih izzivov, saj je treba tako naloge razvoja kot tudi posodabljanja in vzdrževanja aplikacij opravljati na več platformah (Joorabchi, Mesbah in Kruchten, 2013). To zahteva večkratno izvajanje enakih nalog, kar vpliva tako na čas razvoja kot na s tem povezane stroške.

Omenjene težave pri razvoju aplikacij za več platform so razvijalce prisilile v iskanje alternativnih, učinkovitejših večplatformskih razvojnih tehnik. Trenutno je na voljo več pristopov, ki ob prihranjenem času in stroških zahtevajo določene kompromise, povezane z izgledom, razpoložljivostjo funkcionalnosti mobilne naprave ter učinkovitostjo delovanja aplikacij. Zaradi tega izbira večplaformskega razvojnega pristopa predstavlja svojevrsten izziv, s katerim

se spopadajo tudi raziskovalci, ki skušajo razvijalcem olajšati izbiro s predstavitvijo in primerjavo obstoječih pristopov tako s teoretičnega (znanstvenega) (R. M. de Andrade, B. Albuquerque, F. Frota, V. Silveira in A. da Silva, 2015; Charkaoui, Adraoui in Habib Benlahmar, 2014; Palmieri, Singh in Cicchetti, 2012; Smutný, 2012; Adinugroho in Gautama, 2015; Xanthopoulos in Xinogalos, 2013) kot tudi s strokovnega (praktičnega) stališča. Raziskave se pri tem usmerjajo v analize različnih vidikov večplatformskega razvoja (Dalmasso, Datta, Bonnet in Nikaein, 2013; Danielsson, Ameri, Friberg, Examiner in Lindell, 2014). Ob že uveljavljenih razvojnih pristopih raziskovalci razvijajo tudi nove, ki temeljijo na obstoječih, oz. razvijajo nove alternativne (Perchat, Desertot in Lecomte, 2013; Bouras, Papazois, & Stasinos, 2014 El-Kassas, Abdullah, Yousef in Wahba, 2014). Glede na količino različnih ogrodij in pristopov se raziskovalci posvečajo tudi izdelavi platform za ocenjevanje večplatformskih mobilnih ogrodij (Dhillon, 2015).

V članku bomo predstavili različne vrste večplatformskega razvoja, njihove lastnosti, prednosti in slabosti. Glede na razpoložljivost različnih ogrodij in platform se bomo natančneje osredinili predvsem na tiste, ki so razvijalcu v polni funkcionalnosti na voljo brezplačno in je z njimi mogoče razviti mobilno aplikacijo za platforme iOS, Android in Windows Phone. Za vsakega izmed preučenih pristopov bomo podrobneje predstavili najbolj razširjenega predstavnika ter na podlagi vzorčne aplikacije s primerjalno analizo zmogljivosti ocenili primernost uporabe ogrodij v primerjavi z domorodnim razvojem.

2 PRISTOPI RAZVOJA VEČPLATFORMSKIH APLIKACIJ

Cilj večplatformskega razvoja mobilnih aplikacij je zagotoviti enovit pristop k razvoju mobilnih aplikacij za več platform. Poleg domorodnega načina razvoja aplikacij, ki zahteva ločen razvoj za vsako izmed mobilnih platform, je to mogoče zagotoviti na različne načine. Vsi omogočajo z enim samim razvojnim okoljem ter programskim jezikom razviti mobilno aplikacijo, ki jo je mogoče poganjati na različnih platformah. To je pomembno, saj se ob enkratni izbiri razvojnega ogrodja razvijalcem ni več treba ukvarjati z učenjem novih tehnologij oz. novih programskih jezikov za vse platforme, saj lahko uporabijo že obstoječe znanje. Glede na raznolikost načinov večplatformskega razvoja obstajajo različne prednosti in slabosti načina razvoja. V splošnem večplatformski razvoj lahko prinese prednosti, kot so: 1) hitrejši čas razvoja, 2) nižji stroški razvoja, 3) ponovna uporaba programske kode, 4) lažji razvoj, 5) uporaba vtičnikov, 6) večji doseg tržišča ter 7) preprosto in hitro podajanje posodobitev. Omenjene prednosti pa lahko zahtevajo tudi določene kompromise, med katere štejemo: 1) manjšo kakovost 3D vsebin in grafike, 2) omejitve platforme (nezmožnost dostopanja do vseh funkcionalnosti platforme), 3) ni zagotovljene optimalne uporabniške izkušnje, 4) težave pri integraciji s platformo ter 5) počasnejše delovanje aplikacij (Rajput, 2016).

Težave, ki jih prinaša porazdeljenost mobilnih platform, so povzročile razcvet različnih kategorij večplatformskih pristopov, ki se med sabo razlikujejo tako po arhitekturi, načinu razvoja, uporabljenih tehnologijah ter razvojnih orodjih. V splošnem načine razvoja mobilnih aplikacij delimo na pet skupin (Rahul Raj in Tolety, 2012; Xanthopoulos in Xinogalos, 2013).

- **Domorodne aplikacije** (angl. Native applications) pomenijo privzet način razvoja mobilnih aplikacij in so specifične za vsako mobilno platformo, torej uporabljajo razvojna orodja in programski jezik, ki ga določa specifična platforma. Običajno tovrstne aplikacije zagotavljajo največjo podporo funkcionalnostim naprave ter omogočajo aplikaciji specifičen izgled ter optimalno delovanje (Korf in Oksman, 2016).
- **Spletne mobile aplikacije** (angl. mobile web applications) so aplikacije, razvite z namenom izvajanja v brskalniku mobilne naprave. Razvite so s pomočjo spletnih tehnologij (HTML5, CSS3 in JavaScript). Za izvajanje tovrstnih aplikacij mobilna naprava ne potrebuje posebej nameščene aplikacije, saj se vsa vsebina prikazuje v brskalniku naprave.
- **Hibridne aplikacije** (angl. hybrid applications) so kombinacija spletnega in domorodnega razvojnega pristopa. Razvite so s pomočjo spletnih tehnologij in se na mobilni napravi izvajajo znotraj domorodne komponente WebView. Ta je skupna vsem mobilnim platformam in omogoča prikaz vsebine HTML na celoten zaslon naprave. Takšen način razvoja omogoča dostop do funkcionalnosti naprave preko abstrakcijskega nivoja, ki zmožnosti naprave izpostavlja v obliki JavaScript API (angl. Application Programming Interface). Hibridni pristop lahko uporabljamo tako za strežniške kot za samostojne spletne aplikacije. Za razliko od spletnih mobilnih aplikacij moramo

tovrstne aplikacije prenesti na telefon prek aplikacijskih trgovin.

- **Interpretirane aplikacije** (angl. Interpreted applications) delujejo tako, da se aplikacijska koda postavi na mobilno napravo in se interpretira šele v času izvajanja aplikacije. Aplikacija nato komunicira z abstrakcijskim nivojem, prek katerega dostopa do domorodnega API-ja, kar ji omogoča dostop do naprednejših funkcionalnosti mobilne naprave. Tako razvite aplikacije omogočajo uporabo platformsko specifičnih elementov za izgradnjo uporabniškega vmesnika, medtem ko je aplikacijska logika zajeta na način, neodvisen od platforme. To je lahko v obliki množice ukazov v XML ali v katerem drugem opisnem jeziku.
- **Večplatformsko prevedene aplikacije** (angl. cross-compiled applications) temeljijo na prevajalniku, ki omogoča pretvorbo izvorne v domorodno binarno kodo. Prevajalnik je v tem primeru odgovoren za generiranje in izvajanje programske kode za specifično platformo (Xamarin, 2016). Tako lahko razvijalec napiše izvorno kodo v enotnem skupnem programskem jeziku, ki ga prevajalnik nato prevede v domorodno kodo, določeno pri ciljni platformi. V tem primeru je celoten pristop odvisen od učinkovitosti in zanesljivosti prevajalnika.

Tabela 1 prikazuje različne pristope večplatformskega razvoja, pripadajoča ogrodja, njihovo dostopnost ter njihove prednosti in izzive (Rahul Raj in Tolety, 2012).

Tabela 1: **Pristopi večplatformskega razvoja mobilnih aplikacij (Rahul Raj in Tolety, 2012)**

Pristop	Ogrodja	Prednosti	Izzivi
Domorodne	Specifična glede na platformo (brezplačno)	– Domoroden uporabniški izgled in delovanje – Dostop do vseh funkcionalnih lastnosti platforme – Hitrost izvajanja	– Razvoj aplikacij je različen za vsako mobilno platformo.
Spletne	Adobe Air (brezplačno)	– Instalacija aplikacije na mobilno napravo je nepotrebna. – Podatki aplikacije se hranijo na spletnem strežniku, kar lajša posodabljanje aplikacije. – Enoten uporabniški vmesnik	– Nezmožnost distribucije prek aplikacijskih trgovin, kar lahko negativno vpliva na popularnost aplikacije – Odvisnost od hitrosti povezave s spletom – Nezmožnost dostopanja do funkcionalnosti in strojne opreme mobilne naprave – Potreba po upoštevanju velike količine različnih resolucij – Manjši nadzor mobilnega brskalnika nad prikazovanjem vsebin – Omejena podpora gestam (angl. Gestures) – Težja monetizacija aplikacije
Hibridne	Cordova (brezplačno), PhoneGap (brezplačno), MoSync (brezplačno), RhoMobile (brezplačno z omejitvami)	– Distribucija prek aplikacijskih trgovin – Ponovna uporaba uporabniškega vmesnika prek različnih platform – Dostop do funkcionalnosti in zmožnosti mobilne naprave	– Slabša učinkovitost delovanja v primerjavi z domorodnimi aplikacijami, saj se izvajanje dogaja v komponenti brskalnika – Zaradi uporabe JavaScript programskega jezika so aplikacije izpostavljene ranljivostim CSC (angl. Cross space communication). – Ponovna uporaba uporabniškega vmesnika ne ponuja domorodnega izgleda aplikacije. To lahko dosežemo z oblikovanjem, specifičnim za vsako platformo.
Interpretirane	Titanium (plačljiv, brezplačno dostopen osnovni SDK), React Native (brezplačno)	– Prinaša izgled in občutek domorodne mobilne aplikacije. – Omogoča prenos poslovne logike med platformami. – Distribucija aplikacij poteka prek aplikacijskih trgovin. – Strojna oprema in lastnosti naprave so zapakirane znotraj specifičnega API-ja.	– Ponovna uporaba uporabniškega vmesnika je odvisna od abstrakcije na nivoju ogrodja. – Razvoj je odvisen od množice funkcij, ki jih omogoča ogrodje. – Učinkovitost delovanja je slabša zaradi interpretiranja programske kode v času delovanja.
Večplatformsko prevedene	Xamarin (brezplačno), Marmelade (brezplačno z omejitvami), JavaFxPorts (brezplačno)	– Ponuja vse funkcionalnosti, ki jih omogoča domorodna aplikacija. – Omogočen dostop do strojne opreme mobilne naprave – Zmožnost uporabe vseh vmesniških komponent – Glavno prednost predstavlja ustrezna učinkovitost delovanja.	– Ne omogoča ponovne uporabe uporabniškega vmesnika ter platform specifičnih lastnosti, kot so kamera, lokacijske storitve, lokalna obvestila ipd. Ker so te funkcionalnosti odvisne od platforme, se način dostopa razlikuje med platformami.

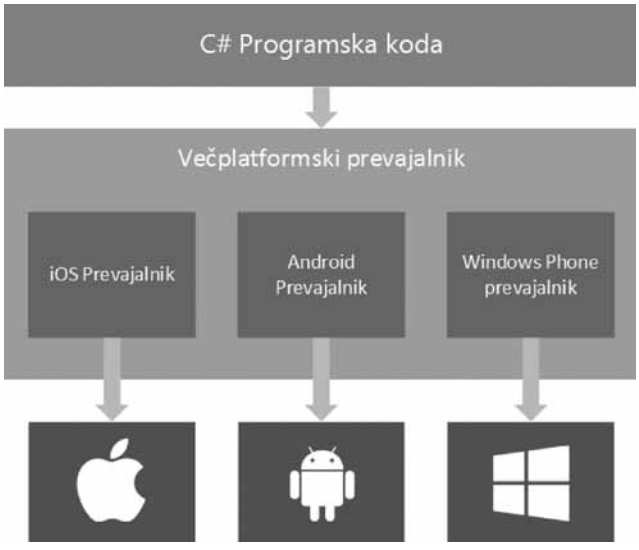
Iz tabele je razvidno, da smo ne glede na izbrani pristop soočeni z določenimi izzivi, ki jih ta prinaša. V povezavi s tem je bil glavni namen članka z vidika zmogljivosti aplikacije primerjati trenutno najbolj aktualna večplatformska ogrodja z domorodnim pristopom. Pri tem smo se omejili na brezplačno dostopna ogrodja, ki omogočajo razvoj mobilnih aplikacij za izbrane tri mobilne platforme. Zaradi tega smo iz primerjalne analize izključili pristop mobilnih spletnih aplikacij, saj se tovrstne aplikacije lahko izvajajo izključno le v brskalniku mobilne naprave. V podrobnejšo analizo prav tako nismo vključili interpretiranega pristopa, saj oba izmed pregledanih predstavnikov ne podpirata trenutno najbolj razširjenje verzije sistema Windows Phone, uporabljene v tej analizi (Windows Phone 8.1) (NetMarketShare, 2016), prav tako pa platforma Titanium v uradni obliki ni na voljo brezplačno (Titanium, 2016).

Ob upoštevanju omejitev se bomo v članku osredinili na pristop večplatformsko prevedenih in hibridnih aplikacij. Glede na to, da oba pristopa vključujeta zbirko različnih ogrodij, smo v nadaljevanju izbrali po enega predstavnika za vsak tip razvoja. V primeru večplatformskega pristopa smo se odločili za ogrodje Xamarin, saj predstavlja enega izmed najbolj celovitih ogrodij, podprtega s strani podjetja Microsoft. Hibridni pristop, za katerega smo se prav tako odločili, pa je predvsem zaradi priljubljenosti spletnih tehnologij, na katerih temelji, precej razširjen. Za potrebe študije smo ogrodje izbrali na podlagi priljubljenosti med razvijalci, in sicer ogrodje Ionic (Noeticforce, 2016; Markov, 2015). Oba izbrana predstavnika sta na kratko predstavljena v nadaljevanju.

2.1 Predstavnik večplatformsko prevedenega pristopa Xamarin

Xamarin je odprtokodna platforma, ki jo ponuja podjetje Microsoft in omogoča izgradnjo večplatformsko prevedenih mobilnih aplikacij na deljeni programski bazi z uporabo enotnega razvojnega okolja, programskega jezika in API-ja. Podjetje Xamarin je ustanovil Miguel de Icaza 16. maja 2011 (Allen, 2011) z namenom izgradnje mobilnih aplikacij na podlagi ogrodja Mono, ki omogoča večplatformsko izvajanje aplikacij .NET. Uporaba platforme je bila do nedavnega plačljiva, kar pa se je spremenilo, ko je 24. februarja 2016 podjetje prevzel Microsoft (Guthrie, 2016). Dober mesec po prevzemu je namreč podjetje Microsoft objavilo, da je Xamarin SDK postal odprtokodno ogrodje na voljo brezplačno znotraj integriranega razvojnega okolja Visual Studio.

Ogrodje Mono .NET, na katerem temelji platforma Xamarin, omogoča implementacijo širokega spektra ogrodja .NET, ki ga s programskim jezikom C# lahko poganjamo na več mobilnih platformah, in sicer iOS, Android ter Windows Phone. Razvoj za vsako izmed omenjenih platform je podprt prek paketov za razvoj programske opreme (angl. SDK), ki omogočajo povezavo do večine platform specifičnih funkcionalnosti (Xamarin, 2016a). SDK je specifičen tako za Android (Xamarin.Android SDK) kot za iOS (Xamarin.iOS SDK), medtem ko za platformo Windows Phone ni potreben. Glede na platformo, za katero razvijamo aplikacijo, prevajalnik Xamarin zgradi domorodno aplikacijo, kar pa zaradi raznolikosti ciljnih platform poteka na različne načine (slika 1).



Slika 1: **Shema pristopa večplatformsko prevedene aplikacije**

V primeru platforme iOS se C# programska koda vnaprej prevede v zbirni jezik ARM (angl. Assembly language), v katerega se vključi ogrodje .NET. Na drugi strani se v primeru platforme Android C# prevede v vmesni jezik (angl. Intermediate language), ki se zapakira v Mono virtualno okolje (MonoVM). Končna aplikacija se tako izvaja vzporedno z javanskim ali android izvajalnim okoljem (angl. Runtime environment) in komunicira z domorodnimi tipi prek domorodnega vmesnika Java (angl. Java native interface). V primeru obeh platform se pred vključitvijo ogrodja .NET temu prej odstranijo nerabljeni

razredi v namen zmanjšanja velikosti končne aplikacije. V primerjavi z omenjenima platformama je postopek priprave aplikacije za platformo Windows Phone enostavnejši, saj za delovanje ne potrebuje ogrodja Xamarin, a je ta vseeno priporočljiv zaradi možnosti ponovne uporabe programske kode (Xamarin, 2016b). To je v primeru ogrodja Xamarin mogoče doseči na več načinov (tabela 2).

Tabela 2: **Metode ponovne uporabe programske kode**

Metoda ponovne uporabe programske kode	Opis
Xamarin. Mobile	Enoten API za dostop do skupnih virov mobilne naprave za vse vrste razvojnih platform (Xamarin, 2016d)
Deljeni projekt (angl. Shared Asset Project)	Projekt, namenjen organizaciji izvorne kode z uporabo direktiv na ravni prevajalnika za upravljanje platformsko specifičnih delov programske kode
Prenosljive razredne knjižnice (angl. Portable Class Libraries)	Projekt, namenjen izgradnji knjižnic za podprte platforme, ki za dostop do platformsko specifičnih funkcionalnosti uporabljajo vmesnike
Xamarin.Forms	Strani, napisane z uporabo programskega jezika C# ali XAML, ki se v času izvajanja skupaj z vsebujočimi kontrolami preslikajo v platformsko specifične vizualne elemente Predvsem so namenjene aplikacijam, osredinjenim na vnos podatkov.

Z uporabo različnih metod ponovne uporabe programske kode lahko dosežemo do 90 % deljene programske kode (Xamarin, 2016a).

Za razvoj aplikacij ogrodje ponuja dve programski orodji, in sicer Visual Studio (Windows) ter Xamarin Studio (Windows, macOS), ki glede na operacijski sistem, v katerem razvijamo, podpirata razvoj različnih mobilnih platform (tabela 3).

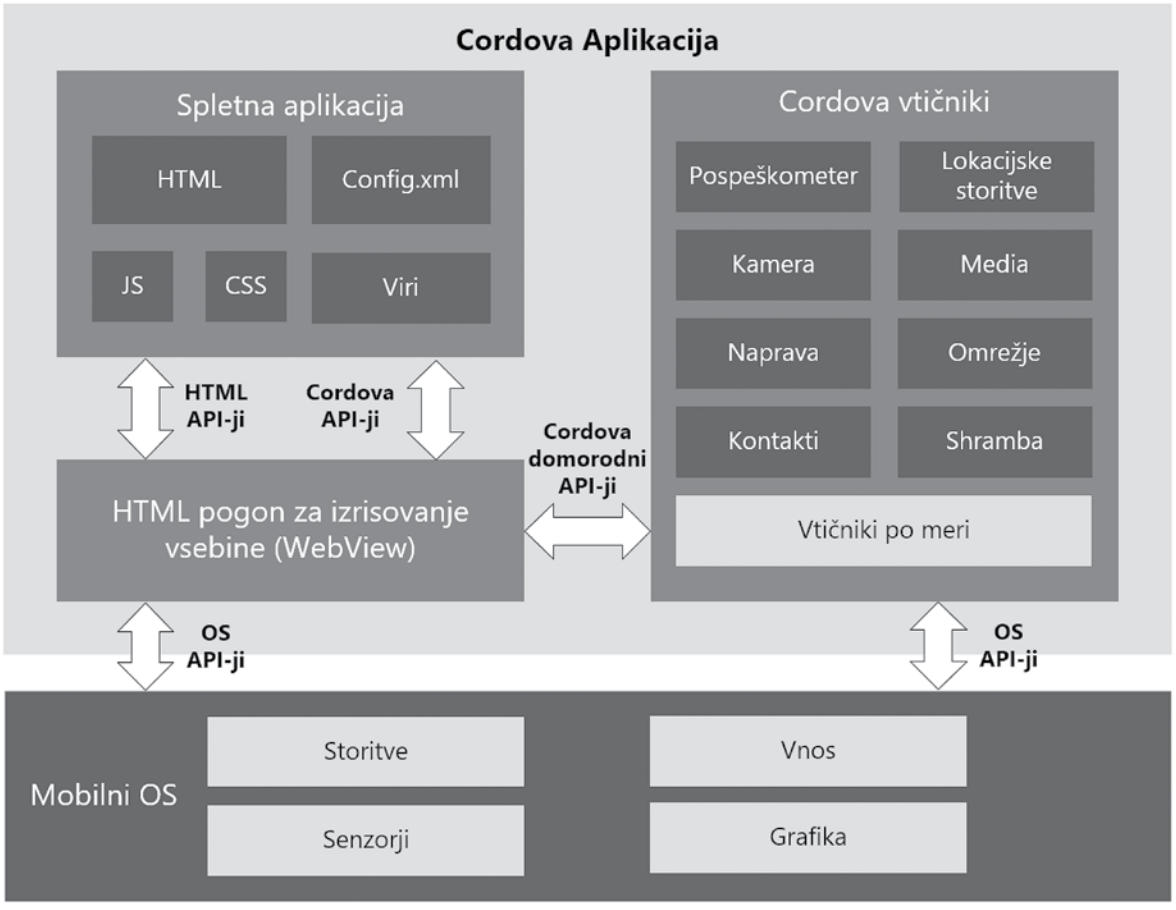
Čeprav je v operacijskem sistemu Windows mogoč razvoj aplikacij za vse podprte platforme, pa za poganjanje iOS aplikacij v vsakem primeru potrebujemo računalnik z nameščenim operacijskim sistemom macOS ter domorodnim razvojnim okoljem XCode, saj je za uspešno prevajanje aplikacije potreben iOS SDK (Xamarin, 2016b).

2.2 Predstavnik hibridnega pristopa Ionic

Ionic je knjižnica za razvoj hibridnih mobilnih aplikacij, ki temeljijo na ogrodjih, kot so Cordova, PhoneGap ali Trigger.io (Ionic, 2016). Vsa omogočajo razvoj večplatformskih aplikacij z uporabo standardnih spletnih tehnologij HTML5, CSS3 in JavaScript. Tako grajene aplikacije se izvajajo znotraj ovojnic, specifičnih za vsako ciljno platformo, in prek API-ja zagotavljajo dostop do zmožnosti mobilne naprave, kot so senzorji, podatki ali omrežje. Arhitekturno zasnovo hibridne mobilne aplikacije v primeru uporabe hibridnega ogrodja Cordova predstavlja slika 2.

Tabela 3: **Podprtost platform glede na operacijski sistem in razvojno orodje**

	macOS	Windows	
Mobilna platforma	Xamarin Studio	Visual Studio	Xamarin Studio
iOS	Da	Da (z macOS računalnikom)	Ne
Android	Da	Da	Da
Windows Phone	Ne	Da	Da



Slika 2: **Arhitektura mobilne aplikacije na osnovi ogrodja Cordova (Cordova, 2012)**

V primeru ogrodja Cordova celoten uporabniški vmesnik mobilne aplikacije predstavlja komponenta WebView, ki omogoča prikaz spletne vsebine znotraj aplikacije. Znotraj te se nahaja programska koda aplikacije, implementirana v obliki spletne strani skupaj z nastavitveno datoteko, ki vsebuje informacije o aplikaciji ter parametre za njeno delovanje. Ta nato prek API klicev komunicira z različnimi vtičniki, ki predstavljajo vmesnike tako za ogrodje Cordova kot za domorodne komponente mobilne naprav (Cordova, 2012).

Neodvisno od podpornega hibridnega ogrodja je glavni namen knjižnice Ionic uporabniku olajšati razvoj čela (angl. Front-end) mobilne aplikacije ter tako zagotoviti pristen izgled in občutek aplikacije predvsem na ravni interakcije z uporabniškim vmesnikom. To omogočata prilagojeno ogrodje CSS ter knjižnica JavaScript. Arhitektura ogrodja temelji na vzorcu View Controller, medtem ko za poslovno logiko skrbi odprtokodno ogrodje JavaScript AngularJS (Ionic, 2016). Knjižnica v verziji 1.3 uradno podpira le razvoj aplikacij za platformi iOS in Android, a je zaradi uporabe spletnih tehnologij prenos aplikacije na platformo Windows Phone mogoč s preprosto modifikacijo nastavitvene datoteke (Hösl, 2014).

Zaradi uporabe spletnih tehnologij je aplikacijo Ionic mogoče razviti z uporabo katerega koli orodja, ki podpira razvoj spletnih aplikacij. Po drugi strani pa mobilna komponenta WebView zagotavlja, da je aplikacijo brez sprememb v programski kodi mogoče poganjati na vseh izbranih mobilnih platformah. Spremembe so potrebne le v primeru prilagajanja izgleda aplikacije pravilom posamezne platforme.

3 PRIMERJAVA PRISTOPOV

Glavni cilj raziskave je bil primerjava zmogljivosti delovanja večplatformsko razvitih mobilnih aplikacij z namenom ugotavljanja njihove primernosti kot alternative domorodnemu pristopu. V sklopu primerjave smo na vseh izbranih platformah opazovali

zmogljivost delovanja izbranih ogrodij Xamarin in Ionic v primerjavi z domorodnim pristopom, ki je podlaga za optimalno delovanje mobilnih aplikacij.

3.1 Opredelitev testov in testne aplikacije

Opredelitev testov je temeljila na standardnih testih mobilnih naprav (PassMark, 2016) ter vidikih odzivnosti aplikacije (DeCapua, 2015). Pri izvedbi testa smo se zgledovali po obstoječem delu Jakoba Danielssona (2014), ki je na podlagi mobilne platforme iOS primerjal različne pristope večplatformskega razvoja. Njegov pristop je temeljil na standardnih testih mobilnih naprav, ki so služili kot primerna podlaga za pripravo testov zmogljivosti naše raziskave. Naša analiza je od omenjene študije prevzela tri teste, ki preverjajo zmogljivost delovanja procesorske enote, spomina in izrisovanja grafike, dodali pa smo primerjavo odzivnosti aplikacije ter časa, potrebnega za vzpostavitev aplikacije.

Na podlagi omenjenih dejstev smo zasnovali pet testov (tabela 4), pri katerih smo opazovali odzivni čas mobilne aplikacije, tj. metriko časovnega obnašanja (angl. Time behaviour metrics) zmogljivostne učinkovitosti, povzeto po standardu ISO/IEC 2010:2011. Metrika je namenjena merjenju povprečnega čakalnega časa, ki preteče od podane zahteve do pridobitve rezultata pod specifičnimi obremenitvami sistema v smislu sočasnih nalog ali sistemske uporabe (angl. Utilisation) (ISO, 2016).

Tabela 4: **Testi zmogljivosti ter opazovana metrika**

Test zmogljivosti	Razlaga	Scenarij izvedbe	Opazovana metrika
Test procesorja	Čas izvedba kalkulacije matematične konstante π na X decimalnih številk natančno z enovitim algoritmom (Code Codex, 2012)	X = 10.000	Odzivni čas (ms)
Test pomnilnika	Čas zapisovanja in branja zbirke X števil	X = 10.000.000	
Test 2D grafike	Čas, potreben za izris 1000 gumbov	X = 1000	
Test odzivnosti	Čas, ki preteče med interakcijo z aplikacijo in njenim odzivom v obliki preprostega obvestila	/	
Test vzpostavitve aplikacije	Čas, potreben za zagon aplikacije	/	

Za izvedbo testov zmogljivosti smo za vsako izmed izbranih mobilnih platform implementirali tri mobilne aplikacije za vsakega izmed preučevanih razvojnih pristopov (slika 3).

Slika 3: **Vzorčne aplikacije**

Domorodni razvoj za vsako izmed platform je potekal s pripadajočim razvojnim orodjem in platformi specifičnem programskem jeziku. Aplikacijo iOS smo razvili z razvojnim orodjem XCode in programskim jezikom Objective-C, aplikacijo Android z orodjem Android Studio in programskim jezikom Java, aplikacijo Windows Phone pa z orodjem Visual Studio ter programskim jezikom C#. V okviru primerjave je razvoj za ogrodje Xamarin potekal nekoliko enostavneje, in sicer z enotnim programskim jezikom C#. Aplikacijo smo tako za iOS kot Android implementirali z orodjem Xamarin Studio, medtem ko smo za Windows Phone uporabili orodje Visual Studio. Najenostavnejše za razvoj se je izkazalo ogrodje Ionic, pri katerem smo po vzpostavitvi projekta z uporabo ukazne vrstice vso programsko logiko za vse izmed platform zapisali v preprostem urejevalniku besedila. Glede na rezultirajoče aplikacije lahko opazimo, da aplikacije, razvite z domorodnim pristopom in ogrodjem Xamarin, posedujejo platformi domoroden izgled, medtem ko izgled aplikacije v primeru ogrodja Ionic ostaja enak skozi vse platforme.

3.2 Predstavitev testnih okolij

Za potrebe testov smo potrebovali tri izvajalna okolja (mobilne naprave) različnih platform, na katere smo namestili aplikacije vseh treh testiranih pristopov. Vse mobilne naprave ter njihovi pripadajoči operacijski sistemi so predstavljeni v tabeli 5.

Tabela 5: **Testna izvajalna okolja**

Atributi	iOS	Android	Windows Phone
Naprava	iPhone 6S	LG Nexus 5	Nokia Lumia 635
Procesor	Dual-core 1.84 GHz	Quad-core 2.3 GHz	Quad-core 1.2 GHz
Interni pomnilnik	2 GB	2 GB	512 MB
Grafična enota	PowerVR GT7600	Adreno 330	Adreno 305
Operacijski sistem	iOS 9.3.2	Android 6.0.1	Windows Phone 8.1

Enakovredno testiranje mobilnih aplikacij na različnih platformah je težko doseči predvsem zaradi razlik v strojni opremi. Enako velja za posnemovalnike mobilnih naprav, saj vsak izmed ponudnikov platforme določa svoj posnemovalnik, prilagojen specifični platformi. Ne glede na definirane razlike nam je uporaba dejanskih mobilnih naprav omogočila realno primerjavo rezultatov znotraj posamezne platforme.

3.3 Omejitve izvedbe testov

Primerjava različnih pristopov na različnih mobilnih napravah zahteva zapis mobilne aplikacije v več različnih programskih jezikih, ki imajo svoja sintaktična pravila. Če razvojna metoda zahteva programsko kodo, ki je specifična za določeno platformo, pride še do dodatne specializacije razvoja, kar lahko vpliva na rezultate analize. Na natančnost testiranja aplikacije lahko vpliva tudi metoda merjenja časa izvedbe določenega testa, saj je v nekaterih primerih (asinhroni klici) nemogoče programsko določiti konec testa. Dodatna omejitev so tudi same mobilne naprave ali njihovi posnemovalniki, saj je v primeru uporabe katere koli možnosti testiranja aplikacije težko zagotoviti enakovrednost delovanja. Glede na naštete pomanjkljivosti smo opredelili omejitve, ki lahko vplivajo na rezultate izvedenih testov.

- Različni programski jeziki. Pri razvoju za različne platforme se ne moremo izogniti različnim programskim jezikom, ki zaradi različne sintakse in tehnologije delovanja vplivajo na hitrost izvajanja procesov.
- Platformsko specifična koda. Različne tehnologije definirajo različne načine implementacije določenih platformsko specifičnih funkcionalnosti, česar v primeru različnih pristopov večplatformskega razvoja ne moremo poenotiti.
- Merjenje časa izvedbe testov. Za določena opravila je določitev časa, potrebnega za končanje naloge, mogoče natančno določiti v programski kodi, medtem ko to v nekaterih primerih ni mogoče (npr. asinhrono delovanje). V teh primerih je meritev mogoča le z uporabo zunanje meritve.
- Omejitve glede mobilnih naprav. Uporaba dejanskih naprav pri testiranju mobilnih aplikacij povzroči razlike, ki temeljijo na razlikah v strojni

opremi naprave. Neposredna primerjava rezultatov različnih mobilnih naprav je tako nemogoča, saj lahko razlike v strojni opremi povzročijo precejšnje spremembe v času izvedbe testov.

Zaradi navedenih omejitev smo v okviru razvitih mobilnih aplikacij uporabili le najosnovnejše komponente ter se z izjemo naslavljanja vizualnih elementov poskušali izogniti naprednejšim funkcionalnostim naprave. Omejitve merjenja časa izvedbe testov smo odpravili tako, da smo vsak test izvedli desetkrat ter nato spremljali povprečje časov izvedbe testov. Ker nismo mogli zagotoviti enakovrednosti mobilnih naprav, smo podrobno analizo testov izvedli s primerjavo koeficientov sprememb povprečnega časa izvedbe testov med posameznimi pristopi.

3.4 Rezultati raziskave

Tabela 6 prikazuje rezultate testov, izvedenih na vseh izbranih mobilnih platformah ter pripadajočih mobilnih napravah. Za vsak test so navedeni povprečni čas izvedbe, maksimalni in minimalni čas ter mediana časa izvedbe.

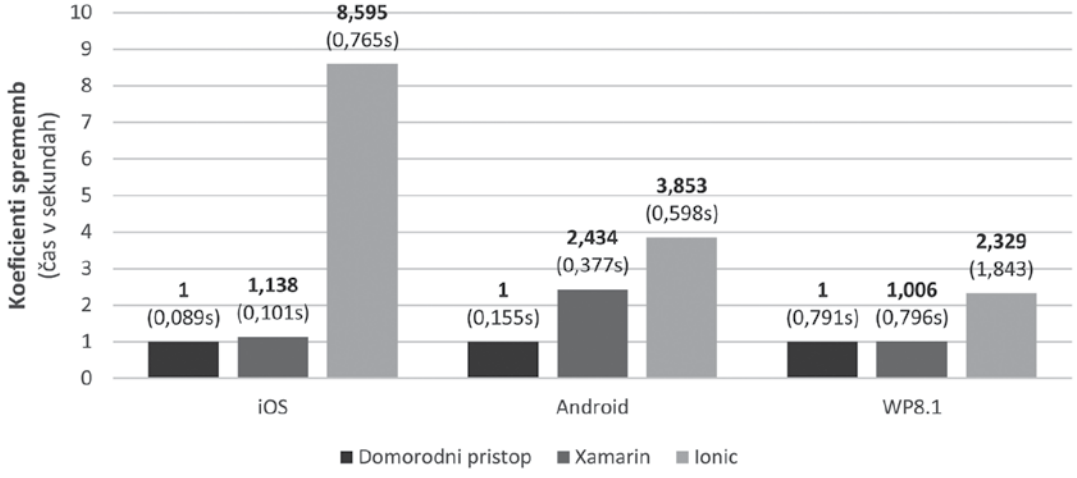
Tabela 6: **Rezultati testov zmogljivosti**

	Povprečni čas (s)	Maksimalni čas (s)	Minimalni čas (s)	Mediana (s)	Povprečni čas (s)	Maksimalni čas (s)	Minimalni čas (s)	Mediana (s)	Povprečni čas (s)	Maksimalni čas (s)	Minimalni čas (s)	Mediana (s)
Naprava	**Apple iPhone 6S (iOS 9.3.2)**				**LG Nexus 5 (Android 6.0.1)**				**Nokia Lumia 635 (Windows Phone 8.1)**			
Test	Test procesorja											
Dom.	0,089	0,118	0,053	0,095	0,155	0,178	0,145	0,148	0,791	0,797	0,785	0,793
Xamarin	0,101	0,112	0,096	0,098	0,377	0,393	0,361	0,381	0,796	0,807	0,783	0,797
Ionic	0,766	0,779	0,722	0,769	0,598	0,609	0,589	0,596	1,843	1,876	1,831	1,839
Test	Test spomina											
Dom.	0,147	0,157	0,140	0,146	0,208	0,232	0,186	0,202	0,701	0,720	0,681	0,700
Xamarin	0,166	0,171	0,152	0,167	0,302	0,325	0,289	0,301	0,713	0,740	0,695	0,709
Ionic	2,296	2,312	2,277	2,297	1,812	1,962	1,653	1,815	7,810	8,133	7,726	7,777
Test	Test 2D grafike											
Dom.	0,470	0,515	0,381	0,483	2,134	2,252	1,862	2,148	4,542	4,569	4,487	4,551
Xamarin	0,578	0,634	0,484	0,593	2,703	2,945	2,597	2,692	4,537	4,622	4,434	4,537
Ionic	3,081	3,229	3,004	3,069	23,97	27,47	22,71	23,01	100,8	106,9	94,49	99,88
Test	Test odzivnosti											
Dom.	0,042	0,056	0,029	0,042	0,072	0,086	0,055	0,071	0,087	0,097	0,071	0,087
Xamarin	0,074	0,078	0,068	0,073	0,088	0,096	0,081	0,088	0,092	0,102	0,083	0,092
Ionic	0,084	0,089	0,080	0,084	0,117	0,128	0,102	0,118	0,136	0,142	0,129	0,136
Test	Test vzpostavitve aplikacije											
Dom.	0,763	0,845	0,662	0,781	1,085	1,174	1,018	1,085	1,295	1,329	1,244	1,297
Xamarin	2,618	2,664	2,530	2,651	1,514	1,600	1,386	1,516	1,549	1,809	1,311	1,537
Ionic	0,782	0,897	0,725	0,765	2,264	2,390	2,181	2,250	2,415	2,476	2,294	2,421

Kot že omenjeno v razdelku 3.3, je enakovredna primerjava različnih mobilnih naprav otežena. Zaradi tega je nadaljnja analiza rezultatov potekala na podlagi koeficientov sprememb povprečnega časa, potrebnega za izvedbo posameznega testa, pri čemer smo za izhodiščno vrednost vzeli rezultat testa domorodne aplikacije.

3.4.1 Test procesorja

Pri testiranju mobilnih pristopov smo najprej opazovali čas, potreben za izračun konstante π na 10.000 mest natančno. Slika 4 prikazuje koeficiente sprememb povprečnega časa izvedbe ter dejanske vrednosti procesorskega testa na različnih mobilnih platformah.

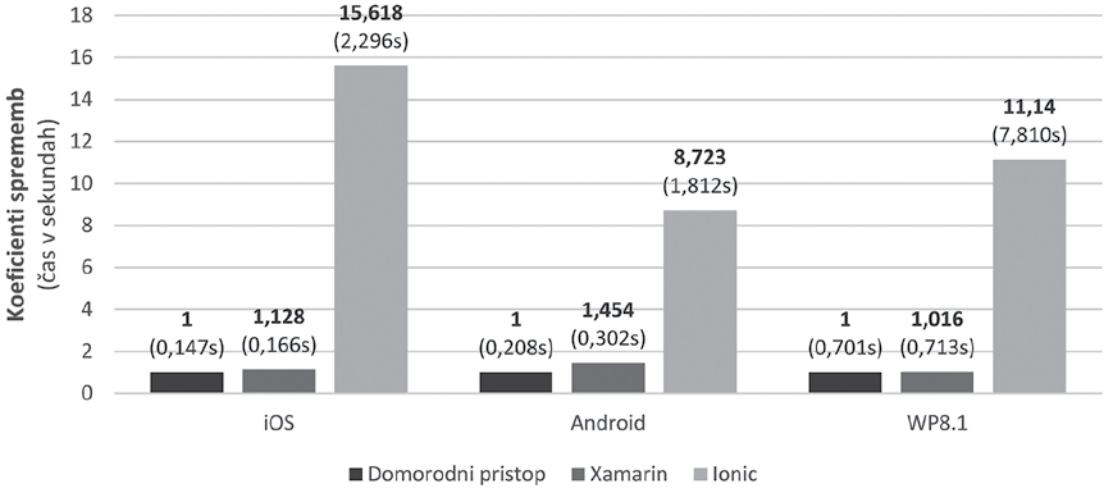


Slika 4: **Koeficienti sprememb (test procesorja)**

Iz slike je razvidno, da je v vseh primerih najkrajši čas za izvedbo testa potrebovala domorodna aplikacija, kateri je z manjšim odstopanjem sledilo ogrodje Xamarin (največje odstopanje v primeru platforme Android). V vseh primerih je največ časa potrebovala aplikacija, razvita s hibridnim pristopom Ionic, ki je v primeru platforme iOS od domorodnega pristopa odstopala za največ (8,6-krat).

3.4.2 Test pomnilnika

Testiranje pomnilnika mobilne naprave je potekalo na podlagi opazovanja časa zapisovanja zbirke 10,000.000 celih števil ter kopiranja nastalega seznama v nov seznam. Slika 5 prikazuje rezultate testa pomnilnika med mobilnimi platformami.

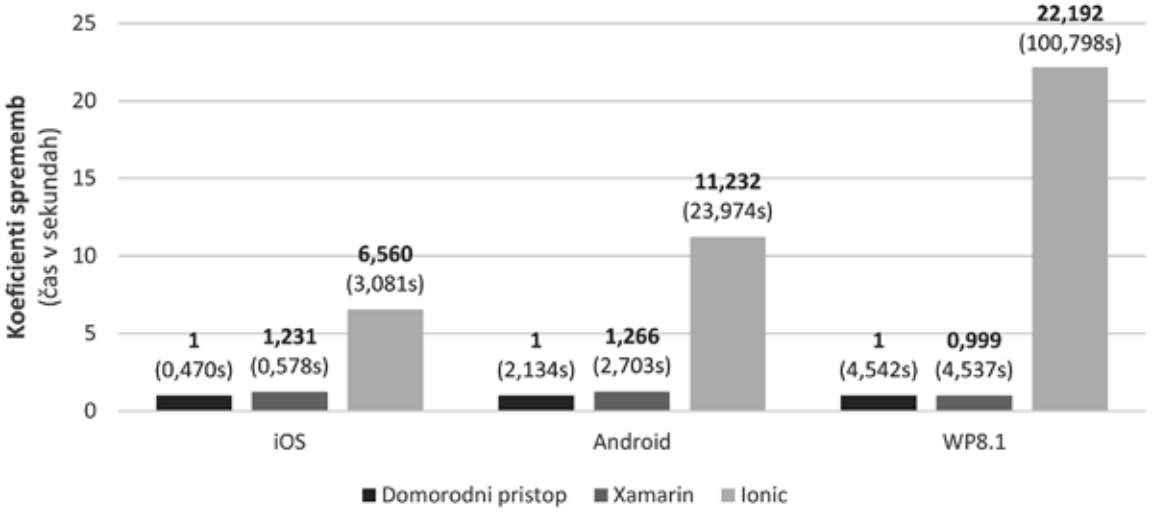


Slika 5: **Koeficienti sprememb (test pomnilnika)**

Rezultati testov pomnilnika so glede vrstnega reda pristopov podobni testu procesorja, a je v tem primeru prišlo do večjega odstopanja predvsem ogrodja Ionic, ki je odstopal za 8,7-krat v primeru platforme Android, za 11,1-krat v primeru platforme Windows Phone ter 15,6-krat v primeru platforme iOS. Razlog za večje odstopanje je mogoče pripisati tudi sami implementaciji zbirke na različnih platformah.

3.4.3 Test 2D grafike

Za testiranje izrisa vsebine smo definirali test, ki je na zaslon postopno izrisal 1.000 gumbov, pri tem pa smo merili čas, ki ga je aplikacija potrebovala, da je vse gumbe prikazala na zaslonu. Rezultati testa 2D grafike so predstavljeni na sliki 6.

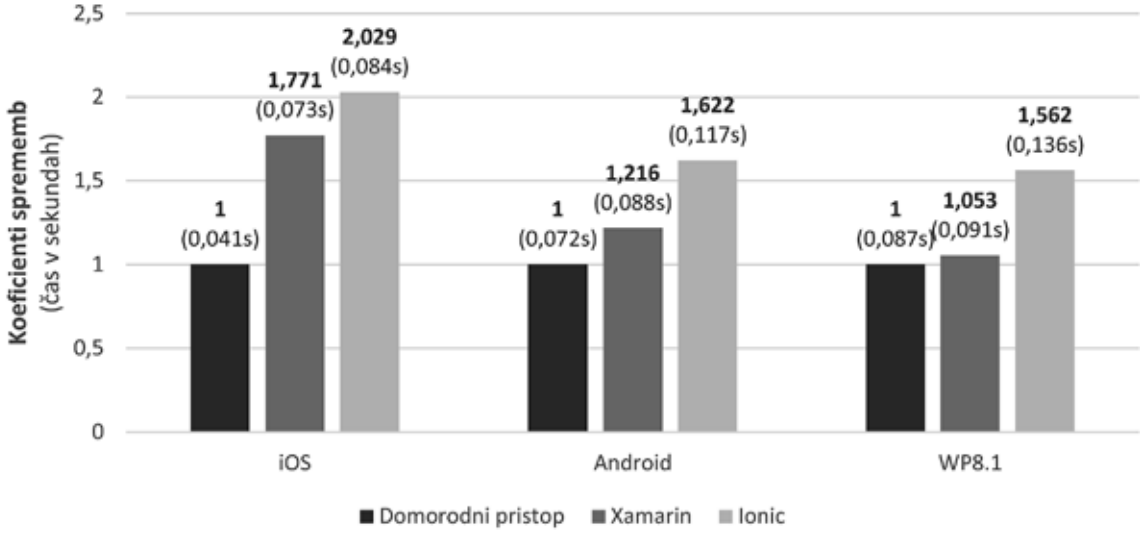


Slika 6: **Koeficienti sprememb (test 2D grafike)**

Izrisovanje vsebine na zaslon je grafično obremenjujoča operacija, kar je vplivalo predvsem na delovanje hibridne aplikacije. Na njeno zmogljivost na posamezni platformi vpliva tudi implementacija mobilne kontrole WebView, ki v primeru hibridnega pristopa skrbi za izrisovanje celotne vsebine mobilne aplikacije. Ionic se je tako v primeru vseh platform pri testu izkazal za najpočasnejšega. Najboljši rezultat je tudi v tem primeru dosegel domorodni pristop, kateremu je z manjšim odstopanjem sledil Xamarin.

3.4.4 Test odzivnosti

Delo z mobilno napravo poteka prek interakcije z različnimi vizualnimi kontrolami, ki se glede na uporabnikovo zahtevo odzovejo s spremembo vsebine. Raziskave so pokazale, da je za povprečnega uporabnika odziv instanten, če ta traja do 100 ms (Nielsen, 1993). Na podlagi tega smo definirali preprost test odziva mobilne aplikacije, pri katerem smo merili čas med interakcijo z aplikacijo in prikazom preprostega obvestila. Rezultati testa so prikazani na sliki 7.



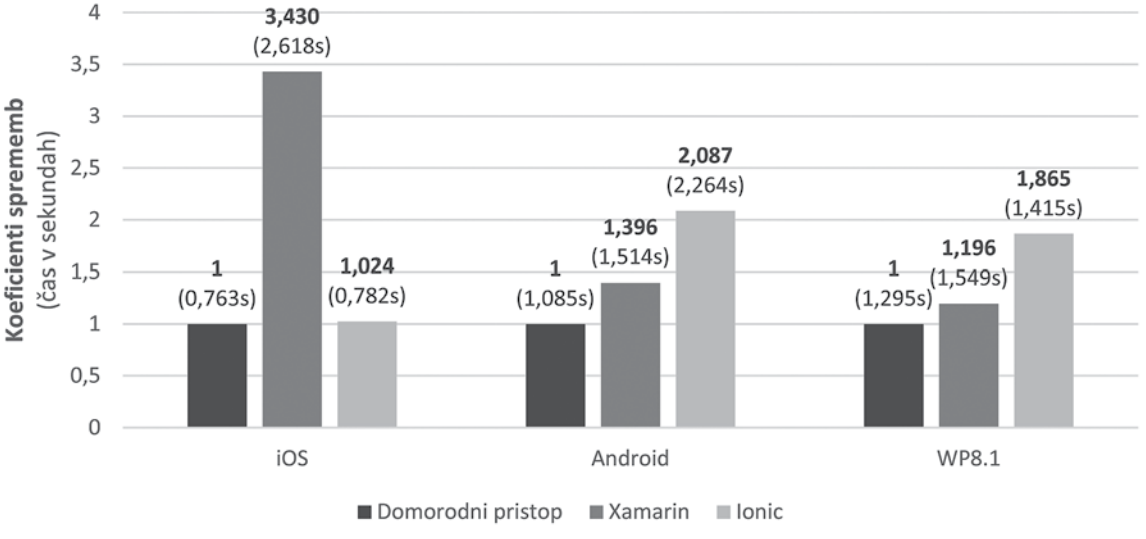
Slika 7: **Koeficienti sprememb (odzivni čas)**

Rezultati testov vseh pristopov so pokazali, da je odzivni čas aplikacije na vseh platformah krajši od mejne vrednosti 100 ms ali nekaj nad to, saj je najdaljši povprečni čas znašal le 0,136 sekunde. Pri tem moramo upoštevati tudi čas prikaza obvestila, ki pogosto vključuje tudi kratko animacijo. V vseh primerih je znova najboljše rezultate dosegla domorodna aplikacija, kateri sta sledili ogrodji Xamarin in Ionic.

3.4.5 Test vzpostavitve aplikacije

Ne glede na kakovost mobilne aplikacije si uporabniki želijo, da je aplikacija na voljo v kar najkrajšem možnem času. Slika 8 prikazuje koeficiente sprememb časa, ki ga aplikacija potrebuje za vzpostavitev in dosegljivost za uporabniško interakcijo.

Opazovanje časa zagona mobilne aplikacije je pokazalo, da največ časa v primeru platforme IOS potrebuje ogrodje Xamarin, in sicer 3,4-krat več v primerjavi z domorodnim pristopom, medtem ko je bil čas ogrodja Ionic ekvivalenten domorodnemu pristopu. V primeru platform Android in Windows Phone je največ časa za zagon potreboval Ionic in to za dvakrat več kot pri domorodnem pristopu v primeru platforme Android.



Slika 8: **Koeficienti sprememb (čas zagona)**

4 RAZPRAVA

Izvedba testov zmogljivosti razvitih aplikacij je bila podlaga za ocenjevanje primernosti različnih pristopov za razvoj zmogljivih ter zaradi tega učinkovitih mobilnih aplikacij za tri najbolj razširjene mobilne platforme. Pri vseh testih smo za izhodišče določili domorodni pristop, s katerim smo na podlagi koeficientov sprememb primerjali ogrodji Xamarin in Ionic. V tabeli 7 predstavljamo združene rezultate koeficientov sprememb posameznih zmogljivostnih testov skupaj s povprečnimi izračuni vseh testov.

Tabela 7: **Združeni rezultati sprememb koeficientov**

	Večplatformsko prevedeni pristop (Xamarin)			**Hibridni pristop (Ionic)**		
	iOS	Android	WP	iOS	Android	WP
Test procesorja	1,138	2,434	1,006	8,595	3,853	2,329
Test pomnilnika	1,128	1,454	1,016	15,618	8,723	11,140
Test 2D grafike	1,231	1,266	0,999	6,560	11,232	22,192
Odzivni čas	1,771	1,216	1,053	2,029	1,622	1,562
Čas nalaganja	3,430	1,395	1,196	1,024	2,087	1,865
Povprečje	1,740	1,553	1,054	6,765	5,503	7,817

Glede na združene rezultate lahko opazimo velike razlike med preučevanima pristopoma. Ugotavljamo, da je med domorodnim pristopom in večplatformsko prevedenim pristopom le minimalna razlika, saj skupni povprečni koeficient sprememb vseh testov ne preseže vrednosti 1,74 v primeru platforme iOS, se nekoliko zmanjša v primeru platforme Android (1,55) in je skorajda enakovreden domorodnemu pristopu v primeru platforme Windows Phone (1,054). Večje razlike lahko opazimo v primerjavi s hibridnim pristopom, saj je najboljši rezultat dosegla platforma Android, vendar je kar 5,5-krat počasnejša v primerjavi z domorodnim pristopom. Sledi platforma iOS s koeficientom 6,76, medtem ko je najslabši rezultat dosegla platforma Windows Phone s koeficientom 7,8.

Na podlagi rezultatov lahko trdimo, da je z vidika zmogljivosti implementirane aplikacije večplatformsko prevedeni pristop primerna alternativa domorodnemu pristopu za dnevno uporabo pri razvoju mobilnih aplikacij. V primeru hibridnega pristopa pa lahko opazimo, da je šibka točka aplikacije predvsem zmogljivostno zahtevna poslovna logika. Zaradi tega je pristop primeren predvsem za mobilne aplikacije, ki ne vsebujejo veliko poslovne logike ali pa se ta nahaja na strojno močnejšem viru, do katerega nato dostopamo preko spletnih storitev.

5 SKLEP

Napredek mobilnih tehnologij prinaša vse več različnih pristopov k večplatformskemu mobilnemu razvoju, še več pa je ogrodij, ki jih implementirajo. Če je še pred časom domorodni pristop veljal za edinega, ki zagotavlja največjo učinkovitost delovanja aplikacije, se razlika v primerjavi z drugimi pristopi vztrajno zmanjšuje. To je potrdila tudi naša primerjalna analiza, saj se je pri merjenju zmogljivosti večplatformsko prevedeni pristop (ogrodje Xamarin) v primeru vseh mobilnih platform najbolj približal domorodnemu pristopu. Glede na rezultate ocenjujemo, da je v primeru ogrodja Xamarin večplatformsko prevedeni pristop primerna alternativa za domorodni razvojni pristop.

Nasprotno se je hibridni pristop izkazal za najmanj zmogljivega. Eden od razlogov za to je sama arhitektura pristopa, ki temelji na izvajanju aplikacije znotraj mobilne komponente WebView, katere zmogljivost je odvisna od zmogljivosti mobilnega brskalnika platforme. Pomanjkljivosti se nanašajo predvsem na strojno zahtevnejša opravila, kar pomeni, da je pristop lahko še vedno zanimiv za razvoj strojno nezahtevnih aplikacij ali takšnih, ki tovrstna opravila odlagajo na zmogljivejše vire.

Kljub razlikam v rezultatih je treba omeniti, da ti temeljijo predvsem na dveh specifičnih ogrodjih, kar pomeni, da lahko spremenjena izbira ogrodij vpliva na drugačne rezultate. Pri izbiri večplatformskih

ogrodij je treba prav tako upoštevati, da je zmogljivost le eden izmed faktorjev in da je treba pri odločitvi upoštevati tudi že obstoječo bazo znanja, kompleksnost grajenih aplikacij ter vse prednosti in slabosti ogrodja.

Rezultati primerjalne analize so namenjeni tako novim kot tudi že izkušenim razvijalcem mobilnih aplikacij, ki želijo na hiter in učinkovit način razviti zmogljive večplatformske mobilne aplikacije.

V prihodnje nameravamo raziskave na področju večplatformskega pristopa razširiti tako z vključevanjem večjega števila pristopov kot s primerjavo večjega števila različnih ogrodij znotraj enega samega pristopa. Na ravni vzorčne aplikacije nameravamo vključiti tudi teste 3D animacij ter obstoječe teste obogatiti z dodatnimi scenariji. Za večjo zanesljivost podatkov na posameznih platformah nameravamo vključiti še večje število naprav za vsako izmed testiranih platform.

6 LITERATURA

Dostopnost vseh elektronskih virov je bila potrjena dne 30. 7. 2016.

[1] Adinugroho, T. Y. in Gautama, J. B. (2015). Review of Multi-platform Mobile Application Development Using WebView: Learning Management System on Mobile Platform. *Procedia Computer Science*, *59*, 291–297. http://doi.org/10.1016/j.procs.2015.07.568.

[2] Allen, J. (2011). The Death and Rebirth of Mono. https://www.infoq.com/news/2011/05/Mono-II.

[3] Bouras, C., Papazois, A. in Stasinos, N. (2014). A framework for cross-platform mobile web applications using HTML5. V: *Proceedings - 2014 International Conference on Future Internet of Things and Cloud, FiCloud 2014* (str. 420–424). http://doi.org/10.1109/FiCloud.2014.75.

[4] Charkaoui, S., Adraoui, Z. in Habib Benlahmar, E. (2014). Cross-platform mobile development approaches. *Information Science and Technology (CIST), 2014 Third IEEE International Colloquium in*, 188–191. http://doi.org/10.1109/CIST.2014.7016616.

[5] Cisco. (2016). Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015–2020 White Paper. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html.

[6] Cordova. (2012). Cordova Overview. http://cordova.apache.org/docs/en/latest/guide/overview/index.html.

[7] Dalmasso, I., Datta, S. K., Bonnet, C. in Nikaein, N. (2013). Survey, comparison and evaluation of cross platform mobile application development tools. *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 323–328. http://doi.org/10.1109/IWCMC.2013.6583580.

[8] Danielsson, J., Ameri, A., Friberg, M., Examiner, A. in Lindell, R. (2014). Comparison study of cross-platform developing tools for iPhone devices.

[9] DeCapua, T. (2015). Front-end vs back-end performance metrics for mobile apps. http://techbeacon.com/understanding-front-end-vs-back-end-performance-metrics-mobile-apps.

[10] Dhillon, S. (2015). An Evaluation Framework for Cross-Platform Mobile Application Development Tools. *Software: Practice and Experience*, *45*(10), 1331–1357. https://atrium2.lib.uoguelph.ca/xmlui/bitstream/handle/10214/4949/Dhillon_Sanjeet_201212_MSc.pdf?sequence=3&isAllowed=y.

[11] El-Kassas, W. S., Abdullah, B. A., Yousef, A. H. in Wahba, A. (2014). ICPMD: Integrated cross-platform mobile development solution. V: *Proceedings of 2014 9th IEEE International Conference on Computer Engineering and Systems, ICCES 2014* (str. 307–317). http://doi.org/10.1109/ICCES.2014.7030977.

[12] Guthrie, S. (2016). Microsoft to acquire Xamarin and empower more developers to build apps on any device. http://blogs.microsoft.com/blog/2016/02/24/microsoft-to-acquire-xamarin-and-empower-more-developers-to-build-apps-on-any-device/.

[13] Hösl, T. (2014). Using the Ionic Framework for Windows (Phone) 8.1 apps. https://www.hoessl.eu/2014/12/on-using-the-ionic-framework-for-windows-phone-8-1-apps/.

[14] Ionic. (2016). Ionic Document Overview. http://ionicframework.com/docs/overview/.

[15] ISO. (2016). ISO - International Organization for Standardization. http://www.iso.org/iso/home.htm.

[16] Joorabchi, M. E., Mesbah, A. in Kruchten, P. (2013). Real challenges in mobile app development. V: *International Symposium on Empirical Software Engineering and Measurement* (str. 15–24). http://doi.org/10.1109/ESEM.2013.9.

[17] Korf, M. in Oksman, E. (2016). Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options. https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options.

[18] Markov, D. (2015). Comparing The Top Frameworks For Building Hybrid Mobile Apps. http://tutorialzine.com/2015/10/comparing-the-top-frameworks-for-building-hybrid-mobile-apps/.

[19] NetMarketShare. (2016). Mobile/Tablet Operating System Market Share. https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=1.

[20] Nielsen, J. (1993). Response Times: The 3 Important Limits. https://www.nngroup.com/articles/response-times-3-important-limits/.

[21] Noeticforce. (2016). 10 best hybrid mobile App UI Frameworks: HTML5, CSS and JS.

[22] Palmieri, M., Singh, I. in Cicchetti, A. (2012). Comparison of cross-platform mobile development tools. V: *2012 16th International Conference on Intelligence in Next Generation Networks, ICIN 2012* (str. 179–186). http://doi.org/10.1109/ICIN.2012.6376023.

[23] PassMark. (2016). PassMark PerformanceTest mobile - mobile device benchmark software. http://www.passmark.com/products/pt_mobile.htm.

[24] Perchat, J., Desertot, M. in Lecomte, S. (2013). Component based framework to create mobile cross-platform applications. V: *Procedia Computer Science,* Vol. 19, str. 1004–1011. http://doi.org/10.1016/j.procs.2013.06.140.

[25] R. M. de Andrade, P., B. Albuquerque, A., F. Frota, O., V. Silveira, R. in A. da Silva, F. (2015). Cross Platform App : A Comparative Study. *International Journal of Computer Science and Information Technology*, *7*(1), 33–40. http://doi.org/10.5121/ijcsit.2015.7104.

[26] Rahul Raj, C. P. in Tolety, S. B. (2012). A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. V: *2012 Annual IEEE India Conference, INDICON 2012* (str. 625–629). http://doi.org/10.1109/INDCON.2012.6420693.
[27] Rajput, M. (2016). Pros and Cons of Developing Cross Platform Mobile Apps. http://www.smallbizdaily.com/pros-cons-developing-cross-platform-mobile-apps/.
[28] Smutný, P. (2012). Mobile development tools and cross-platform solutions. V: *Proceedings of the 2012 13th International Carpathian Control Conference, ICCC 2012* (str. 653–656). http://doi.org/10.1109/CarpathianCC.2012.6228727.
[29] Statista. (2016a). Number of apps available in leading app stores as of June 2016. http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.
[30] Statista. (2016b). Number of mobile phone users worldwide from 2013 to 2019 (in billions). http://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/.
[31] Titanium. (2016). Titanium pricing. http://www.appcelerator.com/pricing/.
[32] Xamarin. (n. d.). Xamarin. https://www.xamarin.com/.
[33] Xamarin. (2016a). Introduction to Mobile Development. https://developer.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development/.
[34] Xamarin. (2016b). Understanding the Xamarin Mobile Platform. https://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_1_-_understanding_the_xamarin_mobile_platform/.
[35] Xanthopoulos, S. in Xinogalos, S. (2013). A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications. *Proceedings of the 6th Balkan Conference in Informatics*, 213–220. http://doi.org/10.1145/2490257.2490292.

Boris Ovčjak je asistent za področje informatike na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Njegovo raziskovalno področje zajema področje mobilnih in spletnih tehnologij, računalništvo v oblaku in storitveno usmerjena arhitektura, vključno z vidikom sprejetosti omenjenih tehnologij. Sodeloval je na več projektih s področja razvoja informacijskih sistemov, prav tako pa je avtor in soavtor več strokovnih in znanstvenih prispevkov.

▪

Gregor Polančič je docent na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Med njegova raziskovalna področja spadajo tehnološki, metodološki in uporabniški vidiki sistemov za komuniciranje, sodelovanje in upravljanje poslovnih procesov, vključno z implikacijami sodobnih storitveno usmerjenih informacijskih rešitev na omenjena področja.

▪

Tatjana Welzer Družovec je redna profesorica za področje informatike na Fakulteti za elektrotehniko, računalništvo in informatiko, Univerze v Mariboru. Je nosilka več predmetov s področja podatkovnih baz, varnosti in zasebnosti ter medkulturnega komuniciranja. Je vodja Laboratorija za podatkovne tehnologije - LPT. Njeno raziskovalno področje je predvsem povezano z modeliranjem podatkovnih baz in širše sistemov, kjer se upoštevajo tudi vplivi ekspertnih okolji, različnih kultur in ponovne uporabe že obstoječih rešitev. Tako kot drugi člani LPT je aktivna v številnih mednarodnih projektih. Rezultate raziskav je objavila v številnih znanstvenih in strokovni prispevkih v domači in tuji literaturi.