C# 9 i .NET 5
arhitektura softvera

Upotreba mikroservisa, DevOps platforme i Azure projektnih obrazaca u arhitekturi softverskih rešenja

Prevod IV izdanja

Gabriel Baptista
Francesco Abbruzzese

kompjuter biblioteka

Packt›

C# 9 i .NET 5
arhitektura softvera

Skenirajte QR kod, registrujte knjigu i osvojite nagradu

Softverska arhitektura je praksa primene struktura i sistema koji ubrzavaju proces razvoja softvera i podižu kvalitet aplikacije. Uspešan arhitekta softvera mora posedovati znanje i ključne veštine i poznavati najbolje prakse predstavljene u ovom revidiranom i proširenom drugom izdanju, koje obuhvata i najnovije .NET 5 i C#9 funkcije.

Ovo izdanje sadrži dodatno objašnjenje principa softverske arhitekture, uključujući i nova poglavlja o sistemskim platformama Azure Service Fabric, Kubernets i Blazor. Takođe, sadrži obimniju diskusiju o bezbednosti, mikroservisima i DevOps platformi, kao i savete za primenu GitHub hosting platforme u ciklusu razvoja softvera.

Prvo ćete naučiti da transformišete korisničke zahteve u arhitektonske potrebe i utvrdićete razlike između funkcionalnih i nefunkcionalnih zahteva. Zatim ćete naučiti da izaberete pravo Cloud rešenje za svoju infrastrukturu, uz pažljivu primenu faktora koji pomažu pri upravljanju aplikacijom u Cloud okruženju.

Nakon toga ćete otkriti obrasce dizajna softvera i razne softverske pristupe koji će vam omogućiti da rešite uobičajene razvojne probleme. Na kraju, moći ćete da kreirate i isporučite visoko skalabilne aplikacije, koje ispunjavaju poslovne zahteve organizacije za koju radite i koje su spremne za upotrebu.

Naučićete:

- različite tehnike prevazilaženja stvarnih arhitektonskih izazova i razmatranja dizajnerskih rešenja
- da primenjujete slojevitu arhitekturu, arhitekturu orjentisanu na usluge (SOA) i mikroservise
- da efikasno upravljate mikroservisima pomoću alata Containers, Docker, Kubernets i Blazor
- da ubrzate isporuku globalnih rešenja upotrebom Azure alata i funkcija
- da programirate i održavate Azure funkcije pomoću jezika C#9 i njegovih najnovijih funkcionalnosti
- kada je razvoj vođen testiranjem (TDD) najbolji pristup razvoju softvera
- pisanje automatizovanih funkcionalnih testova
- najbolje DevOps principe koji omogućavaju okruženje kontinuirane integracije/kontinuirane isporuke (CI/CD)

ISBN: 978-86-7310-563-5

kompjuter biblioteka
osnovana 1986.
www.kombib.rs

Packt›

C# 9 i .NET 5

arhitektura softvera

Gabriel Baptista
Francesco Abbruzzese

Prevod II izdanja

kompjuter biblioteka

Packt›

Izdavač:

kompjuter biblioteka

Obalskih radnika 4a, Beograd

Tel: 011/2520272

e-mail: kombib@gmail.com

internet: www.kombib.rs

Urednik: Mihailo J. Šolajić

Za izdavača, direktor:
Mihailo J. Šolajić

Autor: Gabriel Baptista
Francesco Abbruzzese

Prevod: Slavica Prudkov

Lektura: Nemanja Lukić

Slog: Zvonko Aleksić

Znak Kompjuter biblioteke:
Miloš Milosavljević

Štampa: „Pekograf“, Zemun

Tiraž: 500

Godina izdanja: 2021.

Broj knjige: 540

Izdanje: Prvo

ISBN: 978-86-7310-563-5

Software Architecture with C# 9 and .NET 5

Gabriel Baptista

Francesco Abbruzzese

ISBN 978-1-80056-604-0

O AUTORIMA

Gabriel Baptista je softverski arhitekta, tim-lider koji primenjuje Microsoft platformu na širok raspon projekata, od maloprodaje do industrije. Stručnjak je za Azure rešenja i objavljivani profesor računarstva koji predaje programiranje, softversko inženjerstvo i arhitekturu. Govori na Kanalu 9 (Channel 9), jednoj od najprestižnijih .NET stek zajednica i su-osnivač je startapa SMIT, za programska rešenja koja slede DevOps filozofiju kao ključ za isporučivanje korisničkih potreba.

Moja draga porodica, Murilo, Heitor i Denise, uvek me ohrabruje da idem dalje. Moji roditelji, Elisabeth i Virgílio, i moje bake, Maria i Lygia, me uvek motivišu i inspiršu. Posebno sam zahvalan Packt timu, koji se potrudio da proizvede odlično novo izdanje knjige.

Francesco Abbruzzese je autor biblioteka MVC Controls Toolkit i Blazor Controls Toolkit. Doprinosi širenju i evangelizaciji Microsoft veb steka od prve verzije, ASP.NET MVC. Njegova kompanija, Mvcct Team, nudi veb aplikacije, alate i servise za veb tehnologije. Sa AI sistema, gde je implementirao jedan od prvih sistema za podršku odlučivanju za finansijske institucije, prešao je na 10 najboljih naslova video igara, kao što je Puma Street Soccer.

Mojim voljenim roditeljima, kojim dugujem sve. Posebno sam zahvalan celom Packt osoblju i recenzentima koji su doprineli opštem poboljšanju koda ove knjige.

O RECENZENTIMA

Mike Goatly kodira od malih nogu, kada je dobio svoj prvi ZX Spectrum 48k. Njegova profesionalna karijera je putovanje kroz mnoge industrije, uključujući video na zahtev, fintech i trgovinu. Uvek je bio fokusiran na Microsoft stek i učio je C#/.NET od prve v1 beta verzije, a od 2014. godine koristi Azure za izgradnju softvera zasnovanog na cloud-u. Mike trenutno poseduje Azure Developer Associate i DevOps Engineer Expert sertifikate.

Želim da zahvalim svojoj divnoj supruzi i deci za njihovo strpljenje i što su mi dali vremena da pomognem u stvaranju ove knjige.

Kirk Larkin je stručnjak za C#, .NET i ASP.NET Core sa diplomom računarskih nauka. Vodeći je softverski inžinjer i Microsoft MVP sa više od 15 godina profesionalnog iskustva. On je glavni za pitanja o Stack Overflow-u na ASP.NET Core-u, autor Pluralsight Guides-a i primarni autor brojnih tema o ASP.NET Core dokumentaciji. Živi u Velikoj Britaniji sa suprugom i dve ćerke.

Predgovor

Ova knjiga obuhvata najčešće korišćene projektne obrasce i radne okvire koji su uključeni u moderne, distribuirane softverske arhitekture zasnovane na cloud-u. U ovoj knjizi govorimo o tome kada i kako da upotrebite svaki od obrazaca, predstavljanjem praktičnih scenarija iz stvarnog sveta. U ovoj knjizi su predstavljene i tehnike i procesi kao što su DevOps, mikroservisi, Kubernetes, neprekidna integracija i cloud računarstvo, pa ćete naučiti da razvijate najbolja softverska rešenja za svoje klijente.

Ova knjiga će vam pomoći da razumete kakav proizvod vaši klijenti očekuju od vas. Knjiga će vas voditi kroz procese rešavanja najvećih problema sa kojima se možete suočiti tokom razvoja. Takođe, sadrži savete „uradite" i „ne radite", koje bi trebalo da sledite kada upravljate aplikacijom u okruženju zasnovanom na cloudu. Učićete o različitim arhitekturnim pristupima, kao što su slojevite arhitekture, arhitekture orijentisane na servise, mikroservise, Single Page Applications (jednostranične aplikacije) i cloud arhitekture, i razumećete kako da ih primenite na specifične poslovne zahteve.

Na kraju ćete upotrebom Azure-a rasporediti kod u udaljena okruženja, ili na cloud. Svi koncepti u ovoj knjizi će biti objašnjeni pomoću praktičnih primera iz stvarnog sveta, gde su principi dizajna važni kada kreirate bezbedne i robusne aplikacije. Do kraja ove knjige moći ćete da razvijete i isporučite visoko skalabilne i bezbedne poslovne aplikacije, koje ispunjavaju zahteve poslovanja krajnjeg korisnika.

Vredi pomenuti da ova knjiga neće obuhvatiti samo najbolju praksu koju bi softverski arhitekta trebalo da prati za razvoj C# i .NET Core rešenja već, takođe, govori o svim okruženjima koja morate da savladate da biste razvili softverski proizvod u skladu sa najnovijim trendovima.

Ovo drugo izdanje knjige ima poboljšan kod i objašnjenja i prilagođeno je novim mogućnostima koje nude C# 9 i .Net 5. Dodali smo sve nove radne okvire i tehnologije koji su se pojavili protekle godine, kao što su gRPC i Blazor, a Kubernetes smo opisali detaljno u posebnom poglavlju.

Za koga je ova knjiga

Ova knjiga je za svakog softverskog arhitektu koji želi da poboljša svoje znanje na temu Azure Solutions-a sa C#-om. Takođe je za inženjere i starije programere koji žele da postanu arhitekte, ili žele da izgrade poslovne aplikacije pomoću .NET steka. Potrebno je iskustvo u radu sa C#-om i .NET-om.

Šta ova knjiga pokriva

Poglavlje 1, *Važnost softverske arhitekture* – objašnjava osnove softverske arhitekture. U ovom poglavlju ćemo predstaviti pravi način razmišljanja za suočavanje sa zahtevima klijenata, a zatim i opisati kako da izaberete prave alate, obrasce i radne okvire.

Poglavlje 2, *Nefunkcionalni zahtevi* – vodi vas kroz važnu fazu razvoja aplikacije, fazu prikupljanja i uzimanja u obzir svih ograničenja i ciljeva koje aplikacija mora da ispuni, kao što su skalabilnost, dostupnost, elastičnost, performanse, višenitni rad, interoperabilnost i bezbednost.

Poglavlje 3, *Dokumentovanje zahteva pomoću Azure DevOps platforme* – opisuje tehnike za dokumentovanje zahteva, programskih grešaka i drugih informacija o aplikacijama. Iako je većina koncepata uopštena, u poglavlju smo fokusirani na upotrebu Azure DevOps-a i GitHub-a.

Poglavlje 4, *Izbor najboljeg cloud rešenja* – daje opširan pregled alata i resursa koji su dostupni u cloud-u i na Microsoft Azure-u. Ovde ćete naučiti kako da pretražujete odgovarajuće alatke i resurse i kako da ih konfigurišete tako da zadovoljavaju vaše potrebe.

Poglavlje 5, *Primena arhitekture mikroservisa na vašu poslovnu aplikaciju* – pruža opširan pregled mikroservisa i Docker kontejnera. Ovde ćete naučiti kako arhitektura zasnovana na mikroservisima koristi sve mogućnosti koje nudi cloud, a i videćete kako da upotrebite mikroservise da biste postigli fleksibilnost, veliku propusnost i pouzdanost u cloud-u. Naučićete kako da upotrebite kontejnere i Docker za mešanje različitih tehnologija u arhitekturi, kao i kako da softversku platformu učinite nazavisnom.

Poglavlje 6, *Azure Service Fabric* - opisuje Azure Service Fabric koji je orkestrator mikroservisa specifičan za Microsoft. Ovde ćete implementirati jednostavne aplikacije zasnovane na mikroservisima.

Poglavlje 7, *Azure Kubernetes servis* – opisuje Azure implementaciju Kubernetes-a koji je de-facto standard za orkestraciju mikroservisa. Ovde ćete pakovati i raspoređivati aplikacije mikroservisa na Kubernetes.

Poglavlje 8, *Interakcija sa podacima u C#-u – Entity Framework Core* – objašnjava do detalja kako aplikacija može da komunicira sa različitim mehanizmima skladištenja pomoću **Object-Relational Mappings** (**ORMs**) funkcije, a posebno pomoću Entity Framework Core-a 5.0.

Poglavlje 9, *Kako da izaberete cloud skladište podataka* – opisuje glavne mehanizme za skladištenje koji su dostupni u cloud-u i Microsoft Azure-u. Ovde ćete naučiti kako da izaberete najbolje mehanizme za skladištenje, radi postizanja paralelizma čitanja/pisanja koji vam je potreban, kao i kako da ih konfigurišete.

Poglavlje 10, *Upotreba funkcija Azure platforme* – opisuje model računarstva bez servera i kako da ga upotrebite u Azure cloud-u. Ovde ćete naučiti kako da dodelite resurse cloud-a samo kada su oni potrebni za pokretanje nekih izračunavanja, pa da, prema tome, platite samo za aktuelno vreme izračunavanja.

Poglavlje 11, *Projektni obrasci i .NET 5 implementacija* – opisuje uobičajene softverske obrasce upotrebom .NET 5 primera. Ovde ćete učiti o važnosti obrazaca i najboljoj praksi njihove upotrebe.

Poglavlje 12, *Razumevanje različitih domena u softverskim rešenjima* – opisuje modernu Domain-Driven Design metodologiju softverskog razvoja, kako da je upotrebite da se suočite sa kompleksnim aplikacijama koje zahtevaju nekoliko domena znanja, kao i kako da je upotrebite da biste iskoristili prednosti arhitektura zasnovanih na cloud-u i mikroservisima.

Poglavlje 13, *Implementiranje ponovne upotrebe koda u C# 9* – opisuje obrasce i najbolju praksu maksimimalizacije višekratnosti koda u .NET 5 aplikacijama pomoću jezika C# 9. Takođe ćemo govoriti o važnosti refaktorizacije koda.

Poglavlje 14, *Primena servisno-orijentisanih arhitektura pomoću .NET Core frejmvorka* – opisuje servisno orijentisanu arhitekturu koja omogućava izlaganje funkcionalnosti aplikacije kao krajnje tačke na vebu ili u privatnoj mreži, tako da korisnici mogu da stupe u interakciju kroz različite tipove klijenata. Ovde ćete naučiti kako da implementirate krajnje tačke servisno-orijentisane arhitekture pomoću ASP.NET Core-a i gRPC-a, i kako da ih samo-dokumentujete pomoću postojećih OpenAPI paketa.

Poglavlje 15, *Predstavljanje ASP.NET Core MVC frejmvorka* – opisuje detaljno ASP.NET Core radni okvir. Ovde ćete naučiti kako da implementirate veb aplikacije zasnovane na **Model-View-Controller** (**MVC**) obrascu i kako da ih organizujete u skladu sa instrukcijama Domain-Driven Design-a, opisanog u *Poglavlju 12*, *Razumevanje različitih domena u softverskim rešenjima*.

Poglavlje 16, *Blazor WebAssembly* – opisuje novi Blazor radni okvir koji koristi moć WebAssembly-a za pokretanje .NET-a u pretraživaču korisnika. Ovde ćete naučiti kako da implementirate Single Page Applications u jeziku C#.

Poglavlje 17, *Najbolja praksa C# 9 kodiranja* – opisuje najbolju praksu koju bi trebalo da sledite kada razvijate .NET 5 aplikacije pomoću jezika C# 9.

Poglavlje 18, *Testiranje koda pomoću slučajeva jediničnog testa i TDD pristupom* – opisuje kako da testirate aplikacije. Ovde ćete naučiti kako da testirate .NET Core aplikacije pomoću xUnit-a i videćete kako lako možete da razvijete i održavate kod koji zadovoljava specifikacije, pomoću dizajna vođenog testiranjem.

Poglavlje 19, *Upotreba alata za pisanje boljeg koda* – opisuje metrike koje procenjuju kvalitet softvera i kako da ga merite pomoću alatki uključenih u Visual Studio.

Poglavlje 20, *Razumevanje DevOps principa* – opisuje osnove metodologije razvoja i održavanja DevOps softvera. Ovde ćete naučiti kako da organizujete neprekidnu integraciju/ciklus neprekidne isporuke vaše aplikacije. Takođe, opisuje kako da automatizujete ceo proces raspoređivanja, od kreiranja novog izdanja u izvornom skladištu, kroz različite korake testiranja i odobravanja, do finalnog raspoređivanja aplikacije u stvarnom proizvodnom okruženju. Ovde ćete naučiti kako da upotrebite Azure Pipelines i GitHub Actions da biste automatizovali ceo proces raspoređivanja.

Poglavlje 21, *Izazovi primene CI scenarija* – dopunjuje opis DevOps-a scenarijima neprekidne integracije.

Poglavlje 22, *Automatizacija za funkcionalne testove* – ovo poglavlje posvećeno je automatskim funkcionalnim testovima – odnosno, testovima koji automatski verifikuju da li je verzija cele aplikacije u skladu sa dogovorenim funkcionalnim specifikacijama. Ovde ćete naučiti kako da simulirate operacije korisnika pomoću alatki za automatizaciju i kako da upotrebite ove alatke, zajedno sa xUnit-om, za pisanje funkcionalnih testova.

Da biste izvukli maksimum iz ove knjige...

- Knjiga obuhvata mnogo tema. Važno je da ih razumete kao smernice kojima ćete, možda, želeti da se vratite više puta, u različitim okolnostima.
- Ne zaboravite da je potrebno da imate instaliran Visual Studio Community 2019, ili noviju verziju.
- Uverite se da razumete C# .NET principe.

Preuzmite primere datoteka sa kodom

Paket koda za ovu knjigu hostovan je na GitHub-u, na adresi https://bit.ly/3xacr13

Kod za knjigu možete preuzeti i sa našeg sajta: https://bit.ly/3gqS3D0

Preuzmite slike u boji

Takođe smo obezbedili PDF fajl koji sadrži kolorne slike snimaka ekrana/dijagrama, koji su upotrebljeni u ovoj knjizi: https://bit.ly/3xbUGys

Korišćene konvencije

U ovoj knjizi postoji veliki broj konvencija koje su upotrebljene za tekst.

`KodUTekstu:` Ukazuje na reči koda u tekstu, nazive tabela baze podataka, nazive direktorijuma, nazive fajlova, ekstenzije fajla, nazive putanja, kratke URL-ove, korisnički unos i Twitter statuse. Na primer: „ Oni su kopirani u finalni znakovni niz samo jednom kada pozovete metod `sb.ToString(),` da biste dobili finalni rezultat.„Blok koda je postavljen na sledeći način:

Blok koda je postavljen na sledeći način:

```
[Fact]
public void Test1()
{
     var myInstanceToTest = new ClassToTest();
     Assert.Equal(5, myInstanceToTest.MethodToTest(1));
}
```

Svi unosi ili ispisi komandne linije napisani su na sledeći način:

```
kubectl create -f myClusterConfiguration.yaml
```

Podebljana slova: Ukazuju na novi termin, važnu reč ili reči koje vidite na ekranu, na primer, u menijima ili okvirima za dijalog, takođe se prikazuje i u tekstu na ovaj način. Na primer: „U prozoru **Solution Explorer** prikazaće se opcija **Publish...** kada kliknete desnim tasterom miša.”

Upozorenja ili važne napomene se pojavljuju ovako.

Saveti i trikovi se pojavljuju ovako.

Stupite u kontakt

Povratne informacije naših čitalaca su uvek dobrodošle.

Štamparske greške: Iako smo preduzeli sve mere da bismo obezbedili tačnost sadržaja, greške se dešavaju. Ako pronađete grešku u ovoj knjizi bićemo vam zahvalni ako nam to javite. Otvorite stranicu knjige: https://bit.ly/3sFt4OQ i u komentaru knjige upišite grešku koju ste otkrili.

Piraterija: Ako pronađete ilegalnu kopiju naše knjige u bilo kojoj formi na internetu, molimo vas da nas o tome obavestite i date nam adresu lokacije, ili naziv web sajta. Pošaljite nam poruku na adresu copyright@packt.com i pošaljite nam link ka sumnjivom materijalu.

Postanite član Kompjuter biblioteke

Kupovinom jedne naše knjige stekli ste pravo da postanete član Kompjuter biblioteke. Kao član možete da kupujete knjige u pretplati sa 40% popustai učestvujete u akcijama kada ostvarujete popuste na sva naša izdanja. Potrebno je samo da se prijavite preko formulara na našem sajtu. Link za prijavu: http://bit.ly/2TxekSa

Skenirajte QR kod
registrujte knjigu
i osvojite nagradu

1

Razumevanje važnosti softverske arhitekture

Softverska arhitektura je jedna od najčešćih tema softverske industrije danas, a sigurno je da će njen značaj biti sve veći u budućnosti. Što više gradimo kompleksna i fantastična rešenja, to su nam više potrebne odlične softverske arhitekture, za održavanje. Ali, brzina kojom nastaju nova softverska rešenja neprestano raste, pa se i dalje javljaju nove arhitektonske mogućnosti. Iz tog razloga ste odlučili da pročitate ovu knjigu; iz tog razloga smo mi odlučili da napišemo njeno drugo izdanje.

Pisanje o ovoj važnoj temi, koja nudi toliko alternativnih tehnika i rešenja, uopšte nije jednostavan zadatak. Glavni cilj ove knjige nije da izgradimo iscrpnu i beskonačnu listu dostupnih tehnika i rešenja, nego da pokažemo kako su različite familije tehnika povezane i kako one, u praksi, utiču na konstrukciju održivog rešenja.

Potreba da se fokusiramo na kreiranje stvarnih, efikasnih, poslovnih rešenja je sve veća; korisnicima je uvek potrebno više novih funkcija u aplikacijama. Štaviše, potreba za čestim nadgradnjama aplikacija (zbog brzih promena tržišta) uvećava našu obavezu da obezbedimo sofisticiranu softversku arhitekturu i razvojne tehnike.

Ovim poglavljem obuhvaćene su sledeće teme:

- Razumevanje pojma softverske arhitekture
- Neki modeli procesa razvoja softvera koji vam, kao softverskom arhitekti, mogu biti od pomoći
- Proces prikupljanja odgovarajućih informacija za projektovanje visokokvalitetnog softvera
- Tehnike dizajniranja za pomoć u procesu razvoja
- Uvod u studiju slučaja ove knjige

Studija slučaja ove knjige će vas voditi kroz proces kreiranja softverske arhitekture za turističku agenciju pod nazivom **World Wild Travel Club** (**WWTravelClub**). Svrha ove studije slučaja je da vam pomogne da razumete teoriju koja je objašnjena u svakom poglavlju, kao i da obezbedi primer razvoja poslovne aplikacije korišćenjem servisa Azure, Azure DevOps, C# 9, .NET 5, ASP.NET, i drugih tehnologija predstavljenih u ovoj knjizi.

Do kraja ovog poglavlja moći ćete tačno da razumete šta je misija softverske arhitekture. Takođe, naučićete šta je Azure i kako da kreirate nalog na toj platformi. Takođe smo opisali softverske procese, modele i druge tehnike koje će vam omogućiti da vodite tim.

Šta je softverska arhitektura?

To što danas čitate ovu knjigu, treba da zahvalite računarskim naučnicima koji su odlučili da je razvoj softvera oblast inženjerstva. To se desilo u prošlom veku, krajem šezdesetih godina, kada su predočili da je način razvoja softvera prilično sličan načinu konstruisanja građevine. Otuda i naziv **arhitektura softvera**. Kao što arhitekta projektuje zgradu i nadgleda njenu izgradnju na osnovu datog projekta, glavni cilj softverskog arhitekte je da obezbedi da je softverska aplikacija dobro implementirana; a dobra implementacija zahteva projektovanje odličnog rešenja.

U profesionalnom razvojnom projektu morate da uradite sledeće:

- Da definišete zahteve klijenta
- Da projektujete odlično rešenje koje ispunjava date zahteve
- Da implementirate projektovano rešenje
- Da izvršite validaciju rešenja sa klijentom
- Da isporučite rešenje u radno okruženje

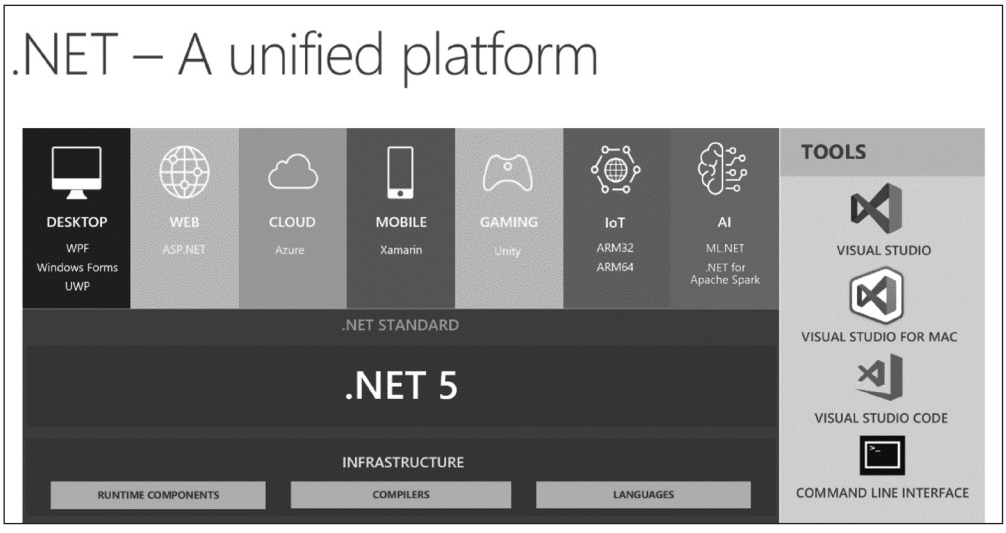
Softversko inženjerstvo definišite ove aktivnosti kao životni ciklus razvoja softvera. Svi teoretski modeli procesa razvoja softvera (waterfall, spiral, incremental, agile, itd) su nekako povezani sa ovim ciklusom. Bez obzira na to koji model upotrebite, ako tokom projektovanja ne izvršite osnovne zadatke koji su prethodno predstavljeni, nećete isporučiti prihvatljiv softver kao rešenje.

Glavna poenta projektovanja odličnih rešenja temelj je svrhe ove knjige. Morate da razumete da odlična rešenja iz stvarnog sveta sa sobom nose i nekoliko osnovnih ograničenja:

- Rešenje mora da zadovolji zahteve korisnika
- Rešenje mora da bude isporučeno na vreme
- Rešenje mora da se pridržava budžeta projekta
- Rešenje mora da isporuči dobar kvalitet
- Rešenje mora da garantuje bezbedan i efikasan budući razvoj

Odlična rešenja bi trebalo da budu održiva, a morate razumeti da ne postoji održiv softver bez odlične softverske arhitekture. U današnje vreme odlične softverske arhitekture zavise od modernih alatki i modernih okruženja da bi se savršeno uklopile u zahteve korisnika.

Zbog toga ćemo u ovoj knjizi upotrebiti neke odlične alatke, koje pruža Microsoft. Kompanija je objavila .NET 5 kao jedinstvenu platformu za softverski razvoj koja nam daje odličnu mogućnost da kreiramo fantastična rešenja.



Slika 1.1: .NET 5 platforma

.NET 5 je isporučen zajedno sa jezikom C# 9. Uzimajući u obzir pristup .NET-a, koji cilja toliko mnogo različitih platformi i uređaja, C# je sada jedan od najčešće upotrebljavanih programskih jezika na svetu, a koristi se i za male uređaje i za ogromne servere, u različitim operativnim sistemima i okruženjima.

Takođe, u ovoj knjizi ćemo upotrebiti **Azure**, Microsoft cloud platformu, gde se nalaze sve komponente koje kompanija obezbeđuje za izgradnju naprednih rešenja softverske arhitekture. Jedna od njih je **Azure DevOps**, okruženje za upravljanje životnim vekom aplikacije, gde možete da gradite rešenja pomoću najnovijih pristupa za razvoj softvera.

Biti softverski arhitekta podrazumeva razumevanje prethodno pomenutih tehnologija, ali takođe i mnogih drugih. U ovoj knjizi vas vodimo na putovanje na kom ćete, kao softverski arhitekta koji radi u timu, obezbediti optimalna rešenja korišćenjem raspoloživih alatki. Počnimo putovanje kreiranjem Azure naloga.

Kreiranje Azure naloga

Microsoft Azure je jedno od najboljih trenutno dostupnih cloud rešenja na tržištu. Važno je znati da ćemo unutar Azure platforme pronaći selekciju komponenti koje nam mogu pomoći da definišemo arhitektonska rešenja dvadeset prvog veka.

Ako želite da vidite različite komponente koje ima Microsoft Azure, pogledajte fantastičan veb sajt koji je kreirao Alexey Polkovnikov: `https://azurecharts.com/`.

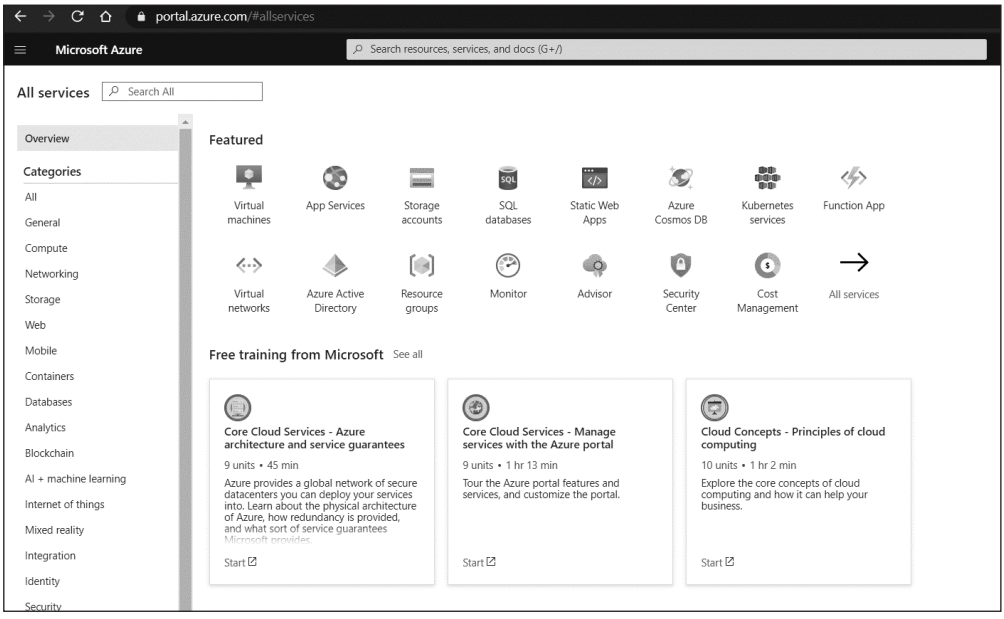
Ovaj pododeljak će vas voditi kroz proces kreiranja Azure naloga. Ako već imate nalog, možete da preskočite ovaj deo.

Počnite pristup u Azure posetom veb sajtu na adresi `https://azure.microsoft.com`. Tu ćete pronaći informacije koje su vam potrebne za početak pretplate. Prevod je obično podešen automatski.

1. Kada pristupite ovom portalu, moći ćete da se prijavite. Ako to nikada niste radili, moguće je upotrebiti opciju **Start free**, pa ćete moći da koristite neke Azure funkcije bez trošenja novca. Pogledajte opcije za besplatne planove, na adresi `https://azure.microsoft.com/en-us/free/`.
2. Proces kreiranja besplatnog naloga je prilično jednostavan, jer pratite obrazac koji zahteva da imate **Microsoft Account** ili **GitHub Account**.

3. Tokom procesa će biti zatražen broj vaše kreditne kartice, da bi se utvrdio vaš identitet i radi zaštitite od neželjene pošte i botova. Međutim, neće vam biti ništa naplaćeno, ako ne nadogradite nalog.
4. Da biste završili prijavu, potrebno je da prihvatite ugovor o pretplati, detalje o ponudi i iskaz o privatnosti.

Kada popunite obrazac moći ćete da pristupite Azure portalu. Kao što možete da vidite na sledećoj slici, panel prikazuje komandnu tablu koju možete da prilagođavate i meni sa leve strane, gde možete da podesite Azure komponente koje želite da koristite. U ovoj knjizi, vraćaćemo se na ovaj snimak ekrana da bismo podesili komponente koje će nam pomoći da kreiramo modernu softversku arhitekturu. Da bi prešli na sledeću stranicu, selektujte ikonicu menija sa leve strane (takozvani hamburger meni) i kliknite na **All services**.



Slika 1.2: Azure portal

Kada imate kreiran Azure nalog spremni ste da razumete kako softverski arhitekta može da vodi tim za razvoj softvera, koristeći sve mogućnosti koje Azure nudi. Međutim, važno je da imate na umu da bi softverski arhitekta trebalo da ide dalje od tih tehnologija, zato što ovu ulogu igraju ljudi od kojih se očekuje da definišu način isporuke softvera.

U današnje vreme softverski arhitekta ne projektuje samo osnovu softvera, on takođe određuje kako se vodi ceo proces razvoja i raspoređivanja softvera. U sledećoj temi ćemo opisati neke najčešće paradigme softverskog razvoja širom sveta. Počećemo opisom onoga što zajednica naziva tradicionalno softversko inženjerstvo. Nakon toga, opisaćemo agilne modele koji su promenili način na koji se u današnje vreme gradi softver.

Modeli procesa razvoja softvera

Kao softverski arhitekta, važno je da razumete neke uobičajene procese razvoja koji se trenutno koriste u većini preduzeća. Proces razvoja softvera definiše kako ljudi u timu kreiraju i isporučuju softver. Generalno, ovaj proces se svrstava u teoriju softverskog inženjerstva, a naziva se **model procesa razvoja softvera**. Od vremena kada je razvoj softvera definisan kao inženjerski proces predloženi su mnogi modeli procesa za razvoj softvera. Pogledajmo sada tradicionalne modele softvera, a zatim ćemo pregledati agilne, koji su trenutno uobičajeni.

Pregled tradicionalnih modela procesa razvoja softvera

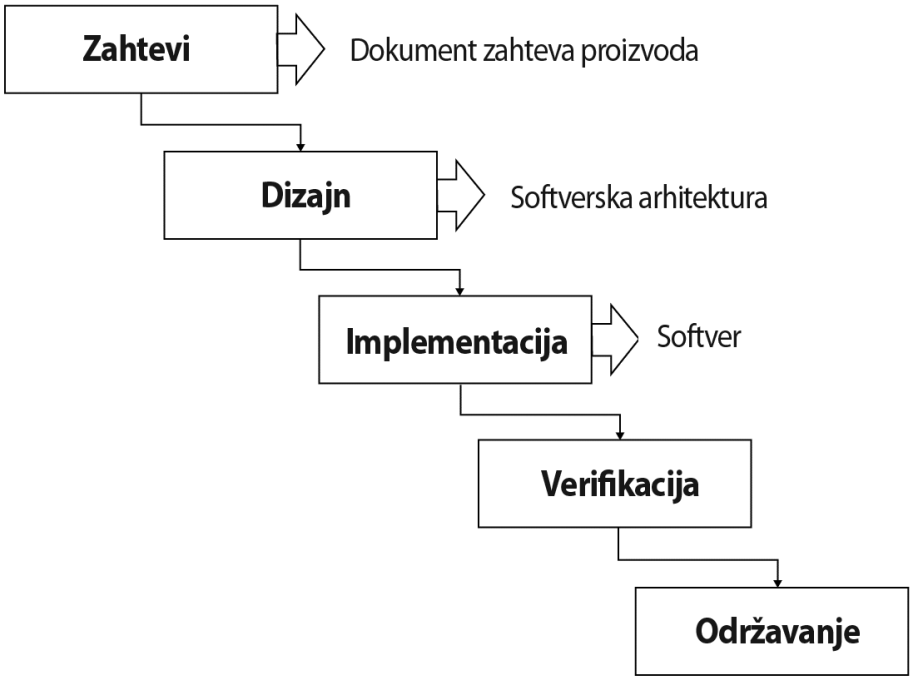
Neke od modela koji su predstavljeni u teoriji softverskog inženjerstva već smatramo tradicionalnim i prilično zastarelim. U ovoj knjizi ih nećemo sve opisati, ali ćemo vam pružiti kratka objašnjenja onih koji se i dalje koriste u nekim kompanijama - waterfall i incremental modeli.

Razumevanje principa waterfall modela

Ova tema možda deluje čudno u knjizi o softverskoj arhitekturi iz 2020. godine, ali da, i dalje ima kompanija u kojima najtradicionalniji model procesa razvoja softvera ostaje smernica za razvoj softvera. Ovaj proces izvršava sve osnovne zadatke u sekvenci. Svaki projekat razvoja softvera sastoji se od sledećih koraka:

- Zahtevi, kada je kreiran dokument zahteva od proizvoda, a to je osnova za razvoj softvera
- Dizajn, kada je razvijena softverska arhitektura, u skladu sa zahtevima
- Implementacija, kada je softver programiran
- Verifikacija, kada su izvršeni testovi u aplikaciji
- Održavanje, kada ciklus ponovo počinje, nakon isporuke

Pogledajmo dijagramski prikaz:

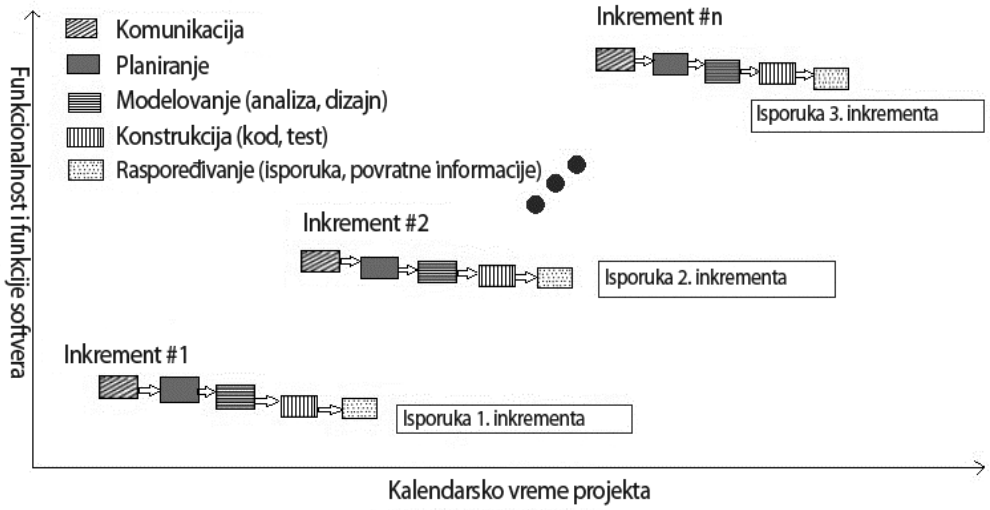


Slika 1.3: Ciklus razvoja modela waterfall (https://en.wikipedia.org/wiki/Waterfall_model)

Često upotreba waterfall modela izaziva probleme vezane za kašnjenje sa isporukom funkcionalne verzije softvera, kao i nezadovoljstvo korisnika, zbog odstupanja između očekivanja i isporučenog finalnog proizvoda. Osim toga, prema mom iskustvu, pokretanje testova u aplikaciji samo nakon završetka razvoja, uvek je veoma stresno.

Analiza incremental modela

Inkrementalni razvoj je pristup kojim je pokušano prevazilaženje najvećeg problema waterfall modela: korisnik može da testira rešenje samo na kraju projekta. Ideja ovog modela je da korisnicima pruži mogućnost da vrše interakciju sa rešenjem što je moguće ranije, da bi dobili povratne informacije koje će biti korisne tokom razvoja softvera.



Slika 1.4: Ciklus inkrementalnog razvoja (https://en.wikipedia.org/wiki/Incremental_build_model)

Incremental model predstavljen na prethodnoj slici bio je predstavljen kao alternativa za waterfall pristup. Ideja modela je da za svaki inkrement pokrene skup praksi koja se odnosi na razvoj softvera (**Komunikacija**, **Planiranje**, **Modelovanje**, **Konstrukcija** i **Raspoređivanje**). Iako je ovo ublažilo problem nedostatka komunikacije sa klijentom, za velike projekte, malo koraka je i dalje ostao problem, zato što su koraci ostali previše dugi.

Kada se inkrementalni pristup koristio u širokim razmerama – uglavnom krajem prošlog veka – prijavljeni su mnogi problemi vezani za projektnu birokratiju, zbog velike količine potrebne dokumentacije. Ovaj nezgodan scenario doveo je do uspona veoma važnog pokreta u industriji razvoja softvera – **agile razvojni postupak**.

Razumevanje agile modela procesa razvoja softvera

Početkom ovog veka razvoj softvera je smatran jednom od najhaotičnijih aktivnosti u inženjeringu. Procenat neuspešnih softverskih projekata bio je neverovatno visok, a ta činjenica je dokazala potrebu za drugačijim pristupom rešavanju fleksibilnosti u razvoju softvera.

Godine 2001. svetu je predstavljen Agile Manifesto, a od tada su predstavljeni različiti agile modeli procesa. Neki od njih su preživeli do danas, pa ostaju veoma uobičajeni.

Agile Manifesto je preveden na više od 60 jezika. Posetite veb sajt na adresi `https://agilemanifesto.org/`.

Jedna od najvećih razlika između agile modela i tradicionalnih modela je način interakcije programera sa klijentom. Poruka koju svi agile modeli prenose je ta da što brže isporučujete softver korisniku - to bolje. Ta ideja je ponekad zbunjujuća za programere softvera koji to shvataju kao – *pokušajmo da kodiramo, i to je sve!*

Međutim, postoji važno zapažanje Agile Manifesta, koje mnogi ljudi ne pročitaju kada počnu da koriste agile model:

Slika 1.5: Manifesto for Agile software development

Softverski arhitekta mora to da zapamti. Agilni procesi ne podrazumevaju nedostatak discipline. Štaviše, kada koristite agilni proces, brzo ćete saznati da ne postoji način da razvijete dobar softver bez discipline. Sa druge strane, kao softverski arhitekta, trebalo bi da razumete da *soft* znači fleksibilnost. Softverski projekat koji odbija da bude fleksibilan, teži da se uruši vremenom.

Osnova fleksibilnog pristupa je 12 principa agilnosti:

1. Neprekidno isporučivanje vrednog softvera, koji će zadovoljiti klijenta, mora da bude najviši prioritet svakog programera.
2. Menjanje zahteva mora da se shvati kao prilika da klijenta učinimo konkurentnijim.
3. Koristite nedeljni vremenski okvir za isporuku softvera.
4. Softverski tim moraju da čine poslovni ljudi i programeri.
5. Softverskom timu se mora verovati i tim bi trebalo da ima odgovarajuće okruženje za izvršenje projekta.
6. Najbolji način komunikacije sa softverskim timom je licem u lice.
7. Možete razumeti najveće postignuće softverskog tima kada softver zaista funkcioniše u proizvodnji.
8. Agilni model dobro funkcioniše kada obezbeđuje održiv razvoj.
9. Što više ulažete u tehnike i dobar dizajn, to ste agilniji.
10. Jednostavnost je od suštinske važnosti.
11. Što je tim bolje organizovan, kvalitet isporuke je bolji.
12. Softverski tim teži poboljšanju s vremena na vreme, analiziranjem i prilagođavanjem svojih procesa.

Čak 20 godina nakon izdanja Agile Manifesta, njegov značaj i veza sa aktuelnim potrebama softverskog tima ostaju nepromenjeni. Sigurno je da postoji mnogo kompanija u kojima ovaj pristup nije dobro prihvaćen, ali kao softverski arhitekta, trebalo bi da shvatite to kao priliku da promenite praksu i razvijate tim u kom radite.

Postoji mnogo tehnika i modela, predstavljenih softverskoj zajednici, koji koriste agilni pristup. U sledećim podnaslovima ćemo govoriti o Lean softverskom razvoju, Extreme Programming-u i Scrum-u, pa ćete, kao softverski arhitekta, moći da odlučite koji ćete pristup koristiti da biste poboljšali isporuku softvera.

Lean razvoj softvera

Nakon Agile Manifesta, zajednici je predstavljen pristup Lean softverskog razvoja, kao adaptacija dobro poznatog pokreta u automobilskom inženjerstvu: Toyota-in model proizvodnje automobila. Svetski Lean metod proizvodnje isporučuje visok nivo kvaliteta, čak i sa malo resursa.

Mary i Tom Poppendieck naveli su sedam Lean principa za razvoj softvera koji su, u stvari, povezani sa agile modelom i pristupom mnogih kompanija u ovom veku. Ja sam ih ovde naveo:

1. **Eliminišite otpad**: Otpadom možete smatrati bilo šta što će ometati isporuku stvarnih potreba klijenta.
2. **Ugradite kvalitet**: Organizacija koja želi da garantuje kvalitet mora da ga promoviše u procesima koji grade kod od početka, umesto da ga razmatra samo nakon što je kod testiran.
3. **Kreirajte znanje**: Kompanije koje su postigle izvrsnost imaju zajednički obrazac generisanja novog znanja disciplinovanim eksperimentisanjem, dokumentacijom i garancijom da se to znanje širi kroz celu organizaciju.
4. **Odložite predaju**: Planirajte odluke do njihove poslednje šanse, pre nego što izazovete štetu u projektu.
5. **Isporučite brzo**: Što brže isporučite softver, imate više eliminacije otpada. Kompanije koje se za konkurentnost bore primenom vremenske učestalosti, imaju značajnu prednost u odnosu na svoje konkurente.
6. **Poštujte ljude**: Davanje razumnih ciljeva timu, zajedno sa planovima koji će ih voditi u samo-organizovanju rutine, stvar je poštovanja ljudi sa kojima radite.
7. **Optimalizujte celinu**: Lean kompanija poboljšava ceo ciklus vrednosti; od momenta kada primi nov zahtev do zahteva koji isporučuje kada je softver završen.

Lean principi omogućavaju da tim, ili kompanija, poboljša kvalitet funkcija koje su klijentu zaista potrebne. Takođe, dovodi do smanjenja vremena utrošenog na funkcije koje neće biti upotrebljene do vremena kada je softver isporučen klijentu. U Lean pristupu, tim u isporuci softvera vode odluke o funkcijama koje su klijentu važne, a upravo je to ono što Agile Manifesto promoviše softverskim timovima.

Extreme Programming (ekstremno programiranje)

Neposredno pre izdanja Agile Manifesta, neki od učesnika koji su dizajnirali dokument, posebno Kent Beck, predstavili su svetu **Extreme Programming** (**XP**) **metodologiju** razvoja softvera.

XP se zasniva na vrednostima jednostavnosti, komunikacije, povratnih informacija, poštovanja i hrabrosti. Prema Beck-u, u njegovoj drugoj knjizi o ovoj temi, to je

kasnije smatrano društvenom promenom u programiranju. Ono što je sigurno je da promoviše ogromnu promenu u toku razvoja.

XP ukazuje na to da bi svakom timu trebalo obezbediti jednostavnost, tako da izvršavaju samo zadatak koji se od njih traži, da komuniciraju svakodnevno licem u lice, da predstavljaju softver rano da bi dobili povratne informacije, da poštuju stručnost svakog člana tima, da hrabro govore istinu o napretku i procenama i da smatraju rad tima kao celinu.

XP takođe sadrži i skup pravila. Ova pravila tim može da menja, ukoliko otkrije nešto što ne funkcioniše pravilno, ali je važno stalno podržavanje vrednosti metodologije.

Ova pravila su podeljena na planiranje, upravljanje, dizajniranje, kodiranje i testiranje. Don Wells je prikazao XP na veb sajtu `http://www.extremeprogramming.org/`. Iako su mnoge kompanije i stručnjaci strogo kritikovali neke ideje metodologije, postoji dobra praksa koja se i danas primenjuje:

- **Pisanje zahteva softvera pomoću korisničkih priča:** Korisničke priče su smatrane agilnim pristupom opisu potreba korisnika, zajedno sa testovima prihvatljivosti, koji će se koristiti kao garancija pravilne implementacije.
- **Deljenje softvera u iteracije i isporučivanje malih izdanja**: Praksu iteracije u softverskom razvoju brane sve metodologije nastale posle waterfall modela. Isporučivanje bržih verzija smanjuje rizik od neispunjenih očekivanja klijenta.
- **Izbegavanje prekovremenog rada i garantovanje održive brzine**: Ovo je jedan od najtežih zadataka sa kojim se softverski arhitekta mora suočiti, jer prekovremeni rad otkriva da nešto ne funkcioniše pravilno u procesu.
- **Zadržavanje jednostavnosti**: Dok razvijate rešenja, prilično je uobičajeno da pokušate da predvidite koje bi funkcije klijent želeo da ima. Ovaj pristup povećava kompleksnost razvoja i odlaže vreme izlaska na tržište. Drugačiji pristup će, u sistemu koji razvijate, prouzrokovati visoke troškove i, verovatno, nizak nivo funkcija koje su zaista korisne.
- **Refaktorizacija**: Pristup neprekidne refaktorizacije koda je dobar, jer omogućava razvoj softvera i garantuje poboljšanje dizajna, što će zaista biti potrebno zbog normalnih tehničkih promena na platformama koje koristite za razvoj.
- **Stalna dostupnost klijenta**: Ako pratite pravilo iz XP-a, trebalo bi u timu da imate stručnog klijenta. To je nešto što je teško dobiti, ali je suština da garantujete da je klijent uključen u sve delove razvoja. Kao bonus, ako je klijent blizak vašem timu, to znači da razume teškoće i stručnost tima, što doprinosi povećanom poverenju između strana.

- **Neprekidna integracija**: Ova praksa je jedna od osnova aktuelnog DevOps pristupa. Što je manja razlika između ličnog skladišta koda i glavnog skladišta koda, to je bolje.
- **Prvo kodirajte jedinični test**: Jedinično testiranje je pristup u kom programirate specifičan kod za testiranje jedne jedinice (klasa/metod) projekta. To se pominje u aktuelnoj razvojnoj metodologiji kao **Test-Driven Development** (**TDD**) (razvoj vođen testiranjem). Glavni cilj je da garantujete da svako poslovno pravilo ima sopstveni slučaj jediničnog testa.
- **Kod mora da bude napisan u skladu sa dogovorenim standardima**: Potreba određivanja standarda za kodiranje je povezana sa idejom da bez obzira na to koji programer radi na specifičnom delu projekta, kod mora da bude napisan tako da ga svaki drugi programer razume.
- **Programiranje u paru**: Programiranje u paru je još jedan pristup koji je teško postići u svakom trenutku razvoja softverskog projekta, ali sama tehnika – jedan programer kodira, a drugi aktivno posmatra i komentariše, kritikuje i savetuje – korisna je u kritičnim scenarijima.
- **Testovi prihvatanja**: Usvajanje testova prihvatanja za zadovoljavanje korisničke priče je dobar način da garantujete da nove verzije softvera neće prouzrokovati štetu po aktuelne potrebe. Još bolja opcija je da se ti testovi prihvatanja automatizuju.

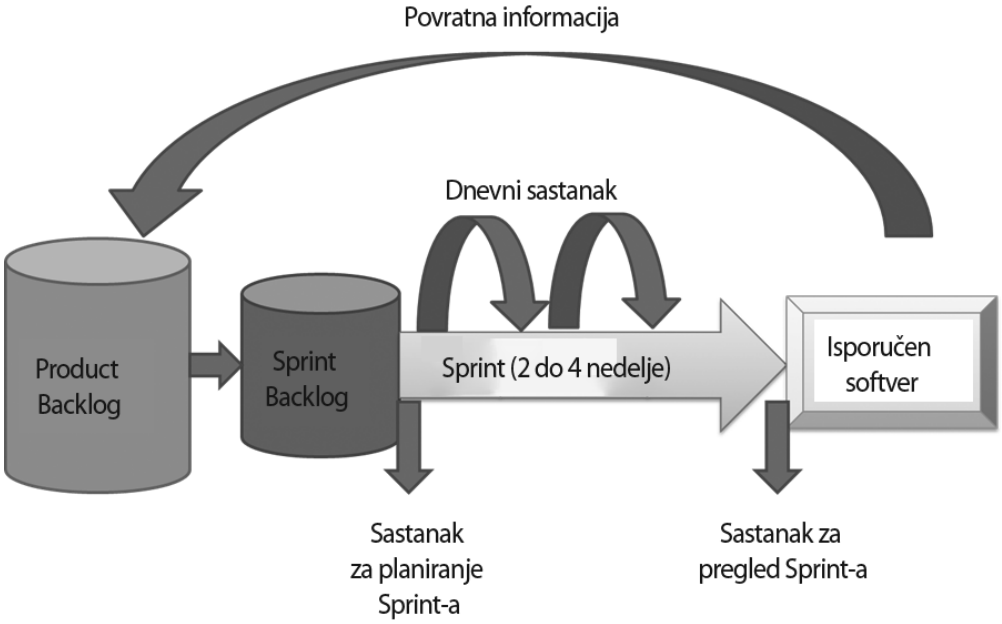
Vredi pomenuti da se mnoga od ovih pravila danas smatraju vitalnom praksom u različitim metodologijama razvoja softvera, uključujući DevOps i Scrum. Govorićemo o DevOps-u kasnije u ovoj knjizi, u *Poglavlju 20*, *Razumevanje DevOps principa*. Pogledajmo sada Scrum model.

Predstavljanje Scrum modela

Scrum je agilni model za upravljanje projektima razvoja softvera. Model potiče iz Lean principa i jedan je od najčešće korišćenih pristupa za razvoj softvera u današnje vreme.

Molim vas pogledajte na sledećem linku više informacija o Scrum radnom okviru: `https://www.scrum.org/`.

Kao što možete da vidite na sledećoj slici, osnova Scrum-a je fleksibilan zaostatak korisničkih zahteva (**Product Backlog**), o kom bi trebalo razgovarati u svakom agilnom ciklusu, koji nazivamo **Sprint**. Cilj Sprint-a (**Sprint Backlog**) određuje Scrum Team, sastavljen od Product Owner-a (vlasnika proizvoda), Scrum Master-a i Development Team-a. **Product Owner** je odgovoran za određivanje prioriteta koji će biti isporučeni u datom Sprint-u. Tokom Sprint-a, ta osoba će pomoći timu da razvije potrebne funkcije. Osoba koja vodi tim u Scrum procesu naziva se **Scrum Master**. Ta osoba vodi sve sastanke i procese.



Slika 1.6: Scrum proces

Važno je da napomenemo da Scrum proces ne govori kako bi softver trebalo da bude implementiran, niti koje će aktivnosti biti izvršene. Da ponovimo, morate zapamtiti osnove razvoja softvera, koje su opisane na početku ovog poglavlja; to znači da Scrum mora da bude implementiran zajedno sa modelom procesa. DevOps je jedan od pristupa koji može da vam pomogne da upotrebite model procesa razvoja softvera zajedno sa Scrum-om. Pogledajte *Poglavlje 20*, *Principi DevOps platforme*, da biste to bolje razumeli.

Prikupljanje odgovarajućih informacija za dizajniranje softvera visokog kvaliteta

Fantastično! Upravo ste počeli projekat razvoja softvera. Sada je vreme da primenite svoje znanje da biste isporučili najbolji mogući softver. Vaše prvo pitanje je verovatno – *Kako da počnem?* Pa, kao softverski arhitekta, vi ćete sami odgovoriti na ovo pitanje. Možete biti sigurni da će se vaš odgovor evoluirati sa svakim softverskim projektom koji vodite.

Definisanje procesa razvoja softvera je prvi zadatak. To se generalno radi tokom procesa planiranja projekta, ili se može desiti pre samog početka procesa.

Još jedan veoma važan zadatak je prikupljanje zahteva za softver. Bez obzira na to koji proces razvoja softvera odlučite da koristite, prikupljanje stvarnih potreba korisnika deo je teškog i neprekidnog posla. Naravno, postoje tehnike koje će vam pomoći da to uradite, a možete biti sigurni da će vam prikupljanje zahteva pomoći da definišete važne aspekte softverske arhitekture.

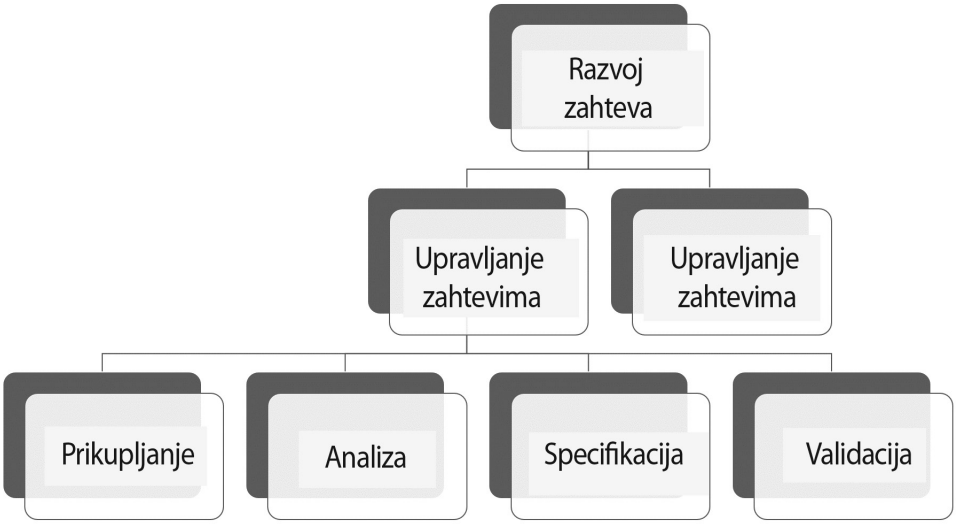
Većina stručnjaka za razvoj softvera ova dva zadatka smatra ključem za uspeh na kraju razvoja projekta. Kao softverski arhitekta, potrebno je da ih omogućite da biste izbegli što je moguće više problema dok vodite svoj tim.

Proces prikupljanja zahteva

Postoji mnogo različitih načina predstavljanja zahteva. Najtradicionalniji pristup se zasniva na pisanju savršene specifikacije pre početka analize. Agile metodi predlažu da, umesto toga, napišete korisničke priče, čim ste spremni da počnete razvojni ciklus.

Ne zaboravite: Ne pišete zahteve samo za korisnika; pišete ih i za sebe i za svoj tim.

Istina je da bi, bez obzira na to koji pristup odlučite da primenite u projektu, trebalo da pratite neke korake za prikupljanje zahteva. To mi nazivamo **inženjering zahteva**.



Slika 1.7: Proces inženjeringa zahteva

Tokom tog procesa, morate biti sigurni da je rešenje izvodljivo. U nekim slučajevima, analiza izvodljivosti je deo procesa planiranja projekta, pa ćete kada počnete prikupljanje zahteva već imati spreman izveštaj o izvodljivosti. Prema tome, pogledajmo ostale delove ovog procesa, jer će vam pružiti mnogo važnih informacija za softversku arhitekturu.

Otkrivanje tačnih potreba korisnika

Postoji mnogo načina da otkrijete šta je tačno potrebno korisniku za specifičan scenario. Ovaj proces je poznat kao prikupljanje ili otkrivanje. Generalno, to se može uraditi pomoću tehnika koje će vam pomoći da razumete korisničke zahteve. Sledi lista uobičajenih tehnika:

- **Moć mašte**: Ako ste stručnjak u oblasti u kojoj obezbeđujete rešenja, možete upotrebiti svoju maštu da biste otkrili zahteve novog korisnika. Iznošenje ideja se može obaviti zajednički, tako da grupa stručnjaka može definisati potrebe korisnika.
- **Upitnici**: Ovaj alat je koristan za detektovanje uobičajenih i važnih zahteva, kao što su broj i vrsta korisnika, upotreba sistema i uobičajeno korišćen **operativni sistem** (**OS**) i veb pretraživač.
- **Intervjui**: Intervjuisanje korisnika pomaže vam kao arhitekti da otkrijete zahteve korisnika koji možda nisu obuhvaćeni upitnicima ili vašom maštom.
- **Zapažanja**: Ne postoji bolji način da razumete svakodnevnu rutinu korisnika od provedenog dana sa njim.

Kada primenite jednu ili više ovih tehnika imaćete odlične, vredne informacije o potrebama korisnika.

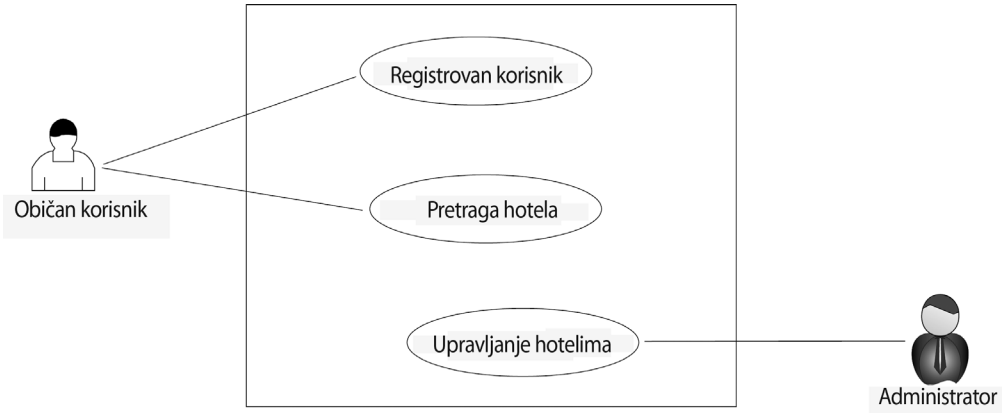
Ne zaboravite: Možete upotrebiti te tehnike u bilo kojoj situaciji kada je potrebno da prikupite zahteve, bez obzira na to da li su oni za ceo sistem ili za jednu priču.

U ovom trenutku možete da počnete analizu korisničkih potreba i da otkrijete zahteve korisnika i sistema. U sledećem odeljku videćete kako da to uradite.

Analiziranje zahteva

Kada otkrijete potrebe korisnika, vreme je da počnete analiziranje zahteva. Da biste to uradili, možete upotrebiti sledeće tehnike:

- **Kreiranje prototipa**: Prototipi su fantastični za pojašnjavanje i materijalizaciju sistemskih zahteva. U današnje vreme, imamo mnogo alata koji mogu pomoći u oponašanju interfejsa. Odličan alat otvorenog koda je **Pencil Project**. Više informacija o ovom alatu pronaći ćete na veb sajtu `https://pencil.evolus.vn/`.
- **Slučajevi upotrebe**: **Unified Modeling Language** (**UML**) (objedinjeni jezik za modelovanje) model slučaja upotrebe je opcija ako vam je potrebna detaljna dokumentacija. Model se sastoji od detaljne specifikacije i dijagrama. **ArgoUML** je još jedan alat otvorenog koda, koji vam može pomoći u ovom zadatku. Na *slici 1.8* prikazan je kreiran model:



Slika 1.8: Primer dijagrama slučaja upotrebe

Dok analizirate zahteve sistema moći ćete da razjasnite koje su tačno potrebe korisnika. To je korisno kada niste sigurni koji je stvarni problem koji bi trebalo da rešite, a mnogo je bolje nego prosto programiranje sistema i nadanje najboljem. Vreme uloženo u analizu zahteva je vreme uloženo u potonji bolji kod.

Pisanje specifikacija

Nakon što završite analizu, važno je da je registrujete kao specifikaciju. Dokument specifikacije možete napisati korišćenjem tradicionalnih zahteva ili korisničkih priča, koje se obično koriste u agilnim projektima.

Specifikacija zahteva predstavlja tehnički ugovor između korisnika i tima. Postoje neka osnovna pravila koja ovaj dokument mora da sledi:

- Sve zainteresovane strane moraju da razumeju tačno šta je napisano u tehničkom ugovoru, čak i ako nisu tehničari.
- Dokument mora da bude jasan.
- Potrebno je da klasifikujete svaki zahtev.
- Upotrebite buduće vreme za predstavljanje svakog zahteva:
  - Loš primer: Običan korisnik se sam registruje.
  - Dobar primer: Običan korisnik će se sam registrovati.
- Dvosmislenost i kontroverzu je potrebno izbegavati.

Neke dodatne informacije mogu biti korisne timu da bi bolje razumo kontekst projekta na kom će raditi. Slede neki saveti kako da dodate korisne informacije:

- Napišite uvodno poglavlje da biste preneli celu ideju rešenja.
- Kreirajte rečnik da biste olakšali razumevanje.
- Opišite vrstu korisnika koju će rešenje obuhvatati.
- Napišite funkcionalne i nefunkcionalne zahteve:
  - Funkcionalne zahteve je prilično jednostavno razumeti, zato što opisuju tačno šta će softver izvršavati. Sa druge strane, nefukcionalni zahtevi određuju ograničenja koja se odnose na softver, što znači skalabilnost, robusnost, bezbednost i performanse. Ove aspekte opisaćemo u sledećem odeljku.
- Priključite dokumente koji mogu pomoći korisniku da razume pravila.

Ukoliko odlučite da napišete korisničke priče, dobar savet koji bi trebalo da sledite je da pišete kratke rečenice koje predstavljaju svaki trenutak u sistemu, za svakog korisnika, kao što je sledeća:

```
As <user>, I want <feature>, so that <reason>
```

Ovaj pristup objašnjava tačan razlog iz kog će data funkcija biti implementirana. To je takođe dobar alat, koji će vam pomoći da analizirate najkritičnije priče i odredite prioritet uspeha projekta. Takođe, može da budu dobar za informisanje automatizovanih testova prihvatljivosti, koji bi trebalo da budu izgrađeni.

Razumevanje principa skalabilnosti, robusnosti, bezbednosti i performansi

Otkrivanje zahteva je zadatak koji vam omogućava da razumete softver koji razvijate. Međutim, kao softverski arhitekta, potrebno je da obratite pažnju na mnogo više od samih funkcionalnih zahteva za dati sistem. Razumevanje nefunkcionalnih zahteva je važno i jedna je od najranijih aktivnosti za softverskog arhitektu.

Pregledaćemo ih detaljnije u *Poglavlju 2*, *Nefunkcionalni zahtevi*, ali sada je važno da znate da principi skalabilnosti, robusnosti, bezbednosti i performansi moraju biti primenjeni na proces prikupljanja zahteva. Pogledajmo sada svaki od ovih koncepata:

- **Skalabilnost**: Kao softverskom programeru, globalizacija vam daje mogućnost da imate rešenje koje se koristi širom sveta. To je fantastično, ali je potrebno da vi, kao softverski arhitekta, dizajnirate rešenje koje obezbeđuje tu mogućnost. Skalabilnost je mogućnost aplikacije da povećava svoju procesnu moć čim za tim ima potrebe, zbog broja resursa koji su upotrebljeni.
- **Robusnost**: Bez obzira na to koliko je aplikacija skalabilna, ako ne možete da garantuje stabilno i neprekidno pokrenuto rešenje nećete biti mirni. Robusnost je važna za kritična rešenja, u kojima nemate mogućnost održavanja u bilo koje vreme zbog nekih problema koje aplikacija rešava. U mnogim industrijama se softver ne može zaustaviti i mnogo se rutina pokreće kada niko nije dostupan (tokom noći, tokom praznika, itd). Dizajniranje robusnog rešenja će vam dati slobodu da živite, dok vaš softver dobro funkcioniše.

- **Bezbednost**: Ovo je još jedna veoma važna oblast, o kojoj je potrebno raspravljati nakon faze prikupljanja zahteva. Svako brine zbog bezbednosti, a postoje različiti zakoni koji se bave njome i koji su preporučeni u različitim delovima sveta. Vi, kao softverski arhitekta, morate razumeti da bi bezbednost trebalo pružiti dizajnom. Ovo je jedini način da rešite sve probleme o kojima bezbednosna zajednica trenutno razgovara.
- **Performanse**: Proces razumevanja sistema koji ćete razvijati verovatno će vam dati ideju o tome šta će biti potrebno da uradite da biste dobili željene performanse sistema. Ovu temu morate razmotri sa korisnikom da biste identifikovali većinu "uskih grla" sa kojima se suočavate tokom faze razvoja.

Vredi pomenuti da su svi ovi koncepti, u stvari, zahtevi za novom generacijom rešenja potrebnih svetu. Ono što razlikuje dobar softver od neverovatnog softvera je količina posla urađenog za zadovoljenje zahteva projekta.

Pregled specifikacije

Kada napišete specifikaciju, vreme je da utvrdite sa zainteresovanim stranama da li se sa tim slažu. To možete da uradite na sastanku, ali možete i online, pomoću alatki za timski rad.

Potrebno je da predstavite sve prototipe, dokumente i informacije koje ste prikupili. Čim se svi slože sa specifikacijom, spremni ste da počnete istraživanje najboljeg načina za implementaciju dela projekta.

Vredi pomenuti da možete upotrebiti ovde opisan proces i za kompletan softver i za mali deo softvera.

Tehnike projektovanja kao koristan alat

Definisanje rešenja nije jednostavno. Određivanje njegove tehnologije uvećava poteškoće. Istina je da ćete tokom karijere softverskog arhitekte otkriti mnogo projekata u kojim će klijent doneti rešenje *spremno za razvoj*. Ovo može postati prilično komplikovano ako to rešenje smatrate ispravnim rešenjem; uglavnom će postojati arhitekturne i funkcionalne greške koje će, u budućnosti, prouzrokovati probleme u rešenju.

Postoje neki slučajevi kada je problem još gori – kada klijent ne zna najbolje rešenje za problem. Neke tehnike projektovanja mogu nam pomoći da rešimo ovaj problem, a ovde ćemo predstaviti dve tehnike: **Design Thinking** i **Design Sprint**.

Ono što morate da razumete je da ove tehnike mogu biti fantastična opcija za otkrivanje stvarnih zahteva. Kao softverski arhitekta posvećeni ste tome da svom timu pomognete da koristi odgovarajuće alate u pravo vreme, a ti alati mogu biti odgovarajuće opcije koje će obezbediti uspešnost projekta.

Design Thinking

Design Thinking je proces koji vam omogućava da prikupljate podatke direktno od korisnika, fokusirajući se na postizanje najboljih rezultata za rešavanje problema. Tokom ovog procesa, tim će imati mogućnost da upozna sve *osobe* koje će vršiti interakciju sa sistemom. To će imati odličan uticaj na rešenje, jer možete da razvijete softver fokusirajući se na korisničko iskustvo, što može imati fantastičan uticaj na rezultate.

Proces se zasniva na sledećim koracima:

- **Saosećajte**: U ovom koraku morate izvršiti istraživanje polja da biste otkrili brige korisnika. Tad ćete otkriti informacije o korisnicima sistema. Proces je dobar jer će vam pomoći da razumete za šta i za koga razvijate dati softver.
- **Definišite**: Kada znate za šta su korisnici zabrinuti, vreme je da definišete njihove potrebe da biste ih ispunili.
- **Zamišljajte**: Potrebe će pružiti mogućnost da osmislite neka moguća rešenja.
- **Prototip**: Ta rešenja možete razviti kao makete, da biste utvrdili da li su dobra.
- **Test**: Testiranje prototipa će vam pomoći da razumete prototip koji je najviše povezan sa stvarnim potrebama korisnika.

Ova tehnika se fokusira na ubrzanje procesa otkrivanja pravog proizvoda, razmatranjem **minimuma održivosti proizvoda (eng. minimum viable product** (**MVP**)). Sigurno je da će proces kreiranja prototipa pomoći zainteresovanim stranama da razumeju finalni proizvod i, istovremeno, angažovati tim da isporuči najbolje rešenje.

Design Sprint

Design Sprint je proces fokusiran na rešavanje kritičnih poslovnih pitanja kroz dizajn, tokom petodnevnog sprinta. Ovu tehniku je predstavio Google, a to je alternativa koja vam omogućava da brzo testirate i učite od ideje, kada je potrebno da izgradite i lansirate rešenje na tržište.

Proces se zasniva na jednoj radnoj nedelji koju stručnjaci posvećuju rešavanju pomenutog problema, u "ratnoj sobi" pripremljenoj za tu svrhu. Radna nedelja je podeljena ovako:

- **Ponedeljak**: Fokus ovog dana je identifikovanje cilja sprinta i mapiranje izazova za postizanje tog cilja.
- **Utorak**: Nakon razumevanja cilja učesnici počinju skiciranje rešenja. Vreme je da pronađete klijente za testiranje novog rešenja.
- **Sreda**: Ovo je dan kada je potrebno da tim odluči koje rešenje ima najveću šansu da reši problem. Osim toga, u sredu tim mora da iscrta ova rešenja na tabli, pripremajući plan prototipa.
- Četvrtak: Vreme je za kreiranje prototipa koji je planiran na tabli.
- **Petak**: Kad završi prototip, tim ga predstavlja klijentima i uči iz informacija dobijenih njihovom reakcijom na dizajnirano rešenje.

Kao što možete da vidite, u obe tehnike ubrzano prikupljanje reakcija klijenata dolazi od prototipa koji materijalizuje ideje tima u nešto opipljivije za krajnjeg korisnika.

Uobičajeni slučajevi u kojima proces prikupljanja zahteva utiče na rezultate sistema

Sve informacije o kojima smo do sada pružili u ovom poglavlju su korisne, ukoliko želite da dizajnirate softver koji prati principe dobrog inženjerstva. Ova diskusija se ne odnosi na razvijanje upotrebom tradicionalnog ili agile metoda, već je fokusirana na profesionalnu ili amatersku izgradnju softvera.

Takođe je dobra ideja znati više o nekim slučajevima u kojima je neuspešno izvršenje aktivnosti, o kojima ste čitali, prouzrokovalo neke probleme za softverski projekat. Sledeći slučajevi su namenjeni da opišu šta može poći naopako, kao i kako prethodno opisane tehnike mogu da pomognu razvojnom timu u rešavanju tih problema.

U većini slučajeva, jednostavna akcija mogla bi da garantuje bolju komunikaciju između tima i klijenta, a taj jednostavan tok komunikacije bi transformisao stvarni problem u stvarno rešenje. Istražimo sada tri uobičajena slučaja u kojima je prikupljanje zahteva uticalo na rezultate performansi, funkcionalnosti i upotrebljivosti.

Slučaj 1 – moj veb sajt je previše spor da bi otvorio datu stranicu!

Performanse su jedan od najvećih problema sa kojim ćete se vi, kao softverski arhitekta, suočiti tokom svoje karijere. Razlog što je ovaj aspekt bilo kog softvera toliko problematičan je to što nemamo beskonačne računarske resurse za rešavanje problema. Osim toga, troškovi izračunavanja su i dalje visoki, posebno ako govorite o softveru sa velikim brojem konkurentnih korisnika.

Ne možete rešiti probleme performanse pisanjem zahteva. Međutim, nećete imati problema ako ih pravilno napišete. Ideja je da zahtevi moraju predstavljati željene performanse sistema. Jednostavna rečenica, koja to opisuje, može pomoći celom timu koji radi na projektu:

Nefunkcionalni zahtev: Performanse – bilo koja veb stranica ovog softvera bi trebalo da odgovori u roku od najmanje 2 sekunde, čak i kada joj pristupa 1.000 korisnika istovremeno.

Prethodna rečenica čini da svi (korisnici, testeri, programeri, arhitekte, menadžeri, itd) budu sigurni da svaka veb stranica ima cilj koji bi trebalo da postigne. To je dobro za početak, ali nije dovoljno. Odlično okruženje za razvoj i raspoređivanje aplikacije je takođe važno. To je mesto gde .NET 5 može mnogo da vam pomogne; posebno ako govorite o veb aplikacijama, ASP.NET Core se smatra, u današnje vreme, jednom od najbržih opcija za isporučivanje rešenja.

Ako govorite o performansama kao softverski arhitekta, trebalo bi da razmotrite primenu tehnika koje su navedene u sledećim odeljcima, zajedno sa specifičnim testovima, da biste mogli garantovati ovaj nefunkcionalni zahtev. Takođe je važno pomenuti da će vam ASP.NET Core pomoći da ih jednostavno primenujujete, zajedno sa nekim **Platform as a Service** (**PaaS**) rešenjima, koje nudi Microsoft Azure.

Keširanje

Keširanje je odlična tehnika za izbegavanje redova, koji mogu da troše vreme, a generalno daju isti rezultat. Na primer, ako preuzimate modele automobila dostupne u bazi podataka, broj automobila u bazi podataka se može uvećati, ali se neće menjati. Kada imate aplikaciju koja konstantno pristupa modelima automobila, dobra praksa je keširanje te informacije.

Važno je da razumete da je keš uskladišten u backend-u i da se deli u celoj aplikaciji (*keširanje u memoriji*). Ono na šta bi trebalo da se fokusirate kada radite na skalabilnom rešenju je da možete konfigurisati *distribuiran keš* da biste ga razrešili pomoću Azure platforme. U stvari, ASP.NET obezbeđuje oba keša, pa možete odlučiti da upotrebite onaj koji vam više odgovara. U *poglavlju 2*, *Nefunkcionalni zahtevi*, opisani su aspekti skalabilnosti u Azure platformi.

Primena asinhronog programiranja

Kada razvijate ASP.NET aplikacije, imajte na umu da bi aplikacije trebalo da budu dizajnirane za istovremeni pristup više korisnika. Asinhrono programiranje vam omogućava da to jednostavno uradite, pružajući vam ključne reči `async` i `await`.

Osnovni koncept ovih ključnih reči je da `async` omogućava da se bilo koji metod pokreće asinhrono. Sa druge strane, ključna reč `await` omogućava vam da sinhronizujete poziv asinhronog metoda, bez blokiranja programske niti koja ga poziva. Ovaj jednostavan obrazac za razvoj će pomoći da se aplikacija pokreće bez "uskih grla" performansi i brzim reagovanjem. U ovoj knjizi govorićemo više o ovoj temi u *poglavlju 2*, *Nefunkcionalni zahtevi*.

Raspodela objekta

Jedan veoma dobar savet za izbegavanje manjka performansi je da razumete kako funkcioniše **Garbage Collector** (**GC**). GC je mehanizam koji će automatski osloboditi memoriju kada završite sa njenom upotrebom. Postoje neki veoma važni aspekti ove teme zbog kompleksnosti GC mehanizma.

Neke tipove objekata GC ne prikuplja ukoliko ih ne otpustite. Lista uključuje svaki objekat koji komunicira sa I/O-om, kao što su fajlovi i strimovanje. Ako ne koristite pravilno C# sintaksu za kreiranje i otpuštanje ove vrste objekata, doći će do curenja memorije, što će pogoršati performanse aplikacije.

Nepravilan način upotrebe I/O objekata je:

```
System.IO.StreamWriter file = new System.IO.StreamWriter(@"C:\sample.
txt");
file.WriteLine("Just writing a simple line");
```

Pravilan način upotrebe I/O objekata je:

```
using (System.IO.StreamWriter file = new System.IO.StreamWriter(@"C:\
sample.txt"))
{
file.WriteLine("Just writing a simple line");
}
```

Vredi pomenuti da ovaj ispravan pristup takođe obezbeđuje da će fajl biti napisan (naziva se Flush). U neispravnom primeru, sadržaji možda čak neće ni biti napisani u fajl. Čak i ako je prethodna praksa obavezna za I/O objekte, preporučljivo je nastaviti primenu na sve jednokratne objekte. U stvari, upotreba analizatora koda u rešenjima sa upozorenjima kao greškama sprečiće da slučajno napravite ove greške! To će pomoći GC-u i zadržaće aplikaciju pokrenutom sa odgovarajućom količinom memorije. U zavisnosti od tipa objekta, greške mogu da budu velike i na kraju će se javiti i neke druge loše stvari, na primer, iscrpljenje porta/veze.

Još jedan važan aspekt za koji bi trebalo da znate je da vreme koje GC provede u prikupljanju objekata ometa performanse aplikacije. Zbog toga, izbegavajte dodelu velikih objekata; u suprotnom, može doći do problema čekanja da GC završi svoj zadatak.

Dobijanje boljeg pristupa bazi podataka

Jedna od najčešćih slabosti performansi je pristup bazi podataka. Razlog zašto je ovo i dalje veliki problem je nedostatak pažnje prilikom pisanja upita ili lambda izraza za preuzimanje informacija iz baze podataka. U ovoj knjizi ćemo opisati Entity Framework Core u *poglavlju 8*, *Interakcija sa podacima u jeziku C# – Entity Framework Core*, ali je važno da znate šta da izaberete i koje su ispravne informacije o podacima za čitanje iz baze podataka. Filtriranje kolona i linija je imperativ za aplikaciju koja želi da postigne dobar performans.

Dobro je to što se najbolja praksa koja se odnosi na keširanje, asinhrono programiranje i dodelu objekata u potpunosti uklapa u okruženje baze podataka. Jedino je važno izabrati ispravan obrazac za dobijanje softvera boljih performansi.

Slučaj 2 – potrebe korisnika nisu pravilno implementirane

Što se više tehnologija koristi u različitim oblastima, teže je isporučiti tačno ono što je korisniku potrebno. Možda vam ova rečenica deluje čudno, ali morate razumeti da programeri, generalno, proučavaju kako da razviju softver, ali retko uče kako da zadovolje potrebe specifične oblasti. Naravno, nije lako naučiti kako da razvijete softver, ali je još teže razumeti specifične potrebe u specifičnim oblastima. Razvoj softvera u današnje vreme isporučuje softver za sve moguće tipove industrija. Pitanje koje se ovde postavlja je *kako programer, bez obzira na to da li je softverski arhitekta ili nije, može dovoljno da napreduje da bi isporučio softver u oblasti za koju je odgovoran?*

Prikupljanje zahteva softvera će vam pomoći u ovom teškom zadatku; pisanje zahteva će vam pomoći u organizaciji arhitekture sistema. Postoji nekoliko načina minimalizovanja rizika implementiranja nečeg drugačijeg od onoga što je korisniku zaista potrebno:

- Kreiranje prototipa interfejsa, da biste brže razumeli korisnički interfejs.
- Dizajniranje toka podataka za otkrivanje praznina između sistema i korisničke operacije.
- Česti sastanci radi ažuriranja aktuelnih potreba korisnika i prilagođavanja inkrementalne isporuke.

Ponovo, kao softverski arhitekta potrebno je da definišete kako će softver biti implementiran. Uglavnom, nećete vi biti taj koji će ga programirati, ali ćete uvek vi biti odgovorni za ovaj zadatak. Zbog toga, neke tehnike mogu da budu korisne da biste izbegli pogrešnu implementaciju:

- Zahtevi se pregledaju sa programerima da biste bili sigurni da oni razumeju šta je potrebno da razviju.
- Inspekcija koda radi potvrde unapred definisanog standarda koda. O tome ćemo govoriti u *poglavlju 19*, *Upotreba alata za pisanje boljeg koda*.
- Sastanci za eliminisanje prepreka.

Ne zaboravite, implementacija koja je usklađena sa potrebama korisnika je vaša odgovornost. Upotrebite svaku alatku koju možete da biste to ispunili.

Slučaj 3 – upotrebljivost sistema ne zadovoljava potrebe korisnika

Upotrebljivost je ključna tačka uspeha softverskog projekta. Način na koji je softver predstavljen i kako rešava problem može pomoći korisniku u odlučivanju da li želi da ga upotrebi, ili ne. Kao softverski arhitekta imajte na umu da je, u današnje vreme, obavezno isporučivanje softvera sa dobrom upotrebljivošću.

Postoje osnovni koncepti upotrebljivosti koje nećemo opisati u ovoj knjizi, ali dobar način da zadovoljite tačne potrebe korisnika, kada je reč o upotrebljivosti, je da razumete ko će koristiti softver. Design Thinking pristup vam može u tome mnogo pomoći, kao što smo i opisali ranije u ovom poglavlju.

Razumevanje korisnika pomaže da odlučite da li će se softver pokretati na veb stranici ili na mobilnom telefonu, ili čak u pozadini. To razumevanje je veoma važno softverskom arhitekti, jer će elementi sistema biti bolje predstavljeni, ukoliko pravilno mapirate ko će ih koristiti.

Sa druge strane, ukoliko vam to nije važno, isporučićete softver koji funkcioniše. To može biti dobro kratko vreme, ali neće u potpunosti zadovoljiti stvarne potrebe korisnika koji je tražio da dizajnirate softver. Morate imati na umu opcije i razumeti da je dobar softver dizajniran tako da se pokreće na mnogim različitim platformama i uređajima.

Biće vam drago da znate da je .NET 5 odlična međuplatformska opcija za to. Prema tome, možete da razvijate rešenja za pokretanje aplikacija u Linux, Windows, Android i iOS sistemima. Možete da pokrenete aplikacije na velikim ekranima, tabletima, mobilnim telefonima, pa čak i na dronovima! Možete da ugradite aplikacije na ploče za automatizaciju ili u HoloLens za mešovitu realnost. Softverske arhitekte moraju da budu otvorenog uma da bi dizajnirali tačno ono što je potrebno njihovim korisnicima.

Studija slučaja –World Wild Travel Club

Kao što smo pomenuli na početku ovog poglavlja, studija slučaja ove knjige vodi vas na putovanje kreiranja softverske arhitekture za turističku agenciju pod nazivom **World Wild Travel Club** (**WWTravelClub**).

WWTravelClub je turistička agencija koja je osnovana da bi promenila način na koji ljudi donose odluke o svom godišnjem odmoru i drugim putovanjima oko sveta. Za to je kreiran online servis koji svakom aspektu putnog iskustva obezbeđuje pomoć kluba stručnjaka posebno izabranih za svaku destinaciju.

Koncept ove platforme je da istovremeno možete da budete posetilac i stručnjak za destinaciju. Što više učestvujete kao stručnjak za destinaciju to ćete dobiti više bodova. Ove bodove možete razmeniti za karte koje su u prodaji online, upotrebom platforme.

Klijent je došao sa sledećim zahtevima za platformu. Važno je da znate da, generalno, klijenti nemaju zahteve spremne za razvoj. Zbog toga je veoma važan proces prikupljanja zahteva:

- Prikaz običnom korisniku
    - Promotivni paketi na početnoj stranici
    - Pretraga paketa
    - Detalji za svaki paket
        - Kupovina paketa
        - Kupovina paketa sa uključenim klubom stručnjaka
            - Komentari o iskustvu

    - Postavljanje pitanja stručnjaku
    - Procena stručnjaka
  - Registracija običnog korisnika
- Prikaz stručnjaka za destinaciju
  - Isti prikaz kao i za običnog korisnika
  - Odgovori na postavljena pitanja za stručnjaka za destinaciju
  - Upravljanje bodovima koje ste osvojili odgovaranjem na pitanja
  - Razmena bodova za karte
- Prikaz administratora
  - Upravljanje paketima
  - Upravljanje običnim korisnicima
  - Upravljanje stručnjacima za destinacije

Važno je napomenuti da WWTravelClub namerava da ima više od 100 stručnjaka za destinaciju po paketu, a ponudiće oko 1.000 različitih paketa putovanja širom sveta.

Razumevanje potreba korisnika i zahteva sistema

Da biste rezimirali potrebe korisnika WWTravelClub-a, možete da pročitate sledeće korisničke priče:

- `US_001`: Kao običan korisnik želim da pregledam promotivne pakete na početnoj stranici da bih lako mogao da pronađem svoje sledeće putovanje.
- `US_002`: Kao običan korisnik želim da pretražujem pakete koje ne mogu da pronađem na početnoj stranici, da bih mogao da istražim druge mogućnosti putovanja.
- `US_003`: Kao običan korisnik želim da vidim detalje paketa, da bih mogao da odlučim koji paket da kupim.
- `US_004`: Kao običan korisnik želim da se registrujem, da bih mogao da počnem kupovinu paketa.
- `US_005`: Kao registrovan korisnik želim da obradim plaćanje, da bih mogao da kupim paket.
- `US_006`: Kao registrovan korisnik želim da kupim paket sa uključenim preporukama stručnjaka, da bih mogao da imam ekskluzivno iskustvo putovanja.

- US_007: Kao registrovan korisnik želim da postavim pitanje stručnjaku, da bih mogao da otkrijem šta je najbolje da radim na putovanju.
- US_008: Kao registrovan korisnik želim da postavim komentar o svom iskustvu, da bih mogao da dam povratnu informaciju o svom putovanju.
- US_009: Kao registrovan korisnik želim da ocenim stručnjaka koji mi je pomogao, da bih mogao sa drugim korisnicima da podelim koliko su oni bili fantastični.
- US_010: Kao registrovan korisnik želim da se registrujem za stručnjaka destinacije, da bih mogao da pomognem ljudima koji putuju u moj grad.
- US_011: Kao stručni korisnik želim da odgovorim na pitanja o mom gradu, da bih mogao da osvojim bodove koje ću u budućnosti razmeniti.
- US_012: Kao stručni korisnik želim da razmenim bodove za karte, da bih mogao više da putujem po svetu.
- US_013: Kao administrator želim da upravljam paketima, da bi korisnici mogli da imaju fantastične mogućnosti za putovanje.
- US_014: Kao administrator želim da upravljam registrovanim korisnicima, da bi WWTravelClub mogao da garantuje dobar kvalitet usluge.
- US_015: Kao administrator želim da upravljam stručnim korisnicima, da bi se odgovorilo na sva pitanja koja se odnose na naše destinacije.
- US_016: Kao administrator želim da ponudim više od 1.000 paketa širom sveta, da bi druge države mogle da iskuse usluge WWTravelClub-a.
- US_017: Kao CEO želim da imam više od 1.000 korisnika koji istovremeno pristupaju veb sajtu, da bi posao mogao efikasno da se proširuje.
- US_018: Kao korisnik želim da pristupim WWTravelClub-u na mom maternjem jeziku, da bih bolje razumeo ponuđene pakete.
- US_019: Kao korisnik želim da pristupim WWTravelClub-u u Chrome, Firefox i Edge veb pretraživačima, da bih mogao da upotrebim veb pretraživač koji želim.
- US_020: Kao korisnik želim da znam da su informacije o mojoj kreditnoj kartici bezbedno sačuvane, da bih mogao bezbedno da kupujem pakete.

Imajte na umu da, kad pišete priče, date informacije koje se odnose na nefunkcionalne zahteve, kao što su bezbednost, okruženje, performanse i skalabilnost.

Međutim, neki sistemski zahtevi mogu biti izostavljeni kada pišete korisničke priče, a trebalo bi da budu uključeni u specifikaciju softvera. Ti zahtevi mogu da se odnose na pravne aspekte, hardverske i softverske preduslove, ili čak na upozorenja za pravilnu isporuku sistema. Oni bi trebalo da budu mapirani i navedeni kao korisničke priče. Sistemski zahtevi WWTravelClub-a su predstavljeni sledećom listom. Vidite da su zahtevi napisani u budućem vremenu, jer sistem još ne postoji:

- `SR_001`: Sistem će upotrebiti Microsoft Azure komponente za isporuku potrebne skalabilnosti.
- `SR_002`: Sistem će morati da poštuje **General Data Protection Regulation** (**GDPR**) zahteve.
- `SR_003`: Sistem će se pokretati na Windows, Linux, iOS i Android platformi.
- `SR_004`: Svaka veb stranica ovog sistema će morati da reaguje za najviše 2 sekunde kada istovremeno pristupa 1.000 korisnika.

Ideja postojanja ove liste korisničkih priča i sistemskih zahteva je da vam pomogne da razumete koliko može biti kompleksno razvijanje platforme, ako o njoj razmišljate kao arhitekta softvera.

Rezime

U ovom poglavlju ste naučili svrhu softverskog arhitekte u timu za razvoj softvera. Takođe, u ovom poglavlju smo opisali osnove modela procesa razvoja softvera i proces prikupljanja zahteva. Imali ste mogućnost da naučite da kreirate Azure nalog, koji ćete upotrebiti tokom studije slučaja iz ove knjige, koja je predstavljena u prethodnom odeljku. Učili ste i o funkcionalnim i nefunkcionalnim zahtevima i kako da ih kreirate upotrebom korisničkih priča. Ove tehnike će vam pomoći da isporučite bolji softverski projekat.

U sledećem poglavlju ćete imati mogućnost da saznate koliko su važni funkcionalni i nefunkcionalni zahtevi za softversku arhitekturu.

Pitanja

1. Koje veštine su potrebne arhitekti softvera?
2. Kako Azure može da pomogne softverskom arhitekti?
3. Kako softverski arhitekta odlučuje koji model procesa razvoja softvera je najbolje upotrebiti u projektu?

4. Kako softverski arhitekta doprinosi prikupljanju zahteva?
5. Koje vrste zahteva softverski arhitekta mora da proveri u specifikaciji zahteva?
6. Kako Design Thinking i Design Sprint pomažu softverskom arhitekti u procesu prikupljanja zahteva?
7. Kako korisničke priče pomažu softverskom arhitekti u procesu pisanja zahteva?
8. Koje su dobre tehnike za razvoj softvera sa veoma dobrim performansama?
9. Kako softverski arhitekta proverava da li su korisnički zahtevi pravilno implementirani?

Dodatna literatura

Ovde imate listu nekih knjiga i linkova koje možete dodatno da pročitate, da biste prikupili više informacija o ovom poglavlju.

Za više informacija o platformi Azure, pogledajte sledeće:

- `https://www.packtpub.com/virtualization-and-cloud/hands-azuredevelopers`
- `https://azure.microsoft.com/en-us/overview/what-is-azure/`
- `https://azure.microsoft.com/en-us/services/devops/`
- `https://www.microsoft.com/en-us/hololens`
- `https://azurecharts.com/`

Više informacija o .NET-u 5 nalazi se ovde:

- `https://docs.microsoft.com/en-us/dotnet/`
- `https://docs.microsoft.com/en-us/aspnet/`
- `https://www.packtpub.com/web-development/hands-full-stack-webdevelopment-aspnet-core`
- `https://docs.microsoft.com/en-us/aspnet/core/performance/performancebest-practices`

Linkovi za modele procesa razvoja softvera:

- https://agilemanifesto.org/
- https://www.amazon.com/Software-Engineering-10th-Ian-Sommerville/dp/0133943038
- https://www.amazon.com/Software-Engineering-Practitioners-RogerPressman/dp/0078022126/
- https://scrumguides.org/
- https://www.packtpub.com/application-development/professionalscrummasters-handbook
- https://en.wikipedia.org/wiki/Incremental_build_model
- https://en.wikipedia.org/wiki/Waterfall_model
- http://www.extremeprogramming.org/
- https://www.gv.com/sprint/