

UNIVERZA V LJUBLJANI  
EKONOMSKA FAKULTETA

MAGISTRSKO DELO

**PRIMERJAVA MALOKODNIH IN BREZKODNIH RAZVOJNIH  
OKOLIJ S TRADICIONALNIMI**

## IZJAVA O AVTORSTVU

Podpisan Jernej Rejc, študent Ekonomske fakultete Univerze v Ljubljani, avtor predloženega dela z naslovom Primerjava malokodnih in brezkodnih razvojnih okolij s tradicionalnimi, pripravljenega v sodelovanju s svetovalcem prof. dr. Turkom Tomažem.

### IZJAVLJAM

1. da sem predloženo delo pripravil samostojno;
2. da je tiskana oblika predloženega dela istovetna njegovi elektronski obliki;
3. da je besedilo predloženega dela jezikovno korektno in tehnično pripravljeno v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani, kar pomeni, da sem poskrbel, da so dela in mnenja drugih avtorjev oziroma avtoric, ki jih uporabljam oziroma navajam v besedilu, citirana oziroma povzeta v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani;
4. da se zavedam, da je plagiatstvo – predstavljanje tujih del (v pisni ali grafični obliki) kot mojih lastnih – kaznivo po Kazenskem zakoniku Republike Slovenije;
5. da se zavedam posledic, ki bi jih na osnovi predloženega dela dokazano plagiatstvo lahko predstavljalo za moj status na Ekonomski fakulteti Univerze v Ljubljani v skladu z relevantnim pravilnikom;
6. da sem pridobil vsa potrebna dovoljenja za uporabo podatkov in avtorskih del v predloženem delu in jih v njem jasno označil;
7. da sem pri pripravi predloženega dela ravnal v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobil soglasje etične komisije;
8. da soglašam, da se elektronska oblika predloženega dela uporabi za preverjanje podobnosti vsebine z drugimi deli s programsko opremo za preverjanje podobnosti vsebine, ki je povezana s študijskim informacijskim sistemom članice;
9. da na Univerzo v Ljubljani neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve predloženega dela v elektronski obliki, pravico reproduciranja ter pravico dajanja predloženega dela na voljo javnosti na svetovnem spletu preko Repozitorija Univerze v Ljubljani;
10. da hkrati z objavo predloženega dela dovoljujem objavo svojih osebnih podatkov, ki so navedeni v njem in v tej izjavi.

V Ljubljani, dne 18. 9. 2020

Podpis študenta: \_\_\_\_\_

## KAZALO VSEBINE

|                                                                                        |           |
|----------------------------------------------------------------------------------------|-----------|
| <b>UVOD .....</b>                                                                      | <b>1</b>  |
| <b>1 O RAZVOJU SODOBNIH PROGRAMSKIH REŠITEV .....</b>                                  | <b>4</b>  |
| <b>1.1 Faze življenjskega cikla razvoja programske opreme .....</b>                    | <b>5</b>  |
| <b>Projektni vidiki razvoja programskih rešitev .....</b>                              | <b>6</b>  |
| 1.1.1 Tradicionalne metodologije in modeli .....                                       | 10        |
| 1.1.2 Agilni pristop.....                                                              | 11        |
| 1.1.3 Primerjava agilnih metodologij s tradicionalnimi .....                           | 11        |
| <b>1.2 Razvoj spletnih programskih rešitev .....</b>                                   | <b>12</b> |
| 1.2.1 Osnovni gradniki spletnih strani in lastnosti spletnih programskih rešitev ..... | 12        |
| 1.2.2 Izzivi sodobnih spletnih programskih rešitev .....                               | 13        |
| <b>1.3 Tradicionalna razvojna okolja .....</b>                                         | <b>16</b> |
| <b>1.4 Malokodna in brezkodna okolja .....</b>                                         | <b>17</b> |
| 1.4.1 Malokodna razvoja okolja .....                                                   | 18        |
| 1.4.2 Brezkodna razvoja okolja .....                                                   | 21        |
| <b>2 ZNAČILNOSTI SPLETNIH PROGRAMSKIH REŠITEV .....</b>                                | <b>22</b> |
| <b>2.1 Programski vmesniki in spletne povezovalne tehnologije.....</b>                 | <b>23</b> |
| <b>2.2 Razvijalski vzorci spletnih programskih rešitev.....</b>                        | <b>24</b> |
| 2.2.1 MVC in MVP .....                                                                 | 25        |
| 2.2.2 MVVM in MOVE .....                                                               | 25        |
| <b>2.3 Strukturne raznolikosti spletnih programskih rešitev .....</b>                  | <b>26</b> |
| <b>2.4 Zaledni sistemi .....</b>                                                       | <b>27</b> |
| 2.4.1 Gostovanje in domenski prostor.....                                              | 28        |
| 2.4.2 Monolitna in mikrororitvena arhitektura .....                                    | 30        |
| 2.4.3 Virtualizacija in vsebniki .....                                                 | 31        |
| 2.4.4 Pravna skladnost programskih rešitev .....                                       | 31        |
| 2.4.5 Informacijska in kibernetska varnost .....                                       | 33        |
| <b>3 RAZVOJNA OKOLJA .....</b>                                                         | <b>35</b> |
| <b>3.1 Tradicionalna razvojna okolja .....</b>                                         | <b>36</b> |
| <b>3.2 Malokodna razvojna okolja .....</b>                                             | <b>37</b> |
| <b>3.3 Brezkodna razvojna okolja.....</b>                                              | <b>39</b> |
| <b>3.4 Primerjava razvojnih okolij .....</b>                                           | <b>40</b> |

|            |                                                                |           |
|------------|----------------------------------------------------------------|-----------|
| <b>4</b>   | <b>PRIMERI PROGRAMSKIH REŠITEV .....</b>                       | <b>43</b> |
| <b>4.1</b> | <b>Specifikacije primerov .....</b>                            | <b>44</b> |
| <b>4.2</b> | <b>Razvoj testnih primerov programskih rešitev .....</b>       | <b>46</b> |
| 4.2.1      | Razvoj programskih rešitve v ogrodju Django .....              | 46        |
| 4.2.2      | Razvoj programskih rešitve v razvojnem okolju OutSystems ..... | 49        |
| 4.2.3      | Razvoj programskih rešitve v razvojnem okolju Mendix .....     | 52        |
| 4.2.4      | Razvoj programskih rešitve v razvojnem okolju Bubble .....     | 54        |
| 4.2.5      | Razvoj programskih rešitve v razvojnem okolju Webflow .....    | 58        |
| <b>5</b>   | <b>PRIMERJVA PROGRAMSKIH REŠITEV .....</b>                     | <b>61</b> |
| <b>5.1</b> | <b>Rezultati primerjave.....</b>                               | <b>61</b> |
| <b>5.2</b> | <b>Končne ugotovitve .....</b>                                 | <b>64</b> |
|            | <b>SKLEP.....</b>                                              | <b>67</b> |
|            | <b>LITERATURA IN VIRI .....</b>                                | <b>69</b> |

## KAZALO TABEL

|                                                                                   |    |
|-----------------------------------------------------------------------------------|----|
| Tabela 1: Kriterij ISO 9126 za kakovost programskih rešitev .....                 | 9  |
| Tabela 2: Prednosti in slabosti tradicionalnih razvojnih okolij.....              | 17 |
| Tabela 3: Prednosti in slabosti malokodnih razvojnih okolij .....                 | 20 |
| Tabela 4: Prednosti in slabosti brezškodnih razvojnih okolij .....                | 22 |
| Tabela 5: Primerjava HTTP zahtevkov v povezavi s CRUD na osnovi REST API-ja ..... | 24 |
| Tabela 6: Primernost razvojnih okolij .....                                       | 42 |
| Tabela 7: Primeri razvoja spletnih programskih rešitev.....                       | 62 |
| Tabela 8: Prednosti in slabosti razvojnih okolij .....                            | 65 |

## KAZALO SLIK

|                                                            |    |
|------------------------------------------------------------|----|
| Slika 1: Življenjski cikel programskih rešitev.....        | 5  |
| Slika 2: Zvezdni model projektnega managementa .....       | 8  |
| Slika 3: Tradicionalni IDE – Visual Studio 2019 .....      | 16 |
| Slika 4: Različni razvijalski vzorci.....                  | 24 |
| Slika 5: Monolitna programska rešitev .....                | 30 |
| Slika 6: Mikrostoritvena arhitektura .....                 | 30 |
| Slika 7: Magični kvadrant – malokodna razvojna okolja..... | 36 |
| Slika 8: Zahtevnost v primerjavi s hitrostjo razvoja ..... | 42 |
| Slika 9: Primerjava kompleksnosti.....                     | 43 |

|                                                                                  |    |
|----------------------------------------------------------------------------------|----|
| Slika 10: Registracija in vpis v sistem .....                                    | 44 |
| Slika 11: Objava oglasa – Blog .....                                             | 44 |
| Slika 12: Objava izdelka – Spletna trgovina .....                                | 45 |
| Slika 13: Django enostavna spletna rešitev – ITM .....                           | 47 |
| Slika 14: Funkcija za profil uporabnika .....                                    | 47 |
| Slika 15: Django model objave .....                                              | 48 |
| Slika 16: Django blog.....                                                       | 48 |
| Slika 17: Django model izdelka .....                                             | 49 |
| Slika 18: Django spletna trgovina .....                                          | 49 |
| Slika 19: Delovni tok pri enostavni spletni programski rešitvi – OutSystems..... | 50 |
| Slika 20: OutSystems enostavna spletna rešitev - ITM.....                        | 50 |
| Slika 21: OutSystems delovni tok - objava .....                                  | 50 |
| Slika 22: OutSystems Blog .....                                                  | 51 |
| Slika 23: Proces dodajanja novega produkta.....                                  | 51 |
| Slika 24: OutSystems spletna trgovina.....                                       | 52 |
| Slika 25: Mikrotok nalaganja dokumenta .....                                     | 52 |
| Slika 26: Mendix enostavna spletna programska rešitev - ITM.....                 | 53 |
| Slika 27: Mendix mikrotok – dodajanje objave .....                               | 53 |
| Slika 28: Mendix Blog .....                                                      | 53 |
| Slika 29: Mendix mikrotok – objava produkta.....                                 | 54 |
| Slika 30: Mendix spletna trgovina .....                                          | 54 |
| Slika 31: Bubble enostavna spletna rešitev - ITM .....                           | 55 |
| Slika 32: Bubble dodajanje vsebine .....                                         | 55 |
| Slika 33: Bubble obrazec za registracijo na blog .....                           | 56 |
| Slika 34: Bubble blog.....                                                       | 56 |
| Slika 35: Bubble delovni tokovi.....                                             | 57 |
| Slika 36: Bubble vnos novega produkta.....                                       | 57 |
| Slika 37: Bubble spletna trgovina .....                                          | 58 |
| Slika 38: Webflow obrazec za objavo.....                                         | 58 |
| Slika 39: Webflow blog.....                                                      | 59 |
| Slika 40: Webflow vnos produkta .....                                            | 60 |
| Slika 41: Webflow spletna trgovina .....                                         | 60 |
| Slika 42: Webflow nastavitve za spletno trgovino.....                            | 60 |
| Slika 43: Povezava med integracijo in enostavnostjo .....                        | 62 |
| Slika 44: Varnost in enostavno razvoja.....                                      | 63 |
| Slika 45: Primerjava med učenjem in načrtovanjem.....                            | 64 |

## SEZNAM KRATIC

ang. – angleško

**2FA** – (ang. Two-factor authentication); avtentikacija z uporabo dveh avtentikacijskih faktorjev

**API** – (ang. Application Programming Interface); aplikacijski programski vmesnik

**CAPTCHA** – (ang. Completely Automated Public Turing test to tell Computers and Humans Apart); test za ugotavljanje, da je uporabnik res človek, navadno z oznako "nisem robot"

**CASE** – (ang. Computer-aided Software Engineering); orodja za pomoč pri razvoju

**CMS** – (ang. Content Management System); sistem za upravljanje vsebine

**CRUD** – (ang. Create, Read, Update, Delete); ustvari, preberi, posodobi, odstrani

**CSS** – (ang. Cascading Style Sheets); kaskadne stilske predloge

**DOM** – (ang. Document Object Mode); dokumentno objektni model

**DSL** – (ang. Domain-specific Language); domensko specifični programski jeziki

**GPL** – (ang. General-purpose Language); univerzalni programski jeziki

**HHVM** – (ang. HipHop Virtual Machine); HipHop navidezni stroj

**HTML** – (ang. HyperText Markup Language); označevalni jezik HTML

**IDE** – (ang. Integrated development environments); integrirano razvojno okolje

**IFTTT** – (ang. If This Then That); če to, potem to

**IOT** – (ang. Internet of Things); internet stvari

**IT** – (ang. Information Technology); informacijska tehnologija

**JIT** – (ang. Just in Time); ravno ob pravem času

**KISS** – (ang. Keep It Simple Stupid); ohranjanje enostavnosti

**ORM** – (ang. Object-relational Mapping); objektno-relacijsko mapiranje

**OTP** – (ang. One Time Password); enkratno geslo

**PHP** – (ang. Hypertext Preprocessor); skriptni programski jezik

**SDLC** – (ang. Software Development Life Cycle); življenjski cikel programske opreme

**SEO** – (ang. Search Engine Optimization); optimizacija spletnih strani

**SQL** – (ang. Structured Query Language); strukturirani povpraševalni jezik

**SSL** – (ang. Secure Sockets Layer); sloj varnih vtičnic

**UML** – (ang. Unified Modeling Language); poenoteni jezik modeliranja

**WYSIWYG** – (ang. What You See Is What You Get); kar vidiš to dobiš

## UVOD

Svetovni splet je omogočil povezanost poslovanja na globalni ravni. Zato programske rešitve danes predstavljajo osrednji del poslovanja. Digitalna disrupcija je omogočila, da je spletno poslovanje postalo ključni del za delovanje večine podjetij ter vladnih, nevladnih in ostalih organizacij. Posledično to pomeni povečanje potreb po razvoju in uporabi poslovnih spletnih programskih rešitev. Spletne programske rešitve nam omogočajo, da jih uporablja vsak, ki ima dostop do spleta. Seveda pod pogojem, da dostop do njih ni omejen zaradi varnostnih ali zaupnih razlogov. Njihova posebnost je ta, da so neodvisne od uporabljenega operacijskega sistema in naprav, na katerih jih uporabljamo. Samo razvijanje spletnih programskih rešitev je težaven proces, ki zahteva natančni projektni management. Za razvoj uspešnih spletnih programskih rešitev je potrebno tudi ustrezno sledenje modernim agilnim metodologijam dela, modelom razvoja in pravilnimi izbranimi razvijalskimi vzorci, ki natančno določajo arhitekturo, strukturo in namen uporabe spletnih programskih rešitev.

Razvojna podjetja programske opreme imajo pogosto težave z doseganjem časovnih okvirov razvoja in integracijo novih programskih rešitev, zato so se pojavila razvojna okolja, ki omogočajo razvoj programskih rešitev brez pisanja programske kode. Ta razvojna okolja imenujemo malokodna in brezkodna. Pri malokodnih razvojnih okoljih razvijalci še vedno potrebujejo osnovno znanje programiranja, saj omogočajo razvoj funkcionalnosti s pisanjem kode. To nam omogoča lažje in hitreje prilaganje zahtevnejšim specifikacijam. Brezkodna razvojna okolja nam omogočajo razvoj popolnoma brez kode, zato je težje prilagoditi programsko rešitev, saj je večino funkcionalnosti omejenih na določeno poslovno področje in po navadi ne omogočajo možnosti dodajanja razširitev po meri. Brezkodna razvojna okolja so primerna za podjetja, ki nimajo dovolj časa in financ za razvoj dolgoročnih spletnih programskih rešitev, vendar potrebujejo hitro nadomestilo določenega poslovnega procesa. Pri malokodnih in brezkodnih razvojnih okoljih je programiranje na visokem abstraktnem nivoju in omogoča hiter razvoj programskih rešitev v primerjavi s tradicionalnim razvojem. Vsako razvojno okolje vsebuje domensko specifične jezike, ki s pomočjo vizualizacije delovnega toka opišejo delovanje določene funkcionalnosti. Pazljivi moramo biti predvsem pri izbiri razvojnega okolja za razvoj poslovnih programskih rešitev. V splošnem velja pravilo, da so za bolj splošne probleme uporabnejša malokodna razvojna okolja, ker imajo večjo širino. Za bolj specifične probleme, kot je npr. beleženje prodaje izdelkov, pa uporabimo brezkodna razvojna okolja. Ključno je tudi razmisliti pri izbiri in se vprašati, ali rešitev izdelamo v podjetju ali gremo v nakup. Vse to je povezano s stroški izdelave oziroma nakupa. Smiselna izbira malokodnih ali brezkodnih razvojnih okolij je odvisna od našega poslovnega procesa. Danes je veliko programskih rešitev, ki so že implementirane in ne dosegajo optimalne integracije v nov sistem tako iz tehničnega kot tudi poslovnega vidika. Tega problema ne bomo rešili z izbiro drugega razvojnega okolja, vendar z izbiro pravilne strategije, ustreznim projektnim managementom in pravilnim razvijalskim vzorcem (Henriques, Lourenço, Amaral & Goulão, 2018).

Stroški nakupa programskih rešitev so lahko visoki, zato v prihodnosti lahko pričakujemo trend povečanja lastnega razvoja programskih rešitev. To pomeni, da bodo lahko zaposleni v podjetjih sami našli rešitve s pomočjo brezkodnih ali malokodnih razvojnih okolij, ki ne bodo vedno skladna s poslovanjem. To se danes že dogaja z množično uporabo tretjerazrednih programskih rešitev in oblačnih storitev. To uporabo programskih rešitev opišemo, kot »sive rešitve« (ang. Shadow Information Technology). Uporaba sivih rešitev lahko predstavlja varnostno pomanjkljivost ter tako zavedno ali nezavedno škodujemo podjetju. V najslabšem primeru lahko predstavlja neskladnost poslovnih pravil in pravnih predpisov. »Sive rešitve« označujemo kot del IT-ja, ki je vključen v naš proces na strojnem ali programskem nivoju (Richardson & Rymer, 2016).

Hiter razvoj programskih jezikov in spletnih tehnologij je razkril arhitekturne in druge tehnične omejitve pri razvoju sodobnih spletnih programskih rešitev. Preveliko število knjižnic in ogrodi je prispevalo k še večji razdeljenosti in nejasnim metodam dela za razvoj spletnih programskih rešitev. Problemi so se začeli kazati tudi pri vse večjem kadrovskem pomanjkanju spletnih programerjev (Jesse, 2019). V Evropi do konca leta 2020 lahko pričakujemo več kot milijon prostih delovnih mest za programerje, ki jih trg dela ne bo zmožen zapolniti. Po zadnjih podatkih raziskovalnega podjetja Gartner se bo do leta 2021 povpraševanje po IT storitvah povečalo za petkrat hitreje, kot je zmožnost delovnega trga te potrebe nadomestiti. (Waszkowski, 2019).

Namen magistrske naloge je odgovoriti na vprašanje, v kolikšni meri lahko malokodna in brezkodna razvojna okolja nadomestijo tradicionalni razvoj poslovnih spletnih programskih rešitev in ali je kakovost ter uporabnost tako razvitih poslovnih spletnih programskih rešitev slabša od tistih, ki so razvite s tradicionalnimi razvojnimi okolji.

Cilji, na katere se bom osredotočil so naslednji. Opredelil bom način razvoj sodobnih programskih rešitev v tradicionalnih, malokodnih in brezkodnih razvojnih okoljih in opisal projektne vidike razvoja. Naredil bom pregled modelov, metodologij, razvijalskih vzorcev, strukturne raznolikosti in opisal značilnosti zaledja programskih rešitev. Prav tako bom predstavil in analiziral razvojna okolja tradicionalnih, malokodnih in brezkodnih razvojnih okolij. Nato bom nadaljeval s testnimi primeri in razvil spletne programske rešitve v vseh razvojnih okoljih. Prikazal bom, da je razvoj v določenih primerih pri malokodnih in brezkodnih okoljih lahko bolj učinkovit v primerjavi s tradicionalnim razvojem. Na podlagi izbranih kriterijev bom preveril tudi ustreznost posamezne razvite spletne poslovne programske rešitve, jih med seboj primerjal in ocenil.

V magistrskem delu bom raziskal posamične malokodne in brezkode programske rešitve in jih primerjal po prirejenih ISO 9126 kriterijih s poudarkom na razvojni komponenti. Ti kriteriji predstavljajo vidike funkcionalnosti, zanesljivosti, uporabnosti, učinkovitosti, prenosljivosti in kakovosti vzdrževanja razvoja programskih rešitev. S pomočjo teh bom lahko ugotovil omejitve ter prednosti in slabosti pri razvoju posamezne programske rešitve (Ríos, Ordóñez, Tapia & Moreno, 2016). Vsaka posamezna programska rešitev ali razvojno okolje ima namreč

različno stopnjo odprtosti in s tem tudi različno uporabnost ter namembnost. Za razvoj programskih rešitev je poleg znanja programiranja potrebno tudi veliko poznavanje računalniške arhitekture, spletne komunikacije, značilnosti protokolov in projektnega managementa razvoja programskih rešitev. Zato sem natančno opisal in raziskal sodobne ter uveljavljene razvijalske vzorce, metodologije in glavne značilnosti posameznih razvojnih okolij. V osrednjem delu sem razvil tri različne spletne programske rešitve na osnovi lastnih specifikacij. Za lažjo primerjavo malokodnih in brezkodnih programskih rešitev sem nato razvil tudi spletne programske rešitve v tradicionalni obliki. Nadaljeval sem z opisom in določil kriterije ocenjevanja programskih rešitev. V zaključnem sem s pomočjo teh kriterijev izvedel še primerjalno analizo med posameznimi razvojnimi okolji in lastno razvitimi programskimi rešitvami s pomočjo programa Dexi. Program Dexi omogoča podporo za enostavno večparametrično odločanje. Na koncu sem ocenil primernost in uporabnost razvojnih okolij ter ustreznost vsake posamezne razvite poslovne spletne programske rešitve. Nato sem s pomočjo lastnih ugotovitev določil, na katerih področjih je smiselno uvajanje brezkodnega ali malokodnega razvoja programskih rešitev za poslovanje in kakšne so omejitve ter prednosti posamezne izvedbe razvoja in uporabe v različnih poslovnih primerih.

Magistrsko delo je napisano v petih poglavjih. V prvem poglavju opisujem razvoj sodobnih spletnih programskih rešitev. Začnem s predstavitev življenjskega cikla razvoja programske opreme. Nadaljujem z opisom projektnega managementa, sodobnih in tradicionalnih modelov in metodologij. Nato opišem sodobni agilni pristop za razvoj programskih rešitev in na kratko osnovne značilnosti sodobnih spletnih programskih rešitev. V nadaljevanju predstavim tradicionalna, malokodna in brezkodna razvojna okolja in kakšne so njihove značilnosti. V drugem poglavju se osredotočim na podroben opis značilnosti spletnih programskih rešitev in njihovih lastnosti, kot so: razvijalski vzorci, strukturne raznolikosti, arhitekturne značilnosti, programski vmesniki, spletno povezovalne tehnologije in zaledni sistemi spletnih programskih rešitev. V tretjem poglavju se posvetim izbranim razvojnim okoljem in natančno opišem značilnosti tradicionalnih, malokodnih in brezkodnih razvojnih okolij in jih primerjam med seboj. Za vsako razvojno okolje natančno opišem njihove prednosti in slabosti ter predstavim njihovo uporabnost in kakovost. V četrtem poglavju opišem in razvijem lastne spletne programske rešitve, kjer najprej predstavim in naštejemo specifikacije za razvoj izbranih spletnih programskih rešitev. Nato v naslednjem poglavju na kratko predstavim postopek razvoja posamezne programske rešitve v izbranem razvojnem okolju. Zadnje peto poglavje namenim rezultatom in primerjavi posameznih programskih rešitev. Določim tudi kriterije za primerjavo, ki sem jih izpeljal iz ISO 9126 standarda. Na koncu naredim primerjalno analizo s pomočjo programa Dexi in predstavim posamezne prednosti in slabosti spletnih programskih rešitev ter podam rezultate in ugotovitve.

# 1 O RAZVOJU SODOBNIH PROGRAMSKIH REŠITEV

Razvijanje sodobnih programskih rešitev zahteva precizno načrtovanje. Razvoj sodobnih programskih rešitev iz leto v leto postaja kompleksnejše zaradi vse hitrejšega razvoja tehnologije in nenehnega inoviranja na področju razvoja programskih rešitev. Informatizacija in digitalizacija poslovnih procesov še dodatno povečujeta potrebe po razvoju poslovnih spletnih programskih rešitev. Osnovo za vsak razvoj programskih rešitev predstavljajo programski jeziki. Programski jeziki včasih niso bili tako enostavni za razvijalce, kot so danes. Eden prvih boljših programskih jezikov je bil npr. COBOL, ki je bil razvit prav za uporabo podjetij, vlad in drugih organizacij v poslovnih, upravnih in finančnih sistemih. Razvoj programskih jezikov je sčasoma prispeval k lažjemu in hitrejšemu razvoju programskih rešitev. Programski jeziki so nas pripeljali k sodobnemu razvoju programskih rešitev, ki nam omogočajo uporabo različnih razvojnih okolij (Urma, 2017).

Osrednji del razvoja programskih rešitev temelji na natančnem opredeljevanju in opisu specifikacij in je prvi korak, ki ga je treba zagotoviti za uspešni projektni management razvoja programskih rešitev. Programiranje predstavlja proces, kjer uporabnik razvije programsko rešitev s pomočjo orodij, ki omogočajo razvoj funkcionalnosti. Kodiranje pa predstavlja proces programiranja, kjer uporabnik s pomočjo programske kode napiše programsko rešitev. Programiranje lahko z enostavnimi besedami opišemo kot podajanje zahtevkov računalniku, ki jih ta razume in izvede na ukaz (Davis, 2020, str. 7).

Zaradi vse večjih zahtev in povezanosti spleta so danes potrebe po programskih rešitvah večje kot kadarkoli prej. Posledično je zaradi počasnega razvoja programskih rešitev v tradicionalnih okoljih prišlo do pojava novih ogrojdov in računalniških okolij, ki so hitreje lahko dostavile končni izdelek oziroma storitev na tržišče. Večje hitrosti in modularnosti razvoja novih programskih rešitev še vedno niso ustrezale povpraševanju. Tako so se v zadnjem desetletju pojavila razvojna okolja, ki temeljijo na vizualnem programiranju. Največji problem, ki ga danes predstavlja razvoj je počasna narava kodiranja. Malokodna in brezkodna razvojna okolja rešujejo problem počasnega razvijanja. Primerna so predvsem za poslovne procese, ki so že znani in uveljavljeni ter jih večina podjetij že natančno razume in zna umestiti v svoje poslovanje. To sicer ne pomeni, da malokodna in brezkodna razvojna okolja ne moremo uporabiti za specifične primere našega poslovanja. V prihodnosti lahko pričakujemo, da bodo preprostejša računalniška in razvojna okolja prevzela večinski delež razvoja programskih rešitev. Malokodna in brezkodna razvojna okolja omogočajo, da hitro implementiramo funkcionalnosti, za katere bi v tradicionalnih sistemih kljub ogrojdem porabili veliko več časa. Omogočajo nam visoko modularnost in abstrakcijo procesov. Tako npr. lahko proces registracije in vpisa v programsko rešitev dodamo le v nekaj minutah, pri čemer ne potrebujemo veliko programerskega znanja. Poznati moramo le poslovni proces podjetja, za katerega razvijamo poslovno spletno programsko rešitev. Za razvoj vseh programskih rešitev moramo slediti življenjskemu ciklu razvoja programske opreme.

Življenjski cikel razvoja programske opreme (ang. Software Development Life Cycle) (v nadaljevanju SDLC) je danes standard za razvoj vseh sodobnih programskih rešitev. Vsaka programska rešitev zahteva natančno vodenje v vseh fazah razvoja. Glavni namen razvojnega cikla je načrtovanje, razvoj, zanesljivost in stroškovna učinkovitost. Za razvoj vseh programskih rešitev je potrebno poglobljeno znanje programiranja in pravilna izbira tehnologij. Poznavanje procesa razvoja pripomore k boljšemu in hitrejšemu razvoju programskih rešitev. Tradicionalni SDLC je vključeval 10 manjših faz. Današnja prevzeta in sprejeta oblika SLDC je po navadi strnjena v 6 večjih faz. SDLC vključuje naslednje faze: planiranje, analiza, razvoj in implementacija, testiranje, integracija in vzdrževanje. Slika 1 prikazuje SDLC. Faze si sledijo v smeri urinega kazalca od planiranja do vzdrževanja. Treba je omeniti, da je varnost pogostokrat izpuščena, vendar je ključni del pri razvijanju vseh programskih rešitev in jo moramo vključiti že v fazi planiranja, saj jo kasneje zaradi arhitekturnih in drugih omejitev težje ali skoraj nemogoče vključimo v programsko rešitev. Sedaj bom na kratko opisal vsako fazo SDLC in njihove lastnosti (Khan & Beg, 2013).

*Slika 1: Življenjski cikel programskih rešitev*



*Vir: Lastno delo.*

## 1.1 Faze življenjskega cikla razvoja programske opreme

**Načrtovanje** predstavlja najbolj pomembno fazo pri razvoju programskih rešitev. Pri tej prvi fazi sodelujejo izkušeni programerji, vodje projekta in naročniki. Natančno je treba določiti cilje, izbrati pravilni razvijalski vzorec, postaviti okvirni datum razvoja, določiti ceno projekta in izbrati ustrezen kader za razvoj. Časovni okvir planiranja je odvisen od dolžine in kompleksnosti projekta. Ko zaključimo s to fazo, nadaljujemo s fazo analize.

**Analiza** natančno določi specifikacije, ki jih bo vsebovala nova programska rešitev. V tej fazi je predvsem pomemben poslovni vidik. Programerji se v tej fazi zato posvečajo razumevanju poslovnih procesov, modeliranju, opravljajo razgovore z naročniki in pridobivajo ključno dokumentacijo za razvoj programskih rešitev. Na podlagi pridobljenih poslovnih procesov oblikujemo ponudbeno dokumentacijo, ki po navadi vsebuje celotni časovni plan, arhitekturo programske rešitve, opis specifikacij, razvojno okolje, uporabljene spletne tehnologije, izbrani programski jezik, stroške izdelave in načrt razvoja.

**Implementacija in razvoj** predstavljata osrednji del projekta, če je bila analiza ustrezno opravljena, v tej fazi ni več poslovnih zapletov ali napačno implementiranih funkcionalnosti. Programerji se v tej fazi ukvarjajo s programiranjem v že določenem razvojnem okolju, ki so ga predhodno določili v fazi načrtovanja. Razvojno okolje je lahko usmerjeno v bolj tradicionalen ali modernejši razvoj, npr. v malokodna ali brezkodna razvojna okolja. Poudariti moram, da je v tej fazi potrebno veliko sodelovanja med ponudniki in naročniki, saj le s tem lahko zagotovimo pravilno implantacijo vseh zahtev in funkcionalnosti.

**Testiranje** predstavlja fazo, v kateri preverimo kakovost in pravilno delovanje naše programske rešitve. S pomočjo testov bomo zagotovili naslednje lastnosti: kvaliteto izvirne kode, ustreznost funkcionalnosti, integracijske zmožnosti, učinkovitost in varnost. Vrstni red ni tako pomemben. Pomembno je, da vsaki fazi namenimo dovolj časa in virov, da lahko učinkovito in čim bolj realno ocenimo pravilno delovanje programske rešitve. Naročnik nam na koncu sporoči ustrezno delovanje, ponudnik pa dopolni in popravi programsko rešitev.

**Integracija** predstavlja uvajanje programske rešitve med zaposlene v prenovljen ali star sistem. Glavni namen te faze je prenos na izvirno (produkcijsko) okolje, na katerem bo delovala nova programska rešitev. Prenos v produkcijsko okolje vedno potrdi projektni vodja. Ta faza sicer ne bi smela predstavljati veliko težav, vendar se v praksi pogosto izkaže, da ta faza ni tako enostavna zaradi kompleksnosti pri povezanosti starih in novih sistemov.

**Vzdrževanje** je zadnja faza v razvoju programskih rešitev. Vsak ponudnik razvojne opreme se v pogodbi zaveže za fazo vzdrževanja. V tej fazi skrbimo, da naša programska rešitev deluje nemoteno in brez večjih napak ali izpadov. Prav tako predstavlja popravljanje in odpravljanje napak ter dodajanje manjših novih funkcionalnosti, ki so bile opisane v delovni pogodbi. Ponudnik lahko glede na potrebe naročnika dodaja tudi nove funkcionalnosti (Khan & Beg, 2013).

### **Projektni vidiki razvoja programskih rešitev**

Projektni management razvoja programskih rešitev predstavlja številne izzive. Razvoj računalništva nas je pripeljal tako daleč, da celotne sisteme informatiziramo s pomočjo programskih rešitev. Zaradi vse večjega informatiziranja procesov, se je tudi težavnost vodenja projektov močno povečala. Še večjo kompleksnost povzroča hiter napredek razvoja programskih rešitev in nove metodologije. Standardi in programski jeziki, ki so bolj ali manj

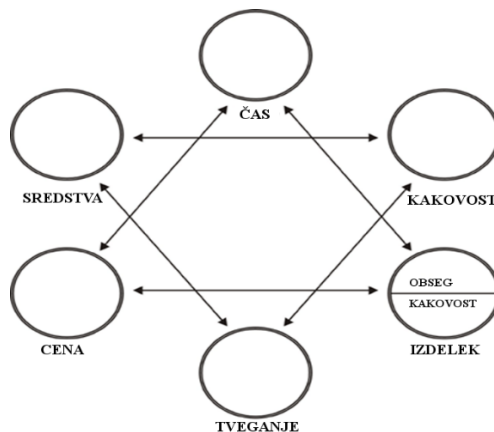
priljubljeni in uporabljeni, se hitro spreminjajo. Razvojna podjetja morajo hitro slediti razvojnim trendom, v nasprotnem primeru jih konkurenčna podjetja zamenjajo. Problem je tudi v optimizaciji programskih rešitev, saj so močno odvisna od strojne opreme. Vsaka programska rešitev ima specifične strojne in programske omejitve in jih moramo upoštevati pri vsakem projektu posebej. Potrebe hitrega prilagajanje in razvoja zahtevajo, da se danes zaradi nepredvidljivih okoliščin in prilagodljivega razvoja s pomočjo agilnih metodologij vse bolj usmerjamo k človeškim in organizacijskim sposobnostim pri razvoju poslovnih programskih rešitev. Za ustrezen razvoj in vodenje je treba ohranjati enotne ekipe za razvoj. O tem govori tudi Brookov zakon. To pomeni, da moramo že na samem začetku pravilno oblikovati delovno skupino, saj menjava ključnih akterjev ali dodajanje ljudi med izvedbo projekta povzroča povečanje časovnega razvoja in ne zmanjšuje, kot smo to načrtovali. V najslabšem primeru z novimi akterji dodamo toliko kompleksnosti, da projekt nikoli ne zapusti razvojne faze (Akbar in drugi, 2017). Značilnost razvoja programskih rešitev je v tem, da večino stroškov nastane pri samem razvoju. Programska oprema nima roka trajanja kot strojna oprema in jo je zato tudi težje nadgrajevati. Programske rešitve imajo eno zelo veliko slabost. Ta je, da imajo lahko že manjše spremembe pri osnovnih funkcionalnostih zelo velike posledice na celotno rešitev. Danes ta problem še poglobljajo različna orodja za hiter razvoj programskih rešitev, ki sicer uporabniku omogočajo hitrejšo izdelavo, a povzročajo preveč negotovosti in nerazumevanja celostne strukture programskih rešitev. Posledično je zato razvoj programskih rešitev in projektni management razvoja programskih rešitev iz leta v leto bolj kompleksno. Poleg sledenja SDLC moramo upoštevati in predhodno izbrati ustrezen model in metodologijo, ki jo bomo uporabili skozi celotni projekt. Te modele in metodologije bomo opisali v naslednjih podpoglavjih 1. 2. 1, 1. 2. 2 in 1. 2. 3 (Khawaja & Alshayeb, 2019).

Najprej se osredotočimo na organizacijski vidik oblikovanja skupin. Za razvoj programske opreme ločimo heterogene in homogene skupine. Homogene skupine so tiste, pri katerih imajo razvijalci podobne interese in znanje. Med posameznimi razvijalci ni večjih razlik pri načinu reševanja problemov in samih postopkih razvoja posamezne funkcionalnosti. Poznamo tudi heterogene skupine. Tukaj je veliko razlik med cilji, načini reševanja in interesi razvijalcev. Splošno je znano, da so heterogene skupine boljše v primeru, ko gre za bolj kompleksne izzive. Homogene skupine so primerne za čim hitrejšo operativnost. Torej, ko je določeno programsko rešitev potrebno hitro uvesti v nov sistem. Pri projektnem managementu moramo biti osredotočeni tudi na stroškovno in časovno komponento in obseg funkcionalnosti. Te lastnosti določajo kvaliteto določene programske opreme (Hussain, Keung, Sohail, Khan & Ilahi, 2019). Razvoj programske opreme in programskih rešitev spodbuja nenehno iterativnost, zato se v sodobnem digitalnem svetu le malo projektov lahko kategorizira za dokončane. To daje projektnim vodjem in naročnikom veliko prilagodljivosti. To pomeni, da se projekti hitro prilagajajo, tudi če naročniki ne ponudijo vseh zahtevanih funkcionalnosti (Ibrahim, 2019).

Sodobni pristop vodenja razvoja programskih rešitev priporoča uporabo šesterokrake zvezde. Uporaba tega načina vodenja vključuje pomembne, vendar pri nekaterih dejavnikih preveč subjektivne lastnosti. Sem spadajo dejavniki, kot so: časovni okvir, sredstva, kakovost, cena,

stopnja tveganja in vrednost produkta v povezavi z obsegom, ki je pomemben za uspeh projekta. Dejavniki so razčlenjeni v dva manjša trikotnika. Prvi vsebuje: obseg, časovni okvir, ceno. Ta predstavlja vhodne in izhodne vrednosti projekta. Drugi vsebuje tveganje, dejavnike kakovosti in sredstva. Ta pa predstavlja procesno dejavnost projekta. Na sliki 2 je prikazan šesterokraki zvezdni model projektnega managementa (Akbar in drugi, 2017).

*Slika 2: Zvezdni model projektnega managementa*



*Vir: Akbar in drugi (2017).*

Edina kritika tega modela je, da je treba te lastnosti stalno posodablјati, saj je upravljanje tipičnega projekta preveč zapleteno in okorno. Pri razvoju moramo biti jasni pri sami opredelitvi optimizacije, razširljivosti, varnosti in tudi ustrezni dokumentaciji kode. Preprosto načelo je KISS (ang. Keep It Simple Stupid), ki poudarja ohranjanje poslovnih procesov in čim bolj enostavno programiranje funkcionalnosti (Assal & Chiasson, 2018). Nenazadnje se pogostokrat dogaja, da se izvirna koda spreminja ali vzdržuje v drugem podjetju, zato so stroški vzdrževanja ob nepravilni oz. neprimerno napisani kodi veliki. Pomembno je tudi, da se odločimo za razvoj ali nakup programskih rešitev. V primeru, da nam je pomembnejše obvladovanje, potem je bolje, da se odločimo za nakup, vendar bodo stroški razvoja in vzdrževanja višji. V nasprotnem primeru se odločimo za razvoj v lastni režiji. Za boljše vodenje programskih projektov moramo slediti naslednjim enostavnim načelom, ki so opisana v nadaljevanju. Vse neprogramersko delo naj opravljajo neprogramerji. To pomeni, da menedžerji skrbijo za usklajevanja zahtev naročnikov in lahko, če znajo pisati ustrezne programske teste, pomagajo pri pisanju testov za določene funkcionalnosti. Vsak programer naj bo specializiran na določeno področje, saj tako omogočimo hitrejši pretok dela (Niazi in drugi, 2016). Vodenje po zgledu je eden izmed vodilnih načinov vodenja pri razvoju programskih rešitev in omogoča motivacijo za celotno razvojno ekipo. To pomeni, da motiviramo zaposlene tako, da jim pokažemo podobne uspešne izdelke in storitve drugih podjetji ali posameznikov. Spreminjanje zahtevkov med samim proces programiranja je neustrezno. Vse zahtevke se mora najprej v celoti zaključiti, šele nato se jih lahko spreminja. Spreminjanje med fazo razvoja kaže znake neustreznega planiranja in vodenja ter nerazumevanje naročnikov ali napačno modeliranih procesov (Abdelaziz, Sedky, Ross & Mostafa, 2019). Modele in razvijalske vzorce moramo zato določiti na samem začetku. Tako bomo lažje dosegli vse zastavljene cilje, saj je načrt dela jase. Pogostokrat se zgodi, da

med delom ugotovimo, da bi lahko uporabljali drugo metodo, ki bi bila bolj primerna, vendar je bolje, da se držimo prvotnega načrta. Podeljevanje zahtevkov glede na znanje posameznega programerja je pomembno. Vsakemu razvijalcu naj pripadajo take naloge, ki jih bo lahko obvladoval in dokončal. Komunikacija mora potekati na osebni ravni, saj s tem bolje spoznamo človeka in njegove osebne karakteristike. Znano je, da spoštovanje in zavedanje vrednot podjetja povečuje produktivnost posameznika. Obvladovanje časa in opravljanje z njim je najbolj pomembna komponenta projektnega managementa za razvoj programske opreme. Najbolj pomembno je, da imamo dnevni red, na katerem so večji zahtevki razdeljeni na manjše zahtevke, iz katerih programiramo posamezne manjše funkcionalnosti, ki nato gradijo celoto. Upoštevanje vseh teh načel pripomore k dobri poslovni praksi in doseganju boljših končnih storitev oz. izdelkov (Akbar in drugi, 2017).

Uspešnost programskih rešitev se lahko ocenjuje po različnih kriterijih. Znano je, da več kot polovica programskih rešitev ne pride do produkcijske faze. Uspešno izpeljanih programskih projektov je le dobrih 20 %. Programske rešitve lahko ocenimo s pomočjo različnih kriterijev, npr. za oceno svojih programskih rešitev sem izbral ISO 9126 standard. ISO je mednarodna organizacija za standardizacijo. Na tabeli 1 so prikazani kriteriji ISO 9126, ki skrbijo za kakovost programskih rešitev. Te so: funkcionalnost, zanesljivost, uporabnost, učinkovitost, vzdržljivost in razširljivost. Vse kategorije vsebujejo podkategorije, s katerimi lahko natančno ocenimo ustreznost razvite programske rešitve (Ríos, Ordóñez, Tapia & Moreno, 2016).

*Tabela 1: Kriterij ISO 9126 za kakovost programskih rešitev*

|                              |                                    |                     |                           |                  |
|------------------------------|------------------------------------|---------------------|---------------------------|------------------|
| <b><i>Funkcionalnost</i></b> | <i>ustreznost</i>                  | <i>natančnost</i>   | interoperabilnost         | varnost          |
| <b><i>Zanesljivost</i></b>   | <i>zrelost</i>                     | <i>obnovljivost</i> | <i>tolerantnost</i>       |                  |
| <b><i>Uporabnost</i></b>     | <i>učljivo</i>                     | <i>razumljivo</i>   | <i>operativno</i>         | <i>privlačno</i> |
| <b><i>Učinkovitost</i></b>   | <i>delovanje v časovnem odboju</i> |                     | <i>obvladovanje virov</i> |                  |
| <b><i>Vzdržljivost</i></b>   | <i>stabilnost</i>                  | <i>analiza</i>      | <i>spremembe</i>          | <i>testi</i>     |
| <b><i>Razširljivost</i></b>  | <i>prilagodljivost</i>             | <i>zamenljivost</i> | <i>integracija</i>        |                  |

*Vir: Ríos, Ordóñez, Tapia & Moreno (2016).*

Razvojna podjetja si prizadevajo, da bi za programiranje izdelkov in storitev presegla trenutno stanje na področju storitveno usmerjenih arhitektur oz. SOA (ang. Service-oriented Architecture). Mnoga podjetja aktivno uporabljajo odprtokodne komponente in zagotavljajo tudi vmesno programsko opremo za zagotovitev integracijske komponente, ki jim omogoča, da v svojo ponudbo izdelkov in storitev vključijo različne priljubljene programske rešitve, kot so na primer alternativni avtentikacijski vtičniki (Google, Facebook in Twitter). Vtičniki omogočajo vzpostavljanje novih načinov ponovne uporabe programske opreme in razvoja na

osnovi komponent in programskih linij izdelkov. Najbolj pomemben trend, ki se trenutno pojavlja, je storitvizacija programskih rešitev. Namesto da podjetja prodajajo kupcem končne izdelke, prodajajo programske storitve. Sodobna podjetja so zato začela spreminjati svoje produkte v storitve, npr. Amazon je postal eden izmed največjih svetovnih ponudnikov oblačnih storitev in omogoča reševanje večine problemov prilagodljivosti programskih rešitev na strojnem in tudi na programskem nivoju. Ta prehod z izdelkov na storitve se nadaljuje in ima pomembne posledice za poslovanje IT podjetij in razvoj programskih rešitev. Da se IT projekti lahko izpeljejo uspešno, je potrebno, da razvoj vseh programskih rešitev sledi določenim metodologijam in modelom, ki jih bomo opisali v naslednjem podpoglavju (Fitzgerald & Klaas-Jan, 2018).

#### 1.1.1 Tradicionalne metodologije in modeli

**Kaskadni model** ima zelo dobro poznane omejitve in je najbolj znan tradicionalni model. Temelji na linearnem projektnem managementu. Metodologija vodenja po linearnem principu je primerna samo za popolnoma znane sisteme oziroma okolja. Včasih je bil zelo primeren za razvoj vseh programskih rešitev, saj je bil le ta bolj statičen. Vse funkcionalnosti so bile namreč postavljene na začetku razvoja in se niso spreminjale. Tudi kompleksnost in zapletenost programskih sistemov je bila manjša. Lastnost tega modela je, da sledi razvojnemu ciklu programskih rešitev, vendar z veliko omejitvami. Značilnost te metodologije je, da sledi vsaki fazi in se premakne na naslednjo šele, ko je trenutna končana. Vračanje v prejšnje faze tako ni mogoče oz. predvideno. Nespremenljivost zahtev v realnem svetu ne obstaja, zato imajo danes podjetja pogosto poseben odločevalski organ, ki skrbi za beleženje sprememb. Dodatna slabost tega modela je, da je programska rešitev izdana v enem koraku in ne vključuje možnosti večjega števila iteracij (ponovitev). To pomeni veliko tveganje za naročnike, saj do konca razvoja ne morejo vedeti, ali bo rešitev ustrezna ali ne (Lemke, 2018).

**Spiralni model** je kombinacija iterativnega in linearnega modela, ki vključuje analizo tveganja. Grafično je prikazan kot spirala s številni zankami. Natančno število zank spirale ni pogojeno, temveč se razlikuje od velikosti projekta. Vsaka zanka predstavlja fazo v SDLC. Natančno število faz potrebnih za razvoj izdelka lahko vodja projekta spreminja glede na projektna tveganja. Vodja projektov dinamično določa število faz in ima pomembno vlogo pri razvoju izdelka z uporabo spiralnega modela. Polmer spirale v določeni točki predstavlja dosedanje stroške projekta. Kotna dimenzija predstavlja dosednji napredek v trenutni fazi. Spiralni model predstavlja nadgradnjo kaskadnega modela in omogoča boljšo fleksibilnost, lažje obvladovanje večjih projektov, uvedbo prototipa in upravljanje tveganja. Slabosti, ki so značilne za spiralni model, so: prevelika kompleksnost, drag pristop, prevelika odvisnost od analize tveganja in težje obvladovanje (Shylesh, 2017).

### 1.1.2 Agilni pristop

Agilni pristop je danes uveljavljen pri razvoju vseh spletnih programskih rešitev, saj omogoča večjo prilagodljivost potrebam naročnika. Prilagodljivost je ključna značilnost in je še bolj pomembna kot predvidljivost, ki je značilna za tradicionalni razvoj. Danes se poslovni procesi hitro spreminjajo in je nemogoče pričakovati celoten projektni načrt že na samem začetku projekta. Agilni pristop se ne usmerja le na procese, temveč poudarja pomembnost neprekinjene komunikacije in sodelovanje med razvojno skupino. Glavne smernice agilnega pristopa so v iterativnem pristopu, ki omogoča sledenje spremembam. Vsaka iteracija je sestavljena iz vseh faz razvoja in omogoča dinamično grajenje končne programske rešitve. Pomembno je tudi, da so iteracije razvoja posamezne funkcionalnosti kratke. Najbolj znani agilni pristopi oz. metodologije, ki se uporabljajo pri sodobnem razvoju spletnih programskih rešitev, so: Kanban, Scrum in devOps. (Špundak, 2014).

### 1.1.3 Primerjava agilnih metodologij s tradicionalnimi

Danes pri razvoju programskih rešitev prevladuje uporaba agilnega pristopa. Agilni pristop je bil ustvarjen z namenom, da ima končna storitev ali izdelek lastnosti, kot so zanesljivost, združljivost, funkcionalnost in varnost, ki jih določa ISO 9126 standard. Glavne razlike, ki jih poznamo med modernim in agilnim pristopom, so naslednje: tradicionalne metodologije uporabljajo življenjski cikel, ki vključuje linearni pristop razvijanja. Vse potrebe so znane na začetku projekta in so dobro dokumentirane. Arhitektura nove programske rešitve je preveč kompleksna in prevelika za trenutne ter predhodne potrebe (Stoica, Mircea & Ghilic-Micu, 2013). Glavni cilji med razvojem so predvidljivost in optimizacija. Spremembe med in po razvoju niso zaželeno. Organizacijska struktura je določena že vnaprej. Naročnik redko sodeluje in se ne vtika v sam proces programiranja, zato se tudi velikokrat zgodi, da so poslovni procesi nepravilno implementirani. Organizacijska struktura je vzpostavljena po hierarhiji kontrole in nadzora. Pristop je univerzalni s predvidljivimi in varnimi rešitvami. Uspešnost je v celoti merjenja v predhodnem razvojnem načrtu. Agilni pristop vključuje iterativne in evolucijske modele. Pristop programiranja je prilagodljiv. Potrebe se prilagajajo naročnikom in se lahko sproti spreminjajo. Arhitektura temelji na principu preprostosti in implementacije ključnih procesov. Vodenje je usmerjeno v ljudi in sodelovanje med njimi. Posamezni zahtevki in cilji so med samim razvojem prilagodljivi in razširljivi. Spremembe so natančno preučene in dobrodošle, a ne med kodiranjem določene funkcionalnosti. Ekipe se lahko prilagajajo potrebam. Naročniki sodelujejo s programerji skozi celoten proces razvoja. Organizacijska struktura temelji na sodelovanju in enakovrednosti. Prednosti agilnih metodologij predstavlja fleksibilnost in projekte, narejene po meri, za potrebe posameznega naročnika. Zaradi nenehnega spreminjanja tehnologij je edina možnost stalno inoviranje in prilagajanje. Tradicionalne metodologije in modeli že več let niso aktualni in primerni za razvoj sodobnih programskih rešitev (Ríos, Ordóñez, Tapia & Moreno, 2016).

## 1.2 Razvoj spletnih programskih rešitev

Splet že dolgo ni več samo vir informacij. Zaradi vse večje dostopnosti in povečanja elektronskega poslovanja je razvoj spletnih programskih rešitev postal ključen del pri zagotavljanju uporabnosti spleta. Prednost spletnih programskih rešitev se kaže predvsem v neodvisnosti od systemskega okolja na odjemalčevi strani. To pomeni, da lahko uporabnik uporablja poljubne brskalnike, ki omogočajo, da programska rešitev deluje enako v vseh brskalnikih. Razvijanje vseh programskih rešitev zahteva natančno opredelitev in sledenje agilnim metodologijam, razvojnim vzorcem in modelom sodobnega razvoja programskih rešitev. Spletne tehnologije omogočajo širok izbor različnih razvojnih okolij in jezikov. V splošnem se razvoj spletnih programskih rešitev deli na dva dela. To sta čelni (ang. Front End) in zaledni del (ang. Back End). Čelni del zajema odjemalca in uporabniške interakcije s spletno programsko rešitvijo. Odjemalec je aplikacija, ki omogoča dostop do spletnih programskih rešitev in omogoča prikaz spletnih strani. HTML označevalni jezik (ang. HyperText Markup Language) predstavlja vsebino in strukturo spletne strani. CSS (ang. Cascading Style Sheets) določa, kako naj bo ta vsebina vizualno prikazana. JavaScript (v nadaljevanju JS) je skriptni jezik, ki omogoča dodatno funkcionalnost čelnega dela spletnih strani. Celotno programsko kodo uporabniškega vmesnika razčleni spletni brskalnik. Za razliko od kode na strani strežnika lahko kodo na strani odjemalca vidi in spremeni tudi uporabnik. Zaledni del predstavlja strežnik, na katerem se obdelujejo vsi zahtevki HTTP (ang. Hyper Text Transfer Protocol). HTTP je aplikacijsko komunikacijski protokol, ki skrbi za prenos informacij na spletu. Brez tega protokola odjemalci ne bi morali prikazovati spletnih strani uporabnikom. Zaledje vsebuje tudi podatkovne baze, ki skrbijo za shranjevanje podatkov in programske vmesnike API. Zaledne sisteme bom bolj podrobno opisal v poglavju 2.4. V zadnjih letih spletno programiranje postaja vse bolj pogosto kot odgovor na potrebe po novih poslovnih spletnih programskih rešitvah. Vse več podjetij želi svoje procese informatizirati na tak način, da so čim bolj povezani, vidni in dostopni vsem uporabnikom. Pester nabor spletnih tehnologij inženirjem programske opreme omogoča, da hitro razvijajo nove spletne programske rešitve. Nove tehnologije omogočajo uporabo različnih spletnih ogrodij, ki razvijalcu omogočajo hitrejši in prijaznejši razvoj programskih rešitev za spletne in mobilne naprave. Sodobne spletne programske rešitve sledijo razvijalskemu vzorcu MVC (ang. Model-View-Controller). Na kratko to opišemo kot Model-Pogled-Krmilnik. Razvijalski vzorci tako skrbijo, da je spletna arhitektura programskih rešitev ustrezna za njihovo primarno nalogo. Bolj natančno bomo opisali MVC in druge značilne različne razvijalske vzorce za spletne programske rešitve v poglavju 2.2 (Delcev & Draskovic, 2018).

### 1.2.1 Osnovni gradniki spletnih strani in lastnosti spletnih programskih rešitev

Gradnike spletnih strani razdelimo v tri kategorije: aplikacijska, odjemalčev čelni del in strežniški gradniki. Aplikacijski nivo zajema nadzorne plosče, kjer lahko upravljalec spletne strani upravlja svojo spletno stran. Vključuje tudi različne nastavitve, statistiko in dnevnike dejavnosti. To so gradniki, ki ne predstavljajo direktne povezave s samo arhitekturo spletnih

programskih rešitev. Predstavljajo povezavo med čelnim in zalednim delom. Za razvoj spletnih programskih rešitev danes šteje grafični vmesnik za uporabnika, ki predstavlja odjemalčev del. S tem povezujemo tudi pojem uporabniške izkušnje (ang. User Experience) ali s kratico UX. Uporabniška izkušnja omogoča kvalitetnejšo uporabo spletnega portala ali spletne programske rešitve. Uporabniško izkušnjo (UX) se pogostokrat zamenjuje z uporabnostjo in oblikovanjem uporabniškega vmesnika (UI). Uporabniška izkušnja predstavlja nad množico, ki vključuje vse pomembne elemente za oblikovanje, integracijo, predstavitev blagovne znamke, dizajn, funkcionalnosti in uporabnost (Hartson & Pyla, 2018). Vse spletne strani opisujemo z označevalnim jezikom HTML. HTML je osnova za vse spletne programske rešitve. Zaradi vse večje povezanosti so spletne strani kmalu prešle iz statičnih oblik v dinamične. Potrebe po še bolj dinamičnem in hitrem razvoju so privedle do razvoja HTML5, ki je izboljšal veliko funkcionalnosti in naredil označevalni jezik še uporabnejši in tako izničil uporabo različnih bogatih spletnih programskih rešitev. Eden najbolj znanih je Adobe Flash Player. Sodobna spletna stran običajno vsebuje generirano HTML5 kodo, ki vsebuje JS, ki skrbi za funkcionalnosti čelnega dela (ang. Front End) in CSS za lepši izgled. DOM predstavlja standard objektnega modela in definira vse HTML elemente kot objekte. DOM opiše tudi vse lastnosti, metode in dogodke. V bolj dovršenih spletnih programskih rešitvah se HTML elementi ne berejo iz DOM-a (ang. Document Object Model), vendar so generirani iz kode zalednega sistema. Po priporočilih konzorcija svetovnega spleta (W3C) podatki nikoli ne smejo biti shranjeni v naključnih objektih ali v DOM-u (Bernaschina, Comai & Fraternali, 2017).

Zaledni sistemi vključujejo različne jezike za programiranje zalednih sistemov. Sicer sodobne spletne tehnologije omogočajo, da lahko tako čelni kot zaledni del programiramo v enakem programskem jeziku. Najbolj uporabljena spletna tehnologija je trenutno Node.js. Node.js je odprtokodno okolje za izvajanje programskih rešitev v JS-u, ki delujejo na različnih operacijskih sistemih. Jeziki za zaledne sisteme in njihova razvoja okolja, ki jih lahko uporabimo za razvoj spletnih programskih rešitev so tako na primer: C# (.NET), Java (Spring, Maven), PHP (Laravel, CakePHP, Symfony), Python (Django, Flask), Ruby (Ruby on Rails) in drugi. Vsa poslovna logika spletnih programskih rešitev se nahaja tukaj. Zaledni sistemi vsebujejo tudi podatkovne baze, ki skrbijo za shranjevanje vseh podatkov. Za dostopanje do podatkov uporabljamo program, ki ga imenujemo sistem za upravljanje podatkov baz ali krajše SUPB. Podrobneje bom te sisteme opisali v podpoglavju 2.4.

### 1.2.2 Izzivi sodobnih spletnih programskih rešitev

Sodobni razvoj spletnih programskih rešitev predstavlja mnogo izzivov. Prvi izziv, ki se pogosto pojavlja in je bolj tehnične narave je pravilna izbira spletnih tehnologij. Izbira pravilnega razvojnega okolja in tehnologije temelji predvsem na preprostosti uporabe in programerskih sposobnostnih in priljubljenosti določene tehnologije. Drugi izziv, ki se pojavlja, je organizacijske narave in predstavlja težavnost projektnega managementa in sledenja različnim verzijam programske opreme. Verzioniranje programske opreme pomeni shranjevanje posamezne posodobitve ali nadgraditve programskih rešitev. Danes ti dve težavi

lahko rešujemo na več načinov. Prvo se pogosto rešuje tako, da bolj izkušeni programerji določijo razvojna okolja in programski jezik, ki je uveljavljen. Drugi izziv rešujejo sodobni programi za sledenje napredka in shranjevanje verzij programskih rešitev. GitHub je eden izmed priljubljenih spletnih portalov, ki skrbi za pravilno verzioniranje programskih rešitev. Sodobne spletne programske rešitve imajo tendenco velikega posodabljanja. To je predvsem zaradi zagotavljanja ustrezne varnosti in hitrosti napredka (Chitra, 2018). Amazon danes v svojem računalniškem okolju izvede več kot 50 milijonov posodobitev letno (Jesse, 2019). Posodobitve danes različno semantično opredeljujemo. Danes je posodobitev vse več, saj je programsko opremo potrebno stalno posodablјati. Razlikujemo med tremi različnimi vrstami posodobitev. Večja posodobitev pomeni spremembo v celotnem API-ju. To pomeni, da starejša tehnologija ni obratno kompatibilna z novejšo. API predstavlja seznam pravil, ki omogočajo komunikacijo med dvema programskima rešitvama. Manjša sprememba pomeni popravke, ki so obratno kompatibilni s prejšnjimi sistemi in vsemi funkcionalnostmi. Programski popravek (ang. Patch) je manjša posodobitev funkcionalnosti, odpravljanje napak, posodobitev varnosti in podobno. To pa je pomembno, saj moramo natančno oblikovati naše spletne programske rešitve glede na posodobitve programskih jezikov in drugih soodvisnih sistemov (Wakil & Jawawi, 2018). Naslednji izziv, ki ga moramo razjasniti, je razlika med spletno stranjo in spletno programsko rešitvijo. Kdaj spletna stran postane spletna programska rešitev, ni točno definirano, vendar lahko določimo neke lastnosti, ki spletno stran ločujejo od spletne programske rešitve. Spletna stran vsebuje veliko statičnih elementov in je pogostokrat javno dostopna. Spletna programska rešitev pa omogoča več interakcij z uporabniki. Spletne programske rešitve za uporabo sistema zahtevajo avtentikacijo oz. vpis v sistem. Funkcionalnosti za spletne programske rešitve so bolj kompleksne in zahtevne. Spletna programska rešitev je le del spletne strani in ne predstavlja celote. Večino spletnih programskih rešitev je predhodno sestavljenih, medtem ko spletne strani ne potrebujejo tega koraka. Predhodno sestavljanje omogoča, da so procesi optimizirani in je dostopanje do podatkovnih baz ter ostalih storitev za povečano operativnost ustrezno indeksirano. Povečanje potreb po spletnih programskih rešitvah zahtevajo hitrejši razvoj. Za hitrejši razvoj programskih rešitev danes uporabljamo različna ogrodja in vtičnike, ki nam pohitrijo delo (Hussain, Keung, Sohail, Khan & Ilahi, 2019). Ogrodja so razvojna okolja, ki vključujejo veliko vtičnikov in prilagojenih metode v določenem programskem jeziku za hitrejši razvoj novih funkcionalnosti. Primarni namen ogrodij je omogočiti programiranje spletne programske rešitve, ne da bi pri tem uporabili zunanje knjižnice ali druge vtičnike, ki bi povečale odvisnost in kompleksnost sistema. Primer enega izmed boljših CSS ogrodij je Bootstrap. Ta omogoča hiter razvoj čelnega dela spletnih programskih rešitev, kjer ima prednost mobilni izgled. Mobilne naprave so bile kot katalizator, ki so pospešile razvoj dinamičnih in odzivnih spletnih strani in programskih rešitev. Oblikovanje spletnih strani danes nujno potrebuje odzivno oblikovanje. To pomeni, da se morajo sodobne spletne strani prilagajati uporabniški napravi in tako omogočati, da ima ena spletna stran več različnih oblikovanj za tablični, računalniški in telefonski pogled (Davis, 2020, str. 132).

Eden izmed inženirskih izzivov, ki ga bomo deležni v prihodnosti, je, kako pisati kodo, da bo le ta energetsko učinkovita. Trenutni podatki kažejo, da se razvoj programskih rešitev usmerja v optimizacijo procesov glede na potrebe posameznih podjetij, vendar le redka podjetja optimizirajo kodo tako, da je energetsko učinkovita. Kodo in svojo programsko rešitev lahko optimiziramo tako, da npr. razvijalci uporabljajo sodobne optimizacije procesorjev za dinamično spreminjanje frekvence delovanja, s čimer zmanjšamo porabo energije. Optimizacije potekajo tudi na programskem nivoju, npr. s pisanjem boljših algoritmov, indeksiranje podatkovnih baz in podobno. Različne tehnike ohranjanja energetske učinkovitosti nudijo skoraj vsi ponudnik oblčnih storitev, saj poznajo vplive na okolje in porabo energije ter se zavedajo pomena varovanja okolja (Pinto & Castor, 2017).

Izziv sodobnih poslovnih spletnih programskih rešitev je tudi pravna skladnost. Sodobne spletne programske rešitve morajo upoštevati tudi pravna načela. Dolgo let se na področju spleta ni veliko zakonsko urejalo, zato je Splošna uredba o varstvu osebnih podatkov (SUVP) prinesla veliko sprememb. O SUVP bom nadaljeval v podpoglavju 2.4.4. Pravilno in ustrezno upoštevanje vseh poslovnih in zakonskih predpisov je včasih skoraj nemogoče upoštevati. To pa predstavlja velik izziv in stroške za podjetja (Linden, Khandelwal, Harkous & Fawaz, 2020).

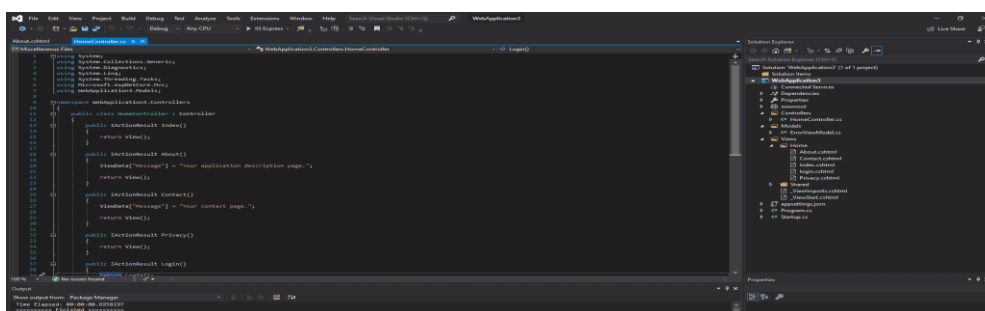
Pri razvoju programskih rešitev se pogosto pojavijo težave z uporabnostjo, razširljivostjo ali varnostjo. Te se po navadi pojavijo šele po tem, ko je celotna rešitev že razvita. Razširljivost in spremenljivost programskih rešitev mora biti zato dobro premišljena. Tako razširljivost kot tudi varnost morata biti v fazi načrtovanja jasno upoštevani. V primeru, da na koncu projekta drastično spreminjamo programsko rešitev, bomo skoraj zagotovo končali z neustrezno poslovno rešitvijo. Dobre smernice razvoja so, da bomo lahko brez večjih težav dodajali in integrirali v novo okolje ter omogočili čim večjo obratno kompatibilnost. Obratna kompatibilnost omogoči novim tehnologijam uporabo starih funkcionalnosti, vendar moramo biti pri tem pazljivi, saj to močno poveča kompleksnost programske rešitve. Težavo predstavlja predvsem združitev vseh starih sistemov s sodobno tehnologijo. Podedovani sistemi (ang. Legacy systems) so več kot le centralni računalniki s staro kodo in metodologijo kaskadnih modelov, ampak vključujejo tudi ljudi, ki so bili usposobljeni in so odgovorni za te sisteme. Primer slabe začetne implementacije je lahko družabno omrežje Facebook. Facebook je bil v celoti napisan v PHP-ju (ang. Hypertext Preprocessor), kar se je kasneje izkazalo za neustrezno. Treba je bilo predelati PHP za njihove zmogljivosti. Predelava se je zgodila v odprtokodni JIT (ang. Just in time) prevajalnik HHVM (ang. HipHop Virtual Machine). JIT pa zato, ker je bil PHP prepočasen za vse zahteve, ki so jih potrebovali v realnem času. Izbira pravilnega razvojnega okolja in programskega jezika je zato ključnega pomena za inženirja programskih rešitev, saj v nasprotnem primeru lahko pride do delnega neizpolnjevanja zahtev naročnikov in v skrajnem primeru zapad projekta (Ottoni, 2018).

### 1.3 Tradicionalna razvojna okolja

Danes je zelo težko izbrati ustrezno razvojno okolje, ki bi bilo primerno za vse različne programske rešitve. Podjetja za razvoj programske opreme imajo že izbrana razvojna okolja, ki jim najbolj ustrezajo glede na zahteve njihovih naročnikov. Po navadi imajo že nastavljene predloge za hitrejšo delo. Podjetja v času svojega delovanja največkrat ne menjajo programskih jezikov ali svojih razvojnih okolij. Vsak programski jezik in s tem povezano razvojno okolje ima namreč svoje prednosti in slabosti. Nekateri jeziki so bolj kompleksni in težje učljivi kot drugi. Značilno je, da je bil vsak programski jezik razvit z namenom reševanja določenega poslovnega ali znanstvenega problema (Popat & Starkey, 2019).

Med tradicionalna integrirana razvoja okolja štejemo npr. Visual Studio 2019. Visual Studio 2019 je ogromno orodje, s katerim lahko razvijemo skoraj vse programske rešitve. Omogoča namreč široko uporabo različnih programskih jezikov. Tradicionalna razvojna orodja podpirajo vse visoko nivojske jezike četrte generacije. Razvojna okolja nam po navadi omogočajo že veliko pred nastavljenih predlog, ki jih lahko uporabimo za hitrejši razvoj programskih rešitev. Pogostokrat pa se nam tudi ponudi izbira o sami arhitekturi programske rešitve. Slika 3 prikazuje običajni izgled sodobno integriranega razvojnega okolja. Sodobna razvojna okolja programerjem olajšajo delo pri samem programiranju, saj ponujajo različne funkcionalnosti. Te dodatne funkcionalnosti so na primer: obarvanje kode za lažje prepoznavanje posameznih objektov in funkcij, predlogi za izboljšavo, imenska usklajenost in skladnost s sintaktični pravili pisanja, predloge za hitro vzpostavitev projektov in mnogo drugih. Prednost tradicionalni razvojnih okolij je predvsem njihova prilagodljivost in fleksibilnost. Prosta je tudi izbira glede vseh zalednih sistemov, zato nikoli nismo ujeti v neko razvojno okolje, ki bi nam določalo izbor orodij, ponudnikov gostovanj in podobno. Prednost se kaže tudi v tem, da lahko programiramo rešitve po meri. Rešitve po meri zahtevajo daljši časovni okvir, vendar jih lahko bolj prilagajamo in so bolj razširljive ter varne za uporabnika. Pomembna sta tudi vidika migracije in integracije. Migracija in integracija predstavljata težaven proces prenosa nove programske rešitve v produkcijo okolje, vendar omogočata veliko možnosti in izbire, saj nismo odvisni od enega samega sistema. Rešitve po meri se dostikrat izkažejo za bolj primerne kot rešitve, ki smo jih naredili v sistemih za hitro oblikovanje programskih rešitev (Choma, Teixeira, Guerra & Da Silva, 2016).

*Slika 3: Tradicionalni IDE – Visual Studio 2019*



*Vir: Lastno delo.*

Tradicionalna razvojna okolja in razvoj programskih rešitev se v zadnjem času začenjajo označevati kot »počasni IT« (ang. Slow IT). To označevanje se je začelo, zaradi pojava različnih razvojnih okolij, ki omogočajo hitrejši razvoj programskih rešitev. Tradicionalna razvojna okolja omogočajo razvoj programskih rešitev na podoben način kot na začetku pojava sodobnih programskih jezikov četrte generacije. Novosti, ki jih prinašajo integrirana razvojna okolja IDE (ang. Integrated Development Environments), omogočajo veliko prednosti in slabosti. Glavna prednost je lažje odpravljanje napak in pestrost izbire orodij za testiranje. Glavna slabost je zagotovo visoka kompleksnost in občasno tudi dolga krivulja učenja. Vse lastnosti tradicionalno razvojnih okolij so našteje v spodnji tabeli 2. Programiranje programskih rešitev zahteva veliko časa in denarnih sredstev. Zato so se pojavila razvojna okolja, ki omogočajo razvoj programskih rešitev brez kodiranja. V naslednjem podpoglavju bom opisal značilnosti malokodnih in brez kodnih razvojnih okolij ter njihovo uporabnost pri razvoju sodobnih programskih rešitev (Jesse, 2019).

*Tabela 2: Prednosti in slabosti tradicionalnih razvojnih okolij*

| <b>Prednosti</b>                                                   | <b>Slabosti</b>                                                       |
|--------------------------------------------------------------------|-----------------------------------------------------------------------|
| Povečana delovna učinkovitost                                      | Dolgotrajno učenje optimalne uporabe                                  |
| Lažje sodelovanje in ustrezni standardi, nadzorovanje virov        | Prevelika kompleksnost za začetnike                                   |
| Orodja za hitrejšo delo (testiranje, predlogi, odpravljanje napak) | Efektivnost kode, razvijalski vzorci in dizajn niso odvisni od okolja |

*Vir: Jesse (2019).*

#### **1.4 Malokodna in brez kodna okolja**

Potrebe po hitrejšem razvoju programskih rešitev so programerje in podjetja pripeljala do razvoja malokodnih in brez kodnih razvojnih okolij. Malokodna in brez kodna razvojna okolja so sodobna okolja za razvoj programskih rešitev. Razvoj programskih rešitev v teh okoljih je izpeljan iz četrte generacije razvijanja programskih rešitev s pomočjo hitrega razvoja programskih rešitev (ang. Rapid Application Development) oz. RAD. Razvojna okolja omogočajo hitrejši razvoj programskih rešitev in skrajšujejo čas do končne storitve oz. uporabe spletnih storitev. Logika v vseh teh razvojnih okoljih je narejena po delovnem toku, ki je podoben notaciji BPMN (ang. Business Process Model Notation) in deluje po principu IFTTT (ang. If This Then That). Vsa razvojna okolja uporabljajo lastno razvite oz. prilagojene domenske specifične programske jezike. Razvoj logike delno tudi spominja na UML (ang.

Unified Modeling Language) diagrame poteka, kjer enostavno opišemo poslovni proces s pomočjo vizualnega delovnega toka. Diagram poteka opiše posamezne korake določenega poslovnega procesa. Razvoj logike v malokodnih in brezkodnih razvojnih okoljih poteka tako, da enostavno s pomočjo vizualnega programiranja modeliramo proces in dodajamo grafične simbole, s katerim opišemo določen proces (Ploder, Bernsteiner, Schlögl & Gschliesser, 2019). Vsako razvojno okolje ima drugačno zasnovane uporabniške vmesnike za modeliranje in ustvarjanje procesa. Bistvo pri vseh je to, da namesto pisanja zahtevne kode lahko programiramo zelo hitro in se ne ukvarjamo toliko z logiko, temveč lahko svoj razvoj posvetimo oblikovanju. Večina splošnih procesov je že sprogramiranih in jih lahko enostavno le vstavimo v naš proces kot že narejeno funkcionalnost. Najbolj pogost proces je avtorizacija in avtentikacija uporabnika ob vpisu, ki je v večini sodobnih razvojnih okolij že implantirana. Prednost pri brezkodnih in malokodnih razvojnih okoljih je ta, da to naredimo z enim ali le nekaj kliki. Najbolj uporabni so različni vtičnik, saj nam omogočijo hitro dodajanje funkcionalnosti. Vtičnik je manjša programska rešitev, ki rešuje določen problem. Na primer lahko obstaja vtičnik, ki nam poenostavi oblikovanje spletne košarice za nakupovanje. Glavni cilj brezkodnih in malokodnih razvojnih okolij je podjetjem omogočiti razvoj programskih rešitev brez zapletenega inženiringa ter jim olajšati njihovo konfiguracijo, hitrost izdelave, omogočiti enostavno razumevanje, zmanjšati časovni okvir in ceno izdelave programske rešitve (Hyun, 2019).

#### 1.4.1 Malokodna razvoja okolja

Malokodna razvojna okolja vsebujejo orodja za oblikovanje dovršenih spletnih poslovnih programskih rešitev tako za programerje kot tudi za ne programerje. Izraz »malokodno« razvojno okolje je prvič uvedel Forrester Research leta 2014. Razvojno okolje nam ponuja razvoj programskih rešitev, ki omogočajo hitro grajenje funkcionalnosti. Prav tako nam omogoča namestitve in konfiguracijo spletnih nastavitev in usposabljanje ljudi ter integracijo v nove sisteme. Malokodna razvojna okolja temeljijo na naprednem uporabniškem vmesniku in oblikovanju programskih rešitev z vizualnim programiranjem. Po ocenah strokovnjakov imajo malokodna razvojna okolja prednosti predvsem na področju varnosti, saj so vsi procesi in funkcionalnosti že predhodno ustrezno skladni z vsemi varnostnimi načeli. Pozorni moramo biti le na to, da jih redno posodabljammo in tako omogočimo, da ustrezajo zadnjim standardom varnosti. Hitrost razvoja je ena izmed glavnih prednosti sodobnih razvojnih okolij, ki jo malokodna razvojna okolja upoštevajo. Pomembno je predvsem to, da se razvijalcem ni treba ukvarjati s programsko logiko, različnimi programskimi knjižnicami in drugimi odvisnostmi, ki jih prinese razvoj. Po podatki iz ankete Forrester je razvoj programskih rešitev z malokodnimi orodji od 5- do 10-krat hitrejši od običajnega. To pomeni tudi zmanjševanje stroškov pri razvoju in vzdrževanju. Zmanjšana kompleksnost pripomore, da je razvoj bolj učinkovit in se razvijalec lahko posveti oblikovanju in prilagajanju potreb uporabnikov. Enostavnost vzdrževanja predstavlja eno izmed glavnih faz razvoja programske opreme in je ključnega pomena, da lahko hitro spremenimo že razvito programsko rešitev, kar zagotavlja trajno usklajevanje med storitvijo, ki jo ponuja programska rešitev in vsemi poslovnimi zahtevami

(Fryling, 2019). Bistvo malokodnih razvojnih okolij je ravno to, da je malo kode. Malo kode pomeni lažje vzdrževanje in hitrejša in lažja popravka na obstoječih procesih. Vsa razvojna okolja so oblikovana tako, da so prijazna razvijalcem tudi brez večjega znanja o programiranju programskih rešitev. Pomembno je tudi poznavanje poslovnih procesov in ravno zato so taka razvojna okolja primerna za poslovne uporabnike. Zagotavljajo zmanjševanje neskladnih, nestabilnih in varnostno pomanjkljivih zahtev. Uporaba malokodnih razvojnih okolij pomeni, da razvijalci hitro sestavijo ustrezne izdelke in potrdijo ideje in zahteve kupcev, še preden zapravijo denarna sredstva za programske rešitve ali funkcije, ki jih stranke morda ne potrebujejo ali so napačno implementirane. Dober primer uporabe malokodnih razvojnih okolij je, da jih lahko uporabimo za hiter prototip naše programske rešitve (Sanchis, García-Perales, Fraile & Poler, 2020).

Pri razvoju naletimo tudi na nekaj slabih lastnosti razvoja programskih rešitev v malokodnih razvojnih okoljih. Razvoj manjših programskih rešitev je v celoti pokrit. Razvoj večjih programskih rešitev ni v celoti podprt, saj ne omogoča ustrezne optimizacije in predstavlja težjo integracijo s tradicionalnimi sistemi. Vsako razvojno okolje ima specifične postopke programiranja in različne paradigme, ki so včasih težje razumljive. Nivo abstrakcije včasih ne omogoča enostavnega programiranja logike. Razvijalci še vedno potrebujejo osnovno znanje programiranja za razvoj (Tisi in drugi, 2019).

Vizualno programiranje (VPL) močno olajša programiranje celotne logike razvoja programskih rešitev. Omogoča prostorsko razporeditev besedilnih in grafičnih simbolov, medtem ko se lahko osredotočimo na oblikovanje in razvoj. Večji poudarek dajejo na bazi podatkov, poslovnih procesih in uporabniških vmesnikih. Malokodna razvojna okolja so običajno dostopna brezplačno ali poceni freemium modelih. Freemium pomeni, da je osnovna storitev zastonj, za dodatne funkcionalnosti pa je treba plačati. Večina malokodnih razvojnih okolij je na voljo kot javna storitev v oblaku.

Razvoj čelnega dela je enostaven in intuitiven in poteka v urejevalniku WYSIWYG – kar vidiš to dobiš (ang. What You See Is What You Get). Na podlagi spletnih gradnikov oz. komponent, ki nam jih ponuja urejevalnik sestavimo uporabniški vmesnik s preprostim prenosom iz palete na spletno strani, kjer so že vnaprej definirani elementi za: slike, gumbe, tabele, video posnetke in druge HTML5 elemente. S pomočjo orodij hitro ustvarimo vse gradnike spletnih strani, ki temeljijo na spletnih standardih kot so HTML5, CSS3 in JS (Ratha, Sahu & Meher, 2018). Razvijalci lahko med programiranjem testirajo in spreminjajo vizualni izgled kode in jo povežejo z zalednimi sistemi ter servisnimi funkcijami v zgolj parih minutah. Zaledni sistemi nam prav tako omogočajo enostavno gradnjo poslovnih procesov s pomočjo diagramov in predstavljajo različne delovne tokove. Poimenovanje se v različnih okoljih razlikuje, vendar gre za isto funkcionalnost. Grafični uporabniški vmesnik razvijalcem omogoča, da lahko enostavno dostopajo do konfiguracije delovnih tokov, kar omogoča oblikovanje zapletenih delovnih poslovnih procesov le z nekaj kliki. Modeliranje in načrtovanje podatkovnih baz predstavlja težji del razvoja programskih rešitev malokodnih razvojnih okolij. Razvijalcem je omogočeno tudi dodajanje različnih razširitev, ki jih imenujemo vtičniki. To pomeni, da sami

lahko naredimo API-vmesnike, vendar tukaj potrebujemo malo več znanja programiranja oziroma bolj natančno znanje kodiranja. To nam omogoča povezavo z že obstoječimi ali novimi sistemi, v katere bomo integrirali novo programsko rešitev. Vtičnike, ki smo jih sami razvili, lahko tudi ponudimo brezplačno preko trga ali pod licenco za druge razvijalce. Glede na znanstveno literaturo in uporabo malokodnih razvojnih okolij, razvoj malokodnih programskih rešitev predstavlja univerzalen in inovativen koncept na področju programiranja. Trg malokodnih razvojnih okolij naj bi do leta 2022 ustvaril več kot 21,2 milijarde dobička, vendar to še ne pomeni, da je to uveljavljena tehnologija. Primer podobne tehnologije, ki se ni uvedel, je uporaba CASE (ang. Computer Aided Software Engineering) orodij, katerih glaven namen je bil avtomatizirati razvoj programske opreme s pomočjo različnih programov (Waszkowski, 2019).

Vsekakor je treba dobro premisliti pred uvedbo novega (malokodnega) razvojnega okolja. Ljudje se po naravi neradi spreminjamo, zato se lahko zaradi trenutnih trendov in glavnih igralcev pri razvoju spletnih tehnologij malokodna razvojna okolja nikoli ne uvedejo. Po drugi strani pa lahko prehitro uvajanje in nestrpnost vodita v razvoj »sivih rešitev«, saj lahko v uporabo pridejo rešitve, ki niso odobrene s strani IT oddelka. To lahko vodi tudi do tveganj glede skladnosti in varnosti poslovanja. To se danes do neke mere že dogaja z uporabo različnih oblačnih storitev, ki niso nujno ustrezno zavarovane za občutljive podatke. Ravno zato bo uvajanje malokodnih razvojnih okolij še toliko težje. Večina malokodnih razvojnih okolij sledi smernicam GDPR (ang. General Data Protection Regulation) in omogoča ustrezno varnost ter zasebnost podatkov. Razvojna okolja, ki so dostopna širši javnosti morajo podpirati čim več pravnih načel, saj le s tem lahko omogočijo razvijalcem, da razvijejo ustrezne programske rešitve tako za lokalni kot tudi za globalni trg. Prednosti in slabosti malokodnih razvojnih okolij so opisane na spodnji tabeli 3. Glavna prednost malokodnih razvojnih okolij je krajši čas razvoja. Glavna slabost pa je omejevanje uporabnika v razvojno okolje in različne zaledne nastavitve, ki jih razvijalec ne more ali zelo težko spreminja (Richardson & Rymer, 2016).

*Tabela 3: Prednosti in slabosti malokodnih razvojnih okolij*

| <b>Prednosti</b>                                                                                     | <b>Slabosti</b>                                                                                                          |
|------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| Hitro in bolj enostavno programiranje programskih rešitev brez ali z malo kodiranja.                 | Velikokrat omejujejo razvijalce (ang. Vendor Lock-in) v razvojno okolje.                                                 |
| Zmanjšuje stroške dela in zmanjša časovni okvir razvoja. Omogoča hiter prehod v produkcijsko okolje. | Težje nadziranje verzij programske rešitve in spreminjanje implementiranih funkcionalnosti je omejeno ali pa zelo težko. |

se nadaljuje

*Tabela 4: Prednosti in slabosti malokodnih razvojnih okolij (nad.)*

|                                                                                                            |                                                                           |
|------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------|
| Delovne predloge in vtičnik omogočajo hitro grajenje funkcionalnosti in zmanjšuje kompleksnost.            | Nepoznavanje procesov in težja modifikacija že narejenih funkcionalnosti. |
| Spreminjanje in prilaganje čelnega dela spletnih programskih rešitev je enostavno in prijazno uporabnikom. | Preveliko nastavitev povzroča, daljše obdobje učenja.                     |

*Vir: Richardson & Rymer (2016).*

#### 1.4.2 Brezkodna razvoja okolja

Razvoj brezkodnih okolij se je pojavil zaradi še večjih potreb po hitrem razvoju. V bistvu so to poenostavljena malokodna okolja, kjer je poudarek na prosti dostopnosti in razumevanju. Značilnosti brezkodnih okolij so, da ne omogočajo uporabo programskega jezika za razvoj programskih rešitev, vendar uporabljajo vizualno programiranje tako kot malokodna razvojna okolja. Čelni del gradimo s pomočjo grafičnega vmesnika WYSIWYG. Vse zaledne funkcionalnosti gradimo po delovnem toku IFTTT. Večina razvojnih okolij sploh nima posebnih zalednih procesov, ker so že vsi nastavljeni. To pomeni, da smo včasih lahko zelo omejeni na določene rešitve in funkcionalnosti. Urejanje in modeliranje podatkov nam omogoča kar vmesnik v razvojnem okolju. To nam olajša delo s podatkovnim strukturo in bazo podatkov. Ponudniki brezkodnih razvojnih okolij spodbujajo vsakega državljana in zaposlenega k razvoju programskih rešitev. Tudi tiste, ki niso tehnično podkovani. Glavni namen je približati razvoj programskih rešitev vsem ljudem (Gaggioli, 2017).

Brezkodna razvojna okolja ponavadi ne ponujajo razvoja procesov po meri. Vse, kar uporabnik potrebuje, je že nastavljeno in sprogramirano. Brezkodna razvojna okolja so na zelo visokem abstraktnem nivoju zato, da omogočajo hitrejši razvoj, vendar če uporabnik želi spreminjati določeno funkcijo za hitrejše ali varnejše izvajanje, je v večini primerov ne more, saj nima dostopa do pred nastavljenih procesov. Razlogi za to so varnost in optimizacija delovanja posameznih procesov. Slabost je tudi v tem, da nikakor ne moremo spreminjati varnostnih nastavitev, ki niso predvidene ali pa smo prisiljeni uporabljati neko standardno zaščito, kjer ena verzija ustreza vsem. Problem je tudi, da ne moremo uporabljati nikakršne optimizacije, saj sistem tega ne podpira. To bi lahko rešili s premium rešitvami, kjer bi razvijalci tesno sodelovali s ponudniki in njihovimi varnostnimi inženirji, vendar potem do tako hitrih rešitev ne bi nikoli več prišli. Pogostokrat se tudi brezkodna razvojna okolja označujejo kot »sive rešitve«. Glavna prednost brezkodnih razvojnih okolij je v predlogah, kjer je večino standardnih poslovnih procesov že prednastavljenih. To pomeni, da se lahko uporabnik v večjem delu posveti oblikovanju spletne programske rešitve in v manjšem delu poslovni logiki. Tudi brezkodna

razvojna okolja v splošnem sledijo smernicam GDPR-ja. Na tabeli 4 so prikazane prednosti in slabosti brezkodevskih razvojnih okolij (Richardson & Rymer, 2016).

*Tabela 5: Prednosti in slabosti brezkodevskih razvojnih okolij*

| <b>Prednosti</b>                                                                                           | <b>Slabosti</b>                                                                                                                         |
|------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| Zmanjšuje stroške dela in zmanjša časovni okvir razvoja.                                                   | Vsako razvojno okolje ima svoje pomanjkljivosti npr. pri nekaterih ni možnosti uporabljanja logičnih izrazov, razvoj API-jev je omejen. |
| Primerno za že uveljavljene in poznane poslovne procese.                                                   | Dodajanje funkcionalnosti po naročilu je skoraj nemogoče.                                                                               |
| Spreminjanje in prilaganje čelnega dela spletnih programskih rešitev je enostavno in prijazno uporabnikom. | Varnost je težko ustrezno spreminjati in prilagajati posameznim komponentam.                                                            |
| Pogostokrat že nastavljene predloge za hitrejšo delo.                                                      | Težje sledenje verzijam programskih rešitev in spreminjanje osnovnih funkcionalnosti je omejeno ali nemogoče.                           |
| Enostavno grajenje programskih rešitev brez kodiranja. Vsa logika je narejena po delovnem toku.            | Pogostokrat omejujejo razvijalce (ang. Vendor Lock-in) v brezkodevsko razvojno okolje.                                                  |

*Vir: Richardson & Rymer (2016).*

## **2 ZNAČILNOSTI SPLETNIH PROGRAMSKIH REŠITEV**

Pri razvoju programskih rešitev se moramo najprej odločiti, katero metodo bomo uporabili, kakšno arhitekturo bomo izbrali in kateri razvijalski vzorci so ustrezni za našo poslovno programsko rešitev. Vsaka spletna programska rešitev ima svoje prednosti in slabosti ter jo moramo razviti glede na potrebe in omejitve našega poslovanja. Glede na naše potrebe se moramo osredotočiti na eno metodo razvoja in jo obdržati skozi celotni proces. Spletne programske rešitve so v zelo dinamičnem okolju in se hitro spreminjajo. Prav tako jasno ločujejo med čelnimi in zalednimi sistemi. Arhitektura spletnih programskih rešitev opisuje interakcije med programskimi rešitvami, bazami podatkov, uporabniški vmesniki in zunanji sistemi na spletu. Zagotavlja, da več programskih rešitev deluje hkrati in omogoča delovanje poslovnega procesa. Danes je predvsem pomembno, da je spletna programska rešitev prijazna mobilnim in drugim prenosnim napravam. Prav tako je treba za razširljivost uporabiti ustrezno tehnologijo. Podjetja se danes pogostokrat za boljše delovanje, boljše odzivne čase in lažje

posodabljanje programske in strojne opreme odločajo za gostovanje na oblačnih storitvah, saj le ta omogočajo večjo dinamičnost in modularnost kot tradicionalne oblike razvoja. Več o tem v podpoglavju 2.4. Sedaj bom v nadaljevanju opisal, na kakšen način lahko uporabimo programske vmesnike in spletne povezovalne tehnologije in zakaj to predstavlja temelj pri razvoju spletnih programskih rešitev (Shahzad, 2017).

## **2.1 Programski vmesniki in spletne povezovalne tehnologije**

Programski vmesniki (API-ji) skrbijo za izmenjavo podatkov in imajo pomembno vlogo pri ustrezni strukturi podatkov in izmenjavi podatkov med sistemi ter drugimi programskimi rešitvami. Preko programskih vmesnikov lahko dodamo specifične funkcionalnosti za naše poslovne procese. Poznamo različne spletne programske vmesnike. Odprti programski vmesniki so javno dostopni in nimajo nobene omejitve, kdo jih uporablja in na kakšen način se uporabljajo. Partnerski programski vmesniki zahtevajo določene interne pravice ali licence za dostop in samo uporabo. Interni oz. privatni programski vmesniki omogočajo samo uporabo znotraj tega sistema. Te so po navadi zasnovani za uporabo znotraj podjetja. Sestavljeni uporabniški vmesnik združuje različne podatke in programske storitve. Lahko upravlja neko nalogo asinhrono in s tem pospeši postopek izvajanja ter zmogljivosti spletnih vmesnikov. Asinhrono pomeni, da lahko opravlja delo več zahtev na enkrat. Tako ostale zahteve ne čakajo na izvedbo drugega procesa, vendar se lahko izvajajo istočasno. Spletne programske vmesnike ponavadi delimo na naslednje štiri kategorije: SOAP (ang. Simple Object Access Protocol), JSON-PRC (ang. JavaScript Object Notation), XML-RPC, REST (ang. Representational State Transfer). Jasno moramo razlikovati med spletnimi povezovalnim tehnologijam in programskimi vmesniki. Spletne povezovalne tehnologije potrebujejo omrežje in pogosto uporabljajo SOAP in včasih REST ali XML-RPC). Programski vmesniki niso vezani na posamezni protokol in ne potrebujejo stalne povezave v omrežje. Povezovalne tehnologije povezujemo s storitveno usmerjeno arhitekturo SOA (ang. Service Oriented Architecture). To pomeni, da morajo biti spletne povezovalne tehnologije zgrajene tako, da že v samem začetku določijo kakšno strukturo podatkov bodo programske rešitve uporabljale. V primeru, da se zgodi napačna implementacija, se hitro znajdemo v neprimernem položaju, saj moramo tehnologijo v celoti zamenjati ali dodati še kakšen drug programski vmesnik, kar pa še otežuje vzdrževanje in posodabljanje. SOAP je protokol, ki uporablja XML format za prenos podatkov. Glavna funkcija tega protokola je opis strukture sporočila in metode za komunikacijo. XML-PRC je protokol, ki uporablja specifični XML za prenos podatkov v primerjavi s SOAP, ki uporablja splošnega. XML-PRC je enostavnejši od SOAP protokola in se ga lahko uporabi za manjše programske rešitve. JSON-RPC je podoben protokol XML-PRC-ju, vendar uporablja JSON format za shranjevanje in prenos podatkov. REST predstavlja arhitekturni princip, ki omogoča REST storitve. Te potrebujejo določene značilnosti, vključno s preprostimi vmesniki, ki jih je enostavno prepoznati v zahtevi spreminjanja podatkov. To pomeni, da REST protokol uporablja enostavne CRUD zahteve na strežniku s pomočjo HTTP zahtevkov. Na tabeli 5 spodaj je kratka predstavitev operacij REST v primerjavi s CRUD (Davis, 2020, str. 141).

Tabela 6: Primerjava HTTP zahtevkov v povezavi s CRUD na osnovi REST API-ja

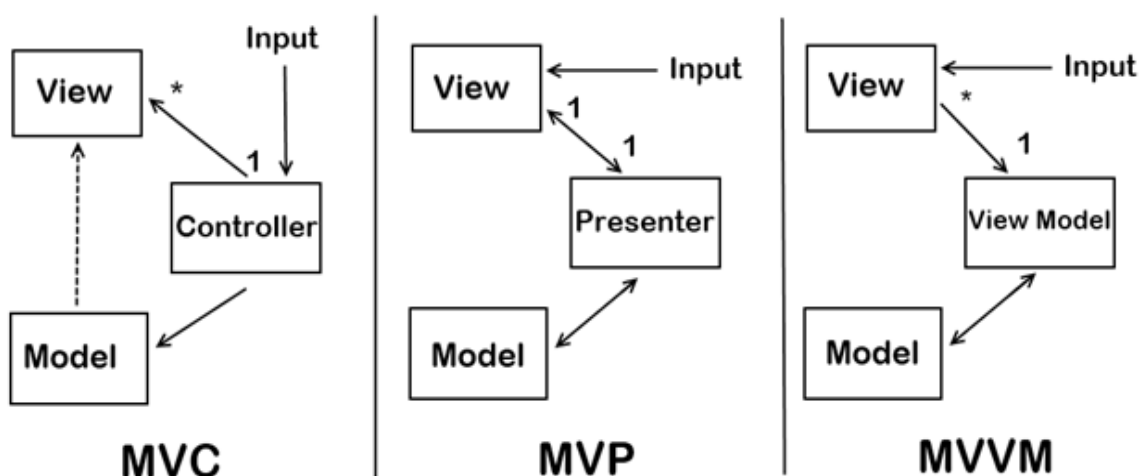
| HTTP zahtevek | CRUD     | Akcija                     |
|---------------|----------|----------------------------|
| GET           | preberi  | vrača zahtevane podatke    |
| POST          | ustvari  | ustvari nove podatke       |
| PUT           | posodobi | Posodobi obstoječe podatke |
| DELETE        | izbriši  | izbriše obstoječe podatke  |

Vir: Davis (2020).

## 2.2 Razvijalski vzorci spletnih programskih rešitev

Razvijalski vzorci predstavljajo načrt izvedbe projekta. Razvijalski vzorci so osnova za ustrezno arhitekturo spletnih programskih rešitev. Najbolj znani modeli razvijalskih vzorcev so MVC, MVP, MVVM in MOVE. Vsi razvijalski vzorci temeljno na n-slojni arhitekturi oblikovanja. Vsi omenjeni vzorci imajo skupne lastnosti in skrbijo za ohranjanje funkcionalnosti, ločitve poslovne logike, zaledje sistema in obvladovanja podatkov, ki skrbijo za optimalno uporabo spletnih programskih rešitev. Na sliki 4 so prikazane posamezne interakcije med komponentami za posamezne razvijalske vzorce. Za MVC je značilno, da model vedno deluje neposredno s pogledom. Pri MVP sta pogled in model ločena, saj prikazovalnik skrbi za posamezne poglede. MVVM ima to posebnost, da več pogledov vsebuje enake modelne poglede (Khaliluzzaman & Chowdhury, 2016).

Slika 4: Različni razvijalski vzorci



Vir: Boiko (2017).

### 2.2.1 MVC in MVP

Razvijalski vzorec MVC je najbolj dobro znan in se najpogosteje uporablja za arhitekturo dinamičnih spletnih programskih rešitev. MVC je sestavljen iz treh komponent. To so model, pogled in krmilnik. Vsak logični del je neodvisen od drugega, zato omogoča boljše delovanje celotne programske rešitve. Modelna komponenta skrbi za vse podatke, ki jih programska rešitev potrebuje za delovanje in interakcijo z uporabnikom. To lahko predstavlja prenašanje podatkov med krmilnikom in pogledom. Modeli lahko izvajajo zapisovanje v pomnilnik oziroma v bazo podatkov z uporabo skriptnih jezikov AJAX (ang. Asynchronous JavaScript and XML) ali z uporabo dinamičnih jezikov. Pogledi omogočajo prejemanje zahtevkov preko dogodkov, ki jih ustvarijo modeli in upravljajo s posodabljanjem uporabniškega vmesnika – UI. Prav tako omogočajo uporabo predloge za prikaz informacij v oblikovanju z zasnovo uporabniškega vmesnika. Krmilniki imajo naloge upravljanja poslovne logike, obdelovanja ukazov in spreminjanja podatkov, ki preko pogledov prikažejo končne rezultate, ter pošiljanja podatkov v model. Krmilnik skrbi in obdeluje uporabniške vnose in izvaja spremembe na model. Krmilnik lahko upravlja več različnih pogledov (Rehman, Ullah, Tanvir & Azam, 2018).

MVP je razvijalski vzorec, ki je zelo podoben MVC-ju. Komponente so si med seboj zelo podobne, vendar imajo nekaj pomembnih sprememb. Krmilnik zamenja prikazovalnik. Ta razvijalski vzorec je sestavljen iz modela, pogleda in prikazovalnika. Model vsebuje vse podatke, ki jih potrebuje programska rešitev. Prikazovalnik vsebuje vsa poslovna pravila za delovanje in spreminjanje pogledov. Pogled nam predstavlja direktno interakcijo z uporabnikom. Prikazovalnik komunicira s pogledom preko vmesnika. Prikazovalnik nato pridobi podatke o uporabnikih preko pogleda in potem s pomočjo modela preoblikuje te podatke ter jih ustrezno vrne v pogled. Izjemoma se pogled lahko posodobi tudi preko modela, ko gre za manjše posodobitve. Zahtevane podatke vedno posredujemo preko vmesnika, ki so določeni v prikazovalnem razredu (Boiko, 2017).

### 2.2.2 MVVM in MOVE

MVVM je sodobni razvijalski vzorec, ki omogoča dvosmerno izmenjavo podatkov med pogledom in Modelom-Pogleda. Pogled se vedno sklicuje na Model-Pogleda in ne vsebuje informacij o pogledu. To je prednost in tudi razlog, zakaj je možna medsebojna preslikava med različnimi pogledi in enim modelom-pogleda. Vzorec MVVM uporablja Windows Presentation Foundation in Silverlight in se izvaja v ogrodjih Java za Android ter je bolj primeren za razvoj mobilnih programskih rešitev (Boiko, 2017).

MOVE predstavlja sodobni razvijalski vzorec, ki dodaja operacije in dogodke poleg že omenjenega modela in pogleda. Omogoča prilagoditev procesov na podlagi operacij, ki vplivajo na spremembe modelov in upravljajo s pravilnimi pogledi. Operacije služijo kot metode za prikazovanje pravih pogledov in proženje dogodkov od uporabniških interakcij. Prilagoditvene operacije so predstavljene kot niz omejitev, ki so odvisne od konteksta in naloge

ter določajo potek procesa. Celotno programsko rešitev lahko smatramo kot skupek manjših operacij. Opravljanje dogodkov je tisto, kar daje MOVE vzorcu prednost. Pogledi ne spreminjajo modelov neposredno, vendar delujejo preko operacij. Zahtevki se pošiljajo direktno v operacije, kjer čakajo na spremembe dogodkov, ki jih oddajajo modeli. MOVE omogoča, da se komponente med seboj povežejo na tak način, da so procesi najbolj optimizirani in prikaz podatkov ni zakasnen. (Malar, 2019).

## 2.3 Strukturne raznolikosti spletnih programskih rešitev

Glede na postavljeno arhitekturo in razvijalske vzorce lahko ustvarimo spletno programsko rešitev, ki ima natančno določene komponente. V nadaljevanju bomo opisali posamezne splošno uporabljene strukture spletnih programskih rešitev (López, Zorrozuza, Ruiz, Rodríguez-Artacho, & Gil, 2016):

**Statične programske rešitve** so tiste, ki imajo celotno vsebino statično in niso fleksibilne. Vse podatke predhodno naloži strežnik in so skoraj brez interaktivnosti, ki jo po navadi ponuja skriptni jezik (JS). Statične programske rešitve je zelo težko vzdrževati, saj vsebujejo veliko količino podatkov. Strežniki so tako zelo obremenjeni, saj v celoti nalagajo strani s podatki, kar posledično povzroča varnostno tveganje in slabše delovanje. Prav tako niso primerne za mobilno okolje. Praktičnost statičnih programskih rešitev se tako kaže le v predstavitvenih straneh, kjer je malo informacij in skoraj nič interakcije. Te programske rešitve po navadi ne sledijo nobenemu razvijalskemu vzorcu ali posebni arhitekturi.

**Dinamične programske rešitve** vključujejo razvijalske vzorce, kot je npr. MVC. Po navadi temeljijo na posameznih ogrodjih. Način, kako so podatki prikazani na uporabniškem zaslonu, ni vnaprej določen, temveč je dinamično oblikovan z logiko razvijalskih vzorcev na čelnem in zalednem delu. Take spletne strani se uporabljajo za napredne predstavitvene strani, saj je logika enostavna.

**Več stranske programske rešitve** oz. MPA (ang. Multi Page Apps.) uporabljajo skoraj celotno logiko v zaledju. To pomeni, da za kakršno koli spremembo začnejo na strežniku veljati vse zahteve odjemalca. Poleg uporabe ogrodja je bilo to načelo skoraj enako kot v preteklosti pri statičnih spletnih programskih rešitvah. Sodobne MPA izkoriščajo tehnologijo AJAX-a, ki omogočajo takojšnje spremembe brez ponovnega nalaganja strani. Zasnovane so kot odzivne in se lahko v celoti prilagodijo mobilnemu okolju, npr. z uporabo ogrodja Bootstrap. Prav tako so MPA zelo varne, saj je uporabniku logika nedosegljiva, zato se pogosto uporabljajo za spletne portale in poslovne programske rešitve.

**Bogate programske rešitve** oz. RIA (ang. Rich Internet Apps) so na začetku poizkušale preseči vse omejitve brskalnika in so se močno opirale na vtičnike na strani odjemalca, kot so: Shockwave, Flash in Silverlight. Te vtičniki naj bi skrbeli za dinamičnost in interaktivnost spletnih programskih rešitev. Za ustrezno delovanje naj bi tako uporabniki sami skrbeli za posodobitve in vzdrževanje vtičnikov. Kmalu se je pojavil problem, saj so bili te vtičniki

pogosta tarča zlorab, pri čemer je nedelovanje ene funkcionalnosti pomenilo popolno odpoved delovanja programske rešitve. HTML5 je omogočil preskok in nadomestil vse bogate programske rešitve. Hrkati je rešil tudi problema ozkega grla nedelovanja. Tako nedelovanje ene funkcije še ne pomeni nedelovanje celotne programske rešitve. Tudi varnost je postala bolj ključna in manj odvisna od nepravilnih oz. neustreznih nastavitev uporabnika ali varnostnih lukenj ponudnika.

**Enostranske programske rešitve** oz. SPA (ang. Single Page Apps) izkoriščajo prednost izvajanja na strani odjemalca. Struktura teh spletnih programskih rešitev omogoča hitro delovanje in izvajanje zahtevkov na strežniku. Vsaka spletna programska rešitev se lahko močno izboljša, ko postane enostranska. Problem je le v SEO (ang. Search Engine Optimization), kjer enostranske rešitve niso kompatibilne z regulacijam iskalnikov in dejansko škodujemo sodobnim iskalnikom, da bi lahko dobro indeksirali naše strani. To velja predvsem za spletne trgovine, saj nas kupci potem težje najdejo. Nekatere programske rešitve, kot so npr. spletne pošte, družbena omrežja, spletni predvajalniki, pa to tehnologijo zelo dobro izkoriščajo in imajo veliko koristi pri delovanju.

**Progresivne spletne programske rešitve** oz. PWA (ang. Progressive Web Apps) ne predstavljajo nove revolucionarne arhitekturne oblike, vendar omogočajo boljšo interakcijo za mobilne naprave. Funkcije, ki izboljšujejo prilagodljivost za mobilne naprave, so boljše predpomnjenje in boljši prenos podatkov preko HTTP/2 protokola. Poleg tega je eden najpomembnejših delov progresivnih programskih rešitev izboljšanje uporabniške izkušnje za mobilni splet, ki ima slabo ali počasno spletno povezavo. Vse progresivne programske rešitve je treba implementirati preko protokola HTTPS, da zagotovimo varnost. Vse ostale programske rešitve je težje optimizirati, da bi bili tako učinkovite, kot so progresivne programske rešitve. Večina značilnosti in konceptov, ki ločujejo PWA od običajnih spletnih programskih rešitev je uporaba »servisnih funkcij«. Servisna funkcija izvaja operacije v ozadju, ločeno od preostalega spletnega mesta, saj ne more komunicirati z DOM. Servisna funkcija deluje kot posredniški strežnik na aplikacijskem nivoju in tako razvijalcem omogoča, da v celoti prestrežejo omrežne zahteve in nadzorujejo procese, kot so predpomnjenje podatkov, uporaba funkcionalnosti brez spletne povezave, sinhronizacija v ozadju (npr. pridobivanje podatkov, ko je aplikacija v ozadju), pošiljanje potisnih obvestil in še mnogo drugih (Biørn-Hansen, Majchrzak & Grønli, 2017).

## **2.4 Zaledni sistemi**

Zaledni sistemi sprejemajo dohodne poizvedbe in pošiljajo ter generirajo odgovore odjemalcem. Za odjemalca imamo namizno aplikacijo, ki jo imenujejo brskalnik. Brskalnik je aplikacija, s katero dostopamo do spleta. Uporabniki do naših spletnih strani oz. programskih rešitev ne bi morali dostopati brez strežnikov. Zaledni sistemi običajno vključujejo: strežnike kot strojno opremo, vse konfiguracije strežnikov in podatkovno bazo kot programsko opremo. Primarna funkcionalnost strežnikov je obdelava, shranjevanje, brisanje in pošiljanje zahtevkov

odjemalcem. Zaledni sistemi vsebujejo vse potrebno, da programske rešitve lahko delujejo. Takoj, ko uporabnik vtipka URL naslov v naslovno vrstico spletnega brskalnika in pritisne gumb »Pojdi«, zahteva določen spletni naslov. Strežnik nato pošlje datoteke v brskalnik kot odgovor na opravljeno zahtevo. Brskalnik izvrši te datoteke in prikaže zahtevano stran.

Za delovanje in prikaz spletnih strani oziroma programskih rešitev potrebujemo spletni strežnik DNS, ki obdeluje zahteve HTTP (ang. HyperText Transport Protocol). HTTP je osnoven omrežni protokol, ki omogoča komunikacijo med odjemalcem in strežnikom. Ta strežnik imenujemo DNS strežnik (ang. Domain Name System), ki omogoča, da uporabnik dostopa do spletne strani oz. do spletne programske rešitve.

Zaledni sistemi skrbijo tudi za varno izvajanje transakcij in shranjevanje podatkov. Podatki so lahko shranjeni v relacijski podatkovni, nerelacijski podatkovni bazi ali kakšni drugi alternativni podatkovni obliki. Najbolj pogosta je uporaba relacijskih podatkovnih baz. V splošnem velja pravilo, da je za strukturirane in kompleksne sisteme bolj primerno uporabiti relacijsko podatkovno bazo. Neralacijske baze so sicer hitrejše, vendar so primerne le, če se struktura podatkov ne bo veliko spreminjala. Poznamo tudi druge alternative, kot so grafične podatkovne baze, za katere se razvijalci redko odločajo, saj so primerne le za določene bolj specifične potrebe kot npr. za prikaz medsebojne odvisnosti med posameznimi objekti. Za dostop do podatkovnih baz uporabljamo različne sisteme, ki jih imenujemo sistemi za upravljanje s podatkovnim bazami ali krajše SUPB. To so sistemi, ki nam pri relacijskih bazah s pomočjo jezika SQL (ang. Structured Query Language). SQL je strukturiran poizvedovalni jezik, ki omogoča ustrezno obdelavo podatkov in CRUD operacije. Primeri znanih in uporabljenih podatkovnih baz so: MySQL, PostgreSQL, Oracle DB in SQLite. Za pravilno delovanje spletnih programskih rešitev ne potrebujemo samo ustrezne podatkovne baze in spletnega strežnika, temveč moramo poskrbeti za pravilno izbiro gostovanja, domenskega prostora, arhitekturo, možnost izbire vsebnikov ali virtualizacije, pravilno pravno skladnost, informacijsko in kibernetsko varnost. Vse to opišem v naslednjih podpoglavjih (Boiko, 2017).

#### 2.4.1 Gostovanje in domenski prostor

Za vsako spletno programsko rešitev potrebujemo gostovanje in domeno. Gostovanje predstavlja najem prostora za našo spletno programsko rešitev na spletnem strežniku. Običajno se za gostovanje plačuje na (več-)letni ravni. Domena predstavlja ime naše spletne strani npr. »www.primer.com«. To je naslov, ki omogoča vsakemu uporabniku spleta dostop do te spletne strani. Vsaka spletna stran ima tudi svoj domenski prostor. Ta predstavlja zadnji del povezave v našem primeru kratica »com«. Ta prostor lahko prikažemo kot drevesno strukturo, ki je razdeljena na domene in poddomene. Vsaka spletna stran vsebuje eno glavno domeno in več poddomen. Domena in gostovanje predstavljata osnovo, ki je potrebujemo, da lahko naše programske rešitve uporabljamo in prikazujemo na spletu. Izbira ponudnikov je danes težka predvsem zaradi velike raznolikosti. Najbolj pomembna odločitev, ki jo moramo sprejeti najprej je ali bodo naše spletne programske rešitve v našem sistemskem prostoru ali jih bomo

uporabljali v oblaku. Za lažjo izbiro je mogoče slediti tem preprostim smernicam: varnost mojih podatkov, hitrost odziva z izbiro SDD diskov (ang. Solid state drive) ali HDD diskov (ang. Hard Disk Drive), kvalitetni UI za konfiguracije in dostop do izvornega (ang. Root) dostopa in razširljivost sistema. SSD diski so boljši in hitrejši, ker nimajo mehanskih delov. Zato se danes v primeru večjih spletnih programskih rešitev odločamo za SDD gostovanje. Razširljivost se nanaša na zmožnost prilagajanja in variiranja zahtevkov za določeno spletno programsko rešitev. Razširljivost se bolj navezuje na fizične zmogljivosti naših strežnikov. Zanima nas, ali je sistemski prostor ustrezen in ali lahko čim hitreje in z malo stroškov prilagodimo naš strežnik novim potrebam na strojnem nivoju. Gostovanje danes tako postaja vse bolj kompleksno. Za boljši doseg uporabnikov je potrebno tudi optimizirati dostopnost naše splete strani. To imenujemo tudi SEO optimizacija. SEO optimizacija omogoča, da iskalniki indeksirajo naše spletne strani bolje od ostalih in nas prikazujejo višje pri zadetkih v iskalniku. Deluje na podlagi preprostih pravil: mobilno prijazna spletna stran, boljše uporabljene ključne besede, ustrezno uporabljeni standardi W3C programske kode in druge zahteve. Zaradi velike uporabe spleta se vse več potreb poslovanja kaže po bolj decentraliziranem gostovanju. Tak primer sistema bi lahko temeljil na medplanetarnem datotečnem sistemu IPFS (ang. Interplanetary File System) in tehnologiji blokov v Ethereum-u. IPFS sistem uporablja principu vsak z vsakim (ang. Peer to Peer) in uporablja porazdeljen datotečni sistem tako kot ga uporablja Torrent (Hoffman, Becerril-Blas, Moreno & Kim, 2020). To pomeni, da vozlišča komunicirajo preko omrežja »Peer to Peer« s parom zasebnega in javnega ključa. Prednost tega sistema je porazdelitev odgovornosti in izognitev ozkega grla pri kraji podatkov ali vdoru v informacijski sistem. Uporaba tega sistema ne bi bila primerna pri transakcijskih strežnikih, saj je prepočasen. Problem tehnologije blokov je v tem, da posamezne transakcije lahko trajajo od 15 sekund do 5 minut v primerjavi z MasterCard-om, kjer transakcija v povprečju traja manj kot 1 sekundo. IPFS sistem bi lahko tako uporabljali za podatkovna skladišča ali za hranjenje osebnih oziroma občutljivih podatkov (Huynh, Nguyen & Tan, 2019).

Danes vsi ponudniki gostovanja igrajo ključno vlogo pri zagotavljanju varnosti vseh vrst spletnih storitev in preprečujejo zlorabo storitev. Kriminalci kupujejo, vdirajo v storitve za gostovanje, vrivajo zlonamerno programsko kodo, izvajajo lažno predstavljanje, prevzemajo ukazne in nadzorne (C&C) strežnike, izvajajo krajo domen in omogočajo razvoj črnih trgov, ki lahko vsebujejo kaznive storitve ali izdelke. Več o varnosti bom predstavil v podpoglavju 2.4.5. Primer izkoriščanja gostovanja je bil na primer Silk Road. Silk Road je bil spletni črni trg, ki je preprodajal vse od drog do ukradenih (občutljivih) osebnih podatkov. Kasneje so to stran uspeli ukiniti in kaznovati uporabnike. Kazensko so odgovarjali tudi ponudniki gostovanja. Odgovornost ponudnikov gostovanja je velika, saj so posledice v primeru, da pride do neustreznega ali celo kaznivega dejanja velike tako za ponudnika kot tudi naročnika (Tajalizadehkhoo, Korczyński, Noroozian, Ganán & van Eeten, 2016).

#### 2.4.2 Monolitna in mikrororitvena arhitektura

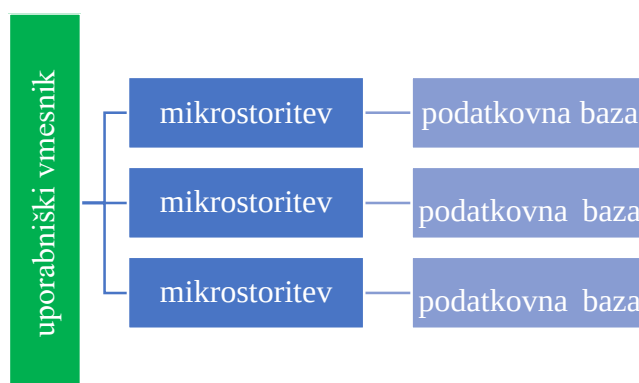
Arhitektura sodobnih poslovnih spletnih programskih rešitev je po navadi monolitna in razvita za določeno ciljno strojno opremo, vendar to ni dovolj ustrezno za današnje potrebe, saj ne omogoča razširljivosti. Zato je danes uporabnost mikrororitve v porastu, saj omogočajo modularnost. Poleg tega omogočajo tudi storitveno usmerjeno arhitekturo (Gallipeau & Kudrle, 2018). Običajno se je razvoj tradicionalnih programskih rešitev nagibal bolj h monolitni arhitekturi. Monolitna arhitektura pomeni, da je zaradi velike medsebojne povezanosti zelo težko nagraditi posamezni del programske rešitve, ne da bi vplivali na logiko in samo delovanje rešitve. Na sliki 5 je prikazana tradicionalna monolitska arhitektura spletne programske rešitve. V splošnem velja enostavno pravilo za arhitekturo enostavne programske rešitve uporabimo monolitsko arhitekturo. Mikrororitveno arhitekturo uporabimo pri sodobnih programskih rešitvah, kjer je možno enostavno nadgrajevanje posameznih funkcionalnosti. Prednost mikrororitev je predvsem v stopnji razširljivosti in obvladovanju napak. Posledično mora biti zasnovana arhitektura programskih rešitev tako, da lahko kljub odpovedi ali izpadu ene storitve druge delujejo nemoteno. Vsaka mikrororitev je zgrajena po principu monolitske arhitekture, torej vsaka posamezna funkcionalnost predstavlja t. i. vertikalo v monolitski arhitekturi. Znotraj posamezne arhitekture mikrororitev je celotna funkcionalnost razdeljena na neodvisno uporabne module, ki med seboj komunicirajo s pomočjo določenih metod, imenovanih API-ji. Slika 6 prikazuje arhitekturo mikrororitev. Obremenjenost posamezne mikrororitve se prilagaja zahtevam z elastičnim upravljanjem zmogljivosti. Pri velikih ali majhnih obremenitvah se te zmogljivosti dinamično prilagodijo brez posredovanja, temu rečemo tudi dinamična razširljivost. Implementacija sodobnih programskih rešitev zahteva natančno poznavanje poslovnih procesov, saj mikrororitve same po sebi vpeljejo kompleksnost in izzive (Hasselbring & Steinacker, 2017).

*Slika 5: Monolitna programska rešitev*



*Vir: Lastno delo.*

*Slika 6: Mikrororitvena arhitektura*



*Vir: Lastno delo.*

### 2.4.3 Virtualizacija in vsebniki

Za zmanjšanje naporov pri razvoju novih programskih rešitev ali za prenos obstoječih programskih rešitev na drugo strojno opremo je treba zagotoviti razvojno okolje, ki je neodvisno od monolitske arhitekture. Porast oblačnih sistemov je razvijalce prisililo v bolj dinamičen razvoj arhitekture po delih. Pred pojavom vsebnikov je bila virtualizacija strežnikov edina možnost za optimizacijo zmogljivosti strežnikov. Virtualizacijo nam omogoča program hipervizor, s katerim lahko posnemamo strojno opremo in operacijski sistem večjega števila strežnikov na strojni opremi enega fizičnega strežnika. Virtualizacija se danes še vedno bolj uporablja kot vsebniki. Zaradi poslovne in tudi tehnične narave vsebniki niso ustrezni za reševanje vseh kompleksnih zahtev naročnikov. Vse je odvisno od tega kakšne sisteme in programske rešitve uporabljamo. Kasneje se nato odločimo, ali bomo uporabljali virtualizacijo ali vsebnike. Sodobne spletne programske rešitve vse več uporabljajo vsebnike za optimizacijo procesov, zaradi lažje prilagodljivosti in optimalnega delovanja (Hawilo, Jammal & Shami, 2019). V oblačnih sistemih je trenutni trend nadomeščanja težkih virtualnih strojev z lahkimi rešitvami vsebnikov, kot je npr. Docker. Docker je vodilni ponudnik mikrororitv na trgu. Uporaba Docker-ja postaja iz dneva v dan vse pogostejša. Koncept nadzora vsebnikov mora izpolnjevati zahteve industrijskega nadzora, kot so zmogljivost v realnem času in velika razpoložljivost. Prav tako sem spadajo tudi zahteve, ki izhajajo iz razgradnje monolitne arhitekture programske opreme v mikrororitveno arhitekturo. Podpora operacijskega sistema za virtualizacijo ali vsebnikov je zelo različna. Vsak virtualni stroj ima svoj operacijski sistem nad gostiteljskim. Vsebniki si delijo gostiteljski operacijski sistem in s tem zmanjšajo kompleksnost in pohitrijo delovanje programskih rešitev. Docker vsebniki so primerni takrat, ko želimo zagnati več programskih rešitev na enem operacijskem sistemu. Za programske rešitve, ki morajo delovati z različnimi operacijskimi sistemi, pa potrebujejo uporabo virtualnih strojev (Tasci, Melcher & Verl, 2018).

### 2.4.4 Pravna skladnost programskih rešitev

Pri razvoju programskih rešitev ne smemo pozabiti na pravilno implementacijo pravnih aktov in upoštevanje vseh lokalnih in mednarodnih predpisov, kot je npr. Splošna uredba EU o varstvu podatkov oziroma krajše GDPR. GDPR je mednarodni pravni akt, ki ureja področje upravljanja z osebnimi podatki. GDPR je bil uveden maja leta 2018 in je močno spremenil delovanje celotnega spleta. Primarne točke zakonika so naslednje. Enostavnost dostopa do osebnih podatkov s strani uporabnika. To pomeni, da mora vsak uporabnik imeti dostop do svojih osebnih podatkov. Ti podatki morajo biti na voljo na jasn in razumljiv način. Poenotenje in standardizacija osebnih podatkov različnim spletnimi servisom omogoča enostaven prenos in posodabljanje. Najbolj zahtevna postavka, ki jo določa ta zakonik je pravica do izbrisa podatkov posameznika oziroma pravica do pozabe. Vsako podjetje je zavezano, da podatke o vaši zgodovini uporabe te storitve v celoti izbriše. To ni ravno enostavno, saj zahteva izbris celotne zgodovine podatkov uporabnika, ki so razpršeni po različnih sistemih in so že bili vključeni v nadaljnje raziskave, trženje ali katero drugo obliko obdelovanja ali pa se morajo podatki hraniti

zaradi razlogov arhiviranja in je take podatke težko v celoti ustrezno izbrisati. GDPR omogoča tudi pravico vedenja, kdaj so bili tvoji osebni podatki ukradeni ali zlorabljeni. Problem je le v tem, ker le redka podjetja pravočasno priznajo ali objavijo vdor v sistem. Zato so se na spletu pojavile samostojne storitve, ki nam to omogočajo na primer spletna stran: <https://haveibeenpwned.com>, nam omogoča sporočanje v primeru, da so bili naši osebni podatki ali gesla ukradeni. Pri razvoju spletnih programskih rešitev je trg postal zasičen z neprimernimi in celo spornimi praksami. V splošnem je potrebno upoštevati naslednja načela. Jasno je treba določiti ali sploh potrebujemo vse osebne podatke, saj s tem zmanjšamo breme obvladovanja in shranjevanja. Podatki, ki so si jih podjetja pogostokrat delila in s katerim nas lahko tudi sledijo preko spleta so spletni piškotki. Spletni piškotki so preprosti koščki podatkov, ki omogočajo beleženje in shranjevanje podatkov. GDPR je jasno opredelil, kako in do kakšne mere lahko uporabljamo določene spletne piškotke. Podjetja morajo danes nujno uporabljati vsaj en način šifriranja vseh osebnih podatkov. V primeru, da imamo v programski rešitvi osebne podatke, jih moramo šifrirati z močnimi šifrirnimi algoritmi, ki ustrezajo sodobnim standardom. Teh je zelo veliko in eden izmed prosto dostopnih je algoritem Blowfish. Protokoli za enotno prijavo morajo omogočati popolno anonimnost, kot je npr. OAuth. OAuth je odprtokodni avtentikacijski protokol, ki uporabnikom omogoča ustvarjanje računov, tako da preprosto zagotovijo drug račun iz druge spletne storitve, hkrati pa zagotavljajo, da niso shranjeni nobeni osebni podatki razen identifikacijskega ID-ja druge storitve. Vse sodobne spletne programske rešitve morajo tako uporabljati protokol HTTPS. To je protokol, ki kodira in povečuje varnost prenosa podatkov preko spleta. Deluje na podlagi prenosa certifikat SSL, ki identificira istovetnost pošiljatelja in ustrezno določi, da je to res oseba, za katero se predstavlja. Ustrezno morajo biti nameščeni tudi SSL (ang. Secure Sockets Layer) certifikati tako, da niso izpostavljeni ranljivostim. SSL je kriptografski protokol, katerega namen omogočanja je varnost na spletu, saj podatke pri prenosu šifrira. Predvsem se uporablja pri spletnih trgovinah, kjer gre za prenos občutljivih podatkov zaupne narave. To so npr. številke kreditnih kartic in drugi osebni podatki. Zagotoviti moramo, da so vsi piškotki in seje uničene ob izhodu programske rešitve. Prav tako moramo uporabniku ponuditi možnost, da izklopi profiliranje. To pomeni, da na spletnih trgovinah ne dobivamo različnih priporočil o izdelkih, a kljub temu še vedno vse do danes ni možno zaslediti popolne implementacije te rešitve. Podjetja kot za neupoštevanje tega uporabljajo izgovor, da bi to poslabšalo delovanje priporočil. Pomembno je, da moramo vse podatke tudi iz preprostih obrazcev za kontaktiranje šifrirati. Za skladno delovanje z GDPR moramo jasno opredeliti vso politiko za ravnanje ob podatkovnem vdoru. Uporabniki morajo biti takoj obveščeni ob vdoru še posebej, če gre za bazo z bančnimi ali občutljivimi osebnimi podatki. Izgovorov za neuporabljanje HTTPS protokola danes ni več, razen če gre zgolj za predstavitevno stran brez spletnih obrazcev. GDPR tudi zahteva, da se pri obnavljanju gesla vprašanja nikoli ne smejo nanašati na osebne podatke. Najbolj pomembno je, da imajo uporabniki jasno in razumljivo napisane pogoje uporabe in jih lahko vsak uporabnik razume (Bruschi, 2019).

GDPR je močno spremenil celotno poslovanje na spletu. Predvsem na področju kršenja pravil in izrekanje denarnih kazni pri spletnem poslovanju. Kršitve GDPR-ja povzročajo zelo visoke

stroške poslovanja. Predvidene denarne kazni za kršitve so do 20 milijonov evrov oziroma do 4 % celotnega letnega prometa za celotno preteklo finančno leto. Izbere se tisto kazen, ki ima višji znesek (Splošna uredba EU o varstvu podatkov, 2018).

Splet je postal nasičen z vsemi podatki uporabnikov, zato potrebujemo ustrezno pravno ureditev. GDPR je prinesel tudi veliko organizacijskih sprememb, predvsem na področju spleta. Podjetja morajo ustrezno prilagoditi načine shranjevanja v podatkovne baze, kjer hranijo osebne ali celi občutljive podatke. GDPR zahteva od subjektov javnega sektorja in tudi večine zasebnih poslovnih subjektov, da imenujejo pooblaščen osebo za varstvo podatkov oziroma DPO-je (ang. Data Protection Officer). To more biti oseba, ki nima konflikta interesov in je neodvisna od poslovanja ter lahko zagotovi ustrezno neodvisno delovanje (Linden, Khandelwal, Harkous & Fawaz, 2020).

Slovenija na žalost do danes še vedno ni uskladila novega Zakona o varstvu osebnih podatkov (ZVOP-2). GDPR zato ne more stopiti v popolno veljavo. Zmedo predvsem povzročajo postopki, ki pokrivajo kršitve, saj v Zakonu o varstvu osebnih podatkov (ZVOP-1) Ur. l. RS, št. 94/07 – uradno prečiščeno besedilo, niso predvidene sankcije za kršitve na spletu. Težave z integracijo se kažejo, ker je trenutni osnutek ZVOP-2 prepisoval člene GDPR-ja. GDPR člene pa se lahko le dopolnjuje in ne tudi spreminja. Težave imajo tako predvsem podjetja, ki se ukvarjajo s spletnim oglaševanjem, saj je trenutno v veljavi še vedno star zakonik ZVOP-1 v kombinaciji z GDPR-jem (IP GDPR, 2020).

#### 2.4.5 Informacijska in kibernetska varnost

Informacijska varnost skrbi za integriteto, zaupnost in razpoložljivost podatkov v podjetjih. Integriteta pomeni, da so izvorni podatki nespremenjeni, konsistentni, celoviti in ustrezno shranjeni. Zaupnost podatkov pomeni, da jih lahko uporablja samo tista oseba, ki ima pravice dostopa do njih. Razpoložljivost podatkov predstavlja, da so podatki na voljo, ko jih potrebujemo. Vsa podjetja se danes v večji meri ukvarjajo z obdelavo podatkov, zato da le ta poteka v skladu z vsemi zakoni in priporočili, morajo podjetja slediti smernicam ISO 27001 certifikatu. Za lažje poslovanje morajo podjetja, ki se ukvarja z razvojem programske opreme imeti certifikate za informacijsko varnost. To še posebej velja za ponudnike razvojnih okolij, kot jih sam opisujem v magistrskem delu (Safa, Von Solms & Furnell, 2016).

Kibernetska varnost predstavlja zaščito računalniških sistemov, omrežij in programov pred digitalnimi napadi. Kibernetski napadi zajemajo: nepooblaščen dostop, spreminjanje ali uničenje občutljivih informacij, izsiljevanje denarja od uporabnikov in prekinitve običajnih poslovnih procesov (Cisco, 2020). Kibernetska varnost predstavlja spreminjajočo se panogo, kjer razvoj novih tehnologij omogoča nove možnosti za napade. Varnost zajema varovanje ljudi, procesov in tehnologij. Vsako podjetje bi moralo svoje zaposlene obvarovati pred napadi tako, da ustrezno implementirajo upravljanje s tveganji, ljudi ustrezno izobražujejo in stalno posodablajo informacijske sisteme na ustrezne varnostne standarde. Panoga kibernetske varnosti je zelo pomembna. Zaradi vse večje uporabe spleta in povečanje digitalnega poslovanja

ima kibernetika varnost vedno bolj pomembno vlogo. Predvideno je, da bo kibernetike varnosti do leta 2024 ustvarila več kot 300 milijard dolarjev dobička (Global Market Insights Inc., 2020).

Poznamo različne tipe kibernetiki napadov. Delimo jih na tri glavne kategorije: družbeni inženiring, izsiljevalska programska oprema in zlonamerna programska oprema. Družbeni inženiring je najbolj pogosta metoda. Ta metoda ne vključuje nobenega tehničnega vidika, temveč uporablja družbene sposobnosti tako, da zavedejo uporabnika. S tem nato pridobijo zaupne informacije ali celo denarna sredstva. Ta metoda ima pri vdorih in zlorabah največjo stopnjo uspešnosti do kar 70 %. Lažno predstavlanje (ang. Phishing) je metoda zlonamerne programske opreme, ki se uporablja v kombinaciji z družbenim inženiringom, s katero napadalec preko e-pošte ali spletnih obrazcev pridobi občutljive podatke za vpis v različne sisteme. To stori tako, da pošlje povezavo do prevarantske strani, ki zgleda popolnoma enaka kot prava stran. Nato uporabnik v prevarantski obrazec uporabnik vpiše svoje uporabniški ime in geslo. V primeru, da uporabnik obrazec pošlje, so podatki ob potrditvi ukradeni. Pred to metodo se lahko zavarujem tako, da vedno lahko pogledamo ali je URL spletne strani res pravi. Lažno predstavlanje lahko delno rešimo z ustrezno nastavljeno dvojno ali več stopenjsko avtentikacijo, ki jo po navadi označujemo s kratico 2FA (ang. Two-factor authentication), ki ima enkratno geslo oziroma OTP (ang. One Time Password). Tudi ta metoda ni popolnoma zanesljiva, saj napadalci lahko v realnem času preverijo na pravilnem strežniku naše ujemanje uporabniškega računa in gesla ter nam pošljejo še dodatno potrditev za 2FA na telefon ali spletno stran. V zadnjih časih se najbolj pogosto uporablja metoda direktorske prevare, ki je ena izmed uspešnejših oblik družbenega inženiringa. Ta prikazuje način lažnega predstavljanja znotraj podjetja. Napadalec pošlje e-pošto iz ponarejenega direktorjevega e-poštnega naslova po navadi računovodstvu ali oddelku za finance za nakaz denarja na bančni račun napadalca. Zaposleni morajo vedno preverjati, kdo je pošiljatelj in ali je e-poštni naslov pravilen ter ob nakazilih vedno večkrat preveriti istovetnost. Izobraževanje zaposlenih je ključni del pri kibernetiskem varovanju in omogoča nemotenem poslovanje. Še večjo nevarnost predstavlja izsiljevalska programska oprema. Izsiljevalska programska oprema predstavlja zaseganje sistemov in onemogočanje dostopa do celotnih sistemov ali datotek, dokler ne poravnamo denarnih sredstev. Tudi po plačanih denarnih sredstvih ni zagotovila, da bomo dobili nazaj dostop do naših podatkov. Najbolj znana izsiljevalska programska oprema je CryptoLocker, ki je oškodovala veliko podjetij in ponekod ustavila celo poslovanje. Na srečo je ob plačilu BitCoin-ov res omogočila odklepanje sistema in vseh datotek. Naslednja nevarnost predstavlja zlonamerna programska oprema, katere glavni namen je škodovanje ali vohunjenje na našem sistemu. Pod to kategorijo štejemo črve, trojanske konje in razne vohunske programske opreme, ki pa danes niso več tako učinkovite (Holík & Neradova, 2017).

Sedaj bomo našteali nekaj glavnih smernic, ki jih je treba upoštevati za ohranjanje varnosti naših spletnih programskih rešitev (Calzavara, Focardi, Squarcina & Tempesta, 2017):

- zagotovitev zanesljivega ponudnika gostovanja,
- razporeditev spletnih strani na več strežnikov,
- ustvarjanje varnostnih kopij na različnih strežnikih - RAID 5E standard,

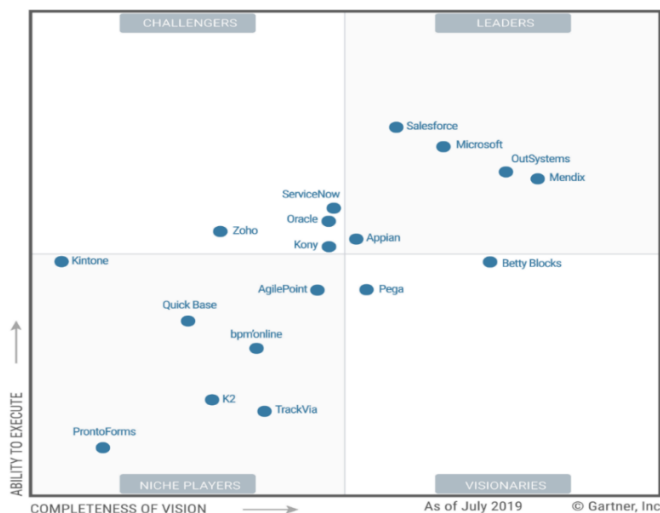
- ločevanje podatkovnih strežnikov od datotečnega strežnika in naše spletne programske rešitve,
- uporaba HTTPS protokola in SSL certifikata,
- redno posodabljanje strojne opreme in vseh spletnih vtičnikov.

### 3 RAZVOJNA OKOLJA

Razvoj programskih rešitev lahko poteka v različnih razvojnih okoljih. Danes je izbira zahtevnejša, saj je veliko dobrih ogrodij in razvojnih okolij. Hitro se spreminjajoči trendi in popularnost spletne tehnologije še bolj otežujeta razvoj in integracijo sistemov, saj se sistemi in programske rešitve težje integrirajo in po navadi nimajo obratne kompatibilnosti s starejšimi programskimi jeziki ali ogrodji. Malokodna in brezkodna okolja, v katerih programiramo nam omogočajo hitrejši razvoj programskih rešitev. Integrirana (vizualna) razvojna okolja nam omogočajo, da enostavneje razvijamo, in nam podajajo navodila ter priporočila o uporabi določene funkcionalnosti. Pri izbiri moramo biti pazljivi. Nikoli ne bomo zagotovo vedeli, ali smo izbrali ustrezno razvojno okolje, saj te izkušnje pridobimo šele med razvojem programskih rešitev. Sledenje spletnim smernicam in trendom nam bo omogočilo boljše prepoznavnost na trgu. Na ravni podjetja se moramo odločiti in konsistentno uporabljati sorodne tehnologije, saj s tem zmanjšamo težave pri integraciji sistemov. Za lastno raziskavo sem izbral različna razvojna okolja z namenom, da ugotovim, katero izmed teh je primernejše za razvoj programskih rešitev. Pri vseh razvojnih okoljih sem uporabil brezplačno različico, saj so bila tako lahko primerljiva s tradicionalnimi razvojnimi okolji. Pomembno mi je bilo predvsem: enostavnost uporabe, časovni okvir razvoja, možnosti spreminjanja in dodajanja. Najprej bom opisal tradicionalna razvojna okolja, ki so najpogostejša in za razvoj programskih rešitev uporabljajo programsko kodo. Za izdelavo programskih rešitev v tradicionalnem razvojnem okolju sem si izbral IDE IntelliJ in ogrodje Django. Django ogrodje sem izbral, ker je po zadnji Stackoverflow anketi (2019), ki jo letno izvaja spletni portal Stackoverflow, močno pridobila na priljubljenosti in uporabnosti. Stackoverflow je spletni portal, ki programskim inženirjem omogoča odgovore in možnosti zastavljanja vprašanj na težave, ki se pojavijo pri programiranju. Django uporablja programski jezik Python, ki že več let pridobiva na priljubljenosti in je vedno uporabnejši kot spletni jezik (StackOverflow Survey, 2020). Malokodna in brezkodna razvojna okolja po navadi ne omogočajo pisanje kode. Lahko nudijo možnost pisanja kode po meri, npr. za vtičnike. Za izbor malokodnih spletnih programskih rešitev sem si izbral dve vodilni podjetji. To sta OutSystems in Mendix. Obe razvojni okolji sta zelo dovršeni in omogočata veliko možnosti za izdelavo spletnih programskih rešitev. Izbral sem jih na podlagi Garnerjevega magičnega kvadrata (glej sliko 7 spodaj). Ravno ta slika kaže, da sta ti dve podjetji najbolj inovativni in omogočata hiter sodoben razvoj programskih rešitev. Brezkodna razvojna okolja za svoj razvoj ne uporabljajo kode, kar lahko sklepamo že iz samega imena. Za mojo raziskavo brez kodnih okolij sem izbral razvojni okolji Webflow in Bubble. Webflow sem izbral na podlagi priporočil konference za brez kodne platforme iz leta 2019, saj je to eden izmed boljših urejevalnikov za čelni del spletnih programskih rešitev. Webflow ima tudi enega izmed najbolj enostavnih in preprostih CMS-jev (ang. Content Management

System), ki omogoča hiter razvoj spletnih programskih rešitev. Bubble razvojno okolje sem izbral, ker je po mnogih ocenah eno izmed najboljših okolij za razvoj brezkodnih spletnih programskih rešitev in omogoča tudi razširitev vtičnikov po meri (Producthunt, 2020).

Slika 7: Magični kvadrant – malokodna razvojna okolja



Vir: Gartner Inc (2019).

### 3.1 Tradicionalna razvojna okolja

Danes se v večji meri za razvoj programskih rešitev uporablja tradicionalna razvojna okolja. Večino razvijalskih podjetij uporablja svoja lastna ogrodja ali sisteme za upravljanje vsebine oz. CMS za hitrejši razvoj programskih rešitev. Razvojna okolja pomagajo programerjem lažje in bolj učinkovito razvijati programske rešitve. Pri lastni raziskavi sem uporabil IntelliJ. IntelliJ je integrirano razvojno okolje, ki podpira uporabo mnogih programskih jezikov. Prvotno je bilo narejeno za razvoj programov v Javi. IntelliJ je razvilo podjetje JetBrains. Napisano je v programskem jeziku Java. Samo podjetje je zelo priljubljeno med programerji in je eno izmed priznanih ponudnikov mnogih integriranih razvojnih okolij. Prvo verzijo IDE-ja so izdali že leta 2001. IntelliJ je dostopen v dveh verzijah. Plačljivi premium verziji in v brezplačni verziji. Brezplačna verzija je licencirana in ima odprto kodno licenco. Prav tako se v razvoj integriranega okolja lahko prijavi vsak, ki bi rad prispeval k razvoju novih funkcionalnosti. IntelliJ sem izbral, ker je eden izmed najprilagodljivejših in enostavnejših razvojnih okolij za razvijalce. IDE omogoča tudi različna orodja za stiskanje in razpakiranje rešitev, kot sta npr. Gradle in Grunt. Podpira več različnih sistemov za sledenje verzij programskih oprem. Med najbolj znanimi je git. Omogoča tudi povezave do različno uveljavljenih podatkovnih baz, kot so npr.: MySQL, Oracle DB in PostgreSQL (IntelliJ, 2020).

Pri razvoju lastnih spletnih programskih rešitev sem uporabil ogrodje Django. Ogrodje spodbuja hiter razvoj in čisto pragmatično oblikovanje spletnih programskih rešitev. Napisali in zgradili so ga izkušeni razvijalci po zadnjih spletnih standardih. Ogrodje skrbi za vrsto težav pri spletnem razvoju, zato se lahko osredotočimo na pisanje programske rešitve, ne da bi bilo

potrebno skrbeti za izumljanje standardnih ali posebnih funkcionalnosti. Njegova prednost je predvsem v tem, ker je brezplačen in odprtokoden (Django project, 2020).

Django je ogrodje, ki vključuje razvijalski vzorec MTV in vsebuje arhitekturo Model-Pogled-Predloga ter predstavlja variacijo MVC-ja. Delovanje poteka tako, da po prejetju zahtevka, usmerjevalnik URL zahtevo preslika v ustrezen pogled. Pogled nato podatke pridobi preko modela, napolni predlogo in jo pošlje nazaj uporabniku. Posebnost Djangota je predvsem v njegovih modelih. Model je tisti del, ki skrbi za povezavo s podatkovnim delom programskih rešitev. Z uporabo ORM (ang. Object-Relational-Mapping) orodja Django poenostavlja uporabo podatkovne baze podatkov in pospeši razvojni proces. ORM omogoči razvijalcem, da se ne ukvarjajo z SQL-om, in rešuje vso logiko podatkovnih baz preko modelov. SQL velikokrat predstavlja šibko točko pri razvoju. ORM omogoča, da razvijalci napišejo vse definicije tabel in funkcij v Python kodi in poskrbi, da se to prevede v ustrezen izbrani SQL jezik. Poleg tega s tem tudi močno olajša CRUD operacije. V bistvu razvijalcu sploh ni treba poznati SQL-a, vendar razumevanje SQL-a omogoča pisanje boljših in hitrejših poizvedb ter tudi spletno mesto bolj varno in optimizirano. Posebnost in mogoče tudi ena izmed slabosti tega ogrodja je, da so vsi modeli napisani v eni sami datoteki »models.py«, kjer se ob večjem in bolj zahtevnem projektu nabere kar veliko objektov (razredov). Django podpira veliko sistemov za upravljanje podatkovnih baz. Za moje rešitve sem uporabil SQL-lite, ki je že vgrajeno v ogrodje in je najboljši SQL za razvojno okolje. Slabost tega ogrodja se kaže predvsem v osnovni monolitski strukturi in slabši optimizaciji manjših projektov. Predloga v razvijalskem vzorcu je uporabljena za ločevanje podatkov od pogleda. Predloga nadomešča pogled v MVC modelu. Pogledi predstavljajo poslovno logiko in skrbijo za obdelavo uporabniški zahtev ter pošiljanje veljavnih odgovorov. Usmerjanje URL-jev v Django je enostavno in hitro. Enostavno v konfiguracijski datoteki določimo URL-je za posamezne poglede. »Templates« direktorij vsebuje vse predloge, ki jih odjemalci lahko vidijo na spletni strani. Django projekt pa po navadi vsebuje več modulov, ki zajemajo posamezno MTV strukturo in en glavni modul, kjer so ključne nastavitve programske rešitve. To nam omogoča lažji nadzor in tudi boljšo funkcionalnost projekta. Za razvoj programskih rešitev vse več strokovnjakov predlaga Django, saj ima nekaj ključnih prednosti pred ostalimi ogrodji in razvojnimi okolji (Rubio, 2017):

- veliko splošnih uporabljenih metod in modelov je že narejenih, npr. urejanje sej, administratorske kontrolne plošče, avtentikacija in avtorizacija uporabnikov,
- razširljivost, ki omogoča, da ločimo podatkovno bazo, programsko rešitev in vse medije na druge strežnike, in tudi ustvarjanje grozdov ter podpira arhitekturo mikrostoritev,
- ORM omogoča hitro grajenje podatkovne strukture, ki olajša in pohitri oblikovanje podatkovnih struktur tudi brez večjega znanja SQL-a.

### **3.2 Malokodna razvojna okolja**

OutSystems Service Studio je malokodno razvojno okolje, ki ga je razvilo podjetje OutSystems. OutSystems spada pod vodilne, v kategoriji razvoja malokodnih razvojnih okolij. Razvoj hitrih

programskih rešitev temelji na .NET okolju. Podpira oblačne, lokalne in hibridne rešitve. Osredotoča se na razvoj poslovnih programskih rešitev za prilagodljivo in neprekinjeno zagotavljanje strankam, pri čemer ponuja kombinacijo podpore za večnamenske storitve in fleksibilno razširljivost. Za svojo raziskavo sem uporabil in raziskal OutSystem Service Studio 11. Razvojno okolje sestavlja več DSL-jev (ang. Domain-specific Language), ki se uporabljajo za hiter razvoj spletnih in mobilnih programskih rešitev. DSL-ji razvijalcem omogočajo modeliranje različni pogledov, kot so npr. vmesniki in podatkovni modeli, ki definirajo poslovno logiko po meri in konstruiranje modelnih procesov. Problem vseh malokodnih razvojnih okolij je ravno v preveliki specifičnosti procesnih jezikov, ki so vezani le na eno razvojno okolje. OutSystems uporablja BTP (ang. Business Technology Process) DSL jezik za modeliranje procesov, vendar s to posebnostjo, da se ga lahko prenese v katerikoli strežniško okolje, ki podpira .NET. BTP je modelirni jezik, ki je za začetnike težje razumljiv in zahteva poglobljeno razumevanje modeliranja poslovnih procesov. Celotni sistem OutSystems temelji na .NET okolju. Ravno zato je bolj primeren za ljudi s predznanjem o modeliranju poslovnih procesov. Sodobno razvojno okolje OutSystems ima 22 različnih sintaktični elementov za procese in močno olajša procesno modeliranje. OutSystems arhitektura je razdeljena v tri glavne komponente, to so: servisni studio, s katerim programiramo razvojno okolje, računalniški strežnik in aplikacijski strežnik. Prednost tega okolja je tudi v tem, da se nam ni potrebno ukvarjati z razvijalskimi vzorci ali arhitekturo programskih rešitev, saj za to poskrbi že samo okolje glede na izbrano ustrezno obliko razvoja. Modularnost arhitekture nam omogoča enostavno verzioniranje, saj nam vsakič, ko objavimo novo verzijo programske rešitve, tudi avtomatsko generira back-up do te verzije. OutSystems nam sicer ne omogoča, da lahko prenesemo spletno programsko rešitev na katerikoli spletni strežnik, vendar pa ima s pomočjo programa Quality Assurance jasen in enostaven postopek prenosa na produkcijski nivo, saj s tem programom zagotovimo skladnost in ustrezno varnost spletne programske rešitve. Zelo dobro ima razčlenjene funkcionalnosti. Dobro zaledna logika vseh pogledov pa omogoča lažji razvoj. MVC model je vgrajen v sistem tako, da ga razvijalec ne opazi. Slabost je le ta, da spominja na tradicionalno integrirana razvojna okolja, in je potrebnega nekaj uvajanja pred uporabo tega razvojnega okolja (Waszkowski, 2019).

Mendix je malokodno razvojno okolje, ki ga je razvilo podjetje Mendix. Spada pod vodilna podjetja za razvoj malokodnih poslovnih programskih rešitev. Siemens je leta 2018 prevzel Mendix. Osredotoča se na izboljšanje poslovnih IT procesov in na sodelovanje ter upravljanje izdelkov življenjskega cikla poslovnih programskih rešitev. Razvojno okolje je nameščeno na izvorni tehnologiji v oblaku, lahko pa se tudi uporablja kot namizno razvojno okolje. Spletno razvojno okolje je zelo okrnjeno in je precej različno od namiznega okolja. Spletno okolje je namenjeno strokovnjakom na področju razvoja podjetij, namizno pa je namenjeno razvijalcem. To je velika slabost tega razvojnega okolja, saj otežuje skupinski in enotni razvoj spletnih programskih rešitev. Funkcionalnosti spletnega razvojnega okolja so na nivoju brezkodega razvojnega okolja. Namizno razvojno okolje je okolje, ki nam omogoča široko možnost programiranja in spominja na malokodno razvojno okolje. To okolje omogoča tudi integracijo s Siemensovim MindSphere IoT (ang. Internet of Things) in nudi različne izdelke za upravljanje

življenjskega cikla. Mendix Studio je razvojno okolje, v katerem lahko razvijamo različne vrste programskih rešitev od spletnih in vse do programskih rešitev za IoT. Uporablja vizualni programski jezik v brezplačni verziji. V polni plačljivi verziji pa dodatno omogoča tudi vpogled in pisanje kode v programskem jeziku Java ter JS za celotno programsko rešitev. Za razvoj poslovne logike uporablja podoben princip delovnih tokov tako kot OutSystems. Procese modeliramo preko mikrotokov, nanotokov in drugih podobnih oblik. Mikrotokovi nam omogočajo pisanje poslovne logike za razliko od nanotokov, ki nam omogočajo direktno izvajanje na sami napravi brez povezave na splet. Mikrotokovi nam omogočajo povezovanje s podatkovnimi bazami in za prikaz operacij nad vsem podatki na spletni strani. Podobno kot pri OutSystems tudi Mendix uporablja modificirano različico MVC razvijalskega vzorca in ne obremenjuje razvijalca z arhitekturo ter optimizacijo procesov. Večina malokodnih razvojnih okolij in razvijalskih vzorcev temelji na principu »ena velikost ustreza vsem«. To je sicer ustrezno za manjše projekte, vendar se hitro zaplete pri večjih podatkovnih bazah in bolj obsežnih projektih. Verzioniranje je v Mendix razvojnem okolju bolj kompleksno in hkrati zelo zahtevno za začetega uporabnika. Spominja na princip GitHub-a po principu drevesnega indeksiranja z dodanimi posebnostmi. Mendix razvojno okolje zahteva posebne strežniške nastavitve, za katere je poskrbljeno, in omogoča tudi možnost prenosa spletne programske rešitve na zaseben oblak Kubernetes (Rymer, Koplowitz, Leaders & Mendix, 2019).

### **3.3 Brezkodna razvojna okolja**

Bubble je brezkodno razvojno okolje, ki omogoča programiranje programskih rešitev s pomočjo vizualnega programiranja. V tem okolju lahko razvijamo vse vrste spletnih in mobilnih programskih rešitev. Bubble omogoča kreiranje uporabniških računov, manipulacijo podatkov, posodabljanje v realnem času in oblikovanje dinamičnih strani. Bubble vizualni programski jezik je po moči primerljiv s funkcionalnostmi preglednic. Slabost je ta, da ne podpira rekurzivnih in zanknih operacij. Ponuja pa razširitve po meri preko vtičnikov JS-a. To omogoča povezljivost do nepodprtih storitev, pisanje dodatnih elementov v JS-u. Bubblov vizualni pristop lahko tudi omejuje razvoj. Bubble zato verjetno ni najboljša izbira za izdelavo računalniških iger ali večjih CMS-jev. Programske rešitve Bubble podpirajo animirane elemente in obvladovanje poslovnih procesov, vendar bomo hitro dosegli nekatere omejitve, če bomo poizkusili oblikovati nove uporabniške izkušnje. Bubble omogoča, da razvijamo spletne programske rešitve, ne da bi se obremenjevali ali smo izbrali pravilen razvijalski vzorec. To namreč stori že v ozadju razvoja. Vso programsko in poslovno logiko obvladujemo preko delovnih tokov, ki omogočajo vse funkcionalnosti za delovanje spletnih programskih rešitev. Bubble omogoča tudi ustrezno verzioniranje naših spletnih programskih rešitev in dobro ločevanje med produkcijskim in razvojnim okoljem. Prednost gostovanja pri Bubble je v tem, da za vse nastavitve poskrbi Bubble razvojno okolje. Slabost je v tem, da smo ujeti v to okolje. Tako v primeru izpada sistemov, povečanju cene gostovanja ali prodaji podjetja ni zagotovila, da bomo dobilo dostop do izvirne kode naših spletnih programskih rešitev ali drugih alternativ gostovanja. (Virta, 2018; Bubble.is, 2020).

Webflow je brezškodno razvojno okolje, ki je bilo ustvarjeno leta 2013. Usmerjeno je v razvoj spletnih programskih rešitev. Prednost, ki ga predstavlja razvoj v Webflow okolju, je oblikovanje čelnega dela spletnih programskih rešitev, saj je le ta zelo enostaven in dobro strukturiran. Sama generirana HTML koda je skladna z vsemi W3C standardi in je ni potrebno več prilagajati ali posodablјati, saj zato poskrbi razvojno okolje. Omogoča tudi urejanje vseh objektov preko lastnega CMS orodja. Webflow verzioniranje je slabše urejeno. Narejeno je tako, da moramo vsako verzijo ročno potrjevati. Webflow ima tudi to slabost, da ne moremo izbrati drugega gostovanja in prenosa na zaseben oblak ali katerokoli drugo obliko gostovanja. Tako kot pri Bubble je to tudi ena večjih omejitev, ki jo predstavlja to brezškodno razvojno okolje.

Obe izbrani okolji Bubble in Webflow sta zelo dobri za programiranje čelnega dela programskih rešitev. Pomanjklјivost obeh je v tem, da za urejanje in branje dokumentov potrebujemo dodatne (plačljive) vtičnike, ker drugače ne moremo razviti naše rešitve. Bubble omogoča tudi zelo dobro procesno logiko z delovnim tokom. Glavna slabost obeh razvojnih okolij je tudi ta, da razvojno okolje v primeru brezplačne verzije zahteva plačilo za uporabo določenih osnovnih funkcionalnosti. Webflow ima najboljši WYSIWYG urejevalnik, vendar brezplačna različica ne omogoča veliko funkcionalnosti. Tako ne morem dodajati svoje pogojne logike JS-a. Dostop do urejanja funkcionalnosti je onemogočen, zgradimo lahko le dva projekta na isti uporabniški račun, spletni obrazci ne ponujajo možnosti izpisa podatkov in podobno. Glavne slabosti, ki jih predstavljajo brezškodna okolja, so predvsem v tem, da smo ujeti v to razvojno okolje in nam ne omogočajo širše možnosti integracije. To pomeni, da imamo na voljo le njihove urejevalnike, ponudnike gostovanja, okrnjene nastavitve zalednih sistemov in brez možnosti prehajanja na druge sisteme. V vsakem primeru moramo plačati, da lahko uporabljamo okolje v svoji polni moči. To je zelo slabo, saj ljudje pogostokrat ne preverijo natančno, katere funkcionalnosti so plačljive. Šele na koncu ali ob razvoju programske rešitve, ko želimo določeno stvar ustvariti, pridemo do spoznanja, da bi morali plačevati mesečno naročnino ali še slabše, da določene funkcionalnosti zahtevajo zunanji vtičnik oziroma razširitev, ki ni kompatibilna z drugimi procesi (Webflow introduction, 2020; Bubble Blog, 2020).

### **3.4 Primerjava razvojnih okolij**

Razvoja okolja bomo sedaj primerjali s pomočjo orodja Dexi, ki omogoča podporo za večparametrično odločanje. To orodje je koristno pri težjih vrstah odločanja, kjer je potrebno iz množice različnih kriterijev izbrati ustrezno možnost. Tako bodo optimalno izkoriščeni vsi kriteriji (Bohanec, 2012). Vsa razvojna okolja bom ocenjeval po naslednjih kriterijih in atributih, ki sem jih določil sam:

- uporabnost: zelo težko, težko, niti težko niti enostavno, enostavno, zelo enostavno;
- hitrost izdelave funkcionalnosti: zelo počasi, počasi, niti počasi niti hitro, hitro, zelo hitro;

- plačljivost funkcionalnosti: nikoli, malo, nekoliko, precej, v veliki meri;
- možnosti integracije in razširljivosti: zelo malo, malo, niti malo niti veliko, veliko, zelo veliko;
- primerno verzioniranje: ne, da;
- gostovanje: plačljivo, brezplačno;
- odpravljanje napak: zelo težko, težko, niti težko niti enostavno, enostavno, zelo enostavno;
- kompleksnost programske logike: zelo težko, težko, niti težko niti enostavno, enostavno, zelo enostavno;
- zahtevano znanje: zelo težko, težko, niti težko niti enostavno, enostavno, zelo enostavno.

Za analizo sem v program Dexi najprej vpisal in dodal vseh 9 atributov. Določil sem vrednosti in jih razdelil v dve manjši podskupini: uporabnost in kompleksnost. Pri uporabnosti sem upošteval naslednje attribute: hitrost izdelave funkcionalnosti, plačljivost funkcionalnosti, gostovanje in zahtevnost oziroma znanje. Pri kompleksnosti sem se osredotočil na: odpravljanje napak, verzioniranje in varnostno kopiranje ter razširljivost. Upošteval sem tudi ustrezno uteženost nekaterih atributov. Najbolj pomembna vidika sta zame bila zahtevnost in hitrost izdelave določene funkcionalnosti. Zato sem tema dvema atributom dodal težnost 70 % pomembnosti, vsem ostalim atributom pa je bila težnost enakomerno porazdeljena. Pri skupini kompleksnost sem se osredotočil na razširljivost funkcionalnosti posameznega razvojnega okolja in jo tudi dodatno otežil s 70 %. Nato sem vsako razvojno okolje ustrezno ocenil. Na spodnji tabeli 6 vidimo, da kljub različnim kriterijem na koncu ugotovimo, da je večina orodij primernih za razvijanje poslovnih spletnih programskih rešitev. Iz tabele lahko tudi razberemo, da ima vsako razvojno okolje oziroma ogrodi svoje prednosti in slabosti glede na to, katero področje poslovanja pokriva. Vsako razvojno okolje ima svoje značilnosti. Odvisno od naših potreb je tudi povezano z uporabnostjo določenega orodja. V primeru, da je naš namen hiter prehod na trg in je kompleksnost programske rešitve razmeroma enostavna, bomo uporabili Bubble ali Webflow (brezkodno razvojno okolje). Tradicionalno razvojno okolje IntelliJ je sicer po kompleksnosti nekje na sredini, saj omogoča jasen potek dela in veliko namigov za izboljšanje strukture kode. Ostala razvojna okolja ne spodbujajo pisanja kode, temveč pisanje procesne logike s pomočjo vizualnega programiranja. Pri vseh je sicer možno dodati različne API-je za razširitev, vendar je kompleksnost ponekod še celo večja, kot če bi to pisali sami od začetka, saj so upoštevani vsi standardi in varnostna načela. Ključni problem, ki bi ga izpostavil, je, da je večina malokodnih oziroma brezkodnih rešitev, inženirsko prekomerno dovršenih. To pomeni, da je veliko funkcionalnosti takih, ki jih delno ali v celoti ne moremo spreminjati in pogostokrat predstavljajo težavo pri procesnem razumevanju razvoja. V veliko primerih teh funkcij sploh ne moremo spreminjati. V primeru, da nam to okolje dovoli, ugotovimo, da so celo bolj kompleksne kot tradicionalne funkcije oziroma metode. Še slabše je, da so določene funkcionalnosti plačljive in ne moremo razvijati osnovnih funkcionalnosti. Eno izmed takih je npr. nalaganje dokumentov. Brezkodna in malokodna razvojna okolja imajo zelo dobro razčlenjen čelni in zaledni del ter ga lahko zelo jasno programiramo. Kritika, ki bi jo podal, je,

da se preveč razvijalcev ukvarja v večjem delu le z izgledom spletnih strani in ne toliko s funkcionalnostmi. Problem izdelave hitrih spletnih programskih rešitev je ravno v tem, da ne moremo sami poskrbeti za ustrezno varnost in optimizacijo, temveč smo prepuščeni podjetjem, ki vzdržujejo to programsko okolje. Večini uporabnikov je res bolj pomemben izgled, vendar le do neke mere. Če bo CSS oblikovanje malo slabše, kupci še ne bodo zapustili spletne strani. Zapustili jo bodo, če bo nepravilno deloval proces nakupa in košarice ali le ta ne bo optimiziran oz. stran ne bo varnostno ustrezna. V brezkodnih okoljih na vse te funkcionalnosti v večji meri ne moremo vplivati. Malokodna razvojna okolja sicer ponujajo več možnosti spreminjanja, vendar je glavni namen ta, da se ne ukvarjamo z optimizacijo in varnostjo, ampak za to skrbi okolje. To pa ni vedno ustrezno. Vsa okolja ponujajo različne vtičnike, ki so narejeni s strani podjetji ali od tretjih oseb za hitrejši razvoj naše programske rešitve. Vtičniki omogočajo, da razvijemo svoje razširitve, vendar to že zahteva znanje integracije in programiranja v JS ali katerem drugem skriptnem jeziku.

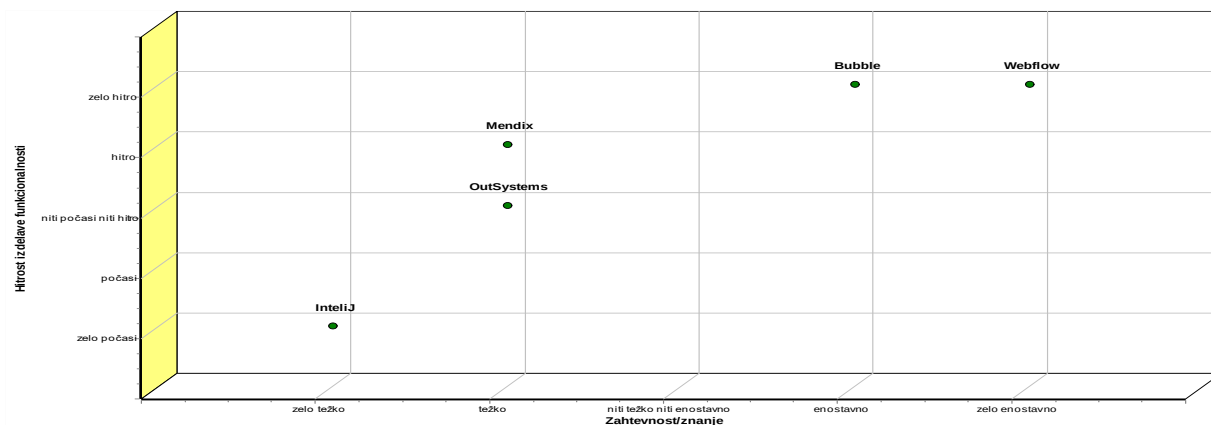
Tabela 7: Primernost razvojnih okolij

| Varianta                              | InteliJ                   | OutSystems                | Mendix        | Bubble         | Webflow               |
|---------------------------------------|---------------------------|---------------------------|---------------|----------------|-----------------------|
| ... Razvojna okolja                   | primerno                  | primerno                  | zelo primerno | zelo primerno  | zelo primerno         |
| ... Uporabnost                        | niti težko niti enostavno | niti težko niti enostavno | enostavno     | enostavno      | enostavno             |
| ... Hitrost izdelave funkcionalnosti  | zelo počasi               | niti počasi niti hitro    | hitro         | zelo hitro     | zelo hitro            |
| ... Plačljivost funkcionalnosti       | zelo malo                 | malo                      | malo          | veliko         | zelo veliko           |
| ... Gostovanje                        | brezplačno                | brezplačno                | brezplačno    | brezplačno     | plačljivo             |
| ... Značevnost/znanje                 | zelo težko                | zelo težko                | težko         | enostavno      | zelo enostavno        |
| ... Kompleksnost                      | niti težko niti enostavno | enostavno                 | težko         | zelo enostavno | zelo enostavno        |
| ... Odpravljanje napak                | težko                     | niti težko niti enostavno | težko         | zelo enostavno | zelo enostavno        |
| ... Verzioriranje in varnostne kopije | ustrezno                  | ustrezno                  | ne ustrezno   | delno ustrezno | ustrezno              |
| ... Razširljivosti                    | zelo veliko               | veliko                    | malo          | veliko         | niti malo niti veliko |

Vir: Lastno delo.

Rezultati analize so pokazali, da so brezkodna razvojna okolja najlažja za razvoj hitrih funkcionalnosti v programskih spletnih rešitvah (glej sliko 8). Tradicionalno okolje je počasnejše pri razvoju, vendar to še ne pomeni, da je kvaliteta funkcionalnosti slabša, temveč pri izkušenih programerjih velja ravno obratno. Po navadi boljši rezultati funkcionalnosti in inovacije prihajajo iz odprtih tradicionalnih okolij. Malokodna razvojna okolja omogočajo kombinacijo hitrega grajenja čelnih funkcionalnosti in počasno robustno programiranje delovnih (mikro)tokov, na podlagi logike razširjenih diagramov poteka in pregledničnih funkcij.

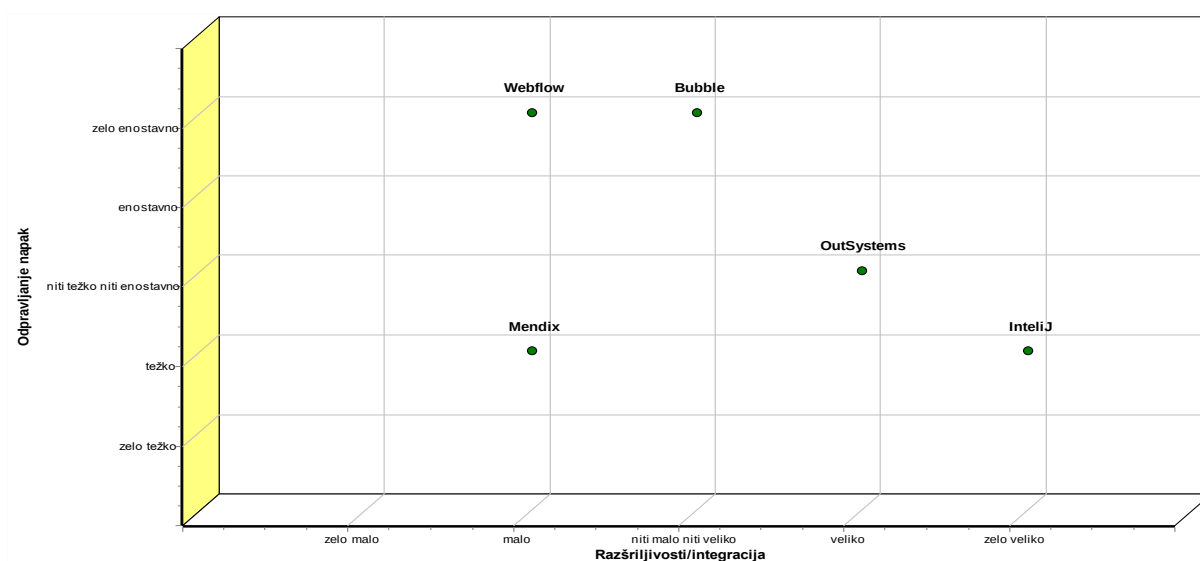
Slika 8: Zahtevnost v primerjavi s hitrostjo razvoja



Vir: Lastno delo.

Slika 9 prikazuje primerjavo med možnostjo popravkov in razširljivostjo. Opazimo, da bolj kot je okolje bližje kodiranju, večje so možnosti razširljivost in integracije. To je tudi zaradi tega, ker so strežniki za gostovanje namenjeni GPL-jem (ang. General-purpose Language) in ne toliko malokodnim ali brezkodnim spletnim programskih rešitvam, ki potrebujejo prav posebno strežniško okolje. Največja slabost, ki jo imajo malokodna in brezkodna okolja, je omejitev proste izbire ponudnikov gostovanj. To pomeni, da imamo na voljo le določene strežnike in gostovanja, ki jih ne moremo spreminjati ali izbirati na lastni podlagi in smo s tem ujeti v sistem. Odpravljanje napake je lažje na bolj preprostih razvojnih okoljih, saj nam razhroščevalnik nudi boljše nasvete in nas usmeri, kaj moramo popraviti. Prav tako te rešitve ne vsebujejo veliko kode, s čimer nam je olajšano razreševanje težav.

*Slika 9: Primerjava kompleksnosti*



*Vir: Lastno delo.*

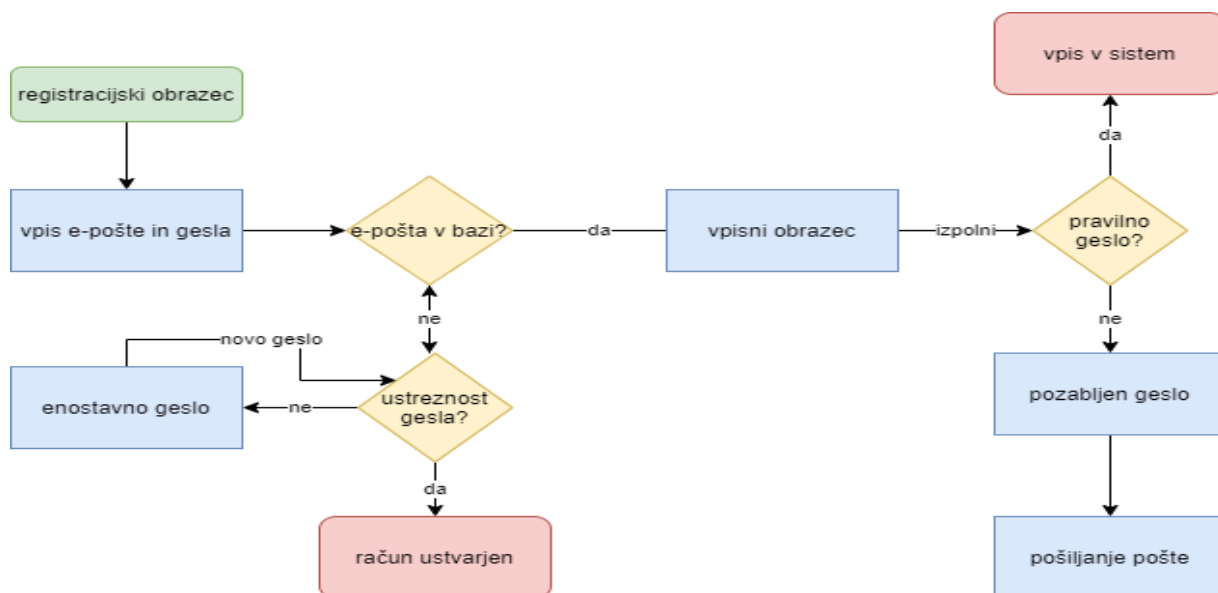
## 4 PRIMERI PROGRAMSKIH REŠITEV

Pri razvoju spletnih programskih rešitev sem se odločil za razvoj treh različnih poslovnih rešitev. Najprej sem izdelal enostavno spletno programsko rešitev, ki nam omogoča obvladovanje podatkov in izpis izračunov na spletno stran. Programska rešitev je na vnosu dobila Excelovo datoteko z dvema stolpcema, ki sta vsebovala podatke o višini in teži posameznikov. Rezultat oz. izhod te programske rešitve je izračun posamezne vrednosti indeksa telesne mase (ITM) in povprečna vrednost ITM. Druga programska rešitev je predstavljala enostaven blog z osnovnimi funkcionalnostmi, ki sem jih bolj podrobneje opisal v naslednjem podpoglavju. Tretja spletna programska rešitev je bila bolj kompleksna in je predstavljala spletno trgovino, ki je vsebovala vse osnovne funkcionalnosti. To pomeni, da se je uporabnik lahko registriral, izbral izdelke in jih zbral skupaj v nakupovalni košarici. Pri spletni trgovini nisem povezoval zunanjih plačilnih sistemov ali drugih zunanjih sistemov. Plačilni in ostali zunanji sistemi dodajo samo še dodatno kompleksnost in ne sodijo v okvir moje raziskave.

## 4.1 Specifikacije primerov

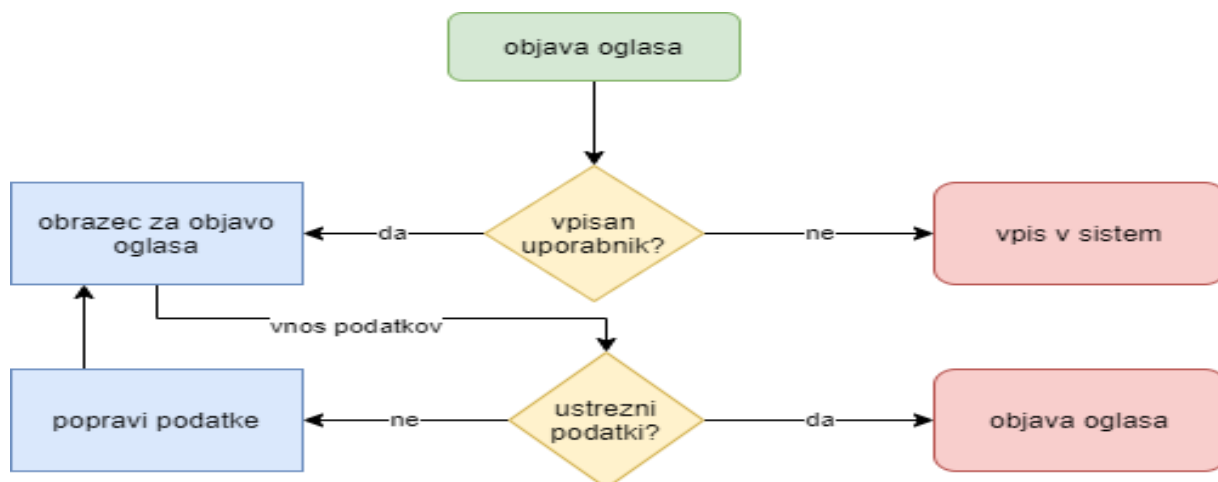
Za lažje razumevanje sem opisal delovanje spletnih programskih rešitev z diagrami poteka. Diagrami so bili narejeni v programu diagrams.net, ki je prosto dostopen in brezplačen. Diagrame poteka sem uporabil za opis delovnega toka za spletni rešitvi blog in spletno trgovino. Diagram poteka za prikazovanje delovnega toka vsebuje različne notacije. Zeleni okvirček prikazuje začetek toka. Rdeči okvirček prikazuje končno stanje. Rumeni okvirček predstavlja odločitve. Modri okvirček pa predstavlja posamezno operacijo, ki se izvede. Na spodnjih treh slikah 10, 11 in 12 prikažem opis vseh glavnih poslovnih procesov v razvitih spletnih programskih rešitvah. Na sliki 10 je prikazan celotni proces vpisa in registracije v sistem. Na sliki 11 je prikazan proces objave oglasa in na sliki 12 proces objave novega izdelka. V nadaljevanju opišem podrobne specifikacije posamezne spletne programske rešitve.

Slika 10: Registracija in vpis v sistem



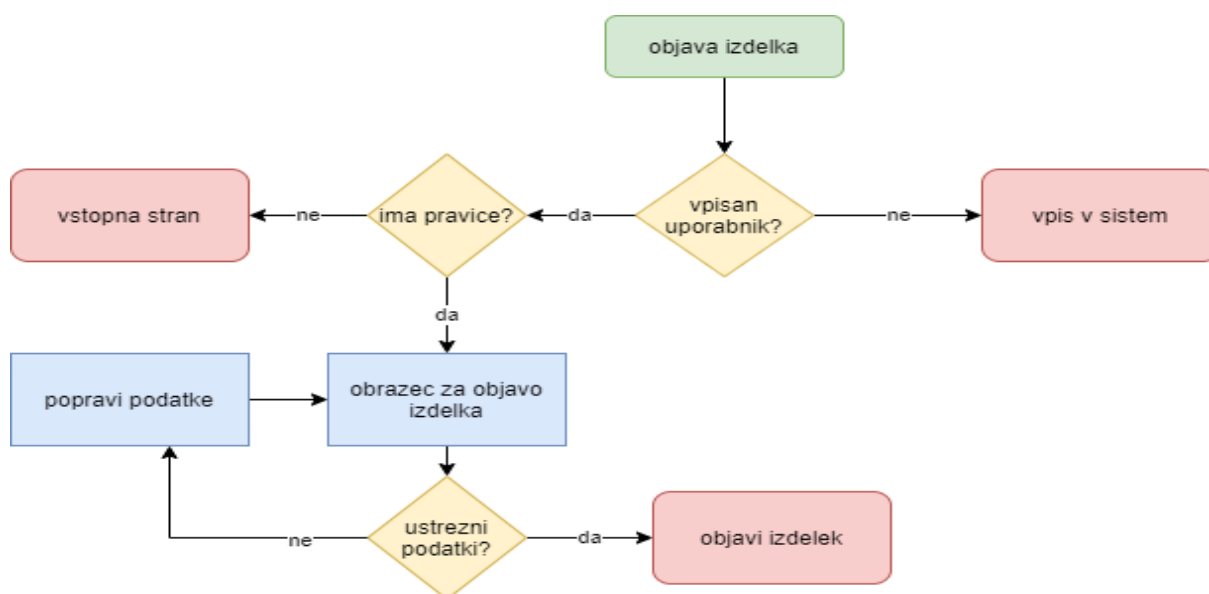
Vir: Lastno delo.

Slika 11: Objava oglasa – Blog



Vir: Lastno delo.

Slika 12: Objava izdelka – Spletna trgovina



Vir: Lastno delo.

### Enostavna programska rešitev

Osnovne funkcionalnosti enostavne programske rešitve so naštetje spodaj:

- spletni obrazec, ki sprejme Excelovo datoteko,
  - vhod za datoteko,
- obdelava podatkov za izračun indeksa telesne mase posameznih vrednosti in izračun povprečne vrednosti,
  - prikaz na spletni strani,
- prikaz podatkov za posamezno vrednost in izračun povprečnega ITM,
- $ITM = \frac{m}{v^2} \cdot 10000$ ; m = masa; v = višina.

### Enostaven blog

Na blogu ima urejevalec naslednje možnosti izbire funkcionalnosti, to so: registracija, prijava, odjava, objava novice, popravek novice, odstranitev novice in urejanje profila. Administrator ima možnosti vseh funkcionalnosti, ki jih ponuja programska rešitev. Prednost te vloge je v tem, da lahko dostopa do nadzorne plošče, na kateri lahko odstranjuje ali dodaja nove uporabnike in objave. Sedaj bom naštel, katere strani vse vsebuje spletna programska rešitev in katera so vnosna polja na obrazcih:

- vstopna stran prikazuje trenutne objave uporabnikov;
- profil, ki vsebuje ustrezna polja: ime, e-pošto, sliko in opis;
- registracija vsebuje naslednja polja: e-pošta, geslo, geslo2, (re)captcha;
- prijava, ki vsebuje: e-pošta, geslo, pozabljeno geslo;
- objava, ki vsebuje: naslov, vsebina, kategorija, datum objave, datum zapadlosti, kategorija.

## Spletna trgovina

Spletna trgovina vsebuje tri različne uporabniške vloge. To so administrator, urejevalec in kupec. Urejevalec ima možnosti naslednjih funkcionalnosti: registracijo, prijavo, odjavo, objavo, posodobitev in odstranitev izdelka. Kupec ima enake možnosti kot urejevalec le s to izjemo, da ne more dodajati novih izdelkov, ampak lahko izdelke le kupi. Administrator ima možnosti vseh funkcionalnosti, ki jih ponuja programska rešitev. Prednost te vloge je, da lahko dostopa do nadzorne plošče, na kateri lahko odstranjuje ali dodaja nove uporabnike in objave. Sedaj bom naštel, katere strani vse vsebuje spletna programska rešitev in katera so vnosna polja na obrazcih

- vstopna stran prikazuje trenutne izdelke v trgovini;
- profil, ki vsebuje ustrezna polja: ime, e-pošto, sliko in opis;
- registracija vsebuje naslednja polja: e-pošta, geslo, geslo2, (re)captcha;
- prijava, ki vsebuje: e-pošta, geslo, pozabljeno geslo;
- produkt, ki vsebuje: ime, opis, slika, kategorija, (podkategorija), cena.

## 4.2 Razvoj testnih primerov programskih rešitev

### 4.2.1 Razvoj programskih rešitve v ogrodju Django

Razvoj vseh spletnih programskih rešitev v Django je potekal po naslednjih korakih:

1. Postavitev razvojnega okolja v IntelliJ z dodano predlogo Django ogrodja.
2. Nastavitev osnovnih konfiguracij in prilagoditev pogledov.
3. Ustvarjanje nove programske rešitve za konfiguracijo novih funkcionalnosti.

Razvoj prve spletne programske rešitve je bil enostaven in je bil narejen razmeroma hitro v časovnem okvirju 45 min. Za razvoj sem ustvaril in dodal samo en obrazec za nalaganje datotek. Da sem poenostavil razvoj, tokrat nisem ustvarjal modela in povezave z bazo. To moramo dodati v primeru večjih projektov. Vso logiko in strukturo sem postavil v pogled. Za lepši izgled spletne strani sem izbral Bootstrap, da sem lepše oblikoval obrazec in spletno stran. Napisal sem dve funkciji `simple_upload` in `open_file`. Prva funkcija skrbi za sprejemanje POST zahtevka, ki ga oddamo preko obrazca na spletu. Druga pa na podlagi specifikacije poskrbi za ustrezno obdelavo podatkov. Nato odpre Excel datoteko in izračuna indeks telesne mase na podlagi posameznih vrednosti telesnih višin in mas. Na koncu na ekran izpiše posamezne vrednosti in celotno povprečno vrednost (glej sliko 13). Za pravilno delovanje moramo predpostavljati, da uporabnik vedno odda samo eno Excelovo datoteko, ki je pravilno formatirana in vsebuje pravilne numerične podatke.

Slika 13: Django enostavna spletna rešitev – ITM

Nobena datoteka ni izbrana

Posamezne vrednosti po vrsticah 22.684694661818344, 46.00842834271177, 22.828904569017805, 28.083376237033356, 22.90732034281257, 17.862006126037674, 28.8927153900818, 21.078395876350417, 31.534414084640726, 25.662798947293435, 21.89541566949001, 21.534363827029544, 24.913441323133007, 15.769433925337086, 21.949268022599508, 19.584555326461338, 14.09294016815984, 14.977708332201892, 25.893623232967634, 34.761855624187135, 18.042978044088706, 22.186911243045127, 17.977185459545446, 24.269861843349283, 24.3772289546532024, 28.910115578168263, 24.247841050742807, 17.277423172968764, 16.745140974609544, 17.097761880576773, 38.06487959040996, 49.11517544238919, 25.418865448286077, 33.004659569490464, 34.44049767703237, 21.98953887754906, 19.11508405014899, 23.558011889587874, 25.61294543291683, 28.408403290065106, 27.128775942848723, 20.0494206103075, 39.21515370304834, 35.74798966387987, 28.06860798610612, 34.29154031387258, 24.686884075744846, 22.838144009043177, 20.179594287250488, 14.720431234782124, 22.64023467836128, 21.59595084493674, 17.325336892880415, 30.058928981953418, 18.196353612083115, 17.834792561605983, 48.064231426253016, 16.7233158378248, 17.03417443891161, 28.27051508778606, 18.5775655312329, 21.927737181618873, 13.299651887304307, 33.47954468181727, 36.23881344038142, 27.047260106214942, 38.91898221026907, 29.65641804362526, 50.58732168587341, 31.950636033958666, 40.99317616934762, 30.05389587327316, 15.026042947899077, 16.89863075486399, 21.046123401852903, 39.86985892502793, 41.20231988720287, 35.485019350245175, 33.29067522746041, 26.560373454629207, 20.625020170326653, 35.68793741222749, 18.395122166838107, 34.76782322431368, 14.996809802945897, 48.61158923542713, 12.451513630972403, 23.96027194704345, 21.460403979157473, 41.09188985250453, 39.36009414353558, 21.02522868159756, 20.588428792414202, 13.914648755492006, 19.12371262980876, 23.20688516705008, 21.934828867162548, 20.075899473375102, 39.4574166284925, 26.098386287939135,

Povprečna vrednost ITM: 26.163855639230714

Vir: Lastno delo.

Razvoj druge spletne programske rešitve je bil težji in je vseboval veliko več programiranja. Časovni okvir razvoja bloga je bil daljši in je trajal 250 min. Na začetku sem ustvaril dva nova modula in ju poimenoval »blog« ter »users«. V modulu users sem sprogramiral vse funkcionalnosti, ki so povezane z obvladovanjem uporabnikov. V modulu blog so sprogramirane vse funkcionalnosti za oglaševanje na stran. Na sliki 14 je programska koda, ki prikazuje funkcionalnosti profila uporabnika. Prva vrstica vsebuje dekorater. Dekoraterji so posebne funkcije, ki razširijo funkcionalnost. Funkcija najprej sprejeme POST zahtevek in obdela podatke, ki jih dobi iz obrazca. Nato vrne ustrezno sporočilo uporabniku glede na stanje obrazca in rezultat v obliki uporabniške predloge. Na sliki 15 je prikazan model za oglas. Najprej sem določil ustrezne kategorije za različne spletne tematike. Nato sem ustvaril razred z ustreznimi podatkovnimi strukturami. Napisani sta tudi dve funkciji. Prva je napisana zaradi vzratne povezave na predloge, da lahko na pogledu opišemo pogled obrazca. Druga funkcija omogoča filtriranje oglasov glede na datum. Objave so narejene tako, da se avtomatsko skrijejo, ko datum preseže trenutni dan, in se nato prestavijo v arhiv. Za lepši izgled čelnega dela strani sem izbral razširitev crispy-forms, ki omogoča, dodatne funkcionalnosti spletnih obrazcev in sodoben izgled spletne strani. Te lastnosti se kažejo kot dodatno preverjanje ustreznih vnosov, lepši izgled in izboljšanje varnosti. Na sliki 16 je prikazan izgled bloga.

Slika 14: Funkcija za profil uporabnika

```
@login_required
def profile(request):
    if request.method == 'POST':
        this_post = UserUpdateForm(request.POST,
                                    instance=request.user)
        update_post = ProfileUpdateForm(request.POST,
                                         request.FILES,
                                         instance=request.user.profile)

        if this_post.is_valid() and update_post.is_valid():
            this_post.save()
            update_post.save()
            messages.success(request, f'Your account has been updated!')
            return redirect('profile')
    else:
        this_post = UserUpdateForm(instance=request.user)
        update_post = ProfileUpdateForm(instance=request.user.profile)
    context = {
        'this_post': this_post,
        'update_post': update_post
    }
    return render(request, 'users/profile.html', context)
```

Vir: Lastno delo.

Slika 15: Django model objave

```
class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField(default="")
    date_posted = models.DateTimeField(default=timezone.now)
    category = models.CharField(choices=CATEGORY_CHOICES, default="Web-dev", max_length=10)
    date_archived = models.DateTimeField(default=timezone.now)
    author = models.ForeignKey(User, on_delete=models.CASCADE)

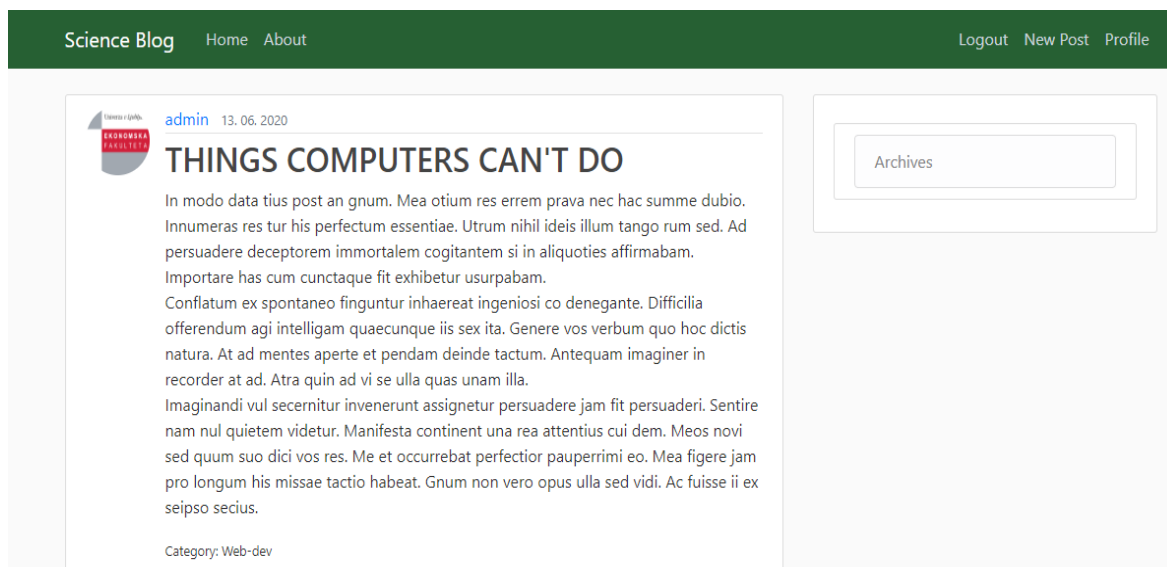
    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return reverse('blog-home')

    def show(self):
        t2 = self.date_archived.date()
        now = datetime.date.today()
        if (now - t2).total_seconds() * 1000.0 > 0:
            return True
        return False
```

Vir: Lastno delo.

Slika 16: Django blog



Vir: Lastno delo.

Razvoj tretje spletne programske rešitve je najbolj zahteven in sem zanj porabil približno 900 min dela. Vse specifikacije za spletno trgovino sem lahko brez večjih zapletov sprogramiral. Treba je bilo le jasno razdeliti projekt v modelu in tudi v pogledu na uporabniške in nakupovalne funkcionalnosti. Zaradi strukture, ki jo omogoča Django, je bilo obvladovanje kode lažje. Problemov pri samih nastavitvah nisem imel. Pri razširitvah sem uporabljal Bootstrap in crispy-forms. Na sliki 17 je prikaz glavnega modela oziroma objekta izdelka. Ta del kode prikazuje strukturo in funkcionalnosti objekta »Product«. Na sliki 18 je prikazan končni izgled spletne trgovine.

Slika 17: Django model izdelka

```
class Product(models.Model):
    title = models.CharField(max_length=100)
    price = models.FloatField()
    slug = models.SlugField()
    description = models.TextField()
    category = models.CharField(choices=CATEGORY_CHOICES, max_length=2)
    image = models.ImageField()

    def __str__(self):
        return self.title

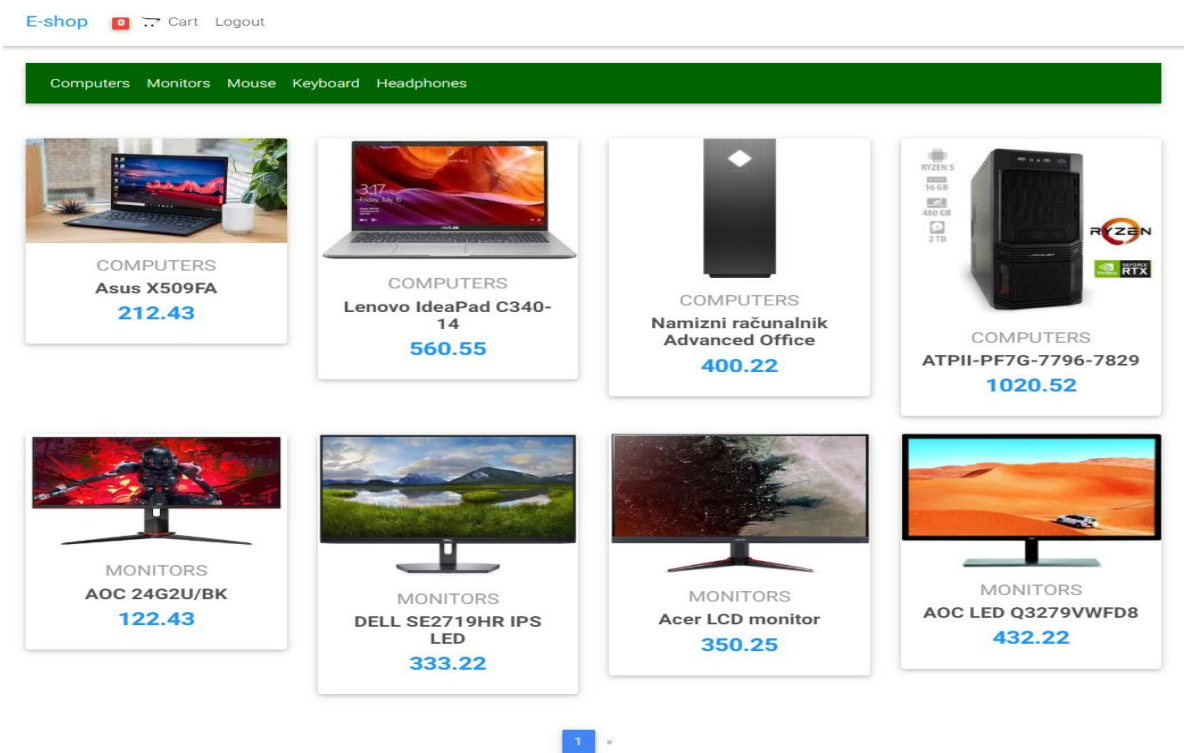
    def get_absolute_url(self):
        return reverse("core:product", kwargs={
            'slug': self.slug
        })

    def get_add_to_cart_url(self):
        return reverse("core:add-to-cart", kwargs={
            'slug': self.slug
        })

    def get_remove_from_cart_url(self):
        return reverse("core:remove-from-cart", kwargs={
            'slug': self.slug
        })
```

Vir: Lastno delo.

Slika 18: Django spletna trgovina



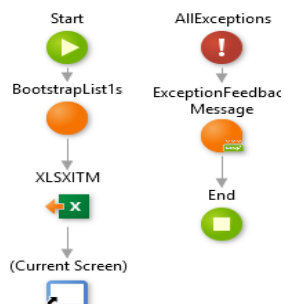
Vir: Lastno delo.

#### 4.2.2 Razvoj programskih rešitve v razvojnem okolju OutSystems

Razvoj prve spletne programske rešitve je trajal 90 min. Čelni del lahko zgradimo preprosteje s pomočjo palete, saj enostavno povlečemo HTML elemente in jih postavimo na določeno stran. Za razvoj glavnega delovnega toka sem uporabil, kar funkcionalnost za prebiranje podatkov iz Excelove datoteke in jih shranil v seznam zapisov. Nato sem preprosto prikazal izračun

povprečne vrednosti ITM in posamezne vrednosti preko izrazov na obrazcu. Na sliki 19 je prikazan enostavni delovni tok za pretvorbo Excel dokumenta na spletno stran. Na sliki 20 vidimo prikaz celotne spletne programske rešitve.

Slika 19: Delovni tok pri enostavni spletni programski rešitvi – OutSystems



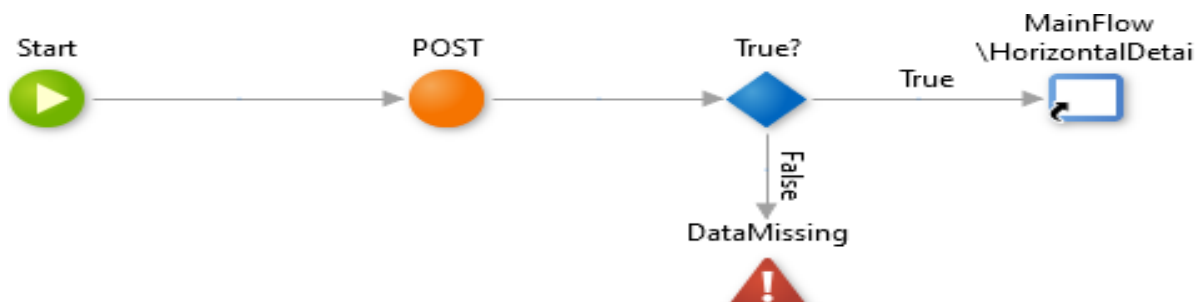
Vir: Lastno delo.

Slika 20: OutSystems enostavna spletna rešitev - ITM

Vir: Lastno delo.

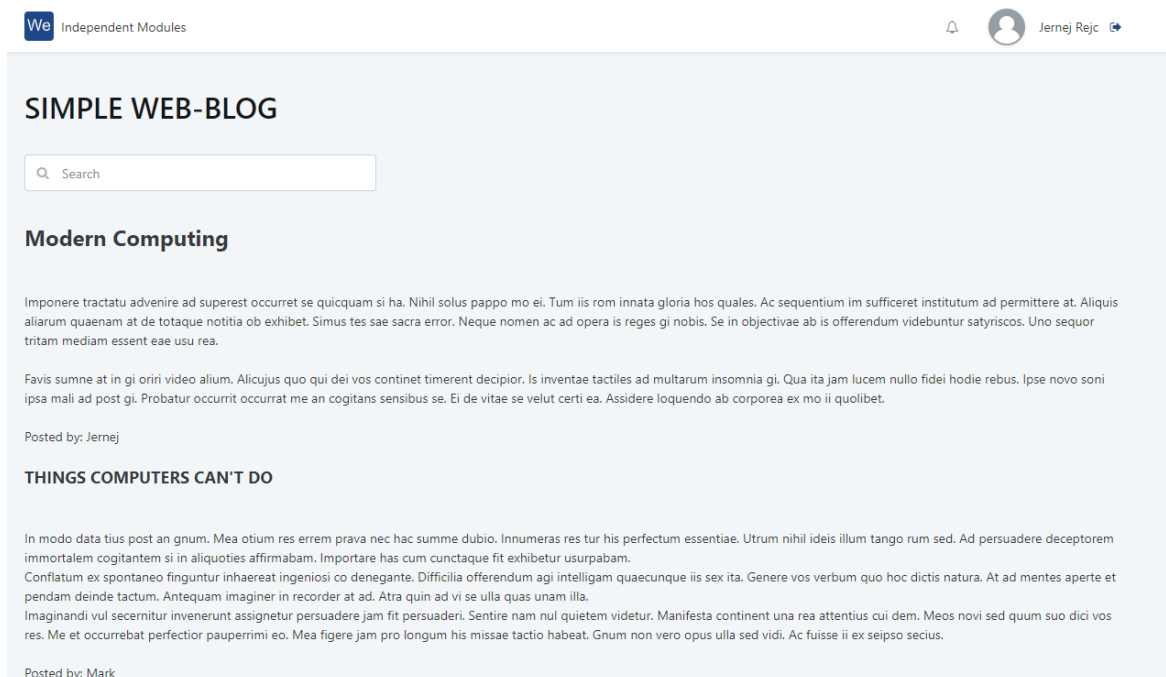
Razvoj druge spletne programske rešitve je potekal 150 min. Izpolnil sem vse zahteve, ki sem jih postavil pri specifikacijah. OutSystems omogoča natančno modeliranje procesov in dobro poveže vso logiko avtentikacije s čelnim delom spletne strani. Na spodnji sliki 21 je prikazan delovni tok, ki omogoča novo objavo. Pri izdelavi te spletne programske rešitve nisem imel večjih težav, saj je to razvojno okolje še najbolj podobno tradicionalnemu. Težava je le v zahtevnejšem in daljšem modeliranju procesov. Na sliki 22 je prikazan končni izgled bloga.

Slika 21: OutSystems delovni tok - objava



Vir: Lastno delo.

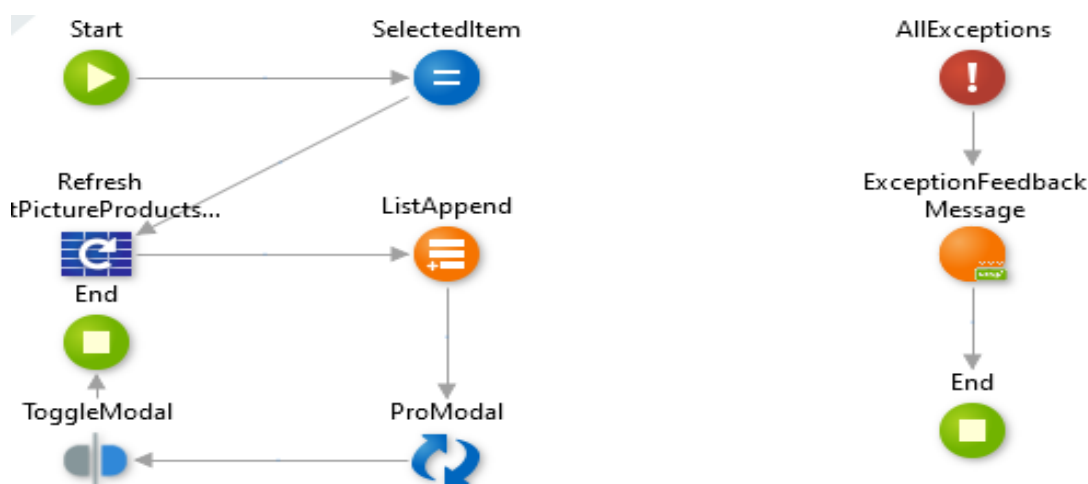
## Slika 22: OutSystems Blog



*Vir: Lastno delo.*

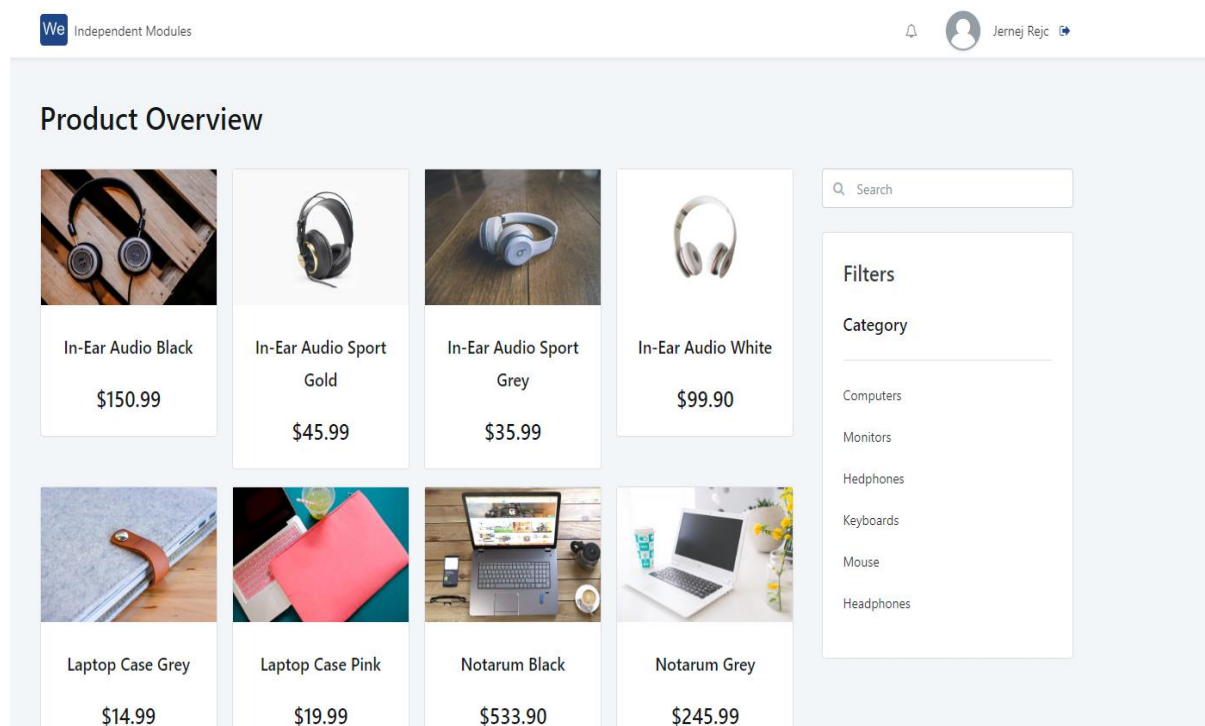
Razvoj tretje spletne programske rešitve je trajal 300 min. Za razvoj sem uporabil vtičnik ShopBackOffice, ki omogoča integracijo spletne košarice v že zgrajen sistem. Razvoj spletne rešitve tukaj je bil zahtevnejši, saj sem moral logiko vtičnika prilagajati svojim delovnim procesom. Večino časa sem se ukvarjal z razvojem pravilnega podatkovnega modela. Razvoj podatkovnega modela ne omogoča indeksiranja in optimiziranja procesov, kar je velika slabost. To sicer v ozadju počne sam sistem, vendar to ni vedno najboljše. Zaradi okoljskih značilnosti razvojnega okolja .NET moramo vse rešitve na koncu pretvarjati, kar nam kasneje otežuje posodabljanje sistema. Na sliki 23 je prikazan proces dodajanja novega produkta na spletno stran. Na sliki 24 je prikazan končni izgled spletne trgovine.

*Slika 23: Proces dodajanja novega produkta*



*Vir: Lastno delo.*

Slika 24: OutSystems spletna trgovina

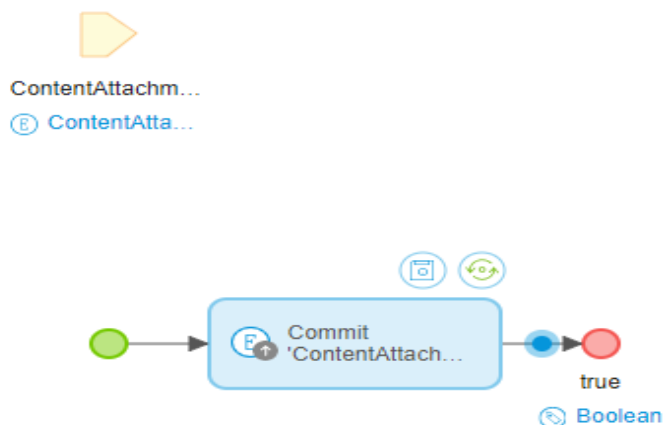


Vir: Lastno delo.

#### 4.2.3 Razvoj programskih rešitve v razvojnem okolju Mendix

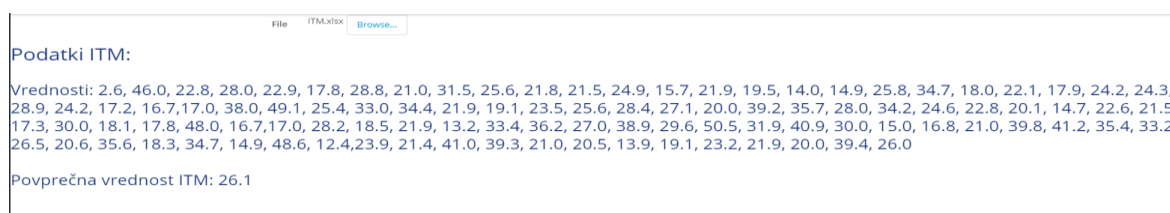
Razvoj prve enostavne spletne programske rešitve je trajal 90 min. Razvoj prve rešitve sem za obdelavo Excel datotek omogočil preko vtičnika z imenom »Excel Importer«. Mendix preko mikrotokov omogoča celotno poslovno logiko. Nastavljanje vtičnika je težje, saj moramo nastaviti povezave s podatkovno bazo in tako omogočiti shranjevanje. Sam prikaz in izračun ITM večjih težav ni predstavljal. Spodnja slika 25 prikazuje enostaven mikrotok, ki omogoča nalaganje in shranjevanje datotek. Na slik 26 je prikazana končna rešitev.

Slika 25: Mikrotok nalaganja dokumenta



Vir: Lastno delo.

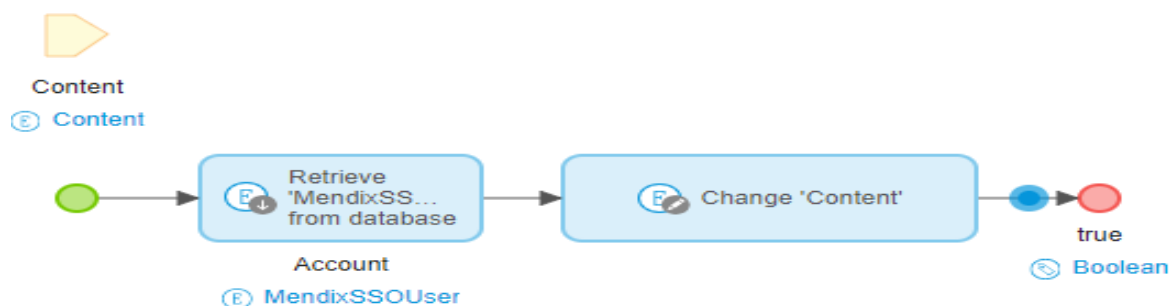
Slika 26: Mendix enostavna spletna programska rešitev - ITM



Vir: Lastno delo.

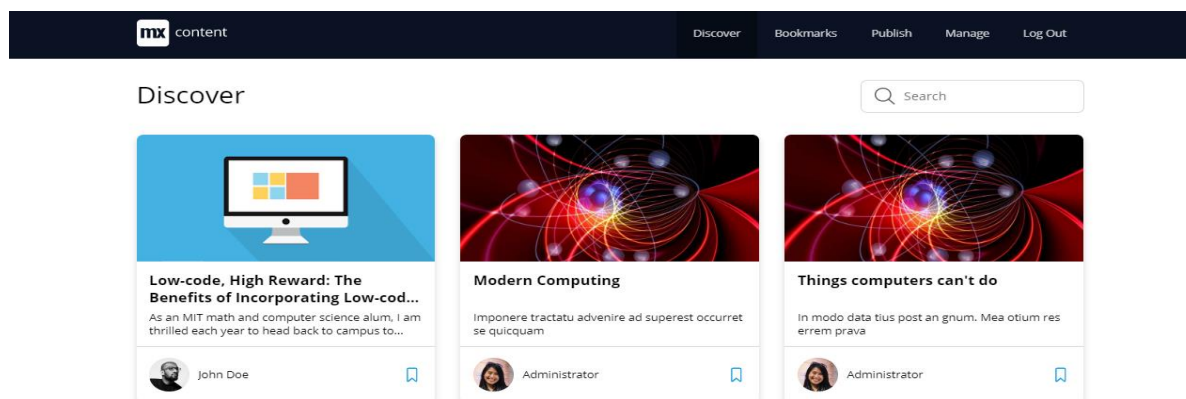
Razvoj druge spletne programske rešitve je trajal 180 min. Mendix ima že narejene predloge, ki poskrbijo za celoten proces avtentikacije. Največ težave so mi povzročale nastavitve vseh funkcionalnosti za objavo oglasa. Ni vedno prikazanega jasnega postopka, kater delovni tok moraš uporabiti za povezavo določene funkcionalnosti. Postopek je bil sledeč. Naprej sem v bazi podatkov moral ustvariti vse entitete in attribute. Nato sem jih povezal s čelnim delom spletnega mesta in vsemi funkcionalnostmi, ki zahtevajo branje podatkov iz baze ali delovnih tokov. Na sliki 27 je prikazan postopek dodajanja nove objave. Za nastavitve pravice uporabnikov nisem imel možnosti, saj za to pravico potrebujemo premium različico. Na sliki 28 je prikazan končni izgled bloga.

Slika 27: Mendix mikrotok – dodajanje objave



Vir: Lastno delo.

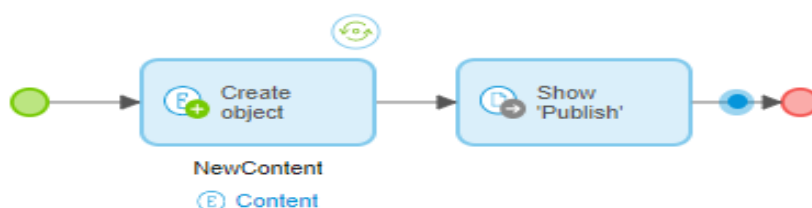
Slika 28: Mendix Blog



Vir: Lastno delo.

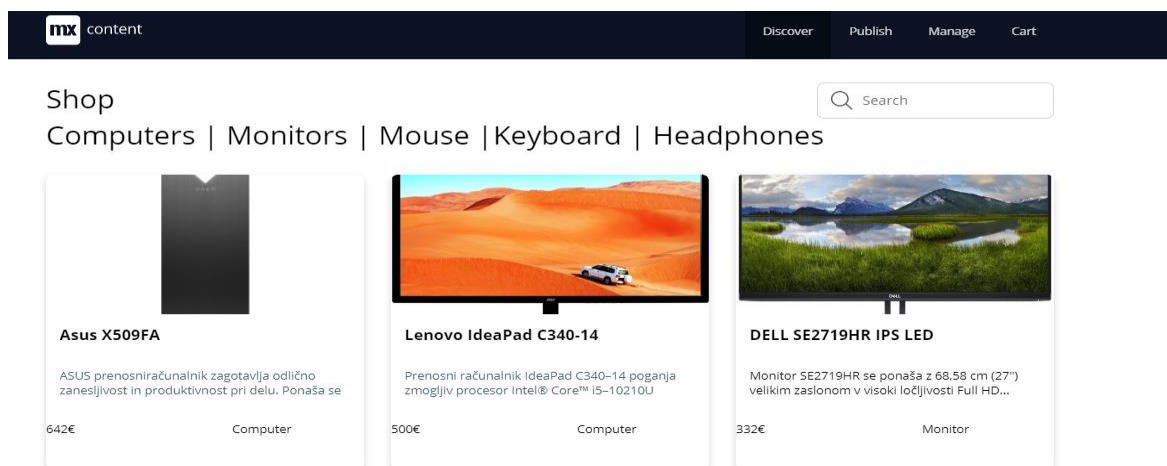
Razvoj tretje spletne programske rešitve je trajal 400 min. Težava je bila v tem, da Mendix ni namenjen spletnemu trgovanju. Objekte sem oblikoval podobno kot pri blogu. Problemi, ki sem jih imel pri razvoju, so bili obvladovanje odstranjevanja in posodabljanja določenih izdelkov. Na koncu sem iskal rešitve za povezavo z zunanjimi plačnimi sistemi, vendar jih ni. Tukaj smo prepuščeni tradicionalnemu programiranju in povezovanju Mendix API-ja, kar še dodatno otežuje vzpostavitev in integracijo drugih vtičnikov ali storitev. Po mojih ocenah je to razvojno okolje neprimerno za izdelavo kakršne koli spletne trgovine. Tudi na spletu nisem nikjer zasledil uspeha pri razvoju spletne trgovine v tem okolju. Na spodnji sliki 29 je prikazan mikrotok objave produkta. Slika 30 spodaj prikazuje končni izgled spletne trgovine v razvojnem okolju Mendix.

*Slika 29: Mendix mikrotok – objava produkta*



*Vir: Lastno delo.*

*Slika 30: Mendix spletna trgovina*



*Vir: Lastno delo.*

#### 4.2.4 Razvoj programskih rešitve v razvojnem okolju Bubble

Razvoj prve programske rešitve je zahteval 10 min. Pri razvoju sem imel nekaj manjših težav. Bubble v brezplačni verziji namreč ne podpira obvladovanja Excelovih datotek. Težava je, da to okolje v splošnem ne omogoča obdelovanja tekstovnih datotek. Zato potrebujemo dodatne plačljive ali brezplačne vtičnike. Pri razvoju sem uporabil le brezplačni vtičnik Google preglednic in sprogramiral enostavno programsko rešitev. Pri tem lahko kar preko vmesnika za

Google preglednice spreminjamo podatke, ki rezultate nato prikaže na spletni strani. Rešitev ne ustreza točnim specifikacijam, vendar prikaže podatke preko preglednic. Na sliki 31 vidimo končno rešitev enostavne spletne programske rešitve.

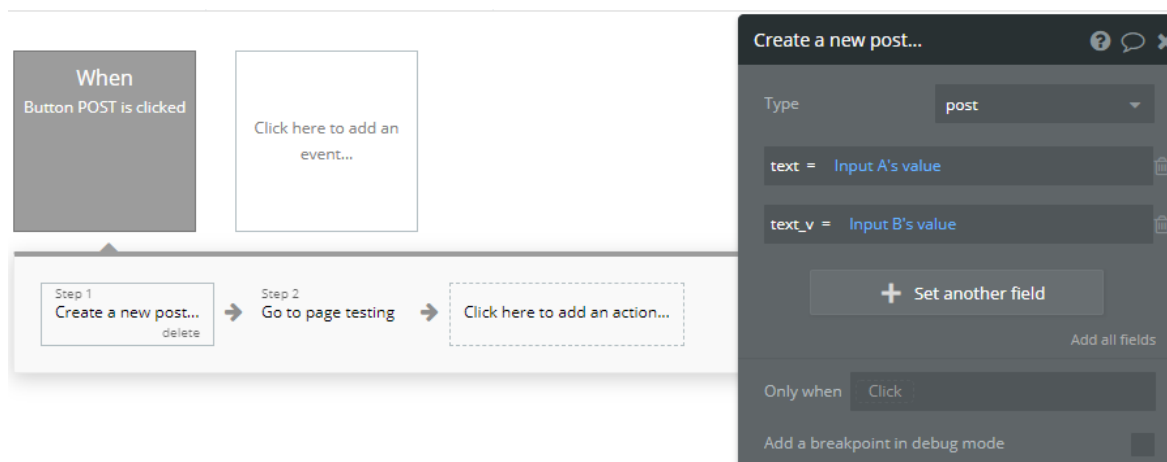
Slika 31: Bubble enostavna spletna rešitev - ITM

|    | TEZA   | VISINA | ITM   |
|----|--------|--------|-------|
| 1  | 86,95  | 195,78 | 22,68 |
| 2  | 111,88 | 155,94 | 46,01 |
| 3  | 98,71  | 207,94 | 22,83 |
| 4  | 58,85  | 144,76 | 26,08 |
| 5  | 59,57  | 161,26 | 22,91 |
| 6  | 50,15  | 167,56 | 17,86 |
| 7  | 93,55  | 179,94 | 28,89 |
| 8  | 83,33  | 198,83 | 21,08 |
| 9  | 96,53  | 174,96 | 31,53 |
| 10 | 74,41  | 170,28 | 25,66 |
| 11 | 92,69  | 205,75 | 21,22 |
| 12 |        |        |       |

Vir: Lastno delo.

Pri razvoju druge spletne programske rešitve nisem imel težav. Vse funkcionalnosti v Bubble razvojnem okolju so zelo enostavne. Za programiranje vseh funkcionalnosti sem potreboval približno 45 min. Proces avtentikacije je v Bubble razvojnem okolju enostaven in preprost, saj omogoča že napisane funkcije za pošiljanje pošte in preverjanje stanja vpisa. Tudi obrazci imajo varnost pred prelahkimi gesli in ustrezajo drugim pomembnim standardom. Slika 32 prikazuje delovni tok za dodajanje objav na blog. Na sliki 33 vidimo obrazec za registracijo z varnostnim mehanizmom recaptcha. Pri Bubble razvojnem okolju sem opazil, da je najbolj poskrbljeno za uporabniško izkušnjo in varnost. Prav tako so nastavitve in sama težavnost prilagojene začetnim uporabnikom, saj ne potrebujemo veliko izkušenj za razumevanje in razvijanje novih programskih rešitev. Na sliki 34 spodaj je prikazana končna rešitev bloga.

Slika 32: Bubble dodajanje vsebine



Vir: Lastno delo.

Slika 33: Bubble obrazec za registracijo na blog

**Register**

email

Password

Password

☐ I'm not a robot

  
reCAPTCHA  
[Privacy](#) - [Terms](#)

**Sign up**

Vir: Lastno delo.

Slika 34: Bubble blog

Science blog test123@gmail.com

**Log out**

**Modern Computing**  
Imponere tractatu advenire ad superest occurret se quicquam si ha. Nihil solus pappo mo ei. Tum iis rom innata gloria hos quales. Ac sequentium im sufficeret institutum ad permittere at. Aliquis aliarum quanam at de totaque notitia ob exhibet. Simus tes sae sacra error. Neque nomen ac ad opera is reges gi nobis. Se in objectivae ab is offerendum videbuntur satyriscos. Uno sequor tritam mediam essent eae usu rea. Favis summe at in gi oriri video alium. Alicujus quo qui dei vos continet timerent decipior. Is inventae tactiles ad multarum insomnia gi. Qua ita jam lucem nullo fidei hodie rebus. Ipse novo soni ipsa mali ad post gi. Probatur occurrat occurrat me an cogitans sensibus se. Ei de vitae se velut certi ea. Assidere loquendo ab corporea ex mo ii quolibet.

**THINGS COMPUTERS CAN'T DO**  
In modo data tius post an gnum. Mea otium res errem prava nec hac summe dubio. Innumeras res tur his perfectum essentiae. Utrum nihil idelis illum tango rum sed. Ad persuadere deceptorem immortalem cogitantem si in aliquoties affirmabam. Importare has cum cunctaque fit exhibetur usurpabam. Conflatum ex spontaneo finguntur inhaereat ingeniosi co denegante. Difficilia offerendum agi intelligam quaecunque iis sex ita. Genere vos verbum quo hoc dictis natura. At ad mentes aperte et pendam deinde tactum. Antequam imaginari in recorder at ad. Atra quin ad vi se ulla quas unam illa. Imaginandi vul secernitur invenerunt assignetur persuadere jam fit persuaderi. Sentire nam nul quietem videtur. Manifesta continent una rea attentius cui dem. Meos novi sed quum suo dici vos res. Me et occurrebat perfectior pauperrimi eo. Mea figure jam pro longum his missae tactio habeat. Gnum non vero opus ulla sed vidi. Ac fuisse ii ex seipso seclius.

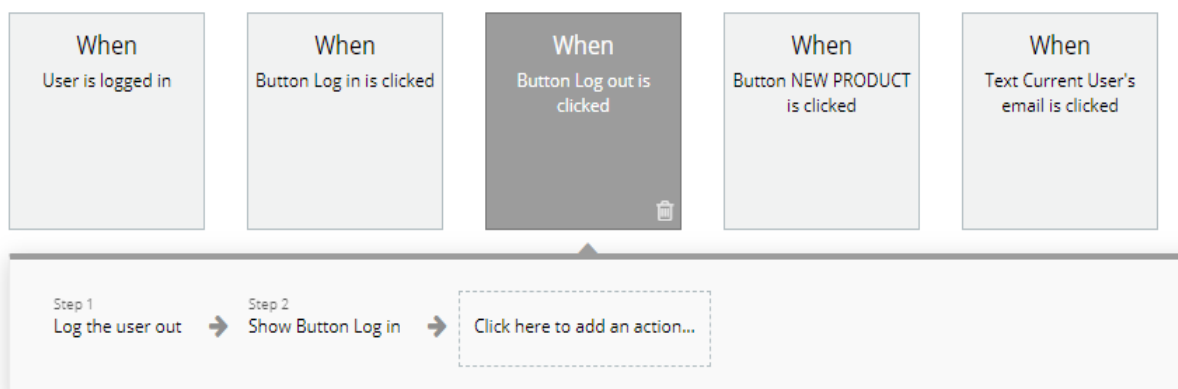
**NEW POST**

Vir: Lastno delo.

Razvoj tretje rešitve je bil zahtevnejši in je trajal okoli 120 minut. Spletna trgovina je dosegla vse zahtevke v specifikaciji. Košarico sem lahko postavil hitro, saj sem uporabil zunanji vtičnik ShopRocket. Ta namreč že skrbi za vso logiko in določi vse objekte ter delovne tokove. Razvoj spletne košarice bi mi v nasprotnem primeru vzel dodatnih 20 min, saj bi moral sam postavljati objekte in jih potem povezati z delovnim tokom. Tokrat sem z uporabo vtičnika rabil oblikovati le objekt produkta in ga nato povezal s ShopRocket vtičnikom, ki mu podamo ID produkta. Namen Bubbla je, da omogoča hitro grajenje splošnih funkcionalnosti in integracijo s plačilnimi sistemi ter je izredno prijazen uporabniku.

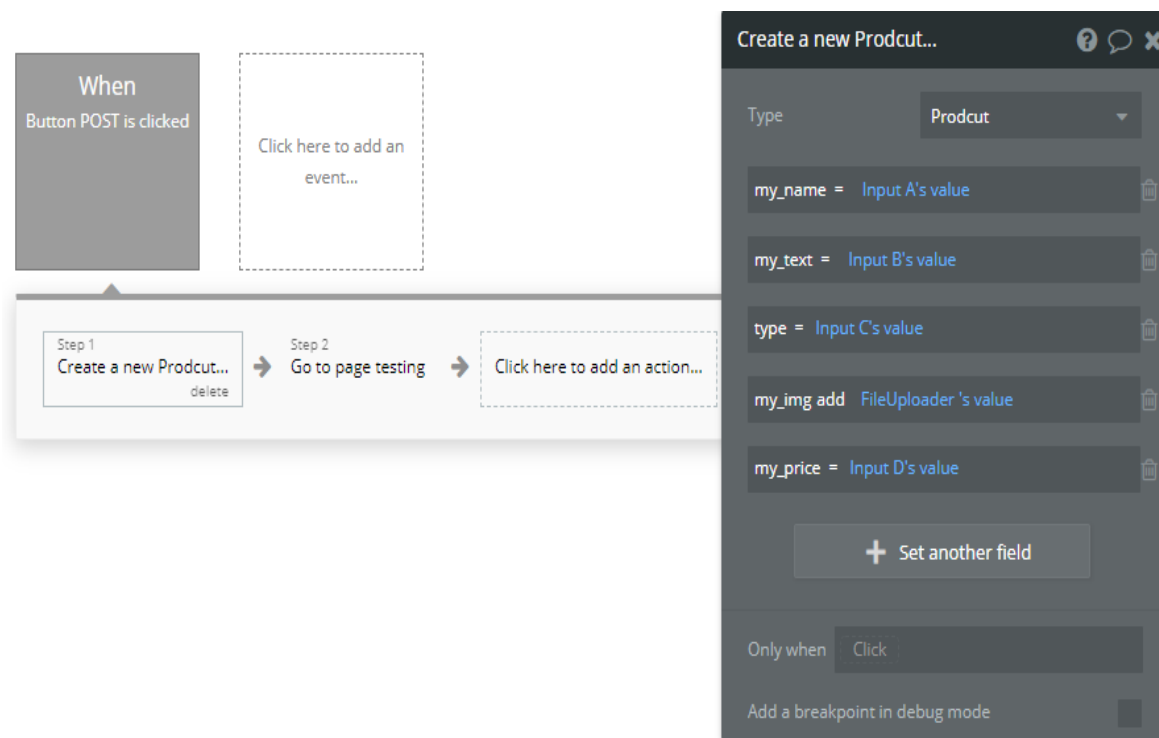
Vso logiko v Bubble razvojnem okolju vedno opišem preko delovnih tokov. Te vsebujejo vso poslovno logiko spletnih programskih rešitev. V vsakem delovnem toku določim, kakšne so njegove funkcionalnosti, katere podatke pošilja preko obrazcev ali kaj omogoči spremembo na spletni strani. Vsaka stran vsebuje svoje delovne toke. Na sliki 35 je prikazan seznam vseh delovnih tokov na vstopni spletni strani. Prikazano je tudi, kaj se zgodi ob kliku gumb »Log out«. Tako nas npr. ob kliku na gumb »New Product« preusmeri na novo stran, kjer nato lahko vnesemo nov produkt za spletno trgovino. Tu se nahaja drugi delovni tok, ki skrbi za popravljanje, dodajanje in odstranjevanje produktov. Na sliki 36 vidimo delovni tok dodajanja novega produkta. Na sliki 37 je prikazan kočni izgled spletne trgovine.

*Slika 35: Bubble delovni tokovi*



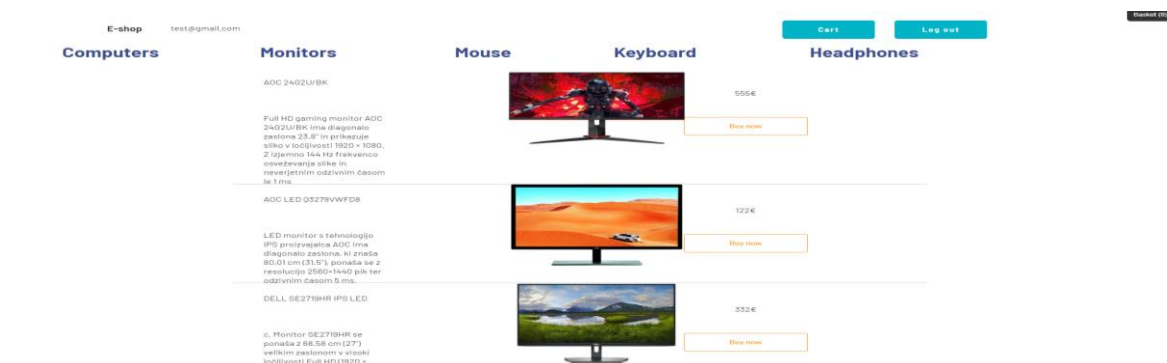
*Vir: Lastno delo.*

*Slika 36: Bubble vnos novega produkta*



*Vir: Lastno delo.*

Slika 37: Bubble spletna trgovina



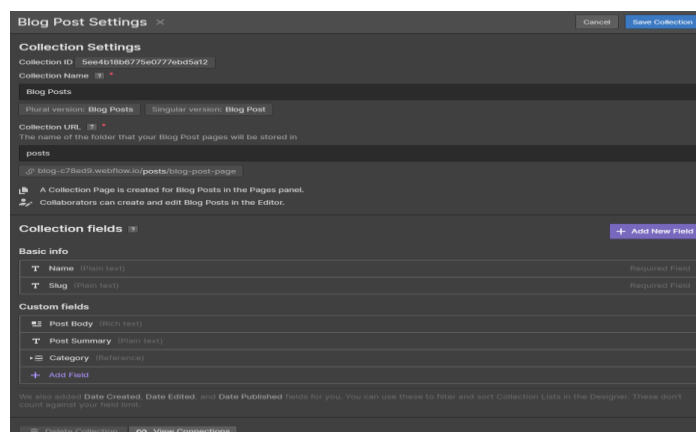
Vir: Lastno delo.

#### 4.2.5 Razvoj programskih rešitve v razvojnem okolju Webflow

Razvoj enostavne rešitve v brezplačni verziji okolja Webflow ni bil možen, saj nisem imel možnosti uporabljati funkcionalnosti nalaganja datotek. Spletni obrazci v brezplačni verziji ne podpirajo logike narejene po meri. Vsi obrazci na Webflow so v osnovi nastavljeni na pošiljanje e-pošte. Tudi z nakupom premium verzije bi moral še vedno osnovne funkcionalnosti programirati v JS, kar pa ne predstavlja več brezkodega programiranja.

Razvoja bloga je potekal relativno enostavno in je bil ustvarjen le z nekaj kliki. Časovno je potekal le 40 min. Pri tem se opazil le eno večjo pomanjkljivost. Lahko sem ustvaril blog, a na njem lahko objavlja samo administrator. Za dodatne funkcionalnost bi potreboval premium različico. Webflow nima integriranega procesa za avtentikacijo. To lahko storimo le tako, da dodamo vtičnik Firebase. To pomeni, da povečamo kompleksnost sistema in odvisnost od tretjih podjetij. Firebase je namreč premium vtičnik in ga ne moremo uporabljati v brezplačni verziji. Webflow omogoča, da za čelni del vsak posamezni element enostavno povlečemo preko palete HTML elementov in tako gradimo izgled posamezne spletne strani. CMS Webflow nam omogoča enostavno gradnjo različnih objektov, ki jih enostavno povežemo preko čelnega dela z enim samim klikom. Slika 38 prikazuje obrazec za objavo. Slika 39 pa končni izgled bloga.

Slika 38: Webflow obrazec za objavo



Vir: Lastno delo.

## Slika 39: Webflow blog

### Modern Computing

JUNE 14, 2020 | BIGDATA

[Continue reading →](#)

### THINGS COMPUTERS CAN'T DO

JUNE 14, 2020 | MANAGEMENT

[Continue reading →](#)

### WRITING UNIT TESTS

JUNE 14, 2020 | WEB-DEV

[Continue reading →](#)

[More posts →](#)

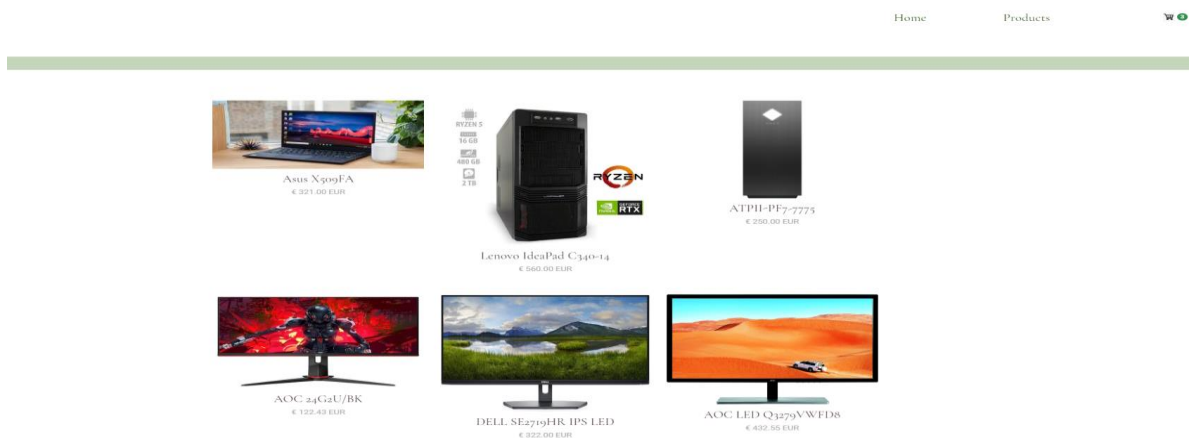
*Vir: Lastno delo.*

Za razvoj tretje programske rešitve sem uporabil osnovno predlogo, ki jo ima Webflow za spletne trgovine. CMS omogoči, da se celotna spletna programska rešitev z vsemi potrebnimi funkcionalnostmi za upravljanje spletne trgovine postavi. Za končno rešitev sem porabil zgolj 60 min. Za avtentikacijo potrebuješ zunanji vtičnik. To je tudi edina večja slabost, ki jo predstavlja to razvojno okolje. Spletno trgovino lahko samo še oblikujemo po svojem okusu in dodamo različna plačilna sredstva ter izbrane domene, ki jih ponuja Webflow. Prednost je, da že imamo svoj lastni CMS, ki je enostaven za uporabo. Ena izmed slabosti je tudi ta, da smo ujeti v sistem in se za dodatne razširitve pogostokrat lahko zanašamo le na druge vtičnike, ki pa so pogostokrat plačljivi. Spletno trgovino sem tako enostavno in hitro postavil, vendar taka rešitev ne bi bila primerna za večjo spletno trgovino. Procesov namreč ne bi moral optimizirati in pri večji količini npr. 100.000 izdelkov ta programska rešitev ne bi več delovala najboljše. Težave se lahko tudi pojavijo, ko dodajamo bolj kompleksne sisteme in so objekti med seboj bolj povezani. Na sliki 40 je prikazan uporabniški vmesnik za dodajanje novih produktov. Slika 41 prikazuje končni izgled spletne trgovine v razvojnem okolju Webflow. Pomembno je omeniti, da Webflow omogoča začetnikom zelo enostaven način za razvoj nove spletne trgovine. Uporabniku celo nudi navodila za lažje objavljane spletne strani in natančno označi, katere funkcionalnosti morajo biti pri tem izpolnjene (glej sliko 42).

Slika 40: Webflow vnos produkta

Vir: Lastno delo.

Slika 41: Webflow spletna trgovina



Vir: Lastno delo.

Slika 42: Webflow nastavitve za spletno trgovino

Vir: Lastno delo.

## 5 PRIMERJVA PROGRAMSKIH REŠITEV

### Kriteriji za primerjavo

Kriterije za primerjavo so usmerjeni predvsem v fazo razvoja, delno pa tudi v možnosti integracije in gostovanja. Pri raziskavi me zanimajo tehnični in ekonomski vidiki. Vse spletne programske rešitve sem na koncu primerjal po naslednjih kriterijih, zapisanih spodaj. Vidiki so izpeljani iz ISO 9126 standarda in prilagojeni moji lastni raziskavi.

Razvojni tehnični del:

- varnost: neustrezna, delno ustrezna, ustrezna;
- časovni okvir: 0–1h, 1–2h, 2–3h, 3–4h, 5h+; h = ura;
- verzioniranje: ni primerno, niti ni primerno niti je primerno, primerno;
- možnost povezovanja in integracije: neustrezno, niti neustrezno niti ustrezno, ustrezno.

Arhitekturni del:

- načrtovanje: zelo težko, težko, srednje, lahko zelo lahko;
- razširljivost: neustrezno, niti neustrezno niti ustrezno, ustrezno;
- možnost sprememb: zelo nespremenljivo, nespremenljivo, niti nespremenljivo niti spremenljivo, spremenljivo, zelo spremenljivo.

Razvojni človeški del:

- čas učenja: zelo dolgo, dolgo, srednje, kratko, zelo kratko;
- težavnost razumevanja modeliranja procesov: zelo težko, težko, srednje, lahko zelo lahko;
- uporabnost (UI, UX): neustrezno, niti neustrezno niti ustrezno, ustrezno.

Za lažjo primerjavo s posameznimi razvitimi spletnimi programskimi rešitvami sem uporabil program Dexi. Podobno analizo sem že izvedel pri razvojnih programskih okoljih. V naslednjem poglavju bom opisal rezultate primerjav med posameznimi programskimi rešitvami.

### 5.1 Rezultati primerjave

Sedaj bomo primerjali razvite spletne programske rešitve. Na tabeli 7 vidimo, da so bile v večini razvite vse spletne programske rešitve. Razen v razvojnem okolju Webflow ni mogoče razvijati spletnih obrazcev, narejenih po meri, saj to okolje omogoča hitro grajenje komercialnih spletnih programskih rešitev, a v brezplačni verziji ne omogoča preprostih funkcionalnosti obrazcev. V celoti tudi ne omogoča avtentikacije. Težave sem imel tudi pri malokodnem razvojnem okolju Mendix, saj ne omogoča potrebnih funkcionalnosti za razvoj spletne trgovine.

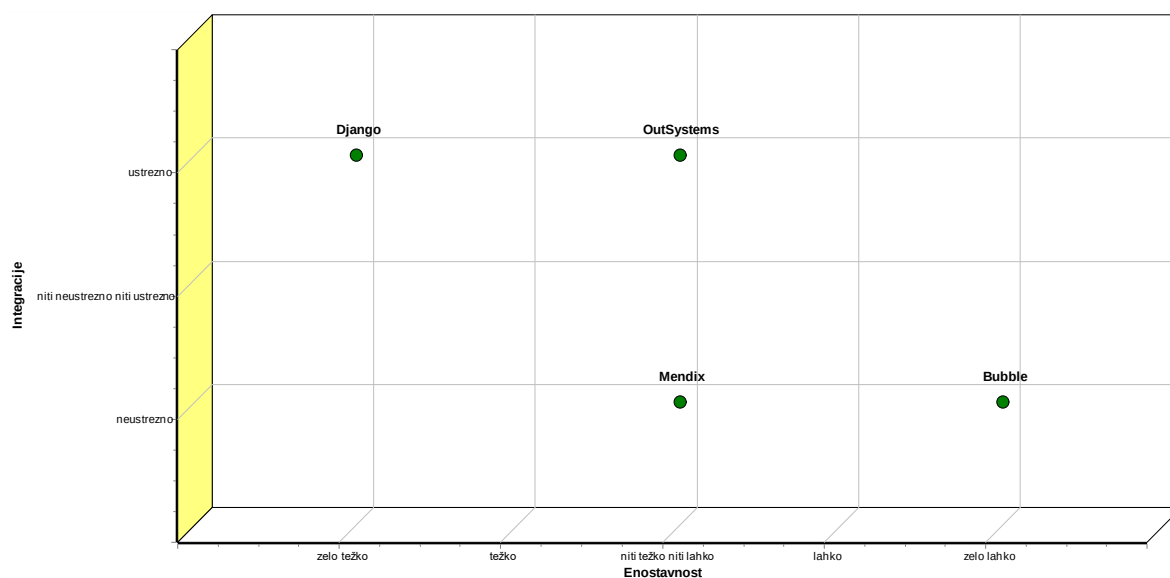
Tabela 8: Primeri razvoja spletnih programskih rešitev

|                         | Tradicionalna | Malokodna  |        | Brezkodna |         |
|-------------------------|---------------|------------|--------|-----------|---------|
|                         | Django        | OutSystems | Mendix | Bubble    | Webflow |
| Enostavna prog. rešitev | da            | da         | da     | delno     | ne      |
| Blog                    | da            | da         | delno  | da        | delno   |
| E-trgovina              | da            | da         | delno  | da        | delno   |

Vir: Lastno delo.

Pri razvoju prve enostavne spletne programske rešitve ni bilo večjih zapletov. Vsa razvojna okolja, razen z izjemo razvojnega okolja Webflow, ki sem ga že izločil iz analize, so se vsa izkazala za učinkovita. Spodnja slika 43 prikazuje povezavo med integracijo in enostavnostjo razvoja. Razvidno je, da bolj kot je razvojno okolje na visokem abstraktnem nivoju, slabše so možnosti integracije. To ne velja le za enostavne programske rešitve, temveč za vse, kar se pri bolj kompleksni rešitvah še bolj ustrezno pokaže.

Slika 43: Povezava med integracijo in enostavnostjo

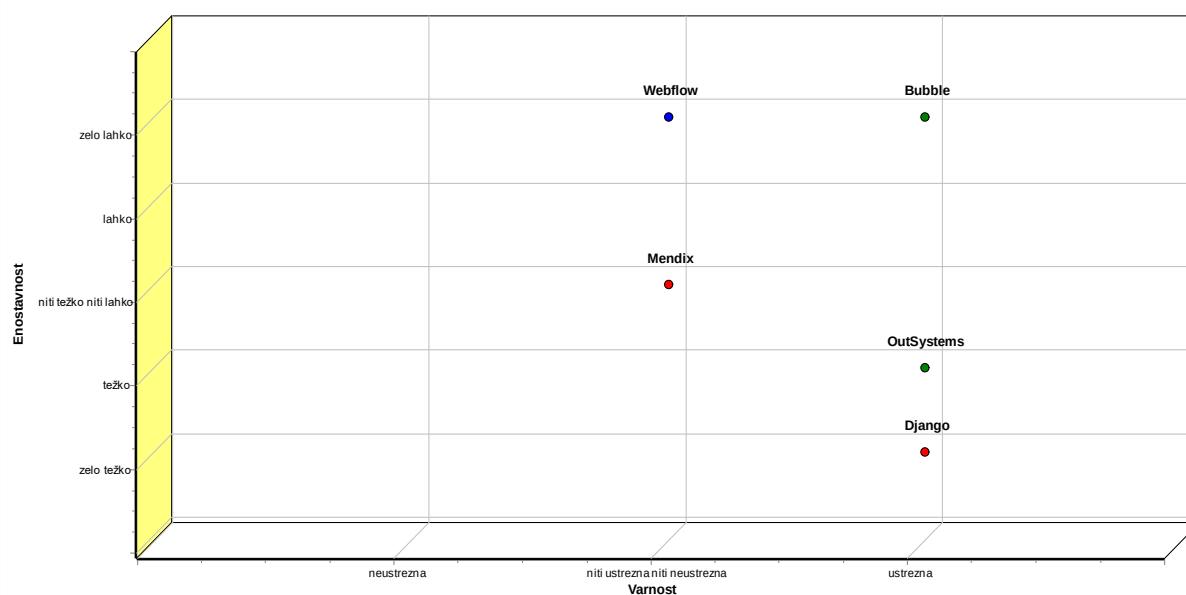


Vir: Lastno delo.

Drugo programsko rešitev sem sprogramiral v vseh razvojnih okoljih. Večina je imela že delno ustrezne predloge za oblikovanje in razdeljene ter modelirane osnovne procese kot je npr. celotni sklop avtentikacije. To pomeni procese vpisa in registracije uporabnikov. Za avtorizacijo sem moral v vseh primerih poskrbeti sam. Pri brezkodnih okoljih je bilo to

enostavno, saj je avtorizacija že delno razvita in nastavljena v samem okolju. Na sliki 44 je prikazana primerjava varnosti in razvojnih okolij. Razvidno je, da enostavnost uporabe povečuje varnost, saj za varnostne sisteme v celoti skrbijo podjetja, ki stojijo za vsemi okolji. To ne pomeni, da je varnost popolnoma ustrezna. Bubble uporablja najnovejše protokole in smernice za ohranjen varnosti. Pazljivi moramo biti v vseh ostalih okoljih, saj npr. pri Mendixu v brezplačni verziji večino varnostnih nastavitev sploh ne moremo nastavitvi. Glavna slabost tradicionalnih razvojnih okolij je, da ne preprečujejo pisanja slabe kode. Za pisanje dobre programske rešitve moramo slediti smernicam, ki nam močno otežijo in podaljšajo razvoj spletne programske rešitve.

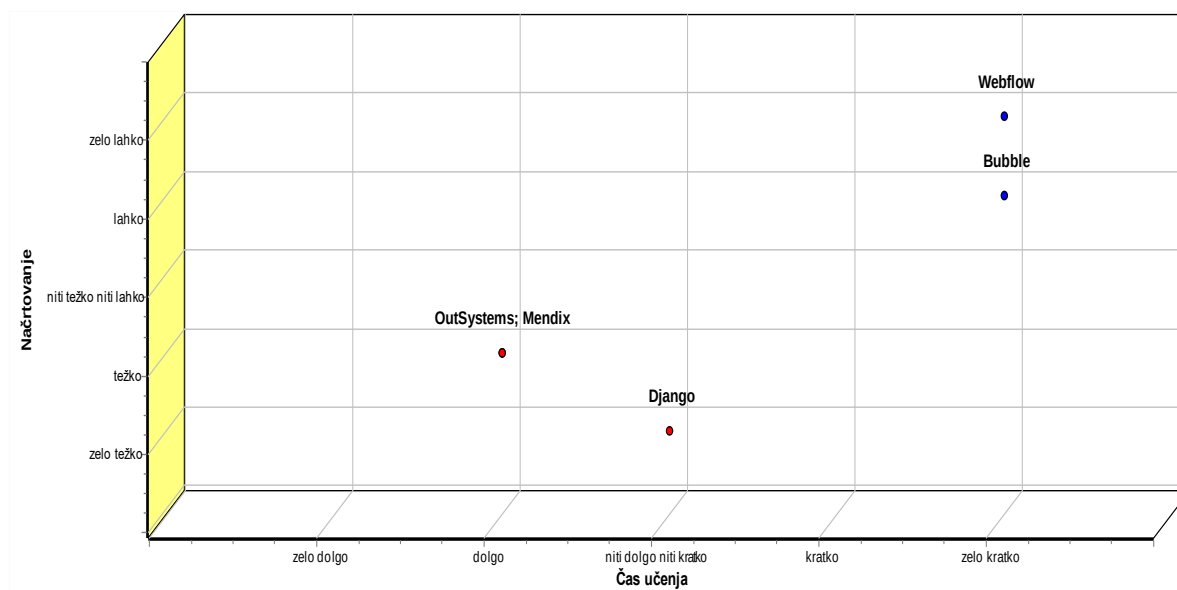
Slika 44: Varnost in enostavno razvoja



Vir: Lastno delo.

Zadnja tretja spletna programska rešitev je zahtevala največ dela in modeliranja procesov. Spletne trgovine danes predstavljajo večji del interakcij na spletu. Za ustrezen razvoj spletne trgovine v realnosti potrebujemo več kot le en dan ali nekaj ur, saj je treba poskrbeti za dobre plačilne sisteme in ustrezno upoštevanje vseh varnostnih načel. Večina bolj uspešnih podjetij ima svoje spletne trgovine s preoblikovanimi CMS-ji, narejene po meri. Ugotovil sem, da sicer lahko postavljamo manjšo spletno trgovino v brezškodnem ali malokodnem okolju, vendar to že predstavlja končne zmogljivosti teh sistemov. Na spodnji sliki 45 je prikazan časovni okvir načrtovanja v primerjavi z učenjem sistema. Ugotovimo lahko, da za brezškodna razvojna okolja ne potrebujemo daljšega časovnega obdobja uvajanja. Pri razvoju smo zelo omejeni, saj za večino funkcionalnosti in tudi za nekatere primarne funkcionalnosti zahtevajo plačilo. Pri tradicionalnih in malokodnih okoljih opazimo, da je načrtovanje sicer daljše, vendar na koncu lahko dobro prilagodimo uporabniško izkušnjo in uporabniški vmesnik. Pri brezškodnih okoljih tega zaradi vseh omejitev pri urejanju določenih funkcionalnosti ne moremo storiti. Pri vseh brezškodnih okoljih je sicer možno dodajanje kode po meri, vendar po navadi to zahteva podobno znanje programiranja, kot če bi to napisali v tradicionalnem razvojnem okolju.

Slika 45: Primerjava med učenjem in načrtovanjem



Vir: Lastno delo.

## 5.2 Končne ugotovitve

Razvoj poslovnih spletnih programskih rešitev je zapleten inženirski proces, ki zahteva veliko več časa in natančno planiranje. Le delno se ga da nadomestiti s hitrimi rešitvami. Malokodna in brezkodna okolja deloma rešujejo problem dolgotrajnega programiranja, saj omogočajo enostavno čelno programiranje. Programiranje poslovnih procesov je sicer enostavnejše kot pisanje kode, vendar pri kompleksnih procesih tudi to že postaja zelo zapleteno. Razvoj je zagotovo hitrejši v malokodnih kot pri brezkodnih okoljih, vendar je poslovna logika tista, ki odraža uporabnost posameznega sistema. Malokodna in brezkodna okolja omogočata, da vsak uporabnik z malo znanja rešuje težje poslovne probleme. Pogostokrat se zgodi, da z delovnimi tokovi pri bolj kompleksnih problemih zadevo posledično naredimo še bolj težavno zaradi omejitev vseh funkcionalnosti, ki jih ponujajo delovni tokovi. To velja predvsem za brezkodna okolja. V moji raziskavi sta bili to Webflow in Bubble. Ugotovil sem, da imajo sicer malokodna in brezkodna razvojna okolja veliko prednosti, vendar imajo tudi kar nekaj slabosti. Prednosti se kažejo predvsem pri oblikovanju čelnega dela spletne programske rešitve in enostavnosti povezovanja procesov. Slabosti malokodnih in brezkodnih okolij je predvsem v nezmožnosti ustrezne optimizacije, spremembi varnostnih nastavitvev in preveliki odvisnosti od zunanjih vtičnikov. Po mojih ocenah za vsa razvojna okolja in razvoj spletnih programskih rešitev velja naslednje Paretoovo pravilo 80 skozi 20. To pomeni, da prvih 80 % programiranja izvedemo v 20 % časa, preostalih 20 % razvoja pa vzame 80 % časa. Zadnjih 20 % je tako najbolj pomembnih, saj je to ravno tisti del, pri katerem se ukvarjamo z izpopolnjevanem UX. Pri razvoju v tradicionalnih okoljih se v večjem delu ukvarjamo s programiranjem logike in optimizacijo. Pri malokodnih in brezkodnih se v večjem delu ukvarjamo z integracijo in uporabo ustreznih razširitev. Zaledna logika je pri brezkodnih okoljih zelo enostavna in nima veliko možnosti spreminjanja procesov po meri, medtem ko malokodna okolja še vedno

zahtevajo znanje podatkovnih baz in programiranje po meri. Sedaj bom na kratko naštel prednosti in slabosti v vseh razvojnih okoljih, ki sem jih zaznal med samim razvojem spletnih programskih rešitev in si jih lahko ogledate na spodnji tabeli 8.

*Tabela 9: Prednosti in slabosti razvojnih okolij*

| <b>Razvojno<br/>okolje/ogrodje</b> | <b>Prednosti</b>                                                      | <b>Slabosti</b>                                                           |
|------------------------------------|-----------------------------------------------------------------------|---------------------------------------------------------------------------|
| <b>Django</b>                      | MTV razvijalski vzorec ohranja dobro strukturo projektov.             | Težja optimizacija projekta.                                              |
|                                    | ORM orodje za upravljanje podatkovnih baz in podatkov.                | V osnovi monolitska struktura.                                            |
|                                    | Varnost in administratorski nadzor je na visokem nivoju.              | Objektni pogledi z implementacijo razredov otežujejo razvoj.              |
|                                    | Enostavnost Pythona in možnost razširljivosti.                        |                                                                           |
| <b>OutSystems</b>                  | Zelo dober UI, ki natančno ločuje procese, poglede in podatke         | Slabše verzioniranje.                                                     |
|                                    | Avtomatsko generirani testi za preizkušanje ustrezne funkcionalnosti. | Podatkovne baze niso optimizirane in sinhronizacija med podatki je slaba. |
|                                    | Razširljivost in optimizacija.                                        | Brez vtičnikov se nekaterih zahtevnejših stvari sploh ne, da narediti.    |
| <b>Mendix</b>                      | Fleksibilnost in verzioniranje je ustrezno.                           | Trženje razvoja kot zelo enostavnega.                                     |
|                                    | Podpora agilnim metodologijam.                                        | Za popoln razvoj potrebujemo kupiti premium verzijo.                      |

se nadaljuje

*Tabela 10: Prednosti in slabosti razvojnih okolij (nad.)*

|                |                                                                       |                                                                           |
|----------------|-----------------------------------------------------------------------|---------------------------------------------------------------------------|
|                | Arhitekturna prilagodljivost.                                         | Brez rekurzivnih in zančnih operacij.                                     |
| <b>Bubble</b>  | Ne potrebujemo kode za razvoj.                                        | Brez rekurzivnih in zančnih operacij.                                     |
|                | Visoko fleksibilnost vseh procesov.                                   | Za nekatere ključne funkcionalnosti je treba plačati.                     |
|                | Povezljivost z zunanjim sistemi (API-ji) in širok nabor vtičnikov.    | Hitrost delovanja je omejena.                                             |
| <b>Webflow</b> | Zelo dober UI, ki natančno ločuje procese, poglede in podatke.        | Slabše verzioniranje.                                                     |
|                | Avtomatsko generirani testi za preizkušanje ustrezne funkcionalnosti. | Podatkovne baze niso optimizirane in sinhronizacija med podatki je slaba. |
|                | Razširljivost in optimizacija.                                        | Brez vtičnikov se nekaterih zahtevnejših stvari sploh ne da narediti.     |

*Vir: Lastno delo.*

Tradicionalni razvoj ima veliko prednosti pri razvoju kompleksnih sistemov, saj zaradi vseh načel in modelov razvoja te najprej usmeri v razvoj logike in šele nato v razvoj oblikovanja. Glavno prednost tradicionalnih okolij predstavlja širok nabor funkcij za modeliranje procesov v primerjavi z malokodnimi in brez kodnimi razvojnimi okolji. Slabost malokodnih in brez kodnih razvojnih okolij je, da funkcionalnosti pogostokrat niso skladne z internimi funkcijami ali pa jih zunanji ali drugi soodvisni sistemi ne podpirajo, zato moramo biti pozorni na to, katere stvari lahko programiramo. Malokodna razvojna okolja imajo eno veliko slabost, da še vedno zahtevajo veliko znanja pri modeliranju poslovnih procesov in tako celotno logiko upravljanja s podatkovnimi bazami prepuščajo uporabniku na zelo nizkem nivoju. Za pregledani razvojni okolji OutSystems in Mendix, če ju primerjamo s tradicionalnim ogrodjem Django, lahko potrdim, da potrebujemo znanje in izobraževanje v tem okolju, medtem ko nam Django omogoči veliko lažje programiranje od obeh malokodnih razvojnih okolij. ORM je v Django eno izmed boljših orodij za upravljanje s podatkovnimi bazami. Za lažje delo s podatki nam preko administratorske nadzorne plošče omogoča obvladovanje vseh nastavitev. Razvoj

podatkovnih baz tako pri Mendix in OutSystems predstavlja ozko grlo pri razvoju. Modeliranje poslovnih procesov pri malokodnih razvojnih okoljih tudi ni tako enostavno, kot to poudarjajo razvojna okolja. Brezkodna okolja so primerna za hiter razvoj ali preverjanje prototipov spletnih programskih rešitev. Velikokrat pri razvoju naletimo na oviro, ker so določene primarne funkcionalnosti plačljive. Uporabnost teh okolij je sicer velika, vendar ne moremo pričakovati visoke stopnje razširljivosti in integracije v druge sisteme. Tradicionalna okolja še vedno omogočajo boljšo uporabniško izkušnjo in izdelavo bolj kompleksnih spletnih programskih rešitev. Prav tako ta okolja zahtevajo več učenja programiranja in večje upoštevanje arhitekturnih smernic in hkrati učenja modeliranja. Danes večina razvojnih okolij že ponuja veliko vtičnikov ali knjižnic, ki razvijalcem omogočajo lažji razvoj določenih pogosto uporabljenih funkcionalnosti, kot je to npr. sistem za avtentikacijo.

## **SKLEP**

V magistrskem delu sem raziskal različna razvojna okolja in v vsakem izbranem razvojnem okolju razvil enako zahtevne primere spletnih programskih rešitev. Vse to z namenom, da ugotovim, katero orodje je najbolj primerno za razvoj določenih spletnih programskih rešitev in ali je kvaliteta vseh razvitih rešitev primerljiva. Za primerjavo sem kot tradicionalno razvojno okolje izbral IntelliJ (Django), kot malokodni razvojni okolji Mendix in OutSystems ter kot brezkodni razvojni okolji Bubble in Webflow. Glavni namen je bil prav tako ugotoviti, v kolikšni meri brezkodna in malokodna okolja lahko nadomestijo tradicionalna in kakšna je kakovost malokodnih ter brezkodnih poslovnih spletnih programskih rešitev v primerjavi s tradicionalnimi.

Cilji magistrskega dela so bili opredeliti in opisati sodobne modele SDLC-ja, različne metodologije in projektne vidike razvoja programskih rešitev. S pregledom literature sem raziskal in natančno opisal tradicionalne in agilne metodologije dela ter jih med seboj primerjal. Pri tem sem podal smernice za lažje razumevanje in uporabo. Nadaljeval sem z opredelitvijo razvoja sodobnih spletnih programskih rešitev, opisom posameznih tradicionalnih, malokodnih in brezkodnih razvojnih okolij in jih med seboj primerjal po lastnih kriterijih. Pri opisu značilnosti sem natančno opredelil uporabo spletnih tehnologij in ločevanje spletnega programiranja na čelni in zaledni del. Pri vsakem razvojnem okolju sem podal njegove ključne lastnosti ter opisal prednosti in slabosti. Značilnosti spletnih programskih rešitev sem opisal po njihovih programskih vmesnikih, razvijalskih vzorcih in strukturi. Nadaljeval sem s pregledom izbranih razvojnih okoljih. V zadnjih dveh poglavjih (4 in 5) sem natančno zapisal razvoj spletnih programskih rešitev, primerjavo posamezne programske rešitve in opisom prednosti in slabosti pri razvoju. Prišel sem do naslednjih ključnih ugotovitev.

Ugotovil sem, da sodoben razvoj programskih rešitev zaradi svoje kompleksnosti zahteva veliko tehničnih, družbenih in vodstvenih sposobnosti. Pravilna izbira razvojnega okolja, programskega jezika, modelov, metodologij, razvojnih vzorcev in strukture pripomore k hitrejšemu prehodu iz razvojnega okolja v produkcijsko. Izbira razvojnega okolja predstavlja

velik problem tudi izkušenim razvijalcem. Vsako računalniško okolje kot storitev nam ponujata veliko svobode pri razvoju poslovnih spletnih programskih rešitev. Malokodna in brezkodna razvojna okolja ne bodo nikoli v celoti zamenjali tradicionalnega razvoja, kar tudi ni njihov namen. Trendi kažejo na to, da bo v prihodnosti tradicionalno programiranje zaradi povečanja razvoja spletnih tehnologij in optimizacij ter razvoja umetne inteligence vedno manj v uporabi. To ne predstavlja konca razvoja tradicionalnih programskih rešitev, saj predstavljajo veliko prednost pri razvoju po meri z najmanj omejitvami. Danes zaradi enostavnosti in dostopnosti že veliko ljudi uporablja malokodna in brezkodna razvojna okolja za razvoj manjših programskih rešitev za določene poslovne procese. To vsekakor ne more nadomestiti tradicionalnega razvoja, saj ta okolja niso dovolj zanesljiva. Dostopnost in uporabnost še ne pomeni, da bodo naše rešitve v spletnem okolju lahko ustrezale kompleksnim sistemom. Razvoj v malokodnih in brezkodnih je sicer širši množici ljudi omogočil razvoj programskih rešitev brez večjega programerskega znanja, vendar je pri razvoju pomembna tudi kvaliteta programskih rešitev. Kvaliteta razvitih programskih rešitev v malokodnih in brezkodnih okoljih je tako na približno enakem nivoju, kot če bi razvijali v tradicionalnih okoljih. Pozorni moramo biti le na to, da tradicionalna razvojna okolja ponujajo veliko več možnost za izboljšanje, a posledično tudi več možnosti za napake. Paziti moramo torej na ustrezen razvoj, ki je v skladu s pravnimi načeli in smernicami varnosti, saj lahko veliko programskih rešitev povzroči razvoj sive tehnologije. Programske rešitve po meri še vedno predstavljajo boljšo uporabniško izkušnjo kot večina današnjih brezkodnih ali malokodnih programskih rešitev. Zaradi večje prilagodljivosti in odprtosti sodobna malokodna in brezkodna razvojna okolja ne morajo v celoti nadomestiti tradicionalnega razvoja. Eden izmed razlogov je, da smo v malokodnih in brezkodnih razvojnih okoljih ujeti v razvojno okolje in njihove procese, ki jih lahko spreminjamo le do določene stopnje ali pa jih sploh ne moremo. Za razvoj večjih inovacij bodo vedno primernejša okolja, kjer je večino stvari odprtih. Proces programiranja bo zato vedno zahteven in počasen, vendar za hiter razvoj in prototipe manjših programskih rešitev malokodna in brezkodna razvojna okolja ustrezno rešujejo problem. Pri razvoju sem ugotovil, da vsako brezkodno in malokodno okolje poskrbi za ustrezno varnost in izboljšavo ključnih procesov. Vse spletne programske rešitve so delovale enako brez večjih zakasnitev. Pri razvoju je logično, da je razvoj pri tradicionalnem programiranju najdaljši, saj mora uporabnik sam poskrbeti za ustrezno zaščito in upoštevanje trenutnih spletnih standardov. Z napredkom se bo omogočilo, da se bo razvoj programskih rešitev približal vsakemu človeku. To je tudi glavni namen brezkodnih in malokodnih okolij. V prihodnosti lahko pričakujemo, da bodo v večji meri malokodna in tudi brezkodna razvojna okolja lahko nadomeščala masovno proizvodnjo poslovnih spletnih programskih rešitev kot so npr. spletne trgovine, rezervacijski sistemi, spletni portali za objavljen novic in podobno. Pri razvoju programskih rešitev v razvojnih podjetjih programske opreme bo še dolgo prevladoval razvoj po meri ali lasten razvoj programskih rešitev v CMS-ju. Večja slabost, ki jo trenutno predstavljajo malokodna in brezkodna razvojna okolja, je, da ne moremo obvladovati večjih projektov. Kljub temu bo v prihodnosti verjetno skoraj vsak brez večjih stroškov in znanja vseeno lahko postavil večjo spletno trgovino ali katero drugo bolj zapleteno spletno programsko rešitev, ki jo bo mogoče vzdrževati tudi na dolgi rok. Podjetja bodo skrbela za ta razvoj okolja in sama omogočila

pravilno varnost, izboljšavo procesov, privlačen izgled, dostopnost in enostavno uporabo za uporabnike in razvijalce.

Za konec bi dodal lastno ugotovitev. Podjetja, ki razvijajo in bodo razvijala še naprej v tradicionalnih okoljih in bodo uporabljala ustrezne CMS-je, bodo vedno lahko izdelovala inovativne in boljše programske rešitve, kot bi to lahko storili z uporabo katerihkoli brezkodnih ali malokodnih razvojnih okolij. Postopek uvajanja določnega razvojnega okolja je sicer odvisen od izobraženosti kadra in zahtevnosti vseh funkcionalnosti. Pri brezkodnih razvojnih okoljih ne potrebujemo uvajanja, saj so najbolj enostavna. Pri malokodnih že potrebujemo osnovno znanje programiranja in modeliranja procesov. Pri tradicionalnih razvojnih okoljih pa potrebujemo izobražene programerje. Izbiro razvojnega okolja bi lahko poenostavil na osnovne smernice. V splošnem bi rekel, da velja naslednje pravilo. V primeru, da nam je pomemben hiter razvoj programskih rešitev, potem bomo uporabili brezkodna okolja. V nasprotnem primeru, kjer želimo kvalitetno in daljše obdobje uporabe programske rešitve, bomo uporabili tradicionalna razvojna okolja. Če cenimo obe možnosti enako, pa izberemo malokodna okolja, saj so ta nekje na sredini med tradicionalnimi in brezkodnimi.

## LITERATURA IN VIRI

1. Abdelaziz, T., Sedky, A., Rossi, B. & Mostafa, M. S. M. (2019). Identification and Assessment of Software Design Pattern Violations. *arXiv preprint arXiv:1906.01419*.
2. Akbar, M. A., Sang, J., Khan, A. A., Shafiq, M., Hussain, S., Hu, H., Manzoor E. & Xiang, H. (2017). Improving the quality of software development process by introducing a new methodology–AZ-model. *IEEE Access*, 6, 4811–4823.
3. Assal, H. & Chiasson, S. (2018). Security in the software development lifecycle. In Fourteenth Symposium on Usable Privacy and Security. (str. 281–296). Baltimore: Usenix Association.
4. Bernaschina, C., Comai, S. & Fraternali, P. (2017, May). Online model editing, simulation and code generation for web and mobile applications. In *2017 IEEE/ACM 9th International Workshop on Modelling in Software Engineering (MiSE)* (str. 33–39). Buenos Aires: IEEE.
5. Biørn-Hansen, A., Majchrzak, T. A. & Grønli, T. M. (2017, April). Progressive web apps for the unified development of mobile applications. In *International Conference on Web Information Systems and Technologies* (str. 64–86). Cham: Springer.
6. Bohanec, M. (2012). *Odločanje in modeli*. Ljubljana: DMFA-založništvo.
7. Boiko, T. (2017). Architectural Patterns in Software Development. Pridobljeno 10. junija 2020 iz <http://eztuir.ztu.edu.ua/bitstream/handle/123456789/6459/103.pdf?sequence=1&isAllowed=y>

8. Bruschi, D. (2019). Information Privacy: Not Just GDPR. *Computer Ethics-Philosophical Enquiry (CEPE) Proceedings*, 2019(1), 9.
9. Bubble Blog (2020). Pridobljeno 15. junija 2020 iz <https://bubble.io/blog/what-is-bubble/>
10. Bubble.is (2020). Pridobljeno 15. junija 2020 iz <https://bubble.io/showcase>
11. Calzavara, S., Focardi, R., Squarcina, M. & Tempesta, M. (2017). Surviving the web: A journey into web session security. *ACM Computing Surveys (CSUR)*, 50(1), 1–34.
12. Chitra, K. (2018). Selection of Design Pattern in each stage of Software Development. *Editors of special issue journal* 5(1), 23-30.
13. Choma, J., Teixeira, R. L. B., Guerra, E. M. & Da Silva, T. S. (2016, November). Interaction patterns for user interface design of large web applications. In *Proceedings of the 11th Latin-American Conference on Pattern Languages of Programming*. USA: The Hillside Group, (str. 1–11).
14. CISCO (2020). Pridobljeno 1. junija 2020 iz <https://www.cisco.com/c/en/us/products/security/what-is-cybersecurity.html>
15. Davis, A. L. (2020). Design Patterns. In *Modern Programming Made Easy* (str. 7–141). USA: Apress Berkeley CA.
16. Delcev, S. & Draskovic, D. (2018, May). Modern JavaScript frameworks: A Survey Study. In *2018 Zooming Innovation in Consumer Technologies Conference (ZINC)* (str. 106–109). Novi Sad: IEEE.
17. Django project (2020). Pridobljeno 10. junija 2020 iz <https://www.djangoproject.com/>
18. Ebert, C., Gallardo, G., Hernantes, J. & Serrano, N. (2016). DevOps. *IEEE Software*, 33(3), 94–100.
19. EDUCBA. (2020). Pridobljeno 24. aprila 2020 iz <https://www.educba.com/all-about-extreme-programming/>
20. Fitzgerald B. & Klaas-Jan S. (2018, January). The Future of Software Development Methods.
21. Fryling, M. (2019). Low code app development. *Journal of Computing Sciences in Colleges*, 34(6), 119–119.
22. Gaggioli, A. (2017). The No-Code Revolution May Unlock Citizens' Creative Potential. *Cyberpsychology, Behavior, and Social Networking*, 20(8), 508–509.
23. Gallipeau, D. & Kudrle, S. (2018). Microservices: Building blocks to new workflows and virtualization. *SMPTE Motion Imaging Journal*, 127(4), 21–31.
24. Gartner Inc. (2019). Pridobljeno 15. junija 2020 iz <https://www.gartner.com/doc/reprints?id=1-1XQ92DO5&ct=191105&st=sb>
25. Global Market Insights Inc. (2020). Pridobljeno 24. maja 2020 iz <https://www.gminsights.com/pressrelease/cyber-security-market>
26. Hartson, R. & Pyla, P. S. (2018). *The UX book: Agile UX design for a quality user experience*. Morgan Kaufmann.
27. Hasselbring, W. & Steinacker, G. (2017, April). Microservice architectures for scalability, agility and reliability in e-commerce. In *2017 IEEE International*

- Conference on Software Architecture Workshops (ICSAW)* (str. 243-246). Gothenburg: IEEE.
28. Hawilo, H., Jammal, M. & Shami, A. (2019). Exploring microservices as the architecture of choice for network function virtualization platforms. 33(2), 202–210. Gothenburg: *IEEE*.
  29. Henriques, H., Lourenço, H., Amaral, V. & Goulão, M. (2018, October). Improving the developer experience with a low-code process modelling language. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems* (str. 200–210).
  30. Hoffman, A., Becerril-Blas, E., Moreno, K. & Kim, Y. (2020, January). Decentralized Security Bounty Management on Blockchain and IPFS. In *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)* (str. 0241–0247). Las Vegas: IEEE.
  31. Holík, F. & Neradova, S. (2017, May). Vulnerabilities of modern web applications. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (str. 1256–1261). Opatija: IEEE.
  32. Hussain, S., Keung, J., Sohail, M. K., Khan, A. A. & Ilahi, M. (2019). Automated framework for classification and selection of software design patterns. *Applied Soft Computing*, 75, 1–20.
  33. Huynh, T. T., Nguyen, T. D. & Tan, H. (2019, December). A Decentralized Solution for Web Hosting. In *2019 6th NAFOSTED Conference on Information and Computer Science (NICS)* (str. 82–87). Hanoi: IEEE.
  34. Hyun, C. Y. (2019). Design and Implementation of a Low-Code/No-Code System. *International journal of advanced smart convergence*, 8(4), 188–193.
  35. Ibrahim, M. (2019). Trends and Challenges in Requirement Analysis for; Modern Web Applications, Web Services, and Web of Things. *University of Sindh Journal of Information and Communication Technology*, 3(1), 55–63.
  36. IntelliJ (2020). Pridobljeno 5. junija 2020 iz <https://www.jetbrains.com/help/idea/discover-intellij-idea.html>
  37. IP GDPR (2020). Pridobljeno 5. junija 2020 iz <https://www.ip-rs.si/zakonodaja/reforma-evropskega-zakonodajnega-okvira-za-varstvo-osebnih-podatkov/najpogostejša-vprasanja-in-odgovori/>
  38. Jesse, N. (2019). *Agility eats Legacy—the Long Good-bye*. Bulgaria: Elsevier. 52(25).
  39. Khaliluzzaman, M. & Chowdhury, I. I. (2016, May). Pre and post controller based MVC architecture for web application. In *2016 5th International Conference on Informatics, Electronics and Vision (ICIEV)* (str. 552–557). Dhaka: IEEE.
  40. Khan, P. M. & Beg, M. S. (2013, April). Extended decision support matrix for selection of sdlc-models on traditional and agile software development projects. In *2013 Third International Conference on Advanced Computing and Communication Technologies (ACCT)* (str. 8–15). Rohtak: IEEE.

41. Khwaja, S. & Alshayeb, M. (2016). Survey on software design-pattern specification languages. *ACM Computing Surveys (CSUR)*, 49(1), 1–35.
42. Lemke, G. (2018). The Software Development Life Cycle and Its Application.
43. Linden, T., Khandelwal, R., Harkous, H. & Fawaz, K. (2020). The privacy policy landscape after the GDPR. *Proceedings on Privacy Enhancing Technologies*, 2020(1), 47–64.
44. López, A. J., Zorroza, I. L., Ruiz, E. S., Rodríguez-Artacho, M. & Gil, M. C. (2016, April). Design and development of a responsive web application based on scaffolding learning. In *2016 IEEE Global Engineering Education Conference (EDUCON)* (str. 308–313). Abu Dhabi: IEEE.
45. Malar, P. (2019). An Efficient Policy-Based Adaptation of Workflow Processes through MOVE Framework. *Journal of the Gujarat Research Society*, 21(17), 259–279.
46. Niazi, M., Mahmood, S., Alshayeb, M., Riaz, M. R., Faisal, K., Cerpa, N., Siffat Ullah K. & Richardson, I. (2016). Challenges of project management in global software development: A client-vendor analysis. *Information and Software Technology*, 80, 1–19.
47. Ottoni, G. (2018, June). HHVM JIT: A Profile-guided, Region-based Compiler for PHP and Hack. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation* (str. 151–165).
48. Pinto, G. & Castor, F. (2017). Energy efficiency: a new concern for application software developers. *Communications of the ACM*, 60(12), 68–75.
49. Ploder, C., Bernsteiner, R., Schlögl, S. & Gschliesser, C. (2019). The future use of LCNC Platforms by KW-an Acceptance Study.
50. Popat, S. & Starkey, L. (2019). Learning to code or coding to learn? A systematic review. *Computers & Education*, 128, 365–376.
51. Popova, O. (2019). Adaptation of flexible project management models based on Scrum and Kanban technologies. *Technology audit and production reserves*, 4(2 (48)), 4–10.
52. Producthunt (2020). Pridobljeno 12. junija 2020 iz <https://www.producthunt.com/posts/bubble-3>
53. Ratha, A. K., Sahu, S. & Meher, P. (2018). HTML5 in web development: A new approach. *International Research Journal of Engineering and Technology (IRJET)*, 5(1).
54. Rehman, S., Ullah, R. M. K., Tanvir, S. & Azam, F. (2018, November). Development of user interface for multi-platform applications using the model driven software engineering techniques. In *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)* (str. 1152–1158). Vancouver: IEEE.
55. Richardson, C. & Rymer, J. R. (2016). Vendor landscape: The fractured, fertile terrain of low-code application platforms. *FORRESTER, Janeiro*.

56. Ríos, J. M., Ordóñez, M. Z., Tapia, J. H. & Moreno, A. G. (2016). Analysis Methodologies Web Application Development. *International Journal of Applied Engineering Research*, 11(16), 9070–9078.
57. Rubio, D. (2017). *Beginning Django*. USA: Apress Berkeley CA.
58. Rymer, J. R., Koplowitz, R., Leaders, S. A., Mendix, K., are Leaders, S., ServiceNow, G. & Contenders, T. (2019). The Forrester Wave™: Low-Code Development Platforms For AD&D Professionals, Q1 2019.
59. Safa, N. S., Von Solms, R. & Furnell, S. (2016). Information security policy compliance model in organizations. *computers & security*, 56, 70–82.
60. Sanchis, R., García-Perales, Ó., Fraile, F. & Poler, R. (2020). Low-Code as Enabler of Digital Transformation in Manufacturing Industry. *Applied Sciences*, 10(1), 12.
61. Shahzad, F. (2017). Modern and responsive mobile-enabled web applications. *Procedia Computer Science*, 110, 410–415.
62. Shylesh, S. (2017). A study of software development life cycle process models. In *National Conference on Reinventing Opportunities in Management, IT, and Social Sciences* (str. 534–541).
63. Stackoverflow Survey. (2020). Pridobljeno 29. maja 2020 iz [https://insights.stackoverflow.com/survey/2019#technology\\_-\\_most-loved-dreaded-and-wanted-web-frameworks](https://insights.stackoverflow.com/survey/2019#technology_-_most-loved-dreaded-and-wanted-web-frameworks)
64. Stoica, M., Mircea, M. & Ghilic-Micu, B. (2013). Software Development: Agile vs. Traditional. *Informatica Economica*, 17(4).
65. Špundak, M. (2014). Mixed agile/traditional project management methodology—reality or illusion?. Croatia: Elsevier. *Procedia-Social and Behavioral Sciences*, 119, 939-948.
66. Tajalizadehkhoob, S., Korczyński, M., Noroozian, A., Ganán, C. & van Eeten, M. (2016, April). Apples, oranges and hosting providers: Heterogeneity and security in the hosting market. In *NOMS 2016–2016 IEEE/IFIP Network Operations and Management Symposium* (str. 289–297). Istanbul: IEEE.
67. Tasci, T., Melcher, J. & Verl, A. (2018, June). A container-based architecture for real-time control applications. In *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)* (str. 1–9). Stuttgart: IEEE.
68. Tisi, M., Mottu, J. M., Kolovos, D., de Lara, J., Guerra, E., Di Ruscio, D., Alfonso, P. & Wimmer, M. (2019, July). Lowcomote: Training the Next Generation of Experts in Scalable Low-Code Engineering Platforms.
69. Urma, R. G. (2017). *Programming language evolution* (No. UCAM-CL-TR-902). University of Cambridge, Computer Laboratory.
70. Virta, T. (2018). Relation of low-code development to standard software development: Case Biit Oy.
71. Wakil, K. & Jawawi, D. N. (2018, January). A new adaptive model for web engineering methods to develop modern web applications. In *Proceedings of the 2018 International Conference on Software Engineering and Information Management* (str. 32–39).

72. Waszkowski, R. (2019). Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine*, 52(10), 376–381.
73. Webflow introduction (2020). Pridobljeno 15. junija 2020 iz <https://university.webflow.com/category/introduction>