Analiza i usporedba suvremenih metodologija razvoja informacijskih sustava

Kazalac, Sandra

Master's thesis / Diplomski rad

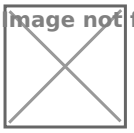
2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:824350

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

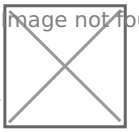
Download date / Datum preuzimanja: **2024-03-29**



Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula





Sveučilište Jurja Dobrile u Puli
Fakultet ekonomije i turizma
„Dr. Mijo Mirković“

SANDRA KAZALAC

ANALIZA I USPOREDBA SUVREMENIH METODOLOGIJA RAZVOJA INFORMACIJSKIH SUSTAVA

Diplomski rad

Pula, 2019.

Sveučilište Jurja Dobrile u Puli
Fakultet ekonomije i turizma
„Dr. Mijo Mirković“

SANDRA KAZALAC

ANALIZA I USPOREDBA SUVREMENIH METODOLOGIJA RAZVOJA INFORMACIJSKIH SUSTAVA

Diplomski rad

JMBAG: 349-ED, izvanredna studentica
Studijski smjer: Poslovna informatika

Predmet: Razvoj informacijskih sustava
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijski sustavi
Znanstvena grana: Poslovna informatika

Mentor: prof. dr. sc. Vanja Bevanda

Pula, siječanj 2019.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisana Sandra Kazalac, kandidat za magistra ekonomije/poslovne ekonomije ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, 08.01., 2019. godine

IZJAVA

o korištenju autorskog djela

Ja, Sandra Kazalac, dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom: Analiza i usporedba suvremenih metodologija razvoja informacijskih sustava koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 08.01.2019. (datum)

Potpis

_____

SADRŽAJ

SAŽETAK
ABSTRACT
1. UVOD 1
2. INFORMACIJSKI SUSTAV 3
3. POSLOVNI INFORMACIJSKI SUSTAVI 14
3.1. Sustavi za upravljanje organizacijskim resursima 14
3.2. Sustavi za upravljanje lancima opskrbe 15
3.3. Sustavi za upravljanje odnosima s klijentima 16
3.4. Sustavi za procesiranje transakcija 17
3.5. Sustavi za upravljanje informacijama 18
3.6. Sustavi za potporu u odlučivanju 19
3.7. Sustavi za upravljanje znanjem 21
3.8. Ekspertni sustavi 23
3.9. Sustavi za potporu izvršnoj razini menadžmenta 24
4. METODOLOŠKI OKVIRI RAZVOJA SUSTAVA 25
4.1. Metodologija razvoja informacijskih sustava 25
4.2. Tradicionalni razvoj sustava 29
4.3. Modeliranje procesa 32
4.4. Modeliranje podataka 35
4.5. Inkrementalni razvoj 39
4.6. Iterativni razvoj 40
4.7. Prototipiranje 41
4.8. Spiralni model 42
5. OBJEKTNO - ORIJENTIRANI PRISTUP RAZVOJU IS-a 44
5.1. Objektno orijentirani pristup 44
5.2. Unified Modelling Language 46
5.2.1. Use Case Dijagram 46
5.2.2. Dijagram klasa 48
5.2.3. Sekvencijalni dijagram 51
5.2.4. Dijagram stanja 53
5.2.5. Dijagram aktivnosti 54

5.3. Rational Unified Process 56
6. ZAJEDNIČKI RAZVOJ APLIKACIJA 59
7. BRZI RAZVOJ APLIKACIJA 61
8. AGILNE METODE RAZVOJA SUSTAVA 65
8.1. Agilni pristup 65
8.2. Kristalne metode 67
8.3. Dinamička metoda razvoja sustava 69
8.4. Scrum 73
8.5. Razvoj temeljen na funkcionalnostima 74
8.6. Ekstremno Programiranje 76
8.7. Adaptivni razvoj sustava 79
8.8. Lean Razvoj 81
9. METODE ZA RAZVOJ IS-a TEMELJENIH NA WEB-u 85
9.1. Ciklus razvoja IS-a temeljenih na web-u 85
9.2. WebML 86
9.3. Razvoj temeljen na komponentama 88
10. CASE ALATI 91
11. USPOREDBA I ANALIZA SUVREMENIH METODA RAZVOJA 95
11.1. Podrijetlo 96
11.2. Resursi 100
11.3. Faze 101
11.4. Prakse 102
11.5. Alati 104
11.6. Izbor metode 109
11.7. Uspješnost projekata IS-a 114
12. ZAKLJUČAK 118
13. LITERATURA 120

SAŽETAK

Radom se teorijski razrađuju informacijski sustavi i razvoj informacijskih sustava s fokusom na agilne metode. Pronalaze se zajedničke karakteristike agilnih metoda kroz tablične prikaze kako bi se kroz istraživanje utvrdila mogućnost izvršenja komparativne analize s ciljem omogućavanja odabira metode razvoja. Upućuje se na prednosti i nedostatke te statističke pokazatelje o uspješnosti primjene metoda radi ukazivanja na potencijalna očekivanja ishoda primjene odabrane/odabranih metoda.

Ključne riječi: *informacijski sustavi, poslovni informacijski sustavi, metodologije razvoja sustava, modeliranje, UML, agilne metode, CASE alati, usporedba metoda razvoja, uspješnost metoda razvoja*

ABSTRACT

The thesis is a theoretical consideration of the field of information systems and information systems development with focus on agile methods. It presents agile methods' common properties and features through data tables research in order to establish the possibility of conducting a comparative analysis aimed at facilitating the selection of a development method. It points out the distinctive individual method advantages and disadvantages and statistical indicators of method application success in order to establish the potential outcome expectation of chosen method application.

Keywords: *information systems, business information systems, systems development methodologies, modelling, UML, agile methods, CASE tools, development methods comparison, development methods success*

1. UVOD

Razvoj novih metoda neizbježna je potreba budućnosti, znak je napretka i novih mogućnosti. Ljudi pronalaze primjenu informacijskih sustava na sve više područja pa je jasno kako se područje razvoja informacijskih sustava moralo dinamično mijenjati kako su specifični sustavi i nove tehnologije mijenjali tehnološku okolinu.

Ovim radom je obuhvaćena razrada teorijskih cjelina bitnih za razvoj informacijskih sustava kroz 13 cjelina, počevši od samih osnova. Nakon uvodnog dijela se u drugoj cjelini razmatraju osnovni pojmovi - odgovori na pitanja: što su podaci, informacije, sustav u općem smislu te sustav u informacijskom smislu? Zatim se kroz treću cjelinu razrađuju pojmovi poslovnih informacijskih sustava – vrste sustava s obzirom na njihovu primjenu.

Metodološki okviri razvoja sustava kao četvrta cjelina obuhvaćaju faze razvoja, modeliranje (koje je okosnica faze analize i, podrazumijevajući idealnu izvedbu, preteča dobrog dizajna) i rane metodologije. Faze razvoja su različito izvedive i različito zastupljene u pojedinim metodologijama koje čine bitnu tematsku cjelinu i koje su predstavljene kategorički s obzirom na njihov nastanak u vremenskim okvirima pa su, u skladu s tim, razrađene: tradicionalne metodologije – model vodopada, V model, prototipiranje i spiralni model, a zatim, s obzirom na pristup i objektno orijentirani pristup razvoju kroz petu cjelinu, zajednički razvoj kroz šestu i brzi razvoj aplikacija kroz sedmu cjelinu.

Osma cjelina bavi se agilnim metodama koje su fokus ovog rada. Agilne metode - poput Scrum-a ili Ekstremnog programiranja - predstavljaju skupinu metoda koje su nastale iz različitih specifičnih potreba, a na nekim objedinjenim principima. Komparativna analiza na tom području može se pokazati izazovnom stoga što, za razliku od tradicionalnih metodologija, ovdje ne postoje olakotne okolnosti formalizacije i standardizacije – ništa nije propisano koliko je predloženo, a čak i pri tom, primjenjuju se različite prakse, fokusi, pokrivenost ciklusa, zahtjevi za vještinama i resursima, itd.

Komparativna analiza metoda/metodologija čini istraživački dio ovog rada no prije upuštanja u istraživački dio u jedanaestoj cjelini također se još kroz devetu cjelinu kratko razmatraju metode za razvoj informacijskih sustava temeljenih na web-u te kroz desetu cjelinu – CASE alati u teoriji - dok ih se u praksi dijelom dotiče i u jedanaestoj cjelini.

Za jedanaestu cjelinu bitno je istaknuti da je hipoteza ovog rada da je usporedba

suvremenih metoda moguća, pri čemu se podrazumijevaju teze da agilne metode dijele neke zajedničke karakteristike u odabranim odrednicama. Korištena metoda je komparativna analiza tabličnom usporedbom s uvrštavanjem relevantnih podataka pojedinih metodologija ili metoda po nekim odrednicama, pri čemu se pokušavaju ustanoviti zajedničke karakteristike. Analitički okvir usporedbe koji predstavlja metodu sačinjen je od pet glavnih odrednica, a to su: podrijetlo, resursi, faze, prakse i alati. Za svaku odrednicu stoji teza mogućnosti usporedbe iznalaženjem zajedničkih karakteristika da bi se, u konačnici uz pomoć njih, potvrdila ili opovrgla hipoteza. Na kraju cjeline je pružen uvid u statističke podatke o izboru metoda, percipiranim prednostima i preprekama te uspješnosti projekata IS-a.

Cilj rada je komparativnom analizom omogućiti odabir metode razvoja s obzirom na različite potrebe u domenama odabranih odrednica komparativne analize.

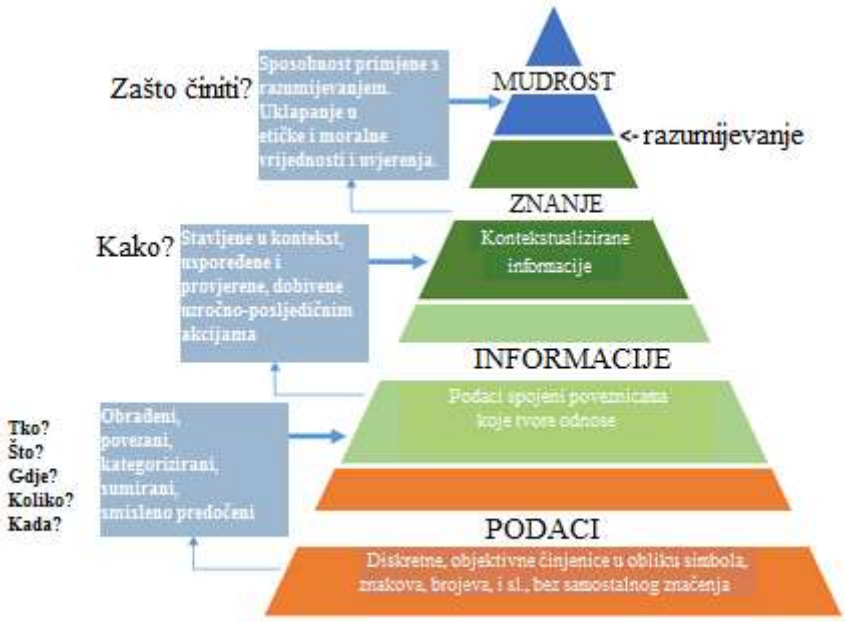
Potvrđenost ili opovrgljivost hipoteze rada s obzirom na razradu jedanaeste cjeline predložena je u zaključnom dijelu rada koji čini dvanaestu cjelinu.

U konačnici su, kroz trinaestu cjelinu, podastrijeti na raspolaganje resursi u literaturi kojima se u ovom radu koristilo kako bi se postigla potrebna iscrpnost podataka i informacija.

2. INFORMACIJSKI SUSTAV

Informacija je jedan od temeljnih čimbenika funkcionalnog djelovanja svih društvenih struktura i organizacija te se smatra resursom bez kojeg većina ljudskih nastojanja i poduhvata ne može funkcionirati u svrhu napretka, rasta i razvoja. Poslovanje je u modernim organizacijama nezamislivo bez sveprožimajućih informativnih kanala.

Informacija se uklapa u jedan skalabilni okvir koji ovisi o stupnju apstrakcije, a može se predočiti DIKW piramidom. DIKW je akronim za „***Data – Information – Knowledge – Wisdom“***, to jest Podatak-Informacija-Znanje-Mudrost, a sam naziv ukazuje na četiri čimbenika koji čine centralnu tematiku modela i tvore hijerarhiju u obliku piramide kao što je vidljivo na slici br. 1. Za potrebe rada, DIKW model je uvršten stoga što pruža okvir za definiranje pojmova koji također čine temelj proučavanja informacijskih sustava i stoga što se model piramide prikladno podudara s piramidalnim modelom razina menadžmenta u organizaciji s kojim ćemo paralele povući u sljedećem potpoglavlju i kasnije kada bude bilo riječi o sustavima za potporu u odlučivanju.



Slika 1: DIKW Piramida;
(Izvor: https://www.climate-eval.org/sites/default/files/images/blog/Data-information-knowledge-wisdom.png (datum pristupa: 13.02.2017.)

Prvu razinu DIKW priamide čine podaci. ***Podaci*** se manifestiraju kao simboli ili znakovi, a predstavljaju podražaje, signale ili činjenična svojstva objekata, događaja i

okoline. (Ackoff, 1989) Simboli uključuju: riječi (tekstualne i verbalne), brojeve, dijagrame i slike (pomične i nepomične) koji čine sastavne dijelove komunikacije. Signali uključuju: osjetilne interpretacije svjetla, zvuka, dodira, itd. (Liew, 2007.)

Tablica 1: Načini manifestiranja podataka

PODACI	PRIKAZANI KAO:
alfanumerički	brojevi, slova i ostali znakovi
slikovni	grafike, slike
audio	tonovi, zvukovi
video	pomične slike, film

(Izvor: Stair i Reynolds, 2010.)

Podatak ili skup podataka koji su nekom obradom u smislu primjene logičkih poveznica dobili neko značenje ili vrijednost predstavljaju ***informaciju***, koja čini drugu razinu piramide. Informacija je koncept koji se možda čini jednostavnim ali se zapravo opisuje iz različitih perspektiva pa tako npr. postoji definicija iz četrdesetih godina koju je iznio matematičar Claude Shannon, a koji je ustvrdio da je informacija ono što smanjuje neizvjesnost. (Stair, Reynolds i Chesney, 2012.) Još jedna definicija informaciju opisuje kao podatak koji primatelju posreduje neku relevantnu novost te, za tog konkretnog primatelja, „umire“ isti tren kada je dobivena stoga što svojstvo novosti može imati samo u tom trenutku. (Radovan, 1992.) Pojam informacije je zapravo kompleksan i stoga što se često kao sinonimi za informaciju u raznoj literaturi koriste i pojmovi poput činjenice, podatka, znanja itd. nastaju razne nedoumice oko prirode pojma. Informacija najčešće odgovara na pitanja: tko, što, kada, koliko?...

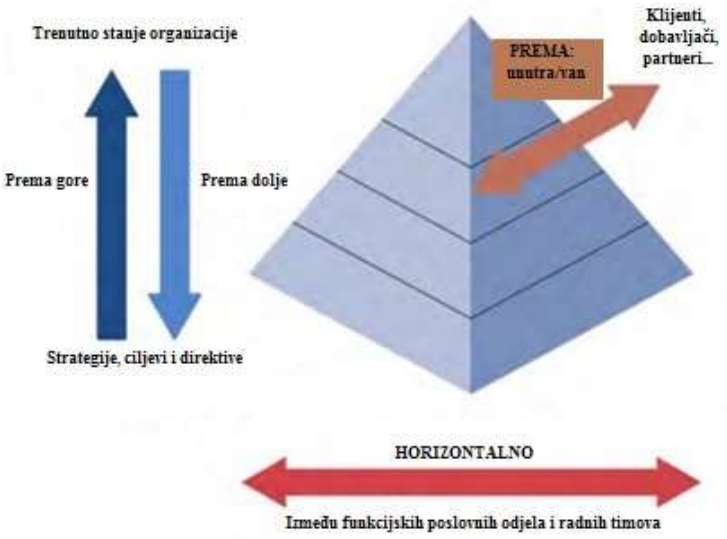
Znanje, treća razina piramide, jednako tako nailazi na različita određenja s obzirom na razinu apstrakcije pojma. Znanje je potrebno kako bi podaci mogli dobiti svoje značenje i pretvoriti se u informaciju jer značenje nastaje primjenom konteksta, a za kontekst je imperativ znanje – znanje koje poveznice primijeniti na podatke da dobijemo korisne informacije. (Valacich i Schneider, 2010.) „Znanje je isprepleteni skup teorija, iskustva, vrijednosti, kontekstualnih podataka, stručnog uvida i utemeljene intuicije kojim se ostvaruju okviri za donošenje procjena i integracije novih iskustava i informacija.“ (Rowley, 2007.) Jedna opća podjela znanja je ona na eksplicitno i prešutno znanje prema kojoj je eksplicitno ono znanje koje je moguće artikulirati, pohraniti, demonstrirati i distribuirati dok je prešutno ono koje se podsvjesno razumije

i primijenjuje te je najčešće rezultat iskustva. (Sunasee & Sewry, 2002) U organizacijama se znanje ugrađuje u organizacijske rutine, procese, prakse i norme. Znanje najčešće odgovara na pitanje kako i bitno je za donošenje odluka.

Četvrta i konačna najapstraktnija razina piramide je ***mudrost***. Službena grana psihologije mudrost općenito opisuje kao naprednu kognitivnu i emocionalnu dimenziju svijesti koja se razvija s iskustvom i vremenom. U organizacijskim okvirima ona predstavlja sposobnost da se poveća učinkovitost, doda neka vrijednost te sposobnost etičke i estetske prosudbe koje su svojstvene isključivo čovjeku, a rezultat su individualnih uvjerenja, kulture, znanja, principa itd. (Rowley, 2007.) Mudrost podrazumijeva sposobnost uviđanja „šire slike" i predviđanja budućih događaja te, u skladu s tim, donošenja „najboljih praksi". (Valacich i Schneider, 2010.)

Informacijski resursi se unutar organizacije po DIKW modelu mogu prometnuti u više razine piramide, pa se tako može razmatrati i resurs znanja, a rjeđe i mudrosti. Za potrebe rada relevantno je pak zadržati se pri konceptu informacijskih resursa.

Informacije, s obzirom na organizacijsku strukturu, imaju svoje tokove kojima se distribuiraju kako bi omogućile normalno funkcioniranje organizacije, a teku u četiri smjera: prema gore, prema dolje, vodoravno, prema unutra i prema van. U tradicionalnom smislu, organizacijske razine se mogu prikazati modelom piramide. Četiri razine piramide predstavljaju jednu razinu koja predstavlja pomoćno osoblje pri dnu i tri razine menadžmenta: operativnu, stratešku i izvršnu (navedenim redoslijedom).



Slika 2. Tok informacija kroz organizaciju;
(Izvor: https://goo.gl/images/Mdm8Q7, datum pristupa:13.02.2017.)

- Pomoćno osoblje bavi se svakodnevnim osnovnim transakcijskim aktivnostima kao što su obrada narudžbi, proizvodnja roba i usluga i posluživanje kupaca.
- **Izvršni menadžment** predstavlja najvišu razinu menadžmenta i bavi se upravljanjem i odlučivanjem o pitanjima koja se tiču organizacije u cjelini kao npr. donošenjem ciljeva
  i strategija.
- **Strateški menadžment** se bavi upravljanjem i odlučivanjem povezanim s postizanjem ciljeva i strategija utvrđenih od strane izvršnog menadžmenta i koordinira operativnom razinom.
- **Operativni menadžment** upravlja svakodnevnim operacijama pomoćnog osoblja.

S obzirom na tok informacija postoje:

- **Informacije koje teku uzlazno** opisuju trenutno stanje organizacije na temelju dnevnih transakcija. Kada dođe do prodaje, primjerice, informacija o prodaji teče od pomoćne razine i proslijeđuje prema gore gdje se konsolidira informacijskom tehnologijom i proslijeđuje ponovno gore donositeljima odluka kako bi bila uzeta u obzir, to jest uključena u statistike bitne pri donošenju odluka.
- **Silazni tok informacija** sastoji se od strategija, ciljeva i smjernica koje potječu s vrha i proslijeđuju se nižim razinama.
- **Informacije koje teku vodoravno** proslijeđuju se između funkcijskih poslovnih jedinica i odjela.
- Konačno, **prema van i prema unutra** teku informacije prema i od kupaca, dobavljača, distributera i drugh partnera izvan organizacije u svrhu izvršavanja poslovnih aktivnosti. (Haag i Cummings, 2013.)

S obzirom na prirodu informacija uz pomoć kojih organizacije ostvaruju svoje ciljeve, informacije se mogu podijeliti na četiri temeljne skupine, a to su:

- **Opisne informacije** - odgovaraju na pitanje "što je?" opisuju stanje poslovanja u određenom trenutku. Bitne su za menadžment jer uključuju informacije poput financijskih rezultata, proizvodnih zapisa, rezultata testova, marketinga i evidencija održavanja.
- **Dijagnostičke informacije** - odgovaraju na pitanje "što nije u redu?" u trenutnom stanju poslovanja što se identificira usporedbom onoga kako jest i

onoga kako bi trebalo biti, to jest identificiranjem jaza. Dijagnostička informacija može se koristiti za utvrđivanje problema koji se razvijaju u poslovanju. Menadžment mora pružiti norme i standarde koji će, u usporedbi sa trenutnim stanjem poslovanja, otkriti problematična područja.

- **Prediktivne informacije** - odgovaraju na pitanje "što ako?" Ove informacije se dobivaju analizom mogućih budućih događaja. Prediktivnom analizom se dobivaju informacije
  kojima se ili utvrđuju trenutni problemi ili izbjegavaju budući.
- **Preskriptivne informacije** - odgovaraju na pitanje "što treba učiniti?". Prediktivne informacije nisu dovoljne za donošenje odluka i uz njih moraju postojati preskriptivne informacije. Evaluacija predviđenih ishoda u usporedbi s ciljevima i vrijednostima pruža temelje za donošenje odluka. (Harsh, Connor i Schwab, 1981.)

Osim prirode informacija kojima organizacija raspolaže, bitne su i neke njihove karakteristike koje utječu na ostvarivanje koristi od njihove primjene. Jednako kao što informacijski resursi mogu predstavljati način postizanja konkurentske prednosti, također mogu i stvoriti otegotne okolnosti i probleme. Vrijednost informacije je izravno povezana s razmjerima u kojima pomaže donositeljima odluka u postizanju organizacijskih ciljeva, a ona mora udovoljavati kriterijima:

- Pristupačnosti - informacijama bi trebali lako moći pristupiti ovlašteni korisnici i preuzeti ih u primjerenom formatu točno u vrijeme kada im je to potrebno.
- Točnosti – točna informacija ne sadrži greške. Ponekad je dobiveni rezultat netočna informacija stoga što su unešeni podaci za obradu bili netočni.
- Potpunosti – potpuna informacija sadržava sve važne podatke.
- Ekonomičnosti – informaciju bi trebalo biti relativno ekonomično za proizvesti. Donositelji odluka uvijek moraju blansirati između vrijednosti informacije i troška njezina dobivanja.
- Fleksibilnosti – fleksibilnu informaciju se može koristiti u različite svrhe, za više mogućih koristi.
- Relevantnosti – relevantna informacija jest ona koja je bitna donositelju odluka, to jest ona koju treba uzeti u obzir.
- Pouzdanosti – pouzdana je ona informacija kojoj njezini korisnici mogu vjerovati, a ona ovisi o izvorima i metodama prikupljanja i obrade podataka.

- Sigurnosti – zaštićenost informacije od neovlaštenih korisnika.
- Jednostavnosti – informacija može biti složena ili preopsežna i u tom slučaju može negativno djelovati na proces donošenja odluka. Jednostavnost podrazumijeva konsolidiranje podataka na način da dobivena informacija bude jasna primatelju i da se njome prenese relevantna suština.
- Pravovremenosti – informacija koja je dostavljena točno kada je potrebna s obzirom na neki budući događaj ili problem.
- Provjerljivosti – informacija koju je moguće provjeriti na izvoru.

Informacija sama po sebi može imati vrijednost u smislu da može biti objektom djelatnosti, to jest predmetom poslovanja (Stair i Reynolds, 2010.), a poslovanje je danas obično poduprto informacijskim sustavima ili sustavom.

Uzevši sustav kao općeniti pojam, može se reći da ga se nalazi u suštini svakog fenomena. Sve što je moguće opažati obično se može i smjestiti u jednu ili više kategorija sustava. Gotovo svaki sustav je dijelom nekog većeg sustava osim teoretski beskonačno velikog, pa je svaki sustav obično ujedno i podsustav. Isto tako, svaki sustav, osim teoretski beskonačno malog, ima podsustav. (System BD, n.d.)

Kao znanstveno područje istraživanja, sustav se može promatrati i izučavati u općem okviru teorije sustava i znanosti o sustavu te u užem kontekstu različitih kategorija sustava, kao što su npr. sustav bioraznolikosti, gospodarski sustav ili informacijski sustav - koji je fokus ovog rada.

Prema etimološkom rječniku, riječ „sustav“ potječe od grčke riječi „ systema“, što znači „organizirana cjelina ili cjelina sastavljena od djelova“ (System OED, n.d.), dok poslovni rječnik pojam sustava definira kao svrhovito organiziranu strukturu koja se sastoji od međupovezanih i međuzavisnih elemenata (komponenti, entiteta, čimbenika, članova, dijelova i sl.) koji (izravno ili neizravno) vrše utjecaj jedni na druge kako bi se održao kontinuitet odvijanja neke specifične radnje ili funkcije i, posljedično tome, postojanja sustava u cjelini, s ciljem ostvarivanja specifične svrhe sustava. (System BD, n.d.) Pojam sustava također se može odnositi na skup pravila, procedura, regulativa, uvjeta i procesa kojima se uvjetuje neko ponašanje ili struktura. (Baianu, 2011.)

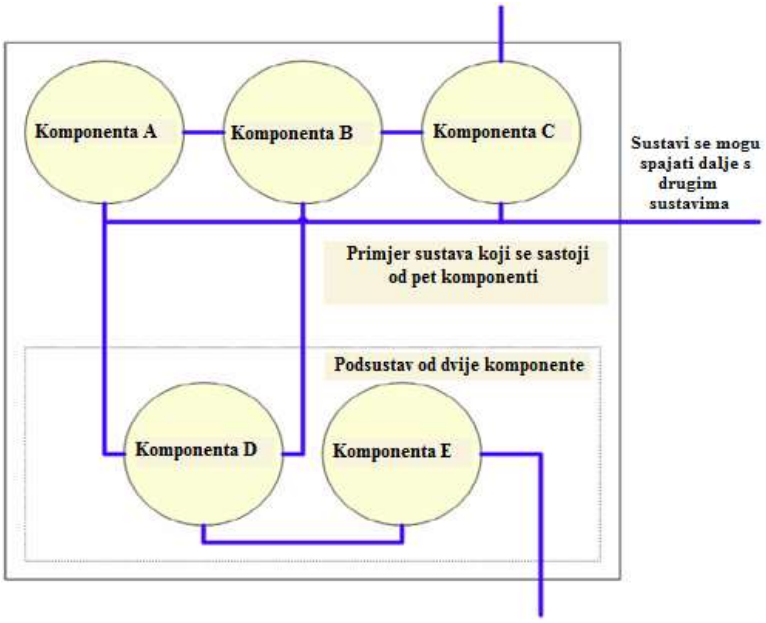
Postoje različite općenite podjele sustava s obzirom na njihovu prirodu kao što su to npr. umjetno i prirodno stvoreni sustavi, otvoreni i zatvoreni sustavi te statički i dinamički sustavi.

- Pod **umjetno stvorenim sustavima** podrazumijevaju se one koje su uspostavili ljudi, a ljudi se koriste sustavima već tisućama godina, od starih civilizacija i jednostavnijih metoda pa do današnjih računalno potpomognutim kompleksnim sustavima.
- Između **otvorenih i zatvorenih sustava** razlika je u tome što otvoreni sustavi vrše nekakvu interakciju ili izmjenu sa svojom okolinom ili drugim sustavom dok su zatvoreni sustavi izolirani.
- Između **statičkih i dinamičkih sustava** razlika je u tome što dinamičke definira nekakav proces transformacije koji je promjenjiv kroz vrijeme dok statički sustavi nemaju vremensku dimenziju već se njihovo stanje odražava kao konstanta, a ako promjena nastaje, nastaje trenutačno s obzirom na nekakav input. (Baianu, 2011.)

Sustavi i njihove komponente mogu biti **fizički i stvarni ili neopipljivi i apstraktni**.

- **Neopipljiv sustav** može biti npr. kulturološki koji se sastoji od ideja, vrijednosti, vjerovanja, etičkih i moralnih načela i sl.
- **Fizički, stvarni sustav** ili nekakav hipotetski sustav može se prikazati apstrahiranjem i modeliranjem, to jest prikazati koristeći simbole u obliku znakova, slika, riječi itd. kojima se mogu predstaviti sve ili samo neke njegove karakteristike.

Slika 4. je prikaz nekog sustava na kojem je vidljivo da se radi o sustavu sačinjenom od pet komponenti (A,B,C,D i E) od kojih dvije (D i E) čine podsustav tog sustava. Definirane su veze i povratne veze (tzv. „feedback mehanizmi“ koji će biti definirani u nastavku) između komponenti i tri ulaza/izlaza sustava, od kojih je jedan iz podsustava. Utvrđen je djelokrug sustava definiranom granicom koju predstavlja pravokut kojim je omeđen skup povezanih komponenti.



Slika 3. Apstraktni prikaz nekog sustava;
(Izvor: Watson, 2007.)

Ovakvi prikazi omogućavaju i olakšavaju razumijevanje načina funkcioniranja sustava što je od izrazite važnosti za analizu i dizajn sustava. (Baianu, 2011.) U konačnici, na temelju prikazanog, karakteristike sustava može se svesti na sljedećih nekoliko odrednica:

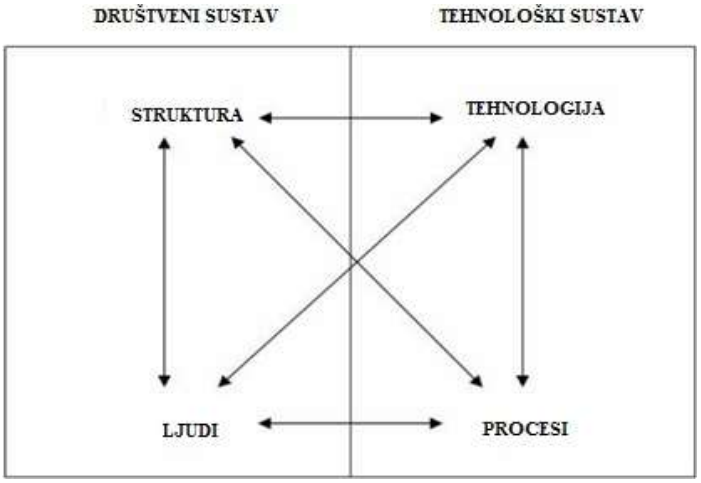
- Komponente koje sačinjavaju sustav.
- Veze kojima su definirane interakcije i međuzavisnosti
- Organizaciju kojom je definirana struktura i integracija
- Emergentna svojstva tj. svojstva ili funkcije koje se manifestiraju samo kada je sustav cjelovit a ne posjeduje ih ni jedna komponenta individualno. (Emergentna svojstva, n.d.)
- Granicu koja definira područje djelovanja sustava
- Okolinu kojom je sustav okružen
- „Feedback“ mehanizme koji su između komponenti prikazani kao povratne veze, a radi se o uspostavljenom odnosu koji generira uzročno posljedičnu petlju u kojoj ponašanje prve komponente utječe na ponašanje druge, da bi ponašanje druge zatim, povratnom vezom, utjecalo na ponašanje prve itd. u kružnom kontinuitetu. (Aström i Murray, 2009.) „Feedback“ mehanizam ipak u terminima informacijskih sustava također može imati i drugo značenje, a toga ćemo se dotaknuti u sljedećem potpoglavlju.

- Ograničenja, kao npr. u terminima kapaciteta ili brzine. Ograničenja mogu biti svojstvena samom sustavu ili biti nametnuta od okoline.
- Sučelja koja predstavljaju područja interakcije sustava s okolinom
- Unose, tj. inpute, koje sustav prima od okoline (npr. materija, energija, podatak)
- Rezultate koje sustav „vraća“, tj. outpute, kao produkt izvršavanja svojih funkcija, to jest transformacije unosa, okolini
- Svrhu, kao razlog postojanja sustava (Valacich, George i Hoffer, 2011.)

Sustavi prestaju funkcionirati ili kada se iz njih ukloni neki element, pogotovo element nužan za izvršavanje osnovnih funkcija ili kritičnih procesa ili kada se neki element promijeni u mjeri dovoljno značajnoj da utječe na rezultat izvršenja funkcije sustava. (System BD, n.d.)

Informacijski sustav je sastavni, neodvojivi dio svakog upravljačkog i ciljno orijentiranog sustava. Njegova funkcija je da permanentno opskrbljuje potrebnim informacijama sve razine upravijanja i odlučivanja u danom tehnološkom, odnosno organizacijskom sistemu. (Radovan, 1992.) stoga se za informacijski sustav može reći da podrazumijeva organiziranje ljudi, podataka, procesa i informacijske tehnologije u međusobnu svrhovitu interakciju iz koje se prikupljanjem, unosom, obradom i pohranom podataka i informacija te generiranjem korisne informacije za organizaciju, ostvaruje neki postavljeni zadatak ili cilj organizacije. (Stair i Reynolds, 2010.)

Iz ove definicije može se i ekstrapolirati četiri temeljna čimbenika koja sačinjavaju informacijski sustav, a to su društvena i organizacijska struktura, ljudi, procesi i informacijsko komunikacijske tehnologije. Da bi sva ova četiri elementa optimalno funkcionirala kao cjelina bitno je da postoje i određene procedure, propisi, norme itd. Također, svaki čimbenik utječe na sve ostale i stoga ne može biti da se promjena jednog čimbenika ne odražava na ostale. Ovo je opis sociotehnološkog informacijskog sustava koji se može prikazati modelom pod nazivom Leavittov dijamant, prikazan slikom br. 4. (Cornford i Shaikh, 2013.)



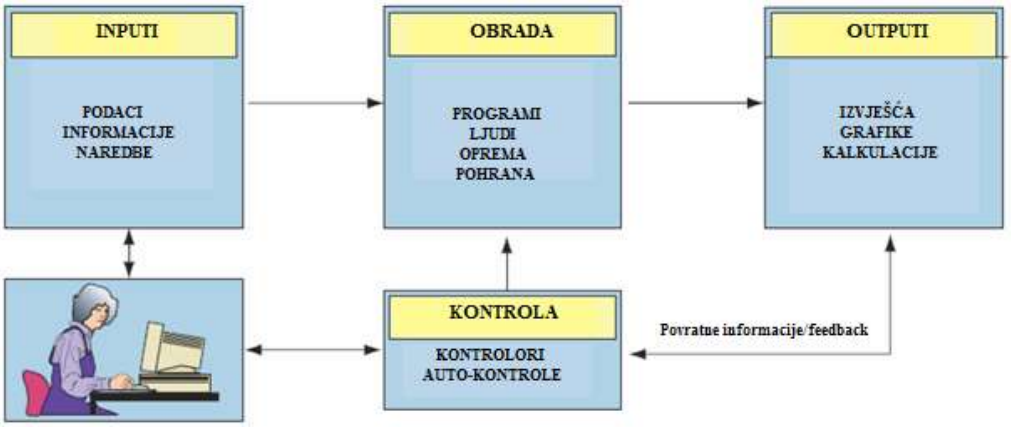
Slika 4. Leavittov dijamant;
(Izvor: Cornford i Shaikh, 2013.)

Međudjelovanjem ovih čimbenika ostvaruje se svrha informacijskog sustava, a to je stvaranje i opskrba informacijama. Prethodno je razlučeno da informaciju dobivamo obradom podataka. Podaci za informacijski sustav predstavljaju input. Taj input mora proći neki proces obrade. Priroda i vrijednost informacija dobivenih stavljanjem podataka u kontekst ovisi o vezama definiranim između podataka. Pretvaranje podataka u informaciju je proces, skup logički povezanih zadataka izvedenih kako bi se dobio neki željeni rezultat. Proces određivanja veza ili relacija između podataka kako bi se dobilo korisnu informaciju zahtjeva znanje. Znanje je svjesnost i razumijevanje skupa informacija i načina na koje se informacije može učiniti korisnom u potpori obavljanja nekog zadatka ili donošenju neke odluke. Neki primjeri procesa obrade podataka jesu:

- **Klasifikacija:** podrazumijeva kategorizaciju i dodijeljivanje podataka u kategorije
- **Preraspoređivanje, sortiranje**: podrazumijeva organizaciju podataka tako da se prikladne stavke grupiraju zajedno ili poredaju određenim redoslijedom.
- **Agregacija**: podrazumijeva svođenje podataka na nekakve veličine - npr. prosjeke, postotke, sume itd.
- **Izvođenje izračuna**: npr. obračun plaća zaposlenika s obzirom na satnicu.
- **Selekcija**: podrazumijeva odabir ili odbacivanje pojedinačnih podataka na temelju nekih kriterija. (Bocij, Greasley i Hickie, 2003.)

Informacija predstavlja output informacijskog sustava, u organizacijskim terminima to obično može biti u vidu dokumenata, izvještaja itd. Ponekad informacija, kao output jednog sustava, može ponovno postati inputom istog ili drugog sustava.

Proces dobivanja informacija (Slika 6)., kao i proizvodnja bilo kojeg proizvoda, može se pokazati zadovoljavajućim ali i problematičnim na neki način. Može se također dogoditi da nije problem u procesu već u inputima. Također može biti da dobivena informacija ukazuje na potrebu izmjene procesa ili inputa. Potreba za unošenjem bilo kakvih izmjena u proces, ukoliko ona postoji, postaje jasna radi mehanizama kontrole i feedback-a sustava, to jest povratne informacije. U informacijskim sustavima, feedback predstavlja informaciju dobivenu od sustava koja se iskorištava kako bi se napravile određene izmjene u inputima ili aktivnostima procesiranja. (Stair i Reynolds, 2010.) No ona može jednostavno biti i izvještavanje o performansama sustava.



Slika 5. Proces generiranja informacija;
(Izvor: http://www.uotechnology.edu.iq/ce/Lectures/SarmadFuad-MIS/MIS_Lecture_3.pdf , datum pristupa: 24.06.2018.)

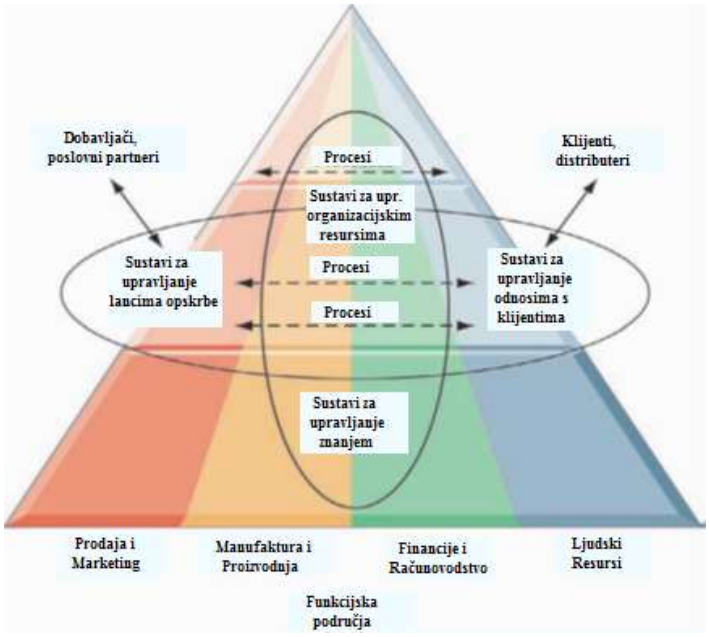
Kontrola sustava je funkcija koja ima nadzornu funkciju, a analizira povratne informacije kako bi se utvrdilo da li sustav ispunjava svoju funkciju kako treba. S obzirom na feedback, kontrolni mehanizmi tada, ako je potrebno i ukoliko su samo-regulirajući, prilagođavaju inpute i/ili elemente obrade kako bi sustav proizveo odgovarajući output. (Bocij, Greasley i Hickie, 2003.)

3. POSLOVNI INFORMACIJSKI SUSTAVI

Informacijski sustavi pružaju potporu i mogućnost izvršavanja nekih poslovnih procesa, komunikacije i koordinacije između različitih funkcionalnih područja te kontrole i upravljanja nad performansama procesa. (Rainer, Prince i Cegielski, 2014.) Poduzeća svih veličina, razne organizacije - profitne i neprofitne, vlade, sveučilišta itd. oslanjaju se na informacijske sustava u postizanju strateških ciljeva. Ovi sustavi imaju kritične uloge u raznim područjima poslovanja koja će biti pojedinačno obrađena ovim poglavljem.

3.1. Sustavi za upravljanje organizacijskim resursima

Ovi sustavi predstavljaju skup integriranih programa kojima se provode i automatiziraju bitne poslovne operacije za cijelu organizaciju, a mogu i prelaziti okvire organizacije. ERP sustav može zamijeniti mnoge aplikacije sjedinjenim skupom programa, čineći sustav učinkovitijim, jeftinijim i lakšim za korištenje.



Slika 6. Integrirani ERP sustav;
(Izvor: Laudon i Laudon, 2014.)

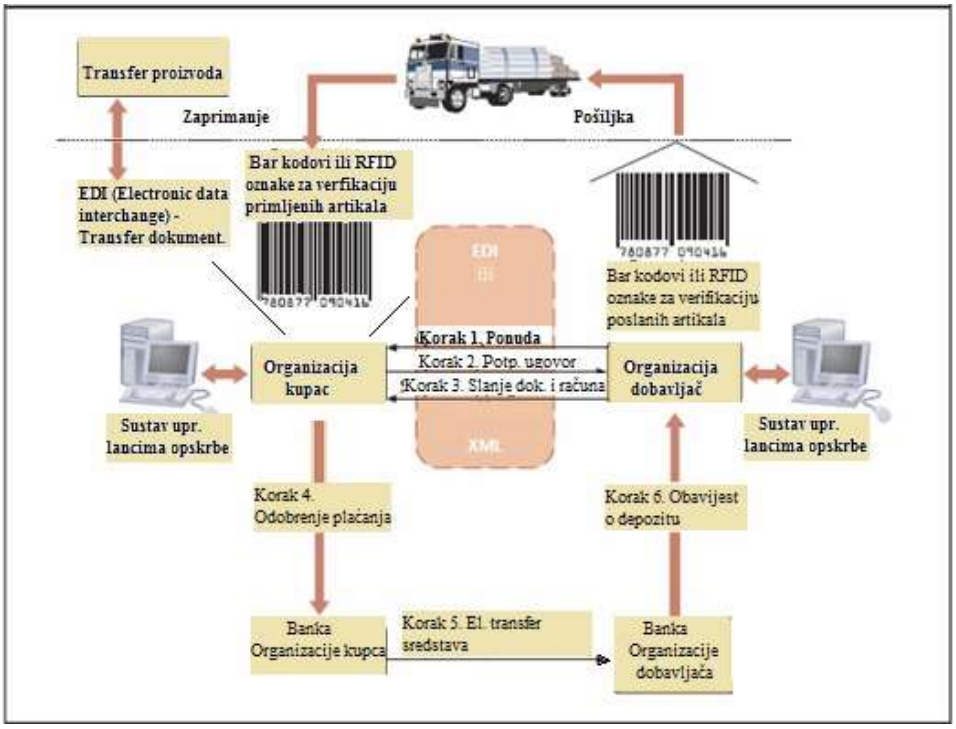
Informacije koje su prije bile fragmentirane na različite sustave, u ERP sustavu se sjedinjuju u jedinstvenoj bazi podataka gdje ih mogu koristiti različiti funkcijski odjeli. (Laudon i Laudon, 2014.)

Iako se raspon ERP sustava može razlikovati od poduzeća do poduzeća, većina pruža integrirani softver kao potporu proizvodnji, prodaji i financijama. Mnogi ERP

sustavi također imaju podsustav za kupovinu koji omogućava naručivanje potrebnih resursa. Uz osnovne poslovne procese, neki ERP sustavi mogu poduprijeti poslovne funkcije poput korisničke usluge, ljudskih resursa i distribucije. Najuspješnije organizacije teže podržati sve svoje funkcije integriranim rješenjem. Primarne koristi implementiranja ERP sustava su olakšavanje usvajanja poboljšanih radnih procesa i povećavanje pristupa pravovremenim podacima koji služe za donošenje odluka. (Stair i Reynolds, 2010.)

3.2. Sustavi za upravljanje lancima opskrbe

Lanac opskrbe je mreža kroz koju se odvija slijed aktivnosti povezanih s proizvodnjom i prodajom proizvoda i/ili usluga. Te aktivnosti uključuju marketing, nabavu sirovina, proizvodnju i dostavu povezane dokumentacije, manufakturu i montažu, pakiranje i dostavu, itd. Informacijski sustavi koji podržavaju ove aktivnosti, a kao podsustavi su povezani u jedinstveni sustav, predstavljaju sustave lanca opskrbe.



Slika 7. Primjer SCM Sustava;
(Izvor: Oz, E., 2008.)

Lanci opskrbe prelaze granice organizacije i obično povezuju organizaciju s dobavljačima i kupcima povezujući tako više organizacija međusobno. Ovi sustavi također imaju koristi od umreženosti internetom (Oz, 2008.) ali od primjene nekih drugih suvremenih tehnologija poput RFID-a. RFID je akronim za Radio-Frequency

Identification a odnosi na tehnologiju malih elektroničkih uređaja koji se sastoje od čipa i antene, a služi istoj svrsi kao bar kod ili magnetna traka na poleđini bankovnih kartica, a to je jedinstvena identifikacija provedena skeniranjem za objekt na kojem se nalazi. (Technovelgy, n.d.)

Raniji SCM sustavi funkcionirali su na "push" principu – vršilo se predviđanje potražnje za proizvodima i na temelju toga generirao raspored dostave. Suvremeni sustavi funkcioniraju na „pull“ principu kojim se dostava vrši točno na vrijeme s obzirom na statistički utvrđenu stvarnu potražnju. (Laudon i laudon, 2014.)

3.3. Sustavi za upravljanje odnosima s klijentima

Upravljanje odnosima s kupcima, klijentima ili strankama je dio strateške politike organizacije kojom se ona usmjerava na kupca s ciljem postizanja konkurentske prednosti. Usmjerenost na kupca u smislu upravljanja odnosima znači uvažavati njegove želje, potrebe, kritike, reklamacije i održavati dvosmjernu komunikaciju različitim kanalima kako bi postojao protok povratnih informacija na obje strane. Organizacije prepoznaju da su kupci temelj uspješnog poslovanja koje, stoga, također ovisi o stvaranju i održavanju učinkovitih odnosa.



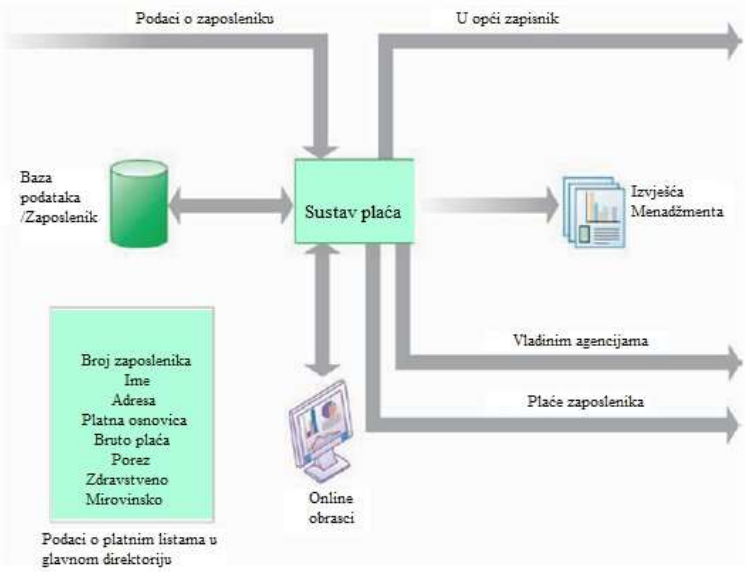
Slika 8. CRM;
(Izvor: Free CRM Applications, 2007.)

Svrha sustava za upravljanje odnosima s kupcima je omogućiti organizacijama ostvarivanje tih učinkovitih odnosa i komunikacije s kupcima kako bi održali postojeći segment tržišta i širili ga dalje. Ovaj sustav također omogućava održavanje i analizu evidencije kupaca s obzirom na trendove, preferencije i obrasce kupovine kao i održavanje dvosmjerne komunikacije u obliku implementiranih kanala. Ovdje su bitni i transakcijski sustavi kojima se prikupljaju podaci za analizu kao i alati poslovne inteligencije koji se koriste u analizi a njih ćemo obraditi kasnije.

3.4. Sustavi za procesiranje transakcija

Mnogi od ranih sustava su dizajnirani kako bi se smanjili troškovi automatiziranjem rutinskih poslova. Procesiranje poslovnih transakcija bilo je predmetom razvoja prvih računalnih programa za većinu organizacija. (Shelly i Vermaat, 2012.) Unutar organizacija, različite funkcije i odjeli mogu se baviti različitim transakcijama pa ih mogu različito i definirati. U računovodstvu bi se, na primjer, transakcija definirala kao „bilo što što vrši promjernu u računskoj bilanci poduzeća“. Definicija koja se koristi za određivanje transakcije na području informacijskih sustava opširnija je i glasi: „Transakcija je bilo što što vrši promjenu u bazi podataka poduzeća.“. (Rainer, Prince i Cegielski, 2014.)

Sustavi za procesiranje transakcija su često među-funkcijski i međuorganizacijski sustavi koji se većinom koriste na operativnoj razini organizacije za provođenje rutinskih i osnovnih ali istovremeno i kritičnih zadataka. Procesiraju podatke nastale kao rezultat odvijanja poslovnih transakcija kao što su to npr. prodaja, kupnja, depozit, povlačenje sredstava i isplate plaća. (Haag i Cummings, 2007.)



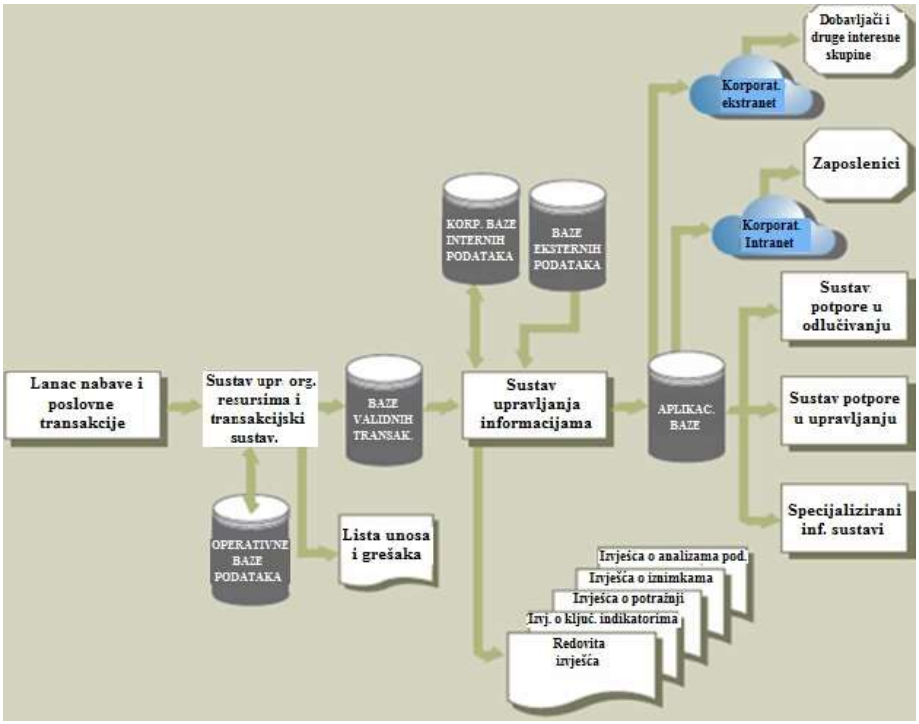
Slika 9. Primjer TPS-a – proces isplate plaća;
(Izvor: Laudon i Laudon, 2014.)

Rani TPS sustavi provodili su transakcije serijski, u smislu da su računala prikupljala unose transakcija kroz određeno vrijeme da bi ih poslije sve skupa provela. Kako su računala bivala sve snažnija, razvili su se online sustavi za procesiranje transakcija - tzv. OLTP sustavi kakvi se većinom danas primjenjuju, a koji procesiraju svaku transakciju u realnom vremenu, to jest čim ona biva unesena. (Shelly i Vermaat, 2012.) Kako bi se zaštitio integritet podataka, TPS sustavi osiguravaju da se, ukoliko

se bilo koji pojedinačni element transakcije ne uspije provesti, ostatak transakcije također neće provesti. (Shelley i Rosenblatt, 2012.) Podaci koje bilježe transakcijski sustavi koriste se za ažuriranje baze podataka i za proizvodnju raznih izvješća za interesne strane unutar i izvan poduzeća. Poslovni podaci prolaze kroz proces obrade transakcija koji podrazumijeva: prikupljanje, uređivanje, ispravak, obradu i pohranu podataka kao i proizvodnju dokumenata koji sadrže te podatke. Neka dostupna TPS rješenja jesu: AccuFund, OprenPro, Quick Books, Timberline i TurningPoint. (Stair i Reynolds, 2010.)

3.5. Sustavi za upravljanje informacijama

MIS su informacijski sustavi čija je svrha da u potpori menadžmentu generiraju točne, pravovremene i primjereno organizirane informacije kako bi uz pomoć njih menadžeri i drugi korisnici mogli informirano donositi odluke, rješavati probleme, nadzirati aktivnosti i pratiti napredak u poslovanju. Proizvodnja, marketing, financije i druge funkcije organizacije podržane su MIS sustavima te dijele zajedničku bazu podataka.



Slika 10. Sustav za upravljanje informacijama;
(Izvor: Stair i Reynolds, 2010.)

MIS sustavi pružaju srednjem menadžmentu izvješća o trenutnoj izvedbi poduzeća, a generiraju tri osnovna tipa izvješća:

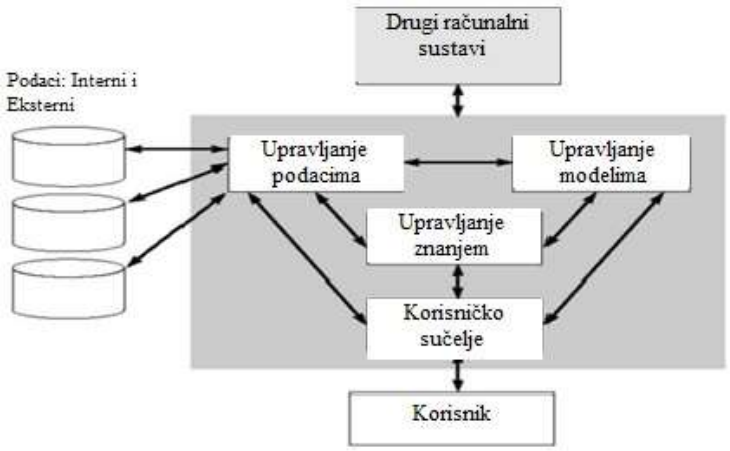
1. **Detaljno izvješće** - obično samo navodi transakcije sa svim povezanim podacima
2. **Sažeto izvješće** - konsolidira podatke iz transakcija i prikazuje ih kao sume, tabele, grafove i sl. kako bi mogli biti lako i brzo pregledani.
3. **Izvješće o iznimkama** - identificira podatke koji se ne podudaraju ili ne podliježu uobičajenim vrijednostima ili stanjima. Iznimke se identificiraju usporedbom s utvrđenim definiranim normalnim rasponima i vrijednostima. (Shelly i Vermaat, 2012.)

Te se informacije koriste pri nadzoru i kontroli poslovanja i procjeni buduće izvedbe i učinkovitosti. Također, ovi sustavi sažimaju i izvještavaju o osnovnim operacijama poduzeća koristeći podatke „povučene“ iz TPS sustava koji se sažimaju te, zatim, prezentiraju u obliku redovitih izvješća. (Laudon i Laudon, 2014) MIS sustavi mogu „vući“ podatke i s vanjskih mreža.

3.6. Sustavi za potporu u odlučivanju

Sustavima za potporu u donošenju odluka se organizacija služi za priskrbljivanje relevantnih informacijskih resursa menadžmentu na temelju kojih donositelji odluka odlučuju o raznim pitanjima vezanim za poslovanje. Ovi sustavi implementiraju složenije procese prikupljanja, obrade i analize podataka od sustava za MIS sustava. Analitički alati i modeli koje primjenjuju potpomažu u donošenju djelomično strukturiranih i nestrukturiranih odluka, za razliku od MIS sustava čiji su analitički procesi jednostavniji. DSS zato vuče podatke i iz TPS i iz MIS sustava te ih procesira dalje zajedno s podacima dobivenih iz vanjskih izvora. (Shelly i Vermaat, 2012.) DSS sustav se sastoji od:

- baze podataka
- sustava za upravljanje podacima i modeliranje podataka
- sustava za upravljanje znanjem
- korisničkog sučelja
- donositelja odluke



Slika 11. Sustav za potporu u odlučivanju;
(Izvor: Turban i Aronson, 2001.)

Vrste odluka koje se donose s obzirom na primjenu objektivnih ili subjektivnih kriterija mogu biti:

- ***Strukturirane odluke*** slijede određena pravila i obrasce i mogu se sagledati i donositi objektivno s obzirom na to da postoje jasno utvrđene metode poput linearnog programiranja i mrežne analize koje se pri odlučivanju primjenjuju kako bi se osiguralo da je donešena odluka ispravna. Ove vrste odluka obično se donose na operativnoj razini organizacije.
- ***Nestrukturirane odluke*** su podložne subjektivnim predodžbama i ne slijede određeni skup
  pravila. Ovo su odluke koje se obično donose na izvršnoj razini menadžmenta.
- ***Djelomično strukturirane*** odluke su one za koje za neke primjenjive kriterije postoje pravila i obrasci, a za neke ne postoje pa su podložni subjektivnoj procjeni. Ove se odluke obično donose na strateškoj razini menadžmenta. (Haag i Cummings, 2007.)

Sustavi za potporu u odlučivanju obično primjenjuju alate poslovne inteligencije i analitike. Poslovna inteligencija predstavlja širok spektar tehnologija i alata koje omogućavaju transformaciju većinom strukturiranih podataka iz baze informacijskog sustava radi analize i ekstrakcije informacija. Alati poslovne inteligencije uključuju npr.:

- **Rudarenje podataka** (orig. data mining) – bottom up analitički proces koji automatski utvrđuje obrasce i poveznice između podataka.
- **OLAP** (online analytical processing) – multidimenzionalna analiza podataka strukturno i kategorički ograniziranih u oblik kocke u kojoj je jedna dimenzija najčešće vremenska, a ostale mogu biti npr. lokacija, proizvod i sl. Podaci se u

kocki analiziraju „slice and dice“ metodom kojom se izdvajaju uzorci u presjecima stupaca i redaka, ali mogu se dobiti i drugačijim perspektivam npr. po pojedinačnim elementima „bušenjem“ ili po hijerarhiji.

- **Razne analize**, npr:

– Analiza osjetljivosti - analizira utjecaj unošenja promjena u dijelovima modela podataka na ostale dijelove.

– Prediktivna "Što ako?" analiza - podrazumijeva primjenu nekih pretpostavki ili predviđanja na temelju kojih se vrše odgovarajuće promjene u modelima, a učinak promjena odgovara na pitanje. Dobiveni ishod ovisi o ispravnosti pretpostavljenih predviđanja.

– Regresivna analiza ciljeva - funkcionira obrnuto od "što ako" analize. Umjesto da se pretpostavlja promjena u modelima i proučava učinak, njome se postavlja željeni učinak i proučava pri kojim će promjenama u modelu on nastati. (Rainer, Prince i Cegielski,

- **Vizualizaciju podataka** – načini predstavljanja podataka koji olakšavaju uviđanje obrazaca, trendova i ovisnosti podataka kao i trenutačno stanje ključnih pokazatelja poslovnih performansi te izvlačenje zaključaka na temelju njih. Radi se o grafičkim prikazima u obliku raznih dijagrama, grafikona i mapa.

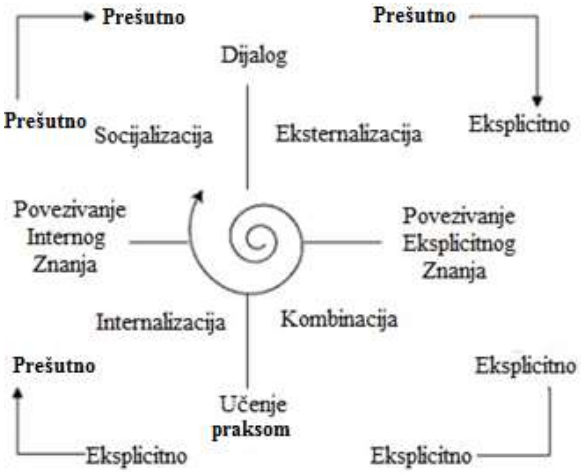
Neki od proizvođača alata poslovne inteligencije su i velike kompanije poput IBM-a, Oracle-a i Microsofta. (Stair i Reynolds, 2010.)

3.7. Sustavi za upravljanje znanjem

Upravljanje znanjem je kombinacija aktivnosti kojima se ostvaruje ciklički proces generiranja znanja koje je sadržano u poduzeću u nestrukturiranom formatu – bilo u obliku podataka ili u umovima zaposlenika. Faze ciklusa jesu: stvaranje, pronalaženje, obrada, pohrana, kontrola i ažuriranje i distribucija znanja. Razlog cikličnosti sustava jest taj što se znanje dinamički poboljšava s vremenom pa je to proces koji se odvija u kontinuitetu. Nikada nije potpun zbog promjenjivosti okoline kojoj se neprestano prilagođava zbog potrebe osuvremenjivanja znanja. (Stair i Reynolds, 2010.)

Znanje, i eksplicitno i prešutno, se kao svojevrstan kapital poduzeća nastoji se pretvoriti u format u kojem se može izmjenjivati između ključnog osoblja i nadograđivati. KMS sustavi pomažu formalizirati prešutno znanje, to jest obuhvatiti stručno, iskustveno znanje ljudskog kapitala, koje je teže formalizirati nego eksplicitno, i načiniti ga dostupnim za korištenje tamo gdje je to potrebno. Model spirale znanja

osmišljen je 1995. godine kao prikaz četiri vrste interakcija prešutnog i eksplicitnog znanja koje isprepletene u spiralu pretvorbom stvaraju znanje.



Slika 12. Model spirale znanja; (Izvor: http://www.tlu.ee/~sirvir/IKM/Theoretical_models_of_Information_and_Knowledge_Management/the_nonaka_and_takeuchi_knowledge_spiral_model_page_3.html (datum pristupa: 16.03.2018.)

1. **socijalizacija** - vrsta interakcije kojom se prešutno znanje pretvara u prešutno znanje. Interakcija kojom se ovo omogućava podrazumijeva prijenos iskustvenog znanja, tehničkih vještina, mentalnih modela itd. a ostvaruje se demonstracijama, promatranjem, oponašanjem, prikazivanjem demonstrativnih videa itd.

2. **eksternalizacija** - vrsta interakcije kojom se prešutno znanje pretvara u eksplicitno a ovo se ostvaruje na različite načine kao npr. objašnjavanjem svrhe nekih odluka, parametara, funkcija, poveznica i odnosa itd.

3. **kombinacija** - vrsta interakcije kojom se eksplicitno znanje pretvara u eksplicitno
znanje. rezultat je utvrđivanje novih obrazaca, odnosa i poveznica.

4. **internalizacija** - vrsta interakcije kojom se eksplicitno znanje pretvara u prešutno, a ostvaruje se korištenjem novih obrazaca i odnosa zajedno s obrazloženjem zašto odgovaraju svrsi odlučivanja, kako bi se pridonijelo prešutnom znanju donositelja odluka. (Nonaka i Takeuchi, 1995.)

Cilj upravljanja znanjem jest pomoći poduzeću da što produktivnije iskoristi akumulirano znanje kreiranjem „najboljih praksi“, to jest utvrđivanjem nejefektivnijih i najefikasnijih načina izvršavanja poslovnih aktivnosti koji vode k općem poboljšanju učinkovitosti i izvedbe poduzeća. Neka od dostupnih KMS rješenja kojima se može

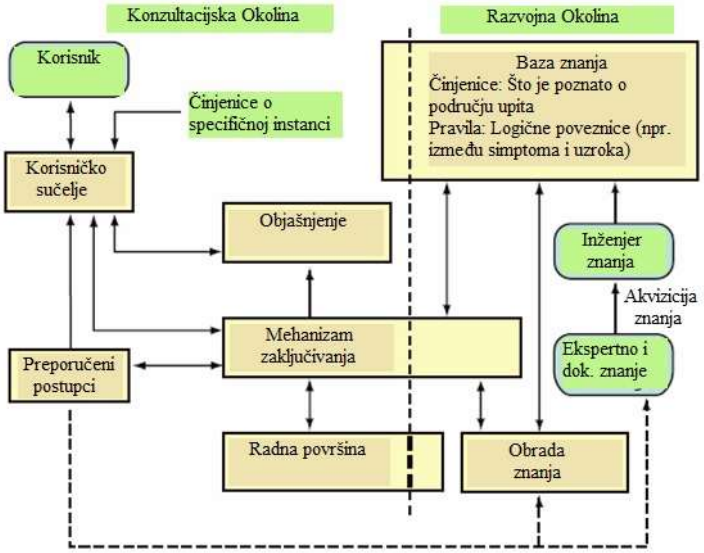
podržati ove aktivnosti jesu: Bloomfire, Communifire, Intelligence Bank, Moxie Knowledgebase, Oxycon i Smart Support.

3.8. Ekspertni sustavi

Ovim sustavima je svrha poduprijeti rad, odlučivanje i zaključivanje na razini znanja stručnjaka na području za koje se primijenjuju, a to omogućava primjena modela zaključivanja i skupine pravila, činjenica i objašnjenja sadržanih u tzv. „bazi znanja“ i kreiranih na temelju formaliziranog ljudskog iskustva na području struke koje je akumulirano minimalno deset godina. Ekspertni sustavi omogućavaju davanje uputstava, predlaganje akcija i donošenje zaključaka relevantnih za izvođenje nekog procesa te su sposobni obrazložiti logiku iza svakog predloška. Također, obično sadrže velike količine podataka koje automatski obrađuju brže i opsežnije nego što je to ljudski moguće. Primjenjuju se na raznim područjima poput medicine, ekonomije, inženjerstva itd.

Znanje je u ekspertnim sustavima predstavljeno u obliku zaključivanja na temelju slučajeva, hijerarhija, okvira, objektnih dijagrama, pravila u obliku premisa i zaključaka formiranih u stilu: „ako a – onda b“. (Haag i Cummings, 2007.) Prijenos stručnog znanja u formalizirani oblik na računalu postiže se u četiri koraka, a to su:

1. Osiguranje dostupnog izvora stručnog znanja
2. Predstavljanje iznošenog znanja u obliku pravila i činjenica koji se pohranjuju u bazu znanja
3. Programiranje metoda zaključivanja koje se primjenjuju na pravilima i činjenicama
4. Prijenos znanja korisniku u obliku predložaka



Slika 13. Ekspertni sustav;
(Izvor: Rainer, Prince i Cegielski, 2014.)

Ekspertni sustav se sastoji od baze znanja, programa za zaključivanje (orig. inference engine), korisničkog sučelja, tzv. „radne ploče“ i podsustava s objašnjenjima. Korisničko sučelje omogućava interakciju korisnika sa sustavom, obično u obliku pitanja i odgovora. „Radna ploča“ (orig. blackboard) predstavlja područje radne memorije odvojene za pohranu podataka o trenutno aktivnom problemu koji sustav treba obraditi. (Rainer, Prince i Cegielski, 2014.)

3.9. Sustavi za potporu izvršnoj razini menadžmenta

ESS sustavi su namijenjeni potpori izvršnih funkcija višeg menadžmenta, konkretno kao potpora u odlučivanju o bitnim strateškim pitanjima i nestrukturiranim vrstama odluka kada ne postoji jasan smjer djelovanja kao što su npr. pitanja spajanja, udruživanja ili bilo kakvih alijansi s drugim poduzećima, proširivanja ili sužavanja tržišnog segmenta i asortimana, djelovanja s obzirom na konkurenciju itd. Ovi sustavi koriste alate za vizualizaciju, filtriranje, sumiranje i praćenje relevantnih internih podataka iz MIS i DSS sustava i vanjskih podataka, pri čemu primjenjuju i alate poslovne inteligencije. Grafički prikazi kojima su podaci prikazani su jednostavni za razumijevanje, a grafičko sučelje sustava je jednostavno za upotrebu.

Osim toga, ESS sustavi također sadrže komunikacijske mogućnosti u vidu slanja poruka i obavljanja običnih i konferencijskih video poziva tijekom kojih pružaju mogućnost prikazivanja vizualiziranih podataka sugovorniku kao praktičan način komuniciranja poslovnih aspekata. (Laudon i Laudon, 2014.)

4. METODOLOŠKI OKVIRI RAZVOJA SUSTAVA

4.1. Metodologija razvoja informacijskih sustava

Razvoj informacijskih sustava podrazumijeva skup aktivnosti koje je potrebno izvesti kako bi se konstruiralo rješenje nekog problema ili ostvarenje neke prilike u obliku informacijskog sustava. (Turban, 2003.) Pritom su se razvile te se primjenjuju i razne metodologije i pristupi koji odgovaraju na pitanje „kako razvijati sustav“, a ti su se načini mijenjali kroz razdoblja unutar zadnjih pedesetak godina. Mogu se okvirno podijeliti na sljedeći način:

- Predmetodološko razdoblje (1960-70) – koncept i primjena formaliziranog načina razvoja sustava je praktički nepostojeća baš kao i projektni menadžment. Prvenstvena briga je u tome da program proradi bez mnogo brige za poslovni kontekst.
- Rano metodološko razdoblje (1970-80) – predstavljanje metodologije razvoja životnog ciklusa sustava u vidu vodopadnog modela kojom se opisuju faze razvoje koje treba izvoditi linearnim slijedom
- Metodološko razdoblje (1980-95) – utvrđivanje pojma i koncepta metodologije kao „preporučenog skupa faza, procedura, pravila, tehnika, alata, dokumentacije, menadžmenta i praksi koje se koriste pri razvoju sustava.“ Fokus je na razvoju boljih sustava i uspostavljanja boljeg razvojnog procesa standardizacijom. U ovom razdoblju se razvijaju strukturni razvoj, modeliranje toka podataka, modeliranje entiteta i njihovih veza, prototipiranje i objektno-orijentirani pristup.
- Postmetodološko razdoblje (počevši od kasnih 90ih) – vrijeme mijenjanja ili odbacivanja dotadašnjih metodologija koje se doimaju ograničavajućima s obzirom na ubrzani tehnološki razvoj i ograničene resurse. Nastupanje agilnog pristupa koji se odlikuje brzim grupnim razvojem, niskom razinom formalizacije, kombiniranja raznih praksi, odbacivanjem mnogih tradicionalnih načela – posebice ekstenzivnog početnog uspostavljanja čvrstih temelja korisničkih zahtjeva. (Avison i Fitzgerald, 2006.)

U raznim literaturama se metodologije razvoja također nazivaju i metodama, pristupima, procesima, okvirima i slično. Uglavnom se konzistentno jasno razlučuju zajednički elementi koji određuju metodologiju razvoja informacijskih sustava, a to su

motivacijski čimbenici, ciklus razvoja, uloge i sudionici, ostvareni rezultati i tehnike i alati.

Pristup razvoju informacijskog sustava ovisi o poslovnoj i projektnoj okolini u kojoj ili za koju se odvija razvoj, a to podrazumijeva uzimanje u obzir mnogih bitnih **motivacijskih čimbenika** koji utječu na izbor. Novi informacijski sustavi su rezultat potrebe rješavanja raznih poslovnih problema, a omogućavaju ih ili su i potaknuti tehnološkim napretcima pa s obzirom na navedeno motivacijski čimbenici mogu biti poslovne ili tehnološke prirode.

- ***Poslovni motivacijski čimbenici*** su globalizacija, razvoj elektroničkog poslovanja, sigurnost i privatnost, suradnja i partnerstva, upravljanje znanjem kao resursom, neprestano poboljšavanje i upravljanje kvalitetom i redizajniranje poslovnih procesa.
- ***Tehnološki motivacijski čimbenici*** su: razvoj mreža i interneta, razvoj mobilnih i bežičnih tehnologija, razvoj objektnih tehnologija uključujući jezike, metode i alate, razvoj tehnologija za omogućavanje i poboljšanje suradnje – telekomunikacija i komunikacijskih tehnologija zasnovanih na web-u i razvoj poslovnih aplikacija. (Whitten i Bentley, 2007.)

Poslovni i tehnološki čimbenici također utječu na preostale čimbenike.

Uloge – uloge koje su dodijeljene sudionicima određuju njihove odgovornosti tijekom razvoja. Različiti sudionici imaju različitu perspektivu sustava koji se razvija s obzirom na uloge, a to su:

- Vlasnici sustava – obično iz redova srednjeg ili izvršnog menadžmenta koji sustav vide u terminima poslovne vrijednosti koju im on treba osigurati svojim funkcionalnostima.
- Korisnici sustava – sustav promatraju iz perspektive funkcionalnosti koje se implementiraju, a bitna svojstva koja utječu na njihov rad su lakoća korištenja, praktičnost, razumljivost, točnost i brzina.
- Analitičari sustava – zaduženi za osmišljavanje konceptualnog rješenja poslovnih problema na temelju kojeg se kasnije sustav dizajnira i programira. Njihova je odgovornost specificirati korisničke zahtjeve i apstraktno i logički prikazati način na koji će se oni ostvarivati sustavom. Analitičari moraju vladati osnovnim znanjima o poslovanju i osnovnim tehničkim znanjima kako bi mogli sagledati širi aspekt problema.

- Dizajneri sustava – prvenstveno se bave tehnologijama kojima se koriste pri dizajniranju informacijskog sustava.
- Graditelji sustava – također tehnološka uloga koja podrazumijeva odgovornost izgradnje sustava po specifikacijama dizajna sustava.
- Vanjski pružatelji usluga - po potrebi može se angažirati osoblje izvan poduzeća i razvojnog tima - bilo koji kadar kojim se dodatno popunjava ili zamjenjuje bilo koja od navedenih uloga sudionika.
- Projektni menadžer – koordinira razvojni tim sastavljen od dosad navedenih sudionika odgovarajućih uloga. Njegova je uloga upravljačka a podrazumijeva primjenu određenih vještina kao što su organiziranje, vođenje, motiviranje itd. (Whitten i Bentley, 2007.)

Ostvareni rezultati – zaključenje ključnih zadataka i aktivnosti koje se izvode tijekom faza razvoja sustava karakterizira ostvarenje rezultata koji su svrha izvođenja tih aktivnosti, a omogućavaju izvođenje daljnjih povezanih aktivnosti. Jedan od ostvarenih rezultata je i sam sustav koji je, u svojem operativnom stanju kao konačni proizvod rezultat ostvarivanja svih ostalih rezultata do tada. Primjeri ostvarenih rezultata od izvođenja aktivnosti su razne vrste dokumentacije, specifikacija, razvijenih modela, planova, rezultata testova, napisanih linija koda, komponente i funkcionalni softver.

Tehnike – načini izvršavanja zadataka uz pomoć raznih alata, znanja i vještina.
(Cadle, 2014.)

Faze razvoja sustava

Ciklus razvoja opisuje faze i njima obuhvaćene aktivnosti koje će se slijediti određenim redoslijedom i rasporedom s obzirom na način dostave konačnog rješenja, a koja može biti linearna, iterativna, inkrementalna ili neka kombinacija navedenog. Formalni opisi uobičajenih faza razvoja se brojem i nazivima razlikuju između raznih literatura. Unatoč tome, razlučive su bitne osnovne skupine aktivnosti, a to su:

1. PLANIRANJE – nakon utvrđivanja problema kojeg je potrebno riješiti, prvo što je potrebno napraviti jest vidjeti o kakvom se sustavu koji predstavlja rješenje za dani problem, s obzirom na okvirni opseg, radi i, na temelju toga, je li moguće razviti taj sustav s obzirom na dostupne resurse uzevši u obzir vrijeme, proračun, znanja i vještine kadrova, trenutnu hardvesku i softversku infrastrukturu itd. Ovakvo istraživanje hipotetskog sustava naziva se studijom izvedivosti. Slijedom toga, ukoliko se donese

odluka o razvoju sustava, planiraju se vrijeme i raspored, proračun, kadriranje, tehnologije itd.

2. ANALIZA – obuhvaća proces utvrđivanja zahtjeva sustava primjenom raznih metoda koje obično uključuju razmatranje i analizu postojećeg sustava u upotrebi (ukoliko on postoji), intervjuiranje naručitelja i korisnika sustava, kreiranje modela sustava, izgradnju prototipa itd. Utvrđeni zahtjevi prolaze i određeni proces validacije kako bi se provjerili na sadržajnu konzistentnost, realističnost i potpunost. Sve navedeno ima svrhu stjecanja razumijevanja sustava koji se razvija, u početku sa korisničkog, a kasnije i sa, odgovarajuće tome, tehničkog gledišta. Stoga se sustav analizira prvo na najvišim razinama apstrakcije na kojima je najlakše razumljiv i korisnicima i razvojnom timu pa se oko njega mogu lako sporazumjeti, a da bi se, zatim, sve više ulazilo u detalje sve do najdetaljnijih analitičkih modela koji opisuju specifikacije koje će se koristiti u fazi dizajna. (Sommerville, 2011.)

3. DIZAJN – obuhvaća logički i fizički dizajn sustava. Logički dizajn sustava određuje ***što*** će sustav raditi na temelju apstraktnih specifikacija, a uključuje dizajn outputa, inputa, procesa, baza podataka, telekomunikacija, kadrova itd. Fizički dizajn sustava određuje **kako** će sustav raditi na temelju tehničkih specifikacija, a uključuje dizajn hardvera, softvera, procedura, telekomunikacija i baza podataka. Zaključno s ovom fazom mora postojati specificirana arhitektura, komponente sustava, sučelja i baze podataka.

4. PROGRAMIRANJE – podrazumijeva primjenu dizajna na kreiranje softvera koji će sadržavati funkcionalnosti potrebne za obavljanje poslovnih zadataka za koje je sustav namijenjen, a to se ostvaruje pretvaranjem specifikacija u računalni kod. Kod strukturnog programiranja pretvorba se izvodi moduliranjem koda, tako da djelovi budu odvojeni u skupove koji predstavljaju zasebne komponente koje se zatim lakše mogu testirati i integrirati. U slučaju kada se želi skratiti proces programiranja može se poslužiti i opcijom korištenja integracije već postojećih modula – svojevrsnih „*template-a*“ koji se zatim mogu po potrebi prilagoditi. Uklapanje postojećih modula može znatno skratiti proces programiranja. (Turban, 2003.)

5. TESTIRANJE – provjera softvera na udovoljavanje specifikacijama i korisničkim zahtjevima podrazumijeva testiranje u svrhu verifikacije i validacije. Navedeno se postiže provođenje testova, uglavnom se oni mogu svesti na testiranje komponenti, integracijsko testiranje i testiranje prihvatljivosti sustava slučajem korisštenja u primjeni – koristeći stvarne inpute sustava i razmatrajući outpute dobivene obradom. Testiranje

umanjuje rizike sustava i čini jedan od alata osiguranja kvalitete. (Sommerville, I., 2011.)

6. IMPLEMENTACIJA – osposobljavanje sustava instalacijom i konfiguriranjem potrebnog softvera i hardvera uz uspostavljanje baza podataka i, ukoliko postoji stari sustav, pretvorba podataka iz baza starog sustava u baze novog.

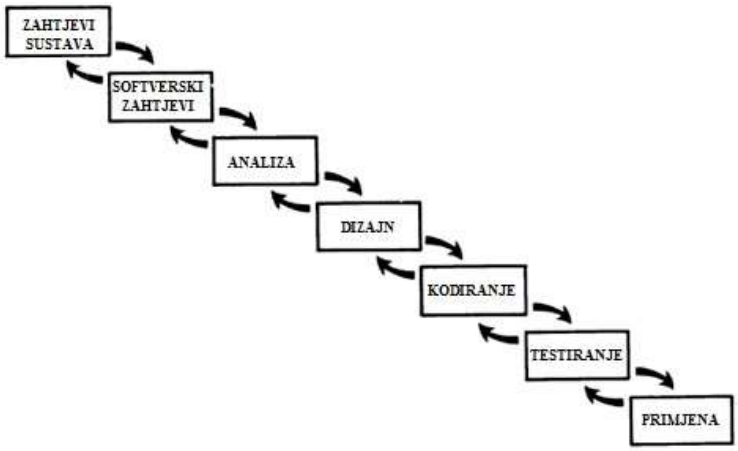
7. KORIŠTENJE I ODRŽAVANJE – sustavi se najčešće nastavljaju razvijati nakon stavljanja u upotrebu pogotovo s otkrićem nekih grešaka ili pojavom potreba za novim funkcionalnostima i svojstvima. Sustav se razvija u upotrebi kroz razne nadogradnje i zakrpe. (Whitten i Bentley, 2007.)

4.2. Tradicionalni razvoj sustava

Tradicionalni ili vodopadni pristup predstavlja najraniji predložak formalizacije procesa razvoja informacijskih sustava. Prijedlog je predstavljen 1970. godine u članku Winstona Roycea pod nazivom „Managing the Development of Large Software Systems“. Brzo je stekao potporu menadžera jer je jednostavno razložen i teče logički od početka do kraja razvojnog procesa s tim da različite literature navode različiti broj faza različitih naziva ali u ovom radu ćemo prikazati originalni predložak.

Vodopadni model

Modelom vodopada se pretpostavlja da se svi zahtjevi mogu korektno, jasno i detaljno utvrditi odmah na početku razvojnog procesa. Prijedlog se našao na udaru kritika u kojima ga se naziva krutim i rigoroznim, pritom uzimajući u obzir prvu prikazanu inačicu modela kojom je razvoj prikazan kao jednostavni top-down linearni proces no prvotno popularizirana verzija modela, poznata i pod nazivom „klasični“ ili „čisti“ model vodopada, zapravo je nedorečeni prikaz s obzirom na to da je konačni prijedlog rada kasnije u članku nadograđen povratnim revizijama koje bi trebale biti ograničene na susjednu fazu ali u praksi to obično ne funkcionira. (Royce, 1970.)



Slika 14. Waterfall model s povratnim vezama;
(Izvor: Royce, 1970.)

U mnogim je opisima modela ovaj zaključni prijedlog zametnut. Princip na kojem funkcionira predložena metodologija jest sekvencionalno napredovanje kroz faze razvoja izvođene redoslijedom kojim su prikazane na slici 14. u kojem svaka sljedeća faza u razvoju može otpočeti tek kada je prethodna zaključena te njena potrebna dokumentacija ispostavljena. Povratne veze predstavljaju feedback koji omogućava da interakcije između faza također teku unatrag ako u svom napredovanju kroz ciklus u određenoj fazi razvoj postane problematičan pa je potreban povratak na neku ili više prethodnih faza kako bi se izvršile promjene koje će omogućiti nastavak razvoja kada se ponovno vrati u fazu koja je bila problematična.

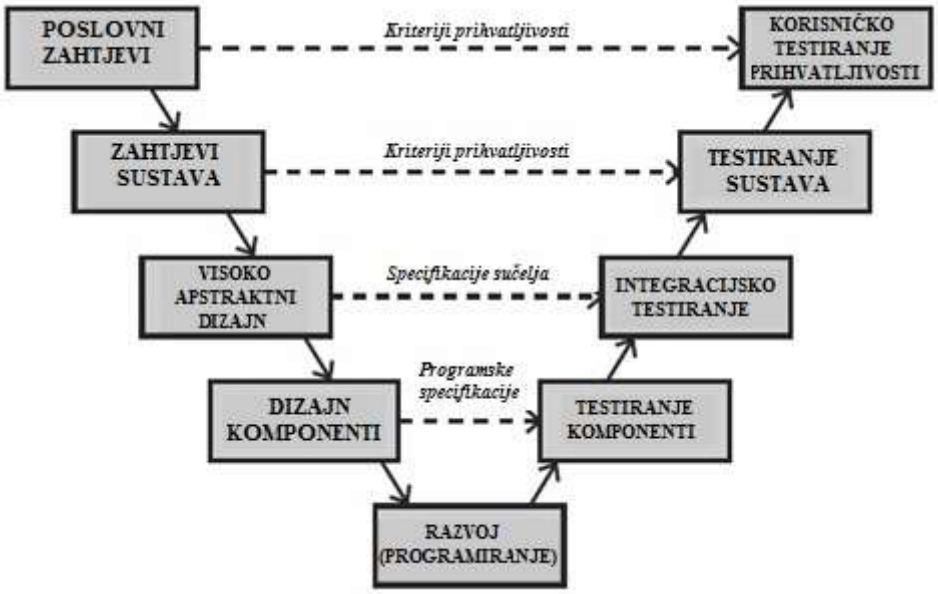
Sustav treba dobro razumjeti prije nego što se otpočne sa razvojem. Ovaj pristup iziskuje mnogo dokumentacije i vremena i rane faze utvrđivanja zahtjeva i analize su kritične za uspjeh i učinkovitost izvedbe. Što dalje uznapreduje razvoj tim postaje sve skuplje i vremenski zahtjevnije raditi promjene zbog načina na koji je izvođenje strukturirano. Što je više faza zaključeno prethodno problematičnoj fazi, tim je veći obujam promjena koji se mora izvršiti u retrospektivi. Što se više vremena provede na ranijim fazama razvoja tim se više vremena uskraćuje kasnijim fazama kako bi razvoj bio završen u što skorijem roku. Najčešće najviše vremena biva oduzeto fazi testiranja što kasnije kao posljedicu može imati implementiranje lošeg proizvoda. (Dennis, Wixom i Roth, 2011.)

Najznačajnije ograničenje ovog pristupa jest očekivanje obuhvaćanja svih zahtjeva na ispravan način i u ukupnim razmjerima prije nego što otpočne dizajn. Korisnici često ne znaju ne mogu eksplicitno izraziti zahtjeve dok ne dobiju priliku iskušati funkcionalnu verziju sustava. Navedeno čini očekivanje inicijalno korektno izvedenog utvrđivanja zahtjeva nerealističnim, a procjenjuje se da je greške

napravljene u fazi utvrđivanja zahtjeva kasnije 200 puta skuplje ispravljati ukoliko se ne otkriju prije faze implementacije. (Avison i Fitzgerald, 2006.)

V model

V model je osmišljen kao rješenje problema prisutnog kod modela vodopada koji se tiče nedostatne alokacije vremena na fazu testiranja u životnom ciklusu u slučaju kada se zbog potrebnih promjena u ranijim fazama to vrijeme potroši na njih, a kasnije prikrati na testiranju kako bi se stiglo na vrijeme implementirati gotovo riješenje sustava i ostati unutar zadanog proračuna. Oblik modela vodopada je modificiran u oblik slova V, kao što je prikazano na slici br. 15.



Slika 15. V model;
(Izvor: Cadle, 2014.)

Ovim modelom predlaže se fokus na testiranje, to jest na validaciju i verifikaciju. Ova dva pojma razlikuju se s obzirom na perspektivu iz koje se vrši testiranje pa tako verifikacija predstavlja testiranje kojim se utvrđuje da li se rješenje sustava razvija na pravi način dok validacija predstavlja testiranje kojim se utvrđuje da li je rješenje koje se razvija pravo rješenje. (Boehm, 1989.) U kontekstu V modela validacija se provodi s obzirom na analizu kojom se utvrđuje „pravo rješenje“, a verifikacija s obzirom na dizajn koji se razvija na „pravi način“.

U V modelu, lijevom stranom oblika slova V teče izvođenje faza razvoja točno utvrđenim redoslijedom kao i u modelu vodopada sve do zadnje faze prije nego što se razvoj nastavi na desnoj strani - izrade programskog rješenja. Paralelno sa izvedbom svake faze do tada planira se testiranje specifično za tu fazu kao što je vidljivo

prikazom isprekidanih strelica koje svaku fazu povezuju s odgovarajućim testiranjem na desnoj strani V modela. Na taj način, u trenutku kada razvoj zaključuje programsko rješenje nastavlja sa unaprijed utvrđenom i isplaniranom verifikacijom i validacijom rješenja.

Nadograđeni V model uključuje studiju izvedivosti kao prvu fazu razvoja i paralelno s njezinom izvedbom planiranje provedbe ispitivanja ostvarivih koristi u poslovanju s obzirom na očekivanja. (Cadle, 2014.)

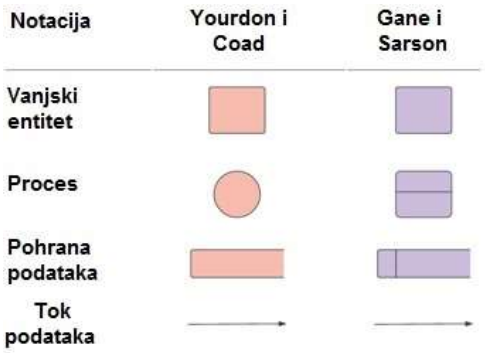
4.3. Modeliranje procesa

Model procesa je grafički način predstavljanja načina na koji bi sustav trebao funkcionirati. Njime se ilustriraju procesi ili aktivnosti koji se izvode i način na koji se podaci kreću između njih. Za potrebe ovog rada, pojasniti ćemo jednu od najčešće korištenih načina modeliranja procesa – dijagram toka podataka.

Dijagram toka podataka pojavljuje se 1975. godine u knjizi „Structured Design“ autora Yourdona i Constantinea u vrijeme kada strukturni razvoj dobiva na popularnosti. (Lucidchart 1, n.d.) Svrha dijagrama toka podataka je modeliranje logike kretanja podataka kroz sustav, prikazujući pritom granice sustava, procese koji se u njemu odvijaju i one procese kroz koje se podaci transformiraju. Iako svojim nazivom ovaj dijagram može zvučati kao da se fokusira na podatke, zapravo je usmjeren na procese.

Postoje logički i fizički dijagram toka podataka, a razlika je u tome što se logički model usredotočuje na poslovne aktivnosti i odgovara na pitanje „što?“, dok fizički model razmatra kako se sustav implementira te odgovara na pitanje „kako?.

Kada se kreira dijagram, može se koristiti jedna od dvije različite skupine notacija koje nose naziv po svojim autorima, a to su Yourdon/Coad i Gene/Sarson notacije. Postoje i druge varijacije ali ovo su najčešće korištene. Njihovu razliku možemo vidjeti na slici 16.



Slika 16. Notacije dijagrama toka podataka;

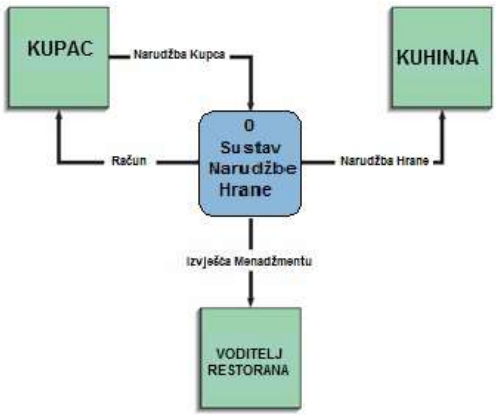
(Izvor: https://www.lucidchart.com/pages/data-flow-diagram, datum pristupa: 09. 07. 2018.)

Iz gornje slike su također vidljiva četiri elementa od kojih se dijagram sastoji, a to su:

1. **Vanjski entitet** - predstavlja osobu, organizaciju, odjel ili sustav koji su izvan granica modeliranog sustava, tj. čine njegovu okolinu ali sustav vrši interakciju sa njima. Vanjski entiteti unose podatke u sustav ili vrše dohvat podataka iz sustava.
2. **Process** – aktivnost ili funkcija koju sustav izvodi. Svaki proces ima jedinstveni identifikacijski broj i naziv. Naziv procesa se, u principu, formira tako da je prva riječ glagol, a ako ima više riječi, a obično ima, tada je zadnja riječ imenica. Svaki proces izvodi samo jednu aktivnost tako da se u nazivu izbjegava koristiti sastavni veznik „i“ jer se njime podrazumijeva da se izvodi više od jedne aktivnosti. Također, svaki proces mora imati bar jedan ulazni tok podataka i bar jedan izlazni.
3. **Tok podataka -** put jednog podatka ili logičkog skupa podataka koji procesima prolaze kroz sustav. Za naziv svakog toka podataka se koristi imenica. Opis toka podataka mora opisivati točne elemente podataka koje tok sadržava. Svaki tok podataka ili ulazi u neki proces i/ili izlazi iz nekog procesa sa strelicama koje pokazuju odgovarajući smjer toka.
4. **Pohrana podataka** - predstavlja spremište podataka u koje se pohranjuju podaci. Svako spremište podataka ima svoj naziv (obično imenica), identifikacijski broj i opis. Sva spremišta podataka trebaju imati bar jedan ulazan tok podataka i jedan izlazni tok podataka.

Tok podataka koji je namijenjen za pohranu i tok namijenjen za dohvat moraju biti prikazani kao dva zasebna toka. (Dennis, Wixom i Roth, 2011.)

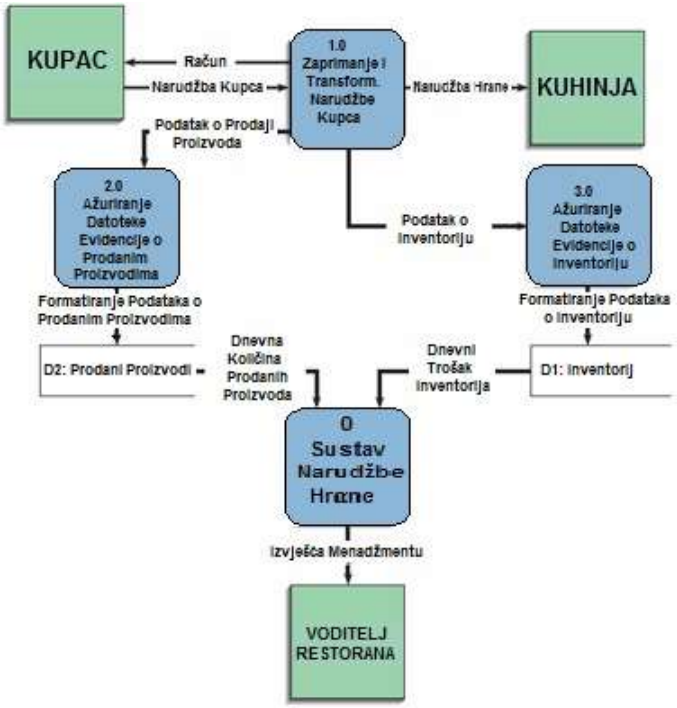
Granice i opseg sustava kao i odnosi sustava sa okolinom prikazuju se nultom razinom dijagrama koja sadrži samo jedan „nulti“ proces koji predstavlja cijeli sustav, taj dijagram se naziva ***kontekstualni dijagram***.



Slika 17. Dijagram toka podataka nulte razine – Kontekstualni dijagram;
(Izvor: Valacich, George i Hoffer, 2011.)

Dijagram toka podataka nulte razine također ne sadrži spremišta podataka, ima vrlo malo tokova i definirane vanjske aktere.

Dijagram prve razine detaljnije prikazuje glavne funkcije sustava razbijanjem nultog procesa na više specifičnih podprocesa koji omogućavaju ostvarenje nultog. Tako dobivamo i manje apstraktan dijagram u odnosu na kontekstualni i, u slučaju na slici br. 18, četiri procesa, dodatne tokove i spremišta podataka.



Slika 18. Dijagram toka podataka prve razine;
(Izvor: Valacich, George i Hoffer, 2011.)

Ovaj dijagram je moguće još razbiti na detalje i iako ovdje nećemo dalje razlagati cijeli sustav, moguće je poslužiti se daljnjom ***dekompozicijom*** kako bi se razložio jedan proces a to je način na koji se mogu razložiti svi procesi i dobiti još složeniji dijagram sustava više razine. Dekompozicija se može vršiti nad procesom sve dok on više ne može smisleno biti dalje razložen.

Pri dekompozijciji je bitno pridržavati se principa očuvanja ulaza i izlaza kako bi se zadržalo logički smisao modela. Dakle, procesi moraju zadržavati identične ulazne i izlazne tokove, a taj se princip naziva održavanjem ravnoteže ili, izvorno, „balancing“. (Valacich, George i Hoffer, 2011.)

4.4. Modeliranje podataka

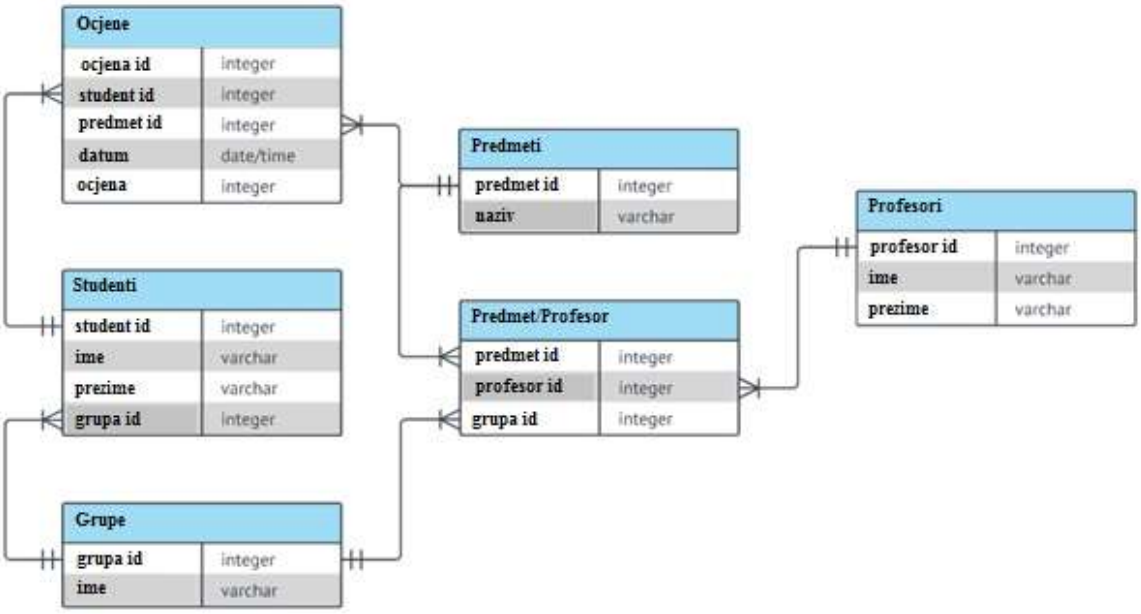
Model podataka je formalni način prikazivanja podataka koji se stvaraju i koriste u poslovnom informacijskom sustavu i načina na koji su organizirani. Njime se prikazuje i sve ono što čini predmet stvaranja podataka - entitete, npr. stvari, ljudi, mjesta itd. i način na koji su oni međusobno povezani. ER modeli i modeli podataka se, s obzirom na razinu detalja, prikazuju u tri razine:

- Konceptualna razina: razina detalja najvišeg nivoa apstrakcije – najmanje detalja. Njome se prikazuje opseg sustava i njegova arhitektura. Za manje sustave se ova razina obično preskače.
- Logička razina: sadrži više detalja od konceptualne u vidu definiranja operativnih i transakcijskih entiteta. Model je, na ovoj razini, nezavisan od tehničkih detalja potrebnih kod implementacije. Ne opisuje način na koji se podaci stvaraju, pohranjuju i obrađuju.
- Fizička razina: prikazuje dovoljno detalja koji omogućuju razvoj i implementaciju baze podataka. (Lucidchart 2, n.d.) Za kreiranje modela podataka obično se koriste CASE alati.

Modeliranje veza između entiteta je način modeliranja podataka pri kojem se koristimo Entity Relationship dijagramom. ER dijagram je originalno razvijen za primjenu kod dizajna baza podataka, a osmišljen je od strane Petera Chena te se pojavljuje u njegovom članku pod nazivom „The Entity-Relationship Model - Toward a Unified View of Data“, 1976. godine. Chen je prepoznao probleme u prenošenju zahtjeva u komunikaciji između analitičara i korisnika što je često rezultiralo nesporazumima i uzrokovalo nezadovoljavajuća rješenja od kojih se odustajalo ili mijenjalo po visokom trošku. ER dijagrami su grafički prikazi podatkovnog sadržaja sustava, a opisuju podatke kao entitete, atribute i veze. Ljudi koji razvijaju sustav koriste se ER dijagramima kako bi njima prikazali svoje shvaćanje korisničkih zahtjeva koje zatim predstavljaju korisnicima. (Riccardi, 2003.) ER dijagramom također mogu biti prikazana neka pravila poslovanja u vidu ograničenja ili smjernica koje se slijede tijekom izvršavanja operacija sustava, kao na primjer ograničenje sredstva plaćanja na kartice, gotovinu, čekove i kupone. (Dennis, Wixom i Roth, 2011.)

Za kreiranje ER dijagrama postoje različite notacije od različitih autora pa se tako razlikuju Chenova, Bachmanova, Barkerova i IDEF1X notacija, a za potrebe ilustriranja primjera ER dijagrama na slici br. 19., u ovom radu će se koristiti Martinova notacija - tzv. „Vranino stopalo“ (orig. engl. „Crow's foot“) koje je dobilo svoj naziv po

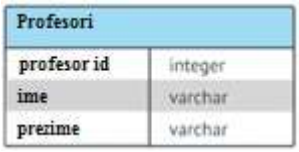
asocijaciji s načinom na koji je osmišljeno prikazivanje **kardinalnosti** veza gdje notacija za obilježavanje mnogo instanci nekog entiteta u relacijama igleda ovako: <



Slika 19. ER dijagram;
(Izvor: https://www.lucidchart.com/pages/er-diagrams?a=0, datum pristupa: 09.07.2018.)

Elementi koje sadrži ER dijagram su:

1. **Entiteti -** entiteti u dijagramu predstavljaju apstrakcije ljudi, događaja, mjesta i ostalih stvari u stvarnom svijetu koje su relevantne za modeliranje podataka sustava, a prikazani su pravokutnicima na sljedeći način:



Slika 20. Entitet;
(Izvor: https://www.lucidchart.com/pages/er-diagrams?a=0, datum pristupa: 09.07.2018.)

Pravokutnik može biti podjeljen u više djelova. Na prvom djelu pri vrhu se nalazi naziv entiteta koji se obično piše kao jedna imenica. U ovom primjeru entitet predstavlja ***učitelje*** o kojima baza podataka sadrži podatke.

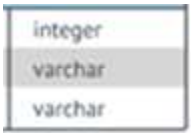
2. **Atributi** - u pravokutniku ispod naziva entiteta nalaze se atributi. Atributi su vrste podataka o entitetima, pa tako se za entitet učitelji mogu primijeniti sljedeći atributi:



Slika 21. Atributi;
(Izvor: https://www.lucidchart.com/pages/er-diagrams?a=0, datum pristupa: 09.07.2018.)

Entitet realno može imati mnogo atributa no samo su neki relevantni za bazu podataka sustava, a to su oni koji su važni i koriste se u poslovnim procesima. Posebni atributi služe kao identifikatori, to jest ključevi, kao npr. u ovom slučaju „teacher id“. Ključevi služe identificiranju instanci entiteta. ***Primarni ključ*** jedinstveno identificira instancu entiteta u promatranom skupu, dakle ne postoje dva entiteta s posve istim vrijednostima tog atributa. Ključ je ***jednostavan*** ako se sastoji od samo jednog atributa. ***Složeni ključ*** sadrži dva ili više atributa, npr. prikazani ER dijagram sadrži entitet „Ocjene“ koji sadrži više ključeva (ocjena id, student id i kolegij id). ***Strani ključ*** se u izradi baze podataka primjenjuje kao atribut nekog entiteta koji već služi kao primarni ključ drugog entiteta kako bi se uspostavila relacija između
izvornog entiteta (tzv. roditelja) i entiteta sa stranim ključem (tzv. djeteta).

Atributi prisutni u ovom ER dijagramu također imaju naznačene tipove podataka koji predstavljaju atribute:



Slika 22. Tipovi podataka;
(Izvor: https://www.lucidchart.com/pages/er-diagrams?a=0, datum pristupa: 09.07.2018.)

Na slici 22. je, npr., u prvoj stavci koja se odnosi na atribut „učitelj id“ navedeno da je podatak za ovaj atribut integer, to jest da je u obliku broja pa se može reći da je identifikacijska oznaka učitelja npr. „6“.

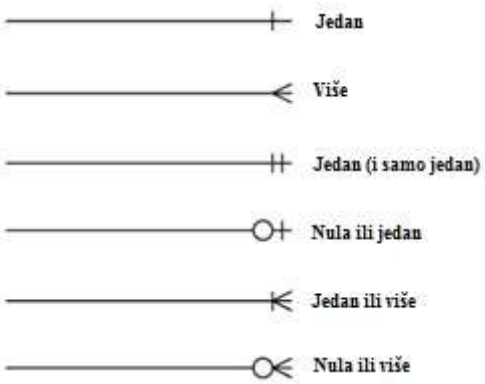
3. **Veze/relacije -** asocijacije između entiteta prikazane linijama kojima su entiteti povezani. Te linije se razlikuju s obzirom na ***kardinalnost***. Kardinalnost prikazuje omjer „roditeljskih“ instanci entiteta prema „dječjim“ instancama i obrnuto pa s obzirom na to postoje odnosi:

1:1 - jedan prema jedan: jedna instanca entiteta „roditelja“ je povezana s jednom instancom entiteta „djeteta“.

1:N - jedan prema više: jedna instanca „roditeljskog“ entiteta je povezana s više instanci „dječjeg“ entiteta no instanca „dječjeg“ entiteta je povezana sa samo jednom instancom „roditeljskog“ entiteta.

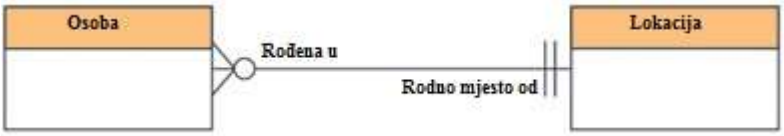
M:N - više prema više: više instanci „roditeljskog“ entiteta je povezano s više instanci „dječjeg“ entiteta. (Lucidchart 2)

Osim toga, ***modalitet*** nalaže i moguću vrijednost od nula instanci.



Slika 23. Notacije odnosa
(Izvor: https://www.lucidchart.com/pages/er-diagrams?a=0 , datum pristupa: 09.07.2018.)

Veze bi trebale biti označene aktivnim glagolima kao bi bile razumljivije. Ukoliko je glagol naznačen u vezi na oba kraja asocijacije tada se i odnos tumači u oba smjera, npr. na sljedećoj slici se može vidjeti da instanca „Osobe“ može biti rođena na jednoj i samo jednoj instanci „Lokacije“, ali instanca „Lokacije“ može biti rodno mjesto 0 ili više instanci „Osoba“:



Slika 24. Primjer konteksta korištenja notacije odnosa;
(Izvor: https://www.lucidchart.com/pages/er-diagrams?a=0 , datum pristupa: 09.07.2018.)

Kada je dijagram kreiran, na njega se može primjeniti ***normalizacija.*** Normalizacija je proces u kojem se na logički model podataka primjenjuje skup pravila kao bi se utvrdilo koliko je dobro kreiran. Rezultat procesa normalizacije trebaju biti stabilne ali fleksibilne veze između entiteta i ispravno utvrđeni entiteti. Normalizacija se sastoji od tri tzv. „normalne forme“, a to su:

Prva Normalna Forma (1NF) – logički model podataka je 1NF ako ne sadrži atribute koji imaju vrijednosti koje se ponavljaju za jedinstvenu instancu entiteta. Ovo obično znači potrebu kreiranja novih entiteta i novih veza kojima će se spojiti stari i novi.

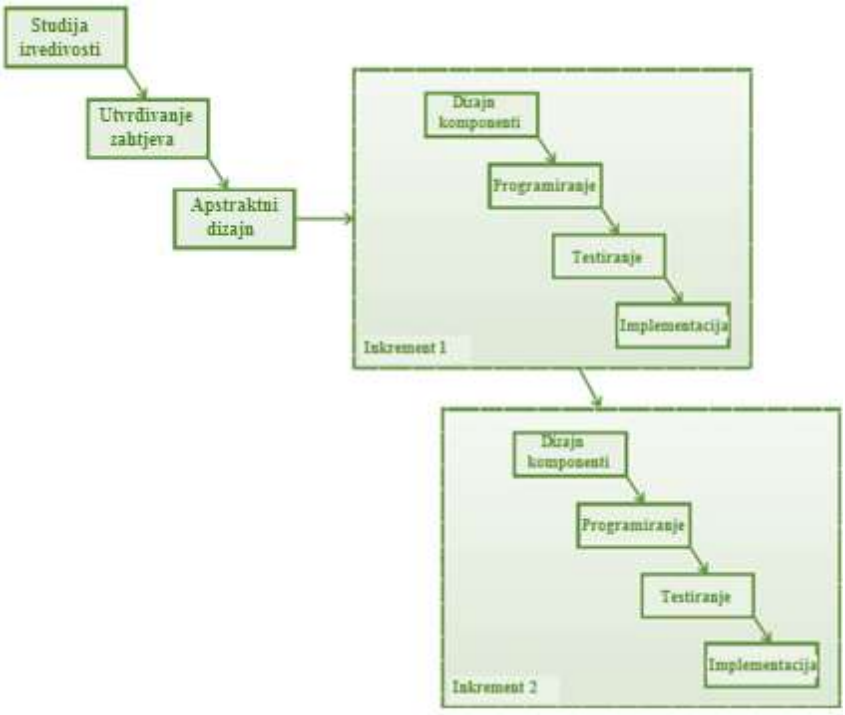
Druga Normalna forma (2NF) – logički model podataka je 2NF ako je i u 1NF i ako sadrži entitete čiji su svi atributi ovisni o identifikatoru. Ovo znači da vrijednost svih atributa koji služe kao identifikatori može odrediti vrijednost svih drugih atributa instance entiteta. Ovo obično znači potrebu kreiranja novih entiteta i novih veza kojima će se spojiti stari i novi.

Treća normalna forma (3NF) - logički model podataka je 3NF ako je i u 1NF i u 2NF i ako kod entiteta ni jedan atribut nije ovisan o atributu koji nije identifikator. Ovo obično

znači potrebu kreiranja novih entiteta i novih veza kojima će se spojiti stari i novi. (Dennis, Wixom i Roth, 2011.)

4.5. Inkrementalni razvoj

Inkrementalni razvoj je djelomično linearan, u smislu da se prvo faze planiranja i analize cjelovitog sustava izvode sekvencionalno, jedna za drugom. Razlika između inkrementalnog razvoja i životnog ciklusa modela vodopada ili V modela je u tome što se, nakon planiranja, analize i dizajna, pojedinačni djelovi sustava koji čine funkcionalne komponente mogu zasebno i/ili paralelno dalje razvijati na temelju početnih faza i implementirati po unaprijed planiranom redoslijedu, pogotovo npr. oni djelovi koji predstavljaju kritične komponente čije je stavljanje u funkciju hitnije od nekih drugih. Obično se radi o onim komponentama čije funkcije čine osnove poslovanja. Time se umanjuje problem dugotrajnog razvoja. Naručitelj rješenja ne mora čekati na korištenje sustava onoliko dugo koliko traje razvoj čitavog sustava već može koristiti funkcionalne komponente koje se razvijaju i implementiraju prioretiziranim redoslijedom. Bitno je da za svaki implementirani inkrement postoji korisnički feedback. Naknadne promjene na prethodnim inkrementima povećavaju troškove projekta pogotovo ako se te promjene odražavaju na ostale inkremente.

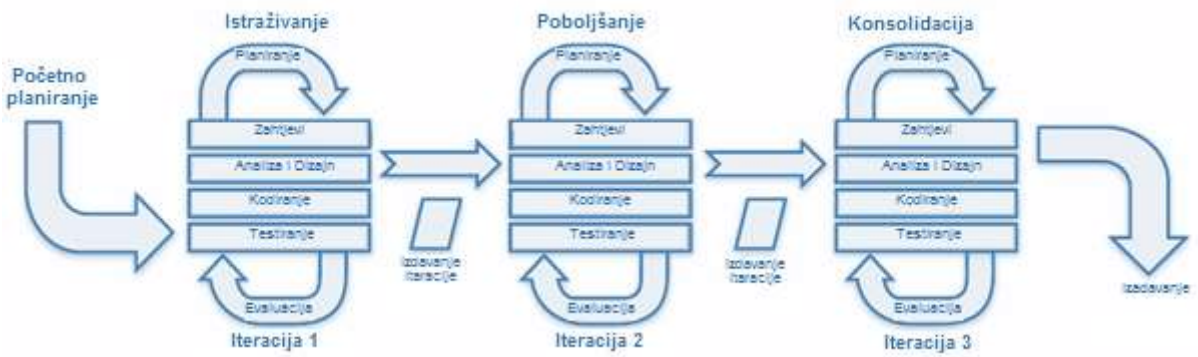


Slika 25. Inkrementalni razvoj;
(Izvor: Cadle, 2014.)

Inkrementalni razvoj ahtijeva strogu i pažnjivu analizu u ranim fazama, jer to je temelj na kojem će se razvijati svi inkrementi koji će uslijediti, dakle ne predviđaju se naknadne izmjene u početnim fazama. Svaki inkrement mora proći regresijsko testiranje sa cjelokupnim dotad implementiranim inkrementima kako bi se osiguralo da se funkcionalno može integrirati s njima što znači da i svi dotadašnji inkrementi sa svakim novim inkrementom prolaze testiranje. Inkrementalni pristup može smanjiti troškove. Ako organizacija odluči da je isporučena funkcionalnost do tada implementiranim inkrementima dovoljna, onda se projekt može zaustaviti i daljnji troškovi izbjeći. (Cadle, 2014.)

4.6. Iterativni razvoj

Iterativni razvoj funkcionira tako da se proces razvoja sa svim njegovim fazama, to jest iteracija, izvodi više puta sve dok ne dobijemo konačno zadovoljavajuće rješenje s kojim će korisnik biti zadovoljan. Pritom se tome ne pristupa na način da svaka iteracija predstavlja pokušaj razvoja konačnog rješenja jer bi u tom slučaju troškovi i vrijeme predstavljali velik problem, već svaka iteracija ima svoju svrhu, rok i razmjere.



Slika 26. Iterativni razvoj;
(Izvor: Cadle, 2014.)

Početnom iteracijom se mogu obuhvatiti osnovne potrebne funkcije i neki korisnički zahtjevi pa dizajnirati i implementirati na temelju toga. Takva iteracija služi kao istraživačka i daje se na uvid korisniku kako bi se utvrdilo jesu li dobro utvrđeni najbitniji zahtjevi. Početnom iteracijom se također utvrđuje primjerenost korištene tehnologije i rizici i problemi u dizajnu. Slijedećom iteracijom se unose promjene s obzirom na feedback na prethodnu i dodaju funkcije koje odgovaraju ostalim zahtjevima ali su manje bitni. Zatim se iteracija opet predstavlja kao predložak korisniku na temelju čijeg feedbacka se ide u daljni razvoj gdje se, zaključenjem funkcionalnih zahtjeva veća pozornost usmjerava na tehnički aspekt dizajna i

stabilizaciju u smislu usavršavanja koda i integracije. Iteracije se izvode sve dok projekt nije gotov ili odbačen. Optimalno bi bilo kada bi se projekt mogao izvršiti u dvije, tri iteracije s rezultirajućim zadovoljavajućim rješenjem. Ponekad se iteriranjem pokušava razviti sustav koji izvršava neke kritične funkcije zbog kojih se mora posebno paziti na neka svojstva kao npr. na rizik od "rušenja" kod sustava za održavanje života. Tada iteracija može imati svrhu koja odgovara udovoljavanju nekog svojstva. (Satzinger, Jackson i Burd, 2010.; Cadle, 2014.)

4.7. Prototipiranje

Prototipiranje je iterativna i inkrementalna metodologija razvoja kojom se često faze analize i dizajna, a ponekad i dijela planiranja stapaju u jednu fazu - prototipiranje. Prototipiranje predstavlja brzi razvoj eksperimentalnog sustava koji se onda daje na uvid i evaluaciju korisniku. Prototip je funkcionalni sustav kojeg korisnik može koristiti i na temelju kojeg onda može definirati i jasnije razlučiti svoje zahtjeve na temelju kojih se onda taj prototip poboljšava i nadograđuje dok ne dosegne zadovoljavajuću razinu funkcionalnosti i upotrebljivosti (ovdje se pri upotrebljivosti misli na značenje iza pojmova koji se u literaturi nazivaju "usability" ili "user friendly" a koji se ne odnose na funkcionalnost, to jest, izvođenje operacije sustava već na praktičnost kojom je korisnik u mogućnosti upravljati sustavom kako bi sustav izvršio operacije).

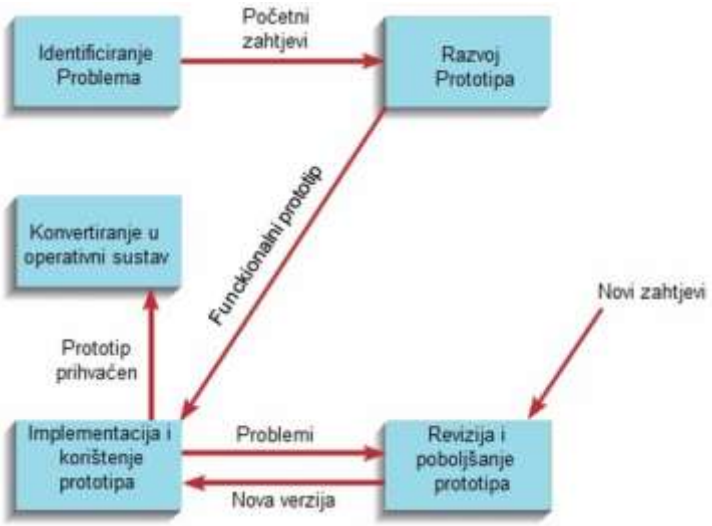
Prototipiranje se odvija sljedećim koracima:

Korak 1: Utvrđivanje osnovnih korisničkih zahtjeva.

Korak 2: Razvoj preliminarnog prototipa (ovdje su korisni CASE alati)

Korak 3: Korištenje prototipa od strane korisnika kako bi dobili feedback na temelju kojeg će se prototip dalje mijenjati i nadograđivati.

Korak 4: Revizija i poboljšavanje prototipa - u skladu s feedbackom dobivenim u trećem koraku.



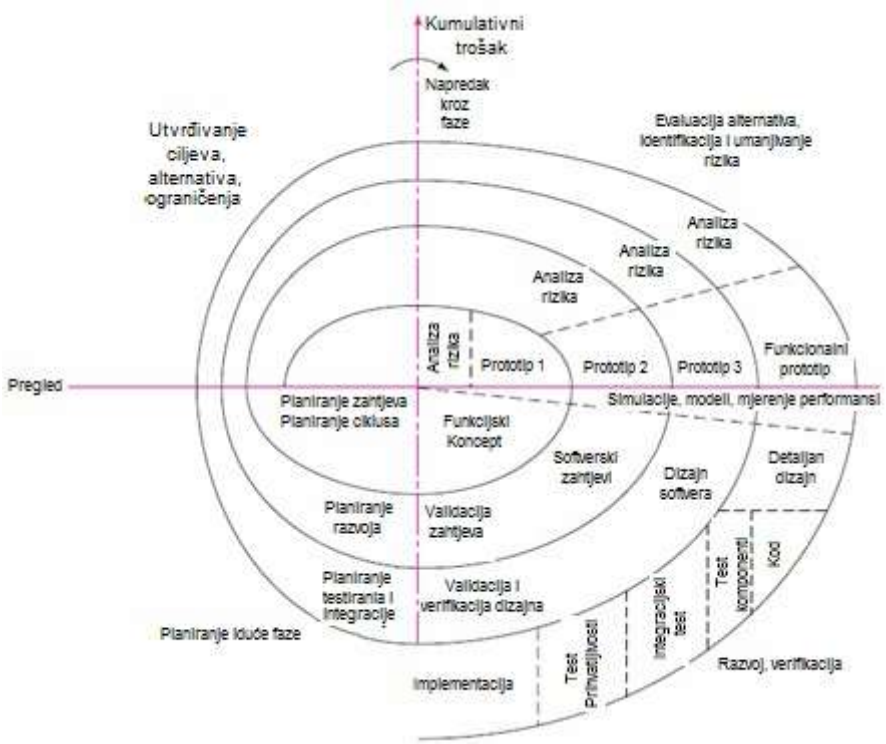
Slika 27. Prototipiranje;
(Izvor: Valachic George Hoffer, 2014.)

Nakon četvrtog koraka se prototipiranje vraća na treći korak sve dok rješenje nije sasvim ili dovoljno zadovoljavajuće i tada se može razvijati konačan proizvod osim u slučaju kada se taj korak preskače jer se događa da korisnik usvaja zadovoljavajući prototip. (Laudon i Laudon, Ne razvija se svaki prototip s ciljem da se nadograđuje dok ne postane reprezentativan konačnom proizvodu. Neki prototipi se razvijaju samo za potrebe istraživanja zahtjeva koji se pri utvrđivanju onda implementiraju kao funkcionalne komponente u originalno rješenje dok se prototip odbacuje.

4.8. Spiralni model

Spiralni model životnog ciklusa razvoja (Slika 28.) pojavljuje se kao formalizirana metodologija životnog ciklusa razvoja 1986. godine u članku autora Barryja Boehma pod nazivom: “A Spiral Model of Software Development and Enhancement“ u kojem je model predstavljen kao „specifičan u odnosu na dotadašnje modele po tome što se njime fokus u pristupu razvoju stavlja na rizik umjesto na dokumentiranje ili kodiranje.“ (Boehm, 1986.)

Metodologijom razvoja kojom se slijedi spiralni model izvođenja razvojnih aktivnosti mogu se primjenjivati kombinacije dosad razrađenih pristupa razvoju ovisno o primjerenosti pristupa za projekt koji će se razvijati. Modelom se predviđa upotreba, vodopadnog modela, iteracija, inkremenata i prototipa, a smatra se dobrom alternativom vodopadnom pristupu razvoja kod velikih rizičnih projekata stoga što se posebno fokusira na otklanjanje rizika baveći se prvo utvrđivanjem i razvojem problematičnih i kritičnih djelova sustava. Kao što sam naziv kaže model izgleda kao spirala po kojoj se izvode aktivnosti razvoja slijedom opisanim po svakoj spirali.



Slika 28. Spiralni model;
(Izvor: Boehm, 1986.)

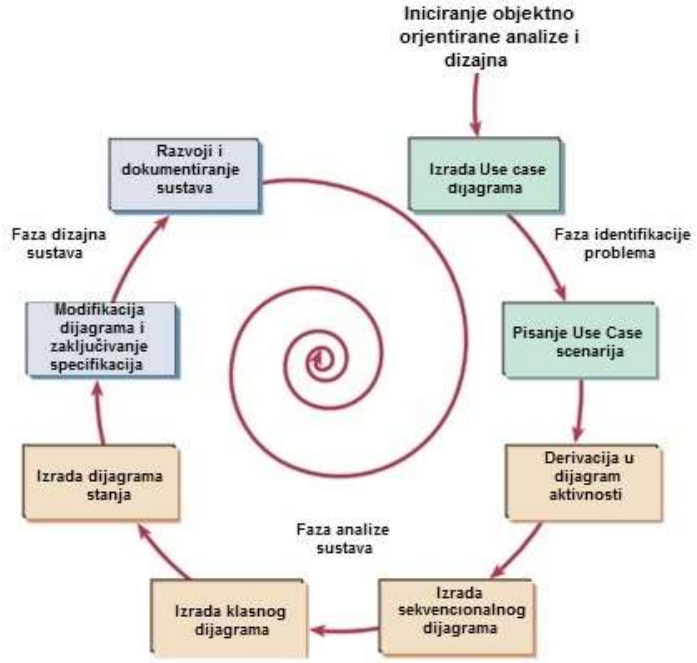
Spirala se odvija na križanju u kojem apscisa i ordinata, to jest, horizontalna i vertikalna os predstavljaju odnos napretka razvoja i troškova koji se akumuliraju. Proces razvoja započinje planiranjem i utvrđivanjem zahtjeva pri začetku prve unutarnje spirale, a zatim nastavlja po spirali slijedeći faze, redom: fazu utvrđivanja ciljeva, alternativa i ograničenja, fazu procjenjivanja alternativa i ublažavanja rizika izvođenjem analize rizika, fazu dizajna čije aktivnosti na spirali odgovaraju verziji prototipa ili iteracije pa prva spirala u slučaju npr. prvog preliminarnog prototipa slijedi aktivnosti dizajna osnovnih prioritetnih funkcionalnosti koje se onda implementiraju čime se zaključuje prva spirala i planiranjem ponovno započinje nova koja zatim slijedi odgovarajuće aktivnosti dalje do sljedeće i tako dalje sve do zaključenja projekta dobivanjem konačnog rješenja ili odbacivanjem.

5. OBJEKTNO - ORIJENTIRANI PRISTUP RAZVOJU IS-a

5.1. Objektno orijentirani pristup

Objekt je apstraktni prikaz nečega u stvarnom svijetu. Prikazivanje objekata apstrakcijom je način kojim se objekt opisuje koristeći relevantne informacije o objektu. Objektno-orijentirani pristup razvoju može primjenjivati bilo koju od prethodno obrađenih metodologija ali ponajviše se povezuje s iterativnim brzim razvojem aplikacija - tzv. RAD metodologijom. Osnovna razlika između tradicionalnog pristupa razvoju kao npr.strukturalnog razvoja i objektno-orijentiranog pristupa je u načinu na koji se složeni problem "razbija" na manje probleme. U tradicionalnom pristupu, dekompozicija problema se radila ili na temelju procesa ili na temelju podataka dok se objektno orijentiranim pristupom radi dekompozicijom na objekte koji sadržavaju i procese i podatke. (Dennis, Wixom i Roth, 2011.)

Proces razvoja objektno-orijentiranih sustava započinje studijom izvedivosti i analizom postojećeg sustava. U ovoj fazi se definiraju objekti novog sustava zajedno s njihovim atributima i operacijama. Analitičari zatim modeliraju interakcije između objekata na način koji će omogućiti sustavu da ispunjava svoju svrhu. Objektno-orijentirani razvoj sustava sastoji se od progresivnog razvoja prikaza komponenti sustava primjenom različitih stupnjeva apstrakcije kroz faze analize i dizajna.



Slika 29. Objektno orijentirani razvoj;
Izvor: Kendall, K. E. i Kendall, J. E., 2006.

U ranim fazama razvoja, model je kreiran kao apstrakcija prikazujući vanjske karakteristike sustava kao što su strukture podataka, tempiranje i slijed procesiranja operacija te način na koji korisnici vrše interakciju sa sustavom. Kako se model razvija postaje sve manje apstraktan i prikazuje način na koji će sustav biti građen i kako bi trebao funkcionirati u stvarnom svijetu. Na temelju tih modela se zatim dizajn implementira koristeći programske jezike i/ili sustav za upravljanje bazama podataka. Tehnike i notacije u obliku dijagrama koje primjenjuje standardni objektno-orijentirani jezik se zovu UML - Unified Modelling Language. (Valacich, George i Hoffer, 2011.)

U nekim slučajevima, analitičari mogu ponovno koristiti postojeće objekte iz drugih aplikacija (Ili iz biblioteke objekata) u novom sustavu, čime se štedi vrijeme koje bi se inače provelo u kodiranju. U većini slučajeva, međutim, čak i uz ponovnu upotrebu objekata, javlja se potrebe za dodatnim kodiranjem radi prilagodbi objekata i interakcija.

Prednosti objektno-orijentiranog pristupa:

1. Smanjuje složenost razvoja sustava i rezultira sustavima koje je jednostavnije i brže razviti i održavati jer je svaki objekt relativno mali i nezavisan.
2. Poboljšava produktivnost i kvalitetu programa. Nakon definiranja, implementacije i testiranja objekta može ga se ponovno upotrijebiti u drugim sustavima.
3. Sustavi razvijeni ovom metodologijom su fleksibilniji - lakkše ih je modificirati, nadograđivati i poboljšavati mijenjanjem vrsta objekta ili dodavanjem novih.
4. Ovaj pristup omogućuje analitičarima pogled na sustav iz druge perspektive temeljene na stvarnom svijetu umjesto na programskom jeziku.
5. Sustav temeljen na generičkim modelima kakve primjenjuje ovaj pristup će imati dulji životni vijek nego sustavi razvijani za rješavanje konkretnih, aktualnih problema.
5. Objektno-orijentirani pristup je idealan za razvoj web aplikacija.
6. Objektno-orijentirani pristup prikazuje različite elemente informacijskog sustava iz perspektive korisnika zbog čega mu je i razumljiviji

Nedostaci objektno-orijentiranog pristupa:

Sustavi razvijeni ovim pristupom, posebice korištenjem Java programskog jezika imaju tendenciju biti sporiji od onih razvijenih drugim programskim jezicima. Također, mnogim programerima fale vještine i iskustvo s objektno-orijentiranim jezicima. (Information systems development, n.d.)

5.2. Unified Modelling Language

Istovremeno s uspostavljanjem radne skupine za rad na Objektno-orijentiranoj analizi i dizajnu, 1997. godine, OMG grupa (izv. Object Modelling Group) s glavnim autorima Rumbaughom, Jacobsonom i Boochem razvija UML (izv. Unified Modelling Language). UML je standardni jezik za kreiranje objektno-orijentiranih modela i dijagrama kojima se predstavlja informacijski sustav primjenom zajedničkih notacija i semantike za vizualiziranje strukture i ponašanja sustava, primjenjujući pritom najbolje prakse. Cilj razvoja UML-a bio je omogućavanje primjene stabilnog i zajedničkog dizajnerskog jezika koji će omogućiti razvojnom osoblju, s jedne strane, razvoj bez sputavanja propisivanjem načina primjene notacija, a s druge strane, čitanje i razlučivanje strukture i ponašanja predstavljenog sustava. Ovaj standard je ubrzo zadobio podršku razvojne zajednice i vodećih tržišnih kompanija na području razvoja sustava, poput IBM-a i Logix-a. (Pender, 2002.)

UML specificira dvije kategorije dijagrama: dijagrame kojima se modelira struktura i dijagrame kojima se modelira ponašanje sustava. Struktrurnim dijagramima se prikazuje statička struktura sustava i njegovih dijelova na različitim razinama apstrakcije i primjene i načina na koji su međusobno povezani. Dijagrami ponašanja sustava prikazuju dinamičko ponašanje objekata sustava koje se opisuje serijom vremenski određenih promijena u sustavu.

Kako bi se učinkovito prikazao složeni sustav, može se kreirati model kojim će se sustav prikazati iz više različitih nezavisnih perspektiva koristeći primjerene vrste dijagrama. Time je moguće lakše i preciznije analizirati, dizajnirati i implementirati sustav s obzirom na konzistentnost i zahtjeve. (Valacich, George i Hoffer, 2011.)

5.2.1. Use Case Dijagram

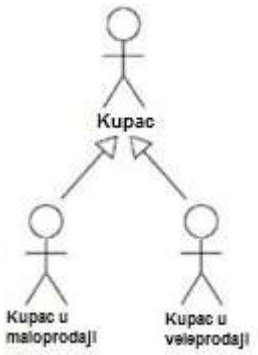
Use case dijagram je grafički prikaz ponašanja sustava iz perspektive korisnika i vrlo je jednostavno razlučiv. Predstavlja prikaz sustava kao jednodimenzionalne "crne kutije" sa opisanom granicom u obliku pravokutnika koja ga dijeli od okoline i funkcijama koje sadržava unutar granice, a one su opisane kao instance korištenja ovalnim simbolima.

Use case ili slučaj korištenja je način na koji korisnik može koristiti funkciju/e sustava da bi ostvario neki cilj, a specifični načini opisuju se use case scenarijima kako bi se utvrdili svi načini na koje korisnik treba moći komunicirati sa sustavom kako bi se

ispunile potrebe korisnika i sve funkcije koje sustav treba sadržavati kako bi se ispunila svrha postojanja sustava.

Korisnici su na dijagramu predstavljeni prepoznatljivim tzv. „stick“ figurama, a nazivaju se **akterima**. Akter može može predstavljati jednu fizičku osobu ali može predstavljati i pravnu osobu npr. instituciju, banku ili neko poduzeće. Akter također može biti i neki vanjski sustav.

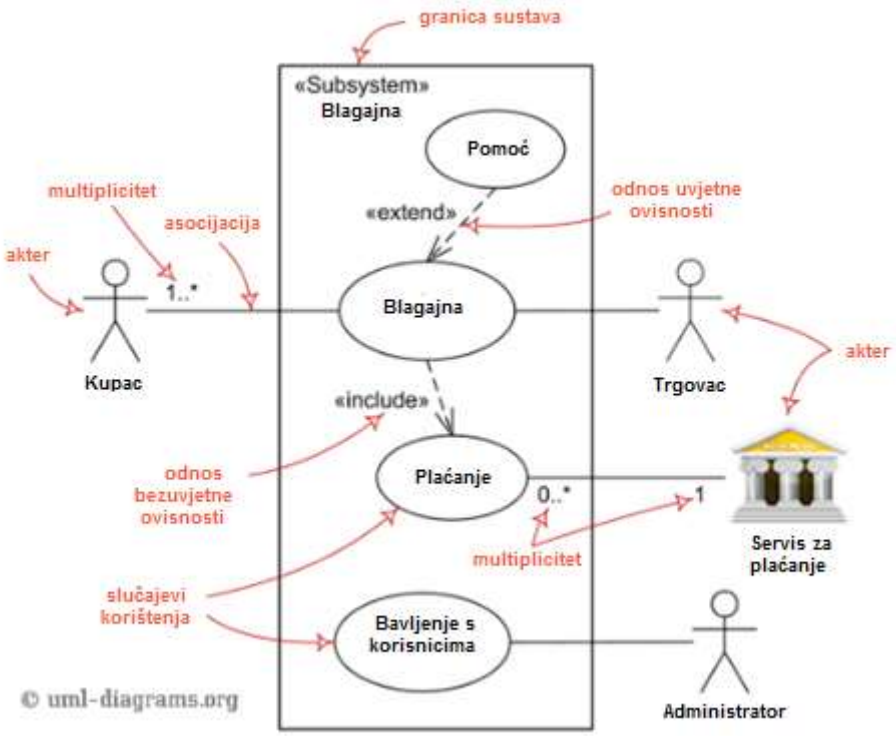
Interakcija koju akteri vrše s funkcijama sustava je predstavljena poveznicama koje se zovu **asocijacije** a prikazane su ravnim linijama koje povezuju aktere i funkcionalnosti. Osim asocijacija postoje još i veze **ovisnosti** kod kojih razlikujemo dvije vrste: ***<<include>>*** i ***<<extend>>***. <<include>> predstavlja vezu između dvije funkcionalnosti čije su operacije ovisne - dakle, izvršavanje jedne podrazumijeva da se i druga mora izvršiti. <<extend>> je veza uvjetne ovisnosti što znači da izvršavanje jedne operacije podrazumijeva izvršavanje druge samo u određenim slučajevima. (Pender, 2002.) Još jedna veza je veza **generalizacije** koja podrazumijeva postojanje nadklasa i podklasa. Npr. u slučaju kada imamo klijenta koji može biti fizička ili pravna osoba, klijent može biti generalna, to jest opća klasa, a fizička i pravna osoba mogu biti podklase i ta se veza prikazuje tako da se podklase povezuju sa nadklasom punim strelicama na strani nadklase, kao što možemo vidjeti ovdje:



Slika 30. Notacija odnosa Nadklasa – Podklasa;
(Izvor: https://www.uml-diagrams.org/use-case-actor.html , datum pristupa: 08.09.2018.)

Po potrebi se u dijagramu može koristiti i **multiplicitet**, koji ima sličnu funkciju kao kardinalnost kod entity relationship dijagrama, a kojim se određuje koliko instanci aktera sudjeluje u asocijaciji. Prikazuje se odgovarajućim brojevima ili brojčanim intervalima i/ili zvjezdicama na na pripadajućim stranama linija veza. Jedinstvenim brojem se koristimo kada je predviđen točan konkretan broj instanci, zvjezdicom se koristimo kada želimo označiti da se radi o, općenito, više instanci ali najčešće je to

interval koji nam govori da se radi o nekom broju koji može biti od ***n*** – do ***n***, a može se prikazati kao **0**..**1**, **0**..*, **1**..*. (Valacich, George i Hoffer, 2011.)



Slika 31. Use case dijagram;
(Izvor: https://www.uml-diagrams.org/use-case-diagrams.html , datum pristupa: 08.09.2018.)

Uzmimo jedan jednostavan primjer sustava plaćanja na slici . Ovaj dijagram možemo razložiti kao jedan slučaj korištenja: Kupac dolazi na blagajnu gdje ga uslužuje prodavač. Kupac na blagajni vrši plaćanje, a ukoliko mu je potrebna nekakva pomoć može je zatražiti. Plaćanje se procesira preko nekakvog vanjskog servisa. Sustav održava administrator. Pritom se možemo nadovezati na notacije kako bismo shvatili prirodu korištene simbolike. Iz Slike dijagrama možemo istakuti nekoliko elemenata koji određuju slučaj korištenja i pobliže ga opisuju. Imamo sustav čije su granice prikazane pravokutnikom i on nosi naziv „Naplata“. Sustav koristi nekoliko aktera iz različitih uloga, kao što je to „Kupac“. Interakcija između „Kupca“ i sustava je prikazana linijom koja ih povezuje i na kojoj je, u ovom slučaju, određen i multiplicitet koji glasi: „1..*“ a njime je rečeno da sustav „Naplate“ koriste 1 ili više (*) „Kupaca“, a gdje je * nedefinirano više. Funkcija „Naplata“ koju koristi „Kupac“ ima potencijalnu funkciju „Pomoć“ kojoj on može pristupiti – kao ekstenziji primarne funkcije, dok je funkcija „Plaćanje“ obuhvaćena podfunkcija funkcije „Naplata“ koju ona uključuje.

5.2.2. Dijagram klasa

Klasni dijagram je najčešće korišteni dijagram u modeliranju kod objektno-orijentiranog razvoja sustava, a kao grafički prikaz prikazuje njegovu statičku strukturu primjenom skupa klasa, sučelja i različitih vrsta veza i odnosa. **Klasa** je apstrakcija, simbolički prikazana kao pravokutnik, kojom se opisuje skup objekata koji dijele iste vrste atributa, operacija, veza i semantika. Na slici 32. je predstavljen primjer jedne klase.

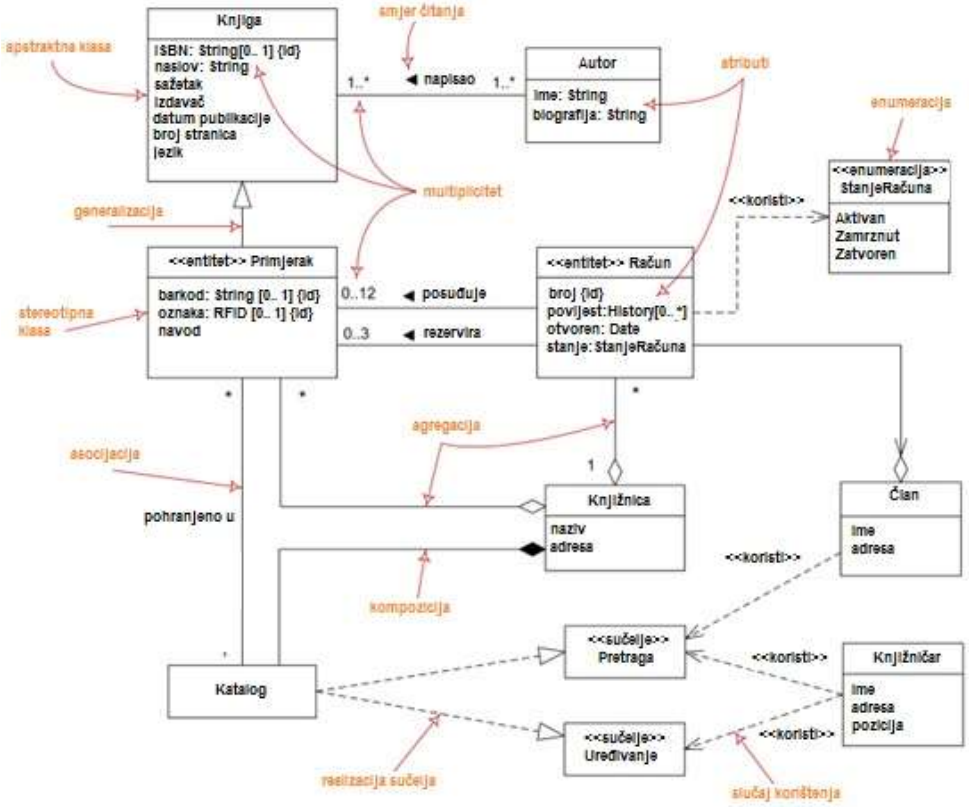


Slika 32. Primjer klase;
(Izvor: https://www.uml-diagrams.org/class-diagrams-overview.html , datum pristupa: 08.09.2018.)

Klasa se naziva „Knjiga“ i predstavlja sve objekte koji joj kategorički pripadaju, to jest sve stavke koje su u bazi podataka tako klasificirane pri unosu. U ovom slučaju će to, primjereno, pojedinačno predstavljati raznorazna književna djela u inventoriju, a za svako književno djelo biti će navedeni atributi koji svojstveno pripadaju pojedinačnom objektu kao članu klase, u ovom slučaju: ISBN (međunarodni standardni knjižni broj), naziv, sažetak, izdavač, godina izdanja, broj stranica i jezik. **Atribut** može biti predstavljen u obliku raznih znakova, može biti jedan znak ili niz znakova, a može biti i statičan ili varijabilan. Ovo je određeno postavljanjem primjerenih funkcija koje uvjetuju ograničenja korištenih znakova, kao što je to na primjeru sa slike vidljivo kod atributa „naslov“ koji ima funkciju „string“, a njome se podrazumijeva da su instance atributa „naslov“ predstavljene nizom znakova – slova, brojeva i slično.

Posebna vrsta atributa u klasi su oni koji predstavljaju **primarne ključeve**. Atribut koji je primarni ključ u instancama objekata klase djeluje kao šifra koja jedinstveno označava objekt. Na primjeru sa slike imamo identificiran primarni ključ u atributu „ISBN“, što znači da svaka konkretna instanca objekta klase, to jest svaka konkretna knjiga ima kao atribut jedan konkretan ISBN broj koji je svojstven njoj i samo njoj te je on prema tome identifikator te konkretne knjige. Klasa može sadržavati više od jednog primarnog ključa, to jest više atributa koji su jedinstveni za svaku pojedinu instancu klase. U toj situaciji se radi o **složenom ili kompozitnom ključu**. Klasa može i sadržavati atribut koji je primarni ključ neke druge klase. U tom slučaju se u toj klasi taj atribut naziva **stranim ključem.**

Klasa može, osim odjeljka s nazivom i odjeljka s atributima, sadržavati i odjeljak s **operacijama**. Odjeljak s operacijama nalazi se ispod odjeljka s atributima. Operacije koje su navedene u klasi predstavljaju neku akciju ili proces koji klasa može provesti. Ako npr. imamo klasu „Katalog“ kao na slici, ona može sadržavati operaciju „search()“ koja omogućava obavljanje pretrage. Klasa „Račun Člana“ bi mogla imati operaciju „zaduži() ili rezerviraj()“. Atributi i operacije mogu biti vidljivi ili nevidljivi ovisno o prilagodbi kojom se specificira privatnost na način da atributu ili operaciji prethodi tekst: „public:“ ili „private:“.



Slika 33. Primjer klasnog dijagrama;
(Izvor: https://www.uml-diagrams.org/class-diagrams-overview.html , datum pristupa: 08.09.2018.)

Posebna vrsta klase je **apstraktna klasa** koja postoji samo kao posredna klasa iz koje druge klase (**stereotipne**) naslijeđuju. Ni jedan objekt ne može biti izravna instanca apstraktne klase ali može biti neizravan preko podklase koja nije apstraktna. U slučaju na slici, apstraktna klasa „Knjiga“ predstavlja generalizaciju stereotipne klase „Primjerak“. **Generalizacija** u ovom slučaju ima istu funkciju kao kod generalizacije u use case dijagramima. Imamo podklase i nadklase gdje su podklase vrste nadklasa. Podklase naslijeđuju od nadklase, što znači da atributi i operacije nadklase također vrijede i za podklase. Generalizacija se prikazuje linijom s praznim trokutom prema nadklasi.

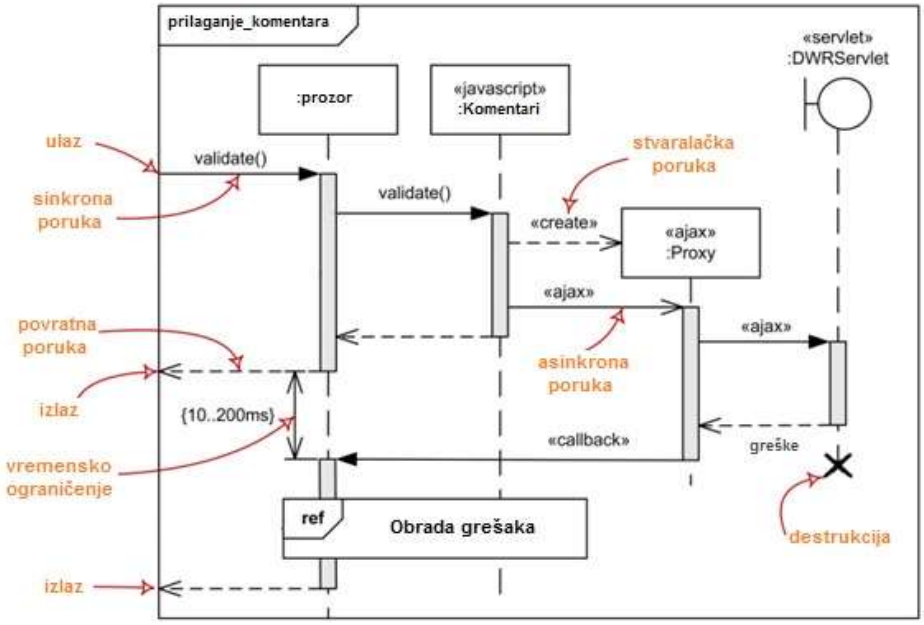
Veza agregacije se koristi kada su neke klase sastavljene od drugih klasa. Postoje dvije vrste agregacije: 1. **slaba agregacija** koja nam govori da klasa koja sačinjava drugu klasu može postojati bez nje. U slučaju našeg dijagrama na slici to je prikazano odnosom klase "Primjerak" i klase "Knjižnica". Knjižnica se sastoji od knjiga ali knjige mogu postojati i nezavisno od knjižnice. Ova vrsta agregacije je prikazana linijom s praznim rombom prema klasi koja čini cjelinu. 2. **kompozicija** ili jaka agregacija koja nam govori da klasa koja sačinjava drugu klasu ne može postojati bez druge klase, znači, dio koji ne može postojati bez cjeline. U slučaju našeg primjera to je vidljivo kod klase "Katalog" koja se odnosi na katalog jedne specifične knjižnice pa stoga on kao takav ne može postojati bez te konkretne knjižnice i zato ova klasa prema klasi "Knjižnica" ima odnos kompozicije koja se prikazuje linijom sa zacrnjenim rombom prema klasi koja predstavlja cjelinu. (Booch, Rumbaugh i Jacobson, 1998.)

Enumeracija je korisnički definiran tip podataka kojim se nazivi dodjeljuju ograničenim numeričkim vrijednostima varijabli – npr. kada se uzmu dani u tjednu: „dan“ je varijabla koja će brojčanom vrijednošću biti ograničena na raspon od 1 do 7 (budući da je sedam dana u tjednu), a svakom od tih brojeva će biti dodijeljen odgovarajući naziv: Ponedjeljak, Utorak, Srijeda, itd.

Realizacija sučelja se ostvaruje kada klasa implementira operacije definirane sučeljem. To je u primjeru vidljivo u slučaju gdje klasa „Član“ koristi (što je prikazano isprekidanom strelicom s opisom ***<<koristi>>)*** sučelje (prikazano pravokutnikom s opisom ***<<sučelje>>,*** naziva „Pretraga“) preko kojeg vrši pretraživanje knjižničke arhive koje se izvršava pokretanjem operacije automatskog pretraživanja klase „Katalog“ čiji se rezultati tada prikazuju preko sučelja (prikazano isprekidanom linijom sa trokutom prema sučelju) i tada nastupa realizacija. Realizacija ovisi o korištenju. (UML diagrams n.d.).

5.2.3. Sekvencijalni dijagram

Ovaj dijagram spada u dijagrame interakcija kojima se prikazuje ponašanje sustava prikazivanjem interakcija između aktera definiranih specifičnim vremenskim redoslijedom. Interakcije su prikazane u obliku „poruka“ koje se izmijenjuju između objekata ili aktera. (Booch, Rumbaugh i Jacobson, 1998.)



Slika 34. Primjer sekvencijalnog dijagrama;
Izvor: https://www.uml-diagrams.org/sequence-diagrams.html , datum pristupa: 10.09.2018.)

Pravokutni okvir koji omeđuje dijagram predstavlja granicu sustava. Sustav ima svoj ulaz i izlaz koji su prikazani ulaznim i izlaznim strelicama.

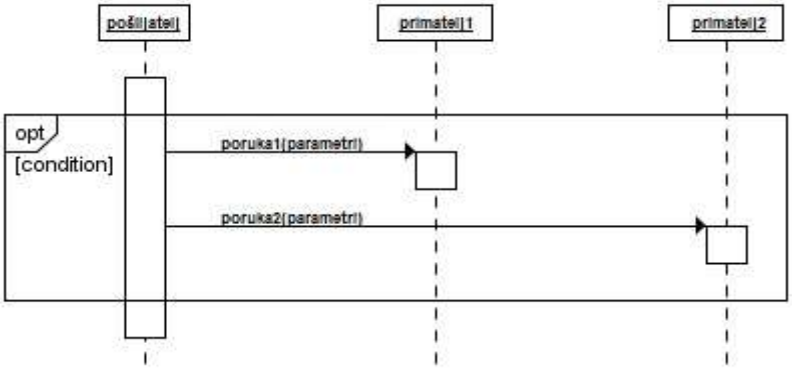
Aktere obično predstavljaju figure kao u Use Case dijagramu ili pravokutnicima dok objekte obično predstavljaju samo pravokutnici. Objekti mogu biti označeni kao različiti stereotipi. Stereotipi opisuju aktere ili objekte notacijom koja je u dijagramu prikazana kao: „***<<stereotip>>“***.

Dijagram ima okomitu i vodoravnu dimenziju. Svaki akter ima svoju „**životnu liniju**“ koja je prikazana isprekidanom okomitom linijom na kojoj je vidljiv redoslijed poruka/poziva. Poruke i pozivi između aktera su prikazani vodoravnim linijama koje se izmjenjuju između životnih linija. Poruke pozivanja pokreću određene događaje i operacije koje jedan akter zahtjeva od drugog da bi zatim rezultat bio vraćen akteru pozivanja, proslijeđen ili predstavljao završetak komunikacije. Smjer komunikacije prikazan je smjerovima strelica, a vrsta strelice čini razliku između vrste poruke. Tako:

- puna strelica pune linije predstavlja ***sinkronu poruku*** – poruku kojom se poziva na operaciju i čeka na procesiranje,
- prazna strelica pune linije predstavlja ***asinkronu poruku*** – poruku čijim slanjem akter pošiljatelj potom ne čeka na procesiranje aktera primatelja
- prazna strelica isprekidane linije predstavlja ***povratnu poruku.*** Povratnu poruku nepotrebno je uključiti ako ne postoji vraćanje neke vrijednosti.

Priroda poruka može biti **stvaralačka** ili **destrukcijska**. Stvaralačkom porukom se kreiraju novi objekti ili akteri u sekvenci dok destrukcija prekida sekvencu ili okončava vremensku liniju objekta/aktera, npr. uslijed neke nastale greške ili prirodnog završetka procesa. (Sequence Diagrams n.d.)

Interakcija između aktera također može biti uvjetovana. Takvu interakciju i povezane notacije možemo vidjeti na slici 34. Uvjetovana interakcija prikazuje se uokvirena pravokutnikom u čijem se gornjem lijevom kutu naznačuje uvjet u uglatoj zagradi.

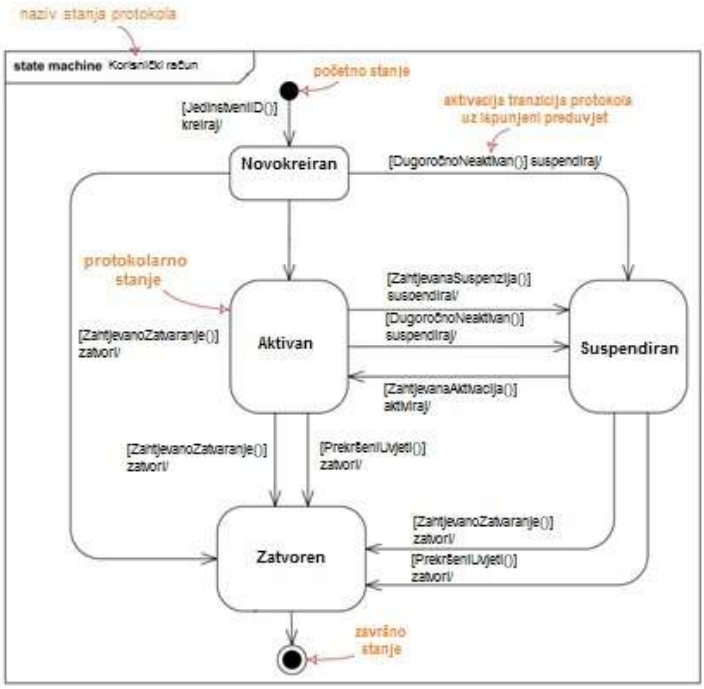


Slika 35. Primjer uvjetovane interakcije, (Izvor: http://www.tracemodeler.com/articles/a_quick_introduction_to_uml_sequence_diagrams/ , datum pristupa: 10.09.2018.)

Ovdje je prikazan hipotetski slučaj u kojem pod određenim uvjetom akter pošiljatelj poruke paralelno šalje poruke različitim primateljima. Osim toga, još jedna moguća uvjetovana vrsta interakcije je interakcijska petlja. Interakcijska petlja koristi identične notacije s osnovnom razlikom u operateru. U prethodnom primjeru taj je operater u gornjem lijevom kutu naznačen kao „opt“ (optional), dok u ovom slučaju operater „loop“ određuje petlju kojom se, dok god je neki uvjet ispunjen, interakcija kontinuirano ponavlja. (Sequence Diagrams Intro. n.d.)

5.2.4. Dijagram stanja

Dijagram stanja, vidljiv na slici 35, prikazuje različita stanja kroz koja klasa prolazi i način na koji prelazi iz jednog stanja u drugo. Uobičajeno je da se modeliraju stanja onih klasa koje mogu imati tri ili više mogućih stanja.



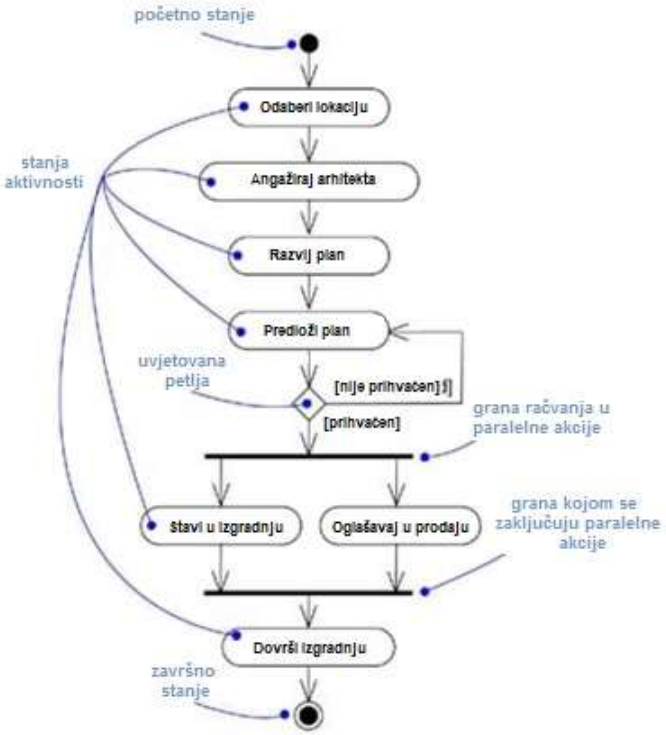
Slika 36. Primjer dijagrama stanja
(Izvor: https://www.lucidchart.com/pages/uml/state-machine-diagram, datum pristupa: 10.09.2018.)

Ovaj dijagram se sastoji od četiri osnovna elementa, a to su: početno stanje koje je prikazano kao zacrnjeni krug, a podrazumijeva neko hipotetsko prvo stanje, prijelazi između stanja koji se prikazuju kao strelice, stanja koja se prikazuju kao epiteti unutar pravokutnika zaobljenih rubova i točka završetka prikazana kao zacrnjeni krug unutar kruga. Uvjeti prijelaza i prijelazne naredbe navode se iznad strelica prijelaza.

Jednostavniji dijagrami razlažu se na manje stanja od kojih se neka mogu dalje detaljnije razložiti, a tu se dakako radi o različitim potencijalnim razinama apstrakcije koje se prikazuju ovisno o potrebi. (Lucidchart State Machine, n.d.)

5.2.5. Dijagram aktivnosti

Dijagram aktivnosti prikazuje izvedbu nekog procesa kao kronološku strukturu toka naredbi. Dijagram aktivnosti je sličan dijagramu stanja utoliko što također prikazuje akcije povodom naredbi koje se odvijaju u procesu, a koje dovode do prijelaza jednog stanja u drugo ali ne bilježi sama stanja.



Slika 37. Primjer dijagrama aktivnosti
(Izvor: https://www.lucidchart.com/pages/uml/activity-diagram, datum pristupa: 11.09.2018.)

Jednako kao u dijagramu stanja, i ovaj dijagram započinje sa zacrnjenim krugom kao početnim stanjem koje inicira početnu aktivnost. Tok aktivnosti i njegov smjer određen je strelicama. Aktivnosti su prikazane u ovalima a opisane su kao naredbe.

Na određenim mjestima prikazana je na strelica toka koja prelazi u račvanje. U tom slučaju radi se o toku u kojem neka aktivnost završava da bi potom poslije nje započele dvije ili više paralelnih aktivnosti. Obrnuta situacija je u slučaju kada aktivnosti, koje su dotad tekle paralelno, istovremeno završavaju te prelaze u jednu granu nakon koje se nastavlja odvijati jedna aktivnost.

Aktivnosti mogu biti povezane s drugim aktivnostima prijelaznim linijama ili na točku odluke prikazanu rombom koja nam govori je li se naredba izvršila ili nije te, s obzirom na taj uvjet, proslijeđuje na odgovarajuće aktivnosti. U nekim slučajevima akcije mogu otpočeti samo ukoliko je prethodno izvršen neki proces. Radi se o uvjetovanoj petlji. Ta uvjetovana petlja je u dijagramu prikazana rombom i može voditi u nastavak toka aktivnosti ili opisivati petlju u kojoj se tok kontinuirano vraća u jednu aktivnost sve dok nije zadovoljen uvjet. Aktivnosti koje zaključuju modelirani proces vode završnom stanju jednako kao i u dijagramu stanja.

Ovaj dijagram se može koristiti u slučajevima kada je potrebno prikazati: logiku nekog algoritma, akcije koje koje se izvšavaju u slučajevima korištenja, poslovni proces, tok interakcija između korisnika i nekog sustava, funkcije i operacije elemenata u aritekturi sustava koji se modelira. (Activity diagram, n.d.)

5.3. Rational Unified Process

Rational Unified Process je rezultat suradnje između nekadašnjih korporacija Rational Software-a i Objectory AB-a, a današnjeg IBM-a. RUP predstavlja okvir koji opisuje proces razvoja softvera temeljen na onim što se smatra najboljim praksama. Radi se zapravo o standardiziranim smjernicama i obrascima koji se prilikom razvoja slijede a dostupni su u obliku pretražive baze znanja. Kombinacija su modela iterativnog procesa razvoja i objektno orijentirane metode dizajna. (Edeki, 2013). Splet šestero najboljih praksi koje se navode u RUP smjernicama sastoji se od:

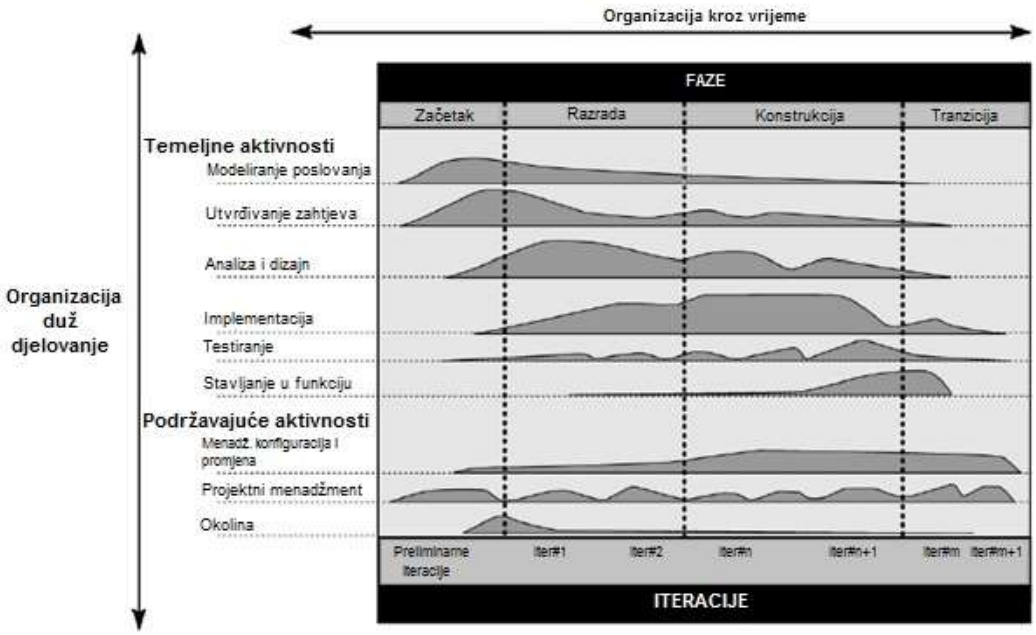
1. **Iterativnog razvoja softvera** – razlog tome što se razvoj iteracija s izgradnjom u inkrementima smatra najboljom praksom je u složenosti današnjih sustava za čiji je razvoj vrlo riskantan tradicionalni pristup razvoja na temelju inicijalnog utvrđivanja zahtjeva i testiranja pri kraju. Rizik se smanjuje dobivanjem feedbacka korisnika kroz iteracije i testiranjem pojedinačnih inkrementa. Ova praksa također olakšava mjerenje napretka razvoja.
2. **Menadžmenta zahtjeva** – RUP-om je opisan način utvrđivanja, organiziranja i dokumentiranja korisničkih zahtjeva.
3. **Izgradnje arhitekture na temelju komponenti** – komponente predstavljaju objekte koji tvore funkcije sustava a uz pomoć kojih se gradi arhitektura sustava bez potrebe da se za te funkcije zasebno „piše“ kod, čime se skraćuje vrijeme razvoja, smanjuje vjerojatnost grešaka u kodu, ostvaruje mogućnost ponovnog iskorištavanja komponenti i integriranja starih i novih komponenti.
4. **Vizualnog modeliranja softvera** – što se postiže UML-om a svrha mu je vizualno predstaviti sustav na različitim nivoima apstrakcije. To olakšava pronalaženje strukturnih i logičkih grešaka otklanjajući potrebu bavljenja kodom i detaljima za koje je bitno da se naknadno usklade sa smislenim i zadovoljavajućim modelom. Također je sustav lakše predstaviti i objesniti drugima uz pomoć modela.

5. **Verificiranja kvalitete softvera** – kontinuirano testiranje kvalitativnih karakteristika softvera kao što su pouzdanost, funkcionalnost, brzina odaziva i sl. korištenjem objektivnih

mjera i usporedbom s utvrđenim kriterijima kroz proces razvoja

6. **Upravljanja promjenama softvera –** iterativnim razvojem se predviđa potreba za unošenjem promjena u skladu s ažuriranim zahtjevima pa zato također postoji i potreba za upravljanje tim promjenama praćenjem učinjenog radi održavanja konzistentnosti čije je postizanje bitno kod integracije inkremenata.

Ove prakse objedinjuju se u procesu razvoja koji se može prikazati njegovim statičkim i dinamičkim aspektima Slikom 38. Statički aspekt predstavljen je vertikalom koja obuhvaća devet čimbenika procesa od kojih svaki predstavlja po jedan tok skupa aktivnosti koji se mora izvesti u fazama kroz vrijeme, a koje je, kao dinamički aspekt, predstavljeno horizontalnom osi.



Slika 38. Rational Unified Process
(Izvor:https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf, datum pristupa: 07.06.2017.)

Slijed skupa aktivnosti ili zadataka u konačnici treba rezultirati nekim svrhovitim outputom koji postaje inputom u sljedećoj fazi. Kao što je to slikom prikazano, sveukupno je devet tih skupova aktivnosti od kojih je šest temeljnih i troje podržavajućih. ***Šest temeljnih skupova aktivnosti jesu***: Modeliranje poslovanja, Utvrđivanje zahtjeva, Analiza i dizajn, Implementacija, Testiranje, Stavljanje u funkciju ***Troje podržavajućih skupova jesu***: Menadžment konfiguracija i promjena, Projektni menadžment i Okolina.

Vremenska dimenzija prikazana je kroz faze koje se odvijaju za svaku iteraciju, a te faze jesu:

1. **Faza začetka –** u kojoj je cilj ustanoviti poslovne funkcije sustava i odrediti njegov raspon i granice. Navedeno se postiže određivanjem svih vanjskih objekata s kojima će sustav vršiti komunikaciju identificiranjem svih slučajeva korištenja. U ovoj fazi se također planski utvrđuju potrebni resursi, izrađuje plan razvoja, uključujući očekivane točke napretka raspoređene u fazama razvoja te izvodi procjena rizika.
2. **Faza razrade –** u kojoj je cilj provesti analizu sustava: odrediti osnovnu arhitekturu i ukloniti najrizičnije elemente projekta. Navedeno se postiže detaljnijom razradom sustava s obzirom na funkcionalnosti i zahtjeve performansi. Na temelju toga zatim nastaje testni prototip nad kojim se može provesti potrebne evaluacije i procjeniti kolika su odstupanja od planiranog i, u iteracijama, prilagoditi.
3. **Faza izgradnje –** u kojoj se razvijaju sve preostale komponente i svojstva i integriraju u potencijalno rješenje da bi se, potom, ono ponovno testiralo na integracijsku kompatibilnost kao i sveobuhvatnu funkcionalnost radi osiguranja kvalitete. Odgovarajući konačni proizvod se izdaje zajedno s korisničkim priručnikom.
4. **Faza tranzicije –** podrazumijeva aktivnosti nužne za uspješno stavljanje proizvoda u upotrebu i uključuje dodatne iteracije, beta izdanja, „zakrpe“, popravke „bug-ova“, dodatke i ostale modifikacije koje se temelje na povratnim informacijama korisnika. U ovoj se fazi također obavlja edukacija korisnika. (RUP IBM, n.d.)

Primjenom RUP-a moguće je postići kvalitetniji konačni proizvod i zadovoljstvo naručitelja/korisnika s obzirom na to da se oni uključuju u razvoj uvidom i testiranjem inačica pa je tako konačni proizvod bliži onom kako je zamišljen od strane korisnika. Također je moguće smanjiti rizik i troškove fokusom na kritične elemente sustava i kontiuniranim testiranjem. S druge strane, RUP zahtjeva dobar menadžment projektom, stručne kadrove i fleksibilnost proračuna i rasporeda.

6. ZAJEDNIČKI RAZVOJ APLIKACIJA

Tradicionalnim SDLC razvojem, analitičari intervjuiraju ili neposredno pojedinačno promatraju korisnike potencijalnog novog sustava tijekom njihovog rukovanja aktualnim sustavom (ako postoji u primjeni), kako bi stekli razumijevanje o njihovim potrebama. Zatim se prikupljaju zahtjevi od kojih će se neki pokazati sličnima ali neki možda i konfliktnima. Analitičar će tada morati ponovno razmotriti konfliktne zahtjeve s korisnicima kako bi se konflikti otklonili no jednom kada su oni utvrđeni u početnoj fazi, kasnije se ne revidiraju već projekt napreduje do njegovog zaključenja što može rezultirati neuspjehom zbog nedostatka daljnjeg usuglašavanja i korisničkog testiranja. Navedeno predstavlja proces koji se može odužiti no postoji i učinkovitiji način, ukoliko je primjenjiv.

Metoda grupnog dizajna aplikacija (orig. engl. „*Joint Application Design*“) koncipirana je u sklopu IBM-a 1977. godine. kao praksa strukturiranog prikupljanja korisničkih zahtjeva u obliku grupnih sastanaka/radionica sa korisnicima u sklopu kojih se kontinuirano utvrđuje konsenzus oko plana, izvedbe i modifikacija vezanih uz dizajn informacijskog sustava i testira potencijalne predloške radi povratnih informacija. Kasnije se ovaj koncept prometnuo u praksu koja obuhvaća čitav proces razvoja tijekom cijelog životnog ciklusa pa se naziv promijenio u „Zajednički razvoj aplikacija“, odbacivši tako ograničenje na fazu dizajna.

Svaki JAD sastanak bi trebao imati jasno definirane ciljeve i utvrđeni raspored i način odvijanja. Za svaki se sastanak, s obzirom na ono što se pokušava postići, potrebno unaprijed pripremiti i po njegovu zaključenju proizvesti odgovarajuće izvješće, to jest dokumentaciju kao bitno referentno polazište za budući razvoj. (JAD Guidelines n.d.)

Proces izvođenja JAD-a odvija se kroz iteracije na način da svaka iteracija ima svoje planiranje, sesiju – kako bi se razmotrilo učinjeno i zaključenje.



Slika 39. Zajednički razvoj aplikacija;
(Izvor: http://www.chambers.com.au/glossary/joint_application_design.php , datum pristupa: 07.06.2017.)

Trajanje održavanja JAD sastanaka ovisi o rasponu i veličini projekta pa tako za male projekte može tarajati i tek pola dana no u prosjeku se radi o vremenu od 5 do 10 dana raspoređenih kroz tri tjedna. U njima obično sudjeluje 10 do 20 sudionika koji se odabiru na temelju raspolaganja potrebnim informacijama relevantnim za aktualna pitanja, a obično čine bitan splet znanja različlitih organizacijskih razina pokrivenih sustavom.

Svaki sastanak vodi osoba koja ga nepristrano usmjerava i administrira bez da i sama sudjeluje u raspravama vlastitim stavovima radi nesmetanog napretka i osiguranja da budu polučeni optimalni rezultati. Ta osoba se naziva „facilitatorom“, a uz nju djeluju još dva asistenta kao notari. Njihova je zadaća da vode spise i bilješke tijekom održavanja sastanaka, pri čemu se često koriste i CASE alatima radi pohrane i obrade informacija.

Sastanci se održavaju u za to posebno pripremljenim i tehnološki opremljenim dvoranama tako da je sudionicima dostupna sva potrebna oprema poput projektora, ploče, računala itd. Postoje i ustanovljena pravila kojih se pridržava. Relevantno je da se kroz raspravu uvažavaju mišljenja sudionika i da se vodi konstruktivni diskurs. Korisno je, pritom, služiti se modelima, use case dijagramima i prototipovima sučelja s navigacijom. (Dennis, Wixom i Roth, 2011.)

JAD sastancima se može postići:

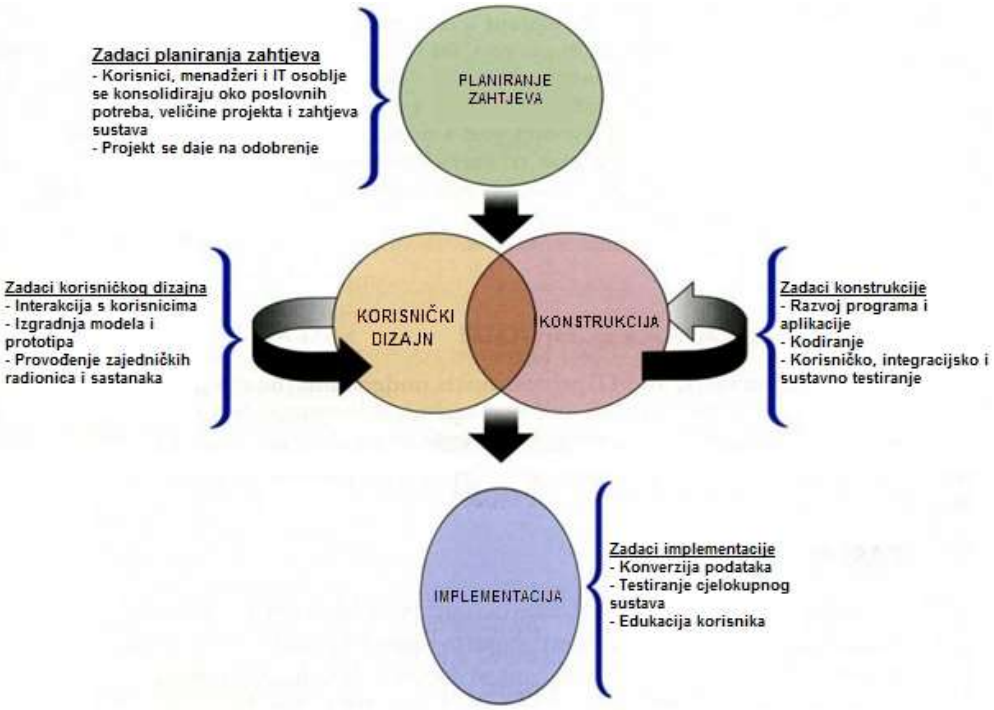
- Jednostavnost, konsolidiranjem onoga za što bi inače možda bilo potrebno mnogo dodatnih sastanaka zbog revizija u strukturiranu radionicu članova razvojnog tima, korisnika i naručitelja.
- Obostrano jasno razumijevanje zahtjeva na način relevantan za svačiju uloguu razvoju
- Identificiranje problema i ključnih sudionika
- Kvantificiranje informacija i potrebnih resursa - utvrđivanje željenog outputa uvjetuje potrebne inpute za svaku fazu
- Lakša implementacija sustava
- Smanjenje troškova edukacije korisnika koji su već pobliže upoznati sa sustavom. (JAD Guidelines n.d.)

7. BRZI RAZVOJ APLIKACIJA

Metodologija brzog razvoja aplikacija (orig. ***R**apid **A**pplication **D**evelopment – RAD*) nastala je kao odgovor na probleme tradicionalne metodologije, pogotovo kod razvoja specifičnih projekata kod kojih vrijeme igra bitnu ulogu. Vremenski zahtjevi su vrlo ograničen resurs u današnjem poslovnom okruženju pa je bilo nužno omogućiti razvoj koji će rezultirati bržim i jeftinijim informacijskim sustavom koji će i dalje udovoljavati bitnim kvalitativnim i funkcionalnim korisničkim zahtjevima.

RAD kombinira iterativni i inkrementalni razvoj i prototipiranje, a podrazumijeva primjenu raznih alata koji omogućavaju automatizaciju rada tijekom faza razvoja, tzv. CASE alata (orig. *Computer Automated Software Engineering*), grupnog razvoja aplikacija (orig. *Joint Application Development* ili, kao abrevijacija, *JAD)*, programskih jezika 4. generacije, generatora koda itd, a kako bi se ubrzala provedba faza životnog ciklusa izvedenog primjenom minijaturnog vodopadnog pristupa razvoju u sklopu svake iteracije, razvijajući tako, na brzinu, funkcionalne pojedinačne dijelove sustava kako bi pri završetku svakog dijela on mogao biti prezentiran korisniku radi razmatranja i povratnih informacija. Ključni i osnovni zahtjevi definiraju se u prvoj iteraciji koja se implementira kako bi je korisnici mogli procjeniti da bi se ono što odgovara zahtjevima uključilo u sljedeću iteraciju, kao i naknadno utvrđeni zahtjevi, a ostalo isključilo iz daljnjeg razvoja. (Dennis, Wixom i Roth, 2011.) Glavni čimbenici o kojima ovisi uspješna primjena ove metodologije jesu ljudi, menadžment i alati. Stručan projektni tim s tehničkim znanjima i često sudjelovanje korisnika su imperativ uspješnog projekta. Također, projekt treba biti podržan dobrim menadžerskim vodstvom jer može biti vrlo složen poduhvat i može zahtjevati umješanost ljudi iz različitih područja. U konačnici, alati, koji omogućavaju automatizaciju i skraćivanje vrijeme trajanja izvedbe projekta moraju biti primjereni i efikasno upotrebljeni.

Osnovne faze brzog razvoja sustava mogu se prikazati Slikom 40.



Slika 40. Faze brzog razvoja aplikacija;
(Izvor: http://www.ftms.edu.my/images/Document/IMM006%20-%20RAPID%20APPLICATION%20DEVELOPMENT/Chapter%202nnote.pdf, datum pristupa: 10.06.2017.)

Faze razvoja, kako su prikazane slikom, jesu:

1. **Planiranje zahtjeva**

Faza uključuje jedan ili više sastanaka u obliku radionica između vlasnika, korisnika i ključnih članova razvojnog tima koji zajedno utvrđuju zahtjeve kroz strukturirani niz koraka utvrđujući pritom djelokrug i opseg projekta u pogledu obuhvaćenih funkcionalnosti, procjene troška i trajanja razvoja i implementacije.

Ciljevi ove faze su: ustanoviti razumijevanje poslovnih problema koji se tiču razvoja i funkcioniranja IS-a, upoznavanje sa sustavima koji su trenutno u funkciji i identificiranje poslovnih procesa koji će biti podržani predlaganim rješenjem.

2. **Korisnički dizajn**

Krajnji korisnici i/ili vlasnici sudjeluju, s članovima razvojnog tima, u sastancima/radionicama koje su dijelom prakse poznate pod nazivom “Joint Application Design“, tj. Grupni razvoj aplikacija, a o kojima će biti detaljnijeg objašnjenja u sljedećem poglavlju. Ove radionice imaju svrhu zajedničke provedbe aktivnosti koje se tiču faza analize i dizajna. Glavni zadaci ove faze jesu:

1) utvrditi detaljni model sustava i granica sustava na temelju JAD radionica, specificirajući poslovne aktivnosti čiju će izvedbu sustav podupirati i vrste podataka koje će obrađivati.

2) kreirati specifikacije dizajna sustava na temelju detaljnog modela, utvrđujući odnose između procedura i podataka, kreirajući pritom funkcije u obliku objekata koji se kao komponente mogu iskorištavati ponovno, nezavisno od aktualnog predloška rješenja.

3) utvrditi konzistentnost specifikacija analizom interakcija i testiranjem uz pomoć prototipa sve dok nije dobiven prototip koji nailazi na odobravanje projektnog tima i korisnika.

4) isplanirati postupak implementacije, obično koristeći metodu timebox-inga pri paralelnom razvoju inkremenata, pri čemu se utvrđuje trajanje, resursi i trošak svake pojedine definirane etape projekta (tzv. „*timebox-a*“).

5) rezimirati i razmotriti planirano JAD sastankom što rezultira odobravanjem, modifikacijom ili odustajanjem od daljnjih faza projekta.

3. Konstrukcija

Ciljevi izvedbe brzog razvoja dizajna jesu:

1) razviti i testirati softver kojim se implementira predloženo rješenje
2) generirati sustav koji karakteristikama udovoljava definiranim prihvatljivim kriterijima performansi
3) pripremiti dokumentaciju sustava uključujući uputstva o upotrebi
4) dizajnirati, razviti i testirati sofftver koji omogućava prijelaz sa starog na novi informacijski sustav ukoliko takav softver već ne postoji
5) pripremiti konverziju sustava iz radne u produkcijsku fazu.

4. Implementacija

Stavljanje sustava u upotrebu, konverzija podataka u baze sustava i provedba edukacije korisnika. (RAD lc development, n.d.)

Brzi razvoj aplikacija ima svoje prednosti i nedostatke a njih je moguće ukratko razložiti tablicom br. 2.

Tablica 2. Prednosti i nedostaci brzog razvoja aplikacija

PREDNOSTI	NEDOSTACI
✓ Prilagođavanje korisničkih zahtjeva u kasnijim fazama razvoja je manje zahtjevno	– Samo sustavi koji mogu biti razvijeni u modulima se mogu razvijati RAD metodologijom
✓ Napredak se lakše može pratiti i mjeriti	– Zahtjeva visoko stručan razvojni tim
✓ Vrijeme izvedbe iteracija može biti kraće uz pomoć upotrebe RAD alata	– Menadžment projektom može biti vrlo složen
✓ Omogućena ponovna iskoristivost komponenti, tj. objekata koji čine funkcije	– Zahtjeva čestu prisutnost i podršku korisnika

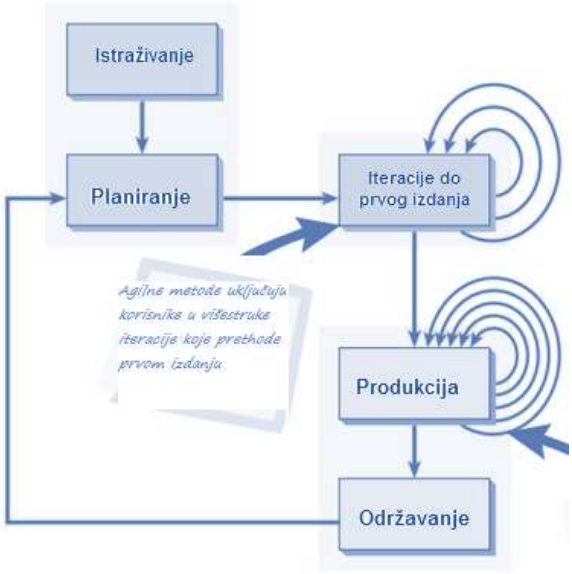
✓ Sudjelovanje korisnika u JAD radionicama poboljšava zadovoljstvo i povećava vjerojatnost uspjeha projekta	– Prikladniji za kraće projekte
✓ Integracija inkrementima rješava mnoge moguće probleme koji bi mogli nastati drugačijim pristupom.	
✓ Kao rezultat automatizacije generiranja koda, smanjuju se greške koje su mogle nastati ručnim pisanjem	

(Izvor: https://www.tutorialspoint.com/sdlc/pdf/sdlc_rad_model.pdf, datum pristupa: 10.06.2017.)

8. AGILNE METODE RAZVOJA SUSTAVA

8.1. Agilni pristup

Informacijski sustavi se u novije doba razvijaju brzim agilnim razvojem. Brzi razvoj omogućava pravovremenu reakciju poduzeća na promjene, ostvarivanje konkurentske prednosti, kao i iskorištavanje prililka koje iskrsnu kao rezultat razvoja novih tehnologija. To je utrka u kojoj su mnoge stranke naručitelji softvera spremni žrtvovati određene elemente dizajna i kvalitativne zahtjeve traženog rješenja u korist brze implementacije traženih funkcionalnosti. Na temelju takvog razvoja okolnosti, početkom 90ih se javlja novi pristup i nove metode razvoja IS-a u okvirima kojih se podrazumjeva primjena evolucijskog iterativno inkrementalnog pristupa pri kojem se sustav gradi u manjim inkrementima sa učestalim iteracijama – otprilike svaka dva ili tri tjedna. Korisnici kontinuirano pružaju povratne informacije i konsoliraju zahtjeve oko tih iteracija. Dokumentacija koja se pritom sastavlja je minimalnog obujma. (Sommerville, 2011.) Takav agilni pristp prikazan je slikom 41.



Slika 41. Agilni razvoj;
(Izvor: Kendall, K.E. i Kendall J. E., 2006.)

Različite metode koje primjenjuju ovaj pristup kolektivno se nazivaju „Agilnim metodama“ a u njih se ubrajaju, među ostalima, i:

- Kristalne metodologije
- Dinamička metoda razvoja sustava
- Scrum
- Funkcionalno usmjereni razvoj
- Ekstremno Programiranje

- Adaptivni razvoj sustava
- Lean Razvoj

U veljači, 2001. godine, 17 pobornika agilnih metoda sastaje se u saveznoj državi Utah u SAD-u te sklapa konsenzus o temeljnim principima tih metoda. Kao rezultat tog konsenzusa nastaje dokument pod nazivom „The Agile Manifesto“ koji predstavlja zbir tih principa – njih, ukupno, dvanaest. (Cadle, 2014.) Prema tom manifestu, principi agilnih metoda razvoja softvera jesu:

„1. Najviši prioritet je ostvarivanje zadovoljstva korisnika kroz rana i kontinuirana izdanja vrijednog softvera.

2. Otvorenost prema promjenjivim zahtjevima, čak i u kasnim fazama razvoja. Agilni procesi podržavaju promjene radi konkurentske prednosti korisnika.

3. Učestalo izdavanje funkcionalnog softvera, u rasponu od par tjedana do par mjeseci uz preferenciju što kraćih razmaka.

4. Suradnja korisnika i članova razvojnog tima/timova na dnevnoj bazi kroz trajanje razvoja.

5. Motivacija ljudi angažiranih na projektima razvoja. Osiguravanje primjerene okoline i potrebne potpore te iskazivanje povjerenje u njihovu sposobnog izvršenja projekta.

6. Najefikasnija i najefektivnija metoda prenošenja informacija prema i unutar razvojnog tima ili timova je neposredna osobna komunikacija.

7. Funkcionalni softver je primarna mjera napretka.

8. Agilni procesi podržavaju održivi razvoj. Sponzori, razvojni timovi i korisnici trebali bi moći održavati konstantan tempo neodređeno dugo vremena.

9. Kontinuirana pažnja posvećena tehničkoj izvrsnosti i dobrom dizajnu pospješuje agilnost.

10. Jednostavnost je ključna.

11. Najbolje arhitekture, zahtjevi i dizajni ostvaruju se kroz samo-organizirajuće timove.

12. Timovi razmatraju načine na koje mogu ostvariti veću djelotvornost u regularnim intervalima te zatim prilagođavaju svoje djelovanje u skladu sa zaključcima tih razmatranja.“ (Agilni Manifest, 2002.)

Taj je manifest doveo i do novih načina razmišljanja o razvoju softvera među strukom s obzirom na metode, procese i cikluse razvoja. Metode popout Scrum-a postale su vrlo popularne s obzirom na lakoću primjene. Za mnoge razvojne timove, manifest je poslužio kao izgovor za zaobilaženje formalnog dokumentiranja no, iako dokumentacija nije čimbenik kojeg se potrebno strogo pridržavati, strogo slijeđenje

reguliranih procesa jest iako neke metode ne definiraju sve faze razvoja ili tek postavljaju nužno slijeđeni skup smjernica kojima se udovoljava tijekom razvoja. Ovo omogućava i uklapanje nekih agilnih metoda s tradicionalnim metodologijama no agilne metode mogu biti neprimjerene ili manje primjerene u slučajevima u kojima:

„a) Veći timovi rade na projektu

b) Projekt podrazumijeva razvoj složenog sustava

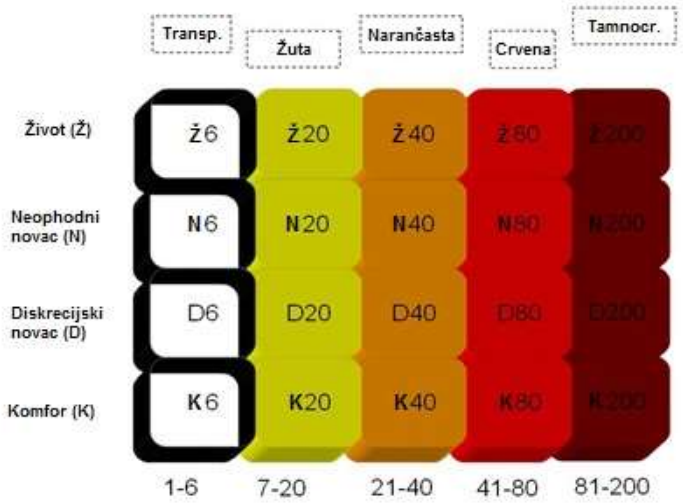
c) Projekt treba udovoljavati određenim zahtjevima kvalitete

d) Projekt je pod strogim ugovornim obvezama

e) Postoji složena korisnička okolina ili je krajnji korisnik nedostupan“ (Cadle, 2014.)

8.2. Kristalne metode

Kristalne metodologije su obitelj metodologija formalno definiranih od strane autora Alistaira Cockburna sredinom 90ih godina. Razlikuju s obzirom na veličinu projekta i kritičnost sustava koji se projektira, a s obzirom na te karakteristike, svaka metodologija u obitelji nosi naziv po odgovarajućoj boji kristala prikazanih Slikom . kojom su obuhvaćene Transparentna, Žuta, Narančasta, Crvena i Tamnocrvena kristalna metoda.



Slika 42. Kristalne metode;
(Izvor: Coffin i Lane, 2006.)

Veličina projekta: Boja kristala koja se koristi ovisi o broju ljudi koji sudjeluju na projektu. Kako broj uključenih sudionika raste tako postaju sve primjerenije metodologije sve tamnijih kristala koje u stupcima idu prema desnoj strani prikaza. Ako, dakle, projekt proširuje broj sudionika sa 20 na 40, tada imamo prijelaz sa Žute

na Narančastu kristalnu metodologiju. Što je verzija kristalnih metodologija tamnija tim podrazumijeva sve veću formaliziranost strukture i menadžmenta.

Kritičnost sustava: Kritičnost podrazumijeva potencijal sustava da uzrokuje štetu. Slova koja se nalaze u redovima ćelija na prikazu predstavljaju kritičnost sustava, a ona su:

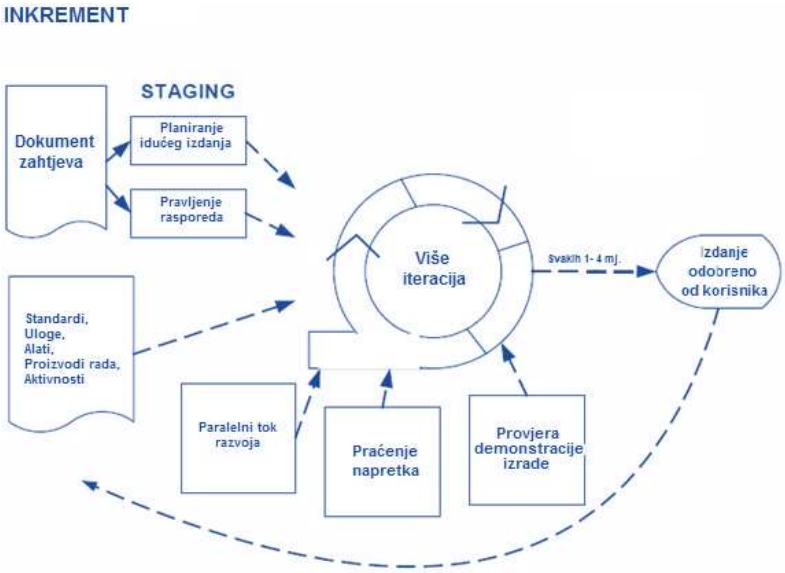
K= Komfor, D= Diskrecijski Novac, N= Neophodni Novac i Ž= Život

Prema tome, red u kojem se nalazimo, a koji je označen jednim od navedenih slova, indikator je potencijala štete koju zakazivanje može uzrokovati. Npr., ako zakaže sustav za održavanje života, uzrokovati će puno više štete nego video igra koja ne dopušta da se spremi napredak i koja će uzrokovati eventualno nedostatak komfora. Prema tom primjeru, potencijal da se dogodi ovo prvo bilo bi označeno slovom Ž i nalazilo bi se u prvom redu dok bi ovo drugo bilo označeno slovom K te bi se našlo u zadnjem redu. Što je veća kritičnost tim je veća potreba za rigoroznijim pristupom.

Neovisno o odabranoj kristalnoj metodologiji, svaka u svojoj osnovi sadrži sedam ključnih principa, a to su:

1. Učestalo izdavanje: korisnici trebaju moći očekivati nadogradnje svakih par mjeseci.
2. Kontinuirane povratne informacije: cijeli projektni tim se redovno sastaje međusobno i s korisnicima kako bi se raspravljalo o aktivnostima projekta i osiguralo da projekt napreduje u pravom smjeru.
3. Neprestana komunikacija: za male projekte se očekuje da cijeli tim bude smješten u istoj prostoriji dok je za veće projekte očekivano da svi članovi bar budu smješteni u istom objektu. Potrebno je održavati komunikaciju s osobom koja određuje zahtjeve.
4. Sigurnost: kristalne metode uzimaju u obzir različite svrhe u koje se sustav razvija i s obzirom na to se utvrđuje nužan fokus na sigurnost koji odražava kritičnost sustava u terminima potencijalne štete po korisnika.
5. Fokus: svi članovi tima trebaju znati bar prva dva prioriteta projekta na kojima svaki član radi te bi trebali moći raditi neometano.
6. Pristup korisniku: pristup jednom ili više korisnika kroz cijeli proces razvitka sustava.
7. Automatizirani testovi i integracija: verifikacija funkcionalnosti i česta integracija komponenti sustava. (Coffin i Lane, 2006.)

Kristalnim metodologijama je predviđen iterativno inkrementalni razvoj, kako je to ilustrirano na Slici 43.



Slika 43. Iterativno inkrementalni razvoj – Kristalne metode; (Izvor: Abrahamsson, Salo i Ronkainen, 2002.)

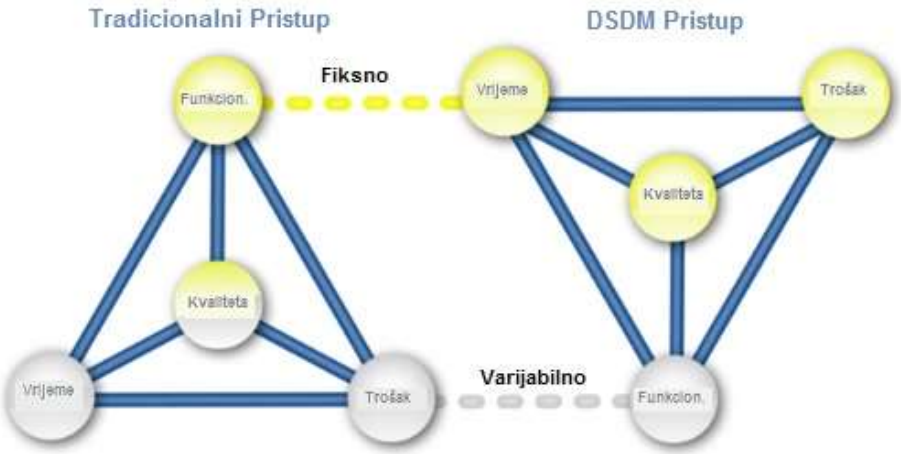
Aktivnosti koje se odvijaju u okvirima razvoja jednog inkrementa jesu: planiranje, pregled i revizija, nadzor, paralelni tok, strategija holističkog diverziteta, prilagodba metodologije, korisnički pregled i radionice retrospektive razvoja. (Abrahamsson, Salo i Ronkainen, 2002.)

8.3. Dinamička metoda razvoja sustava

„Dinamička metoda razvoja informacijskih sustava je agilna metodologija koja pruža okvire za brzi razvoj primjenom inkrementalnog prototipiranja u kontroliranoj projektnoj okolini. Filozofija iza ove metodologije je prilagođena verzija Paretova principa kojom se konstatira da se 80% sustava može razviti i implementirati u 20% vremena od onog koliko bi trebalo da se to učini za kompletno rješenje.“ (Pressman, 2010.)

Sustavi se razvijaju s težnjom za održavanjem balansa između konfliktnih zahtjeva projekta od kojih su najveći i najčešći vrijeme, trošak, funkcionalnosti i kvaliteta. To se najčešće pokazuje kao nerealistični pristup i, u suprotnosti s nastojanjima, često dovodi do više propusta nego što bi ih se alternativno izbjeglo. U tradicionalnim metodologijama je sadržaj funkcionalnosti fiksna varijabla dok su vrijeme i trošak varijabilni. Ukoliko se razvoj ne odvija po planu, projektu se dodijeljuje dodatno vrijeme i pripadajući trošak. Pokazalo se da dodavanje resursa već zakašnjelom projektu ima neželjeni učinak daljnjeg odgađanja realizacije i da ovakvi

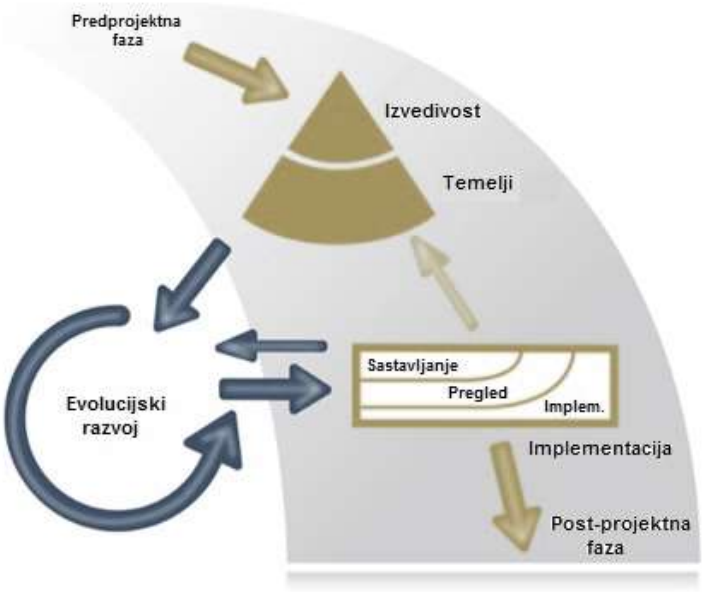
projekti često onda oduzimaju vrijeme testiranju zbog drugih faza što onda kvalitetu čini varijablom. Dinamička metodologija postavlja vrijeme, troškove i kvalitetu kao fiksne varijable, a funkcionalnosti, uglavnom one neprioritetne, čini varijabilnim. (DSDM Consortium 1)



Slika 44. DSDM - usporedba s tradicionalnim pristupom;
(Izvor: https://www.agilebusiness.org/content/philosophy-and-fundamentals , datum pristupa:14.05.2018.)

DSDM metodologija slijedi određene principe kojima se treba navoditi, a to su:

1. Fokus na poslovne zahtjeve
2. Zaključivanje zadataka, faza i rješenja na vrijeme
3. Timski rad i suradnja
4. Kvaliteta se ne smije kompromitirati
5. Inkrementalni razvoj uz prethodno dobro utvrđene zahtjeve
6. Iterativni razvoj
7. Održavanje kontinuirane i jasne komunikacije između svih stakeholdera
8. Demonstriranje kontrole kroz vidljivost planiranja i napretka i formaliziranog načina praćenja i izvještavanja napretka. (Izvor: DSDM Consortium 2)



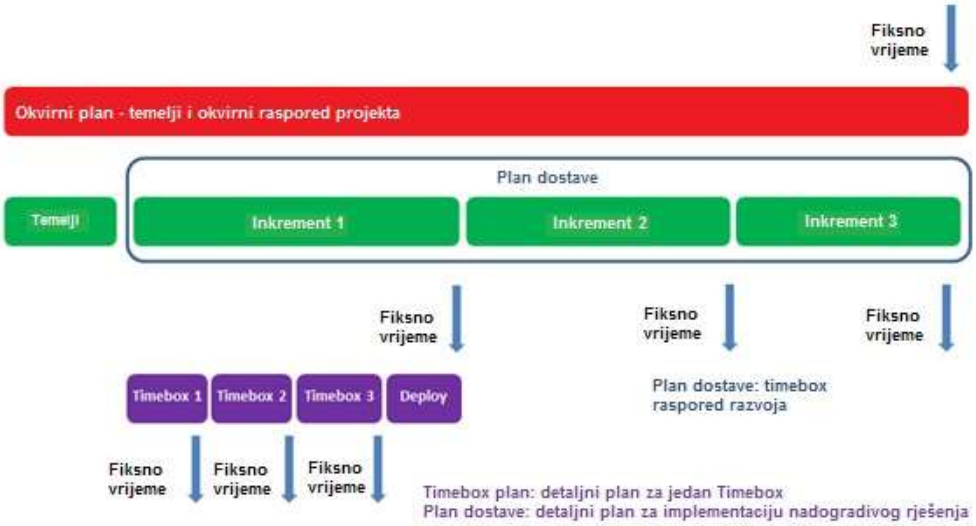
Slika 45. Dinamička metoda razvoja sustava;
(Izvor: https://www.agilebusiness.org/content/project-planning-and-control , datum pristupa:14.05.2018.)

Proces razvoja sustava po DSDM pristupu se odvija kroz sedam faza, a to su:

1. **Predprojektna faza** kojom se osigurava da se započinju pravi projekti, da su dobro usklađeni i da su im jasno definirani ciljevi.
2. **Faza analize izvedivosti** s tehničkog i ekonomskog gledišta.
3. **Faza analize zahtjeva** u kojoj se utvrđuju temeljni zahtjevi bez previše detalja. Ova faza može trajati do par tjedana čak i za velike i složene projekte, a njome se u apstraktnom smislu razlučuju poslovna svrha rješenja, potencijalno rješenje i način na koji će se odvijati razvoj i implementacija.
4. **Faza evolucijskog razvoja** u kojoj se primjenjuju prakse poput iterativnog razvoja, prototipiranja, timeboxing-a, MoSCoW prioretizacije i organiziranim radionicama kako bi se na vrijeme zaključio funkcionalno i tehnički primjeren dizajn. DSDM Consortium 3 (n.d.).

Timebox se može smatrati alatom projektnog menadžmenta, a njome se određuje i vizualizira distribucija vremena na zadatke koji se ispunjavaju kroz proces razvoja sustava. Timebox predstavlja vremenski interval čije trajanje optimalno iznosi i predviđa se oko dva tjedna unaprijed a najviše šest. Zatim se, po okončanju tog timeboxa dodaje drugi u priortiziranim inkrementima i tako dalje dok rješenje nije zaključeno. Osnovna ideja iza toga jest da je planiranje predviđanjem utroška vremena i zadataka najbolje raditi unutar nekog kraćeg zacrtanog perioda jer je predviđanje sve nesigurnije što dalje odmiče vrijeme. Timeboxevi obično nemaju fiksno trajanje, neki

se izvršavaju paralelno, neki mogu otpočeti prije nego što su drugi završili itd. DSDM Consortium 4 (n.d.).



Slika 46. Timeboxing;
(Izvor: Tech Academy, 2013.)

MoSCoW prioritizacija - **M**o**S**CoW je naziv koji je prilagođen i tvoren od samih naziva kategorija prioretizacije korisničkih zahtjeva koji mogu biti konstanta ali se mogu i mijenjati kroz trajanje razvoja sustava pa se u skladu s tim usklađuju sa navedenim kategorijama a to su:

- ***M**ust have* – primarni temeljni korisnički zahtjevi bez kojih rješenje ne može ni funkcionirati.
- ***S**hould have* – bitni zahtjevi koji značajno doprinose vrijednosti sustava ali se po potrebi mogu izostaviti u korist *Must have* zahtjeva.
- ***Co**uld have* – zahtjevi koji poboljšavaju sustav ali su odgodivi i mogu se izostaviti u korist zahtjeva prethodnih kategorija.
- ***W**ant to have* – zahtjevi koji su korisni užoj skupini korisnika i ne predstavljaju neku značajnu vrijednost za cjelokupno rješenje.

Organizirana radionica – u slučajevima kada postoji veliki broj stakeholdera koji trebaju sudjelovati u odlučivanju i rješavanju problema organiziraju se radionice kojima prisutstvuju najkvalificiraniji ljudi s obzirom na problematiku te su s obzirom na tematiku i podijeljene na:

1. Radionica o definiranju zahtjeva
2. Radionica o poslovnim koristima
3. Radionica o tehničkim operacijama sustava
4. Radionica planiranja testiranja prihvatljivosti rješenja

5. Radionica o dizajnu informacijskog sustava (Voight, 2004.)

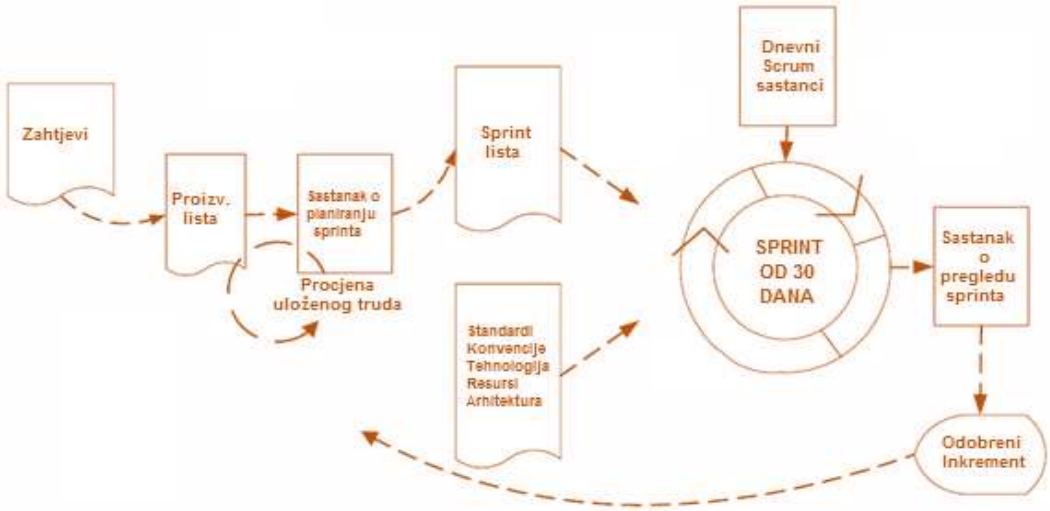
5**. Faza implementacije** – cilj ove faze je dovesti dizajn sustava u operativno stanje izvođenjem triju aktivnosti: integracije, pregleda i primjene. Integracijom „sklapaju“ svi dijelovi u cjelinu, pregledom se procjenjuje cjenlina kao rješenje te se odlučuje je li zadovoljavajuće, a primjenom se rješenje stavlja u uporabu tehničkom implementacijom u informacijskoj okolini.

6. **Postprojektna faza** – procjena ostvarenih poslovnih koristi od primjene sustava s obzirom na ciljane. (Izvor: DSDM Consortium 3)

8.4. Scrum

Kada je Jeff Sutherland iznio i formalno utvrdio Scrum kao metodologiju životnog ciklusa 1993. godine, taj je naziv odabrao zbog analogije sa studijom autora Takeuchija i Nonake iz 1986. godine pod nazivom „The New New Product Development Game“ (Takeuchi i Nonaka, 1986.) objavljene u Harvard Business Review-u, a u kojoj je napravljena usporedba visoko učinkovitih međufunkcijskih timova sa „scrum“ formacijom korištenom u ragbi momčadima kako bi se lopta koja je izišla iz terena vratila u igru. (Sutherland, 2011.)

Scrum je agilna metodologija životnog ciklusa razvoja u kojoj projekti napreduju kroz seriju iteracija koje se nazivaju „sprintevima“ od kojih svaki obično traje dva do četiri tjedna. Scrum je prikladan za projekte koji se moraju moći brzo mijenjati. Originalno je formaliziran za projekte razvoja sustava ali zapravo je primjenjiv za bilo koji kompleksni, inovativni projekt. Razvoj je prikazan slikom



Slika 47. Scrum;
(Izvor: Abrahamsson, Salo i Ronkainen, 2002.)

Scrum se, ukratko, odvija u sljedećim fazama:

- S korisnikom se utvrđuju zahtjevi na temelju kojih se kreira prioritetizacijska lista pod nazivom „product backlog“ ili proizvodna lista koja predstavlja svojevrsni spisak narudžbi u obliku svojstava i funkcionalnosti koje bi trebalo dodati završnom proizvodu.
- Za svaku funkciju koja se dodaje rješenju se vrši procjena potrebnog uloženog truda.
- Održava se sastanak na kojem se planira sprint tako da se uzme dio funkcionalnosti s vrha proizvodne liste. Taj dio koji će se uklopiti u razvoj u jednom sprintu naziva se „sprint backlog“ ili sprint lista. Zatim se odlučuje kako će se taj dio uklopiti.
- Tim ima određeni raspon vremena za sprint – obično dva do četiri tjedna – da dovrši iteraciju ali se sastaje svaki dan kako bi se izvršila procjena napretka. Takav dnevni sastanak se naziva „dnevni scrum“.
- Na kraju sprinta, iteracija bi trebala biti spremna za izdavanje korisniku na uvid.
- Sprint završava sastankom na kojem se vrši pregled i procjena izvedenog sprinta.
- Započinjanjem novog sprinta, tim uzima novu skupinu narudžbi s vrha proizvodne liste i započinje novi sprint.

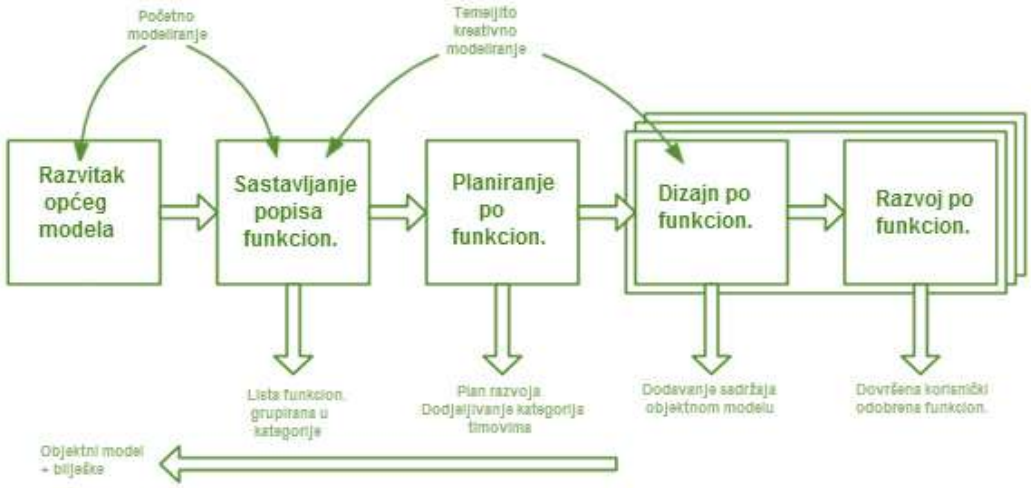
Ciklus se ponavlja dok se sve nužne narudžbe s liste ne implementiraju rješenjem ili dok ne ponestane budžeta i/ili vremena. Scrum-om se osigurava da do kraja projekta one najbitnije funkcionalnosti budu uključene u zadnju verziju sustava. (Izvor: Scrum Alliance)

8.5. Razvoj temeljen na funkcionalnostima

Razvoj temeljen na funkcionalnostima (orig. Feature driven development ili skraćeno FDD) – predstavlja razvoj temeljen na funkcionalnostima, to jest elementima koji čine korisne funkcije koje se mogu implementirati u sustav, predstavlja iterativnu i inkrementalnu agilnu metodologiju razvoja osmišljenu od strane autora Petera Coada i Jeffa De Luce 1999. godine kao predložak za praktično rješenje procesnog modela objektno-orijentiranog razvoja. Taj je predložak nadograđivan kako bi poprimio svoj sadašnji oblik adaptivnog procesa razvoja koji se može primijeniti i na srednje velike projekte i projekte razvoja kritičnih sustava za specifične domene. Posebna pažnja posvećuje se osiguranju kvalitete što je poduprto aktivnostima kao što su provjera

dizajna, testiranje koda, dokumentiranje načina na koji se udovoljava kvalitativnim standardima, korištenjem metrika i obrazaca itd.

FDD se temelji na ideologiji čije su glavne odrednice poticanje komunikacije i suradnje u timovima, rješavanje složenih problema u razvoju primjenom dekompozicije funkcionalnih elemenata i njihovom inkrementalnom integracijom te interpretiranje i predstavljanje tehničkih detalja koristeći verbalne, grafičke i tekstualne metode. (Pressman, 2010.)



Slika 48. Razvoj temeljen na funkcionalnostima;
(Izvor: Ambler, n.d.)

FDD razvoj se odvija u pet faza s time da za početne tri faze možemo reći da predstavljaju tzv. nultu iteraciju. Nulta iteracija započinje s idejom rješenja koje treba razviti ili nadograditi, a za idejom nastupa vizija na temelju koje se onda utvrđuju bitni temelji za dizajn. FDD ne predviđa zasebne faze planiranja i analize u klasičnom smislu pa se temelji za dizajn ostvaruju kroz takav kratak period timskih aktivnosti kao što su: istraživanje ideja, utvrđivanje potreba korisnika, razvojnih praksi i arhitekture. (Grenning, 2012.)

Ove aktivnosti se izvode sa svrhom utvrđivanja potrebnih funkcionalnosti i truda koji će biti uložen, a pritom u timu moraju biti prisutni svi ključni stakeholderi, korisnici, stručnjaci područja za koje se razvija sustav i tehnički i analitički kadar.

Tri faze koje čine okvire ovih aktivnosti jesu:

- ***Kreiranje okvirnog modela*** – naručitelj rješenja i razvojni tim surađuju na kreiranju cjelokupnog modela područja koje će biti obuhvaćeno sustavom. Naručitelj izlaže kontekstualni prikaz sustava i način izvođenja poslovnih procesa, a razvojni tim razvija objektne modele različitih stupnjeva

dekompozicije na temelju tih informacija, da bi se, u konačnici, usuglasili oko jedinstvenog modela koji predstavlja najoptimalnije rješenje.

- ***Sastavljanje popisa funkcionalnosti*** – tim sastavlja prioretiziranu listu funkcionalnosti u suglasnosti s naručiteljem. Dekompozicijom je potrebno postići dovoljno male funkcionalne elemente da svaki bude moguće implementirati unutar dva tjedna. Funkcionalnosti se smisleno grupiraju u skupine koje odražavaju ovisnosti i asocijacije između funkcija.
- ***Planiranje na temelju funkcionalnosti*** – razvojni tim se organizira u manje skupine i planira se implementacija na temelju ustanovljenog popisa funkcionalnosti na način da svaki tim dobije jednu skupinu. Funkcionalnosti se raspoređuju tako da odražavaju redoslijed razvoja i sortiraju pomoću „milestone“ ili „timebox“ metoda (metode prikazivanja vremenskog rasporeda i redoslijeda izvođenja aktivnosti i procesa).
- ***Dizajn i razvoj po funkcionalnostima*** – ove poslijednje dvije faze razvoja zauzimaju oko 75% ukupnog vremena. Izvode se u iteracijama. Identificiraju se asocijacije klasa između funkcionalnosti. Grupirani timovi rade zajedno na dijagramima klasa i sekvenci i općem dizajnu funkcionalnosti. U zadnjoj fazi se usvaja strukturni pristup i nastavlja s kodiranjem, testiranjem, integracijom i procjenom iteracija koje se pri završetku tada dodaju osnovnom rješenju, a njihovi timovi uzimaju nove skupine funkcionalnosti pa ponovno započinju četvrtu fazu itd. (Palmer, 2009.)

8.6. Ekstremno Programiranje

Ekstremno programiranje je metodologija čiji je razvitak proizašao iz problematike dugotrajnih razvojnih ciklusa tradicionalnih metodologija. U početku je promatrana samo kao prilika da se projekti zgotove uz pomoć praksi koje su se prethodno pokazale uspješnima, no nakon uspjeha koji u praksi postiže, napokon se uspostavlja kao službena metodologija 1999. godine u djelu pod nazivom „Extreme Programming Explained“ autora Kenta Becka koji je u to vrijeme radio na sustavu za Chrysler. (Beck, 1999) Naziv „ekstremno“ odnosi se na to da se najbolje prakse primjenjuju do ekstrema.

Iako prakse koje su korištene u ekstremnom programiranje same zasebno nisu bile nove, ipak su bile organizirane i usklađene tako da su funkcionirale na nov i učinkovit način. Prakse koje se primjenjuju u Ekstremnom programiranju su:

Igra planiranja – osobna i učestala interakcija između korisnika i razvojnog tima. U početku je predlagano da bi razvojni tim trebao biti mali ili srednji – u sastavu od tri do dvadeset članova, no kasnije je to ostavljeno na proizvoljnu procjenu. Razvojni tim procjenjuje trud koji je potrebno uložiti u dodavanje funkcija sustavu, a korisnik zatim određuje obujam i vremenski raspored izdanja iteracija.

Mala/kratka izdanja – jednostavnije verzije sustava prolaze brzu produkciju bar jednom u dva mjeseca. Nove verzije se zatim izdaju u vremenu od jednog dana do jednog mjeseca.

Metafora – sustav je definiran metaforičkim pričama korisnika, opisujući skupom metafora kako sustav funkcionira, na čemu se zasniva cijeli razvoj.

Jednostavni dizajn – naglasak je na razvoju najjednostavnijeg mogućeg rješenja koje moguće implementirati. Nepotrebne kompleksnosti i viškovi koda se uklanjaju.

Testiranje – razvoj softvera je s fokusom na testiranje. Testiranje rješenja se implementira prije koda i izvodi se kontinuirano. Korisnici kreiraju funkcionalne testove.

Refaktoriranje – restrukturiranje sustava uklanjanjem redundancija, poboljšavanjem komunikacije, pojednostavljivanjem i fleksibilnošću.

Programiranje u paru – dva programera pišu kod za jednim računalom.

Kolektivno vlasništvo – bilo tko može mijenjati bilo koji dio koda bilo kada.

Kontinuirana integracija – novi dio koda se integrira u osnovni kod čim je gotov. Tako se sustav nadograđuje više puta dnevno. Da bi kod bio prihvaćen mora proći testove.

Radni tjedan od 40 sati – maksimalna tjedna satnica iznosi 40 sati.

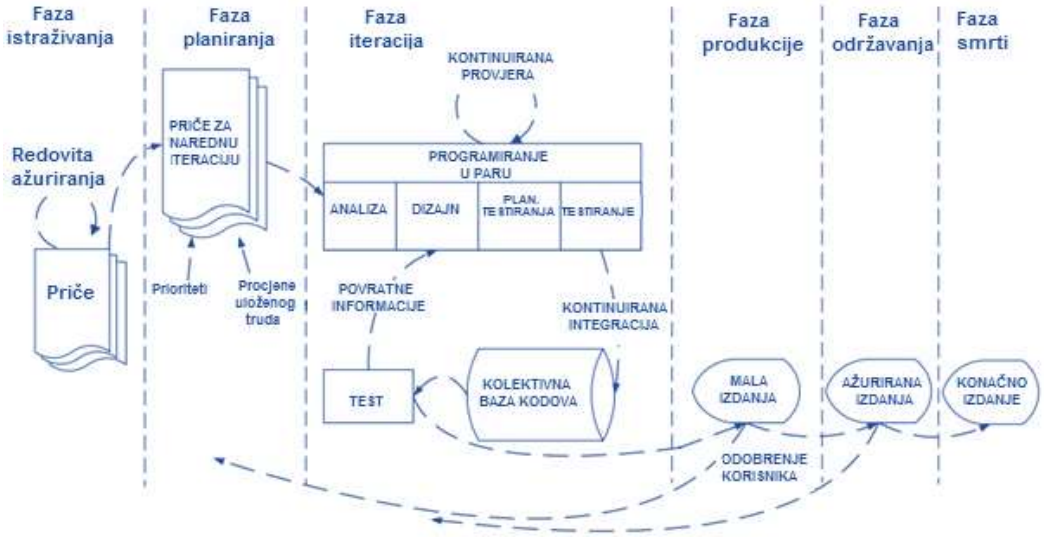
Prisutni korisnik – korisnik treba uvijek biti dostupan.

Standardi kodiranja – standardi kodiranja su utvrđeni i primjenjivani.

Otvorena radna okolina – preferira se velika radna prostorija s više otvorenih odjeljaka s programerima u paru u centru.

Pravedna pravila – tim ima vlastita pravila koja slijedi, a koja se mogu mijenjati bilo kada ali se s njima moraju svi složiti i treba procjeniti učinak njihove primjene. (Beck, 1999)

Životni ciklus XP-a sastoji se od pet faza, a to su: istraživanje, planiranje, iteracije za izdanja, produkcija, održavanje i smrt.



Slika 49. Ekstremno programiranje;
(Izvor: Abrahamsson, Salo i Ronkainen, 2002.)

Istraživanje – u početnoj fazi korisnici ispisuju kartice s „pričama“ za koje žele da budu uključene u prvom izdanju. Svaka kartica predstavlja mogućnost koju program treba sadržavati. Istovremeno, projektni tim se upoznaje s alatima, tehnologijama i praksama koje će primjeniti na projektu i gradi prototip kako bi se razmotrila primjena mogućih arhitektura. Ova faza može trajati od par tjedana do par mjeseci ovisno o tome koliko su programeri upoznati sa primjenjenom tehnologijom.

Planiranje – podrazumijeva utvrđivanje prioritetizacijskog slijeda dodavanja korisničkih kartica, to jest mogućnosti prvoj iteraciji, a zatim i određivanje svih mogućnosti koje će biti dodane. Pritom se procjenjuje koliko će truda zahtjevati dodavanje svake mogućnosti i sastavlja vremenski raspored razvoja iteracije koji ne bi smio trajati dulje od dva mjeseca. Faza planiranja obično traje par dana.

Iteracije za izdanje – podrazumijeva razvoj određenog broja iteracija sustava prije prvog službenog izdanja. Vremenski raspored koji je utvrđen u fazi planiranja se razbija na neki broj iteracija tako da za svaku bude potrebno do četiri tjedna za implementaciju. Prva iteracija koja se razvije sadržava arhitekturu cijelog sustava, što se postiže odabirom „priča“ na kojima se temelji cijela struktura sustava. Svaka iteracija pri svom završetku prolazi funkcionalno testiranje od strane korisnika. Finaliziranjem posljednje iteracije, sustav je spreman za implementaciju.

Produkcija – dodatna testiranja i provjera performansi sustava prije nego što se izda korisniku. U ovoj fazi se još uvijek se mogu utvrditi poželjne promjene i odlučiti hoće li ih se dodavati. Iteracije se ubrzavaju na razvoj od oko tjedan dana. Odgođene

ideje i prijedlozi se dokumentiraju za moguću kasniju implementaciju tijekom održavanja.

Održavanje – nakon što prvo izdanje biva implementirano i druge se iteracije razvijaju dok sustav radi, razvoj novih iteracija se obično odužuje te je moguće da nastane potreba za angažiranjem novih članova razvojnog tima.

Smrt – faza koja nastupa kad korisnik više nema „priča“ za dodati. Time se podrazumijeva da sustav u potpunosti zadovoljava korisničke potrebe u svim aspektima. Upotpunjuje se i zaključuje dokumentacija sustava. XP je najdokumentiranija od svih agilnih metodologija. Ova faza može nastati i ako sustav ne udovoljava zahtjevima ili ako projekt postane preskup za daljnji razvoj. (Abrahamsson, Salo i Ronkainen, 2002.)

Preporuka je da se XP uvodi postepeno ali jednom kada se uvede može polučiti veliki uspjeh pogotovo zato što se zbog svoje fleksibilnosti i evolucijske naravi može dobro nositi s promjenama.

8.7. Adaptivni razvoj sustava

Adaptivni razvoj softvera razvijen je na temelju praksi brzog razvoja, to jest RAD-a. Počiva na principu sposobnosti prilagodbe promjenama, kao što se iz naziva dade naslutiti. U tom kontekstu naglašava se važnost fleksibilnosti, sposobnosti suradnje ljudi u timovima i njihovom kontinuiranom učenju u promjenjivim uvjetima, što se odražava na njihovim projektima. Radi se o iterativnom agilnom procesu razvoja u kratkim intervalima. Taj se proces prikazuje kao dinamički ciklus u tri faze koje nisu nužno sekvencionalne već se mogu preklapati, a to su predviđanje, suradnja i učenje. Te faze obuhvaćaju neke prakse i s njima čine ciklus adaptivnog razvoja softvera, kao što je prikazano na slici



Slika 50. Adaptivni razvoj sustava;
(Izvor: Abrahamsson, Salo i Ronkainen, 2002.)

Ovaj ciklus primjenjuje šest osnovnih principa, a to su: usredotočenost na misiju, usredotočenost na razvoj funkcionalnosti, iteriranje, timeboxing, upravljanje rizikom i otvorenost prema promjenama.

Predviđanje u ovom ciklusu zamjenjuje uobičajenu fazu planiranja konceptualnim odmakom od strogo utvrđenog plana s jasnim sveobuhvatnim krajnjim rezultatom. Klasična ideja planiranja smatra se ograničavajućom jer se ono oslanja na unaprijed poznate načine ostvarivanja rezultata što ne ostavlja mjesta inovacijama i učenju. Ako se rezultati kontinuirano postižu na jednak način tada se teško mogu postizati brže i bolje već samo stangnantno, a stagnacija se ne podudara s idejom prilagodljivosti. Predviđanjem je bitno utvrditi kritične elemente projekta, rizike, složene probleme, a zatim poduprijeti njihovo rješavanje idejama i istraživanjima, pa i metodom pokušaja i pogreške u okvirima eksperimentiranja kako bi se utvrdilo što funkcionira, a što ne. Faza predviđanja obuhvaća ***inicijaciju*** projekta i ***adaptivno planiranje***. Začetkom se, kroz JAD radionicu, utvrđuju misija, ciljevi, ograničenja, resursi, osnovni zahtjevi i veličina projekta te povezani rizici. Adaptivnim planiranjem, također izvedenim kroz JAD radionice, predviđa se izvođenje projekta timebox metodom, to jest dodjeljivanjem vremenskih intervala skupinama aktivnosti kojima se postižu pojedinačne funkcionalnosti. Ti intervali se mogu izvoditi paralelno između iteracija, a određuju se na temelju informacija utvrđenih ***inicijacijom***. U ovoj se fazi također predviđa broj iteracija kao i s njima povezane funkcionalnosti prioretizacijom od strane naručitelja.

Suradnja, kao sljedeća faza ciklusa, podrazumijeva sposobnost efikasnog timskog rada pojedinaca koji međusobno dijele svoja znanja i vještine i skladno djeluju u svrhu ostvarivanja misije i ciljeva projekta. Suradnja je vrlo bitna za ***paralelan razvoj funkcionalnosti*** jer se radi o složenom procesu koji zahtjeva dobar menadžment. Pritom je bitno reći da suradnja mora postojati ne samo među članovima razvojnog tima već i među svim ostalim ljudima uključenim u projekt: naručiteljima, korisnicima, konzultantima itd. Bez suradnje nije moguće efikasno donositi odluke, ispunjavati zadatke, rješavati probleme, itd.

Učenje je svojevrsni input - preduvjet primjene informacija i znanja koja se unose u razvoj projekta, ali i svojevrsni output - produkt procesa razvoja kroz koji proizlaze neke nove spoznaje. Baš zato, u ciklusu adaptivnog razvoja softvera učenje je predstavljeno kao petlja koja se odvija kroz iteracije unutar projekta i kroz različite projekte. Ona tvori intelektualni i kreativni aspekt prilagodljivog razvoja. Učenje se

odvija kroz mehanizme ***kontrole kvalitete*** te ***zaključnih intervjua i izdanja***. Tu su obuhvaćene prakse poput tehničkih i korisničkih testiranja, sastavljanja pregleda rezultata i izvješća i rezimiranja povratnih informacija, a njima se stječe uvid u:

a) kvalitetu rezultata razvoja iz perspektive korisnika
b) kvalitetu rezultata razvoja s tehničkog stajališta
c) funkcioniranje razvojnog tima i primjenjivanih praksi
d) status projekta/napredak (Highsmith, 2002.)

8.8. Lean Razvoj

Pojam „Lean“ se prvi puta primjenjuje u nazivu u kontekstu opisivanja pristupa proizvodnji 1991. godine kada japanska Toyota bilježi uspjeh u automobilskoj industriji. Tada skupina autora - Womack, Jones i Roos - u svojoj knjizi pod naslovom „The Machine That Changed the World: The Story of Lean Production“ nazivaju Toyotin pristup proizvodnji, u kojem se vodi za skupinom vrijednosti i principa, „Lean“ načinom proizvodnje. U kontekstu razvoja softvera se prvo koristi u nazivu konferencije ESPRIT-a - Europskog strateškog programa za istraživanje i razvoj informacijske tehnologije organizirane 1992. godine u Njemačkoj.

Najpribližnijem prijevodu naziva „lean“ bi možda došli kada bi upotrijebili riječ „racionalno“ za opisivanje ovog pristupa ali ćemo se u daljnjem tekstu ipak koristiti anglističkom verzijom. Lean razvoj spada u kategoriju agilnih metodologija razvoja, a fokus stavlja na ostvarivanje vrijednosti za korisnika uklanjanjem viškova i „otpada“ u korist praktičnosti, brzine, smanjenja troškova i nepotrebnih aktivnosti, resursa i funkcionalnosti.

Lean pristup razvoju ne propisuje faze i ne definira korake kojima se razvija sustav no u većini slučajeva se koriste određene prakse koje omogućuju primjenu tog pristupa pridržavajući se propisanih principa na kojima se temelji i koji čine njegovu suštinu. Dakle, može se reći da se pri razvoju nekog sustava primjenjuje Lean pristup ako se pritom pridržava:

1. **Eliminacije „otpada“** – ovo je temeljni princip lean pristupa a po njemu se „otpad“ određuje kao sve ono što ne dodaje na vrijednosti sustava iz perspektive korisnika. U otpad spada sve što možemo svrstati u tri kategorije otpada koje se nazivaju odgovarajućim terminima japanskog jezika, a to su: Muda, Mura i Muri ili, skraćeno, 3M, a znače:

- ***Muda*** – sve aktivnosti čijim se izvođenjem ne dodaje na vrijednosti sustava u razvoju.
- ***Mura*** – sve varijabilnosti ili zastranjivanja zbog kojih razvoj ne ide po planu i ne dodaje na vrijednosti sustava u razvoju
- ***Muri*** – preopterećivanje ljudi i resursa, nerazumna očekivanja koja uzrokuju probleme zbog kojih se smanjuje vrijednost sustava

U razvoju informacijskih sustava se javlja sedam najčešćih oblika otpada ili viškova, a to su:

- djelomično obavljeni zadaci,
- procesi u razvoju koji su višak,
- funkcionalnosti koje su višak,
- prebacivanje sa zadatka na zadatak,
- čekanje i odgađanje
- nepotrebne aktivnosti koje ne spadaju u proces razvoja već su nusprodukt načina na koji se procesi razvoja izvode,
- mane, greške i defekti.

Jedan od načina na koji se može ukloniti otpad iz razvoja jest primjena mapiranja toka stvaranja vrijednosti. Ova tehnika se primjenjuje tako da se prati i bilježi način na koji su grupirane aktivnosti razvoja i kako se izvršavaju od početka do kraja procesa razvoja, a zatim se taj tok analizira kako bi se utvrdilo gdje nastaju 3M viškovi.

2. Potenciranja učenja – područje razvoja informacijskih sustava ima potrebu za kadrom radnika znanja pogotovo zato što je proces razvoja kreativan analitički proces koji zahtjeva primjenu znanja ukoliko će rezultirati uspjehom. „Učeći“ kadar će se najbolje potencirati poboljšanja u ostvarivanju vrijednosti za korisnika pa je stoga potrebno potencirati učenje. Jedan od načina potenciranja učenja je poticanje kroz feedback i iteracije kojima je moguće ustanoviti najbolje prakse, sinkronizacijom razvojnih aktivnosti na dijelovima rješenja koja čine cjelinu kako bi se stekla sustavna perspektiva, razmatranjem mogućnosti unutar ograničenja umjesto nasumičnog isprobavanja kako bi se brzo razjasnila moguća rješenja i mogući utvrdili obrasci djelovanja pri danim ograničenjima ubuduće.

3. Najkasnijeg mogućeg odlučivanja – odlučivanje u kontekstu razvoja sustava može imati dalekosežne posljedice – i dobre i loše. Odluka predstavlja opredjeljivanje za opciju. Jedna opcija o kojoj je odlučeno se odražava u budućnosti

ograničavanjem budućeg izbora opcija i na prošlost prilagodbom rezultata ostvarenih na temelju dotadašnjih odluka, što zahtjeva vrijeme i stvara trošak. Odgađanjem odlučivanja do najkasnijeg mogućeg trenutka bez da nastane šteta je korisno zato što su je u najkasnijem mogućem trenutku vidljiv najveći broj mogućih opcija i najviše infomacija o trenutnoj situaciji u kojoj se odlučuje tako da možemo reći da je to osiguravanje informiranog odlučivanja.

4. Najbržeg mogućeg stavljanja u upotrebu – ovdje su korisne metodologije RAD i DSDM. Primjena iteractivnog i inkrementalnog razvoja i prototipiranja.

5. Opunomoćenja razvojnog tima – ovaj princip počiva na ideji da nitko ne poznaje neki posao bolje od onih koji ga obavljaju i da su zato to osobe koje su najkvalificiranije za donošenje odluka, iznošenja ideja i pružanja smislenog uvida u neku problematiku ukoliko za to postoje određeni uvjeti koje je potrebno osigurati. Tom kadru je potrebno pružiti dužno poštovanje i omogućiti doprinos ostvarivanju vrijednosti rješenja primjenom vodstva, motivacije i stručnosti kao i principom br. 2 – potenciranjem učenja.

6. Osiguranja integriteta – sustav koji se percipira kao rješenje s integritetom jst sustav koji usklađeno, konzistentno i brzo reagira kao cjelina, dizajniran je racionalno, sa smislenom arhitekturom, lako je održiv, prilagođavan, nadograđivan te zadržava svoju korisnost s prolaskom vremena. Ideja je pristup razvoju kojim će se ostvariti ove karakteristike.

7. Sustavne perspektive – u procesu razvoja svatko je zadužen za svoj specifični zadatak ili rješenje za koji je kvalificiran i ima svoju perspektivu o tome što podrazumijeva optimalni rezultat tog izvršenja u vidu ostvarenja različitih performansi i svojstava. To znači da se nad inkrementima primjenjuje individualna optimizacija ali treba uzeti u obzir da optimalno funkcioniranje djelova ne znači optimalno funkcioniranje cjeline. Zato svatko tko radi na određenom dijelu ukupnog rješenja mora imati sustavnu perspektivu i biti upoznat s načinom na koji će se rješenje na kojem radi uklapati s ostalim rješenjima kako bi konačni integrirani sustav funkcionirao optimalno kao cjelina. **(**Poppendieck i Poppendieck, 2003.)

Tijekom „lean“ razvoja uobičajeno se primjenjuju neke individualno korištene ili kombinirane prakse. Te prakse jesu:

- ***kumulativni dijagrami toka i vizualne kontrole*** – koriste se za grafičku vizualizaciju rada koji je u tijeku izvedbe u svakom trenutku u životnom ciklusu razvoja. Cilj vizualizacije rada je omogućiti pravovremeno uočavanje „uskih grla“

i odstupanja kod kojih ne dolazi do akcija koje stvaraju vrijednost. Ove prakse također olakšavaju koordinaciju i suradnju timova jasno delegiranim zadacima i rasporedom posla.

- ***virtualni kanban sustavi*** – u ovoj se praksi primjenjuje kartični sustav u kojemu se karticama označava dostupne zadatke na kojima se može početi raditi, ograničavajući tako količinu posla u izvedbi. Često se kombinira sa vizualnim kontrolama.
- ***iterativno-inkrementalni razvoj***
- ***automatizacija*** – svi zadaci koji se mogu izvršiti automatizacijom su zadaci na kojima se efikasno štedi vrijeme i rad.
- ***„kaizen“ akcije*** – održavanje sastanaka i radionica razvojnih timova ima za cilj pronalaženje suvišnih, neefikasnih radnji u procesu razvoja kako bi se, unošenjem odgovarajućih promjena, postizalo kontinuirano poboljšavanje procesa. Unošenje takvih promjena naziva se „kaizen“ akcijama.
- ***retrospektive*** – sastanci na kojima se okončanjem dijela razvojnog procesa rezimiraju dojmovi i iznose opažanja i mišljenja o svim aspektima procesa u nastojanju iznalaženja potrebnih kaizen „akcija“ (Anderson, 2013.)

9. METODE ZA RAZVOJ IS-a TEMELJENIH NA WEB-u

9.1. Ciklus razvoja IS-a temeljenih na web-u

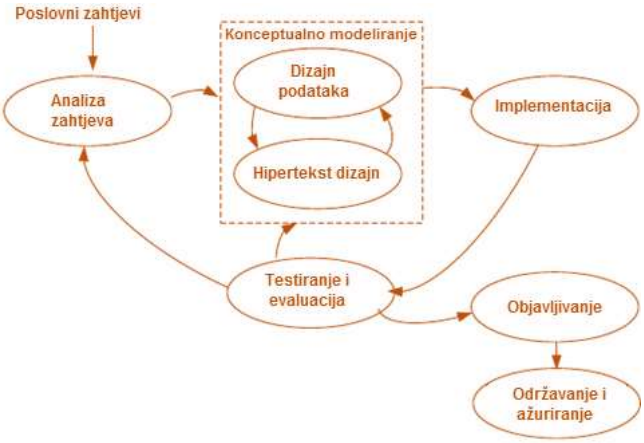
Razvoj mreža, posebice interneta i world wide web-a, potaknuo je informacijsko komunikacijsku revoluciju. Informacije u svim oblicima postale su istovremeno dostupne širom svijeta. To je, dakako, predstavljalo prekretnicu i u načinu poslovanja jer je ekonomija zadobila jednu sasvim novu dimenziju kako je nastao novi efikasni kanal komunikacije i distribucije za trgovinski i uslužni sektor i s njima povezane aktivnosti. Poslovni informacijski sustavi danas su sposobni vršiti komunikaciju sa lokalnim ali i globalnim mrežama te biti bazirani na njima u obliku ekskluzivne usluge ili usluge u „oblaku“.

Internetom su podržani različiti sadržajni oblici informacija, a oni se mogu svrstati u tri kategorije. To su:

- hipertekst – tekst koji je preko hiperpoveznica povezan sa drugim sadržajno relevantnim tekstovima u mrežu kroz koju čitatelj navigira klikom pokazivača miša na hiperpoveznicu ili tzv. link, prebacujući se tako s jednog teksta na drugi.
- multimedija – interaktivni prikaz koji se sastoji od kombinacije različitih vrsta medija: slike, teksta, videa i audio zapisa.
- hipermedija – kombinacija hiperteksta i multimedije u kojoj su multimedijski sadržaji međusobno umreženi hiperpoveznicama preko kojih se korisnik prebacuje sa jednog sadržaja na drugi. (Costagliola, Ferrucci i Francese, 2002.)

Prijenos i preuzimanje navedenih oblika informacija između jedinstvenih adresa na mreži (ili tzv. URL-ova - „Uniform Resource Locator-a“) podržan je protokolima kao što su HTTP („Hipertekst Transfer Protocol“) ili FTP („File Transfer Protocol“), a kreiranje i formatiranje informacijskih sadržaja tako da budu vidljivi na proizvoljne načine na jedinstvenim adresama podržano je programskim jezicima. HTML („Hypertext Markup Language“) i XML („Extensible Markup Language“) su dva standardna jezika od kojih HTML služi kako bismo odredili izgled i poziciju informacijskih sadržaja, a XML služi kako bismo odredili o kojem se sadržaju radi. (Oz, 2008.)

Navedeni sadržaji i protokoli omogućavaju razvoj i dizajn umreženih informacijskih sustava. Klasični životni ciklus razvoja tih sustava ili WDLC (orig. „Web-based Development Life Cycle“) može se jednostavno prikazati ISO 12207 metodologijom životnog ciklusa.



Slika 51. Razvoj IS-a temeljenih na web-u;
(Izvor: http://www.csun.edu/~twang/595WEB/Slides/WebML.pdf, datum pristupa: 03.06.2018.)

Faze ove metodologije konceptualno prate učestale faze raznih drugih metodologija, s iznimkom faze objave, iako se s obzirom na primjenu neke razlikuju u izvedbi, a to su:

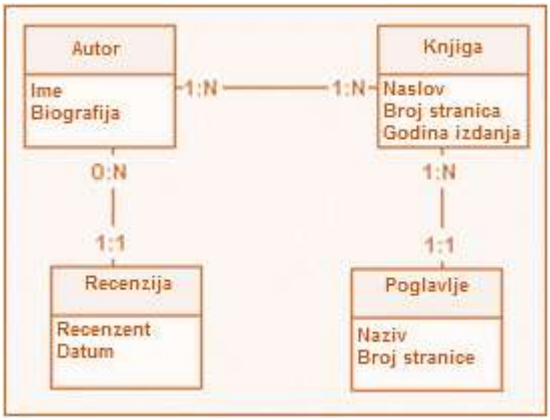
- Planiranje – podrazumijeva komuniciranje dojmova i potreba korisnika, prikupljanje korisničke dokumentacije i uspostavljanja okvirnog rasporeda tijeka razvoja sustava.
- Analiza – podrazumijeva kreiranje referentnog modela za razvoj sustava temeljenog na specificiranim potrebama ustanovljenim u fazi planiranja s posebnim fokusom na tzv. „usability“ ili lakoću/privlačnost korišetnja.
- Konceptualno modeliranje ili dizajn – izgradnja sustava na temelju modela uključujući dizajn sučelja, arhitekture i baza podataka.
- Implementacija – primjena
- Testiranje – use case u praksi radi utvrđivanja problema i pogrešaka
- Objavljivanje – puštanje u opticaj tj. „dizanje“ na mrežu
- Održavanje i nadogradnja. (Brambilla et al., n.d.)

9.2. WebML

WebML je akronimni naziv za „Web modelling language“ a radi se o „vizualnom jeziku za specificiranje strukture sadržaja web sustava ili aplikacije te organizacije i prezentacije tog sadržaja hipertekstom.“ (Ceri et. al. 2000.). WebML također se naziva metodologijom, temeljenom na iterativnom i inkrementalnom pristupu, a koja pruža formalne grafičke specifikacije tj. notacije procesu dizajna podatkovno intenzivnih sustava. Radi se o web metodologiji ostvarenoj u četiri verzije osmišljenoj 1998. godine

u jeku ranih hipermedijskih modela, na temeljima dojmova o Boehmovom spiralnom modelu. Korištenje WebML procesa dizajna ima za cilj ustanoviti strukturu Web sustava ili aplikacije na visokoj razini, kako bi se kasnije mogla iskoristiti kao referenca pri daljnjoj nadogradnji i održavanju, ali i omogućiti sagledavanje sadržaja sustava iz više perspektiva. U ovom procesu definiraju se stranice - navigacija i prezentacija kao i pohranjivanje meta podataka u svrhu kasnijeg dinamičkog generiranja, modeliraju se korisnici i grupe te njihove mogućnosti manipuliranja sadržajima. U sklopu ove cjeline bitno je prikazati konceptualno modeliranje u vidu podatkovnog i hipertekstualnog dizajna.

- **PODATKOVNI DIZAJN** – u modeliranju kompatibilan s UML dijagramima Klasa kao i ER (entity relationship) modelom podataka. Temeljni elementi u modeliranju su entiteti u vezama koji sadržavaju podatke, imaju atribute te međusobne semantičke poveznice koje ukazuju na poziciju u hijerarhiji i notacije koje označavaju ograničenja kako je prikazano slikom 52. Pojedini entiteti imaju jedinstvene identifikacijske ključeve ili šifre.



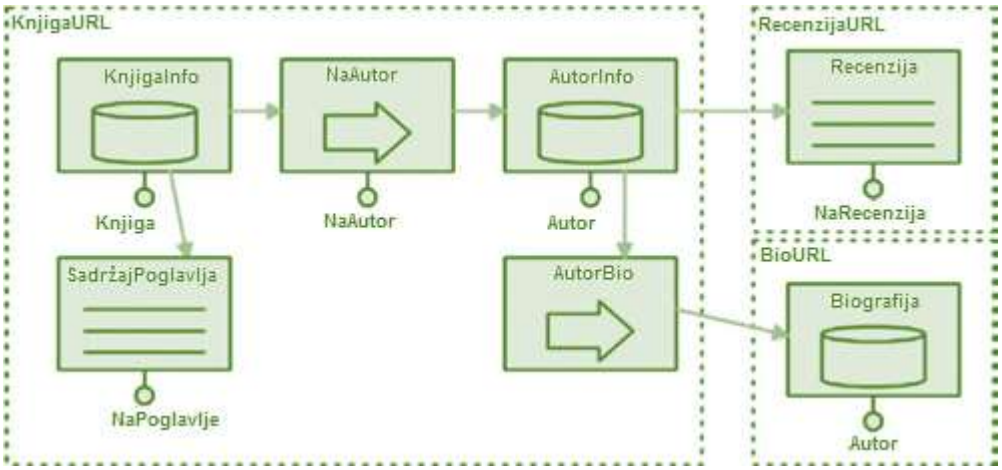
Slika 52. Modeliranje klasa – podatkovni dizajn;
(Izvor:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.472.4706&rep=rep1&type=pdf,
datum pristupa: 04.06.2018.)

- **HIPERTEKST DIZAJN** - utvrđuje kompoziciju i navigaciju web stranice koja je nosioc sadržaja koji se prezentira čitatelju tj. posjetiocu stranice.

Kompozicija se ostvaruje organiziranjem i prezentiranjem međusobno povezanih sadržajnih komponenti stranica. Hipertekstualni dizajn koristi pet elemenata sadržaja koji jesu:

1) Podaci – prikaz podataka o jednoj instanci entiteta
2) Multi-podaci – prikaz podataka o svim instancama entiteta
3) Indeks – prikaz svojstava skupine instanci entiteta

4) Scroller – naredbe koje omogućavaju kretanje kroz sadržaj
5) Unos – vršen preko obrazaca u kojima korisnik vrši input



Slika 53. Navigacija;
(Izvor:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.472.4706&rep=rep1&type=pdf,
datum pristupa: 04.06.2018.)

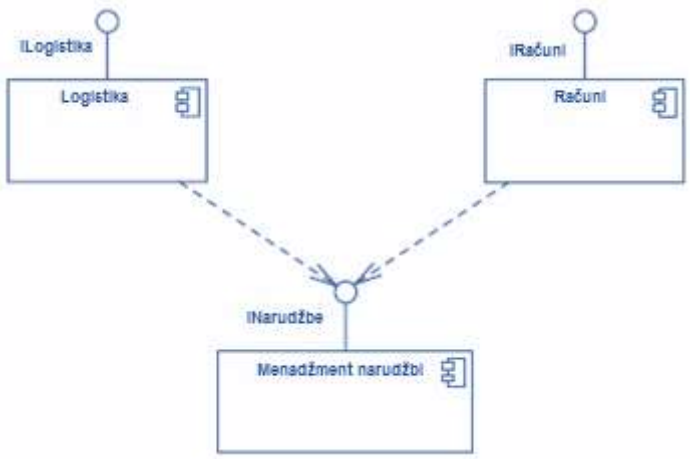
Navigacija stranice ostvaruje se hipervezama ili „link-ovima“ koji tvore veze unutar jedne ili više stranica - tvoreći svojevrsne kontekstualne relacije. (Ceri, S et. al. 2000.)

9.3. Razvoj temeljen na komponentama

Komponenta je modul koji sadrži skup povezanih funkcija ili objekata i sučelje koje omogućava interakciju s drugim komponentama koje mogu koristiti njezine funkcije. Razvoj komponenti je rezultat rastuće potrebe za bržim i jednostavnijim razvojem sustava, pogotovo velikoopsežnih, kao i njihovim mijenjanjem po potrebi.

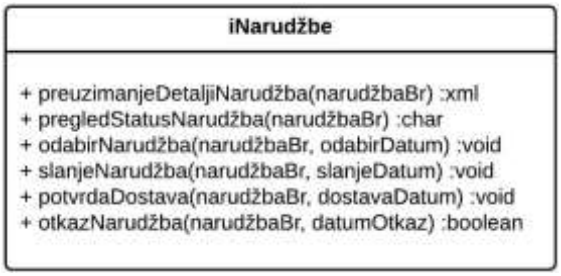
Razvoj komponenti i razvoj informacijskih sustava temeljenih na komponentama su dva različita procesa razvoja. Komponente se razvijaju sa idejom ostvarivanja svojstva ponovne iskoristivosti. Pri razvoju novog sustava potrebne komponente se mogu ponovno iskorištavati, nema potrebe za ponovnim razvojem tih djelova funkcionalnosti. Ukoliko već nisu dostupne gotove komponente one se mogu kupiti, ali je bitno pažljivo utvrditi koje komponente su potrebne i provesti verifikaciju. (Crnkovic, Larsson i Chaudron, 2005.)

Razvoj s komponentama je iterativno inkrementalni proces, a korištenje komponenti je svojstveno razvoju online usluga kao npr. e-trgovine dok su neke komponente dostupne putem cloud-a. Neke komponente mogu funkcionirati zasebno dok neke trebaju druge komponente kako bi funkcionirale.



Slika 54. Povezanost komponenti;
(Izvor: Cadle, 2014.)

Slika 54. prikazuje jednostavni UML dijagram komponenti na kojem su vidljive komponente sa svojom specifikacijom funkcionalnosti i sučeljima kojima se povezuju kako bi međusobno koristile funkcionalnosti koje sadrže. Komponente *logistike* i *računa* imaju veze ovisnosti s komponentom *menadžmenta narudžbi*, kao što je prikazano isprekidanim strelicama, što znači da obje koriste njezine funkcije. Način na koji jedna komponenta „poziva“ funkciju druge komponente lakše je pojasniti detaljnijim prikazom sučelja, u ovom slučaju, konkretno, komponente *narudžbi* prikazane Slikom 54.



Slika 55. Sučelje komponente s funkcijama;
(Izvor: Cadle, 2014.)

Stavke sučelja predstavljaju diskretne funkcije „preuzimanjeDetaljiNarudžbe“ koje mogu biti pozivane od strane drugih komponenti koristeći formatirane poruke čiji je format specificiran u nastavku iste stavke „narudžbaBr“ da bi komponenta zatim onoj koja „poziva“ vratila traženu uslugu koja je u ovom slučaju u obliku XML podatkovne strukture „:xml“. (Cadle, 2014.)

Razvoj informacijskog sustava temeljen na komponentama započinje odabirom potencijalnih komponenti za integraciju. Razmatrane komponente se zatim procjenjuju s obzirom na primjerenost za ciljano područje podržano sustavom, posebice s obzirom na pitanje mogućnosti integracije. Kada je izvršen odabir komponenti, analizira se

način na koji će biti raspoređene i povezane kako bi se ostvarile potrebne funkcionalnosti – dizajnira se arhitektura u koju će se komponente uklapati. Vrši se integracija s obzirom na specifikaciju arhitekture i testiranje kako bi se ustvrdilo da je integracija uspješno i logički ispravno provedena te da su ostvarene sve potrebne funkcionalnosti. (Pressman, 2010.)

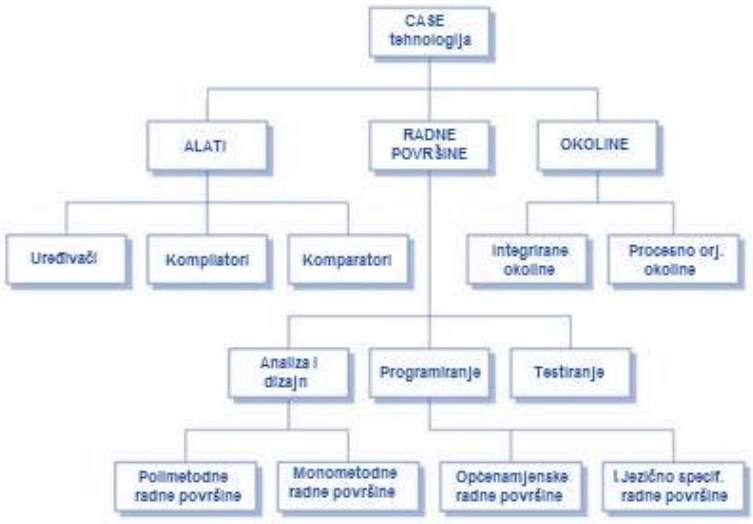
10. CASE ALATI

CASE je akronim za „Computer-aided software engineering“, a naziv je za skupinu tehnologija koja uključuje alate kojima se automatiziraju neki zadaci (pogotovo oni koji se ponavljaju) tijekom razvoja informacijskih sustava što može doprinijeti na uštedi vremena, produktivosti i smanjenju grešaka.

Naziv CASE je originalno osmišljen od strane Nastec korporacije koja je svojedobno izdala nekoliko integriranih alata za obradu teksta i grafike, a to su bili prvi alati temeljeni na tehnologiji mikroprocesora kojima se moglo provesti logičku i semantičku evaluaciju dijagrama dizajna sustava i kreirati rječnik podataka. (Izvor: SBS Inc.)

U jednom izvoru se kao predložena kategorizacija CASE tehnologije navodi podjela na:

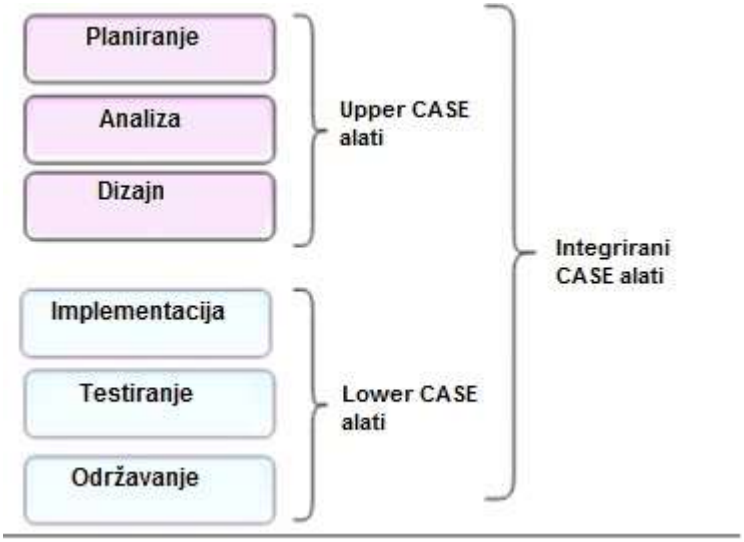
- „**Alate** – u podršci pojedinačnim zadanim procesima poput provjere konzistencije dizajna, kompiliranja programa, usporedbe testnih rezultata itd. Alati mogu biti opće ili pojedinačne primjene.
- **Radne površine** – u podršci fazama procesa ili aktivnostima poput specifikacija, dizajna, itd. Obično se sastoje od skupine alata različitih stupnjeva integracije.
- **Okoline** – u podršci svih ili većine procesa. Obično uključuju više različitih integriranih radnih površina.“ (Fuggetta, A., 1993.)



Slika 56. CASE Tehnologija;
Izvor: https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/CASE/ToolsWBEnv.htm
(datum pristupa: 14.10.2018.)

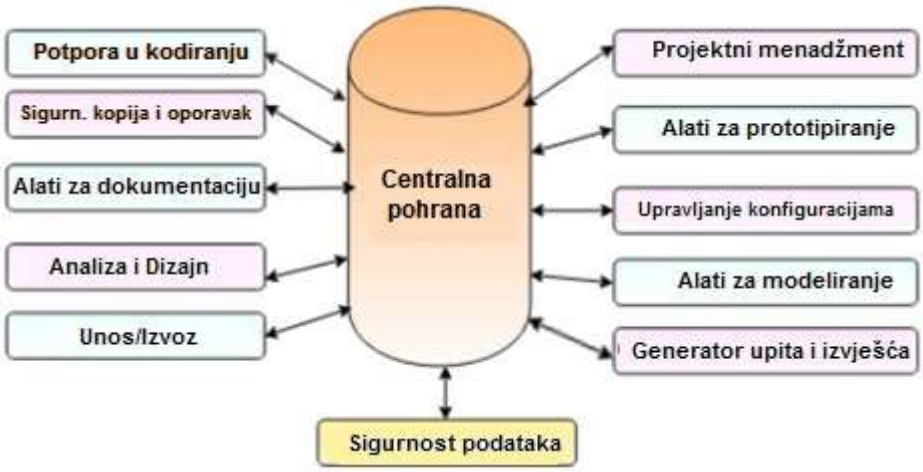
CASE alati su dostupni zasebno ali i kao integrirani paketi programa koji sadrže razna korisna rješenja na jednom mjestu, kao npr. alate za uređivanje sadržaja i alate za generiranje grafičkih prikaza, statistika, izvješća, analiza, koda, dokumentacije, itd.

Općenita podjela CASE alata je prema fazama razvoja pri kojima se primjenjuju. Tako se CASE alati koji potpomažu razvitak primjenom u prvim trima fazama (planiranju, analizi i dizajnu) nazivaju „***Upper CASE***“ alatima, oni u zadnjim trima fazama (implementaciji, testiranju i održavanju) „***Lower CASE***“ alatima, a oni koji se primjenjuju kroz sve faze razvitka i čine međufunkcionalnu poveznicu između skupina, Integriranim CASE alatima ili skraćeno, ***ICASE*** alatima.



Slika 57. Kategorizacija CASE alata;
Izvor: https://www.careerride.com/page/case-tools-663.aspx
(datum pristupa: 14.10.2018.)

Centralno spremište je ključno za CASE alate kao izvor integriranih konzistentnih podataka i čini osnovni dio CASE strukture. Radi se o centralnoj jedinici gdje se pohranjuju specifikacije, dokumentacija zahtjeva, povezana izvješća i dijagrami te druge korisne informacije. U njemu je također sadržan rječnik podataka. ***Rječnik podataka*** je svojstven CASE alatima a radi se o metapodacima kojima se identificiraju elementi podataka, njihovi nazivi i definicije, njihova svojstva i mjere itd. U osnovi su to podaci o podacima. No tu su također bitni i drugi podaci, pogotovo oni koji ukazuju na odnose među podacima i način na koji se logička obrada podataka odražava kroz baze podataka, dokumente, mreže itd. Bilo koji program ili dijagram koji se koristi pri razvoju prima podatke iz centralnog spremišta i pohranjuje podatke u njega.



Slika 58. Centralna pohrana;
Izvor: https://www.careerride.com/page/case-tools-663.aspx
(datum pristupa: 14.10.2018.)

Centralno spremište integrira specifikacije dobivene od različitih alata dozvoljavajući dijeljenje specifikacija bez konverzije između alata. (Loucopoulos i Karakostas, 1995.)

Kod primjene CASE alata pri analizi i razvoju bitno je da su vrlo precizno i dobro utvrđene korisničke potrebe i zahtjevi jer će, u suprotnome, automatizacija loše definiranih procesa dovesti do loše definiranih rezultata, što pobija koristi CASE alata i podržava stari princip koji kaže: „Garbage in, garbage out“, a u osnovi, u ovom konkretnom slučaju znači „loš input, loš output“.

S obzirom na funkcionalnosti kojima su podržane aktivnosti razvoja, CASE podrazumijeva mnogobrojne alate, npr:

- *Alati za kreiranje dijagrama* – koriste se za prikazivanje strukture sustava sa njezinim elementima u vidu komponenti, podataka, toka podataka i veza između komponenti u grafičkom obliku. npr. Flowchart Maker.
- *Alati za modeliranje procesa* – npr. ArgoUML
- *Alati za upravljanje projektima* – koriste se za planiranje projekata, analiziranje troškova, planiranje resursa, pravljenje rasporeda itd. pohranjujući te podatke u realnom

  vremenu, čineći ih dostupnima na nivou organizacije. Npr. Basecamp
- *Alati za dokumentiranje* – generiraju dokumentaciju tijekom cijelog procesa razvoja za tehničke i krajnje korisnike koja sadrži sve potrebne informacije o sustavu. npr. Doxygen, Adobe RoboHelp

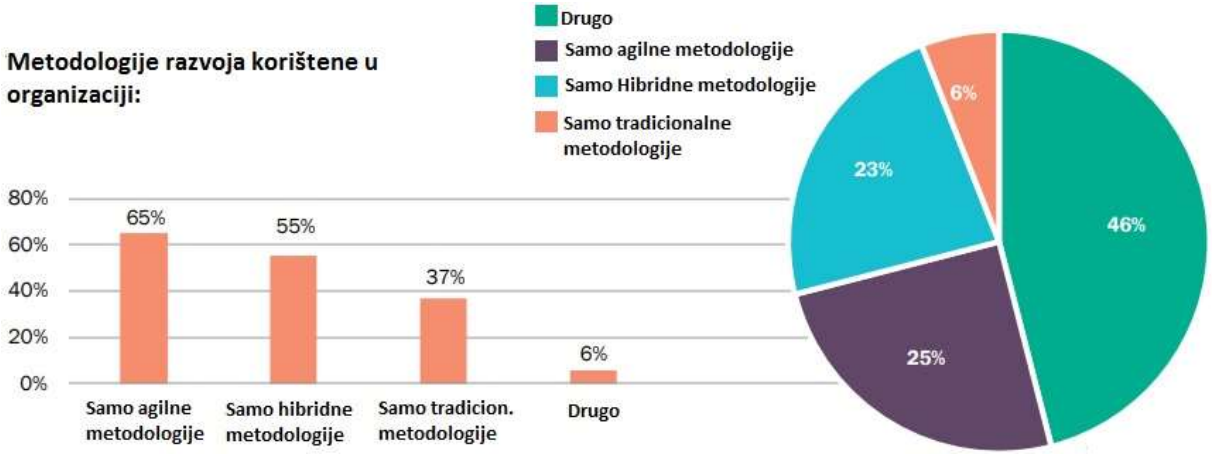
- *Alati za analizu* – pomažu pri prikupljanju zahtjeva, automatski vrše provjeru konzistentnosti, ispravnosti u dijagramima, redundancije podataka i propusta. npr. Accept 360, Accompa,
- CaseComplete, Visible Analyst
- *Alati za dizajn* – pomaže dizajnerima u kreiranju blok strukture softvera koji se može razbiti na module. Npr. Animated Software Design
- *Alati za upravljanje konfiguracijama* – služe kao potpora upravljanju verzijama, revizijama, izdanjima i sl. npr. Fossil, Git, Accu REV.
- *Alati za programiranje* – sadrže programerske okoline pomoću kojih se kodira, izvode simulacije i testovi. npr. Cscope, Eclipse.
- *Alati za prototipe* – prototipi su simulirane verzije sustava koji se razvija i pružaju uvid u neke aspekte završne verzije. Ovi alati sadrže grafičke knjižnice, generiraju korisnička sučelja i razne slične dizajne i omogućavaju brzo prototipiranje. npr. Serena prototype composer, Mockup Builder.
- *Alati za internetski razvoj* – pomažu u izgradnji web stranica s pripadajućim elementima poput obrazaca, teksta, grafika itd. U procesu prikazuju kako izgleda ono što se kreira. npr.
- Fontello, Adobe Edge Inspect, Foundation 3, Brackets.
- *Alati za osiguranje kvalitete* – npr. SoapTest, AppsWatch, JMeter.
- *Alati za održavanje* **–** održavanje se odnosi na modifikacije rješenja nakon što je implementirano, a pritom su korisni automatsko evidentiranje i izvještavanje o greškama i analiza temeljnih uzroka grešaka. (Tutorialspoint, n.d.)

U osnovi, CASE alati se koriste kako bi se: smanjili troškovi automatizacijom repetitivnih zadatak, smanjilo vrijeme razvoja, poboljšala kvaliteta složenih projekata kroz osiguravanje konzistentnosti, poboljšala kvaliteta dokumentacije te da bi se razvili održivi sustavi uz pomoć ispravne kontrole konfiguracija koja omogućava praćenje zahtjeva.

11. USPOREDBA I ANALIZA SUVREMENIH METODA RAZVOJA

Odmak od klasičnih metodologija nije samo procesno strukturološke i ideološke prirode. Pojava zagovornika agilnih metoda označava novo razdoblje obilježeno potrebom, kako za novom generacijom teoretičara, tako i za, sukladno tome, odgovarajućim razvojnim timovima u praksi, a sve na valu razvoja novih tehnologija. Suvremene metode, za razliku od klasičnih metodologija, ne teže standardiziranom i formaliziranom procesu koliko teže prilagodljivosti raznovrsnim zahtjevima.

Neke statistike o općenitom korištenju agilnih i tradicionalnih metoda mogu se naći u sklopu raznih publikacija. Jedno istraživanje objavljeno u HP-ovoj publikaciji iz 2016. godine (HP https://techbeacon.com/sites/default/files/gated_asset/agile-projects-are-more-successful-than-hybrid-projects.pdf, datum pristupa: 14. 09. 2018.) je obradom zaprimljenih odgovora iz različitih organizacija, a na temelju odgovora 403 sudionika, došlo do sljedećih podataka o korištenju tradicionalnih i agilnih metoda:

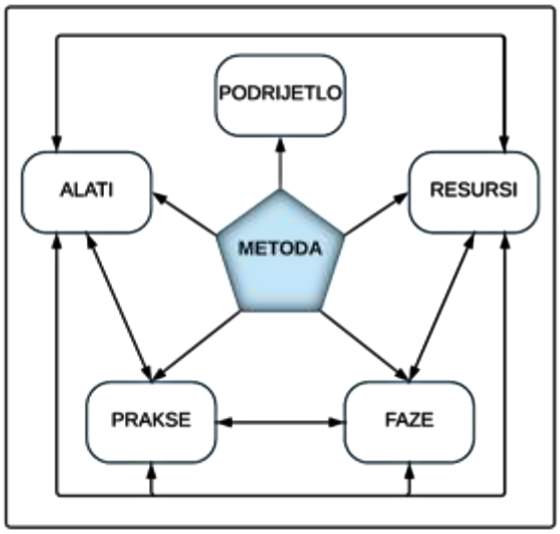


Slika 59. Distribucija primjene metodologija/metoda;
(Izvor: https://techbeacon.com/sites/default/files/gated_asset/agile-projects-are-more-successful-than-hybrid-projects.pdf, datum pristupa: 14. 09. 2018.)

Gornja slika predstavlja statistiku dobivenu iz upitnika čije je pitanje glasilo: Odaberite odgovarajuće vrste razvojnih metoda koje koristite u svojoj organizaciji. Odgovor je dopuštao višestruki izbor što objašnjava postotke iz grafa. Istraživanje je pokazalo da oko trećina sudionika koristi tradicionalne metodologije, oko dvije trećine koriste agilne, a nešto više od polovice koristi neke hibridne metode. Preneseno u pregled presjeka na desnoj strani slike možemo vidjeti ekskluzivnu raspodjelu korištenja.

Kada se pokušava izvršiti usporedba klasičnih i suvremenih metoda ona je moguća većinom unutar faza razvoja promatranih međusobno nezavisno no čak i

unutar faza naišlo bi se na razlike u aspektima poput opsežnosti, dokumentacije, karakteristika timova, prioretizacije zadataka, praksi, alata, uvjeta i okoline itd. Poneke agilne metode i djelomično ili u potpunosti ne „pokrivaju“ određene faze propisane klasičnim metodologijama ili zapravo ne opisuju konkretne akcije unutar faza koliko pružaju vodstvo za nekim principima i ideologijama. Neke od njih je moguće „krojiti“ i kombinirati. Razne karakteristike pojedinih metoda utječu na primjerenost primjene neke metode na razvoj projekata IS-a s obzirom na specifičnosti projekta. Uzrok neuspješnosti ponekih projekata može se pripisati i izboru neodgovarajućeg pristupa razvoju. Bitno je odabrati odgovarajuću metodu ili kombinaciju metoda na temelju ispravno utvrđenih karakteristika projekta i metode. Cilj rada je komparativnom analizom omogućiti taj odabir. Komparativna metoda u ovom radu izvršena je kroz analitički okvir usporedbe koji se sastoji od pet odrednica, a to su: podrijetlo, resursi, faze, prakse i alati.



Slika 60. Analitički okvir usporedbe metodologija;
(Izvor: Izrada autora)

Metoda vršenja usporedbe je tablični prikaz. U sklopu usporedbe uvrštene su i tradicionalne metodologije i objektno orijentirani pristup kako bi se zorno prikazali odmaci između metoda no fokus ovog rada je na agilnim metodama.

11.1. Podrijetlo

Podrijetlo svake metode određeno je relevantnim podacima o njezinu nastanku a to su podaci koji pružaju informacije vezane uz mjesto i vrijeme njezina nastanka te njezina začetnika ili začetnike odnosno autore. Razradu podrijetla metoda lako se može skupno prikazati jednostavnim tabličnim prikazom u koji su uvršteni navedeni

podaci kategorički podijeljeni u tri skupine: tradicionalne, objektno orijentirane i agilne metode – uz uvrštavanje dviju zasebnih kategorija, a to su brzi i grupni razvoj aplikacija.

Tablica 3. Usporedba metodologija/metoda po podrijetlu

	AUTORI	VRIJEME	MJESTO
TRADICIONALNE METODOLOGIJE			
Model Vodopada	Winstonu W. Royceu pripisuje se opis modela kojeg će kasnije Bell i Thayer imenovati modelom „vodopada“ iako postoje osporavajuće izjave koje zaslugu ipak pripisuju ranijem radu Herberta .D. Beningtona (Boehm, B. W. 1987.)	1956./1961./1970.	Washington D.C. (SAD) – Simpozij naprednih metoda programiranja za digitalna računala. TRW korporacija (Ohio, SAD)
V Model	null	Kasne 80-e	Razvijan za Ministarstvo obrane u Njemačkoj i za satelitske sustave i avijaciju u SAD-u
Inkrementalni i iterativni razvoj	Rani spomen može se naći u izvješću B. Randella i F.W. Zurchera pod nazivom „Iterative Multi-Level Modelling: A Methodology for Computer System Design“ (Larman, C. 2003.)	1968.	Rani period razvoja u okvirima IBM-a (Los Angeles, SAD) i NASA-e (Larman, C. 2003.)
Prototipiranje	Frederick P. Brooks rano spominje prototipiranje u knjizi pod nazivom „The Mythical Man-moth“. Rani primjer prakse je u implementaciji prevoditelja za programski jezik Ada.	1975./1983.	Knjiga F.P. Brooksa proizišla je iz njegova rada u IBM-u dok je programski jezika Ada bio projekt za SAD-ov Odjel za obranu.
Spiralni razvoj	U članku Barryja Boehma pod nazivom: “A Spiral Model of Software Development and Enhancement“	1986.	Na temelju rada u TRW korporaciji u SAD-u.
BRZI RAZVOJ APLIKACIJA	James Martin u knjizi „Rapid Application Development“ na temelju ideja Barryja Boehma.	1980-e/1991.	IBM
GRUPNI RAZVOJ APLIKACIJA	Osmislio Chuck Morris. Formalno objavljeno u sklopu IBM-ove publikacije brošure.	1977.	IBM

OBJEKTNO ORIJENTIRANI PRISTUP	Korijeni u ranom radu O.J. Dahla i K. Nygaard na SIMULA jeziku.	1961.	Norveški računalni centar
Unified Modelling Language - UML	G. Booch, I. Jacobson i J. Rumbaugh	1994./1995.	Rational Software u Kaliforniji (SAD) – sada dio IBM-a.
Rational Unified Process - RUP	P. Kruchten uz temelju rada G. Boocha i I. Jacobsona.	1996.-1999.	Rational Software u Kaliforniji (SAD) – sada dio IBM-a.
AGILNE METODE	Metode s agilnim pristupom postaju poznate pod nazivom „Agilne metode“ tek publikacijom iz 2001. pod nazivom „Manifesto for Agile Software Development“ koju potpisuju autori pojedinih agilnih metoda.		
Kristalne metode	Alistair Cockburn – parcijalno u sklopu članka pod nazivom „Surviving object-oriented projects“ a zatim i knjigom „Crystal Clear: A Human-Powered Methodology for Small Teams“	1998./2004.	Boston, Massachusetts (SAD) za IBM
Dinamička metoda razvoja sustava - DSDM	Osmišljena u sklopu nekadašnjeg DSDM Konzorcija koji se danas naziva „Agile Business Konzorcij“, a zatim razrađena u knjizi Jennifer Stapleton pod nazivom „Dsdm: The Method in Practice"	1994./1997.	Engleska kao sjedište konzorcija, a mjesto publikacije knjige je Boston, Massachusets (SAD)
Scrum	Jeff Sutherland, John Scumniotales, Jeff McKenna, Mike Beedle i Ken Schwaber po uzoru na proizvodnu aplikaciju H. Takeuchija i I. Nonake	1993.	Burlington, Massachusetts (SAD) u sklopu Easel Korporacije
Feature Driven Development - FDD	Osmislio Jeff De Luca uz pomoć Petera Coada - prvi put formalno izneseno u knjizi pod nazivom „Java modelling in Color with UML“	1997./1999.	Osmišljeno u sklopu projekta za banku u Singapuru, a mjesto publikacije knjige je New Jersey (SAD)
Ekstremno Programiranje - XP	Kent Beck u knjizi „Extreme Programming Explained“	1999.	Osmišljeno tijekom autorova rada za Chrysler. Mjesto publikacije: Boston, Massachusets (SAD)
Adaptivni razvoj sustava - ASD	Jim Highsmith u knjizi „Adaptive Software Development: A Collaborative Approach to Managing Complex Systems“	1999.	New York, New York (SAD)

Lean Development	Mary Poppendieck i Tom Poppendieck u knjizi pod nazivom „Lean Software Development: An Agile Toolkit“	2003.	Adaptirano i z Toyotinog proizvodnog sustava. Mjesto publikacije: Boston, Massachusetts (SAD)

(Izvor: Izrada autora)

Promotrivši ovu tabličnu usporedbu može se konstatirati kako je većina današnjih metoda proizašla iz posljednjih trideset godina prošlog stoljeća. Glavna bujica zbivanja u razvoju tradicionalnih metodologija uglavnom je imala uporište u SAD-u - u granama vojne industrije - posebice avijacije i navigacije pa zatim, kao prirodni produžetak toga, i NASA-inih potreba u pionirskim misijama otkrivanja svemira. Poligon za razvoj metodologija često su bila sveučilišta ili velike softverske korporacije gdje su stručnjaci na tom polju često bili upoznati s međusobnim radom u zajedničkim primjenama – napomenimo npr. nekadašnju TRW korporaciju ili IBM koji se gotovo svugdje ističe i kasnije. Uglavnom se prve publikacije vezane za pojedine metodologije mogu smjestiti na područje SAD-a, a pogotovo Boston u saveznoj državi Massachusets. Poslije, razvojem industrije i informacijske tehnologije, raste potreba za bržim metodama razvoja koje bi mogle držati korak sa promijenama i sve raznovrsnijim primjenama pa su zato neki nedostaci tradicionalnih metodologija riješeni prvo objektnim pristupom koji je omogućio opetovano korištenje zasebnih funkcijskih jedinica softvera, a zatim i suvremenijim agilnim metodama pogodnim za suvremenu ekonomiju napredne industrije, poduzetništva te privatnih i javnih usluga upravljanja.

Primjene suvremenijih metoda nalaze se u auto-moto industriji, medicini, financijskom tržištu, proizvodnji, informacijskim tehnologijama, javnoj upravi itd. Razvoj agilnih metoda možemo većinom smjestiti u vremenski okvir 90-ih godina prošlog stoljeća. Kao prominentno mjesto bitnih publikacija opet se nameće nakladnička kuća Addison-Wesley smještena u Bostonu. Kao žarište razvoja i dalje se ističe SAD unatoč rastu svjetske ekonomije, seljenju proizvodnje i usluga u različite zemlje svijeta, posebice Azijskih zemalja, te rastu broja metropola i inovativnih razvojnih centara.

Mogućnost usporedbe suvremenih metoda na temelju **podrijetla** se smatra opravdanom jer je:

- ✓ zadovoljena vremenskim okvirom – većina agilnih metoda potječe iz razdoblja 90-ih godina prošlog stoljeća.

- ✓ zadovoljena autorskim kadrom – iako su individualni autori potpisnici različitih metoda, također su zajednički potpisnici bitne publikacije – Manifesta agilnog razvoja softvera – koji objedinjuje vrijednosti i principe na kojima počivaju pojedine metode, stavivši ih na taj način pod zajednički ideološki nazivnik.
- ✓ zadovoljena mjesnim okvirom – većina metoda pojavljuje se na području SAD-a.

11.2. Resursi

Različiti projekti zahtjevaju različite količine različitih resursa i s obzirom na nepredvidivost broja iteracija, posebno kod složenih sustava, izazov je unaprijed uspješno aproksimirati dodjelu resursa za cijeli projekt. Varijabilna priroda jednog resursa utječe na zahtjeve za drugim resursima. Resursi su važan varijabilni čimbenik razvoja koji uvjetuje izvedivost projekta od samog početka kroz cijeli ciklus. Bitno je utvrditi hoće li se projektu dati „zeleno svjetlo“ s obzirom na raspoložive resurse. Uzrok neuspješnih projekata može se naći i u pretjeranom prekoračenju dodijeljenih resursa.

Izbor metode razvoja ovisiti će i o raspoloživim resursima. Različite metode mogu iziskivati različitu količinu resursa za isti projekt i zato će odabir metode ovisiti o raspoloživim resursima. U tablici br. odabrani resursi koji su najčešće najbitniji stavljeni su u odnos s metodama kao odabrani zahjevi i s obzirom na to je prikazano koja je metoda primjerenija za kakve zahtjeve.

Tablica 4. Usporedba Resursa

	Model Vodopada	**V model**	**Iterat. razvoj**	**Prototip.**	**Throw. Prototip.**	**Spiral**	**Objekt. Orijent.**
Resursi							
OSKUDNO VRIJEME	–	–	✓	–	✓	–	✓
UZAK PRORAČUN	–	–	/	–	–	–	/
ZAHTJEVNOST VJEŠTINA	–	✓	✓	✓	/	✓	✓
NEPOZNATA TEHNOLOGIJA	–	–	✓	–	✓	✓	✓
KVALITETA	–	✓	✓	✓	✓	✓	✓

	Kristalne metode	**Scrum**	**XP**	**DSDM**	**FDD**	**ASD**	**Lean**
Resursi							
OSKUDNO VRIJEME	✓	–	✓	✓	–	✓	✓
FIKSNI PRORAČUN	–	–	–	✓	–	–	–
ZAHTJEVNOST VJEŠTINA	✓	✓	✓	✓	✓	✓	✓
NEPOZNATA TEHNOLOGIJA	–	✓	✓	✓	–	✓	✓

KVALITETA	–	✓	✓	✓	✓	✓	✓

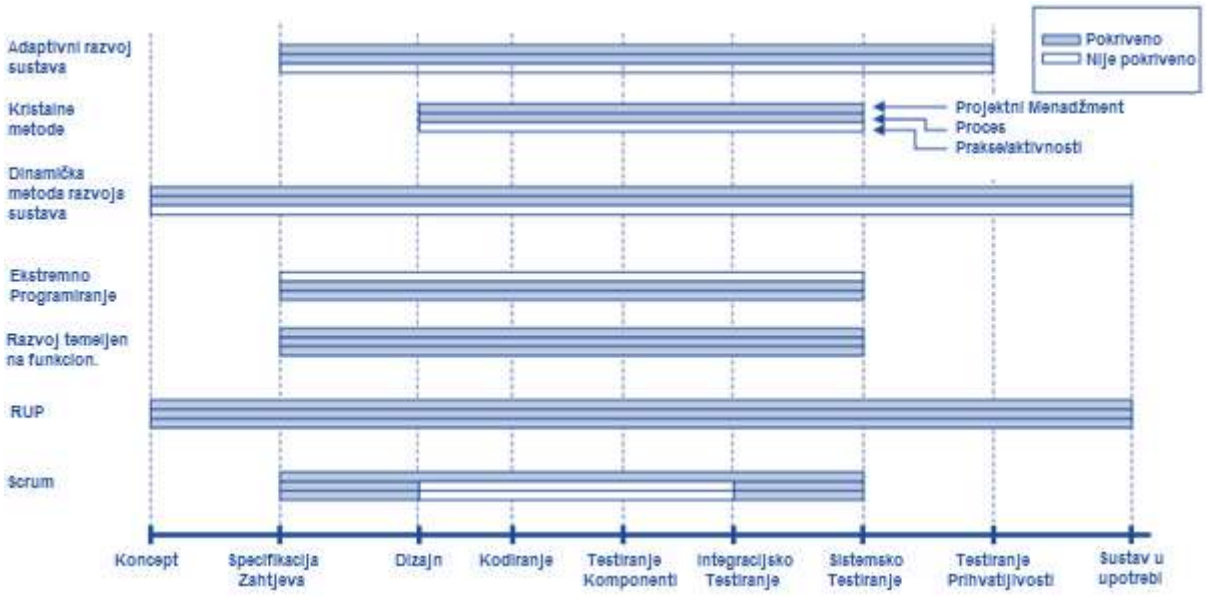
(Izvor: Izrada autora)

Može se vidjeti da je primjerenost primjene više agilnih metoda većinom u slučajevima kada su postavljeni zahtjevi zbog kojih:

- treba uštedjeti na vremenu,
- postoji fleksibilan proračun,
- postoje visoki zahtjevi za vještinama,
- je potrebna tehnologija nepoznata
- želimo kvalitetu

11.3. Faze

U ovoj usporedbi govori se o pokrivenosti ciklusa razvoja u smislu raspona aktivnosti koje pojedine metode predviđaju. Mnoge aktivnosti ovdje uklapaju se u neke od klasičnih faza no razlikuju se u primjenjenim praksama u izvedbi. Ne pokrivaju sve metode cijeli ciklus razvoja, a neke više pružaju smjernice ili vrijednosti u izvedbi no što pružaju konkretne zadatke koje treba izvesti. Također, razlikujemo pokrivenost ciklusa s aspekta projektnog menadžmenta, procesa ili praksi/aktivnosti. Tradicionalne metode imaju pokriće cijelog životnog ciklusa razvoja.



Slika 61. Metodološko pokriće faza;
(Izvor: Abrahamsson, P. et al, 2002. Str. 101)

Iz gornjeg prikaza je vidljivo da:

- ASD, Kristalne metode i DSDM imaju zajedničku karakteristiku u pokrivenosti ciklusa – ne pokrivaju područje **praksi/aktivnosti/proizvoda rada**.

- Scrum - ne pokriva dio područja **procesa** i dio **praksi/aktivnosti/proizvoda rada**
- XP **-** ne pokriva područje **projektnog menadžmenta**
- FDD pokriva sva tri područja
- Sve navedene metode u nekim područjima pokrivaju raspon ciklusa od **dizajna** do **sistemskog testiranja**
- Sve navedene metode, osim Kristalnih metoda, u nekim područjima pokrivaju raspon ciklusa od **specifikacije zahtjeva** do **sistemskog testiranja**.

11.4. Prakse

Tehnološka, poslovna i javna okolina općenito generirale su različite potrebe i zahtjeve, metode su se razvijale u skladu s određenim vrijednostima koje su bile odraz onoga što se u tim prilikama želi postići, a udovoljavanje tim vrijednostima ispunjavalo se uspostavljanjem različitih praksi. Prakse su uvrštene u ovu tablicu s obzirom na primjenjivost kod različitih metoda i iako nisu propisane ipak se najčešće koriste. Uvrštene prakse uspoređuju se za sve kategorije metodologija iako za poneka polja u tablici ili nedostaju podaci ili su oni konfliktne prirode u korištenim izvorima. Prakse su podijeljene s obzirom na razvoj, projekt i resurse. **CILJ** ove raščlambe je da posluži kao **pomoć u izboru metode**.

Tablica 5. Usporedba Praksi

Primjena:	**Model Vodopada**	**V model**	**Iterat. razvoj**	**Prototip.**	**Throw. Prototip.**	**Spiral**	**Objekt. Orijent.**
Razvoj							
PROMJENJIVI ZAHTJEVI	–	–	✓	✓	✓	✓	✓
PRAĆENJE NAPRETKA	–	✓	✓	✓	✓	✓	✓
KONTINUIRANA UKLJUČENOST KORISNIKA	–	–	✓	✓	✓	✓	✓
KONTINUIRANI FEEDBACK	–	–	✓	✓	✓	✓	✓
RADIONICE	–	–	✓	✓	✓	✓	/
BRZE/ČESTE ITERACIJE	–	–	✓	–	✓	✓	✓
INKREMENT. RAZVOJ	–	–	✓	✓	✓	✓	✓
KONTINUIRANA INTEGRACIJA	✓	✓	✓	✓	✓	✓	✓
KONTINUIRANO TESTIRANJE	–	✓	✓	✓	✓	✓	✓
MINIMALNA DOKUMENT.	–	–	✓	✓	–	–	✓

RANA FUNKCIONAL.	–	–	✓	✓	✓	✓	✓
IZRADA PROTOTIPA	–	–	✓	✓	✓	✓	/
Projekt							
MALI TIMOVI	–	–	✓	✓	✓	–	✓
DISLOCIRANI TIMOVI	–	–	✓	✓	✓	✓	✓
MALI SUSTAVI	–	–	✓	✓	–	–	✓
SLOŽENI SUSTAVI	✓	✓	✓	–	✓	–	✓
KRITIČNI SUSTAVI	–	✓	✓	✓	✓	✓	✓
POUZDANI SUSTAVI	✓	✓	✓	–	✓	✓	✓

Legenda:

- ✓ *(primjenjivo je), – (nije primjenjivo), / (nema podatka)*

(Izvor: Izrada autora)

Na temelju tablično uvrštenih podataka mogu se sumirati dominantne karakteristike tradicionalnih i agilnih metoda. Tradicionalne metodologije su formalizirane, linearne i sekvencionalne, prediktivne uz autokratski stil menadžmenta i centralizirane timove. Lako su usvojive i razumljive s obzirom na propisane faze i prakse, obično su potrebni nešto veći timovi za jednostavnim ili složenijim i većim projektima, nema fleksibilnosti i promjenjivih zahtjeva, obujam dokumentacije je visok, rizik je veći, pouzdanost je moguća, uključenost korisnika je minimalna, pokrivenost ciklusa potpuna.

Tablica 6. Usporedba Praksi – Agilne Metode

Primjena:	**Kristalne metode**	**Scrum**	**XP**	**DSDM**	**FDD**	**ASD**	**Lean**
Razvoj							
PROMJENJIVI ZAHTJEVI	✓	✓	✓	✓	✓	✓	✓
PRAĆENJE NAPRETKA	✓	✓	✓	✓	✓	✓	✓
KONTINUIRANA UKLJUČENOST KORISNIKA	✓	✓	✓	✓	✓	✓	✓
KONTINUIRANI FEEDBACK	✓	✓	✓	✓	✓	✓	✓
RADIONICE	✓	✓	✓	✓	✓	✓	✓
BRZE/ČESTE ITERACIJE	✓	✓	✓	✓	✓	✓	✓
INKREMENTALNI RAZVOJ	✓	✓	✓	✓	✓	✓	✓
KONTINUIRANA INTEGRACIJA	✓	✓	✓	✓	✓	✓	✓
KONTINUIRANO	✓	✓	✓	✓	✓	✓	✓

TESTIRANJE							
MINIMALNA DOKUMENT.	✓	✓	✓	✓	✓	✓	–
RANA FUNKCIONAL.	✓	✓	✓	✓	✓	✓	✓
IZRADA PROTOTIPA	/	✓	✓	✓	/	✓	/
Projekt							
MALI TIMOVI	✓	✓	✓	✓	✓	✓	✓
DISLOCIRANI TIMOVI	–	✓	–	✓	✓	✓	✓
MALI SUSTAVI	✓	–	✓	✓	✓	✓	✓
SLOŽENI SUSTAVI	–	✓	✓	–	✓	✓	✓
KRITIČNI SUSTAVI	–	✓	✓	✓	✓	✓	✓
POUZDANI SUSTAVI	–	✓	✓	✓	✓	✓	✓

(*Izvor: Izrada autora)*

Agilne metode su obično pogodnije za manje projekte (Scrum i FDD mogu biti Pogodni i za velike, dok Kristalne metode mogu tek teoretski), proces razvoja je složen, uključenost korisnika je velika, zahtjevnost za vještinama je velika, pogodniji su manji timovi (moguće i dislocirani), menadžment s podjeljenom odgovornosti i odlučivanjem, dokumentacija je u većini slučajeva minimizirana, fleksibilnost je velika, pristup je iterativno inkrementalan uz više iteracija, primjenjuje se brzi razvoj softvera uz kraće i češće iteracije.

11.5. Alati

CASE alati imaju primjenu u agilnom razvoju posebno zbog brzine i uštede. Bilo koji alat koji se može primjeniti kako bi se automatizirale određene aktivnosti poput npr. testiranja ili poduprli određeni aspekti projektnog menadžmenta je alat koji je koristan u praksi. Bitno je da ulaganje u takav alat ne premašuje korist od njegove primjene. Ulaganje u CASE alate nije ograničeno samo na kupovinu samog alata već je bitno uzeti u obzir i kompatibilnost s postojećim tehnologijama, mogućnost konfiguriranja za specifične potrebe, edukaciju korištenja i korisničku podršku. Potrebu i primjenjivost je bitno oprezno utvrditi prije odluke o investiciji zbog statistike koja pokazuje da „u nekim organizacijama do 70% CASE alata koji se ne primjene godinu dana od kupovine nikad ne bude implementirano“, a to znači propalu investiciju. (Jones, W. http://www.umsl.edu/~sauterv/analysis/488_f02_papers/CASE.html, datum pristupa: 06.11.2018.)

Jednom kada je odabrana metoda razvoja, bitno je znati koji primjereni alati su na raspolaganju. Neki aktualni statistički podaci o učestalosti korištenja CASE alata mogu se naći u godišnjem izvješću za 2017. godinu pod nazivom „The 12th annual State of Agile“ (VersionOne https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report , datum pristupa: 18.06.2018.), izrađenom od strane VersionOne kompanije koja pruža alate koji organizacijama omogućavaju provedbu agilnih metoda. Po provedenom istraživanju u kojem su ispitanici imali mogućnost višestrukog odabira vidljiva je sljedeća distribucija najčešće korištenih kategorija alata:

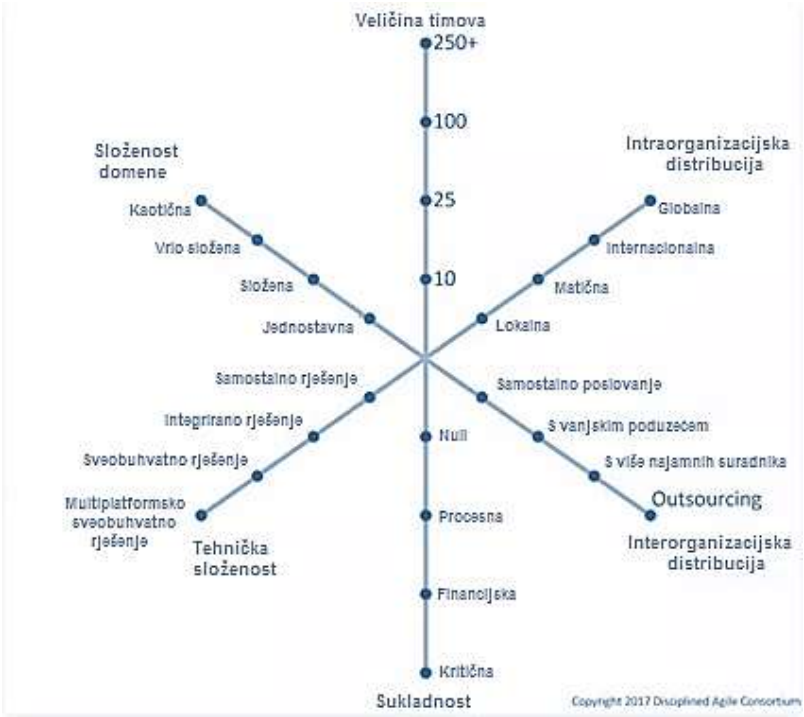
- Kanban ploča – 74%
- Pronalazač bug-ova – 72%
- Taskboard – 71%
- Alat za agilni menadžment projekta – 67%
- Spreadsheet – 65%
- Wiki – 62%
- Alat za automatsko generiranje – 60%
- Alat za testiranje – 57%
- Alat za kontinuiranu integraciju – 52%
- Alat za upravljanje zahtjevima – 46%

Prema navedenom istraživanju, ispitanici su također imali mogućnost višestrukog izbora konkretnih CASE alata

- Atlassian JIRA – 58% - najkorišteniji agilni alat. Služi za planiranje, prioretizaciju, praćenje napretka, ažuriranje i izvješćivanje.
- Axosoft – 1% - Služi za planiranje, prioretizaciju, praćenje napretka, ažuriranje i izvješćivanje.
- Bugzilla – 4% - Alat za praćenje bug-ova i testiranje.
- CA Agile Central – 9% - Alat za planiranje, praćenje, team-building
- CollabNet TeamForge – 2% - Alat za organiziranje složenih, dislociranih timova u svrhu praćenja razvoja.
- Google Docs – 17% - Alat za obradu i formatiranje dokumenata, prezentacija, obrazaca... Integracija s Lucidchartom za izradu dijagrama.
- HP Agile Manager – 2% - Alat za potporu dislociranih timova i procesa uz metriku i praćenje napretka.
- HP Quality Center/ALM – 14% Alat za planiranje, testiranje i praćenje bugova.

- IMB Rational Team Concert – 6% - Alat za timsku suradnju na razvoju sustava – planiranje, praćenje itd.
- LeanKit – 4% - Alat za planiranje, praćenje napretka, metriku, timsku suradnju.
- Microsoft Excel – 46% - alat za izradu tablica, kalkulacija, grafikona, pivot tablica (služi za analizu podataka, praćenje i vizualizaciju)
- Microsoft Project – 21% - Alat za projektni menadžment – za planiranje, raspodjelu resursa za zadatke, praćenje napretka, upravljanje proračunom i analizu obujma posla.
- Microsoft TFS – 21% - Alat za upravljanje izvornim kodom, izvješćivanje, upravljanje zahtjevima, menadžment projekta, automatizaciju, testiranje itd. Pokriva cijeli ciklus razvoja.
- VersionOne – 20% - Alat za lean i agilni razvoj – služi za upravljanje projektom, planiranje, izvješćivanje, analitiku, metrike, praćenje iteracija i izdanja, suradnju timova, praćenje bugova...

Za CASE alatima se najčešće javlja potreba kod suočavanja s povećanim razinama složenosti, a čimbenici koji utječu na razine složenosti mogu se prikazati dijagramom na slici .



Slika 62. Razine faktora složenosti;
(Izvor: http://www.disciplinedagiledelivery.com/tool-support-for-dad/ , datum pristupa: 07.11.2018.)

Čimbenici razine složenosti jesu:

1. **Organizacijska složenost koja** podrazumijeva:

a) Veličinu timova – veličina timova utječe i na način organiziranja timova i međukoordinaciju. Veličina timova će ovisiti o složenosti domene primjene i tehničkoj složenosti.

b) Intraorganizacijsku distribuciju – razvojni timovi unutar organizacije mogu djelovati zajedno u jednoj prostoriji ili više njih, u različitim gradovima ili različitim zemljama. Dislocirani timovi nisu neuobičajena pojava u angažmanu na razvoju sustava. Većina agilnih metoda predviđa potrebu dislociranih timova.

c) Interorganizacijsku distribuciju – suradnja između organizacija, outsourcing, rad s vanjskim suradnicima ili pružateljima usluga

2. **Sukladnost sa standardima i regulativama** – sukladnost sa standardima poput CMMI ili ISO standarda je proizvoljna dok je sukladnost s regulatornim standardima obvezna. Postoje procesna, financijska i kritična sukladnost. Sukladnost sa standardima obično podrazumijeva vođenje neke podržavajuće dokumentacije i preglednih izvješća.
3. **Složenost domene primjene** – različite domene primjene ili područja primjene podrazumijevaju različite razine složenosti s obzirom na uključenu količinu informacija, znanja i stručnosti. Složenije domene podrazumijevaju potrebu za sofisticiranim alatima.
4. **Tehnička složenost** – u smislu zahtjevnosti modeliranja arhitekture i dizajna. (DA Toolkit http://www.disciplinedagiledelivery.com/tool-support-for-dad/, datum pristupa: 07.11.2018.)

Tablicom prikazane su potencijalne potrebe za CASE alatima u situacijama viših razina složenosti navedenih čimbenika. Svaka točka u stupcu „Potreba za alatima“ redom odgovara točki u stupcu „Alati“.

Tablica 7. Faktori složenosti i alati

Faktor složenosti	Potreba za alatima	Alati
Organizacijska složenost: - *Veličina timova* - *Intraorganizacijska distribucija* - *Interorganizacijska distribucija*	• Alati za komunikaciju u suradnji poput e-mail i chat servisa, alata za video konferencije itd. • Alati za planiranje i menadžment projekata • Alat za menadžment konfiguracija • Alat za kontinuiranu integraciju	• Slack, Trello, Skype, Google Hangouts, Huddle, Zoho, Samepage,... • Agilean, ScrumDesk, Wrike, Scrumblr, Basecamp, Planbox, Pivotal Tracker, Asana, Agilefant, Confluence... • Chef, Ansible, Codenvy, AWS OpsWorks, Octopus

		Deploy, IBM BigFix, SCCM,... • Jenkins, TeamCity, Bamboo, GitLab, Travis CI, CircleCI,...
Sukladnost sa standardima i regulativama	• Alat za dokumentaciju • Alat za provedbu automatiziranih testova • Alati za izvješća	• Read The Docs, Swagger, Javadoc, MarkdownPad, Atlassian REST API Browser, Haroopad,... • Selenium, Watir, TestComplete, Katalon Studio, Sahi, Cypress, Unified Functional Testing,... • Pentaho, SpagoBI, Supermetrics, AnswerRocket, Sisense, Flexmonster, OneStream XF,...
Složenost domene primjene	• Alat za menadžment zahtjeva • Alat za modeliranje • Alat za provedbu automatiziranih testova	• Visure, Visual Trace, Confluence, Process Street, SpiraTeam, IBM Rational DOORS, IRIS Business Architect, ... • Lucidchart, Draw.io, Creately, Gliffy, Visio, MyDraw, ... • Isto kao ***Sukladnost sa standardima i regulativama,*** *točka 2.*
Tehnička složenost	• Integrirana razvojna okolina • Alat za kodiranje • Alati za analizu koda • Alati za pronalaženje bug-ova • Alat za automatizirane regresijske testove	• Microsoft Visual Studio, Eclipse, Xcode, IntelliJ IDEA, Android Studio, WebStorm, Aptana Studio 3, RubyMine, ... • Atom, Brackets, Vim, Sublime Text, Visual Studio Code,... • Veracode, Coverity Code Compare, SonarQube, ReSharper C++, Gamma, ... • Bugzilla, Mantis, Redmine, HP ALM, Lighthouse, Zoho bug tracker, Lean Testing, Marker.io, Sentry, ... • SahiPro, Selenium, Watir, TestComplete, IBM Rational Functional Tester, TimeShiftX, TestDrive,...

(Izvor: Izrada autora)

Neki komercijalno dostupni alati za konkretne metode prikazani su Tablicom na kojoj je također vidljivo da se neki alati (VersionOne, Targetprocess, ProjectView, SpiraPlan, Pivotal Tracker, ExtremePlanner) svrstavaju u podršku za više metoda što

znači da je to reprezentativno kada se želi prikazati usporedba tih metoda po dostupnim alatima.

Tablica 8. Dostupni alati po metodama

Metoda	*Alati*
Scrum	Scrumblr, Trello, ScrumDesk, Agilefant, SpiraPlan, VersionOne, ExtremePlanner, Targetprocess, Project Planning and Tracking System, ProjectView
FDD	CASE Spec, TechExcel DevSuite, FDD Tools, FDD Viewer
Lean	Kanban Tool, LeanKit, Lean-Case, VersionOne, KanbanFlow, Targetprocess
DSDM	SpiraPlan, VersionOne, ProjectView,
Ekstremno Programiranje	SpiraPlan, VersionOne, ExtremePlanner, Targetprocess, Project Planning and Tracking System, Enonic, Pivotal Tracker

(Izvor: Izrada autora)

S obzirom na to da se u tablici usporedbe metoda po praksama već može pokazati kako neke metode imaju zajedničke prakse, bilo je logično zaključiti da će alati koji podržavaju te prakse biti multipraktični za primjenu kod više agilnih metoda.

11.6. Izbor metode

Pojedine metode svojim prednostima predstavljaju primjereniji pristup nekim projektima IS-a dok su neki eliminacijski elementi ili problemi kojima će se trebati posvetiti otegotna karakteristika. Prednosti i nedostatke može se utvrditi kroz jednostavan pregled po metodama. Tabličnim prikazom navedeni su neke opće prednosti i nedostaci koji su posljedica pojedinih praksi i pristupa.

Tablica 9. Prednosti i nedostaci metodologija/metoda

	PREDNOSTI	**NEDOSTACI**
TRADICIONALNE METODOLOGIJE		
Model Vodopada	• Jednostavnost usvajanja, • Jednostavnost menadžmenta, • Jasno definirani tijek izvođenja faza, • Definirani rokovi • Pokrivenost dokumentacijom	• Troškovno rizičan, • Vremenski rizičan, • Otpornost prema promjenama, • Funkcionalni kod kasno u • ciklusu, • Pretpostavlja preciznost u

	• Jednostavnost testiranja	• utvrđenim zahtjevima u početku. • Ušteda na vremenu na račun testiranja
V Model	• Jednostavnost usvajanja, • Jednostavnost menadžmenta, • Jasno definirani tijek izvođenja faza, • Definirani rokovi • Pokrivenost dokumentacijom • Jednostavnost testiranja	• Troškovno rizičan, • Vremenski rizičan, • Otpornost prema promjenama, • Funkcionalni kod kasno u ciklusu, • Zahtjevi moraju biti točno definirani odmah u početku, • Trošak testiranja
Prototipiranje	• Rane povratne informacije korisnika, • Veća uključenost korisnika • Sukladnost sa zahtjevima, • Potencijalno smanjenje utroška vremena i novaca	• Rizik od nepotpuno provedene analize zbog fokusa na prototip • Mnogo prototipa može otegnuti vrijeme i povećati trošak • Sustav može dobiti na složenosti
Spiralni razvoj	• Mogućnost promjena, • Koncept rješenja dostupan rano, • Lakše točnije utvrditi zahtjeve, • Bolji menadžment rizika • Primjenjiv kod većih i složenijih projekata	• Složeniji menadžment, • Složeni proces, • Troškovna rizičnost, • Uspjeh cijelog projekta se temelji na dobro provedenoj analizi rizika
BRZI RAZVOJ APLIKACIJA	• Mogućnost promjena, • Ušteda vremena, • Mjerljivost, • Korištenje RAD alata, • Dostupnost povratnih informacija korisnika, • Rana integracija	• Zahtjeva tehničke vještine, • Zahtjeva visoke vještine modeliranja, • Trošak može biti visok, • Složenost menadžmenta, • Zahtjeva čestu dostupnost korisnika
GRUPNI RAZVOJ APLIKACIJA	• Smanjuje resurse potrebne za utvrđivanje zahtjeva, • Suradnja, • Definirane uloge i odgovornosti, • Uključenost korisnika, • Omogućuje simultano prikupljanje i konsolidiranje velike količine informacija	• Zahtjeva dobru organizaciju, pripremu i uvid u problematiku, • Zahtjeva posvećenost vremena i truda korisnika, • Značajna količina vremena utrošena u planiranje i donošenje rasporeda
OBJEKTNO ORIJENTIRANI PRISTUP	• Kontinuirana Integracija, • Ušteda vremena, • Modulirani funkcionalni kod • Lakše modeliranje	• Složenost procesa,

	• Lakše održavanje • Ušteda vremena i novca • Ponovna iskoristivost	• Zahtjevi za vještinama
AGILNE METODE		
Kristalne metode	• Spektar metodologija s obzirom na veličinu i kritičnost projekta, • Iscrpno testiranje, • Kvaliteta • Ušteda vremena • Kontinuirana integracija • Malo dokumentacije • Kontinuirani feedback • Kontinuirano testiranje • Rana funkcionalnost	• Prijelaz iz jedne metodologije spektra u drugu ne funkcionira • /
Dinamička metoda razvoja sustava	• Fleksibilan razvoj zahtjeva • Pridržavanje rokovima i proračunu • Uključenost korisnika • Težište na testiranju • Prioretizacija poslovnih vrijednosti (najbitnije funkcionalnosti rješavane prve)	• Zahtjeva tehničke i poslovne vještine tima • Zahtjeva čestu dostupnost korisnika • Više dokumentacije • Usmjerenost na RAD može utjecati na robusnost koda
Scrum	• Transparentnost, • Timski rad, • Fleksibilnost, • Veliki projekti su podjeljeni u manje sprinteve, • Kontinuirani feedback, • Otvorenost promjenama, • Ne zahtjeva mnogo dokumentacije	• Neizvjestan rok, • Zahtjevnost vještina, • Primjena Scruma s velikim timovima može predstavljati izazov, • Ukoliko jedan član tima ode to može biti štetno • Nema određenog roka i troška
Feature Driven Development	• Konformnost sa standardima i najboljim praksama • Prikladan za velike projekte • Kvaliteta • Ušteda vremena • Kontinuirani feedback • Kontinuirano testiranje	• Korisnici nemaju dokumentaciju na raspolaganju • Ovisnost o glavnom programeru koji djeluje kao koordinator, vodeći dizajner i mentor
Ekstremno Programiranje	• Težište na testiranju • Ušteda vremena • Kontinuirana integracija • Mogućnost kombiniranja s drugim metodologijama • Malo dokumentacije • Kontinuirani feedback • Kontinuirano testiranje • Rana funkcionalnost	• Visoka usredotočenost na kod – mogućnost zanemarivanja dizajna • Potreba za vještinama • Manjak dokumentacije može onemogućiti refakturiranje koda. • Testiranje može biti skupocjeno • Dislocirani timovi nisu idealni • Zahtjeva disciplinu • Teško utvrđivanje resursa i rokova • Nije prikladan za velike projekte
Lean Development	• Manje „praznog hoda“,	

	• Manji trošak vremena i resursa, • Uključenost korisnika, • Potpora kreiranju znanja i učenju, • Team building i odgovornost odlučivanja • Brza funkcionalna rješenja	• Zahtjevnost vještina • Zahtjevnost timske discipline i kohezije

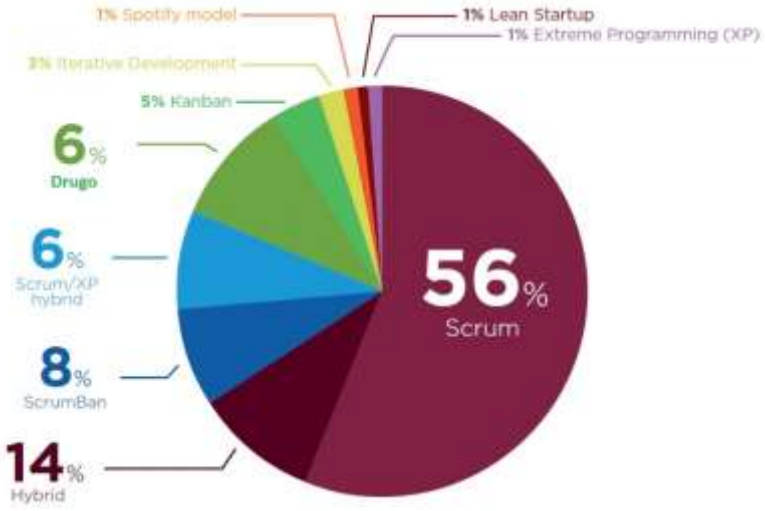
(Izvor: Izrada autora)

Provedeno istraživanje VersionOne kompanije pod nazivom „The 12th annual State of Agile“ (VersionOne https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report , datum pristupa: 18.06.2018.) pruža uvid u percipirane prednosti i prepreke korištenja metoda/metodologija kod ispitanika. Izvješćem su izneseni podaci dobiveni na temelju iskustva 1942 sudionika iz poduzeća diljem svijeta koja djeluju na polju primjene metoda, od čega najviše na području softverskih kompanija ali i iz drugih područja poput financijskog sektora, zdravstva, vlade, teleoperatera itd. Od navedenog ukupnog broja sudionika, 55% djeluje na području Sjeverne Amerike, 27% na području Europe, 711% iz Azije, 7% iz Južne Amerike, 3% iz Australije i Novog Zelanda i 1% iz Afrike. Višestrukim odabirom dobivena je sljedeća distribucija:

1. Što se postiglo usvajanjem agilnih metoda:
   - Poboljšanje sposobnosti upravljanja promjenama prioriteta – 71%
   - Poboljšanje transparentnosti projekta – 66%
   - Poboljšanje usklađenosti poslovanja i informacijske tehnologije – 65%
   - Poboljšanje produktivnosti timova – 61%
   - Poboljšanje morala timova – 61%
   - Poboljšanje predvidljivosti perioda potrebnog za izručivanje proizvoda – 49%
   - Poboljšanje kvalitete softvera – 47%
   - Smanjenje rizičnosti projekta – 47%
   - Bolje upravljanje dislociranim timovima – 40%
   - Poboljšanje održivosti softvera – 33%
   - Smanjenje troškova projekta – 22%
2. Prepreke pri usvajanju agilnih metoda:
   - Neusklađenost organizacijske kulture s agilnim vrijednostima – 53%
   - Nedostatak potpore menadžmenta i sponzorstva – 42%
   - Otpor organizacije na promjene – 53%
   - Nedostatak vještina/iskustva u agilnim metodologijama – 39%

- Nedostatak prakse i edukacije – 35%
- Nekonzistentni procesi i prakse među timovima – 34%
- Nedostupnost korisnika – 31%
- Nepovezanost projektnih alata i podataka/mjerenja – 24%
- Minimalna suradnja i dijeljenje znanja – 21%
- Neusklađenost s regulativama ili konflikt s državnim odredbama – 14%

Po pitanju učestalosti korištenja pojedinih metoda iz istog istraživanja vidljivo je iz pregleda presjeka da je najčešće korištena metoda, s čak 56% - Scrum, zatim slijede hibridna s 14%, ScrumBan sa 8%, Hibrid Scruma i Ekstremnog programiranja sa 6%, Drugo s 6%, Kanban s 5%, Iterativni razvoj s 3%, te po 1% imaju Spotify model, Lean razvoj i Ekstremno programiranje.



Slika 63. Distribucija primjene metodologija;
(Izvor: https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report , datum pristupa: 07.11.2018.)

Što se tiče problematike izbora metode razvoja nekog sustava, može se povesti za određenim specifičnostima/praksama koje su svojstvene svakom procesu i iznaći ih u prikazanim tablicama kako bi se ustanovilo kojoj metodi aproksimativno najviše odgovaraju. Tako, ako se uzme u obzir npr. jasnoća korisničkih zahtjeva, poznavanje tehnologije, kompleksnost sustava, pouzdanost sustava, raspoloživost vremena i vidljivost napredovanja kroz raspored, i ustvrdi da postoji npr. sljedeći slučaj potreba:

- Jasnoća korisničkih zahtjeva - **niska**
- Poznavanje tehnologije - **dobro**
- Kompleksnost sustava - **visoka**
- Pouzdanost sustava - **visoka**

- Raspoloživost vremena – **dugotrajna alokacija vremena razvoja i vremena korisnika**
- Vidljivost napredovanja kroz raspored - **bitna**

Koristeći se tabličnom usporedbom praksi, s obzirom na primjenjivo, može se ustanoviti kako potrebama možda najbolje odgovara agilna **Scrum** metoda jer:

- ✓ Scrum podržava promjenjive zahtjeve pa se nejasni zahtjevi kompenziraju s vremenom.
- ✓ Scrum zahtjeva vješte timove
- ✓ Scrum je pogodan za razvoj složenih sustava
- ✓ Scrum je pogodan za razvoj pouzdanih sustava kroz kontinuirani, feedback, testiranje i integraciju
- ✓ Scrum je prikladan za razvoj velikih sustava/Scrum se izvodi uz kontinuiranu uključenost korisnika
- ✓ Scrum koristi Timebox – alat za praćenje napretka, scrum sastanke, planiranje sprinteva...

Navedeni primjer je indikativan tome kako provedena usporedba metoda može poslužiti kao pomoć pri izboru metode.

11.7. Uspješnost projekata IS-a

Istraživanje Standish Grupe (savjetničke kompanije koja provodi nezavisna međunarodna istraživanja na području informacijske tehnologije) iz 2015. godine (Standish Group, 2015.) na temu uspjeha projekata informacijskih sustava u javnom i privatnom sektoru sadrži nekoliko bitnih pokazatelja relevantnih za ovaj rad.

Prvi pokazatelj je statistika o općenitoj uspješnosti projekata informacijskih sustava.

	2011	2012	2013	2014	2015
USPJEŠNI	29%	27%	31%	28%	29%
PROBLEMATIČNI	49%	56%	50%	55%	52%
PROPALI	22%	17%	19%	17%	19%

Slika 64. Uspješnost projekata u razdoblju 2011. do 2015.;
Izvor: https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf, datum pristupa: 16.11.2018.)

Navedena statistika se odnosi na razdoblje od 2011. do 2015. i dijeli projekte na tri kategorije uspješnosti s obzirom na ispunjavanje tri uvjeta. Tri uvjeta jesu: rok, trošak i pokrivenost domene primjene, a kategorije uspješnosti jesu:

- Uspješni projekti – uspješni projekt podrazumijevaju projekte kojim su ispunjena sva tri uvjeta
- Problematični projekti – projekti kojim su ispunjena dva od tri uvjeta
- Propali projekti – Projekti koji su otkazani ili dovršeni ali neiskorišteni.

Prema tom istraživanju u tom razdoblju je bilo u prosjeku **28,8% uspješnih, 52,4% problematičnih i 18,8% propalih projekata**.

Zatim su na istraživanje provedeno za razdoblje između 2013. do 2017. godine (Standish Group 2, 2018.) primjenjena još tri uvjeta: ostvarivanje korisničke vrijednosti i zadovoljstva korisnika te sukladnost sa strateškim ciljevima. S obzirom na na istraživanje za to razdoblje, uz ukupno šest uvjeta, prosječna uspješnost projekata je također podijeljena s obzirom na to jesu li primjenjene agilne ili tradicionalne metodologije.



Slika 65. Uspješnost projekata po primjeni metoda;
(Izvor: https://vitalitychicago.com/blog/agile-projects-are-more-successful-traditional-projects/, datum pristupa: 16.11.2018.)

Istraživanje je pokazalo da su:

- Agilne metode imale 42% uspješnih, 50% problematičnih i 8% propalih projekata, dok su
- Tradicionalne metodologije imale 26% uspješnih, 53% problematičnih i 21% propalih projekata.

Navedeno vodi do zaključka kako su se agilne metode u tom razdoblju pokazale 16% uspješnijima, 3% manje problematičnima uz 13% manje propalih projekata od tradicionalnih metodologija.

S obzirom na složenost projekta u rasponu od vrlo jednostavnih do vrlo složenih za razdoblje od 2011. do 2015., distribucija uspješnosti je prikazana na slici 66.

	USPJEŠNI	PROBLEMATIČNI	PROPALI
Vrlo složeni	15%	57%	28%
Složeni	18%	56%	26%
Srednje složeni	28%	54%	18%
Jednostavni	35%	49%	16%
Vrlo jednostavni	38%	47%	15%

Slika 66. Uspješnost projekata po složenosti projekta;
(Izvor: https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf , datum pristupa: 16.11.2018.)

Podaci za to razdoblje po složenosti projekta pokazuju da su:

- Vrlo jednostavni projekti bili 38% uspješni ili 47% problematični dok ih je 15% propalo
- Jednostavni projekti bili 35% uspješni ili 49% problematični dok ih je 16% propalo
- Srednje složeni projekti bili 28% uspješni ili 54% problematični dok ih je 18% propalo
- Složeni projekti bili 18% uspješni ili 56% problematični dok ih je 26% propalo
- Vrlo složeni projekti bili 15% uspješni ili 57% problematični dok ih je 28% propalo

Navedena statistika pokazuje progresivni pad uspješnosti i porast propalih projekata kako projekti dobivaju na složenosti. Također je primjetno najznačajnije povećanje jaza između srednje složenih projekata i složenih projekata.

S obzirom na veličinu projekata i korištene metode, uspješnost projekata za razdoblje od 2011. do 2015. je prikazana slikom 67.

VELIČINA	METODE	USPJEŠNI	PROBLEMATIČNI	PROPALI
Veliki projekti	Agilne	18%	59%	23%
	Tradicionalne	3%	55%	42%
Srednji projekti	Agilne	27%	62%	11%
	Tradicionalne	7%	68%	25%
Mali projekti	Agilne	58%	38%	4%
	Tradicionalne	44%	45%	11%

Slika 67. Uspješnost metodoloških primjena u odnosu na veličinu projekata;
Izvor: https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf , datum pristupa: 16.11.2018.)

Istraživanje je pokazalo da su:

Agilne metode

- Pri velikim projektima imale: 18% uspješnih, 59% problematičnih i 23% propalih projekata
- Pri srednjim projektima: 27% uspješnih, 62% problematičnih i 11% propalih projekata
- Pri malim projektima: 58% uspješnih, 38% problematičnih i 4% propalih projekata.

Tradicionalne metode

- Pri velikim projektima imale: 3% uspješnih, 55% problematičnih i 42% propalih projekata
- Pri srednjim projektima: 7% uspješnih, 68% problematičnih i 25% propalih projekata
- Pri malim projektima: 44% uspješnih, 45% problematičnih i 11% propalih projekata.

Navedeno vodi do zaključka kako su se agilne metode u tom razdoblju u odnosu na tradicionalne metodologije pokazale:

- Za velike projekte: 15% uspješnijima uz 4% više problematičnih projekata i 19% manje propalih,
- Za srednje projekte: 20% uspješnijima uz 6% manje problematičnih i 14% manje propalih projekata,
- Za male projekte: 14% uspješnijima uz 7% manje problematičnih i 7% manje propalih projekata.

Istraživanje navodi na zaključak da je primjena agilnih metoda u nedavnom razdoblju povoljnije utjecala na uspješnost projekata nego primjena tradicionalnih metoda iako treba imati na umu da metoda sama po sebi nije garancija uspjeha.

12. ZAKLJUČAK

Suvremene metode predstavljaju kronološki i principijalni odmak od tradicionalnih metodologija. Nastale su iz raznovrsnih potreba, pokrivaju poneke različite aktivnosti, prioritete, prakse itd. ali su također nastale na zajedničkim vrijednostima i principima navedenima u Manifestu agilnog razvoja softvera, ipak primjenjuju i niz zajedničkih praksi, pokrivaju neke zajedničke faze i odlikuju se nekim zajedničkim prednostima i nedostacima kao što se moglo vidjeti kroz analizu i usporedbu. Opravdano je, dakle, da zajedno spadaju u kategoriju zajedničkog naziva.

Ni jedna metodologija ili metoda ne nudi recept za siguran uspjeh – on, naravno ovisi o brojnim faktorima izvedbe od kojih su mnogi uvršteni u analitički okvir rada bilo kao resursi, faze, prakse ili alati. Prije izbora idealne metodologije za razvoj nekog sustava najbolje prvo ostvariti dobro poznavanje vlastite tehnološke, poslovne i organizacijske okoline, funkcionalnih potreba i potreba za resursima kako vremenskim i troškovnim tako i ljudskim. Tek kad je poznato čime se raspolaže može se znati koje potrebe nastaju. Ako postoji poznavanje početnog stanja i potreba pri razvoju sustava te dobra procjena izvedivosti moguće je odrediti prakse koje je potrebno primjeniti.

Hipoteza ovog rada - da je usporedba suvremenih metoda moguća, pri čemu se podrazumijevaju teze da agilne metode dijele neke zajedničke karakteristike u odabranim relevantnim odrednicama koje čine podrijetlo, resurse, faze, prakse i alate, smatra se zadovoljenom, jer je kroz istraživački dio komparativnom analizom pokazano da je:

- ✓ Moguća usporedba agilnih metoda po podrijetlu
- ✓ Moguća usporedba agilnih metoda po resursima
- ✓ Moguća usporedba agilnih metoda po fazama
- ✓ Moguća usporedba agilnih metoda po praksama
- ✓ Moguća usporedba agilnih metoda po alatima

Komparativna analiza, kao metoda rada, izvršena je tabličnom usporedbom s uvrštavanjem relevantnih podataka pojedinih metodologija ili metoda po odrednicama, pri čemu su uspješno ustanovljene zajedničke karakteristike metoda.

Cilj rada je postignut stoga što je komparativnom analizom pružen praktični i pojednostavljeni pregled bitnih karakteristika metoda koje su relevantne pri odabiru metode s obzirom na različite potrebe koje spadaju u odabrane odrednice. Dodatno je

pružen uvid u relevantne statističke podatke koji govore o učestalosti korištenja metoda, prednostima i nedostacima te pokazateljima uspješnosti.

13. LITERATURA

KNJIGE

- Aström, K.J. i Murray, R.M. (2009). Feedback Systems: An Introduction for Scientists and Engineers. Princeton, UK: Princeton University Press. Str. 1. URL:http://www.cds.caltech.edu/~murray/books/AM05/pdf/am08-complete_22Feb09.pdf (datum pristupa: 13.02.2018.)

- Avison, D. E. i Fitzgerald, G. (2006). Methodologies for Developing Information Systems: A Historical Perspective – u: The Past and Future of Information Systems: 1976–2006 and Beyond: IFIP 19th World Computer Congress, TC-8, Information System Stream, August 21–23, 2006, Santiago, Chile. Str. 27. URL: https://www.researchgate.net/publication/227101490_Methodologies_for_Developing_Information_Systems_A_Historical_Perspective (datum pristupa: 23.05.2018.)

- Baianu, I.C. (Ed.) (2011). Complexity, emergent systems and complex biological systems: Complex systems theory and biodynamics. URL: http://cogprints.org/7736/1/BiodynamicsEtComplexity231pMb21.pdf (datum pristupa: 13.02.2018.)

- Bocij, P., Greasley, A. i Hickie S. (2003). Business Information Systems: Technology, Development and Management. New Jersey: Pearson Education. Str. 8.

- Booch, G., Rumbaugh, J. i Jacobson, I. (1998). The Unified Modeling Language User Guide, 1. izdanje. Massachusetts: Addison Wesley. Str. 89. URL: http://www.icst.pku.edu.cn/course/uml/reference/the-unified-modeling-language-user-guide.9780201571684.997.pdf (datum pristupa: 29.05.2018.)

- Cadle, J. (Edt.) (2014). Developing information systems: Practical guidance for IT professionals. Swindon, UK: BCS Learning & Development Ltd. Str. 94, 22,

24, 90.

- Cornford, T. i Shaikh, M. (2013). Introduction to information systems. London, UK: University of London.
  URL: http://www.dphu.org/uploads/attachements/books/books_3326_0.pdf (datum pristupa: 13.02.2018.)

- Dennis, A., Wixom, B.H. i Roth, R.M. (2011). Systems analysis and design, 5. izdanje. John Hoboken, NJ: Wiley & Sons, Inc. Str. 246, 8, 119, 54. URL: http://www.uoitc.edu.iq/images/documents/informatics-institute/Competitive_exam/Systemanalysisanddesign.pdf (datum pristupa: 26.05.2018.)

- Haag, S.Cummings, M. (2007). Management Information Systems for the Information Age, 7. izdanje. Boston, MA: Mcgraw Hill.

- Harsh, S. B., Connor, L. J. i Schwab, G. D. (1981). Managing The Farm Business. Prentice-Hall, Inc., Englewood Cliffs, New Jersey. Str. 14.

- Highsmith, J. (2002). Agile Software Development Ecosystems. Addison Wesley. Str. 173.

- Laudon, K.C. i Laudon, J.P. (2014). Management Information Systems: Managing the Digital Firm, Global Edition, 13. izdanje. Essex, England: Pearson Education Limited. Str. 85, 82. URL: http://www.icto.info/laudon-management-information-systems-13th-global-edition-c2014-1.pdf (datum pristupa: 05.03.2018.)

- Loucopoulos, P. i Karakostas, V. (1995). System Requirements Engineering. McGraw Hill. Str. Pog. 6, str. 6. URL: https://www.researchgate.net/publication/233819114_System_Requirements_Engineering (datum pristupa: 14.10.2018.)

- Nonaka, I. i Takeuchi, H. (1995). The Knowledge-creating Company: How Japanese Companies Create the Dynamics of Innovation. Oxford University Press.

- Oz, E. (2008). Management information systems, 6. izdanje. Course Technology. Str. 293, 271.

- Pender, T.A. (edt.) (2002). UML Weekend Crash Course, New York, NY: Wiley Publishing, Inc. Str. 6, 56. URL: http://www.netmatch.ro/contest2011/docs/UML_Weekend_Crash_Course.pdf (datum pristupa: 29.05.2018.)

- Poppendieck, M. i Poppendieck, T. (2003) Lean Software Development: An Agile Toolkit. Addison Wesley. URL: http://200.17.137.109:8081/novobsi/Members/teresa/optativa-fabrica-de-sw-organizacoes-ageis/artigos/Addison%20Wesley%20-%20Lean%20Software%20Development%20-%20An%20Agile%20Toolkit.pdf (datum pristupa: 17.06.2018.)

- Pressman, R. S. (2010). Software engineering: a practitioner's approach, 7. izdanje. McGraw-Hill. Str. 84, 86, 276. URL: http://www.vumultan.com/Books/CS605-Software%20Engineering%20Practitioner's%20Approach%20%20by%20Roger%20S.%20Pressman%20.pdf (datum pristupa: 15.06.2017.)

- Radovan, M. (1992). Projektiranje informacijskih sustava, Zagreb: Informator.

- Rainer, R.K., Prince, B. i Cegielski, C. (2014). Introduction to Information Systems Supporting and Transforming Business, 5. izdanje. New Jersey: Pearson Education. Str. 42, 491.

- Riccardi, G. (2003). Database management with web site development applications. Boston, MA: Pearson Education, Inc. Str. 61. Izvor:

http://www.aw-bc.com/info/riccardi/database/Riccardi_ch4.PDF (datum pristupa: 27.05.2018.)

- Satzinger, J.W., Jackson, R.B. i Burd, S.D. (2009). Systems Analysis and Design in a Changing World, 5. izdanje. Boston, USA: Cengage Learning. Str. 43.

- Shelly, G.B. i Rosenblatt, H.J. (2012). Systems Analysis and Design, 9. izdanje. Boston, USA: Cengage Learning, Shelly Cashman Series. Str. 15, 18. URL: http://160592857366.free.fr/joe/ebooks/ShareData/System%20Analysis%20and%20Design%209e%20-%20Shelly%20Cashman.pdf (datum pristupa: 06.03.2018.)

- Shelly, G.B. i Vermaat, M.E. (2012). Discovering Computers - Fundamentals:Your Interactive Guide to the Digital World. Boston, MA: Cengage Learning. Str. 473. URL: https://www.cengage.com/custom/static_content/OLC/1133349900/data/shelly30327_1111530327_02.05_chapter05.pdf (datum pristupa: 06.03.2018.)

- Sommerville, I. (2011). Software engineering, 9. izdanje. Addison-Wesley/Pearson Education, Inc. Str. 58. URL: https://edisciplinas.usp.br/pluginfile.php/2150022/mod_resource/content/1/1429431793.203Software%20Engineering%20by%20Somerville.pdf (datum pristupa: 24.05.2018.)

- Stair, R., Reynolds, G. (2010). Principles of Information Systems A Managerial Approach, 9. izdanje. Boston, MA: Course Technology. Str. 5, 363, 442. URL: https://drive.uqu.edu.sa/_/fbshareef/files/principles%20of%20information%20systems%209th%20-stair,%20reynolds.pdf (datum pristupa: 13.02.2017.)

- Stair, R., Reynolds, G. i Chesney, T. (2012). Fundamentals of Business Information Systems, 2. izdanje. Hampshire, Engleska: Cengage Learning. Str. 7.

- Turban, E. (2003). Introduction to information technology, 2. izdanje. New York, NY: Wiley & Sons, Inc. Str. 459.

- Valacich, J. i Schneider, C. (2010). Information Systems Today: Managing in the Digital World, 4. izdanje. NJ, SAD: Pearson. Inc. Str. 11.

- Valacich, J.S., George, J.F. i Hoffer, J.A. (2011). Essentials of systems analysis and design, 5. izdanje. New Jersey: Pearson Education. Str. 6, 159, 362, 367. URL: https://www.bau.edu.jo/inst/hamza/SAbook.pdf (datum pristupa: 13.02.2018.)

- Whitten, J.L. i Bentley, L.D. (2007). Systems analysis and design methods, 7. izdanje. New York, NY: McGraw-Hill/Irwin. Str.7.

ČLANCI

- Ackoff, R. L. (1989). From data to wisdom. Journal of Applied Systems Analysis, 16, 3-9. URL: http://faculty.ung.edu/kmelton/documents/datawisdom.pdf (datum pristupa: 13. 02. 2017.)

- Beck, K. (1999). Embracing Change with Extreme Programming. IEEE Computer 32, str 70-77. URL: http://users.ece.utexas.edu/~perry/education/382v-s06/beck_xp.pdf

- Boehm, B. W. (1979). Guidelines for verifying and validating software requirements and design specifications, in Euro IFIP 79, P. A. Samet, Ed. North Holland. Str. 711–719. URL: http://csse.usc.edu/TECHRPTS/1979/usccse79-501/usccse79-501.pdf (datum pristupa: 24.05.2018.)

- Boehm, B. (1986). "A Spiral Model of Software Development and Enhancement," Proc. Int'l Workshop Software Process and Software Environments, ACM Press, 1985; also in ACM Software Eng. Notes, Aug. 1986

Str.. 22-42. URL: http://csse.usc.edu/TECHRPTS/1988/usccse88-500/usccse88-500.pdf (datum pristupa: 25.05.2018.)

- Ceri, S. et. al. (2000). Web Modeling Language (WebML): a modeling language for designing Web sites. Computer Networks 33 (2000). Str. 137–157. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.472.4706&rep=rep1&type=pdf (datum pristupa: 04.06.2018.)

- Costagliola, G., Ferrucci, F. i Francese, R. (2002). Web Engineering: Models and Methodologies for the Design of Hypermedia Applications. Handbook of Software Engineering & Knowledge Engineering, volume 2, Emerging Technologies. Str. 181 – 199. URL: http://ldc.usb.ve/~abianc/electivas/WebEngineering.pdf (datum pristupa: 18.06.2018.)

- Crnkovic, I., Larsson, S. i Chaudron, M. (2005). Component-based Development Process and Component Lifecycle. Journal of Computing and Information Technology - CIT 13, 2005, 4, 321-327. URL: http://www.mrtc.mdh.se/publications/0953.pdf (datum pristupa: 05.06.2018.)

- Edeki, C. (2013). Agile Unified Process International Journal of Computer Science and Mobile Applications, Vol.1 Issue. 3, September- 2013. Str. 13-17. URL: http://www.ijcsma.com/publications/september2013/V1I304.pdf (datum pristupa: 21.07.2018.)

- Fuggetta, A. (1993). A classification of CASE technology. *IEEE Computer,* **26** (12), 25-38.

- Grenning, J.W. (2012). Agile Requirements, Estimation and Planning - Iteration Zero. Boston, MA: Embedded Systems Conference. URL: https://wingman-sw.com/papers/Iteration0-Grenning-v1r0.pages.pdf (datum pristupa: 13.06.2018.)

- Liew, A. (2007). DIKIW: Data, information, knowledge, and their interrelationships. Business Management Dynamics Vol.2, No.10, 04.2013., pp.49-62. str. 1. URL: http://bmdynamics.com/issue_pdf/bmd110349-%2049-62.pdf (datum pristupa: 13.02.2017.)

- Nonaka, I. i Takeuchi, H. (1986). The New New Product Development Game. Harvard Business Review Siječanj/Veljača. str. 137-146. URL: https://www.pomsmeetings.org/ConfEvents/043/ProductDevelopment00011.pdf (datum pristupa: 16.06.2018.)

- Palmer, S. (2009). An Introduction to Feature-Driven Development. URL: https://dzone.com/articles/introduction-feature-driven (datum pristupa: 14.06.2018.)

- Rowley, J. (2007). The wisdom hierarchy: representations of the DIKW hierarchy Journal of Information Science 2007; 33; 163. SAGE Publications. URL: http://inls151f14.web.unc.edu/files/2014/08/rowleydikw.pdf (datum pristupa: 13.02.2017.)

- Royce, W. (1970). Managing the Development of Large Software Systems. Proc. Westcon, IEEE CS Press. Str. 328-339. Url: https://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf (datum pristupa: 24.05.2018.)

- Sunassee, N. i Sewry, D. (2002). A Theoretical Framework for Knowledge Management Implementation, Proceedings of SAICSIT '02, Str. 235 - 245. Južnoafrička Republika: South African Institute for Computer Scientists and Information Technologists. Str. 236. URL: https://pdfs.semanticscholar.org/2413/36c7817a5e90952bccac40771950cf56c804.pdf (datum pristupa: 13.02.2017.)

- Voigt, B.J.J. (2004). Dynamic System Development Method. Department of Information Technology University of Zurich. URL:

https://files.ifi.uzh.ch/rerg/arvo/courses/seminar_ws03/14_Voigt_DSMD_Ausarbeitung.pdf (datum pristupa: 16.06.2017.)

WEB STRANICE

- Activity diagram (n.d.). URL: https://www.lucidchart.com/pages/uml/activity-diagram (datum pristupa: 11.09.2018.)

- Agilni Manifest (2002.). URL: https://agilemanifesto.org/principles.html (datum pristupa: 13.06.2017)

- Anderson, D.J. (2012). Lean Software Development White Paper. URL: https://msdn.microsoft.com/en-us/library/hh533841(v=vs.120).aspx (datum pristupa: 18.06.2018.)

- Brambilla et al (n.d.). Designing Web Applicationes with WebMl and WebRatio. URL: http://www.csun.edu/~twang/595WEB/Slides/WebML.pdf (datum pristupa: 03.06.2018.)

- Coffin, R. i Lane, D. (2006). A Practical Guide to Seven Agile Methodologies, Part 2: Page 2. DevX: The know-how behind application development. URL: http://www.devx.com/architect/Article/32836/0/page/2 (datum pristupa: 13.06.2017.)

- DA Toolkit. URL: http://www.disciplinedagiledelivery.com/tool-support-for-dad/ (datum pristupa: 07.11.2018.)

- DSDM Consortium 1 (n.d.). The DSDM Agile Project Framework: Philosophy and Fundamentals. URL: https://www.dsdm.org/content/philosophy-and-fundamentals (datum pristupa: 16.06.2017.)

- DSDM Consortium 2 (n.d.). The DSDM Agile Project Framework: Principles.

URL: https://www.dsdm.org/content/principles (datum pristupa: 16.06.2017.)

- DSDM Consortium 3 (n.d.). The DSDM Agile Project Framework: Process. URL: https://www.dsdm.org/content/process (datum pristupa: 16.06.2017.)

- DSDM Consortium 4 (n.d.). The DSDM Agile Project Framework: Timeboxing. URL: https://www.dsdm.org/content/timeboxing (datum pristupa: 16.06.2017.)

- Emergentna svojstva (n.d.) iz: Business Dictionary. URL: http://www.businessdictionary.com/definition/emergent.html (datum pristupa: 14.02.2017.)

- HP Izvor: https://techbeacon.com/sites/default/files/gated_asset/agile-projects-are-more-successful-than-hybrid-projects.pdf, (datum pristupa: 14. 09. 2018.)

- Information systems development (n.d.). URL: https://www.wiley.com/college/turban/0471073806/sc/ch14.pdf (datum pristupa: 08.09.2018.)

- JAD Guidelines (n.d.). URL: http://www.ksinc.com/itpmcptools/JADGuidelines.pdf (datum pristupa: 07.06.2017.)

- Jones, W. URL: http://www.umsl.edu/~sauterv/analysis/488_f02_papers/CASE.html (datum pristupa: 06.11.2018.)

- Lucidchart 1 (n.d.) : All About Data Flow Diagrams (DFDs); URL: https://www.lucidchart.com/pages/data-flow-diagram (datum pristupa: 09. 07. 2018.)

- Lucidchart 2 (n.d.) : All About ER Diagrams; URL: https://www.lucidchart.com/pages/er-diagrams?a=0 (datum pristupa: 09. 07. 2018.)

- Lucidchart State Machine (n.d.). URL: https://www.lucidchart.com/pages/uml/state-machine-diagram (datum pristupa: 10.09.2018.)

- RAD lc development (n.d.). URL: http://www.ftms.edu.my/images/Document/IMM006%20-%20RAPID%20APPLICATION%20DEVELOPMENT/Chapter%202nnote.pdf (datum pristupa: 10.06.2017)

- RUP IBM (n.d.). URL: https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf (datum pristupa: 07.06.2017.)

- SBS Inc. What is Computer Aided Software Engineering? (CASE). Select Business Solutions, Inc. URL: http://www.selectbs.com/analysis-and-design/what-is-computer-aided-software-engineering (datum pristupa: 14.10.2018.)

- Scrum Alliance (n.d.). Learn About Scrum. URL: https://www.scrumalliance.org/why-scrum (datum pristupa: 17.06.2018.)

- Sequence Diagrams Intro. (n.d.). URL: http://www.tracemodeler.com/articles/a_quick_introduction_to_uml_sequence_diagrams/ (datum pristupa: 10.09.2018.)

- Sequence Diagrams (n.d.). URL: https://www.uml-diagrams.org/sequence-diagrams.html (datum pristupa: 10.09.2018.)

- Standish Group (2015). Chaos report 2015. URL: https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf (datum pristupa: 16.11.2018.)

- Standish Group 2 (2018). URL: https://vitalitychicago.com/blog/agile-projects-are-more-successful-traditional-projects/ (datum pristupa: 16.11.2018.)

- Sutherland, J. (2011). Takeuchi and Nonaka: The Roots of Scrum. Scruminc blog. URL: https://www.scruminc.com/takeuchi-and-nonaka-roots-of-scrum/ (datum pristupa: 17.06.2018.)

- System BD (n.d.) iz: Business Dictionary. URL: http://www.businessdictionary.com/definition/system.html (datum pristupa: 14.02. 2017.)

- System OED (n.d.). Online Etymology Dictionary. URL: http://www.etymonline.com/index.php?term=system (datum pristupa: 14.02.2017.)

- Technovelgy. What is RFID?. URL: http://www.technovelgy.com/ct/technology-article.asp (datum pristupa: 18.02.2017.)

- Tutorialspoint (n.d.). Software Case Tools Overview. URL: http://www.tutorialspoint.com/software_engineering/case_tools_overview.htm (datum pristupa: 14.10.2018.)

- UML diagrams (n.d.). UML Class and Object Diagrams Overview. URL: http://www.uml-diagrams.org/class-diagrams-overview.html (datum pristupa: 08.09.2018.)

- VersionOne - The 12th annual State of Agile (2017). URL:

https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report , datum pristupa: 18.06.2018.)

SLIKE

- Slika 1. DIKW Piramida;
  URL:https://www.climate-eval.org/sites/default/files/images/blog/Data-information-knowledge-wisdom.png, (datum pristupa: 13.02.2017.)

- Slika 2. Tok informacija kroz organizaciju;
  URL: https://goo.gl/images/Mdm8Q7, datum pristupa:13.02.2017.)

- Slika 3. Apstraktni prikaz nekog sustava;
  Izvor: Watson, R.T. (ed.) (2007). Information Systems. University of Georgia: Global Text Project. Str. 5. URL:
  http://www.uky.edu/~gmswan3/777/IS_Book.pdf

- Slika 4. Leavittov dijamant;
  Izvor: Cornford, T. i Shaikh, M. (2013). Introduction to information systems. London, UK: University of London.
  URL:http://www.dphu.org/uploads/attachements/books/books_3326_0.pdf
  (datum pristupa: 13.02.2018.)

- Slika 5. Proces generiranja informacija;
  URL:http://www.uotechnology.edu.iq/ce/Lectures/SarmadFuad-MIS/MIS_Lecture_3.pdf , datum pristupa: 24.06.2018.)

- Slika 6. Integrirani ERP sustav;
  Izvor: Laudon, K.C. i Laudon, J.P. (2014). Management Information Systems: Managing the Digital Firm, Global Edition, 13. izdanje. Essex, England: Pearson Education Limited. Str. 86. URL:
  http://www.icto.info/laudon-management-information-systems-13th-global-edition-c2014-1.pdf (datum pristupa: 05.03.2018.)

- Slika 7. Primjer SCM Sustava;
  Izvor: Oz, E. (2008). Management information systems, 6. izdanje. Course Technology. Str. 293.

- Slika 8. CRM;
  Izvor: Free CRM Applications. (2007). *URL: https://www.thebalancecareers.com/free-crm-applications-2917025* (datum pristupa: 06.03.2018.)

- Slika 9. Primjer TPS-a – proces isplate plaća;
  Izvor: Laudon, K.C. i Laudon, J.P. (2014). Management Information Systems: Managing the Digital Firm, Global Edition, 13. izdanje. Essex, England: Pearson Education Limited. Str. 77. URL: http://www.icto.info/laudon-management-information-systems-13th-global-edition-c2014-1.pdf (datum pristupa: 05.03.2018.)

- Slika 10. Sustav za upravljanje informacijama;
  Izvor: Stair, R., Reynolds, G. (2010). Principles of Information Systems A Managerial Approach, 9. izdanje. Boston, MA: Course Technology. Str. 400. URL: https://drive.uqu.edu.sa/_/fbshareef/files/principles%20of%20information%20systems%209th%20-stair,%20reynolds.pdf (datum pristupa: 13.02.2017.)

- Slika 11. Sustav za potporu u odlučivanju;
  Izvor: Turban, E. i Aronson, J.E. (2001). Decision Support Systems and Intelligent Systems, 6. izdanje. Upper Saddle River, NJ: Prentice Hall. Pog. 1.

- Slika 12. Model spirale znanja; URL: http://www.tlu.ee/~sirvir/IKM/Theoretical_models_of_Information_and_Knowledge_Management/the_nonaka_and_takeuchi_knowledge_spiral_model_page_3.html (datum pristupa: 16.03.2018.)

- Slika 13. Ekspertni sustav;

Izvor: Rainer, R.K., Prince, B. i Cegielski, C. (2014). Introduction to Information Systems Supporting and Transforming Business, 5. izdanje. New Jersey: Pearson Education. Str. 492.

- Slika 14. Waterfall model s povratnim vezama;
  Izvor: Royce, W. (1970). Managing the Development of Large Software Systems. Proc. Westcon, IEEE CS Press. Str. 328-339. URL: https://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf (datum pristupa: 24.05.2018.)

- Slika 15. V model;
  Izvor: Cadle, J. (Edt.) (2014). Developing information systems: Practical guidance for IT professionals. Swindon, UK: BCS Learning & Development Ltd. Str. 19.

- Slika 16. Notacije dijagrama toka podataka;
  URL: https://www.lucidchart.com/pages/data-flow-diagram (datum pristupa: 09. 07. 2018.)

- Slika 17. Dijagram toka podataka nulte razine – Kontekstualni dijagram;
  Izvor: Valacich, J.S., George, J.F. i Hoffer, J.A. (2011). Essentials of systems analysis and design, 5. izdanje. New Jersey: Pearson Education. Str. 158. URL: https://www.bau.edu.jo/inst/hamza/SAbook.pdf (datum pristupa: 13.02.2018.)

- Slika 18. Dijagram toka podataka prve razine;
  Izvor: Valacich, J.S., George, J.F. i Hoffer, J.A. (2011). Essentials of systems analysis and design, 5. izdanje. New Jersey: Pearson Education. Str. 159. URL: https://www.bau.edu.jo/inst/hamza/SAbook.pdf (datum pristupa: 13.02.2018.)

- Slika 19. ER dijagram;
  URL: https://www.lucidchart.com/pages/er-diagrams?a=0
  (datum pristupa: 09.07.2018.)

- Slika 20. Entitet;
  URL: https://www.lucidchart.com/pages/er-diagrams?a=0
  (datum pristupa: 09.07.2018.)

- Slika 21. Atributi;
  URL: https://www.lucidchart.com/pages/er-diagrams?a=0
  (datum pristupa: 09.07.2018.)

- Slika 22. Tipovi podataka;
  URL: https://www.lucidchart.com/pages/er-diagrams?a=0
  (datum pristupa: 09.07.2018.)

- Slika 23. Notacije odnosa
  URL: https://www.lucidchart.com/pages/er-diagrams?a=0
  (datum pristupa: 09.07.2018.)

- Slika 24. Primjer konteksta korištenja notacije odnosa;
  URL: https://www.lucidchart.com/pages/er-diagrams?a=0
  (datum pristupa: 09.07.2018.)

- Slika 24. Inkrementalni razvoj;
  Izvor: Cadle, J. (Edt.) (2014). Developing information systems: Practical guidance for IT professionals. Swindon, UK: BCS Learning & Development Ltd. Str. 22.

- Slika 25. Iterativni razvoj;
  Izvor: Cadle, J. (Edt.) (2014). Developing information systems: Practical guidance for IT professionals. Swindon, UK: BCS Learning & Development Ltd. Str. 24.

- Slika 27. Prototipiranje;
  Izvor: Valacich, J.S., George, J.F. i Hoffer, J.A. (2011). Essentials of systems analysis and design, 5. izdanje. New Jersey: Pearson Education. Str. 18. URL:

https://www.bau.edu.jo/inst/hamza/SAbook.pdf (datum pristupa: 13.02.2018.)

- Slika 28. Spiralni model;
  Izvor: Boehm, B. (1986). “A Spiral Model of Software Development and Enhancement,” Proc. Int’l Workshop Software Process and Software Environments, ACM Press, 1985; also in ACM Software Eng. Notes, Aug. 1986 Str.. 22-42. URL:
  http://csse.usc.edu/TECHRPTS/1988/usccse88-500/usccse88-500.pdf
  (datum pristupa: 25.05.2018.)

- Slika 29. Objektno orijentirani razvoj;
  Izvor: Kendall, K. E. i Kendall, J. E. (2006). System Analysis and Design, 8. izdanje. NJ: Prentice Hall. Str. 18.

- Slika 30. Notacija odnosa Nadklasa – Podklasa;
  URL: https://www.uml-diagrams.org/use-case-actor.html
  (datum pristupa: 08.09.2018.)

- Slika 31. Use case dijagram;
  URL: https://www.uml-diagrams.org/use-case-diagrams.html
  (datum pristupa: 08.09.2018.)

- Slika 32. Primjer klase;
  URL: https://www.uml-diagrams.org/class-diagrams-overview.html
  (datum pristupa: 08.09.2018.)

- Slika 33. Primjer klasnog dijagrama;
  URL: https://www.uml-diagrams.org/class-diagrams-overview.html
  (datum pristupa: 08.09.2018.)

- Slika 34. Primjer sekvencijalnog dijagrama;
  URL: https://www.uml-diagrams.org/sequence-diagrams.html
  (datum pristupa: 10.09.2018.)

- Slika 35. Primjer uvjetovane interakcije; URL: http://www.tracemodeler.com/articles/a_quick_introduction_to_uml_sequence_diagrams/ , datum pristupa: 10.09.2018.)

- Slika 36. Primjer dijagrama stanja; URL: https://www.lucidchart.com/pages/uml/state-machine-diagram, datum pristupa: 10.09.2018.)

- Slika 37. Primjer dijagrama aktivnosti; URL: https://www.lucidchart.com/pages/uml/activity-diagram (datum pristupa: 11.09.2018.)

- Slika 38. Rational Unified Process; URL: https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf, datum pristupa: 07.06.2017.)

- Slika 39. Zajednički razvoj aplikacija; http://www.chambers.com.au/glossary/joint_application_design.php (datum pristupa: 07.06.2017.)

- Slika 40. Faze brzog razvoja aplikacija; URL: http://www.ftms.edu.my/images/Document/IMM006%20-%20RAPID%20APPLICATION%20DEVELOPMENT/Chapter%202nnote.pdf (datum pristupa: 10.06.2017.)

- Slika 41. Agilni razvoj;
  Izvor: Izvor: Kendall, K. E. i Kendall, J. E. (2006). System Analysis and Design, 8. izdanje. NJ: Prentice Hall. Str. 16.

- Slika 42. Kristalne metode;
  Izvor: Coffin, R. i Lane, D. (2006). A Practical Guide to Seven Agile Methodologies, Part 2: Page 2. DevX: The know-how behind application

development. Izvor: http://www.devx.com/architect/Article/32836/0/page/2 (datum pristupa: 13.06.2017.)

- Slika 43. Iterativno inkrementalni razvoj – Kristalne metode;
  Izvor: Abrahamsson, P., Salo, O. i Ronkainen, J. (2002). Agile Software Development Methods: Review and Analysis. Espoo, Finland: VTT publication 478. Str. 43. URL: https://arxiv.org/ftp/arxiv/papers/1709/1709.08439.pdf (datum pristupa: 13.06.2017.)

- Slika 44. DSDM - usporedba s tradicionalnim pristupom; URL:
  https://www.agilebusiness.org/content/philosophy-and-fundamentals
  (datum pristupa:14.05.2018.)

- Slika 45. Dinamička metoda razvoja sustava; URL:
  https://www.agilebusiness.org/content/project-planning-and-control
  (datum pristupa:14.05.2018.)

- Slika 46. Timeboxing; URL:
  Izvor: Tech Academy (2013).URL:
  http://agile.tech-academy.co.uk/files/2013/10/hi-levelDevPlan.png
  (datum pristupa:14.05.2018.)

- Slika 47. Scrum;
  Izvor: Abrahamsson, P., Salo, O. i Ronkainen, J. (2002). Agile Software Development Methods: Review and Analysis. Espoo, Finland: VTT publication 478. Str. 34. URL: https://arxiv.org/ftp/arxiv/papers/1709/1709.08439.pdf (datum pristupa: 13.06.2018.)

- Slika 48. Razvoj temeljen na funkcionalnostima;
  Izvor: Ambler, W. (n.d.). Feature Driven Development (FDD) and Agile Modeling. URL: http://agilemodeling.com/essays/fdd.htm (datum pristupa: 17.06.2018.)

- Slika 49. Ekstremno programiranje;
  Izvor: Abrahamsson, P., Salo, O. i Ronkainen, J. (2002). Agile Software Development Methods: Review and Analysis. Espoo, Finland: VTT publication 478. Str. 21. URL: https://arxiv.org/ftp/arxiv/papers/1709/1709.08439.pdf (datum pristupa: 13.06.2018.)

- Slika 50. Adaptivni razvoj sustava;
  Izvor: Abrahamsson, P., Salo, O. i Ronkainen, J. (2002). Agile Software Development Methods: Review and Analysis. Espoo, Finland: VTT publication 478. Str. 74. URL: https://arxiv.org/ftp/arxiv/papers/1709/1709.08439.pdf (datum pristupa: 13.06.2018.)

- Slika 51. Razvoj IS-a temeljenih na web-u;
  Izvor: Brambilla et al (n.d.). Designing Web Applicationes with WebMl and WebRatio. URL: http://www.csun.edu/~twang/595WEB/Slides/WebML.pdf (datum pristupa: 03.06.2018.)

- Slika 52. Modeliranje klasa – podatkovni dizajn; URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.472.4706&rep=rep1&type=pdf (datum pristupa: 04.06.2018.)

- Slika 53. Navigacija; URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.472.4706&rep=rep1&type=pdf (datum pristupa: 04.06.2018.)

- Slika 54. Povezanost komponenti;
  Izvor: Cadle, J. (Edt.) (2014). Developing information systems: Practical guidance for IT professionals. Swindon, UK: BCS Learning & Development Ltd. Str. 89.

- Slika 55. Sučelje komponente s funkcijama;
  Izvor: Cadle, J. (Edt.) (2014). Developing information systems: Practical guidance for IT professionals. Swindon, UK: BCS Learning & Development Ltd.

Str. 90.

- Slika 56. CASE Tehnologija; URL: https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/CASE/ToolsWBEnv.htm (datum pristupa: 14.10.2018.)

- Slika 57. Kategorizacija CASE alata; URL: https://www.careerride.com/page/case-tools-663.aspx (datum pristupa: 14.10.2018.)

- Slika 58. Centralna pohrana; URL: https://www.careerride.com/page/case-tools-663.aspx (datum pristupa: 14.10.2018.)

- Slika 59. Distribucija primjene metodologija/metoda; URL: https://techbeacon.com/sites/default/files/gated_asset/agile-projects-are-more-successful-than-hybrid-projects.pdf (datum pristupa: 14. 09. 2018.)

- Slika 60. Analitički okvir usporedbe metodologija;
  Izvor: Izrada autora

- Slika 61. Metodološko pokriće faza;
  Izvor: Abrahamsson, P et al. (2002). Agile Software Development Methods: Review and Analysis. Finland: VTT. Str. 101. URL: http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf
  (datum pristupa:15.09.2018.)

- Slika 62. Razine faktora složenosti; URL: http://www.disciplinedagiledelivery.com/tool-support-for-dad/
  (datum pristupa: 07.11.2018.)

- Slika 63. Distribucija primjene metodologija; URL:

https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report (datum pristupa: 07.11.2018.)

- Slika 64. Uspješnost projekata u razdoblju 2011. do 2015.; URL: https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf (datum pristupa: 16.11.2018.)

- Slika 65. Uspješnost projekata po primjeni metoda; URL: https://vitalitychicago.com/blog/agile-projects-are-more-successful-traditional-projects/ (datum pristupa: 16.11.2018.)

- Slika 66. Uspješnost projekata po složenosti projekta; URL: https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf (datum pristupa: 16.11.2018.)

- Slika 67. Uspješnost metodoloških primjena u odnosu na veličinu projekata; URL: https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf (datum pristupa: 16.11.2018.)

TABLICE

- Tablica 1: Načini manifestiranja podataka;
  Izvor: Stair, R., Reynolds, G. (2010). Principles of Information Systems A Managerial Approach, 9. izdanje. Boston, MA: Course Technology. Str. 5. URL: https://drive.uqu.edu.sa/_/fbshareef/files/principles%20of%20information%20systems%209th%20-stair,%20reynolds.pdf (datum pristupa: 13.02.2017.)

- Tablica 2. Prednosti i nedostaci brzog razvoja aplikacija; URL: https://www.tutorialspoint.com/sdlc/pdf/sdlc_rad_model.pdf
  (datum pristupa: 10.06.2017.)

- Tablica 3. Usporedba metodologija/metoda po podrijetlu

Izvor: Izrada autora

- Tablica 4. Usporedba Resursa
  Izvor: Izrada autora

- Tablica 5. Usporedba Praksi
  Izvor: Izrada autora

- Tablica 6. Usporedba Praksi – Agilne Metode
  Izvor: Izrada autora

- Tablica 7. Faktori složenosti i alati
  Izvor: Izrada autora

- Tablica 8. Dostupni alati po metodama
  Izvor: Izrada autora

- Tablica 9. Prednosti i nedostaci metodologija/metoda
  Izvor: Izrada autora