Sveučilište Jurja Dobrile u Puli

Odjel za informacijsko-komunikacijske tehnologije

Toni Bileta

Kristalna obitelj metodologija razvoja programskih proizvoda

Završni rad

Pula, 2016. godine

Sveučilište Jurja Dobrile u Puli

Odjel za informacijsko-komunikacijske tehnologije

Toni Bileta

Kristalna obitelj metodologija razvoja programskih proizvoda

Završni rad

JMBAG: 0303046400, redoviti student

Studijski smjer: Informatika

Predmet:Softversko inžinjerstvo

Mentor: doc. dr. sc. Tihomir Orehovački

Pula, rujan, 2016. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Toni Bileta, kandidat za prvostupnika informatike ovime izjavljujem da je ovaj Završni 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 Završnog 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, _____, _____ godine

IZJAVA

o korištenju autorskog djela

Ja, Toni Bileta dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom Kristalna obitelj metodologija razvoja programskih proizvoda 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, _____ (datum)

Potpis

_____

Sadržaj

Uvod

Pri razvijanju bilo kakvog programskog proizvoda potrebno je odabrati pravu metodologiju kako bi se osigurao uspjeh projekta zbog toga što odabir krive metodologije može imati katastrofalne posljedice za sveukupnu uspješnost projekta. Glavna tematika ovog završnog rada je upravo jedna od tih metodologija, a to je kristalna obitelj metodologija za razvoj softvera. Razlog odabiranja ove tematike za završni rad bio je taj što je ona djelovala najzanimljivije od svih koje su bile ponuđene na izbor. Svrha ovoga rada je najprije prikazati nekoliko različitih metoda i metodologija za razvoj softvera kako bi se steklo nekakvo opće poznavanje tih metoda i njihovih svojstava. Poznavanje tih metodologija omogućava da prilikom opisivanja kristalne obitelji metodologija do izražaja dođu njihove međusobne razlike i sličnosti te također da do izražaja dođu specifičnosti kristalnih metodologija u odnosu na ostale. Ovaj rad ukratko će se baviti svojstvima i procesima za svaku od navedenih metodologija te mnogo detaljnije upravo kristalnim metodologijama. U tom djelu cilj će biti što bolje prikazati što je to kristalna obitelj metodologija i na koji način je zamišljena za korištenje od strane svoga autora. Svrha iznošenja njenih svojstava, prednosti i primjena je ta da osoba koja čita ovaj rad što bolje razumije ovu metodologiju i njene primjene. Rezultat toga razumijevanja bi trebala biti mogućnost da donese informiranu odluku o tome da li bi primjena ove metodologije za neki softverski projekt bila isplativa. Ukoliko se osoba odluči za primjenu ovog seta metodologija također je potrebno da bude upoznata načelima i metodama kojima je može što efektivnije primjeniti, prednostima i eventualnim poteškoćama pri radu s njima. Također je bitno da korisnik ove obitelji metodologija zna prepoznati da li je pravilno koristi i da zna koje su potrebne korekcije koje treba primjeniti u slučaju krivog korištenja. Prilikom istraživanja i pisanja ove teme korišten je cijeli niz literatura od članaka pronađenih na internetskim stranicama pa sve do knjiga napisanih od strane umova koji su i najzaslužniji za razvoj najznačajnijih metodologija koje će biti spomenute unutar razrade ovoga rada poput Alistaira Cockburna, Kent Becka, Ken Schwabera i ostalih. Ovaj se rad sastoji od dva veća poglavlja, svaki od njih sa svojim pripadajućim potpoglavljima koja sadrže dodatne detalje o temi. Osim ta dva opsežna poglavlja na kraju se nalazi zaključak koji ukratko sumarizira ovu temu.

Naziv prvog poglavlja je Metodologije te su unutar njega opisane klasične i agilne metode. Za obje od tih metodologija navedena su njihova glavna svojstva i značajke. Potom su i za svaku od njih navedene i opisane najpopularnije od njihovih metoda uz pomoć slika i teksta kojima su opisani procesi i aktivnosti koji su specifični za svaku od njih.

Drugo poglavlje ima naziv Kristalna obitelj metodologija te predstavlja glavnu temu ovoga rada. Unutar toga poglavlja opisan je autor ove metodologije te njegovi stavovi koji su utjecali na njen razvoj. Potom su navedena i opisana svojstva i principi na kojima je ova metodologija zasnovana. Nakon toga opisani su procesi, jedna od uspješnih primjena ove metodologije te usporedba sa dvije agilne metode, ekstremnog programiranja i scrum-a.

Na kraju dolazi zaključak unutar kojega je sumarizirana, ova metodologija, njeni principi i razvojne politike. Osim toga u zaključku je navedeno što smatram možda i najvećim problemom kod korištenja ove metodologije i moje osobno mišljenje da li je korištenje ove metodologije isplativo ili ne.

1. Metodologije razvoja softvera

Razvoj softverskog proizvoda je dug i nerijetko kompliciran proces koji se sastoji od mnogih manjih procesa poput utvrđivanja zahtjeva, oblikovanja i implementacije, verifikacije i validacije te naposljetku održavanja i evolucije softverskog proizvoda. Kako bi se osigurala uspješnost projekta potrebno je odabrati metodologiju razvoja koja nam najviše odgovara za taj projekt te koja će osigurati da ćemo za razvoj proizvoda potrošiti što manje resursa bez kompromitiranja kvalitete završnog proizvoda. Kod odabira metodologija razlikujemo dvije glavne grupe a to su klasične metode i agilne metode koje se međusobno poprilično razlikuju.

1.1 Klasične metodologije

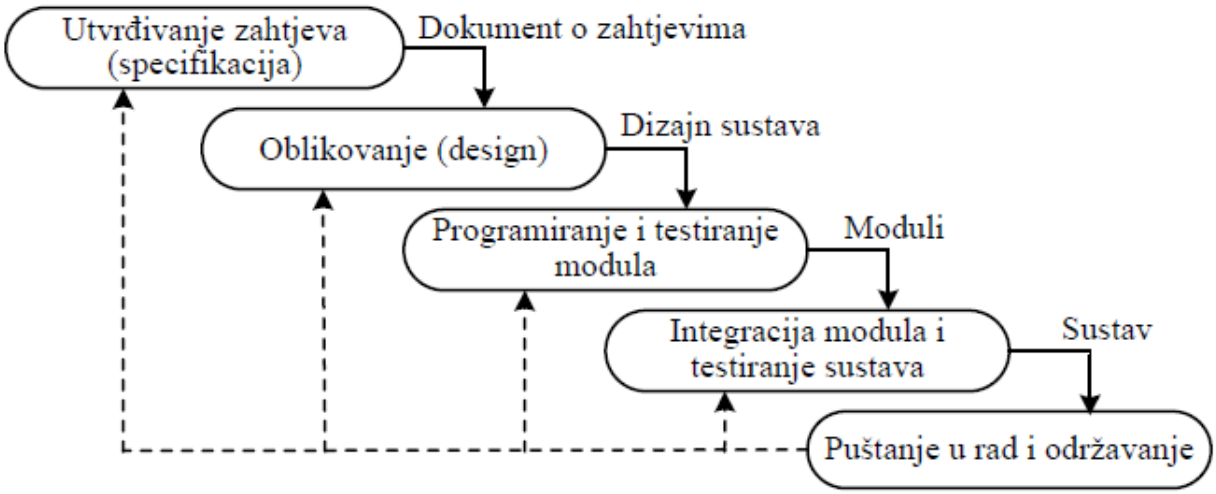
Klasične metodologije prvi puta se pojavljuju 1970-ih te je njihovo glavno svojstvo to da se sastoje od faza koje opisuju cijeli životni ciklus softverskog proizvoda. To znači da razvoj softvera ima svoj početak i kraj te se realizacija softverskog proizvoda u potpunosti izvršava u skladu sa utvrđenim zahtjevima, projektom se pažljivo upravlja te se određuje vremensko trajanje pojedinih aktivnosti i ljudi koji će na njima raditi. Još jedno jako bitno svojstvo klasičnih metodologija je to da se moraju dokumentirati sve aktivnosti unutar životnog ciklusa softverskog proizvoda. Klasične metodologije sadrže različite modele za upravljanje životnim ciklusom softverskog proizvoda i neke od njih su model vodopada, v-model, prototipiranje, inkrementalni razvoj, RAD (rapid application development) model i evolucijski model. Osim na to što se djele na modele, klasične metodologije se također dijele i na funkcionalno orijentirane metode, stariji programski jezici poput Cobola i Fortrana, te na objektno orijentirane metode kod kojih imamo novije programske jezike poput Jave i C++ (Hneif i Ow 2007).

1.1.1 Model vodopada

Model vodopada je jedan od najpoznatijih pripadnika klasičnih metodologija razvoja softvera te je utjecao na razvoj nekih ostalih poput recimo v-modela. Model vodopada je sekvencijalni pristup razvoju softvera, sastoji se od pet ili više uzastopnih faza te način na koji se napreduje kroz svaku od njih podsjeća na padanje vode te je od tuda i došao naziv ovoga modela (Mohammed Sami 2015). To sve znači da se iduća faza projekta ne može izvršiti ukoliko trenutna faza nije u potpunosti izvršena. Također ukoliko se iz nekoga razloga moramo vratiti u prijašnju fazu nekoga projekta, bez

obzira što to baš i nije poželjno, to je moguće samo u onu koja direktno prethodi fazi u kojoj se trenutno nalazimo.

Kao što je vidljivo sa slike model se sastoji od pet glavnih aktivnosti a one su: utvrđivanje zahtjeva, oblikovanje sustava, programiranje i implementacija modula, integracija modula i testiranje sustava te na kraju puštanje sustava u rad i njegovo održavanje (slika 1). Zahvaljujući načinu na koji je model definiran omogućeno nam je detaljno planiranje cijelog procesa, može se jednostavno ustanoviti u kojoj se fazi trenutno nalazimo te je održana čvrsta kontrola nad projektom uz pomoć detaljne dokumentacije.



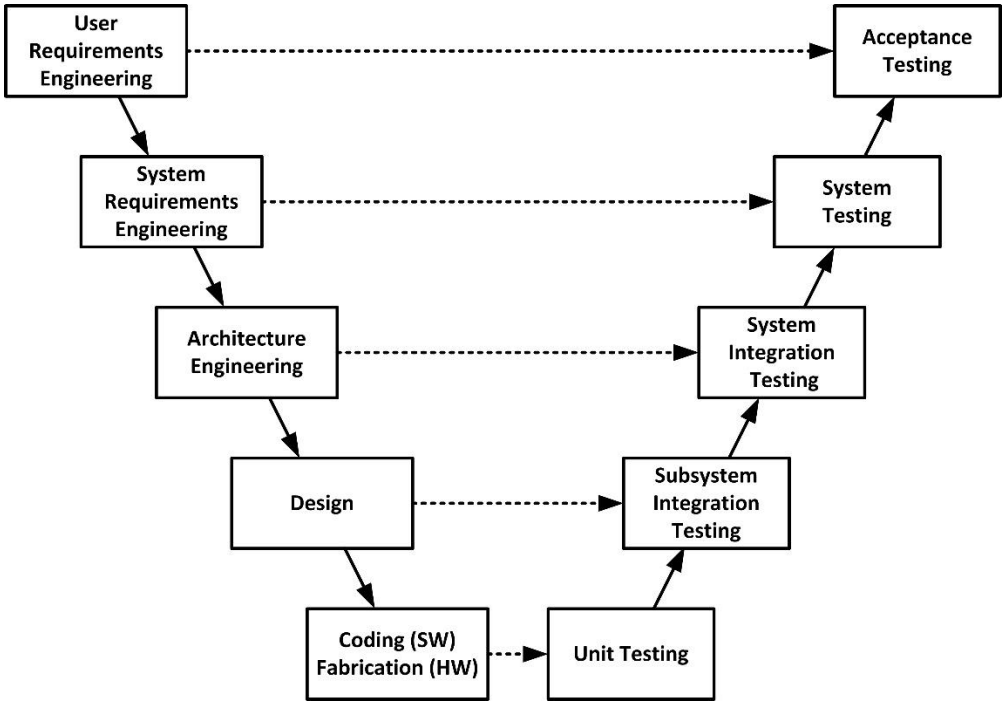
Slika 1 - Model vodopada
Izvor: Manger, R. (2013).

1.1.2 V-model

V-model je kao što je i prije navedeno nastao zahvaljujući modelu vodopada te predstavlja njegovu ekstenziju. Za razliku od modela vodopada gdje se tijek aktivnosti odvija konstantno prema dolje u ovom slučaju se nakon implementacije procesa vraćaju gore te je omogućeno vraćanje iz procesa koji dolaze kasnije u procese koji su već izvršeni ranije.

Kao što slika prikazuje v-model se sastoji od više aktivnosti te nudi više mogućnosti interakcija između pojedinih aktivnosti (slika 2). Zbog svog dizajna v-model je jednostavan za korištenje i svaka od faza ima specifične rezultate. Korištenjem ovog

modela razvojni tim ima veću šansu za uspješno provođenje projekta zbog toga što za razliku od modela vodopada omogućava da se ovdje planovi za testiranje razvijaju ranije te on također omogućava verifikaciju i validaciju proizvoda u ranim stadijima razvoja proizvoda.



Slika 2 - V-model

Izvor: SEI Blog (2013)

1.2 Agilne metodologije

Agilne metode razvoja softverskog proizvoda su osmišljene kao odgovor na klasične metodologije te predstavljaju njihovu suprotnost. Začetnici agilnih metodologija su Ken Beck i Alistair Cockburn koji su se sa ostalim članovima okupili oko Agile Alliance neprofitne organizacije čiji je cilj promoviranje agilnih metoda. Godine 2001. pobornici agilnih metoda razvoja napisali su Agile Software Development Manifesto u kojemu su deklarirane sljedeće vrijednosti agilnog programiranja: interakcije među pojedincima su važnije nego procesi i alati, programska podrška je bitnija nego dokumentacija, suradnja s klijentima je bitnija nego pregovaranje oko ugovora te je reagiranje na promjene važnije nego slijeđenje unaprijed zadanog plana (Williams 2007). Unutar tog manifesta također su iskazani

principi prema kojima se agilne metode trebaju razvijati i primjenjivati, ti principi su sljedeći:

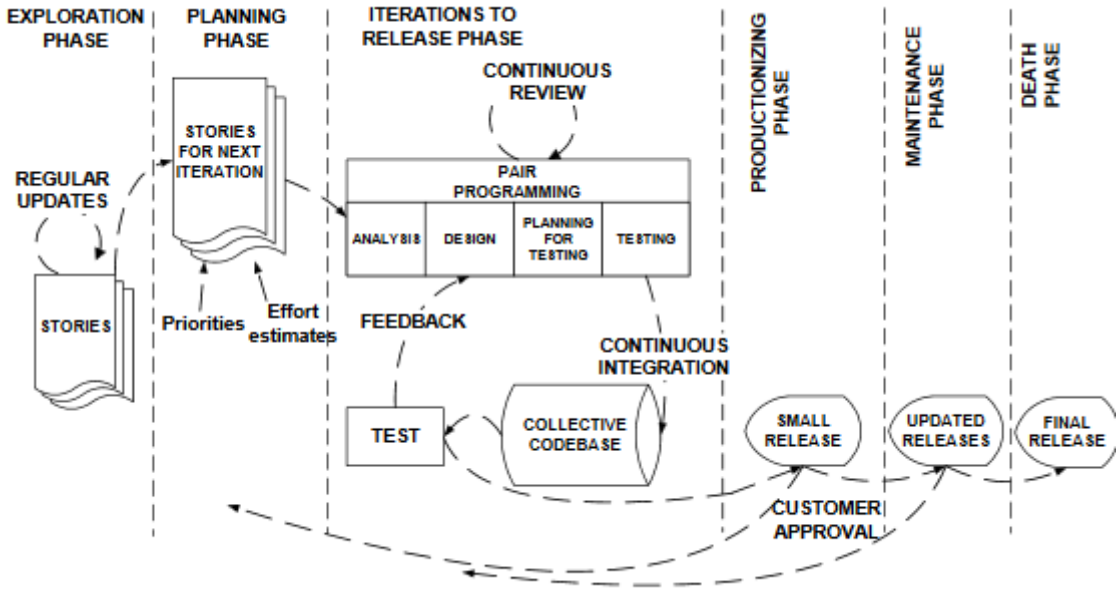
- Zadovoljstvo klijenta
- Zahtjevi za promjenama su dobrodošli u svakom trenutku
- Naglasak na iteracijama
- Na projektu rade ljudi i inženjeri iz poslovnih područja koji su i inicirali projekt
- Na projektu rade motivirane osobe
- Naglasak je na komunikaciji licem u lice
- Osnovna mjera napretka je program koji radi
- Program se stalno razvija
- Jednostavnost
- Timovi koji se sami razvijaju
- Periodičko ispitivanje dobrih i loših postupaka koji se onda popravljaju u sljedećem periodu

Iz svih tih zahtjeva i principa proizašlo je nekoliko različitih metoda agilnog razvoja softverskog proizvoda i neke od najpopularnijih su ekstremno programiranje, scrum, FDD (feature driven development),DSDM (Dynamic Systems Development Method) te naposljetku i kristalna obitelj metodologija koje su i tema ovoga rada.

1.2.1 Ekstremno programiranje (XP)

Ideja ekstremnog programiranja nastala je od strane Kent Becka 2000. godine. Ova metodologija nudi razne prakse, vrijednosti i principe koji trebaju biti prihvaćeni kako bi se moglo voditi poslovni projekt te je ova metoda razvoja namijenjena primarno za manje timove koji ne razvijaju pretjerano kompleksne proizvode. Glavna premisa ekstremnog programiranja je dostavljanje proizvoda visoke kvalitete brzo i konstantno, osim toga kod ekstremnog programiranja se promovira visoka uključenost kupca, brza povratna veza, konstantno testiranje i planiranje te timski rad s ciljem da se programski proizvod dostavlja unutar jednog do tri tjedna (Abrahamsson et. al. 2002). Kod ekstremnog programiranja kupac blisko surađuje sa razvojnim timom koristeći takozvane korisničke priče, potom razvojni tim dostavlja proizvod koji je u potpunosti baziran oko tih korisničkih priča odnosno želja kupaca. Dakle glavni principi ekstremnog programiranja su bliska suradnja sa kupcima, izbacivanje novih verzija

proizvoda u kratkim intervalima i timski rad. Ekstremno programiranje je primarno namijenjeno za timove od dvoje do dvanaest ljudi te je upravo zbog toga jako važna prije navedena komunikacija i timski rad među članovima tima i kupaca. Kao što je vidljivo na sljedećoj slici ciklus ekstremnog programiranja sastoji se od pet faza a one su: istraživanje (otkrivanje zahtjeva), planiranje, iteracije, pretvaranje prototipa u verziju za masivno korištenje, održavanje i smrt (slika 3) te ću u nastavku ukratko pojasniti svaku od faza.



Slika 3 - Životni ciklus XP procesa

Izvor: Wiki.amachu (2013a).

U eksploracijskoj fazi klijenti pišu korisničke priče to jest pišu zahtjeve za funkcije koje bi htjeli vidjeti u prvoj verziji softverskog proizvoda. Istovremeno razvojni se tim upoznaje sa alatima i tehnologijom koju će koristiti pri radu na projektu, ta će se tehnologija potom testirati i otkriti na koji način je se može koristiti za projekt. Ova faza može trajati između jednog tjedna i nekoliko mjeseci ovisno o tome koliko su programeri upoznati sa tehnologijom s kojom moraju raditi.

U fazi planiranja se korisničkim pričama dodaje prioritet i postiže se dogovor o tome koje će se funkcionalnosti nalaziti u prvoj verziji proizvoda. Programeri najprije moraju procijeniti koliko vremena bi trebalo za rad na svakoj od korisničkih priča te se na osnovu toga stvara raspored. Ova faza obično ne traje više od dva dana.

Sljedeća je faza stvaranja iteracija, u njoj se naime stvara više različitih iteracija programskog proizvoda prije njegova izlaska. Raspored koji je dogovoren u fazi planiranja se koristi kako bi se stvorio novi raspored prema kojemu će se raditi svaka od iteracija od kojih će za svaku trebati jednoga do četiri tjedna kako bi se implementirala. Prva od tih iteracija stvara sustav sa njegovom potpuno dizajniranom arhitekturom, to je postignuto koristeći korisničke priče čija implementacija će poboljšati strukturu cijeloga sustava. Također treba napomenuti da su klijenti ti koji odlučuju koja od korisničkih priča će biti izabrana za koju iteraciju, te su upravo oni ti koji provode funkcionalne testove na kraju svake od iteracija.

Faza koja slijedi nakon stvaranja iteracija je pretvaranje prototipa u proizvod koji je spreman za masovno korištenje. U ovoj fazi se proizvod dodatno testira i provjeravaju mu se performanse prije nego što ga se može dati klijentu na korištenje. Tijekom ove faze mogu se naći nove promjene na proizvodu te je potrebno odlučiti da li ih se želi izdati u ovoj verziji ili u nekoj od idućih. Svaka od tih mogućnosti i ideja je dokumentirana za kasniju implementaciju.

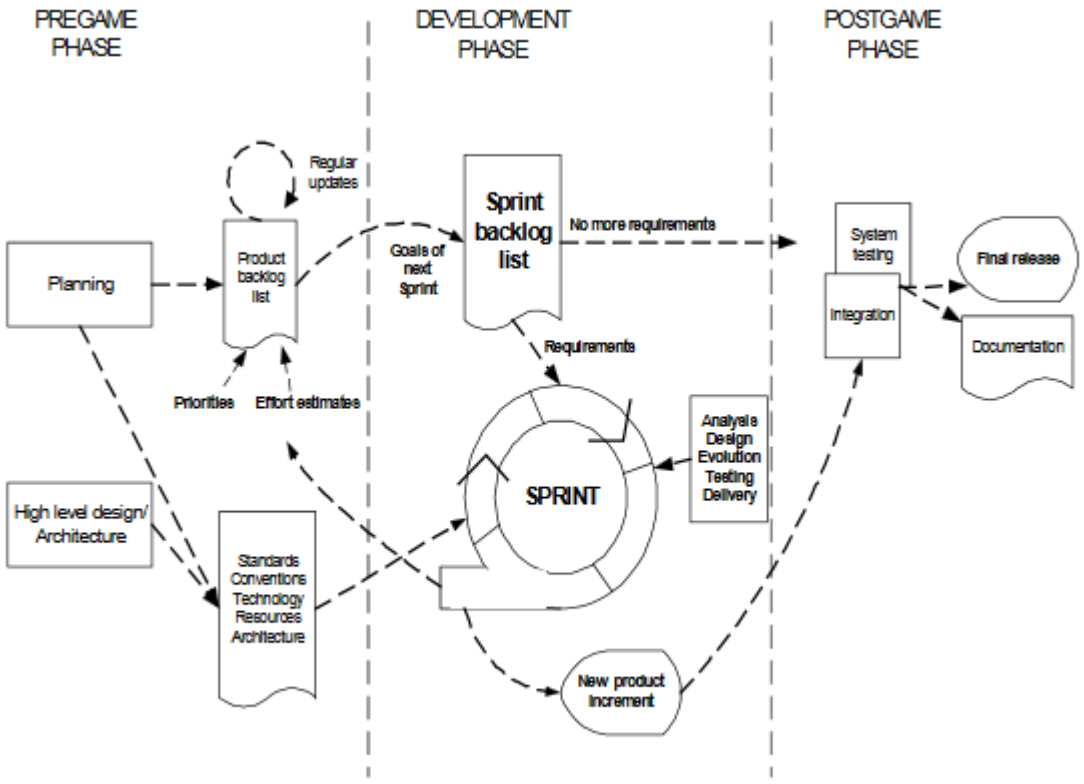
Nakon što je prva verzija proizvoda dovršena i izdana klijentu razvojni tim mora održavati sustav kako ne bi došlo do degradacije performansi i sve to dok rade na novim iteracijama tog softverskog proizvoda. Kako bi se što uspješnije moglo raditi i na novim iteracijama i na održavanju sustava možda dođe do potrebe za zapošljavanjem dodatne radne snage.

Naposljetku dolazimo do faze smrti proizvoda. Do te faze dolazi kada klijenti više nemaju nikakvih korisničkih priča za koje žele da budu implementirane i kada je klijent zadovoljan završnim proizvodom. U ovoj fazi se piše sva potrebna dokumentacija za završni proizvod i ne rade se više nikakve izmjene unutar koda, dizajna ili arhitekture. Smrtna faza se osim zbog uspjeha može dogoditi i ako je projekt neuspješan to jest ako ne zadovoljava zahtjeve kupca ili ako jednostavno postane previše skup kako bi se nastavilo raditi na njemu.

1.2.2 Scrum

Scrum metodologiju osmislio je Ken Swaber 1995. te je ona bila prakticirana još i prije nego što je nastao Agile Manifesto te je kasnije uključena među agilne metodologije zbog toga što se pridržavala istih koncepata i pravila kao i ostale agilne metode. Naime prvo spominjanje scrum-a datira u 1986. kada su Takeuchi i Nonaka unutar jednog članka opisali adaptivni, brzi i samoorganizirajući razvojni proces te kako Ken Schwaber i Mike Beedle navode u svojoj knjizi *Agile Software Development with SCRUM* taj izraz proizlazi iz taktike koja se koristi u ragbiju.

Scrum pristup razvijen je kako bi se što uspješnije upravljalo procesom razvoja sustava. Scrum je empirijski pristup koji primjenjuje ideje teorije kontrole industrijskog procesa na razvoj sustava što rezultira pristup koji uvodi ideje fleksibilnosti, adaptabilnosti i produktivnosti (Beedle i Schwaber 2002). Ovaj pristup ne definira nikakve specifične metode za implementacijsku fazu već se bavi time kako bi članovi tima trebali funkcionirati kako bi sustav bio što fleksibilniji u okolini koja se stalno mijenja. Glavna ideja scrum-a je ta da razvoj sustava uključuje nekoliko varijabli koje će se vjerojatno mijenjati tijekom procesa što čini razvojni proces nepredvidivim i kompleksnim te tim koji radi na njemu treba biti spreman pravovremeno reagirati na te promjene. Rezultat toga procesa trebao bi biti proizvod koji je koristan odmah čim je isporučen. Kao što sljedeća slika prikazuje scrum proces se sastoji od tri faze a one su početna faza, razvojna faza i završna faza (slika 4).



Slika 4 - Životni ciklus scrum procesa

Izvor: Abrahamsson.P et.al. (2002).

Prema Schwaberovom radu *SCRUM Development Proces* te su faze definirane na sljedeće načine. Faza planiranja je podijeljena na dvije manje faze a one su planiranje i dizajniranje arhitekture. Planiranje uključuje definiciju sustava koji se razvija. Kako bi ta definicija bila što jasnija kreira se backlog unutar kojega se nalaze svi trenutno poznati zahtjevi za sustav. Ti zahtjevi mogu doći od strane kupaca, marketinškog odjela ili razvojnog tima. Ti zahtjevi se prioritiziraju i procjenjuje se koliko će truda biti potrebno za njihovu implementaciju. Taj se backlog konstantno ažurira sa novim i detaljnijim zahtjevima te sa preciznijim vremenskim procjenama i novim prioritetima (Schwaber 1996). Planiranje također zahtjeva da se definira tko sve radi na projektu, resursi, alati s kojima će se raditi, procjenu rizika te verifikaciju od strane menadžmenta. Nakon svake od iteracija backlog se provjerava od strane posebnih timova kako bi bili spremni za rad na idućoj iteraciji. U drugoj podfazi planiranja se dizajnira arhitektura sustava na osnovi stavki koje se nalaze unutar backloga. Za slučaj da su potrebne ikakve izmjene ili poboljšanja sustava te se promjene identificiraju

unutar backloga zajedno sa problemima koji bi njihova implementacija mogla prouzročiti. Potom se održava sastanak na kojemu se saslušavaju prijedlozi za implementaciju tih funkcionalnosti te se naposljetku donosi odluka da li će do njih doći.

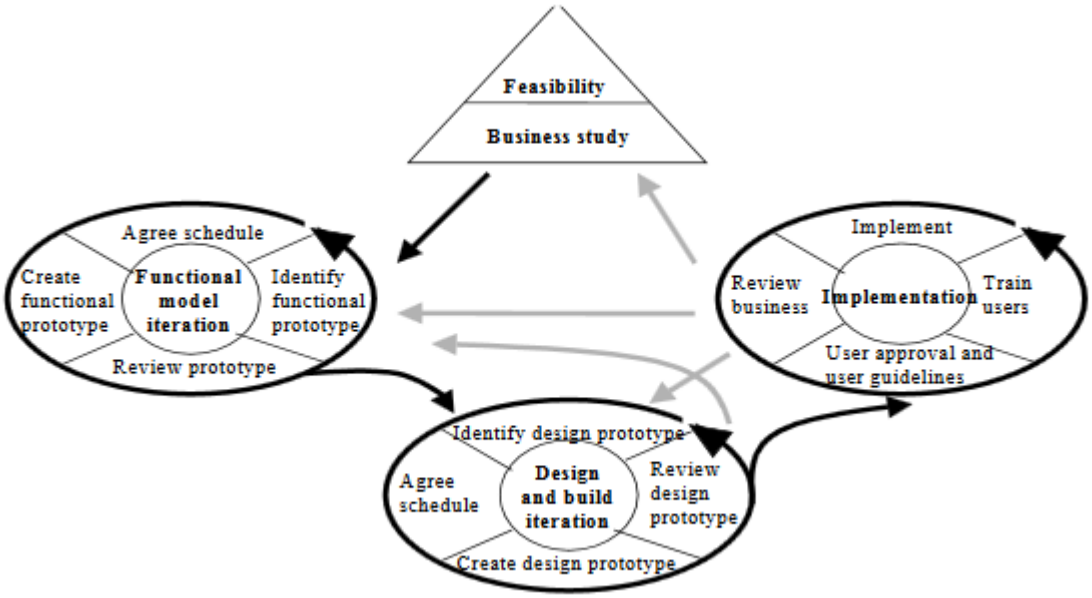
Nakon što je faza planiranja uspješno završena dolazi do faze razvoja te ona predstavlja ono što scrum metodu čini agilnom i ova se faza gleda kao crna kutija gdje se očekuje neočekivano. Različite varijable koje se mogu mijenjati tijekom procesa se prate i kontroliraju kroz razne metode tijekom razvojnog procesa. Dakle umjesto da se te stvari uzimaju u obzir samo na početku razvoja projekta one se stalno kontroliraju kako bi se moglo lakše prilagoditi tim promjenama. Tijekom razvojne faze proizvod se razvija u tako zvanim sprintovima. Sprintovi su iterativni ciklusi unutar kojih se razvijaju i poboljšavaju funkcionalnosti kako bi se stvorili novi inkrementi. Svaki od sprintova sadrži tradicionalne faze razvoja softvera: utvrđivanje zahtjeva, analiza, dizajn, evolucija i isporuka. Za svaki od sprintova je planirano da može trajati između jednog i četiri tjedna te ih unutar razvojnog ciklusa proizvoda može biti između tri i osam (Schwaber 1996).

U završnu fazu se ulazi kada se ustanovi da su varijable iz okoline sve izvršene te u tom slučaju se više nikakve nove stavke ne mogu dodavati ili stvarati. U tom stanju sustav je spreman za izlazak te se u ovoj fazi sve priprema upravo za njega preko zadataka kao što su integracija, testiranje sustava i izrada dokumentacije.

1.2.3 Dinamična metoda razvoja sustava

Prva verzija dinamičnije metode razvoja sustava (engl. Dynamic System Development Method ili DSDM skraćeno) nastala je 1994. zbog potrebe industrije za standardiziranim framework-om za dostavljanje projekata što je do tada bio zadatak koji je izvršavao RAD (Rapid application development). Bez obzira što je početkom 90-ih RAD bio jako popularan također je i bio jako nestrukturiran te je bilo potrebno naći novo rješenje. Rezultat toga bio je nastanak DSDM konzorcija 1994. godine te je njegov cilj bio stvaranje i promoviranje standardiziranog framework-a za brzu dostavu proizvoda. Od 1994. DSDM metodologija se konstantno mijenjala i sazrijevala kako bi pružala podršku za planiranje, upravljanje i izvršavanje iterativnih projekata za razvoj softvera. DSDM je baziran na devet ključnih principa koji su orijentirani prema

poslovnim potrebama, aktivnoj uključenosti korisnika u razvoj, česte dostave novih verzija proizvoda, integrirano testiranje i suradnja sa dioničarima (VersionOne 2016). Kod DSDM-a se svi zahtjevi određuju jako rano i sve promjene koje se mogu dogoditi tijekom razvoja moraju se moći prema potrebi vratiti u prijašnje stanje. Osim toga kod DSDM projekta sav kritičan posao mora biti dovršen na vrijeme te kako bi to bio moguće ne smije se svaki zahtjev smatrati kritičnim. DSDM je također neovisan te ga se može koristiti u kombinaciji sa ostalim iterativnim metodologijama poput recimo ekstremnog programiranja. Sljedeća slika prikazuje faze i njihove pripadajuće procese koji se pojavljuju prilikom rada sa DSDM-om (slika 5).



Slika 5 - Životni ciklus DSDM procesa

Izvor:System Analyst's blog (2010).

DSDM se sastoji od pet faza a one su sljedeće: studija izvedivosti, poslovna studija, iteracija funkcionalnog modela, iteracija dizajna i izgradnje i implementacija. Prve dvije od tih faza se izvode sekvencijalno i izvršavaju se samo jednom dok su zadnje tri faze iterativne i inkrementalne. Kod DSDM-a iteracijama se pristupa kao vremenskim okvirima, taj okvir traje unaprijed određeno vrijeme i iteracija mora biti izvršena unutar tog okvira. Vrijeme koje je dozvoljeno za svaku od iteracija treba biti unaprijed određeno zajedno sa rezultatima koji su od nje očekivani. Taj vremenski okvir može biti između nekoliko dana i nekoliko tjedana.

Studija izvedivosti je faza u kojoj se procjenjuje koliko je DSDM prikladan za dani projekt. Na temelju tipa projekta i organizacijske i ljudske strukture donosi se odluka da li ga se može koristiti ili ne. Također unutar ove faze se određuje koji su rizici s kojima se možemo suočiti ukoliko se odlučimo za DSDM. Shodno tome pripremaju se izvještaj o izvedivosti te osnovni razvojni plan te se također i može napraviti brzi prototip ukoliko ima potrebe za time.

Poslovna studija je faza gdje se analiziraju bitne poslovne i tehnološke karakteristike. Pristup koji je preporučen je stvaranje radionica na kojima bi se okupio određeni broj eksperata koji bi potom razmotrili sve bitne dijelove sustava i ustanoviti koji bi bili prioriteti pri razvoju sustava. Svi poslovni procesi i korisničke klase su potom opisane u dokumentu pod nazivom Bussiness Area Definition. Unutar tog dokumenta detaljno su opisani poslovni procesi, a korisničke klase pomažu pri uključivanju korisnika u razvoj te također i omogućava da se ključni korisnici uključe u ranijim razdobljima razvoja proizvoda. Tijekom ove faze stvaraju se još dva dokumenta a to su Definicija Sistemske Arhitekture i Plan Prototipiranja. Definicija Sistemske Arhitekture predstavlja prvi prikaz sistemske arhitekture i može se mijenjati tijekom razvoja. Plan prototipiranja prikazuje strategiju kojom će se stvarati prototipovi tijekom sljedećih faza te također i plan za upravljanje konfiguracijom.

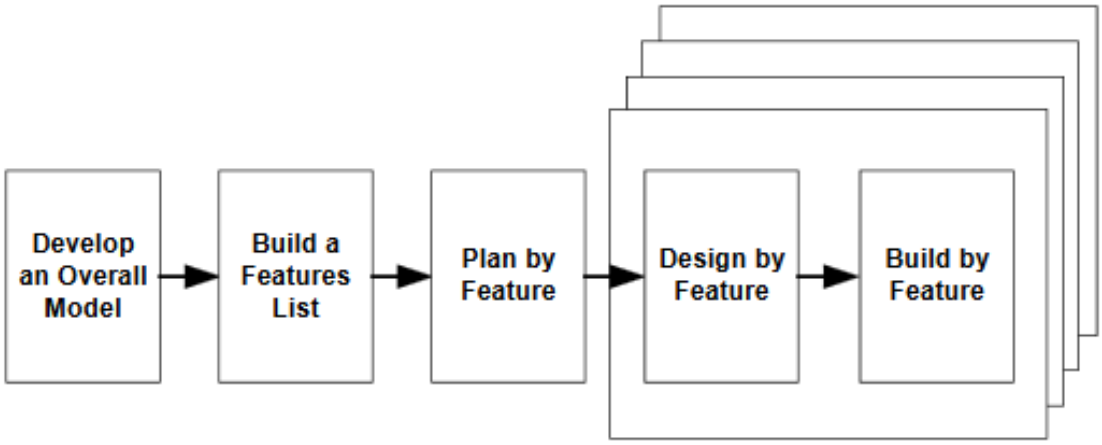
Sljedeća faza koja se izvršava je iteracija funkcionalnog modela te ona predstavlja prvu inkrementalnu i iterativnu fazu. Tijekom svake od iteracija sadržaji i način na koji će se iteracija izvršavati su isplanirani te se nakon što je iteracija izvršena rezultati se analiziraju kako bi pomogli sa idućim iteracijama. Prototipovi stvoreni tijekom ove faze se ne odbacuju već se zadržavaju i unaprjeđuju kako bi mogli biti uključeni unutar završne verzije sustava. Završni rezultat ove faze je funkcionalni model koji sadrži prototip koda i analize modela. Tijekom ove faze stvaraju se još četiri izlaza a oni su prioritizirane funkcije, dokumenti o funkcionalnosti prototipova, nefunkcionalni zahtjevi i analiza rizika za daljnji razvoj.

Faza nakon je dizajn i izgradnja iteracije i unutar ove faze se izgrađuje sustav. Rezultat ove faze je testirani sustav koji ispunjava sami minimum utvrđenih zahtjeva. Dizajn i funkcionalni prototip se potom provjerava od strane korisnika te se daljnji razvoj bazira na povratnoj informaciji koju oni daju.

Naposljetku se dolazi do faze završne implementacije gdje se sustav prebacuje iz razvojnog okruženja u stvarno radno okruženje. U ovoj fazi se treniraju korisnici kako bi znali raditi sa sustavom te im se postepeno daje sve veća kontrola nad sustavom. Izlazi ove faze su korisnički priručnik i izvještaj u kojem se nalazi pregled projekta. Taj izvještaj prikazuje koji su rezultati tog projekta te ovisno o uspješnosti u kojem smjeru će projekt dalje ići. Kod DSDM-a nalazimo četiri moguća tijeka projekta. Ukoliko je sustav ispunio sve zahtjeve nije potrebno dalje raditi na njemu, a ako je recimo otkriveno da poprilična količina zahtjeva nije ispunjena postoji mogućnost da će biti potrebno započeti rad na projektu potpuno nanovo. Ako je otkriveno da nedostaje neka manje kritična funkcionalnost može se započeti rad na projektu od faze iteracije funkcionalnog modela pa nadalje. Naposljetku ukoliko se neke tehničke poteškoće nisu stigle na vrijeme popraviti mogu se sada izvršiti na samom kraju započevši od faze iteracija dizajna i izgradnje.

1.2.4 Razvoj pogonjen mogućnostima

Razvoj pogonjen mogućnostima (engl. Feature Driven Development ili FDD skraćeno) je agilni i adaptivni objektno vođeni proces sa kratkim iteracijama. Prema Palmeru i Felsingu FDD pristup ne pokriva cijeli proces razvoj softvera već se radije fokusira na dizajn i faze izgradnje sustava ali bez obzira na to je dizajniran za rad sa ostalim aktivnostima softverskog razvoja i ne zahtjeva korištenje nikakvog specifičnog modela. Glavni cilj FDD pristupa je korištenje iterativnog razvoja zajedno sa praksama koje su smatrane najefektivnijima u industriji razvoja softvera. Ova metodologija se sastoji od pet sekvencijskih procesa i daje metode, tehnike i smjernice kako bi se uspješno dostavio sustav, FDD također uključuje uloge, artefakte, ciljeve i vremena potrebna za rad na projektu i prigodan je za razvoj kritičnih sustava (Palmer i Felsing 2002). Kod ove metode razvoja softvera imamo pet uzastopnih aktivnosti (slika 6) koje su jasno prikazane na sljedećoj slici te će u nastavku biti pojašnjene.



Slika 6 - Procesi FDD-a

Izvor: De Luca (2002).

Prvi proces je razvijanje općenitog modela. Kada taj proces započinje eksperti koji rade na tom projektu već su svjesni dosega, konteksta i zahtjeva za sustav koji je potrebno izgraditi. Eksperti zbog tog svog poznavanja zahtjeva sustava daju članovima razvojnog tima vodič koji će i njih također detaljno informirati o sustavu koji trebaju izgraditi. Sustav se potom dijeli na manje domene te se dodjeljuju timovi koji će raditi na pojedinoj domeni. Tim timovima se potom daje detaljni vodič o domeni na kojoj moraju raditi. Razvojni timovi potom razvijaju modele za pojedine domene te nakon rasprave odlučuju koji je model pogodan za koju domenu.

Sljedeći proces je stvaranje liste mogućnosti sustava, ta se lista stvara uz pomoć vodiča, modela i postojeće dokumentacije sa zahtjevima. Na tu listu razvojni tim stavlja funkcije koje klijenti najviše žele vidjeti, te su funkcije prezentirane tako da su podijeljene svaka na svoju domenu i sastoje se od glavnih mogućnosti koje bi trebale imati. Te su glavne mogućnosti još podijeljene na podmogućnosti i predstavljaju različite aktivnosti unutar svake od domena. Lista mogućnosti je naposljetku pregledana od strane korisnika i sponzora kako bi bila validirana.

Treći proces u nizu je planiranje prema mogućnosti. Ovaj proces uključuje stvaranje plana u kojemu su setovi mogućnosti poredani prema njihovom prioritetu i međusobnim ovisnostima. Također osim samih mogućnosti unutar tog plana im se dodjeljuje raspored i veći miljokazi.

Zadnja dva procesa na kojima se radi su dizajniranje prema mogućnosti i izgradnja prema mogućnosti. Tijekom ovog procesa biraju se male grupe mogućnosti iz prije napravljenog popisa te se formiraju timovi koji će raditi na tim mogućnostima. Ovi procesi su iterativne procedure tijekom kojih se odabrane mogućnosti razvijaju te svaka od tih iteracija može trajati između nekoliko dana i dva tjedna. Proces stvaranja iteracija sadrži aktivnosti poput inspekcija dizajna, kodiranja, testiranja, integracije i inspekcije koda. Nakon što je iteracija uspješno dovršena njene mogućnosti su dodane u glavnu verziju te započinje rad na idućoj iteraciji.

2. Kristalna obitelj metodologija

Sljedeća i zadnja agilna metodologija koja će biti obrađena u ovom radu je upravo kristalna obitelj metodologija. Naziv kristalna obitelj je ovdje zbog toga što se unutar nje nalazi cijeli niz različitih metodologija i svaka od njih je prikladna za neki drugi tip projekta te je zadatak razvojnog tima da odabere onu koja upravo njima najviše odgovara. Osoba koja je zaslužna za koncepciju kristalne obitelji metodologija je Alistair Cockburn koji je već spomenut u prijašnjim poglavljima ovoga rada kao jedan od začetnika agilnog načina razvoja softvera kakav danas znamo.

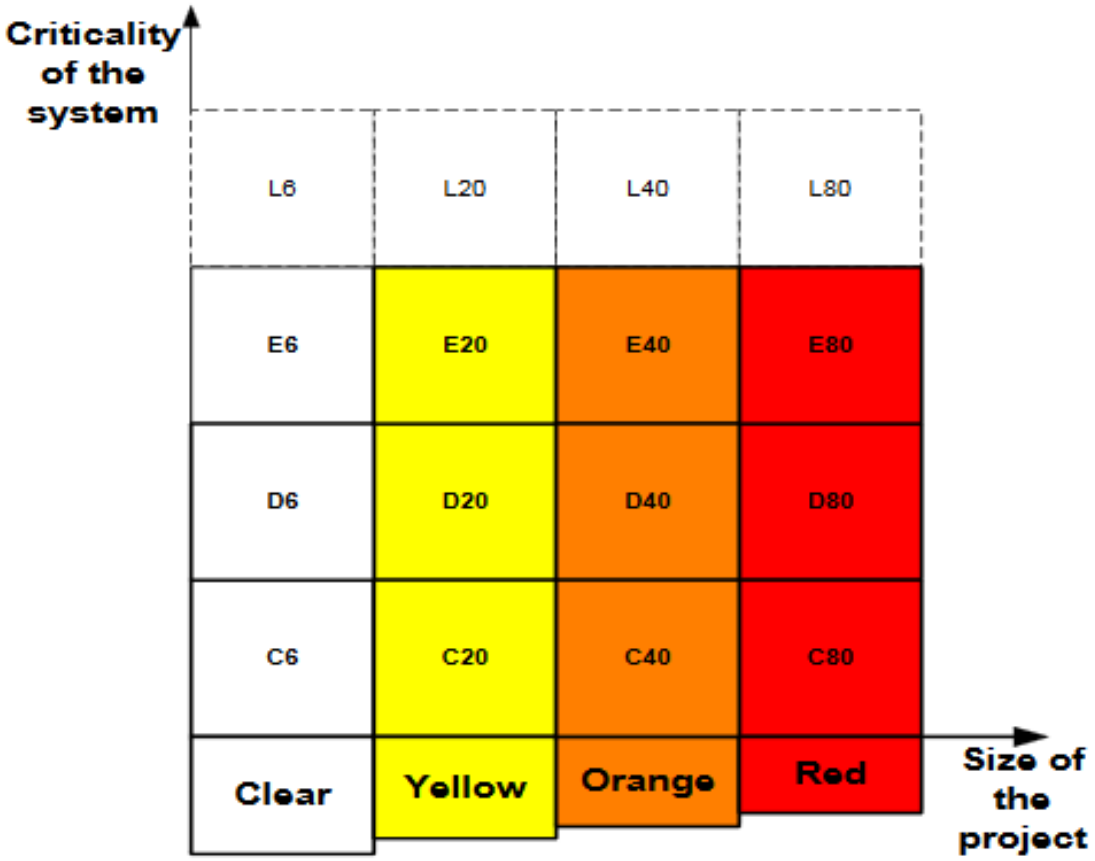
2.1 Nastanak metodologije

Prije nego što krenem sa detaljnim opisivanjem ove obitelji metodologija trebalo bi reći nekoliko riječi o osobi koja je za njih zaslužna. Alistair Cockburn je 1991. godine razvio metodologiju za efikasniji razvoj softvera, on naime smatra kako se jedna metodologija ne može primijeniti za svaku situaciju. Osim toga on smatra kako su metodologije centrirane oko ljudi puno bolje od metodologija koje su centrirane oko procesa te je upravo zbog toga razvio cijeli niz metodologija koje su 1998. godine postale poznate pod nazivom Crystal. Nedugo nakon toga Cockburn je zajedno sa Jim Highsmith-om započeo rad na cijelom serijalu knjiga pod nazivom *Agile Development Series* jedna od kojih je *Crystal Clear A Human-Powered Methodology For Small Teams* te će upravo ona predstavljati glavni izvor informacija za ovo cijelo poglavlje. U intervjuu iz 2001. godine koji je Cockburn dao za Cutter IT Journal se jasno može vidjeti zašto je takav pobornik agilnih metodologija te također zašto preferira metodologije koje su centrirane oko ljudi. Cockburn je naime u ranim 90-ima bio unajmljen od strane IBM-a kako bi stvorio i dokumentirao metodologiju za objektno orijentirani razvoj softvera. Njegov pristup tome zadatku bio je takav da je intervjuirao što više projekata i zapisivao što su mu timovi govorili o razlozima zašto su projekti uspjeli ili propali. Ono što je primijetio prilikom svog istraživanja bilo je to da su ljudi koji su međusobno surađivali i nisu se potpuno oslanjali na alate i dokumentaciju imali veći uspjeh nego ostali te su ti rezultati bili konzistentni sve od 1991. do 1999. koliko je on provodio ta istraživanja. Naposljetku je došao do zaključaka koji su mu omogućili da osmisli kristalnu obitelj metodologija. Glavni zaključak do kojega je došao bio je taj da ljudi koji rade na projektu trebaju više međusobno komunicirati licem u lice a manje preko papirologije (Cockburn 2001). Kao primjer tog zaključka naveo je primjer u

kojemu je jedna banka odlučila testirati tu teoriju te je u jednu prostoriju stavila tim od troje ljudi i ostavili su ih na miru sa zadatkom da osmisle novi sustav. Rezultat toga bio je da je taj tim dostavio željene rezultate u poprilično kratkom vremenu. Međutim ta metodologija ne funkcionira u svim slučajevima, Cockburn je naime primijetio da kako se broj zaposlenika na projektu povećava tako je potrebna sve veća organizacija i kontrola nad njima što na kraju znači da se prijašnja metodologija više ne može primijeniti te je potrebna nova. Upravo zahvaljujući tom zaključku razvijen je cijeli niz kristalnih metodologija za različite kompleksnosti projekata i različite količine ljudi koji rade na tim projektima, ali bez obzira na to koju od tih metodologija se odabere Cockburn naglašava kako je komunikacija i međusobno povjerenje ljudi koji rade na tom projektu od presudne važnosti.

2.2 Crystal

Dakle po čemu se kristalna obitelj metodologija razlikuje od svih ostalih koje su već navedene unutar ovoga rada, kako bi se dobio jednostavan odgovor na to pitanje dovoljno je pročitati kratki opis kojim ju je Cockburn karakterizirao. Taj opis je sljedeći: „Mnoge najbolje metodologije bivaju odbijene od strane razvojnih timova zbog toga što ih smatraju previše invanzivnima ili teškima. Kristalna metodologija ne pokušava biti najbolja već dostatna kako bi je razvojni tim mogao oblikovati i koristiti na način koji njima najviše odgovara“(Cockburn 1998a; 3). Naravno ta definicija jako ukratko opisuje kristalne metodologije jer ovdje ima još mnogo svojstava po kojima je ona drugačija i u nekim segmentima i zanimljivija od ostalih. Kristalna obitelj metodologija je cijeli skup različitih metodologija koje su dizajnirane za projekte različitih veličina i kompleksnosti te svaka organizacija i projektni tim koriste onu koja njima ponajviše odgovara u danom trenutku. Kristalna obitelj metodologija se zasniva na određenim principima i svojstvima i jedna od zanimljivijih stvari je to što ne zahtjeva korištenje nikakvih posebnih alata ili tehnika. Naziv kristal dolazi iz Cockburnove karakterizacije projekata prema veličini i njihovoj kritičnosti te im na osnovu toga dodjeljuje različite boje (slika 7) gdje su projekti za koje je potrebno više koordinacije i komunikacije obojani tamnijim bojama što je jasno prikazano na sljedećoj slici.



Slika 7 - Dimenzije kristalne metodologije

Izvor: Alistair Cockburn (2001).

Boje kojima su projekti okarakterizirani primarno ovise o broju ljudi koji na njima rade. Clear je namijenjen projektima sa osam ili manje ljudi, Yellow za timove između 10 i 20 ljudi, Orange za timove od 20 do 50 ljudi te Red za timove od 50 do 100 ljudi. Treba također i napomenuti kako ima još boja koje je Cockburn svrstao među kristalne metodologije poput Maroon, Blue, Violet i Crystal orange (Cockburn 1998b). Osim različitih boja kojima se opisuju projekti na slici su vidljive i kućice sa različitim slovima i brojevima. Te kućice simboliziraju potencijalne gubitke koji bi bili uzrokovani zakazivanjem sustava te slova unutar njih znače sljedeće: Udobnost (**C**omfort), Diskrecijski novac (**D**iscretionary money), Esencijalni novac (**E**ssential money) i Život (**L**ife). Dakle to znači ukoliko dođe do gubitaka karakteriziranim slovom C tada će doći do gubitka udobnosti osoba koje rade na tom projektu, dok ukoliko dođe do gubitaka karakteriziranim slovom E tada će projekti izgubiti novce koji su poprilično važni. Brojevi koji se nalaze uz svako od tih slova označavaju koliko najviše ljudi radi na tom

projektu. Sa svim tim poznatim informacijama moguće je pogledati dani graf te prepoznati o kakvom se tipu projekta radi, koliko ljudi rade na njemu te koji su njihovi potencijalni gubitci.

2.3 Kristalna svojstva

Cockburn najviše voli naglašavati kako se cijela njegova kristalna metodologija zasniva na sedam bitnih svojstava: česta dostava, bliska komunikacija, reflektivno unaprjeđenje, osobna sigurnost, fokus, tehničko okruženje sa automatiziranim testovima, upravljanjem konfiguracijom i čestim integracijama. Od svih tih svojstava česta dostava, bliska komunikacija i reflektivno unaprjeđenje predstavljaju temeljna svojstva koja se moraju primjenjivati na svakom od projekata. Pravilno upoznavanje i korištenje tih svojstava je jako važno ukoliko želimo primijeniti kristalnu metodologiju za projekt na kojemu radimo te će u nastavku svako od tih svojstava biti detaljnije pojašnjeno.

2.3.1 Česta dostava

Prema Cockburnu česta dostava predstavlja možda i najbitnije svojstvo od svih bez obzira na veličinu projekta ili metodologiju koja se koristi pri radu na njemu. Česta dostava omogućava sponzorima da dobiju važnu povratnu informaciju o statusu projekta, korisnici dobivaju mogućnost uvida da li je trenutno stanje proizvoda ono što su i tražili od razvojnog tima te im mogu dati vrijednu povratnu informaciju. Česta dostava također omogućava razvojnom timu da zadrži fokus te naposljetku razvojni tim može raditi na debugiranju svog razvojnog i procesa izdavanja proizvoda, posljedica čega može biti uzdizanje morala tima zahvaljujući svojim uspjesima. Pojam dostava znači da se softverski proizvod daje određenom setu korisnika na kraju svake od iteracija te ta dostava ne bi nikada trebala trajati više od četiri mjeseca jer se u tom slučaju gubi sigurnost projekta. Kod česte dostave može doći do problema da korisnici mogu smatrati kako previše često dobivaju nove verzije softvera što ih može iziritirati, a u drugu ruku neki korisnici bi mogli smatrati da ne dobivaju nove verzije softvera dovoljno često i to može ostaviti razvojni tim u nezavidnoj situaciji. Prema Cockburnu ponajbolje rješenje za takve situacije je da treba pronaći prijateljski raspoloženog korisnika koji bi testirao dani softver. Ta praksa omogućuje razvojnom timu da vježbaju

izdavanje proizvoda i dobiju vrijednu povratnu informaciju od korisnika koji taj proizvod testira. Kod korištenja česte dostave bitno je imati česte integracije koje bi se trebale odvijati u najboljem slučaju svakih sat vremena dok u najgorem bar jednom tjedno. S obzirom da kod česte dostave softverskog proizvoda korisnicima moraju postojati vremenski rokovi unutar kojih određena verzija proizvoda treba biti dovršena potrebno je primijeniti strategije uz pomoć kojih to postaje izvedivo. Vrijeme unutar kojega iteracija mora biti dovršena je uobičajeno čvrsto zacrtano te ga se ne bi smjelo produživati jer se to pokazalo kao loša strategija s obzirom da u tom slučaju dolazi do izmjene cijelog razvojnog rasporeda te ljudi koji rade na projektu počnu gubiti motivaciju s obzirom da ne vide rezultate svoga napornog rada. Bolja strategija koja se može primijeniti u ovom slučaju je dopuštanje timu da na kraju vremenskog roka izda ono što su do toga trenutka stigli napraviti. Ukoliko razvojni tim ipak ne stiže izbacivati novu verziju proizvoda korisnicima svakih par mjeseci tada bi trebali organizirati da ti korisnici posjete razvojni tim i vide kako softver funkcionira kako bi mogli pružiti povratnu informaciju timu.

2.3.2 Reflektivno unaprjeđenje

Tijekom vremena unutar kojeg je Cockburn radio za IBM na intervjuiranju razvojnih timova kako bi razvio novu metodologiju on je došao do iznenađujućeg zaključka. Ukoliko je projekt u katastrofalnom stanju moguće ga je spasiti i pretvoriti u uspješan projekt ako se tim odluči sastati i raspraviti zbog čega je projekt u lošem stanju te ustanoviti kako bi ga se moglo popraviti i naposljetku i implementirati potrebne promjene. Razvojni tim ne mora trošiti velike količine vremena na ovu aktivnost već je dovoljno da se sastanu svakih nekoliko tjedana te ustanove koje se aspekte projekta može poboljšati i kako. S obzirom da se tijekom trajanja projekta mnogo stvari može promijeniti od ljudstva, tehnologija koje se koriste pa sve do načina na koji se na projektu radi potrebno je da se ljudi koji rade na projektu sastanu te rasprave u kojem smjeru bi bilo najbolje da nastave. Ono što kristalna metodologija omogućava je to da razvojni timovi mogu mijenjati načine na koji pristupaju projektu, tako recimo mogu isprobati programiranje u paru, testiranje modula, razvoj pogonjen testiranjem te mnoge druge metode rada koje bi im mogle odgovarati u danom stadiju razvoja proizvoda.

2.3.3 Osmotska komunikacija

Osmotska komunikacija znači da članovi tima dobivaju određene informacije nesvjesno dok rade na nekom svom zadatku. Način na koji se to obično postiže je takav da se članove tima posjedne u istu prostoriju, te onda kada jedna osoba postavi neko pitanje ostali članovi tima mogu ili saslušati razgovor koji može uslijediti ili nastaviti sa poslom koji su radili i slušati razgovor u obliku pozadinske buke. Osmotska komunikacija omogućava da se pitanja i odgovori među članovima tima prirodno izmjenjuju sa minimalnim ometanjem članova koji nešto rade, u kombinaciji sa svojstvom česte dostave ovdje dolazi do velike količine povratnih informacija od kojih projekt uvelike profitira. Od osmotske komunikacije najveću korist imaju mali timovi zbog toga što im ona omogućava da postignu blisku komunikaciju, koja predstavlja jedno od glavnih svojstava kristalne metodologije. Na taj način članovi tima mogu naučiti nove tehnike od svojih kolega koje im omogućavaju da brže rješavaju probleme sa kojima se mogu susresti pri radu na projektu. Osmotska komunikacija je također izvediva i na većim projektima, ali ju je potrebno pomno organizirati jer ona postaje sve teža za izvesti kako se broj ljudi koji radi na projektu povećava. Kako bi se osmotska komunikacija mogla što efektivnije provesti kristalna metodologija nalaže kako članovi tima trebaju imati svoja radna mjesta što bliže jedan drugome te se soba u kojoj oni rade treba konfigurirati na način da to bude izvedivo.

2.3.4 Osobna sigurnost

Osobna sigurnost znači da osoba može nešto reći ili postaviti neko pitanje bez straha da će ga netko zbog toga uvrijediti. Primjer toga je govorenje šefu kako vremenski rok nije realan ili kolegi da ne odrađuje svoj dio posla dovoljno efektivno. Ovo svojstvo je važno zbog toga što omogućava timu da otkrije slabosti te ih efektivno i u što kraćem roku popravi. Osobna sigurnost omogućava do dolaska još jedne bitne karakteristike među članovima tima, a to je povjerenje. Ukoliko si članovi tima međusobno vjeruju i imaju jednostavnu i bogatu komunikaciju to im drastično povećava produktivnost te samim time i projekt profitira od toga. Svojstvo osobne sigurnosti usko je povezano sa svojstvima česte dostave i reflektivnog unaprjeđenja. Kada je softver dostavljen na vrijeme ljudi prepoznaju tko je zaslužan za to te ih to jača kao tim, dok je poveznica sa reflektivnim unaprjeđenjem ta da će tijekom sastanaka slobodnije govoriti te će to dovoditi do toga da će ti sastanci biti produktivniji. Glavni rizik kod ovoga

svojstva je taj da neki timovi pomiješaju osobnu sigurnost sa pristojnosti što znači da se na prvi pogled čini da prakticiraju ovo svojstvo, ali su zapravo samo pristojni kako bi izbjegli konflikt sa ostalim članovima tima.

2.3.5 Fokus

Fokus je svojstvo koje omogućava da osoba jasno zna na čemu treba raditi te potom da ima dovoljno vremena i mentalne snage za uspješan rad na tome. Poznavanje potrebnog posla uglavnom dolazi od glavnog sponzora projekta koji ima određene zahtjeve na kojima treba raditi. Vrijeme i mentalna snaga pak dolaze iz radne okoline unutar koje ljudi nisu odvojeni od zadatka na kojem rade kako bi radili na nekom drugom zadatku s kojim nisu upoznati. Neke od strategija kojima se ljudi na projektima bave problematikom toga tko što treba raditi mogu biti sljedeće: dizajniranje detaljnog plana kojim projekt treba teći, izdavanje specifičnih zadataka za svakoga od članova razvojnog tima ili recimo dizajniranje izjave od misiji projekta koja se nalazi na javnom mjestu kako bi joj svi članovi tima imali pristup. Čak i kad znaju na čemu točno trebaju raditi developeri se susreću sa idućim problemom a to je činjenica da trebaju izvještavati glavne osobe na projektu o statusu projekta. Ti sastanci zahtijevaju da developer prestane raditi svoj posao te da otiđe na sastanak kako bi mogao dati svoj izvještaj čime efektivno gubi fokus koji mu je prijeko potreban za rad, taj problem se sve više povećava ukoliko developer radi na nekoliko različitih projekata te mu samo izvještavanje o njima oduzima popriličnu količinu vremena koje bi bilo bolje iskorišteno za rad na projektu. Kako bi se izbjegla ova problematika developer ne bi smio raditi na više od jednog i pola projekta, developer bi trebao imati određeno vrijeme pri kojemu može raditi na projekt prije nego što ga se može prebaciti na drugi projekt te naposljetku razvojni tim bi trebao imati određene dijelove radnog dana kada ih se ne smije ometati kako bi se mogli u potpunosti fokusirati na posao. Ukoliko su svi ti uvjeti ispunjeni osobe koje rade na projektu ne bi trebale imati problema s fokusom te bi trebali biti efikasniji u poslu koji rade.

2.3.6 Pristup ekspertnim korisnicima

Konstantan pristup ekspertnim korisnicima omogućava razvojnom timu mogućnost da prakticiraju i testiraju česte dostave proizvoda, brzu povratnu

informaciju o kvaliteti proizvoda, brzu povratnu informaciju o njihovim odlukama o dizajnu i najnovije zahtjeve korisnika. Što više vremena razvojni tim može provesti sa ekspertnim korisnikom to će više projekt napredovati zahvaljujući njegovoj povratnoj informaciji, čak i jedan sat tjedno proveden s ekspertnim korisnikom može razvojnom timu biti od velike važnosti. Važna stvar kod ovog svojstva je vrijeme potrebno da razvojni tim dobije odgovor na svoja pitanja od korisnika, ukoliko je potrebno nekoliko dana da taj odgovor stigne programeri će najvjerojatnije sami procijeniti što bi trebalo učiniti, stoga bi uvijek trebalo imati telefonski pristup ekspertnom korisniku kako bi se takve situacije izbjegle. Najčešće metode rada sa ekspertnim korisnikom su sljedeće: tjedni ili polutjedni sastanci sa korisnikom uz mogućnost telefonskih poziva, postavljanje jednog ili više iskusnih korisnika unutar razvojnog tima i korištenje developera kao korisnika unutar određenog vremenskog perioda. Kao što je navedeno u potpoglavlju česta dostava, korisnička povratna informacija je od velike važnosti za uspješnost projekta i treba je shvatiti kao jedan od prioriteta.

2.3.7 Tehničko okruženje sa automatiziranim testovima, upravljanjem konfiguracijom i čestim integracijama

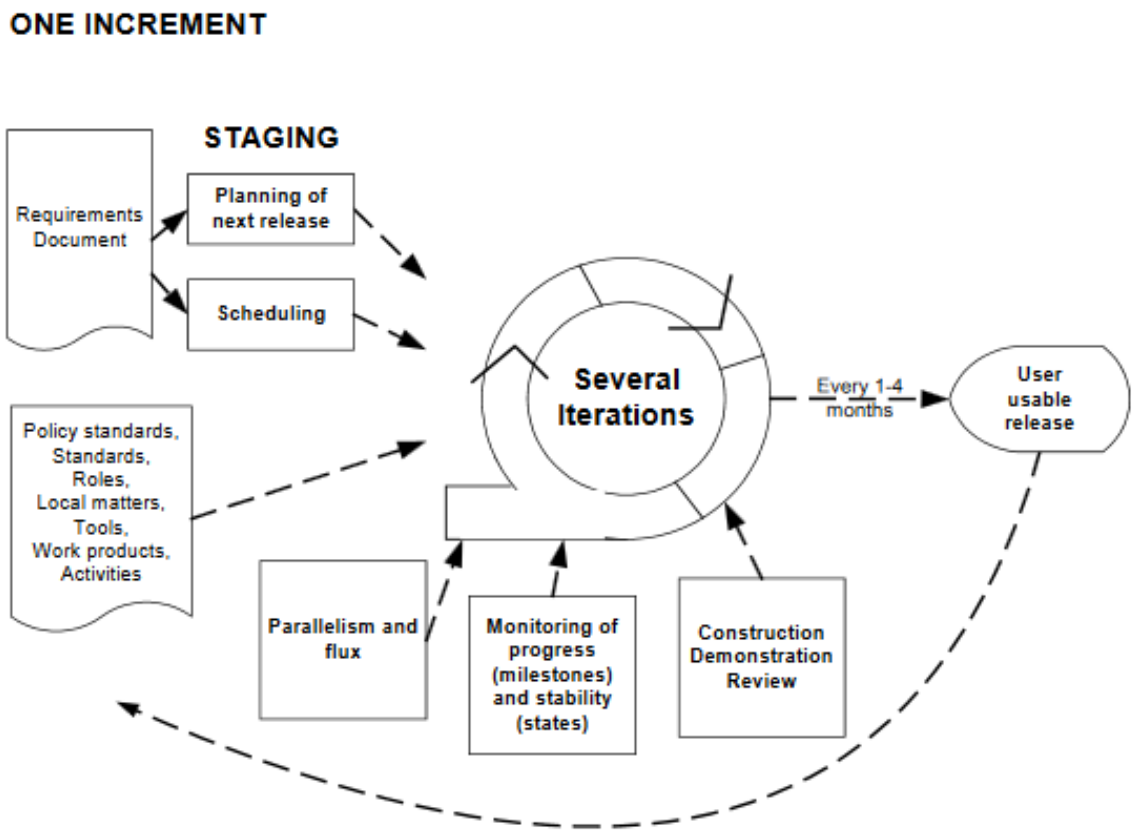
Ovo svojstvo predstavlja osnovne aktivnosti koje se moraju provoditi tijekom razvoja nekoga softverskog proizvoda. Automatizirano testiranje znači da osoba koja je zadužena za provođenje testova ne mora biti prisutna dok se oni obavljaju već može otići raditi na nekom drugom poslu te kada se vrati naići će na gotove rezultate, stoga ne čudi da osobe koje su jednom provodile ručne testove se nisu vratile na njih nakon što su jednom isprobali automatizirano testiranje. Glavna stvar koju programeri vole kod automatiziranog testiranja je to što im omogućava da provjere da li im kod još uvijek funkcionira kako treba nakon što su ga nadogradili sa novijim kodom. Sljedeća stavka je upravljanje konfiguracijom te ona omogućuje developerima da asinkrono provjeravaju svoje radove, naprave backupove ukoliko planiraju napraviti neku izmjenu postojeće verzije te da mogu pripremiti određenu verziju za izbacivanje i u slučaju da su neke promjene potrebne uvijek se na tu verziju mogu vratiti kako bi je popravili. Zadnja stavka je česta integracija te je mnogi timovi primjenjuju između nekoliko puta dnevno pa do jednom svaka dva dana. Česta integracija je bitna zbog toga što omogućava razvojnom timu da brže identificiraju i rješavaju greške prije nego što se one nagomilaju. Ponajbolji timovi koriste sve tri navedene stavke zajedno kako bi

primjenjivali konstantnu integraciju sa testiranjem što im omogućuje da primijete greške uzrokovane integracijom unutar nekoliko minuta.

2.4 Kristalni proces

Trenutno postoje tri glavne kristalne metodologije a to su Crystal Clear, Crystal Orange i Crystal Orange Web koji je usmjeren prema timova koji razvijaju web sadržaje. Od te tri metodologije samo prve dvije su uspješno primijenjene u praksi. Crystal Clear je primarno dizajniran za manje projekte klase D6 (slika 7), ali se uz neke modifikacije može primijeniti i za projekte klasificirane kao E8/D10. Crystal Orange je pak dizajniran za korištenje na projektima srednjih veličina koji broje između 10 i 40 ljudi to jest za kategoriju D40. Poput Crystal Clear-a i Crystal Orange može biti modificiran kako bi ga se koristilo za projekt koji ima višu klasifikaciju. Za razliku od Crystal Clear-a gdje je preporučljivo da svi članovi tima budu unutar iste prostorije tijekom rada kod Crystal Orange-a poslovi su podijeljeni na nekoliko timova sa kros funkcionalnim grupama. Obje metodologije moraju se pridržavati određenih pravila a to su dostava inkremenata na konstantnoj bazi, praćenje napretka miljokazima radije nego dokumentacijom, direktna uključenost korisnika, automatizirano testiranje funkcionalnosti i radionice vezane za proizvod i modificiranje metodologije na početku i sredinom svakog od inkremenata. Glavna razlika između ove dvije metodologije je to što kod Crystal Clear-a inkrementi trebaju biti isporučeni unutar dva mjeseca dok kod Crystal Orange se taj rok može produžiti na četiri mjeseca. Prema Cockburnu ove politike za rad sa Crystal su obavezne ali mogu biti zamijenjene sa adekvatnim praksama iz drugih metodologija poput recimo ekstremnog programiranja ili scrum-a. Osnovni alati koji su potrebni za rad sa ove dvije metodologije se razlikuju. Crystal Clear zahtjeva kompajler, alat za izradu verzija i upravljanje konfiguracijom te bijele ploče koje zamijenjuju formalnu dokumentaciju. Kod Crystal Orange potrebni alati su oni koji se koriste za verzioniranje, programiranje, testiranje, komunikaciju, praćenje statusa projekt, crtanje i mjerenje performansi.

Sve kristalne metodologije uključuju cijeli niz praksi, poput na primjer inkrementalnog razvoja. Kod Crystal Orange-a svaki inkrementi uključuje cijeli niz različitih aktivnosti a one su podjeljene na pripremu, nadgledavanje, pregledavanje te paralelizam i tok aktivnosti, kao što je vidljivo na slici koja slijedi (slika 8), te će one u nastavku biti malo detaljnije pojašnjene.



Slika 8 - Jedan inkrement Crystal Orange-a

Izvor: Wiki.amachu (2013b).

Pripremanje uključuje planiranje idućeg inkrementa sustava te bi on trebao biti zakazan za izlazak svaka tri do četiri mjeseca. Tim u ovoj fazi odabire koji će od zahtjeva biti implementirani unutar inkrementa te zadaju rok do kojega misle da ga mogu dovršiti.

Pregledavanje je bitno zbog toga što svaki od inkremenata uključuje nekoliko različitih iteracija te svaka od njih mora uključivati sljedeće aktivnosti: konstrukciju, demonstraciju i pregled ciljeva inkrementa.

Cilj nadgledanja je ustanovljavanje napretka kojim tim radi tijekom razvojnog procesa. Taj napredak se iskazuje miljokazima i stadijima stabilnosti koji mogu biti jako nepostojani, nepostojani ili dovoljno stabilni kako bi ih se moglo pregledati. Osim kod Crystal Orange nadgledavanje je potrebno i kod Crystal Clear.

Paralelizam i tok aktivnost je dio kada je proizvod ocijenjen kao dovoljno za stabilan za pregled. Kod Crystal Orange to znači da će nekoliko timova nastaviti uspješan rad na proizvodu sa maksimalnim paralelizmom. Kako bi se osigurala ta uspješnost timovi zaduženi za nadgledavanje i arhitekturu pregledavaju svoje radne planove, stabilnost i sinkronizaciju između timova.

Ova četiri pojma predstavljaju glavne aktivnosti unutar jednog inkrementa, ali osim njih može se dodati još neke poput tehnika za modificiranje korištene metodologije uz pomoć raznih intervjua i seminara čiji je cilj poboljšavanje trenutno korištene metodologije.

2.4 Prepoznavanje kristala

Ukoliko se razvojni tim odluči za korištenje neke od kristalnih metodologija moraju naučiti prepoznati da li ih pravilno koriste. To znači da moraju poznavati osnovne principe, pravila i svojstva ovih metodologija te preispitati da li ih se pridržavaju kako bi bili što uspješniji pri radu na svom projektu.

2.4.1 Znakovi pridržavanja

Prvi od znakova da se razvojni tim pridržava pravila ove obitelji metodologija je taj da članovi tima imaju dobru i zdravu međusobnu komunikaciju te je to potpomognuto činjenicom da članovi tima svi sjede jedan blizu drugoga bez da to izaziva iritaciju. Idući pokazatelj je činjenica da se inkrementi uspješno isporučuju unutar dva mjeseca te se njihov napredak prati miljokazima vezanima za efektivnost koda umjesto miljokazima vezanima za dokumentaciju. Treći pokazatelj pridržavanja je da razvojni tim blisko surađuje s korisnicima koji pomažu u izgradnji softverskog proizvoda testiranjem i kvalitetnom povratnom informacijom. Iduća stavka je činjenica da se ima jasna vizija koji je cilj danog projekta tako što su poznati svi zahtjevi te je samim time i poznati opis sustava kojega treba izgraditi. Zadnji pokazatelj korištenja ove metodologije je poznavanje koja osoba je zadužena za koji od modula sustava u izgradnji to jest treba biti poznato koja osoba ili tim mogu raditi kritične izmjene na kojemu od modula (Cockburn 1998a).

2.4.2 Znakovi nepridržavanja

Znakovi nepridržavanja predstavljaju potpunu suprotnost pokazateljima iz prethodne cjeline. Prvi od tih znakova je ako članovi razvojnog tima rade u različitim zgradama ili čak gradovima. Posljedica toga je da komunikacija među članovima tima ispašta zbog toga što ne mogu komunicirati licem u lice što je jedna od glavnih karakteristika kristalnih metodologija. Drugi pokazatelj je ako za izbacivanje inkremenata timu treba više od četiri mjeseca što uzrokuje manje dobivenih povratnih informacije čime daljnji razvoj i evolucija proizvoda stagniraju. Treći pokazatelj je činjenica da nema korisničkog pregledavanja proizvoda prije njegova izbacivanja što rezultira time da će korisnici vjerojatno dobiti proizvod koji ne odgovara njihovim potrebama. Zadnji pokazatelj kojega Cockburn navodi je to da nema bliske suradnje sa korisnicima. Posljedica ovoga je slična kao i za prethodnu stavku a to je da korisnički zahtjevi dolaze preko nekakvog posrednika koji misli da zna što kupci žele vidjeti i to na kraju dovodi do proizvoda koji ponovno nije ono što su korisnici željeli i potraživali (Cockburn 1998a).

2.5 Primjer primjene Crystal-a

U svojoj knjizi *Surviving Object-Oriented Projects* Cockburn opisuje jednu od uspješnih primjena svoje kristalne metodologije na projektu pod nazivom Project Winfried. Project Winfried naletio je na poteškoće već prilikom svog prvog inkrementa, unutar projektnih timova bilo je problema sa komunikacijom, projekt je imao dugu fazu utvrđivanja zahtjeva zbog koje je dizajniranje arhitekture sustava bilo odgođeno na nekoliko mjeseci te općenito ljudi uključeni na projektu nisu znali koji su njihovi zadatci. Ukratko cijeli projekt je bio u stadiju potpunog kaosa. Kako bi se pokušali izvući iz ove očajne situacije odlučeno je da će pokušati primijeniti Crystal Orange metodologiju. Prvo od rješenja bilo je usvajanje iterativnog procesa za svaki od inkremenata što je omogućilo timovima da otkriju svoje slabosti te da se reorganiziraju. Iteracije su također uključivale preglede funkcionalnih mogućnosti sa korisnicima što im je omogućilo da ustanove završne zahtjeve proizvoda i samim time postignu blisku suradnju sa korisnicima za koje je taj proizvod i namijenjen. Uspjeh sa prvim inkrementom motivirao je timove da se fokusiraju na jasniju podjelu poslova i bolju međusobnu komunikaciju (Cockburn 1998b). Na kraju projekt je izvučen iz propasti

uglavnom zahvaljujući inkrementima koji su omogućili timu da poprave razvojni proces i da postignu blisku suradnju sa korisnicima.

2.6 Crystal i ostale metodologije

Zbog principa i pravila koja su zacrtana unutar kristalne obitelji metodologija one su veoma kompatibilne sa ostalim metodologija te upravo i sam Cockburn nalaže kako ukoliko ima potrebe za time razvojni timovi bi trebali kombinirati metodologije poput scrum-a i ekstremnog programiranja sa njegovom metodologijom. Upravo ga je ekstremno programiranje bilo ugodno iznenadilo kad je njegov koncept osmišljen. Naime kristalne metodologije su osmišljene da budu jednostavne za korištenje sa što manje dokumentacije i birokracije te je prema Cockburnu ekstremno programiranje svojom pojavom učinilo kristalne metodologije kompliciranijima. Ove dvije metodologije imaju popriličnu količinu sličnosti uz jednu bitnu razliku a to je činjenica da je ekstremno programiranje u formulu dodalo disciplinu što je utjecalo na metodologiju na takav način da se moglo izbaciti još više dokumentacije i birokracije discipliniranim akcijama. Dakle glavna razlika je ta da su kristalne metodologije dizajnirane tako da daju developerima maksimalni individualizam dok je ekstremno programiranje bazirano na tome da svi moraju pratiti točno određene prakse (Alistair Cockburn 2001). Na kraju povezanost između ove dvije metodologije je takva da ih se može koristiti naizmjenično na način da tim koji je započeo radeći sa kristalnim metodologijama može napredovati na ekstremno programiranje dok tim koji nije imao uspjeha sa ekstremnim programiranjem se uvijek može vratiti na kristalne metodologije. Osim usporedbe sa ekstremnim programiranjem postoje i usporedbe sa scrum-om. Makan Taghavi Dilamani tako u svom radu *A short review on Crystal Clear methodology and its advantages over scrum, the popular software process model* uspoređuje te dvije metodologije. Unutar tog rada on hvali osnovne principe kojima se vode kristalne metodologije poput česte dostave, reflektivnog unaprjeđenja, bliske i osmotske komunikacije te je zaključak njegovog rada taj da ukoliko projekti imaju poteškoće pri radu sa scrum-om trebali bi se prebaciti na kristalne metodologije zbog toga što će im to poboljšati šanse za uspješno dovršavanje projekta (Dilamani 2014).

Zaključak

Kristalna obitelj metodologija je zamišljena od strane Alistaira Cockburna kao član perolake kategorije unutar obitelji agilnih metoda razvoja softverskog proizvoda. To je postigao tako što je iz metodologije izbacio svu nepotrebnu dokumentaciju i fokusirao je oko ljudi koji na tom projektu rade. Uz pomoć sedam osnovnih svojstava koji su opisani unutar ovoga rada sa naglaskom na komunikaciju između članova tima ova obitelj metodologija se ističe od ostalih svojim pristupima te predstavlja jako dobru alternativu ostalim metodologijama i upravo zbog svog dizajna se može koristiti kao potpora drugim metodologijama ili se one mogu koristiti kao potpora kristalnoj obitelji. Kao što primjer opisan u prijašnjem poglavlju pokazuje, ova metodologija je odlično rješenje ukoliko se projekt nađe u stanju konstantnog propadanja te ona može ponuditi prijeko potrebno osvježenje. Okretanje prema ljudima umjesto prema procesima i dokumentaciji je specifičnost koja je istovremeno prednost, ali može i biti nedostatak ove metodologije. Kako bi projekt koji koristi ovu metodologiju bio uspješan treba pronaći članove tima koji su voljni prihvatiti i koristiti metode koje su ovdje propisane. Ukoliko su te metode uspješno prihvaćene i implementirane od strane svih uključenih to dovodi do veoma uspješnih projekata kao što je i spomenuto u prije navedenom primjeru. Kao i uvijek u svakom procesu koji sadrži ljude kao osnovu puno toga može poći po zlu s obzirom da su ljudi jedna jako nepredvidiva varijabla. Ukoliko među članovima tima ima onih koji nisu timski igrači te odbijaju surađivati s ostalima to jest ukoliko odbijaju bilo kakve oblike suradnje tada ova metodologija jednostavno nije primjenjiva pa ili treba promijeniti problematične članove tima ili pronaći novu metodologiju koja bi bila prikladnija za rad na tom projektu. Osim prikladnog ljudstva potrebno je pronaći i/ili dizajnirati prikladan prostor za rad upravo zbog fokusa ove metodologije na ljudsku interakciju i zbog toga ljudi koji rade na projektu moraju biti udobno smješteni jedni blizu drugih. Također treba napomenuti kako korištenje ove metodologije postaje sve zamršenije što je veći broj ljudi koji radi na projektu jer je u tom slučaju potrebno pomno planiranje i njihova organizacija kako bi se principi ove metodologije uspješno primjenjivali. Na kraju treba se reći kako bi mali timovi koji su zainteresirani za razvoj softvera trebali iskušati ovu obitelj metodologija i prema potrebi u kombinaciji s njom prema svome izboru koristiti i ostale metode i metodologije koje smatraju potrebnima za uspješno dovršavanje svoga projekta.

Literatura

Abrahamsson. P, Salo. O, Ronkainen. J, Warsta. J. (2002). Agile software development methods. Universitiy of Oulu

Alistair Cockburn. (2001). Crystal light methods.

URL: http://alistair.cockburn.us/Crystal+light+methods [Pristupljeno 18. rujna 2016.]

Alistair Cockburn. (2010). 7 properties of Highly Successful Projects from Crystal Clear

URL: http://alistair.cockburn.us/7+Properties+of+Highly+Successful+Projects+from+Crystal+Clear [Pristupljeno 18. rujna 2016.]

Cockburn, A. (2002). Agile Software Development. Boston, Addison-Wesley.

Cockburn, A. (1998b). Surviving Object-Oriented Projects. Addison Wesley Longman

Cockburn, A. (2004). Crystal Clear A Human-Powered Methodology For Small Teams, Including The Seven Properties of Effective Software Projects.

Cockburn, A. (1998a). Crystal Clear A Human Powered Methodology for Small Teams. Agile Software Devlopment Series

Codesource.(1998).TheWaterfall Lifecycle Model and its Derivatives.

URL: http://codecourse.sourceforge.net/materials/The-Waterfall-Lifecycle-Model.html [Pristupljeno 19. rujna 2016.]

Dilamani, M., T. (2014). A short review on Crystal Clear methodology and its advantages over scrum, the popular software process model.

URL: http://ispe-usa.com/docs/images/icam2014_submission_29.pdf [Pristupljeno 18. rujna 2016.]

De Luca.(2002). FDD process diagram.

URL: https://www.researchgate.net/figure/30040436_fig1_Figure-61-FDD-process-diagram-De-Luca-2002 [Pristupljeno 19. rujna 2016.]

Hneif,M., Ow, S., H. (2009). Review of Agile Methodologies in Software Deveopment URL: http://www.arpapress.com/Volumes/Vol1/IJRRAS_1_01.pdf [Pristupljeno 18. rujna 2016.]

Manger, R. (2013). Softversko inženjerstvo skripta. Sveučilište u Zagrebu, Prirodoslovno Matematički fakultet, Matematički odsjek

Mohamed Sami.(2015). Software Development Life Cycle Models and Methodologies

URL:https://melsatar.wordpress.com/2012/03/15/software-development-life-cycle-models-and-methodologies/ [Pristupljeno 19. rujna 2016.]

Palmer, S., R., Felsing, J., M. (2002). A practical Guide to Feature-Driven Development. Prentice Hall PTR, Upper Saddle River

Schwaber, K. and Beedle, M. (2002). Agile Software Development With Scrum. Upper Saddle River, NJ, Prentice-Hall.

Schwaber, K. (1996). SCRUM Development Process. Advanced Development Methods. URL: http://www.jeffsutherland.org/oopsla/schwapub.pdf [Pristupljeno 19. rujna 2016.]

SEI Blog.(2013). Using v models for testing.

URL: https://insights.sei.cmu.edu/sei_blog/2013/11/using-v-models-for-testing.html [Pristupljeno 19. rujna 2016.]

Software engineering Project. (2004). Software Project Managment: Methodologies and Techiques. Department of Mathematics & Computer Science Technische Universiteit Eindhoven. URL: http://paul.luon.net/essays/SEP-essay-final.pdf [Pristupljeno 18. rujna 2016.]

System Analyst's blog.(2010). DSDM.

URL: https://systemanalyst1.wordpress.com/2010/12/05/dsdm/ [Pristupljeno 19. rujna 2016.]

VersionOne.(2016).What Is Agile Mehtodology?

URL: https://www.versionone.com/agile-101/agile-methodologies/ [Pristupljeno 18. rujna 2016.]

Williams, L. (2007). A Survey of Agile Development Methodologies.

URL: http://agile.csc.ncsu.edu/SEMaterials/AgileMethods.pdf [Pristupljeno 19. rujna 2016.]

Williams, L. and Cockburn, A.(2003). Agile Software Development: Its about Feedback and Change.

URL:
https://cs.anu.edu.au/courses/comp3120/local_docs/readings/WilliamsAndCockburn_AgileSoftwareDevelopment_ItsAboutFeedbackAndChange.pdf [Pristupljeno 19. rujna 2016.]

Wiki.amachu. (2013a).Life Cycle of a xp process.

URL:
http://www.wiki.amachu.in/index.php?title=File:Life_Cycle_of_a_XP_Process.png [Pristupljeno 19. rujna 2016.]

wiki.amachu.(2013b). One Crystal Orange Increment.

URL:
http://www.wiki.amachu.in/index.php?title=File:One_Crystal_Orange_Increment.png [Pristupljeno 19. rujna 2016.]

Popis slika:

Sažetak

Tema ovoga rada je kristalna obitelj metodologija za razvoj softverskog proizvoda te će osim njih unutar ovoga rada biti navedene i opisane klasične i agilne metodologije. Za obije od tih glavnih metodologija biti će navedene i opisane najpopularnije metode te je svrha njihovog navođenja da se vide njihove međusobne sličnosti i razlike, a samim time i njihov odnos sa kristalnom obitelji metodologija. Nakon djela koji se bavi navođenjem i opisivanjem tih metodologija i njihovih pripadnih metoda fokus će biti prebačen na kristalnu obitelj metodologija. Unutar tog djela rada kristalna obitelj metodologija će biti detaljno opisana to jest biti će navedena njena svojstva, principi, osnovne značajke i aktivnosti koje je potrebno provoditi ukoliko se odlučimo za rad s njima pri razvoju softverskog proizvoda.

Ključne riječi

Agilne metodologije, Aktivnosti, Inkrementi, Iteracije, Komunikacija, Kristalna obitelj metodologija, Procesi, Sedam svojstava, Upravljanje projektom

Summary

The theme of this paper is Crystal Family of Methodologies for Software Development and within it there will be descriptions of agile and classical methodologies. For both of those main methodologies will be listed and described their most popular methods. The point of listing those methods is to both see their likeness to each other and their differences and in the end their relationship with the crystal family of methodologies. After that part of this paper the focus will be shifted towards the crystal family of methodologies which is after all the main topic of this paper. Within that chapter of the paper the crystal family of methodologies will be described in detail by listing its properties, principles, practices and activities which are required to be used and enforced while developing a software product if we are using this family of methodologies.

Key words

Agile methodologies, Activities, Increments, Iterations, Communication, Crystal family of methodologies, Processes, Seven properties, Project management