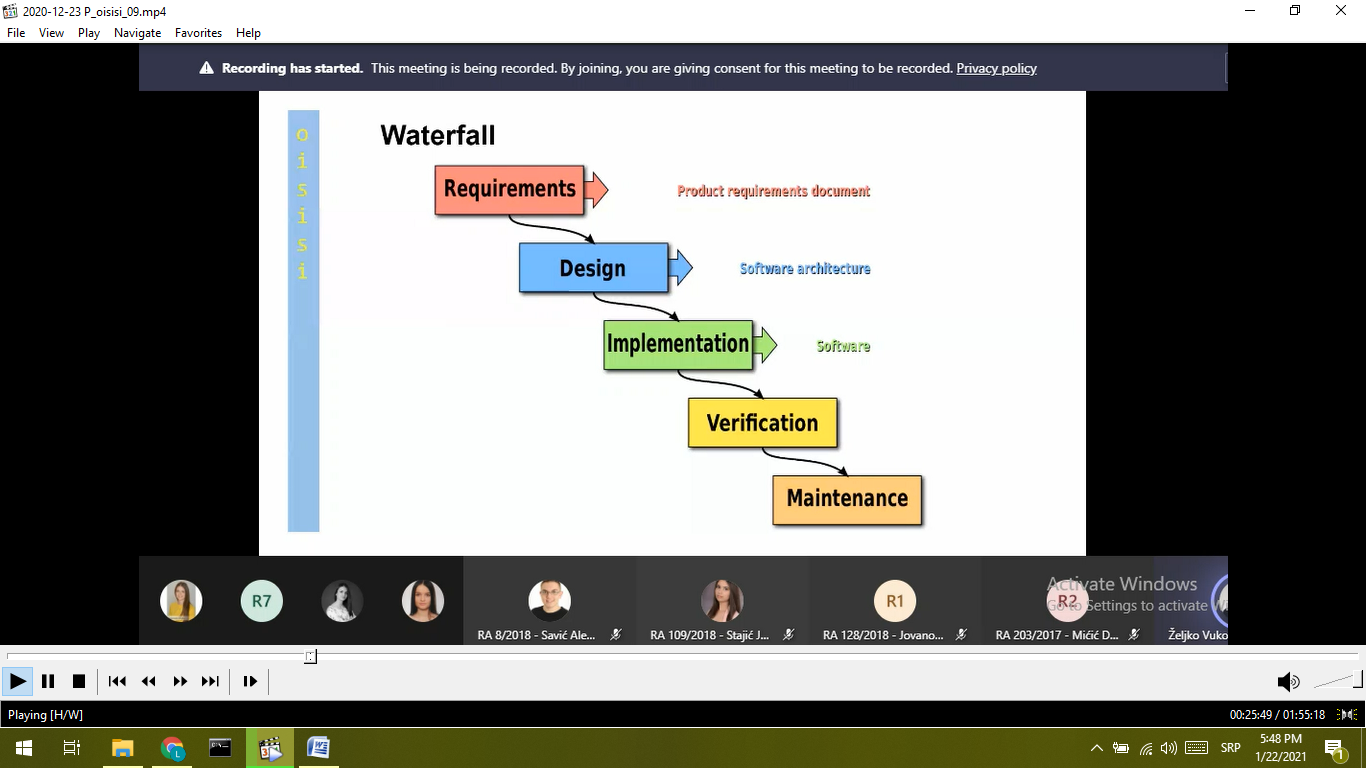
***Modeli životnog ciklusa razvoja softvera***

***RAD BEZ METODOLOGIJE***

Može se primenjivati na jednostavnim primerima, kada ***nije potrebna organizacija ljudi*** koji rade na datom projektu. Međutim ukoliko se neki projekat i realizuje na ovaj način i nastane želja za njegovim daljim ***unapređivanjem ili proširenjem*** onda ipak moramo uložiti malo više truda i ***pristupiti razvoju*** na neki sistematičan način.

***WATERFALL MODEL***

Model vodopada predpostavlja ***postojanje niza faza*** u razvoju softvera. Te faze obuhvataju:

* definisanje zahteva
* dizajn softvera,
* kreiranje softverske arhitekture,
* implementacija arhitekture,
* verifikacija
* uvođenja u ekspolataciju
* održavanje softverskog proizvoda.

Ovaj način razvoja propisuje da ***rezultate*** ***svake*** od ovih ***faza*** treba dovesti do ***idealnih***. Ne prelazimo na sledeću fazu dok nismo zadovoljni onim što imamo u trenutnoj fazi, dok nismo odgovorili na sva postavljena pitanja i kroz sve prošli i proverili. To na čemu trenutno radimo dovodimo do ***savršenstva***. Razlog za to jeste pretpostavka da ukoliko ***pogrešimo u ranijoj fazi*** to nas može mnogo ***više koštati u kasnijim fazama***.

***Prednost***

Proces je sam po sebi prilično jednostavan i lak za primeniti. Sa upravljačke strane ovo se lako sprovodi u praksi. Lako je upravljačkom organu da ne dozvoli prelazak u sledeću fazu dok ne vidi konkretne i zadovoljavajuće rezulate.

Takođe, u jednom trenutku se radi samo na jednoj fazi, zbog čega ljudi mogu mnogo bolje da i kvalitetnije mogu da obavljaju svoj posao.

Rezultati modela vodopada su pokazali dobre rezultate za jednostavne projekte sa jasno definisanim zahtevima. Kada neko tačno zna šta hoće i kada je krajnji kvalitet veoma značajan.

***Problemi***

U realnom svetu preovladavaju ***kompleksni*** i ***nedovljno precizno definis***ani problemi. Najčešće korisnici, klijenti nisu sigurni šta je to što im treba.

Takođe, ovaj metod predviđa neki ***linijski razvoj softvera*** a sam proces razvoja softvera ***po prirodi nije tak***av. U praksi zahtevi se često menjaju javljaju se novi, nekada tek u fazi implementacije, u fazi kodiranja, postajemo svesni nekih problema koji nisu bili vidljivi prilikom prikupljanja zahteva.Ukoliko radimo po ovakvom pristupu i negde u toku projekta dodje do promene obima posla, pojave se novi zahteva, dolazi do pada ovakvog metoda, jer ovde ne postoji način da se vrati korak nazad i nešto izmeni ili dopuni.

Najveći problem jeste da ***korisnici ne vide ništa konkretno*** dok se čitav proces ne završi. U krajnjem slučaju može se desiti da to što dobiju nije ono što su želeli.

***MODIFIKOVANI WATERFALL***

Osnovna ideja jeste da se ***predvidjaju načni*** da se ipak ***vratimo korak u nazad*** i da postoji način da, ukoliko dođemo do problema u nekoj fazi, da se vratimo u prethodnu fazu i ispravimo bar deo koji je problematičan.

Neki problemi su otklonjeni ovim metodom neki ublaženi, međutim i dalje imamo probleme kao što su šta: projekti za koje ***nismo sigurni šta je krajnji cilj*** koji želimo da postignemo, kako uključiti dodatne resurse u toku projekta ukoliko treba ***ranije*** da isporučimo proizvod ili se ***poveća obim posla***, ali najveći problem je što ***krajnji korisnici ne vide*** nikakav opipljiv ***rezultat*** sve do kraja procesa.

***MODIFIKOVANI WATERFALL SA PROTOTIPOM***

Na osnovu rezultata početnih faza konstruišemo nekakav ***prototip*** koji će moći da se koristi za to da se ***sa korisnicima usaglasimo*** da li je to to što su oni želi, nije nužno da to bude nešto što može da se izvršava na računaru. Prototip može da bude krajnje jednostavan, iscrtan na papiru,.. što će samo služiti da se bolje razumemo sa korisnicima.

***Prototip*** koji pravimo u procesu razvoja je nešto što će se na kraju ***baciti*** i što ne možemo da iskoristimo kao osnovu koju ćemo razraditi i dobiti gotov proizvod.

***V MODEL***

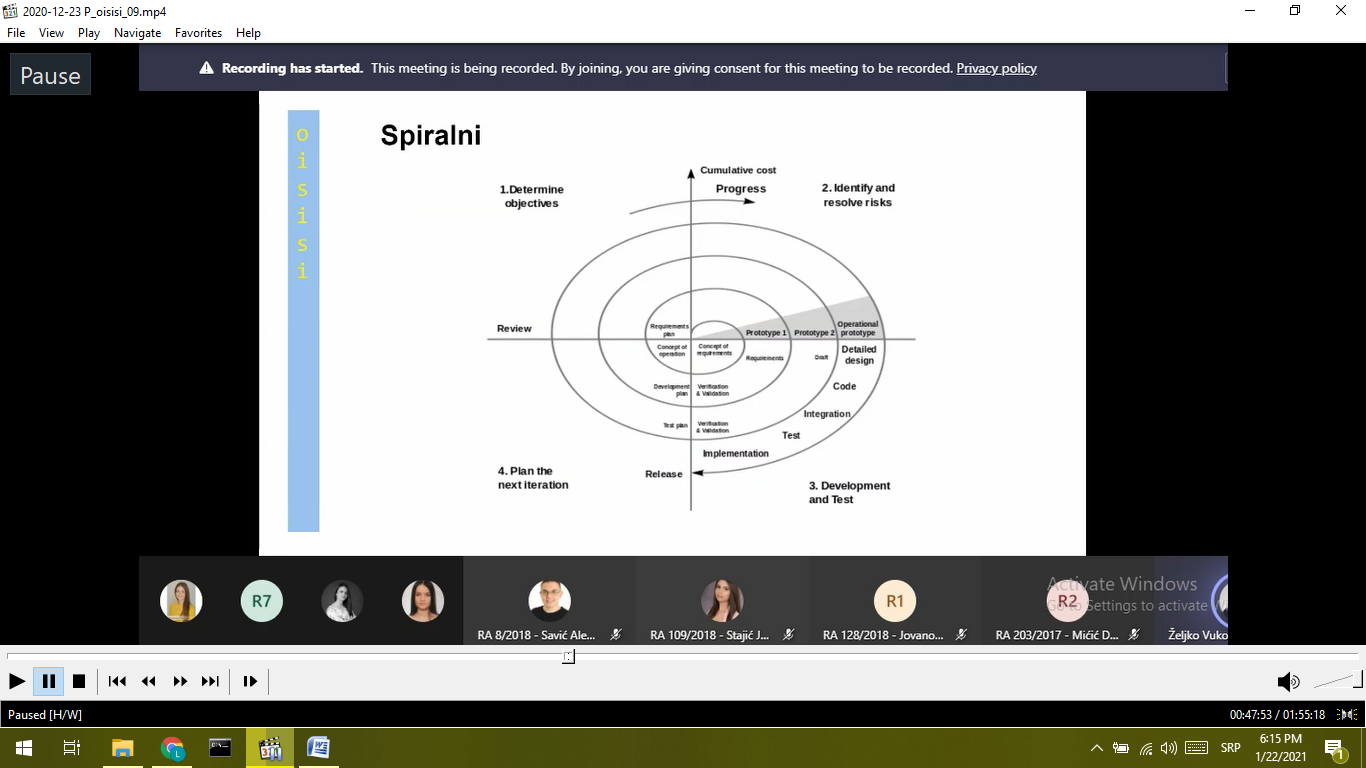
Predviđa neke drugačije poglede na to kako razvoj softvera treba da izgleda, ali u suštini ga prate neki ***slični problemi*** i izazovi kao i model vodopada.

***SPIRALNI MODEL***

***Top down*** pristup, koji predviđa ***kombinaciju*** ***incrementalnog i prototipskog razvoja***. Sam proces predviđa 4 faze koje se u toku razvoja ponavljaju,odvijaju se ciklično.

* Planiranje,
* Analiza rizika
* Realizacija
* Evaluacija

Ono što je ključno mesto ***oko čega se vrti*** spiralni model jesu ***rizici***. Ovaj model polazi od toga da rizike ne možemo eliminisati već ***njima moramo upravljati*** (promena zahteva, novi zahteva u toku razvoja, generalno promena obuhvata proizvoda, organizacioni izazovi komunikacije sa klijentima, unutar timova, mogućnost da u toku razvoja neki članovi tima odu,..) Ono čime se spiralni model bavi jeste da ***omogući da uočimo*** rizike i da njima ***upravljamo***.



Prvo se formulisu i razumevaju ciljevi sistema i proizvoda, što se odnosi na ***prikupljanje zahteva***. Sagledavamo sta nam je cilj i ***pronalazimo sve alternativne načine*** da dođemo do tih formulisanih ciljeva, bilo da smo na početku ili na sredini procesa jedna od mogućih stvari je da promenimo dizaj da počnemo ispočetka, da odustanemo od razvoja dela našeg rešenja da nabavimo nešto gotovo i slično. Za trenutne ciljeve razmatramo sve moguće načine da se taj cilj postigne.

U drugoj fazi ***analiziramo*** te alternativna rešenja, ***sagledavamo sve rizike*** koji prate svaki od tih pristupa, zatim radimo simulaciju, izradu prototipa, evaluaciju,.. Detaljno analiziramo svaki od identifikovanih pristupa nekako ***dokumentujemo*** i odlučimo koji pristup ćemo ***odabrati***.

Treća faza je ***implementacija*** gde se detaljnije razrađuje dizajn, kodira se integrise, testira i dobijamo jedan inkrement. Pri tome u toku ove faze možemo da se ***oslonimo na vodopad*** da bi implementirali neki ograničeni podskup celog projekta, jer je problem vodopada sa kompleksnim stvarim za koje nismo sigurni da li je to ono što želimo. Prethodne faze u spiralnom procesu nam obezbeđuje da samo rasčlanili sistem na podsisteme, da smo detaljno analizirali šta je to što nam je neophodno i onda kada imamo neki jasan zadatak tada nije problem da koristimo vodopad kao neki pristup da realizujemo ono što nam je u tom trenutku i cilj neke faze projekta.

Kada uspešno završimo razvoj i testiranje tog jednog dela prelazi se na ***planiranje sledeće iteracije*** i čitav proces se ponavlja dok ne dodjemo do krajnjeg cilja za čitav sistem.

***Prednosti***

Značajno je to što se ***uvodi analiza rizika***, što se oni prepoznaju i što se njima upravlja, dobro se pokazao kod složenih sistema, dobro je što je ***životni proizvod vidljiv*** već u ranim fazama jer je i ovde predviđena izgradnja prototipa.

***Mana***

Prilično je ***kompleksan***, ima ***visoke troškove***, neophodni su veliki resursi da bi ovo moglo da se sprovede, analiza rizika da bi se radila kako treba trazi da imamo na raspolaganju ***široka ekspertska znanja*** a sa druge strane, specifična za određene oblasti za koje analiziramo moguće alternative. Ovo nije odgovarajući pristup za male projekte.

***SCRUM***

***Iterativni inkrementalni*** okvir koji obezbeđuje podršku u upravljanju fazama softverskih projekata i baziran je na ***agilnim principima***. Osnovni pristup scrum je da se bazira na malom ***timu*** ljudi koji će razvijati softver. Tim treba da bude ***kompaktan*** i ***multifunkcionalan*** u čijem sastavu imamo sve ***ljude sa neophodnim znanjima*** da bi se u okviru tima neki zadatak realizovao.

U okviru ovog pristupa propisane su i određene uloge u okviru timova:

* ***Scrum master*** - osoba koja je zadužena za ***vođenje*** samog ***procesa***
* ***Product owner*** – osoba koja predstavlja krajnjeg korisnika, ***predstavlja klijenta*** u okviru samog tima, on je interfejs tima ka klijentima, tu je da komunicira sa klijentom i odgovara na pitanja tima vezanih za zahteve klijenta. Njegov zadatak je da pribavi odgovore na pitanja tima ili da na osnovu svog iskustva da svoje odgovore.
* ***Tim*** - 5 do 7 članova uključujuci članove sa posebnim ulogama. Ograničena veličina tima je tu da omogući bolju komunikaciju samog tima, da omogući kompaktniji rad, da se članovi tima dobro poznaju, da dobro poznaju specijalnosti jedni drugih, da znaju ko im može dati odgovor na koja pitanja što se tiče tehnlogija kojima se stvari implementiraju.

Ključni princim scrum-a je da u bilo kojoj fazi realizacije projekta, ***klijent može*** da se ***predomisli*** i po pitanju ***zahteva*** i kakvi su ***prioriteti*** zahteva, kao i kada će projekat da se završi. Ideja je da dosta toga može da se menja u hodu. Da možemo po potrebi da uvodimo dodatne resurse itd.

***Product backlog***

Predstavlja ***skup funkcionalnosti*** koje su korisniku neophodne, obično se one definišu na nekom visokom nivou apstrakcije, svaka od njih se na neki način opisuje. Product backlog je ***otvoren svim*** interesnim stranama i timu za implementaciju i predstavnicima klijenta koji u bilo kom trenutku mogu da menjaju taj spisak želja da dodaju nove želje da menjaju prioritete odustanu od želja koje su imali,..

Vlasnik samog dokumenta je ***product owner*** i on je ***zadužen*** da ***procenu poslovne vrednosti*** pojedinih elemenata dok sa druge strane ***tim je zadužen za procenu napora*** koji je poreban da se svaki od tih elemenata product backlog-a realizuje.

***Sprint backlog***

Sadrži ***skup zadataka*** na kojima će tim raditi u okviru ***jednog sprinta***. Jedan sprint je neki period fikse dužine obično oko 30 dana. Na početku sprinta se iz product backloga na osnovu priroteta uzimaju funkcionalnosti i rasčlanjuju na neophodne zadatke da bi se funkcionalnosti mogle realizovati. Pri tome preporuka je da taj pojedinačni zadatak bude tog obima da jedan programer može da ga relizuje u periodu od 4 do 16 radnih sati. Kada se napravi sprint backlog dalje članovi tima sami biraju zadatke na kojima će raditi.

Sprint backlog je ***vlasništvo jednog tima***. U toku jednog sprinta ne mogu se dodavati nove stvari ne mogu se menjati zahtevi. Iako zahtevi na projektu mogu menjati ali da bi se omogućilo da se bar u okiru jednog sprinta obezbedi neka stabilnost i jasnoća šta je to što se trenutno radi, ***zahtevi se ne mogu menjati u toku sprinta***.

Obično se zadaci, zaduženja, predstojeći zadaci, urađeni zadaci,.. prikazuje na sprint tablama, gde se koriste reči kao: to do, in progres, done.

Ono što pomaže u pracenu napretka je ***burndown chart*** prikazuje ***ukupan napor*** na koji smo se opredelili na početku sprinta, svi zadaci, i to gde smo trenutno u toku sprinta po pitanju toga koliko vremena smo potrošili i ***šta je realizovano***. Pomoću njega možemo utvrditi koliko su ***procene napora*** koje je tim dao na početku sprinta ***realne***. Praćenje kvaliteta procene ima za cilj da mi naučimo bolje da procenjujemo. Kvalitet procene nije nesto na osnovu čega se meri nečija posvećenost, znanje, kvalitet procene je nešto što je povratna informacija samom timu.

***Rezultat*** jednog sprinta je nešto što treba da bude ***opipljiv i izvršiv inkrement*** u softveru koji razvijamo. Ideja je da ***nakon svakog sprinta*** bi praktično ***mogli da isporučimo*** do tada razvijen softver. Na taj način ***klijent*** može na kraju svakog spirnta da vidi šta je do tada urađeno da li nešto nije onako kako je očekivao, može da utiče na backlog i zahteve, njihov prioritet i na kraju da kaze da je to dosta da je time zadovoljan i preuzme softver u takvom stanju, i tim završava razvoj. Bolje je da projeket završimo tako što ćemo isporučiti deo funkcionalnosti nego da ne isporučimo nista.

***Svaka iteracija*** sama po sebi ***podrazumeva*** one standardne ***faze vodopada***. U svakoj od ovih iteracija se radi analiza zahteva tima sto se gleda product backlog, u svakom sprintu se radi dizajn time sto se ti zahtevi razdvajaju na zadatke, zatim implementaciju, testiranje i na kraju svakog sprinta se radi isporuka i uvođenje u eksploataciju.

Ovo je jos jedna način da se ***waterfall svede na ono za sta je dobar*** a da se rad sa celokupnim softverskim pojektom na neki način drugačije organizuje u nešto što je pogondiji proces za razvoju kompletnog softvera.

***Ograničavajući faktor*** je ***veličina tima*** od 5 do 7 ljudi. Ovim uslovom ograničena je i kompleksnost projekta koji može da realizuje tako mali tim. U situacijama kada imamo veoma kompleksne projekte kao rešenje se predlaže da se napravi organizacija ***„scrum of scrums“***. Ideja jeste da se čitav projekat podeli na delove koji će se realizovati u okviru jednog scrum tima, a na višem organizacionom nivou scrum masteri i product owner-i svih pojedinčanih scrum timova čine *scrum* *of scrums* i oni komuniciraju i razmenjuju informacije.

Takođe jedan od ***problema*** je i što ljudi vide scrum mastera i product ownere kao ***upravljačke funkcije*** iako one po ideji scrum-a to nisu. Oni su članovi tima koji imaju posebne uloge. Bez toga šta oni radi ne bi bio moguć čitav razvoj ali jednako ne bi bio moguć razvoj ni bez ostalih članova tima.

***RATIONAL UNIFIED PROCESS (RUD)***

Rational je kompanija nije proces racionalan. Postoji niz alata za razvoj softvera koji je ova kompanija izbacila.

Zasniva se na ***skupu gradivnih elemenata*** koji opisuju šta je to što je neophodno proizvesti, koje su to veštine potrebne da bi se softverski proizvod napravio kao i korak po korak objašnjenim načinima na koje je moguće postići definisane ciljeve.

Tri glavna gradivna elementa ovog procesa su:

* ***Uloge*** - definišu skup povezanih znanja, veština, kompetencija i odgovornosti
* ***Proizvod*** - posledica izvršavanja zadataka, uključujući svu dokumentaciju, modele i kod implementacije
* ***Zadaci*** - koji predstavljaju jedinicu posla koja je dodeljena nekoj ulozi i koju treba da obezbedi neki proizvod

U sklopu svake iteracije zadaci su kategorizovani u 9 različitih ***disciplina***:

* Inženjerske discipline:
  + Modelovanje poslovanja
  + Zahtevi
  + Analiza i dizajn
  + Impelemntacija
  + Testiranje
  + Uvodjenje u ekspoloataciju
* Pomoćne discipline:
  + Rukovanje konfiguracijom i promenama
  + Upravljanje projektom
  + Formuliranje radnog okruzenja

Sam model životnog ciklusa predviđa 4 faze razvoja. Ove faze omogućavaju da se proces predstavi na ***visokom nivou apstrakcije***. Svaka faza rezultuje jednim ključnim ciljem i treba da realizuje jedan ključni element primopredaje, na osnovu koga se realizuje cilj koji je postavljen na početku.

***Faza sagledavanja*** (inception)

U ovoj fazi izrađuje se biznis model koji uključuje poslovni kontekst, faktore uspeha i finansijsku prognozu. Takođe generišu se osnovni model slučaja upotrebe, projektni plan, početna procena rizika i opis projekta (osnovni zahtevi projekta, ograničenja i ključne karakteristike).

Na kraju faze, projekat se proverava prema sledećim kriterijumima:

* Podudarnost zainteresovanih strana u definiciji obima i proceni troškova
* Razumevanje zahteva
* Verodostojnost procene troškova, prioriteta, rizika i procesa razvoja.
* Dubina i širina prototipa koji je razvijen.
* Uspostavljanje načina za upoređivanje stvarnih i planiranih troškova

Ako projekat ne pređe ovu prekretnicu (milestone), može se otkazati ili ponoviti nakon što je redizajniran da bolje zadovolji kriterijume.

***Faza razrade(elaboration)***

Faza razrade je mesto gde ***projekat*** počinje da se ***oblikuje***. U ovoj fazi se vrši **analiza domena problema i arhitektura projek*ta*** dobija svoj osnovni oblik.

Ishod faze razrade je:

* Model slučaja upotrebe
* Opis softverske arhitekture
* Izvršna arhitektura koja realizuje značajne slučajeve upotrebe
* Poslovni slučajevi i lista rizika koji su revidirani
* Razvojni plan za celokupan projekat
* Prototipovi koji evidentno umanjuju svaki identifikovani tehnički rizik

Ova faza ***mora proći kriterijume prekretnice*** arhitekture životnog ciklusa (da li je proizvod stavilan, da li su rizici rešeni, detalji izgradnje razrađeni,..)

Ako projekat ne može da pređe ovu prekretnicu, još uvek ima vremena da se otkaže ili redizajnira. Međutim, nakon napuštanja ove faze, projekat prelazi u operaciju visokog rizika gde su promene mnogo teže i štetnije kada se izvrše.

***Faza izgradnje***

Primarni cilj je ***izgradnja softverskog sistema***. U ovoj fazi glavni fokus je na ***razvoju komponenata*** i drugim ***karakteristikama*** sistema. Ovo je faza kada se odvija većina ***kodiranja***. U većim projektima može se razviti nekoliko iteracija konstrukcije u nastojanju da se slučajevi upotrebe podele na upravljačke segmente i tako naprave demonstrativne prototipove.

***Prelazna faza***

Primarni cilj je „tranzit“ sistema iz razvoja ***u proizvodnju***, čineći ga krajnjim korisnikom dostupnim i razumljivim. Aktivnosti u ovoj fazi uključuju ***obuku*** krajnjih korisnika i održavača i ***beta testiranje*** sistema kako bi se validirao prema očekivanjima krajnjih korisnika. Sistem takođe prolazi kroz ***fazu ocenjivanja***, svaki programer koji ne izvodi traženi rad se zamenjuje ili uklanja. Proizvod se takođe proverava prema nivou kvaliteta postavljenom u početnoj fazi.

***Šest najboljih praksi***

* Razvijajti ***iterativno***
* Upravljati ***zahtevima***
* Koristite ***komponente*** - Razbijanje naprednog projekta nije samo predloženo, već je zapravo i neizbežno. Ovo promoviše sposobnost testiranja pojedinačnih komponenti pre nego što se integrišu u veći sistem. Takođe, ponovna upotreba koda je veliki plus i može se lakše postići upotrebom objektno orijentisanog programiranja.
* Modelirati ***vizuelno*** - Pomoću dijagrama predstaviti sve glavne komponente, korisnike i njihovu interakciju („UML“).
* Proveriti ***kvalitet*** - Uvek napraviti testiranje glavnim delom projekta u bilo kom trenutku. Testiranje postaje teže kako projekat napreduje, ali bi trebalo da bude stalni faktor u stvaranju bilo kog softverskog proizvoda.
* ***Kontrolne*** promene - Mnogo projekata kreiraju mnogi timovi, ponekad na različitim lokacijama, mogu se koristiti različite platforme itd. Kao rezultat toga, neophodno je osigurati da se promene napravljene u sistemu konstantno sinhronizuju i verifikuju.

***RAPID-APPLICATION DEVELOPMENT (RAD)***

Opšti pojam za adaptivni pristup razvoju softvera. RAD pristupi razvoju softvera stavljaju ***manje naglaska na planiranje***, a više na adaptivni proces. ***Prototipi*** se često koriste kao ***dodatak*** ili ponekad čak i ***umesto*** projektnih ***specifikacija***.

RAD je posebno pogodan za razvoj softvera koji zahteva ***korisnički interfejs***.

RAD deli proces u 4 faze.

***Faza planiranja zahteva***

Korisnici, menadžeri i članovi IT osoblja razgovaraju i dogovaraju se o poslovnim potrebama, obimu projekta, ograničenjima i sistemskim zahtevima. Završava se kada se tim složi oko ključnih pitanja i dobije ovlašćenje uprave da nastavi.

***Faza korisničkog dizajna***

Korisnici komuniciraju sa analitičarima sistema i razvijaju modele i prototipove koji predstavljaju sve sistemske procese, ulaze i izlaze. RAD grupe ili podgrupe obično koriste kombinaciju tehnika zajedničkog razvoja aplikacija i CASE alata za prevođenje potreba korisnika u radne modele. Korisnički dizajn je kontinuirani interaktivni proces koji omogućava korisnicima da razumeju, modifikuju i na kraju odobre radni model sistema koji zadovoljava njihove potrebe.

***Faza izgradnje***

Fokus je razvoj programa i aplikacij. Međutim, u RAD-u korisnici nastavljaju da učestvuju i još uvek mogu da predlažu promene ili poboljšanja kako se razvijaju stvarni ekrani ili izveštaji. Zadaci ove faze su programiranje i razvoj aplikacija, kodiranje, integracija jedinica i testiranje sistema.

***Faza prekida***

Podrazumeva konverziju podataka, testiranje, prelazak na novi sistem i obuku korisnika. U poređenju sa tradicionalnim metodama, ceo proces je komprimovan (sabijen). Kao rezultat toga, novi sistem je izgrađen, isporučen i pušten u rad mnogo ranije.

***Prednosti***

***Bolji kvalitet***. Interakcijom korisnika sa prototipovima koji se razvijaju, poslovna funkcionalnost može biti mnogo veća od one koja se postiže modelom vodopada. ***Softver*** može biti ***upotrebljiviji*** i ima veće šanse da se usredsredi na poslovne probleme koji su ključni za krajnje korisnike umesto na tehničke probleme od interesa za programere. Međutim, ovo isključuje neke nefunkcionalnie zahteve (AKA ograničenja ili atributi kvaliteta), sigurnost i prenosivost.

***Kontrola rizika***. Veoma rano se može usredsrediti na ključne faktore rizika i prilagoditi im se na osnovu dokaza prikupljenih u ranom delu procesa.

***Više projekata je završeno na vreme i u okviru budžeta***. Fokusiranjem na razvoj inkrementalnih jedinica smanjuju se šanse za katastrofalne neuspehe kao kod metoda vodopada. Za razliku od metoda vodopada, do spoznaje sistema se dolazi mnogo ranije te se na neke probleme i može ranije reagovati.

***Mane***

**Rizik od novog pristupa**. Ljudi ne prijaju promene i bilo koji projekat koji se preduzme sa novim alatima ili metodama će verovatnije propasti prvi put samo zbog zahteva da tim nauči novi pristup.

***Nedostatak naglaska na nefunkcionalnim zahtevima***. Krajnjem korisniku često nisu vidljivi u normalnom radu.

***Zahteva vreme oskudnih resursa***. U modelu vodopada korisnici bi definisali zahteve, a zatim uglavnom odlazili dok su programeri kreirali sistem. Korisnici RAD-a uključeni su od početka i gotovo kroz čitav projekat. To zahteva da preduzeće bude spremno da uloži vreme stručnjaka za domen aplikacije. Paradoks je da što je bolji stručnjak, što su više upoznati sa svojim domenom, to oni više vremena provode stvarno obavljajući posao te ih je teže ubediti da ulože svoje vreme u razvoj softvera. Bez njih RAD projekti neće uspeti.

***Manje kontrole***. Idealno je biti u mogućnosti da se brzo prilagodite i problemima i mogućnostima. Neizbežna je trgovina između fleksibilnosti i kontrole, više jednog znači manje drugog. Ako projekat vrednuje kontrolu više od agilnosti RAD nije prikladan.

***Loš dizajn***. Fokus na prototipove u nekim slučajevima može biti odveden predaleko što rezultira metodologijom „hakovanja i testiranja“ gde programeri neprestano unose manje izmene u pojedinačne komponente i ignorišući probleme arhitekture sistema koji bi mogli rezultirati boljim ukupnim dizajnom.

***Nedostatak skalabilnosti***. RAD se obično fokusira na male do srednje velike projektne timove. Ostala gore navedena pitanja (manje dizajn i kontrola) predstavljaju posebne izazove kada se koristi RAD pristup za sisteme vrlo velikih razmera.

***EKSTEMNO PROGRAMIRANJE (XP)***

Ekstremno programiranje je metodologija razvoja softvera koja je namenjena poboljšanju kvaliteta softvera i reagovanju na promenljive zahteve kupaca. Kao vrsta agilnog razvoja softvera, zalaže se za česte *release*-ve projekta u kratkim razvojnim ciklusima, koji imaju za cilj poboljšanje produktivnosti i uvođenje kontrolnih tačaka na kojima se mogu usvojiti novi zahtevi kupaca.

Ostali elementi ekstremnog programiranja uključuju: programiranje u parovima ili opsežno pregledavanje koda, jedinstveno testiranje celokupnog koda, jednostavnost i jasnoća koda, očekivanje promena zahteva kupaca kako vreme prolazi a problem se bolje razume i česta komunikacija sa kupcem i među programerima.

XP pokušava da smanji troškove promena zahteva višestrukim kratkim razvojnim ciklusima, a ne dugim. U ovoj doktrini promene su prirodan, neizbežan i poželjan aspekt projekata razvoja softvera i za njih treba planirati, umesto da pokušavaju da definišu stabilan skup zahteva.

***Aktivnosti***

XP opisuje četiri osnovne aktivnosti koje se obavljaju u okviru procesa razvoja softvera:

***Kodiranje.*** Jedini zaista važan proizvod u procesu razvoja sistema kod. Bez koda nema radnog proizvoda. Kodiranje se može koristiti za pronalaženje najpogodnijeg rešenja. Kodiranje takođe može pomoći u prenošenju misli o programskim problemima. Kodeks je uvek jasan i sažet i ne može se tumačiti na više načina.

***Testiranje.*** Najvažnije za ekstremno programiranje. Pristup ekstremnog programiranja je da ako malo testiranja može ukloniti nekoliko nedostataka, puno testiranja može ukloniti još više nedostataka. Svaki komad koda koji je napisan testira se pre prelaska na sledeću funkciju.Testiranje integracije širom sistema, u početku, je podstaknuto, za rano otkrivanje nekompatibilnih interfejsa, da se pojedinačni zadaci ponovo povežu pre nego što se se sistem rasčlani na veće delove. Međutim, sistemsko testiranje integracije svedeno je na nedeljne ili ređe, u zavisnosti od stabilnosti ukupnih interfejsa u sistemu.

***Slušanje.*** Programeri moraju da slušaju šta kupcima treba da bi sistem radio, koja „poslovna logika“ je potrebna. Moraju da razumeju ove potrebe dovoljno dobro da kupcu daju povratne informacije o tehničkim aspektima kako se problem može rešiti, ili ne može rešiti.

***Projektovanje.*** Ako se prethodne aktivnosti dobro obavljaju, rezultat bi uvek trebalo da bude sistem koji funkcioniše. Mešutim u praksi, sistem nekon nekog vremena postaje previše složen i zavisnosti unutar sistema prestaju da budu jasne. To se može izbeći stvaranjem dizajnerske strukture koja organizuje logiku u sistemu. Dobar dizajn će izbeći mnoge zavisnosti unutar sistema; to znači da promena jednog dela sistema neće uticati na ostale delove sistema.

***Vrednosti***

Ekstremno programiranje je prepoznalo 5 vrednosti:

***Komunikacija.*** U formalnim metodologijama razvoja softvera, saopštavanje sistemskih zahteva programerima sistema se postiže ***dokumentacijom***. Cilj je pružiti svim programerima zajednički pogled na sistem koji se podudara sa prikazom korisnika sistema. U tu svrhu, ekstremno programiranje favorizuje jednostavne dizajne, uobičajene metafore, saradnju korisnika i programera, čestu verbalnu komunikaciju i povratne informacije.

***Jednostavnost.*** Ekstremno programiranje podstiče započinjanje sa najjednostavnijim rešenjem. Kasnije se mogu dodatni nove funkcionalnosti. Razlika između ovog pristupa i konvencionalnijih metoda razvoja sistema je fokus na dizajniranju i kodiranju za današnje potrebe umesto za sutrašnje, sledeće nedelje ili sledećeg meseca. Nedostatak jeste što to ponekad sutra zahteva više napora za promenu sistema, ali je to više nego nadoknađeno prednošću ne investiranja u moguće buduće zahteve koji bi se mogli promeniti pre nego što postanu relevantni. Kodiranje i dizajniranje za neizvesne buduće zahteve podrazumeva rizik trošenja resursa na nešto što možda neće biti potrebno, a možda i odlaganje ključnih karakteristika. Jednostavnost dizajna i kodiranja treba da poboljša kvalitet komunikacije. Većina programera u timu lako može da razume jednostavan dizajn sa vrlo jednostavnim kodom.

***Povratna informacija***

* Povratne informacije iz sistema: pisanjem i pokretanjem testova programeri imaju direktne povratne informacije o stanju sistema nakon primene promena.
* Povratne informacije kupca: Funkcionalne testove (odnosno testove prihvatanja) pišu kupac i testeri. Dobiće konkretne povratne informacije o trenutnom stanju svog sistema. Ovaj pregled se planira jednom u dve ili tri nedelje, tako da kupac može lako da upravlja razvojem.
* Povratne informacije od tima: Kada kupci postave nove zahteve u igri planiranja, tim direktno daje procenu vremena koje će mu biti potrebno za primenu.
* Povratne informacije su usko povezane sa komunikacijom i jednostavnošću. Mane u sistemu lako se saopštavaju pisanjem jedinstvenog testa koji dokazuje da će se određeni deo koda pokvariti. Direktne povratne informacije iz sistema govore programerima da prekodiraju ovaj deo. Kupac može periodično da testira sistem u skladu sa funkcionalnim zahtevima, poznatim kao korisničke priče.

***Hrabrost.*** Uvek dizajniramo i kodiramo za danas, a ne za sutra. Omogućava programerima da se osećaju prijatno kada prepravljaju svoj kod kada je to potrebno. Baciti kod kada je to potrebano.Upornos potrositi ceo dan na resavanje i ipak ga resiti sutradan.

***Poštovanje.*** Poštovanje drugih kao i samopoštovanje. Programeri nikada ne bi smeli da izvršavaju promene koje prekidaju kompilaciju, čine postojeće unit-testove neuspešnim ili koje na drugi način odlažu rad svojih vršnjaka.

***Kontraverzni aspekti***

* Ovim metodom proces postaje fleksibilan i štedi formalne troškove,ali takođe može dovesti do skupe prerade i povećanja obima projekta što nije ono što je prethodno dogovoreno ili finansirano.
* Postoje potencijalni sukobi u ciljevima projekta i ograničenja između više korisnika. Metoda XP zavisi od toga da li programeri mogu da zauzmu jedinstveno gledište klijenta, tako da se mogu koncentrisati na kodiranje, a ne na dokumentovanje kompromisnih ciljeva i ograničenja.
* Zahtevi su izraženi kao automatizovani testovi prihvatanja, a ne kao specifikacioni dokumenti.
* Zahtevi se definišu postepeno, umesto da se pokušaju dobiti unapred.
* Od programera se obično zahteva da rade u parovima.
* Nestabilni zahtevi, nema dokumentovanih kompromisa korisničkih sukoba i nedostatak sveukupne specifikacije dizajna ili dokumenta.