Softverska kriza

Evidentirana: Oktobar 1968, Oktobar 1969, NATO konferencija :

"Software Engineering techniques"

Ispoljava se kroz:

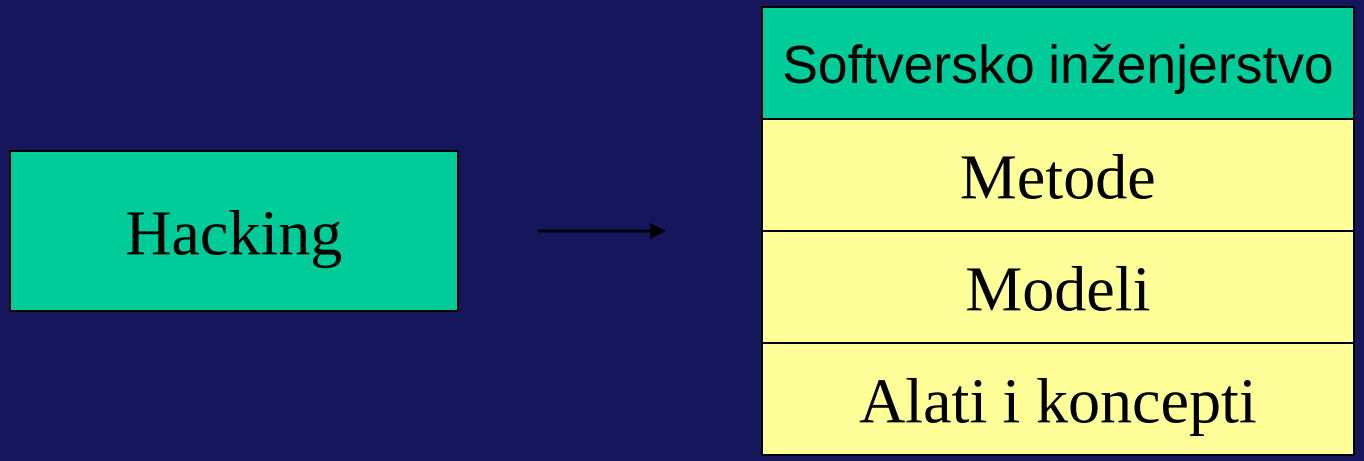
- Loš kvalitet softverskih projekata,
- Niska produktivnost,
- Cena razvoja softvera visoka,
- Dugo vreme realizacije softvera.

Povezano sa: nekoordiniranim i nestrukturnim programiranjem (hacking)

Uticala na: nova istraživanja i razvoj nove inženjerske discipline "Softversko inženjerstvo"

Izlaz

- Poboljšanje procesa proizvodnje softvera,
- Poboljšanje kvaliteta softvera,
- Intenzivan razvoj modula i sredstava za održavanje softvera,
- Intenzivan razvoj tehnika i metoda za razvoj i održavanje softvera



Rezultat

- Strukturno programiranje (1970)
- Modularno programiranje
- Rekurzivno programiranje
- Pojava CASE alata (1980)
- RAD okruženja
- Logičko programiranje
- Objektno orijentisano programiranje (OOP) (1990)
- Objektno orijentisano modelovanje
- Simboličko programiranje
- Vizuelno programiranje

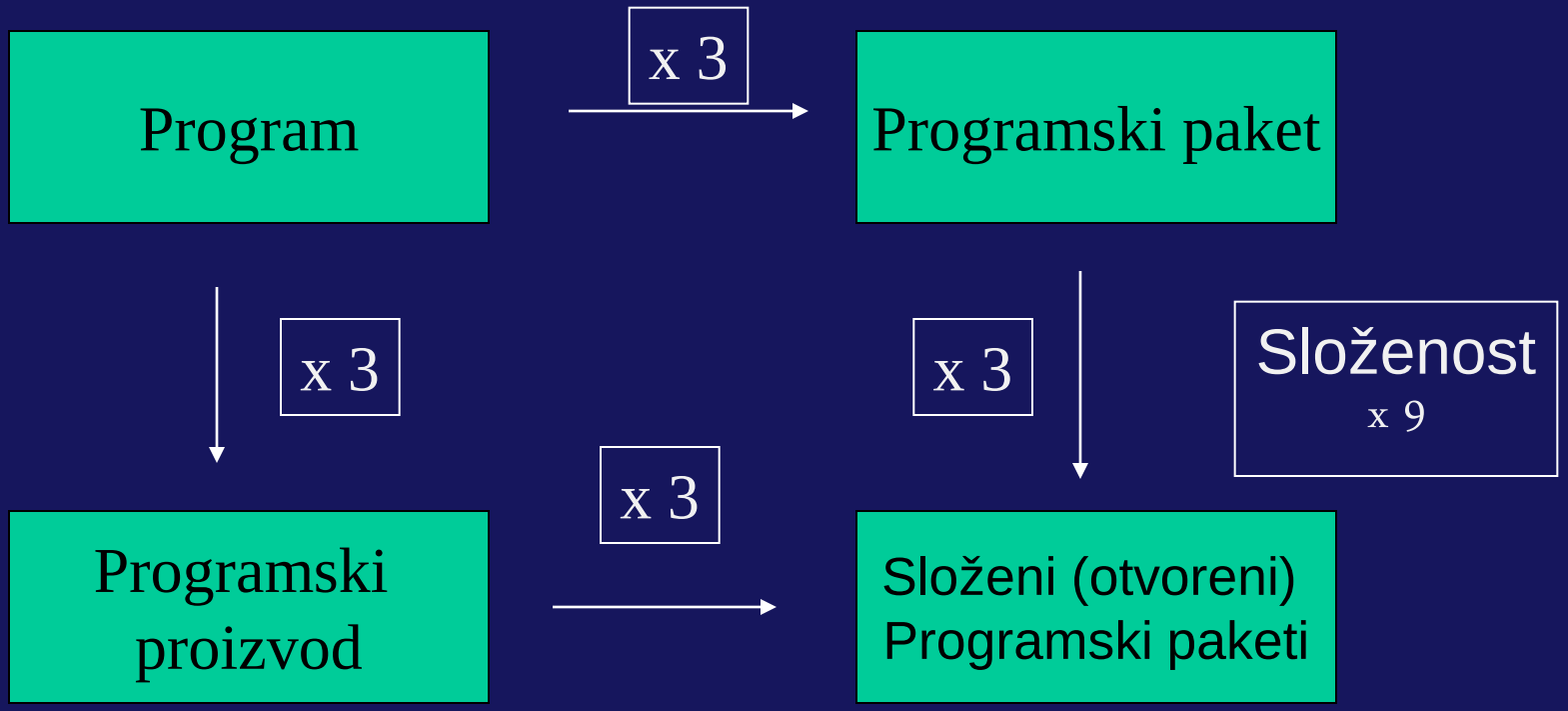
Osnovne paradigme (stilovi) programiranja

- Proceduralno-orijentisan (algoritmi)
- Objektno orijentisan (klase i objekti)
- Logički orijentisan (ciljevi, često izraženi predikatskim računom)
- Pravilima orijentisan (if-then pravila)
- Ograničenjima orijentisan (invarijantne relacije)

Objektno orijentisani stil

1) najbolje prilagodjen najširem skupu aplikacija
2) često služi kao radni okvir u kome se koriste drugi stilovi

Softverski proizvod - softver



Program:

- Razvija ga jedan programer
- Jedna celina
- Jedan proizvodjač - jedan korisnik: autor
- Upotrebljiv na autorovom sistemu za koji je razvijen
- primer: Adresar razvijen u Visual Basicu

Programski proizvod:

- Razvija ga tim
- Potpuno testiran i dobro dokumentovan
- Više pojedinačnih korisnika
- Specijalizovani za jedan tip posla
- Dostupni za različita okruženja
- primer: Microsoft Word

Programski paket:

- Sadrži više programa za različite poslove
- Komponente su funkcionalno povezane
- Dobro definisan interface izmedju komponenata
- Jedan kupac sa mnogo korisnika
- primer: Informacioni sistem jedne firme

Složeni programski paket:

Kombinuje osobine:

- Programa (kompletna celina)
- Programskog proizvoda (programi povezani u celinu, koordinirane komponente, više korisnika)
- Programskog paketa (razvija ga tim, potpuno testiran, radi na različitim platformama, više kupaca)

Kvalitet softvera

- Funkcionalnost
- Pouzdanost
- Efikasnost
- Odgovarajući korisnički interfejs
- Adekvatna dokumentacija
- Transparentnost
- Mogućnost lakog održavanja
- Adaptivnost

Cilj softverskog inženjerstva:

- Sa što manje napora doći do kvalitetnog i pouzdanog softvera
- Povećanje produktivnosti rada programera

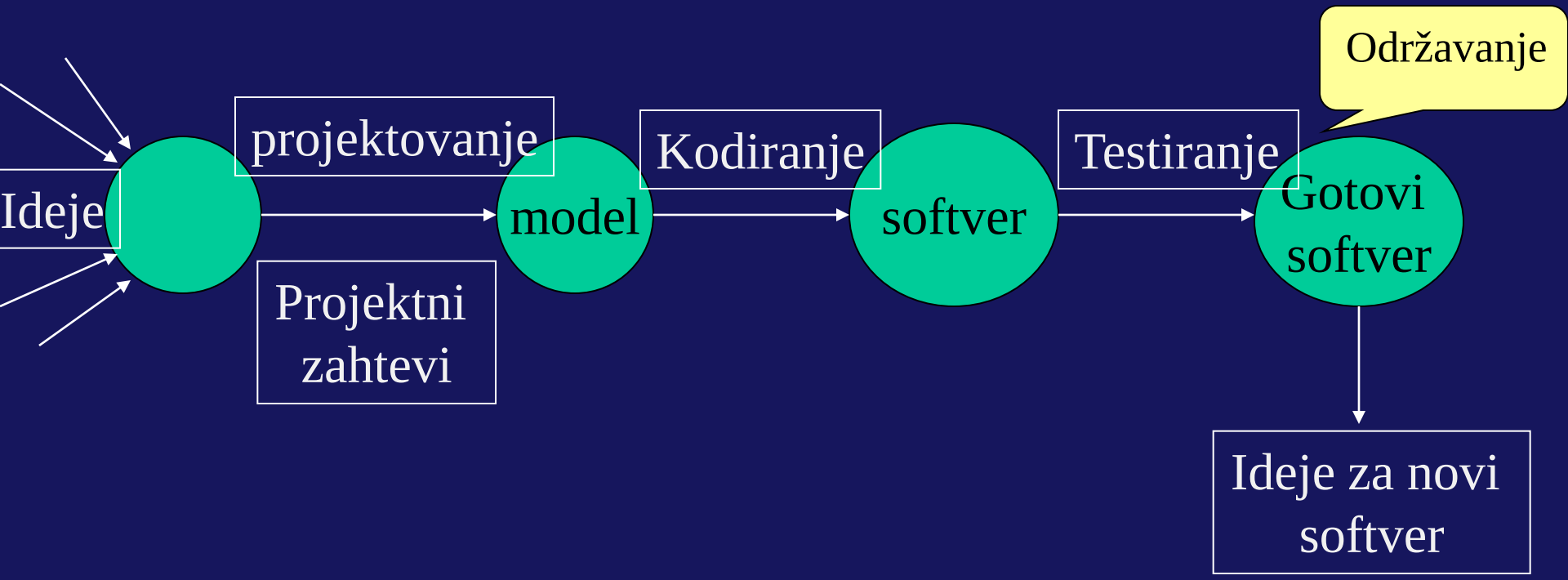
Proizvodnja softvera

- Niz različitih medjusobno zavisnih faza
- Faze disjunktne
- Nakon završetka svake faze kontrolna tačka

Životni ciklus softvera

Period od trenutka identifikacije potreba za softverskim proizvodom do trenutka prestanka korišćenja softverskog proizvoda

šema životnog ciklusa softverskog proizvoda



Faze u životnom ciklusu softvera

1. Iniciranje i definisanje projekta (definicija problema) - 3%
2. Analiza i specifikacija zahteva (specifikacija problema)- 3%
3. Idejno projektovanje – 7%
4. Detaljno projektovanje – 15%
5. Programiranje (Kodiranje)
6. Testiranje
7. Eksploatacija
8. Dokumentacija

I - Iniciranje i definisanje projekta

- **Troši najmanje vremena**
- **U definiciji problema izbeći kontradiktorne zahteve i nepotpunosti**
- **Podfaze:**

  a) iniciranje projekta,

  b) preliminarna analiza,

  c) priprema projektnog zadatka

  d) usvajanje projektnog zadatka

Iniciranje projekta:

- **Bilo koje lice najčešće investitor**
- **Sastoji od opisa problema i zadataka koje novi sistem treba da ispuni.**
- **Rezultat ove faze je projektni zahtev koji sadrži:**
- **a) opis problema**
- **b) ciljeve uvodjenja novog sistema**
- **c) kratak sadržaj funkcija koje novi sistem treba da realizuje**
- **d) očekivane efekte i dobit**
- **e) moguće korisnike**
- **f) ograničenja**
-

Rezultat je projektni zahtev

Preliminarna analiza:

- **Vrši se ako projektni zahtev ima manjkavosti ili nedovoljne podatke za donošenje zaključaka.**
- **Preciziranje se vrši ako u projektnom zahtevu ne postoji dovoljno precizan opis problema što je preduslov za dalje faze. U tom slučaju se projektni zahtev dopunjuje, precizira, izvode se intervjui sa kompetentnim osobama o validaciji projekta.**
- **Posle izvršenih dopuna, ponovo se daje ocena o izvodljivosti.**
- **U izvodljivost se uključuju: funkcionalna, tehnička, ekonomska, zakonska sredstva na osnovu kojih se daje ocena o izvodljivosti.**

Priprema projektnog zadatka: Sadrži:

- opis problema koji se rešava,
- ciljeve,
- prednosti u odnosu na postojeće sisteme,
- ograničenja,
- moguće korisnike,
- očekivane rezultate

Usvajanje projektnog zadatka:

- Usvaja ga nadležni organ (korisnik).
- Posle usvajanja donosi se pisana odluka.

II – Specifikacija problema (softvera)

- detaljniji dokument u odnosu na iniciranje ideje
  - Vrši se usaglašavanje raznih specifičnosti eliminišući kontradiktornosti zahteva (Zahtev ka funkcionalnim karakteristikama, ka ulaznoj i izlaznoj informaciji tj informacionoj bazi problema.)
  - Funkcionalne karakteristike su: niz funkcija koje se dodeljuju na izvršenje, vrednosti parametara koje karakterišu izvršenje svake funkcije, pouzdanost karakteristika programskog proizvoda.
- Usaglašavanje funkcionalnih karakterisitka je najodgovornija operacija koja može ako je nejasna da dovede do nesporazuma izmedju onog ko zahteva proizvod i onog ko ga realizuje.
- Zahteve piše korisnik sa projektantom na osnovu intervjua.
- Specifikacija je formalan opis bitnih osobina i najčešće se daje pomoću tekstualnog jezika koji je jasan i nedvosmislen.

III – Idejno projektovanje

- prva faza kreiranja softvera

- počinje nakon korektnog završetka prve dve faze

- programski proizvod se razbija na niz nezavisnih jedinica medjusobno funkcionalno povezanih. Pri tome se svaka celina (modul) detaljno ne razradjuje tj posmatra se kao crna kutija koja ima niz ulaznih i izlaznih priključaka.

- idejni projekat ne sadrži detalje već osnovne koncepcije proizvoda

- cilj je da se ne projektuje proizvod već model proizvoda. Pri tome projektovani proizvod mora:

a) da ima hijerarhijsku strukturu veza izmedju problema

b) da poštuje principe modularnosti (sistem treba deliti tako da svaki deo bude logički nezavistan i kompletan)

c) svaki modul funkcionalno nezavistan

IV – Detaljno projektovanje

- Počinje posle neformalne provere idejnog projektovanja.
- Ovom fazom počinje stvaranje konkretnog softverskog proizvoda
- Uspešan detaljni projekat lako se pretvara u niz programskih celina pri čemu sledeće faze imaju trivijalni karakter.
- Posle njegovog završetka pristupa se formalnoj reviziji.

To je odbrana uspešnog projekta i odbacivanje projekata koji sadrže krupne principijelne nedostatke. Primedbe za sitne nedostatke se kasnije otklanjaju i ponovo se vrši formalna revizija (naručioci i izvršioci posla)

V – Kodiranje

- Ako je formalna revizija uspešna, pristupa se kodiranju programa.
- Pri tome se koriste najpre pseudokod ili jezik blizak pseudokodu,
- Koriste se različite tehnike kodiranja kojima se obezbedjuje poštovanje principa nezavisnosti programskog sistema.

VI – Testiranje

- Direktna provera pomoću računara
- Ne dokazuje se već opovrgava valjanost softvera
- Dobro smišljenim testovima može se dokazati valjanost softvera

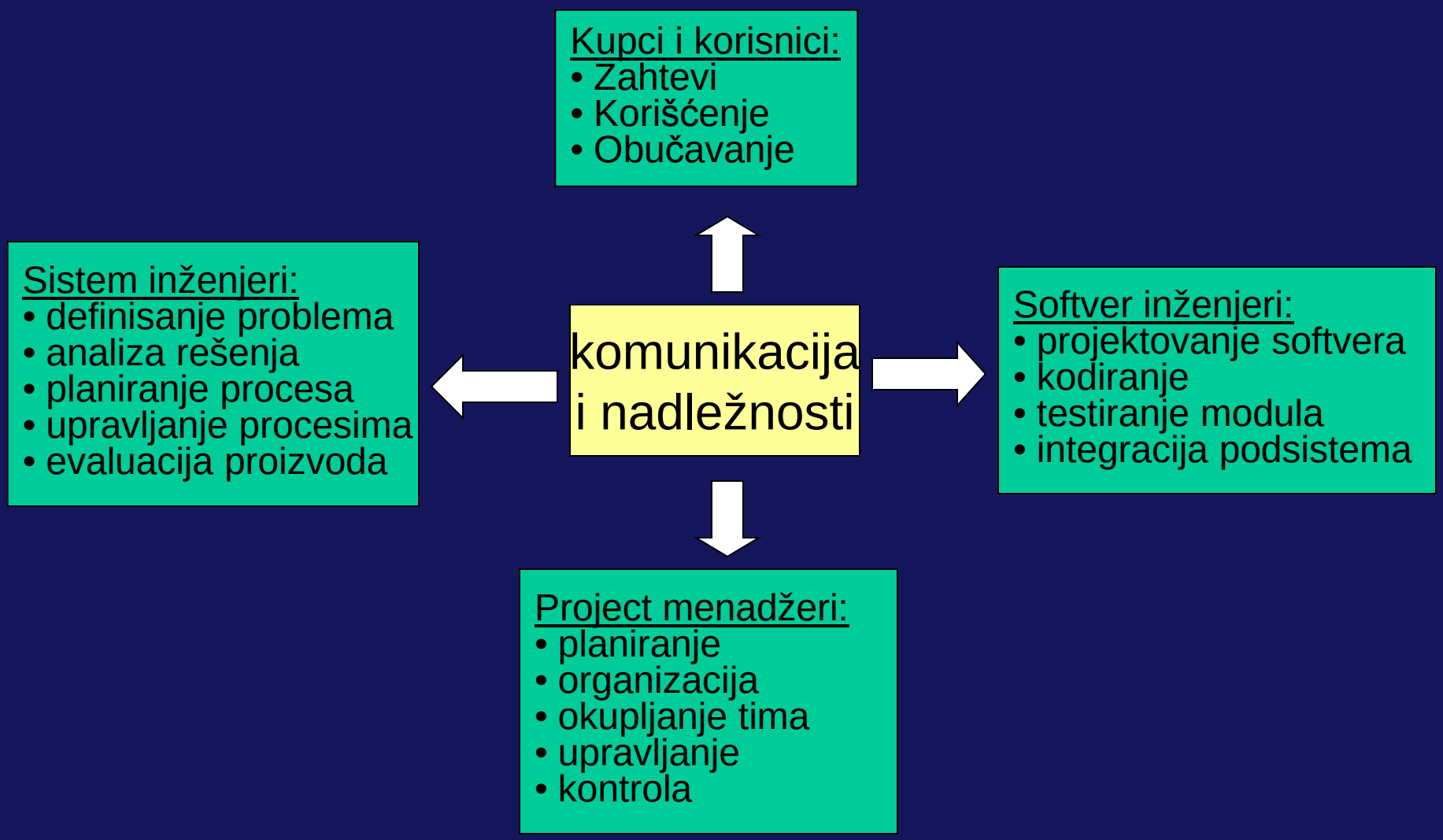
VII/VIII – Eksploatacija i Dokumentacija

prodaja, eksploatacija, održavanje

Potrebna dobra prateća dokumentacija:

a) za prodaju
b) za instaliranje
c) za proceduru rada i korišćenje

Učesnici u razvoju softvera



Najznačajniji resurs u proizvodnji softvera
LJUDI

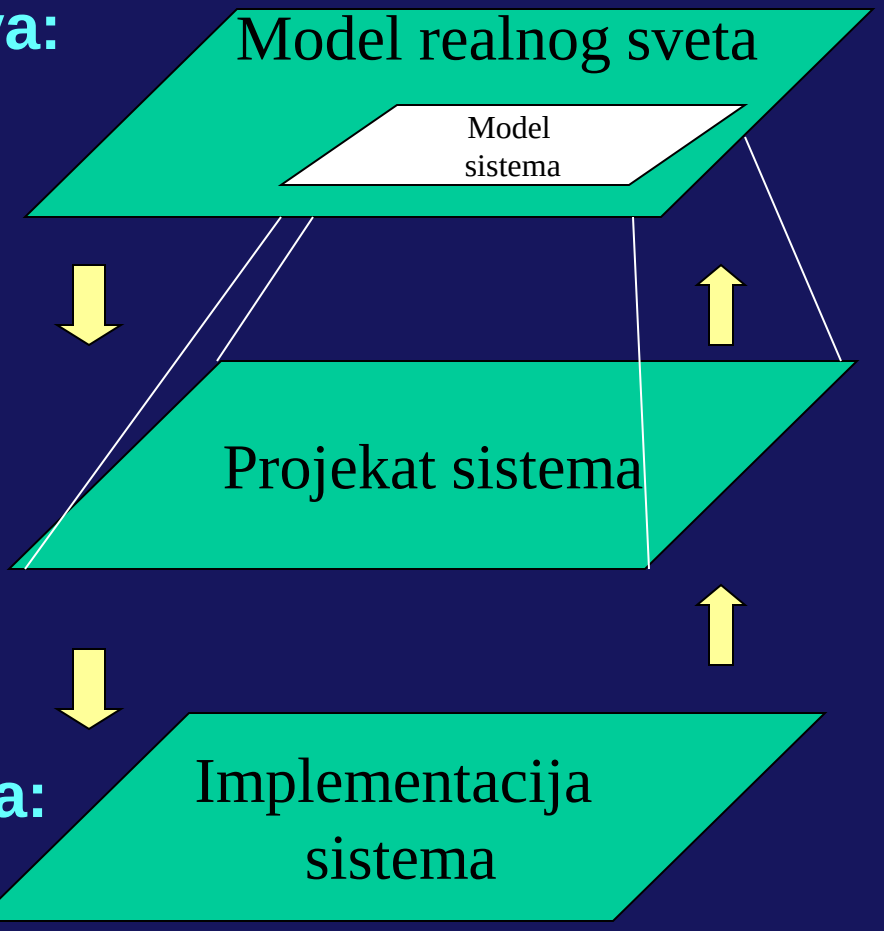
Izražava se vrednost:

čovek godina = broj ljudi * vreme angažovanja

Nivoi apstrakcije

Analiza zahteva:
ZAŠTO?

- zahtevi
- domensko znanje
- ciljevi



Projektovanje:
ŠTA?

- abstrakcije
- modeli
- strukture
- arhitektura

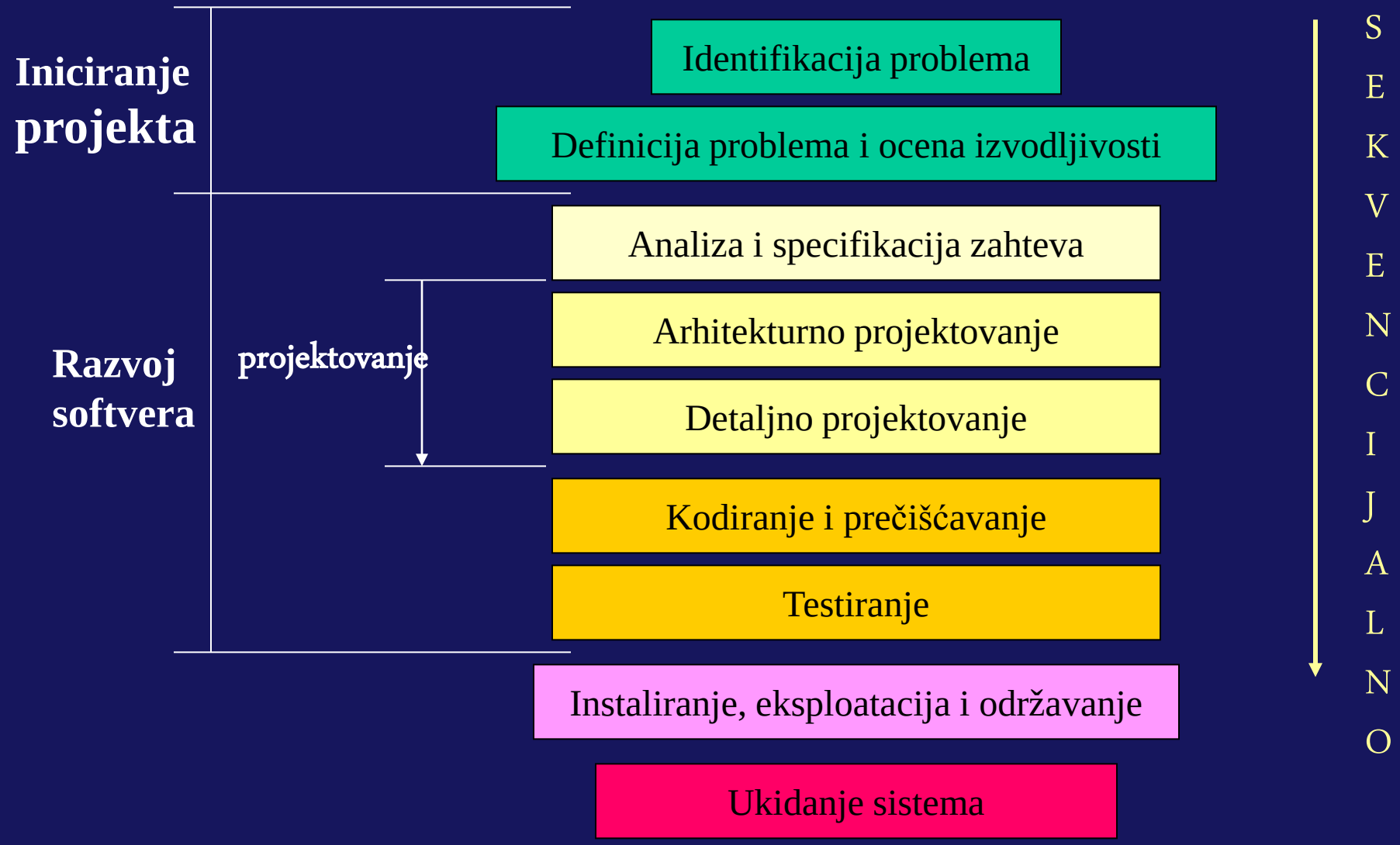
Implementacija:
KAKO?

- algoritmi
- generički servisi
- specifični servisi
- biblioteke

Modeli procesa razvoja softvera:

1) Model vodopada
2) Modifikovani model vodopada
3) Inkrementalni model
4) Model prototipa
5) Višestruko korišćenje softverskih komponenti
6) Automatska sinteza softvera
7) Boehmov spiralni model
8) Model softverska oluja

Konvencionalni model (model vodopada)



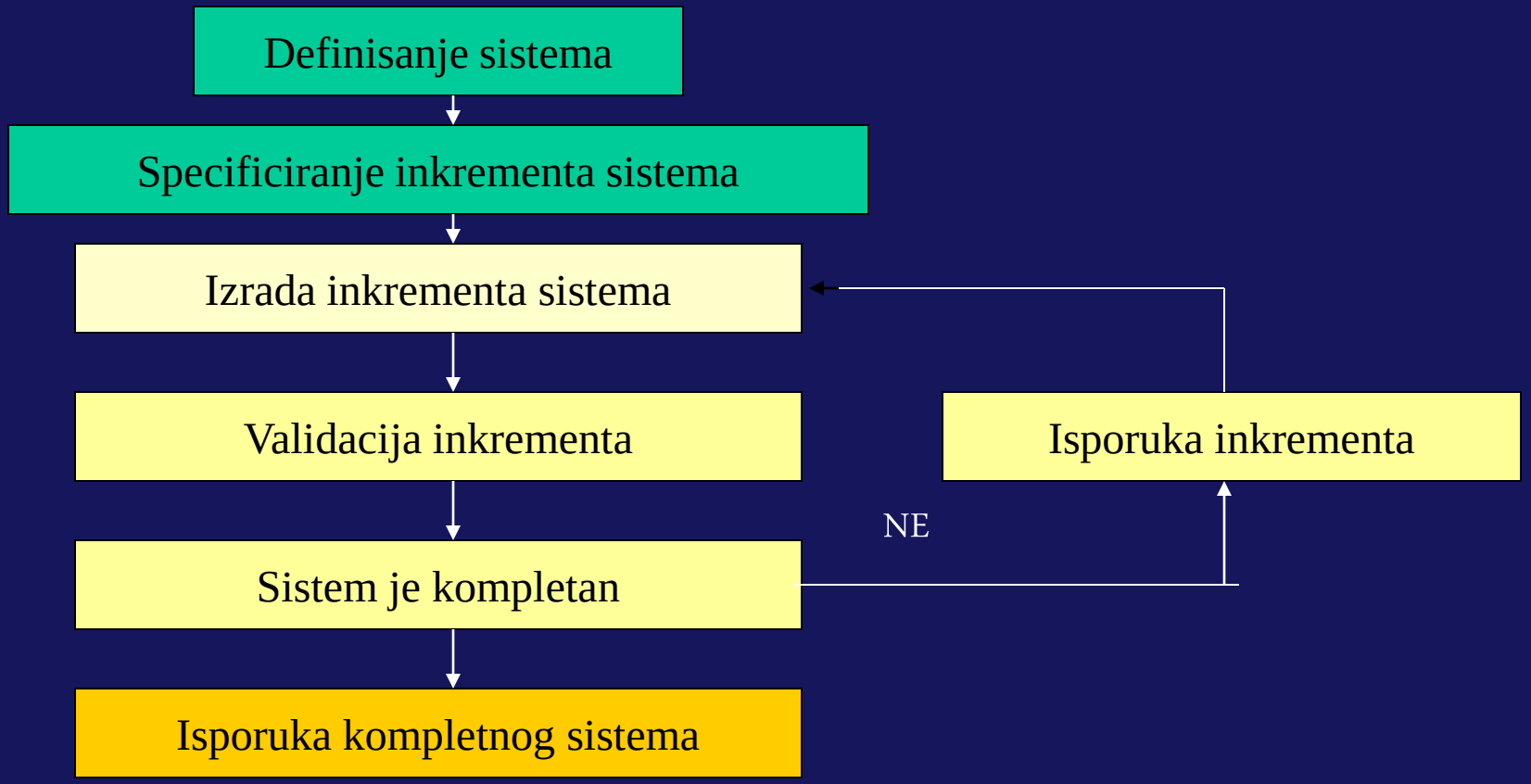
Prednosti modela vodopada

- Podstiče precizno definisanje zahteva pre projektovanja sistema
- Podstiče definisanje interakcije medju komponentama pre kodiranja
- Omogućava vodjenje i nadgledanje projekta i tačnije planiranje resursa
- Zahteva formiranje dokumentacije u svakoj fazi
- Smanjuje troškove razvoja i održavanja

Nedostaci modela vodopada

- Specifikacija zamrzava prepoznavanje novih zahteva u ranim fazama razvoja softvera
- Troši se dosta vremena na izradu specifikacije
- Kasni se sa pisanjem koda
- Ima malo povratnih veza od krajnjeg korisnika do faze kodiranja, te ako softver ne odgovara zahtevima korisnika unošenje izmena u softver je vrlo skupo
- Postoje vrlo slabe veze izmedju softvera koji se razvija i već razvijenih softverskih proizvoda
- Nije pogodan za male poslovne aplikacije, interaktivne aplikacije, ekspertne sisteme i sisteme bazirane na znanju

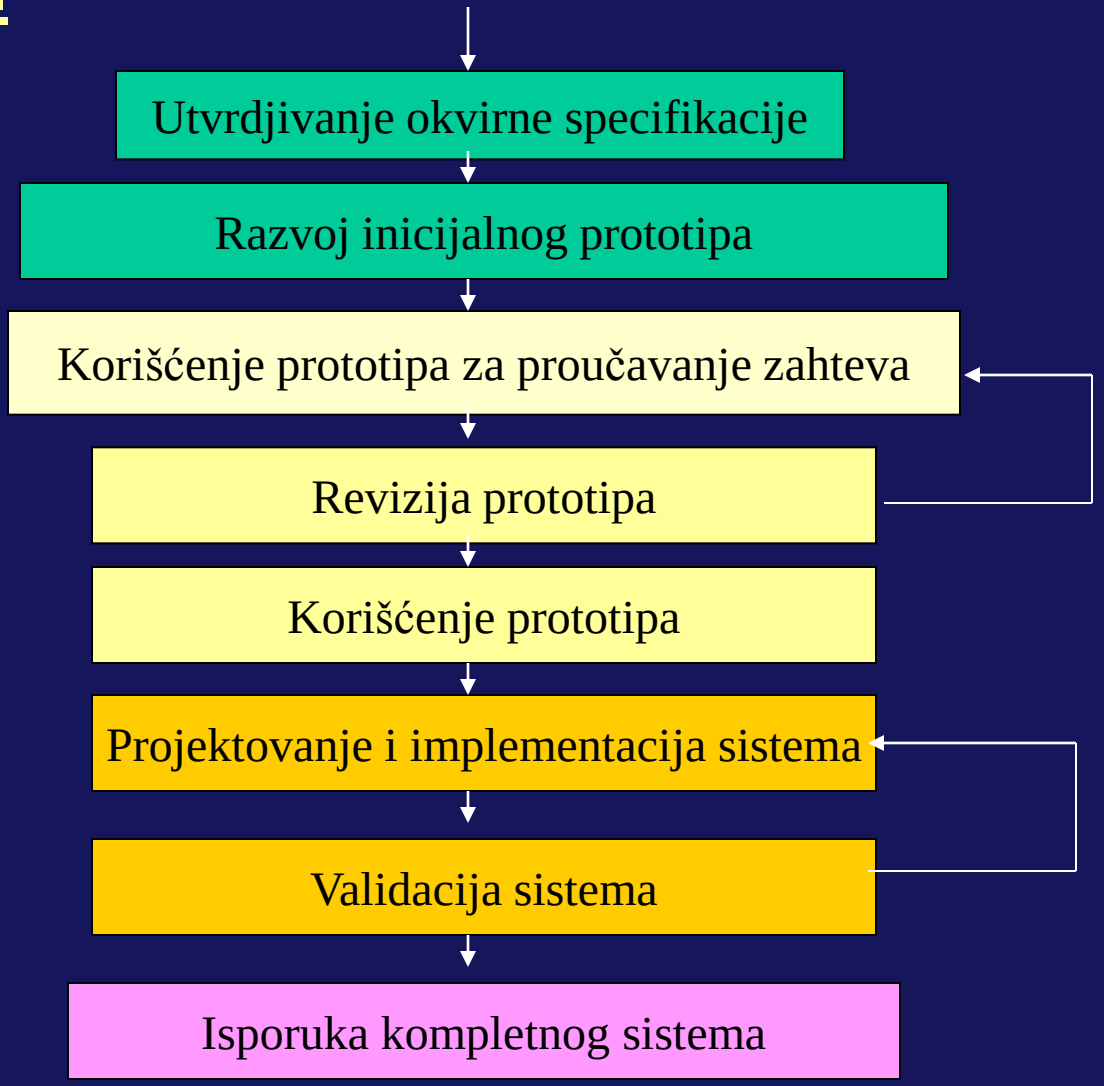
Inkrementalni model



Prednosti inkrementalnog modela

- Skraćuje vreme i redukuje troškove do pojave inicijalne verzije sistema
- Lakše testiranje jer je svaki inkrement prostiji od sistema u celini
- Ranije uključivanje korisnika u proces razvoja softvera čime se obezbedjuje njegov veći uticaj na konačan izgled softvera i automatski se smanjuje potreba za kasnijim izmenama softvera

Prototipni model



Prednosti prototipnog modela

- Pri demonstraciji prototipa mogu se otkriti nesporazumi izmedju projektanta i korisnika
- Mogu se detektovati izostavljeni zahtevi korisnika
- Mogu se identifikovati i otkloniti nejasnoće u funkcionalnosti sistema
- Na razvijenom prototipu se mogu otkriti nekompletni i nekonzistentni zahtevi
- Vrlo brzo se dolazi do skromne verzije sistema – demonstracioni prototip
- Prototip može poslužiti kao baza za pisanje specifikacije sistema koji se razvija

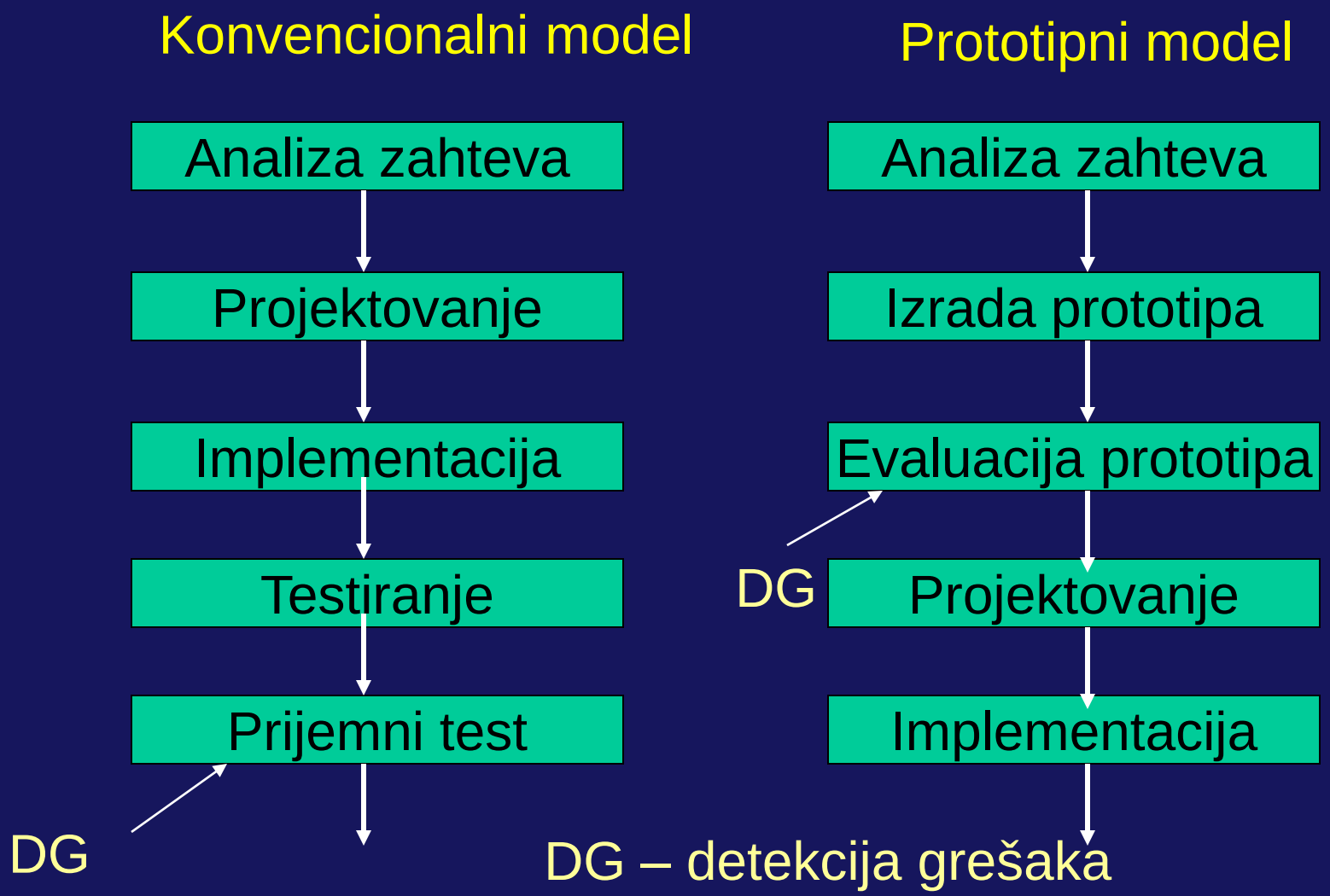
Napomena: Prototip se ne sme koristiti kao specifikacija

- Prototip ne sadrži sve funkcije sistema
- Prototip ne može biti sastavni deo ugovora
- Nefunkcionalni zahtevi se ne mogu na adekvatan način izraziti u prototipu
- Prototip se ne koristi na isti način kao radna verzija sistema

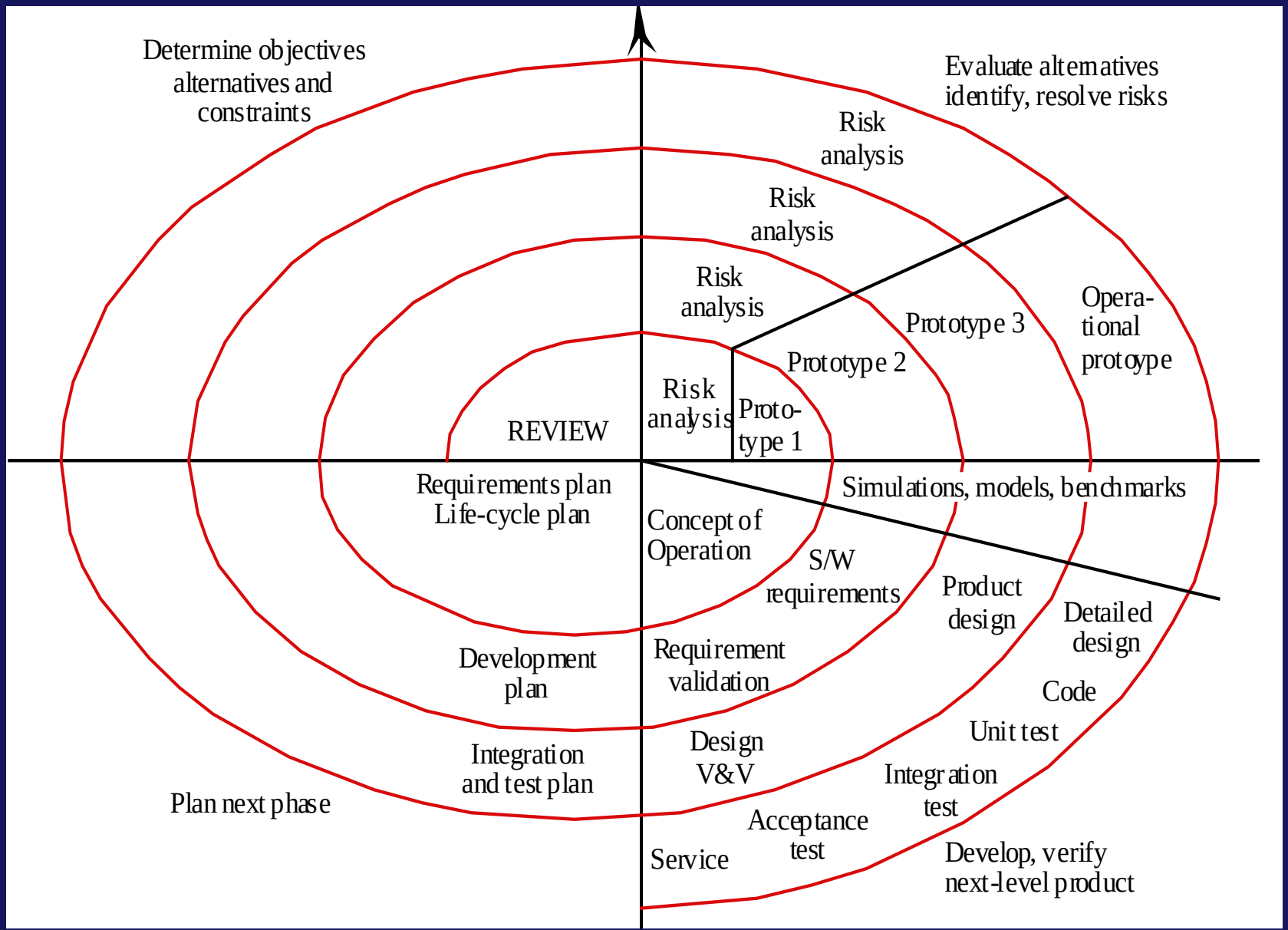
Karakteristike prototipnog prilaza:

- Potrebni jezici visokog nivoa
- Ignorisanje nefunkcionalnih zahteva (brzina, prostor)
- Ignorisanje obrade grešaka
- Redukovanje pouzdanosti i kvaliteta softvera
- Ne treba redukovati ni jedan zahtev vezan za korisnički interfejs

Poredjenje modela

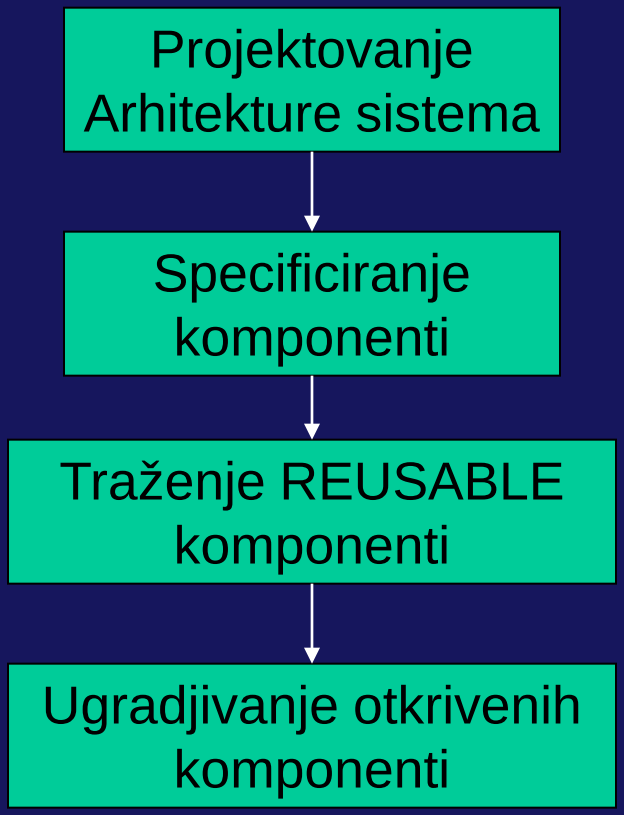


Spiralni model



Višestruko korišćenje softverskih komponenti

Razvoj sa višestrukim korišćenjem komponenti



Razvoj vodjen višestrukim korišćenjem komponenti

Prednosti višestrukog korišćenja softvera

- Maksimalno korišćenje postojećih softverskih komponenti
- Redukuju se ukupni troškovi razvoja
- Manje softverskih komponenti treba specificirati, projektovati, implementirati i proveravati
- Povećava se stabilnost sistema
- Redukuje se ukupan rizik
- Bolje iskorišćenje specijalista
- Lakše ugradjivanje standarda u reusable komponentama
- Može se redukovati vreme razvoja softvera

Uslovi za primenu modela višestrukog korišćenja softvera

- Mora postojati mogućnost da se nadju reusable komponente
- Mora postojati dobra specifkacija reusable komponenti
- Mora postojati opis načina korišćenja reusable komponenti
- Mora postojati dovoljno bogata biblioteka reusable komponenti
- Mora postojati visoki stepen generalizacije softverskih komponenata

Metode koje su ostvarile najveći uticaj na oblast OO modeliranja su:

- Booch metoda
- OMT (*Object Modeling Technique*, Rumbaugh)
- OOSE (*Object Oriented Software Engineering*, Jacobson)
- Fusion
- Shlaer-Mellor
- Coad-Yourdon

CASE alati

- Rational Solution

Mogućnosti koje nudi Rational Software

- **Prikupljanje zahteva i analiza** - integrisani alati za prikupljanje zahteva, razvoj use case-ova, biznis modelovanje i modelovanje podataka
- **Dizajn i konstrukcija** - alati za modelovanje arhitekture i dizajna, razvoj upravljan modelom (model-driven), testiranje komponenti i runtime analiza aktivnosti. Maksimizacija produktivnosti kod rada za biznis aplikacijama, SW proizvodima i sistemima i embedded sistemima
- **Automatizovano testiranje** - za proveru svih aspekata kvaliteta SW: funkcionalnosti, pouzdanosti i performansi
- **Konfiguracija SW** - rešenja za uključivanje kontrole nad promenama u SW (version control), praćenje grešaka i promena, razvojni tim prati promene
- **Praćenje životnog ciklusa** - olakšavanje rada timova i njihove komunikacije i razmene podataka, testiranja, primena proverenih procesa razvoja, izveštaj o napredovanju

Prikupljanje zahteva i analiza

- **IBM Rational RequisitePro** – alat za prikupljanje zahteva i rad sa use-case-ovima, kojim se poboljšava komunikacija unutar tima i smanjuju rizici razvoja
- **IBM Rational Rose Data Modeler** – alat za vizualno modelovanje, koji omogućava da dizajneri baza podataka, analitičari, arhitekte SW i ljudi koji razvijaju SW rade zajedno, u sakupljanju i podeli biznis zahteva, i njihovo praćenje u toku razvoja
- **IBM Rational Rose XDE Modeler** – primena data-driven razvoja, korišćenjem UML-a. Mogu se kreirati modeli nezavisni od platforme ili SW arhitekture, reusable delovi

Dizajn i konstrukcija

- **IBM Rational Rose XDE Modeler**
- **IBM Rational Rose XDE Developer** – nudi veliki broj mogućnosti za model-driven razvoj i runtime analizu, za izgradnju kvalitetnog SW
- **IBM Rational Rose Data Modeler** - ER metodologija korišćenjem UML notacije za bolji spoj ljudi koji razvijaju baze i razvojnog tima SW-a
- **IBM Rational Rose Technical Developer** – izvršenje modela i generisanje potpuno izvršivog koda - visoka produktivnost
- **IBM Rational Rapid Developer**- Kombinuje RAD modelovanje i dizajn sa automatskim generisanjem koda.
- **IBM Rational PurifyPlus** - Kompletni set alata za runtime analizu, za povećanje pouzdanosti i performansi SW-a

Dizajn i konstrukcija

- **IBM Rational Test RealTime** - cross-patform rešenje za runtime analizu i testiranje komponenti, posebno za kod embedded sistemea
- **IBM Rational Ada Developer**-podrška za savremene Ada projekte, kroz ceo lifecycle
- **IBM WebSphere® Studio** - dopuna Rational proizvodima- za brži razvoj timskih aplikacija, user friendly aplikacija usko povezana sa drugim IBM aplikacijama slične namene, omogućava fleksibilnu, portal- like integraciju multi-language, multi-platform i multi-device alata za razvoj aplikacija, za building, testiranje

Automatizovano testiranje (automatsko testiranje sistema)

- **IBM Rational Functional Tester for Java and Web** – kreira prilagodljive i reusable test skriptove u Javi, koriste}i ScriptAssure™
- **IBM Rational Robot** - automatizuje funkcionalno, regresiono i konfiguraciono testiranje za široki opseg aplikacija uključujući u .NET
- **IBM Rational Performance Tester** – otkriva i popravlja production-environment probleme performansi pre njihovog korišćenja u realnosti
- **IBM Rational Team Unifying Platform** – integriše sve aktivnosti testiranja za jednu aplikaciju u centralizovani test menadžment, defect praćenje i kontrolu verzija

Automatizovano testiranje (ručno testiranje sistema)

- **IBM Rational PurifyPlus**
- **IBM Rational Rose XDE Developer Plus**- nudi kompletni vizuelnu sredinu za dizajn i razvoj za j2EE i .NET bazirane sisteme
- **IBM Rational Test RealTime**

Konfiguracija SW

- **IBM Rational ClearCase Change Management Solution** – za menadzment projekata srednje i veće veličine
- **IBM Rational ClearCase Change Management Solution Enterprise Edition** - za menadžment projekata srednje i veće veličine i/ili geografski distribuirane sisteme
- **IBM Rational ClearCase (MultiSite)** - menadžment glavnih faktora srednjih i većih projekata + za distribuirane sisteme
- **IBM Rational ClearCase LT**- Za početne verzije malih projekata
- **IBM Rational ClearQuest (MultiSite)** – Fleksibilno praćenje defekata i promena u toku lifecycle + za distribuirane sisteme

Praćenje životnog ciklusa

- **IBM Rational Team Unifying Platform** - oprema tim infrastrukturnim alatima, procesima i integracijama, da bi rad bio efektivniji. Zajednički pristup važnim tačkama razvoja
- **IBM Rational Suite**- za analitičare, ljude koji razvijaju i testere, za komunikaciju izmedju timova kroz ceo lifecycle
- **IBM Rational Suite for Technical Developers** - podrška u toku celog lifecycle za pisanje komplikovanog koda za kritične delove, kao što su real-time ili embedded sistemi
- **IBM Rational Suite DevelopmentStudio for UNIX** – za razvoj UNIX SW proizvoda
- **IBM Rational Unified Process** -fleksibilni proces razvoja SW, pomaže u tome da ceo tim prati jedinstveni proces prilagodjen potrebama projekta