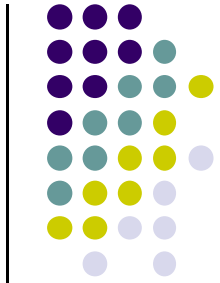
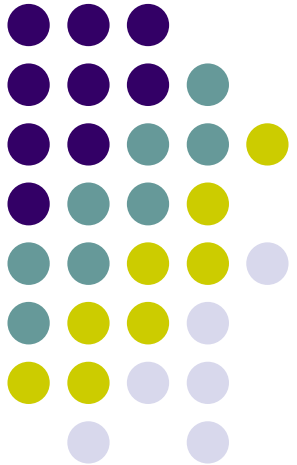


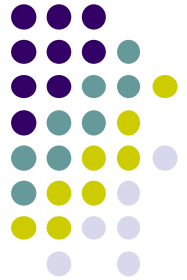
Projektiranje informacijskih sustava



Implementacija

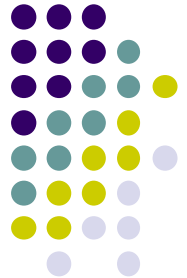
Ak. god. 2009/2010

Implementacija



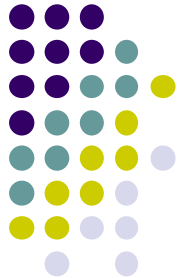
- Faza implementacije SDLC-a je faza u kojoj se sustav izrađuje na temelju rezultata iz prethodnih faza SDLC-a.
- Najvažniji dio realizacije je pisanje programa sustava, ali pored programiranja u fazi implementacije još se definiraju i provode testovi razvijenog sustava te piše dokumentacija samog sustava.

Implementacija



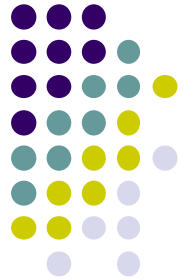
- Implementacija se sastoji od tri koraka:
  1. Izrada sustava (*System Construction*): U ovom koraku sustav se izrađuje i testira kako bi se osiguralo da se ponaša u skladu sa zahtjevima.
  2. Instalacija (*Installation*): U postupku instalacije sustav se stavlja u rad. Ukoliko je postojala starija verzija sustava (čest slučaj) stari sustav se može ugasiti prije no što se novi stavi u funkciju, ili novi sustav može neko vrijeme raditi paralelno sa starim zbog dodatnog testiranja ili se novi sustav uvodi dio po dio, a ne sve odjednom. Ova faza uključuje i obuku korisnika.
  3. Plan podrške (*Support Plan*): Definira post-implementacijsku reviziju sustava (*upgrade* i sl.).

Implementacija



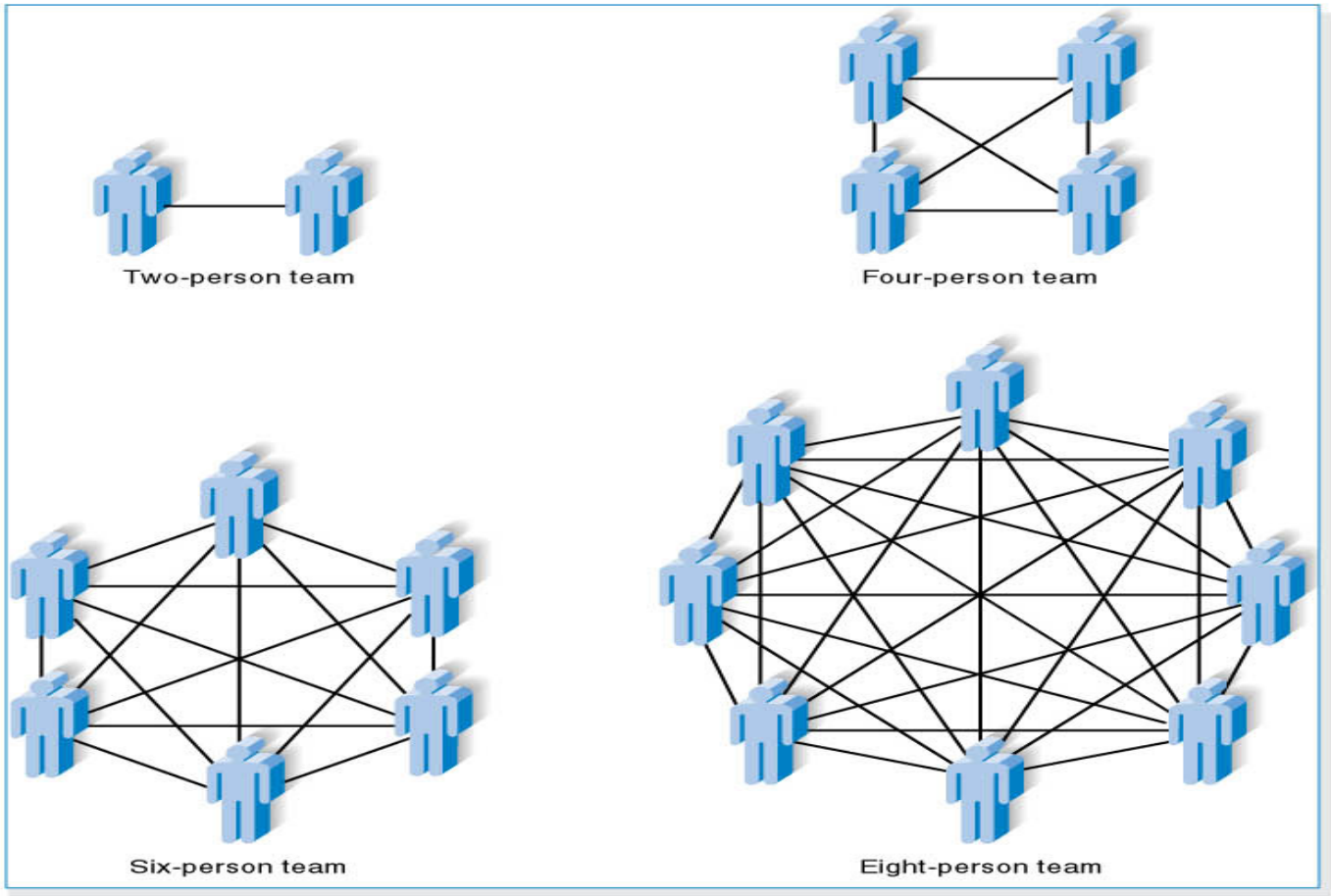
- Posao voditelja projekta u fazi implementacije je:
  1. Dodjeljivanje programskih zadataka (*assigning programming tasks*)
  2. Koordiniranje (*coordinating activities*)
  3. Upravljanje vremenskim planom realizacije (*managing the schedule*)

Dodjeljivanje programskih zadataka

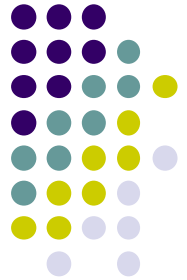


- Tijekom dizajna programa definirani su programi tj. moduli koje je potrebno isprogramirati. Voditelj projekta treba dodijeliti pojedine module pojedinim programerima.
- Programerski paradoks se sastoji u tome da angažiranje dodatnih programera pored optimalnog broja (npr. kod kašnjenja projekta) uzrokuje dodatno kašnjenje projekta.
- Projekti sa velikim timovima se razbijaju u niz manjih neovisnih podprojekata.

Dodjeljivanje programskih zadataka



Koordiniranje

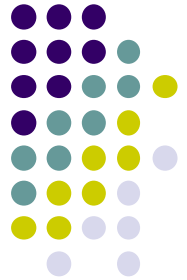


- Koordiniranje implementacije sustava voditelj projekta postiže:
  - Redovitim sastancima (obično na tjednoj bazi)
  - Definiranjem različitih standarda (npr. imenovanje varijabli i sl.)
  - Definiranjem mehanizama za kontrolu promjena i praćenje promjena kôda (alati za praćenje verzija izvornog kôda poput TFS (Team Foundation Server), Concurrent Versions System (CVS),...alati za ticketing itd.).

Upravljanje vremenskim planom realizacije

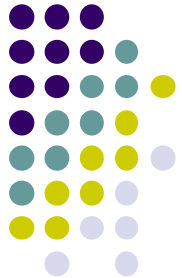
- Koristiti inicijalnu procjenu vremenskog plana (radnog plana)
- Konstantno usklađivati inicijalnu procjenu sa stvarnim stanjem
- Upravljati *scope creepom* tj. dodatnim zahtjevima
- Nadgledati *minor slippage* – mala kašnjenja (1-2 dana) po modulu u konačnici mogu dovesti do velikih kašnjenja

Testiranje



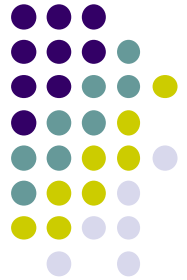
- Testiranje je usko povezano s implementacijom.
- Kod dobro dizajniranih sustava pojedinačni moduli mogu se testirati neovisno.
- Za simuliranje interakcije pojedinačnog modula koji se testira i ostatka sustava tj. ostalih modula koriste se pojednostavljene verzije modula (*stubs*) s kojima je povezan promatrani modul.
- U prosjeku na 10000 linija koda (testiranog) danas dolazi 1 *bug*.

Testiranje



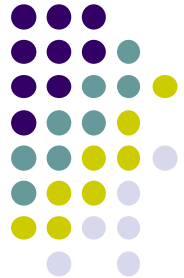
- Testiranje softvera podrazumijeva rad softverskog sustava u kontroliranim uvjetima i evaluaciju rada u tim uvjetima.
- Testiranje se praktički sastoji od tri koraka:
  - Planiranje : Koji ulazni podaci će se koristiti, koji rezultati se trebaju dobiti.
  - Izvođenje : Priprema softvera za testiranje, provođenje testa i određivanje rezultata.
  - Procjena rezultata : Usporedba dobivenih rezultata sa željenim rezultatima.

Testiranje



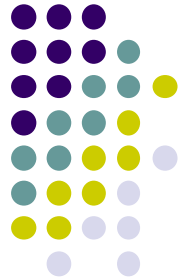
- Kako bi bilo sigurni da određeni softver nema više grešaka trebalo bi napraviti test za sve moguće scenarije korištenja softvera, sve moguće tipove ulaznih podataka odnosno sve moguće tokove izvršavanja softvera.
- To je u praksi nemoguće čak i kod jednostavnijeg softvera.
- No razvijene se određene metode testiranja koje povećavaju vjerojatnost otkrivanja grešaka u kodu uz testiranje samo ograničenog broja scenarija.

Testiranje



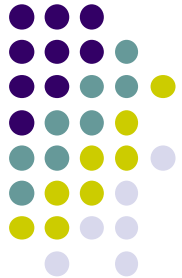
- Kod složenjh softverskih sustava s velikim brojem modula uočeno je da je obično manja skupina modula problematičnija tj. s većom vjerojatnošću grešaka. Pareto princip (Vilfredo Pareto 1848-1923) sastoji se u identificiranju takvih modula i njihovom detaljnijem testiranju.
- '**Pareto principle' -** says that 80% of the errors are concentrated at 20% of the code only – forms a good criteria for testing.
- Testiranje osnovnih putova (*basic path testing*) osigurava da se svaka naredba u kodu izvrši barem jednom u testiranju. To neće pokriti sve moguće tokove izvršavanja, ali izvođenje svake naredbe bar jednom uvećava vjerojatnost pronalaska greške. Za identificiranje takvih testnih podataka razvijena je tehnika zasnovana na teoriji grafova.

Testiranje



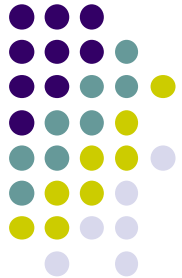
- Prethodne dvije metode zahtijevaju poznavanje interne strukture softvera koji se testira pa spadaju u kategoriju tzv. testiranja staklene kutije (*glass-box testing*).
- Nasuprot tome je tzv. testiranje crne kutije (*black-box testing*) koje se provodi s pozicije korisnika koji ne poznaje internu strukturu softvera.
- Korisnika ne zanima na koji način se određeni zadatak izvršio, koje su se procedure pozvale i sl. Njega samo zanima da li je softver točan (rezultat) i eventualno da li je vremenski odziv softvera zadovoljavajući.

Testiranje



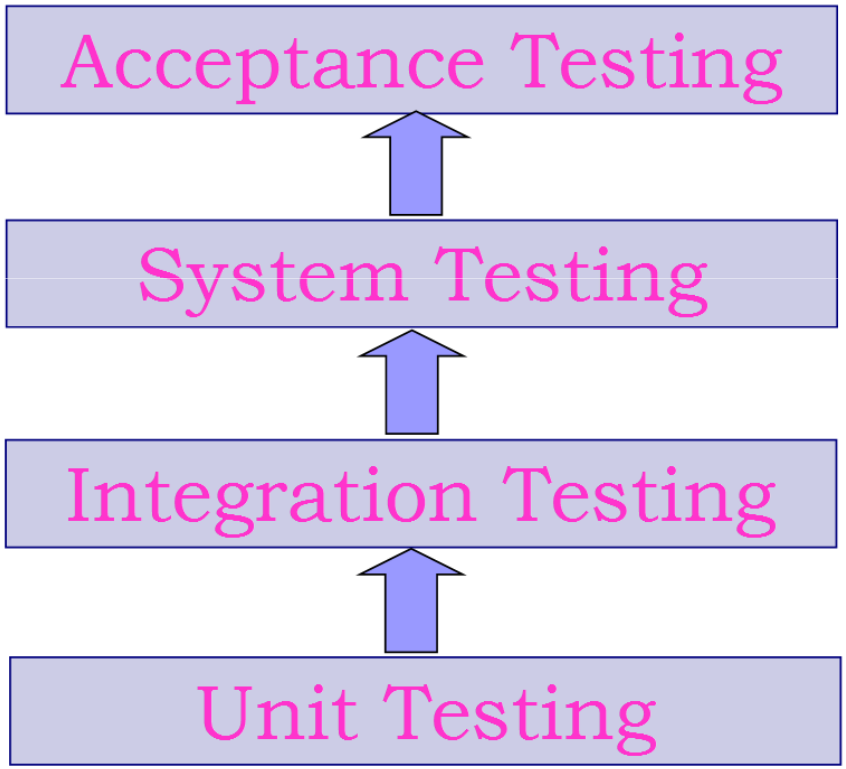
- Jedna metoda je analiza graničnih vrijednosti (*boundary value analysis*) kojom se identificiraju granične točke u softveru i softver se testira samo u tim točkama. Npr. testiranje graničnih vrijednosti ulaznih varijabli.
- Metoda redudancije podrazumijeva razvoj dva softverska sustava koja izvode isti zadatak. Dva se sustava testiraju sa istim podacima, a odstupanja u rezultatima ukazuju na grešku.

Testiranje



- Danas je uobičajeno korištenje metode beta testiranja proizvoda (obično za PC softverske proizvode).
- Korisnicima se isporučuje preliminarna verzija softvera koja se zove beta verzija. Cilj je vidjeti kako se softver ponaša u stvarnim uvjetima prije no što se na tržište isporuči konačna verzija softvera.
- Beta testiranje omogućava ne samo otklanjanje grešaka nego i testiranje zahtjeva tržišta što u konačnici pridonosi kvaliteti konačne verzije softvera.

Razine testiranja

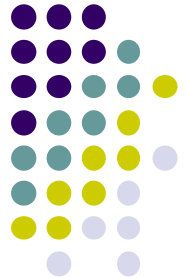


- *Unit Testing* je testiranje pojedinačnih dijelova (modula) aplikacije.
- *Integration Testing* je testiranje nekoliko pojedinačnih modula aplikacije zajedno.
- *System Testing* je testiranje cjelokupne aplikacije.
- *Acceptance Testing* je testiranje softverskog sustava s obzirom na to koliko je ispunio korisnikove zahtjeve. (Npr. beta testiranje se koristi i za ispravljanje bugova, ali i za definiranje dodatnih korisničkih zahtjeva)

Stage	Types of Tests	Test Plan Source	When to Use	Notes
Unit Testing	**Black-box testing:** treats program as black box	Program specifications	For normal unit testing	The tester focuses on whether the unit meets the requirements stated in the program specifications.
	White-box testing: looks inside the program to test its major elements	Program source code	When complexity is high	By looking inside the unit to review the code itself, the tester may discover errors or assumptions not immediately obvious to someone treating the unit as a black box.
Integration Testing	**User interface testing:** the tester tests each interface function	Interface design	For normal integration testing	Testing is done by moving through each and every menu item in the interface either in a top-down or bottom-up manner.
	Use scenario testing: the tester tests each use scenario	Use scenario	When the user interface is important	Testing is done by moving through each use scenario to ensure it works correctly. Use scenario testing is usually combined with user interface testing because it does not test all interfaces.
	Data flow testing: tests each process in a step-by-step fashion	Physical DFDs	When the system performs data processing	The entire system begins as a set of stubs. Each unit is added in turn and the results of the unit are compared to the correct result from the test data; when a unit passes, the next unit is added and the test is rerun.
	System interface testing: tests the exchange of data with other systems	Physical DFDs	When the system exchanges data	Because data transfers between systems are often automated and not monitored directly by the users, it is critical to design tests to ensure they are being done correctly.
System Testing	**Requirements testing:** tests whether original business requirements are met	System design, unit tests, and integration tests	For normal system testing	This test ensures that changes made as a result of integration testing did not create new errors. Testers often pretend to be uninformed users and perform improper actions to ensure the system is immune to invalid actions (e.g., adding blank records).
	Usability testing: tests how convenient the system is to use	Interface design and use scenarios	When user interface is important	This test is often done by analysts with experience in how users think and in good interface design. This test sometimes uses the formal usability testing procedures discussed in Chapter 10.
	Security testing: tests disaster recovery and unauthorized access	Infrastructure design	When the system is important	Security testing is a complex task, usually done by an infrastructure analyst assigned to the project. In extreme cases, a professional firm may be hired.
	Performance testing: examines the ability to perform under high loads	System proposal and infrastructure design	When the system is important	High volumes of transactions are generated and given to the system This test is often done by using special-purpose testing software.
	Documentation testing: tests the accuracy of the documentation	Help system, procedures, tutorials	For normal system testing	Analysts spot-check or check every item on every page in all documentation to ensure the documentation items and examples work properly.
Acceptance Testing	**Alpha testing:** conducted by users to ensure they accept the system	System tests	For normal acceptance testing	Alpha tests often repeat previous tests but are conducted by users themselves to ensure they accept the system.
	Beta testing: uses real data, not test data	No plan	When the system is important	Users closely monitor the system for errors or useful improvements.

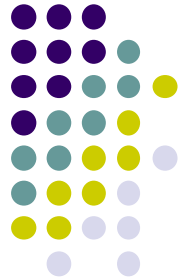
DFD = data flow diagram.

Dokumentacija



- Dokumentacija ima dvije svrhe. Jedna je opisivanje svojstava softvera te načina korištenja softvera i to je ono što nazivamo korisničkom dokumentacijom (*user documentation*).
- Druga svrha je dokumentiranje strukture softvera kako bi se olakšalo njegovo održavanje. Ta se dokumentacija naziva sistemskom dokumentacijom (*system documentation*). Izvorni kôd je dio sistemske dokumentacije popraćen opisom specifikacije softvera. Često se za izradu i održavanje sistemske dokumentacije koriste CASE alati.

Dokumentacija



- Razvoj kvalitetne dokumentacije je zahtjevan. U prosjeku za razvoj jedne stranice dokumentacije treba otprilike 3 sata, dok za razvoj jednog ekrana dokumentacije traje 2 sata.
- Vrijeme potrebno za razvoj dokumentacije treba biti sastavni dio vremenskog plana implementacije.
- Danas se sve češće korisnička dokumentacija realizira kao on-line dokumentacija.