**Brzi razvoj aplikacije (RAD):** Agilne metode su privukle mnogo pažnje, ali neki drugi pristupi brzom razvoju aplikacija su se koristili mnogo godina. Oni su služili razvoju podatkovno intenzivne poslovne aplikacije i oslanjaju se na programiranje i prezentiranje informacija iz baze podataka.

**RAD okolinski alati:** Programski jezik baze podataka, Generator sučelja, Linkovi uredskim aplikacijama, Generatori izvješća

**Generiranje sučelja:** Mnoge aplikacije su bazirane na kompleksnim formama i ručno razvijanje tih formi je aktivnost koja troši vrijeme. RAD okoline sadrže podršku za generiranje prikaza uključujući: Interaktivnu definiciju formi korištenjem „drag and drop“ tehnike**,** Povezivanje (*linkanje*) formi gdje je specificiran slijed formi koje se prezentiraju, Verifikaciju formi gdje su definirani dopušteni rasponi u poljima formi.

**Vizualno programiranje :** Skriptni jezici kao Visual Basic potpomažu vizualno programiranje gdje se prototip razvija stvarajući korisničko sučelje iz standardnih stavki i povezivanjem komponenti sa tim stavkama. Postoji velika baza (*library*) komponenti koja potpomaže ovaj tip razvoja. Komponente se mogu skrojiti tako da odgovaraju specifičnim zahtjevima aplikacije.

**Problemi sa vizualnim programiranjem:** Teško je koordinirati timski temeljen razvoj**,** Nema eksplicitne arhitekture sustava**,** Kompleksne ovisnosti među dijelovima programa mogu uzrokovati probleme s održivošću**,** Implementiranje nestandardnih korisničkih sučelja može biti teško.

**Ponovno korištenje COTS-a:** Efektivan pristup brzom razvoju je konfiguriranje i povezivanje postojećih *off the shelf* sustava. Npr. sustav za upravljanje zahtjevima se može napraviti koristeći:Bazu podataka koja će pohranjivati zahtjeve**,** Procesor riječi koji će dohvaćati zahtjeve i izvješća formata, Proračunske tablice za upravljanje dohvatljivošću.

**Struktura testnog plana softvera:** Proces testiranja, Dohvatljivost zahtjeva, Testni predmeti, Rapored testiranja, Procedure bilježenja testova, Zahtjevi hardvera i softvera, Ograničenja.

**Inspekcijska stopa**: 500 stavki (*statements*) po satu tijekom izvješća, 125 source stavki po satu tijekom individualne pripreme, 90-125 stavki po satu se može očekivati, Inspekcija je po tome skup proces, Inspekcija 500 linija košta otprilike 40 čovjek/sati – oko 2800 funti.

**Faze statičke analize:** Analiza kontrole toka, korištenja podataka, sučelja, toka informacija**,** putanje.

**Verifikacija i formalne metode:** Formalne metode se mogu koristiti kada se stvara matematička specifikacija sustava. Te metode su ultimativna statička verifikacijska tehnika. Uključuju detaljnu matematičku analizu specifikacije i mogu razviti formalne argumente da bi program zadovoljio matematičku specifikaciju.

**Argumenti za formalne metode:** Stvaranje matematičke specifikacije zahtjeva detaljnu analizu zahtjeva i to vjerojatno otkrije greške. Mogu detektirati implementacijske greške prije testiranja kada se program analizira usporedno sa specifikacijom.

**Argumenti protiv formalnih metoda:** Zahtjevaju specijalnu notaciju koju ne mogu razumijeti stručnjaci domene. Veoma je skupo razviti specifikaciju i još skuplje prikazati kako program odgovara specifikaciji. Moguće je dostići istu razinu pouzdanosti programa i jefitnije koristeći druge V & V tehnike.

**Formalna specifikacija i inspekcije:** *State based* model je specifikacija sustava i proces inspekcije provjerava program prema tome modelu. Ovaj pristup programiranju je definiran tako da je jasna veza između modela i sustava. Matematički argumenti (ne dokazi) se koriste da bi se povećala pouzdanost u proces inspekcije

**Use caseovi:**mogu biti osnova za dobivanje testova za sustav. Oni pomažu otkrivanju operacija koje će se testirati i pomažu dizajniranju potrebnih testnih slučajeva. Iz povezanog slijednog dijagrama identificiraju se ulazi i izlazi (*inputs and outputs*) koji će se stvoriti.

**Testiranje objektnih klasa:** Potpuni test klasa uključuje:Testiranje svih operacija koje su povezane sa objektom**,** Postavljanje i ispitivanje svih atributa objekta**,** Provjeravanje objekta u svim mogućim stanjima.Nasljeđivanje otežava dizajniranje objektnih klasa jer informacije koje se testiraju nisu locirane.

**Testiranje sučelja:** Ciljevi su detektirati greške nastale zbog grešaka sučelja ili krivih pretpostavki o sučeljima. Posebno je važno za objektno orijentirani razvoj jer su objekti definirani po njihovim sučeljima

**Tipovi sučelja**: Sučelja parametara, sučelja zajedničke memorije, Proceduralna sučelja, Sučelja prosljeđivanja poruka.

**Greške sučelja:** Pogrešna upotreba sučelja, Nerazumijevanje sučelja, Greške tajmiranja.

**Smjernice za testiranje sučelja**: Dizajnirati testove tako da su parametri pozvanoj proceduri na ekstremnim krajevima svojeg opsega, Uvijek testirati pokazivačke parametre sa null pokazivačima, Dizajnirati testove koji uzrokuju pad (neuspjeh) komponenti, Koristiti testove opterećenja u sustavima za prosljeđivanje poruka, U sustavima sa dijeljenom memorijom, varirati redoslijed aktiviranja komponenti.

**Dizajn testnog slučaja:** Uključuje dizajniranje testnih slučajeva (ulaza i izlaza) koji se koriste za testiranje sustava. Cilj dizajna testnog slučaja je stvoriti skup testova koji su efektivni za validaciju i testiranje mana.Postoji više pristupa dizajnu:Testiranje bazirano na zahtjevima, Testiranje particija, Strukturalno testiranje.

**Testiranje bazirano na zahtjevima:** Generalni princip inženjeringa zahtjeva je da se zahtjevi mogu testirati. Testiranje bazirano na zahtjevima je tehnika testiranja validacije gdje se svaki zahtjev uzima u obzir i iz toga proizlazi skup testova za taj zahtjev.

**Testiranje particija:** Podaci koji se unose i izlazni rezultati često spadaju u različite klase gdje su članovi klase povezani. Svaka od tih klasa je ekvivalentna particijaili domena gdje se program ponaša na ekvivalentan način za svakog člana klase. Testni slučajevi bi se trebali birati iz svake particije.

**Smjernice za testiranje sa sekvencama:** Treba testirati sekvence koje imaju samo jednu vrijednost. Treba koristiti sekvence različitih veličina u različitim testovima. Treba izdvojiti testove tako da se pristupi prvom, srednjem i zadnjem elementu sekvence. Treba testirati sekvence sa duljinom 0.

**Strukturalno testiranje:** Nekad se zove i **white-box testing**. Treba izdvojiti testove prema strukturi programa. Poznavanje programa se koristi za prepoznavanje dodatnih testnih slučajeva. Cilj je iskušati sve stavke programa (ne sve kombinacije putanje).

**Testiranje putanje (*path testing*):** Cilj testiranja putanje je osigurati da je skup testnih slučajeva takav da se svaka putanja kroz program izvrši barem jednom. Početna točka za testiranje putanje je graf toka programa koji pokazuje čvorove koji predstavljaju odluke programa i kružnice koje predstavljaju tok kontrole. Stavke (*statements*) sa stanjima su tako čvorovi grafa toka

**Automatizacija testiranja:** Testiranje je skupa faza procesa. Testni **workbenchovi** pružaju raspon alata da bi smanjili potrebno vrijeme i ukupnu cijenu testiranja. Sustavi kao JUnit podržavaju automatsko pokretanje testova. Većina testnih *workbenchova* su otvoreni sustavi jer testiranje treba biti specifično za organizaciju. Ponekad ih je teško integrirati sa zatvorenim dizajnom i *workbenchevima* za analizu.

Mogu se razviti skripte za simulatore korisničkih sučelja i uzorci za generatore testnih podataka. Moraju se ručno pripremiti testni izlazi (*outputs*) za usporedbu. Mogu se razviti komparatori za datoteke sa specijalnom svrhom.