1.)

Integrációs stratégiák

- Big Bang stratégia: A rendszer minden modulja kész van és egyszerre integráljuk az összeset. Az összeállított rendszert teszteljük. Előnye, hogy kevés tesztet kell ismi visszont a hibák megtalálása rendkívül nehéz ezért nem ajánlott.
- Inkrementális integráció: A moduloket folyamatosan egyre bővítve integrálljuk a rendszerbe. Először csak néhány műszeges modult, hogy működjön a rendszer, ha a teszteken átmegy, akkor bővitjük új moduloskal, új tesztek a kibővített rendszerhez és a korábbiakat is újra futtatjuk.
- Top-down integráció: A hierarchia legfelső szintjén lévő modulall kezdjük az integrációt, a műszeges modulok interface-eit stub-okkal helyettesitjük. Ha átmegy a teszten akkor a stub-okat valós moduloxra cseréljük és az új modulok függőségeit helyettesitjük stub-okkal.
- Bottom-Up integráció: A hierarchia legalján lévő moduloket integráljuk először. Teszt driver irása műszeges a magasabb szinten lévő moduloz interface-einek szimulálására. Sikeres teszt után a test drivereket való, moduloxra cseréljük és eggyel magasabb szinten ismételjük a folyamatot.

2.

- Vízesés modell
- V modell
- ~~Evolúciós modell~~ Iterációs modell
  - agilis szoftver fejlesztés
  - extrém programozás

V modell: A fejlesztés és a tesztelés ábrázolásáról kapta a nevét, párhuzamos tevékenységek. A fejlesztési ág a vízesés modellből származik, ahol még a része volt a tesztelés a modellnek de csak később szerepben. Minden fejlesztési szinthez tartozik egy tesztelési szint is. Tehát nem akkor kezdődik a tesztelők munkája amikor a fejlesztés véget ért. A funkcionális és nem funkcionális ~~(illeg)~~ követelmények meghatározásakor már elkezdik írni az átvételi teszteseteket, rendszer tervnél az integrációs teszteket, modul tervnél pedig az egység teszteket.

A modell hátránya, hogy ~~bár~~ bár elterjedt, nagyon sok van belőle és nehéz jót találni. Hibásan lineáris folyamatként kezeli a szoftver fejlesztést. Akkor jó ha a fejlesztés során nem változnak a követelmények.

A szoftver fejlesztés valós folyamatait sokkal jobban jellemzi az iterációs modell, a mai fejlesztési technikák pl. agilis szoftver fejlesztés extrém programozás is ide tartoznak. A fejlesztés lépései több körben valósulnak meg átfedésekkel. Tervezés - implementáció - tesztelés - ~~átadás~~ bemutatás - megújítás. A megrendelő igényei nem ismertek meg azonnal illetve változhatnak, ezért folyamatosan bemutatásra kerül a szoftver egy működő modulja a megrendelőnek, egyeztetés után a felmerülő igények, hibás miatt újra írás vagy ha az adott verzió jó akkor a szoftver bővítése új modulokkal. Az agilis szoftver fejlesztés előtérbe helyezi a kapcsolattartást a megrendelővel és a felmerülő igények kielégítését, rugalmasan alkalmazkodik.

3.)

A JUnit egy egységes tesztelési környezetet biztosít java programok ellenőrzéséhez POJO, ad egy kontextust a tesztesetekhez és függvény készletet biztosít. A megírt metódusokhoz készíthetünk teszteket. A ~~m~~ tesztekben különféle állításokat fogalmazhatunk meg. A teszt lefuttatásának három ~~k~~ lehetséges kimenetele lehet: pass, fail, error. Átmegy a teszt, elbukik vagy hibát dob a tesztesetet.

~~Annot~~ Annotációk @Test - teszt futtatás
- ~~k~~ várt kimenet ellenőrzése

A parametrizált tesztek esetében alkalmazzuk a @Parametrizáld anotációt. A parametrizáció egy tömb amiben benne vannak a bemeneti értékek és a várt értékek, ahány ilyen egység van annyiszor kell lefuttatni a tesztet. Így nem kell kézzel mindig beírni az értékeket.

4.

- Modul teszt, egység teszt
- Integrációs teszt
- Rendszer teszt
- Átvételi teszt

Egység teszt: Áll a funkcionális elemek és a nem funkcionális elemek ellenőrzéséből.
Funkcionális elemek tesztelése például Junit segítségével.
Objektum orientált környezetben az objektumokat, metódusokat teszteli. Utasítás vezérelt környezetben változók, függvények.
A modul működését ellenőrzi
Nem funkcionális elemek tesztelése, pl.: sebesség, memória szivárgás, ~~szűk~~ szűk keresztmetszetek keresése

- Integrációs teszt: a szoftver egységek együttműködését vizsgálja.
Interfészeket, függőségeket, adatkapcsolatokat. Lehet modul-modul kapcsolat tesztelés, a modulok közötti kommunikáció ellenőrzése.
Rendszer kapcsolatának tesztelése más rendszerekkel.

- Rendszer teszt: A ~~(sz)~~ teljes rendszer tesztelése már éles környezetben a megrendelőnél, hogyan ~~(int)~~ működik a szoftver a vállalat ~~(...)~~ többi szoftverével együtt.

- Átvételi teszt: - Alfa teszt: a fejlesztő cégnél de nem a fejlesztők által végzett teszt, például monkey-test.
- Béta teszt a felhasználók egy kisebb csoportja megkapja a szoftvert és használja, majd visszajelzéseket küld
- Felhasználói átvételi teszt: majdnem minden felhasználó megkapja a szoftvert, használják, de még nem élesben
- Üzemeltetői átvételi teszt: A rendszergazdák vizsgálják, hogy a különböző, pl. biztonsági funkciók megfelelően működnek-e.