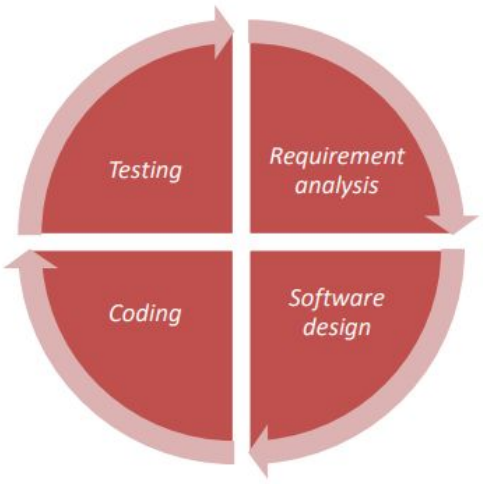
Teoria: Projecte Integrat de Software

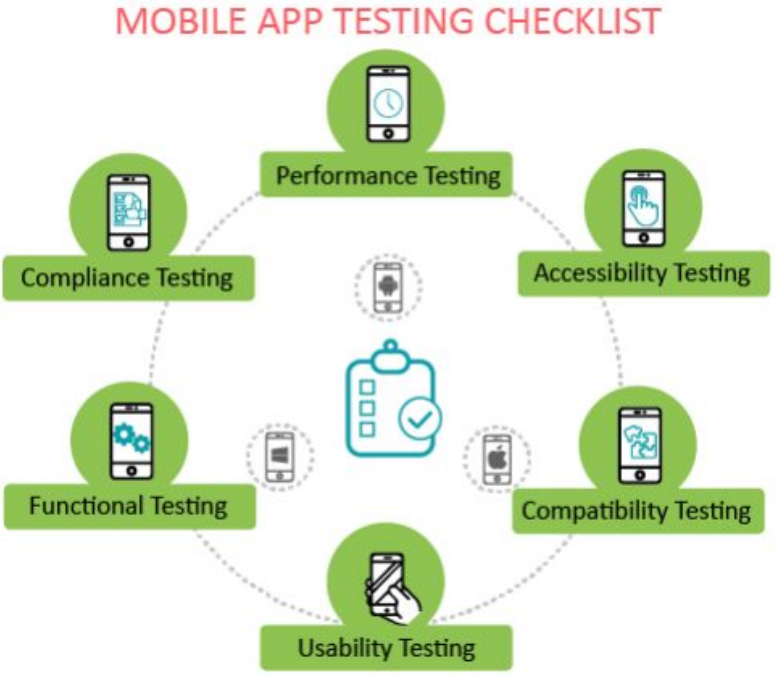
Projecte Integrat de Software UB 2019/20
Grau d'Enginyeria informàtica

Gerard Puch Camacho

Development Lifecycle:



Testing:



Metodologia:

Passos a seguir:

1. **Recopilació de requisits** (Requirements gathering)
2. **Domcumentació de requisits** (Requirements documentation)
3. **Comprensió dels requisits** (Requirements understanding)
4. Tornar al pas 1 o 2.

1.Recopilació de requisits:
Respondre a les següents preguntes (exemple: Hospital): Què? (ex: requisits clínics, d'usuari/pacient, de dispositiu i legals), Qui? (ex: cílinics, enfermerxs, administradors de sistema, pacients, framaceutiques, tecnòlegs, etc) i Com? (ex: entrevistes, enquestes, grups d'enfoc, workshops).

2.Domcumentació de requisits:
Descriu com a mínim el conjunt mínim de requeriments, utilitzar **llenguatge natural** per als **usuaris**, i llenguatge tècnic per als desenvolupadors.

Ha de ser **estructurada:** Definicions, requisits funcionals, requisits del sistema (portabilitat, dispoditius). Els requisits poden estar estructurats en una taula. Es solen incloure **Use-Case** i **Mock-Ups**. Han de seguir els estàndards (com el IEEE).

Un exemple de requisits:

	Example of bad requirement	Example of good requirement
Clear, detailed, complete:	• Students will be able to enroll to undergraduate and post graduate courses	• Students will be able to enroll to undergraduate courses • Students will be able to enroll to post-graduate courses
	1- Students will be able to enroll to undergraduate courses1- Students will be able to enroll to post-graduate courses	1. Course Enrolment 2. Students will be able to enroll to undergraduate courses 3. Students will be able to enroll to post-graduate courses
Concrete, specific:	A professor user will log into the system by providing his username, password, and other relevant information	A professor user will log into the system by providing his username, password and department code
	Each page of the system will load in an acceptable time-frame	Register student and enrol courses pages of the system will load within 5 seconds

3.Comprensió dels requisits:
Procés d'anàlisi dels requisits definits i recerca de la idea més útil i assequible.

Tipus de Test de Software (Software Testing):

Existeixen moltes metodologies de testing, per exemple:
Acceptance, Accessibility, Agile, Ad-hoc, Alpha, API, Automated, Beta, Benchmark, **Black box**, Code-driven, Compatibility, Comparison, Component, Configuration, Compliance, Error-Handling, End-to-end, Endurance, **Functional**, Gray Box, **Integration**, Install/uninstall, Localization, **Non-functional**, Negative, Operational, **Performance**, Regression, Recovery, Requirements, Security, Scenario, Scalability, Stability, Storage, Stress, System, Upgrade, **Unit**, **User Interface**, Volume, **White box**, Workflow.

White-box testing: Es troba dins del software (Estructura, disseny de codi, etc.).

- **Unit Testing.**
- **Integration testing.**

Black-box testing: Basada en la “perspectiva” del usuari final (resultats esperats).

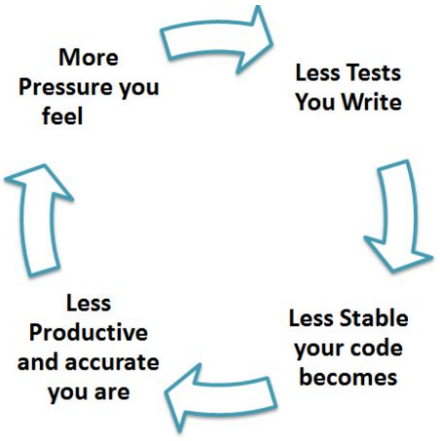
Functional testing: Test de les funcions del software (ús d’aquestes).

Non-Functional testing: Test d’aspectes no relacionats amb les funcions, com la interoperabilitat, l’escalabilitat, càrrega/volum, etc.

Unit Testing

Test de cada funció, classe i component del codi específicament i per separat.
Quan: Durant la fase de desenvolupament (development/coding).
Error típic: Fer el mínim unit testing per estalviar temps, això comporta més defectes i proves de sistema més elevades.
Millor enfocament: Escriure codi per a proves automàtiques d'unitats, verificant el codi per a diversos casos (valors inclosos, valors extrems), a continuació, indicar casos de prova fallits.

S’ha de tornar a fer el unit test cada cop ques es modifica la implementació, però evitant cicles viciosos durant el test:



Integration Testing

És important que les diferents unitats siguin desenvolupades i provades diferents membres de l'equip, prova el funcionament conjunt de les unitats i la comunicació/interfícies entre mòduls.
Mètode 1: Big bang approach, test de tots els mòduls alhora.
Mètode 2: Incremental, test entre dos mòduls inicialment, després amb un tercer, un quart, etc. Aquest apropament facilita la detecció d'errors, però no és útil per a versions inicials del codi.

Performance Testing

Testejar en el software les fites i resultats esperats (en la mesura del possible), això sovint requereix un estudi de test dedicat amb retrospectiva (existents) i prospectiva (noves) de les dades. Aquest procés inclou la definició de procediments i la inclusió de les dades, usuaris i provadors. També requereix mètrica, indicadors i/o criteris.

Usability / Acceptance Testing

Tot si es té un software amb bon rendiment (Performance Testing), és important testeja la utilitat i acceptació dels usuaris finals. Les preguntes que ens hem de fer són:

- Els agrada l'estètica i el disseny?
- És fàcil d'utilitzar el programari? Intuitiu? Els usuaris l'entenen? (**Facilitat d'ús**)
- Si els agrada, amb quina freqüència i quan l'utilitzaran?

Mètodes: Fer servir observadors,, fer enquestes, registrar reaccions dels usuaris (grabació de veu, expressions facials, pantalla de l'activity, etc.).

És essencial triar els usuaris adequats per a fer els tests, així com els entorns de test. També definir la utilitat dels indicadors.

Non-Functional Testing

El programari té un gran rendiment i als usuaris els hi agrada el resultat, però:

- És compatible amb diferents sistemes informàtics, dispositius, etc?
- És portable a altres sistemes?
- És interoperable amb els dispositius americans / japonesos?
- Es pot instal·lar/implementar per a tots els usuaris i el seu entorn?
- És segur? És pot hackejar?
- És escalable a altres dominis, mercats, àrees, usuaris, etc?