2. Eines per al control i documentació de programari

Refacció:

El terme refacció fa referència als canvis efectuats al codi de programació desenvolupat, sense implicar cap canvi en els resultats de la seva execució. És a dir, es transforma el codi font mantenint intacte el seu comportament, aplicant els canvis només en la forma de programar o en l'estructura del codi font, cercant la seva optimització.

Avantatges i limitacions de la refacció:

- **Prevenció** de l'aparició de problemes habituals a partir dels canvis provocats pels manteniments de les aplicacions.
- **Ajuda a augmentar** la simplicitat del disseny.
- **Major enteniment** de les estructures de programació.
- **Detecció** més senzilla d'errors.
- **Permet agilitza**r la programació.
- **Genera satisfacció** en els desenvolupadors.

Limitacions de la refacció:

- **Personal poc preparat** per utilitzar les tècniques de la refacció.
- **Excés d'obsessió** per aconseguir el millor codi font.
- **Excessiva dedicació de temps** a la refacció, provocant efectes negatius.
- **Repercussions en la resta del programari** i de l'equip de desenvolupadors quan un d'ells aplica tècniques de refacció.
- **Possibles problemes de comunicació** provocats pel punt anterior.
- **Limitacions degudes** a les bases de dades, interfícies gràfiques…

NOTA: La **refacció** es considera un aspecte molt important per al desenvolupament d'aplicacions mitjançant programació extrema.

Patrons de refacció més usuals:

- Reanomenar
- Moure
- Extreure una variable local
- Extreure una constant
- Convertir una variable local en un camp
- Extreure una interfície
- Extreure el mètode

Proves i refacció. Eines d'ajuda a la refacció:

- **Casos de proves independents entre mòduls, mètodes o classes**. Els casos de prova han de ser independents per aconseguir que els errors en una part del codi font no afectin altres parts del codi. D'aquesta manera, es poden dur a terme proves incrementals que verifiquin si els canvis que s'han produït amb els processos de refacció han comportat canvis a la resta del programari.
- **Els casos de proves han de ser automàtics**. Si hi ha deu casos de proves, no serà viable que es vagin executant de forma manual un a un, sinó que cal que es puguin executar de forma automàtica tots per tal que, posteriorment, es puguin analitzar els resultats tant de forma individual com de forma conjunta.
- **Casos de proves autoverificables**. Una vegada que les proves s'estableixen de forma independent entre els mòduls i que es poden executar de forma automàtica, només cal que la verificació d'aquestes proves sigui també automàtica, és a dir, que la pròpia eina verifiqui si les proves han estat satisfactòries o no. L'eina indicarà per a quins casos de prova hi ha hagut problemes i en quina part del codi font, a fi que el programador pugui prendre decisions.

Como implementar la refacción:

- Desenvolupar el codi font.
- Analitzar el codi font.
- Dissenyar les proves unitàries i funcionals.
- Implementar les proves.
- Executar les proves.
- Analitzar canvis a efectuar.
- Definir una estratègia d'aplicació dels canvis.
- Modificar el codi font.
- Execució de les proves.

Eines per a l'ajuda a la refacció:

- **Java**: RefactorIt, Condenser, JCosmo, Xrefactory, jFactor, IntelliJ IDEA.
- **Visual C++, Visual C#, Visual Basic .NET**, **ASP.Net**, …: Visual Studio.
- **C++**: CppRefactory, Xrefactory.
- **C#**: C# Refactoring Tool, C# Refactory.
- **SQL**: SQL Enlight.
- **Delphi**: Modelmaker tool, Castalia.
- **Smalltalk**: RefactoringBrowser.

Control de versions:

Un **sistema de control de versions** és una eina d'ajuda al desenvolupament de programari que anirà emmagatzemant, segons els paràmetres indicats, la situació del codi font en moments determinats. És com una eina que va fent fotos de forma regular, cada cert temps, sobre l'estat del codi.

Algunes de les funcionalitats:

- **Comparar canvis en els diferents arxius al llarg del temps**, podent veure qui ha modificat per darrera vegada un determinat arxiu o tros de codi font.
- **Reducció de problemes de coordinació que pot haver-hi entre els diferents programadors**. Amb els sistemes de control de versions podran compartir la seva feina, oferint les darreres versions del codi o dels documents, i treballar, fins i tot, de forma simultània sense por a trobar-se amb conflictes en el resultat d'aquesta col·laboració.
- **Possibilitat d'accedir a versions anteriors dels documents o codi font**. De forma programada es podrà automatitzar la generació de còpies de seguretat o, fins i tot, emmagatzemar tot canvi efectuat. En el cas d'haver-se equivocat de forma puntual, o durant un període llarg de temps, el programador podrà tenir accés a versions anteriors del codi o desfer, pas a pas, tot allò desenvolupat durant els darrers dies.
- **Veure quin programador ha estat el darrer a modificar un determinat tros de codi** que pot estar causant un problema.
- **Accés a l'historial de canvis sobre tots els arxius a mesura que avança el projecte**. També pot servir per al cap de projectes o per a qualsevol altra part interessada (stakeholder), amb permisos per accedir a aquest historial, per veure l'evolució del projecte.
- **Tornar un arxiu determinat** o tot el projecte sencer a un o a diversos estats anteriors.
- **Control històric detallat de cada arxiu**. Permet emmagatzemar tota la informació del que ha succeït en un arxiu, com ara tots els canvis que s'han efectuat, per qui, el motiu dels canvis, emmagatzemar totes les versions que hi ha hagut des de la seva creació…
- **Control d'usuaris amb permisos per accedir als arxius**. Cada usuari tindrà un tipus d'accés determinat als arxius per poder consultar-los o modificar-los o, fins i tot, esborrar-los o crear-ne de nous. Aquest control ha de ser gestionat per l'eina de control de versions, emmagatzemant tots els usuaris possibles i els permisos que tenen assignats.
- **Creació de branques d'un mateix projecte**: en el desenvolupament d'un projecte hi ha moments en què cal ramificar-lo, és a dir, a partir d'un determinat moment, d'un determinat punt, cal crear dues branques del projecte que es podran continuar desenvolupant per separat. Aquest cas es pot donar en el moment de finalitzar una primera versió d'una aplicació que es lliura als clients, però que cal continuar evolucionant.
- **Fusionar dues versions d'un mateix arxiu**: permetent fusionar-les, agafant, de cada part de l'arxiu, el codi que més interessi als desenvolupadors. Aquesta funcionalitat s'haurà de validar manualment per part d'una persona.

Components d'un sistema de control de versions:

- **Repositori** (repository o depot): conjunt de dades emmagatzemades, també referit a versions o còpies de seguretat.
- **Mòdul** (module): es refereix a una carpeta o directori específic del repositori. Un mòdul podrà fer referència a tot el projecte sencer o només a una part del projecte, és a dir, a un conjunt d'arxius.
- **Tronc** (trunk o master): estat principal del projecte. És l'estat del projecte destinat, en acabar el seu desenvolupament, a ser passat a producció.
- **Branca** (branch): és una bifurcació del tronc o branca mestra de l'aplicació que conté una versió independent de l'aplicació i a la qual poden aplicar-se canvis sense que afectin ni el tronc ni altres branques. Aquests canvis, en un futur, poden incorporar-se al tronc.
- **Versió o Revisió** (version o revision): és l'estat del projecte o d'una de les seves branques en un moment determinat. Es crea una versió cada vegada que s'afegeixen canvis a un repositori.
- **Etiqueta** (tag, label o baseline): informació que s'afegirà a una versió. Sovint indica alguna característica específica que el fa especial. Aquesta informació serà textual i, moltes vegades, es generarà de forma manual. Per exemple, es pot etiquetar la primera versió d'un programari (1.0) o una versió en la qual s'ha solucionat un error important.
- **Cap** (head o tip): fa referència a la versió més recent d'una determinada branca o del tronc. El tronc i cada branca tenen el seu propi cap, però, per referir-se al cap del tronc, de vegades s'utilitza el terme HEAD, en majúscules.
- **Clonar** (clone): consisteix a crear un nou repositori, que és una còpia idèntica d'un altre, ja que conté les mateixes revisions.
- **Bifurcació** (fork): creació d'un nou repositori a partir d'un altre. Aquest nou repositori, al contrari que en el cas de la clonació, no està lligat al repositori original i es tracta com un repositori diferent.
- **Pull**: és l'acció que copia els canvis d'un repositori (habitualment remot) en el repositori local. Aquesta acció pot provocar conflictes.
- **Push** o fetch: són accions utilitzades per afegir els canvis del repositori local a un altre repositori (habitualment remot). Aquesta acció pot provocar conflictes.
- **Canvi** (change o diff): representa una modificació concreta d'un document sota el control de versions.
- **Sincronització** (update o sync): és l'acció de combinar els canvis fets al repositori amb la còpia de treball local.
- **Conflicte** (conflict): es produeix quan s'intenten afegir canvis a un fitxer que ha estat modificat prèviament per un altre usuari. Abans de poder combinar els canvis amb el repositori s'haurà de resoldre el conflicte.
- **Fusionar** (merge o integration): és l'acció que es produeix quan es volen combinar els canvis d'un repositori local amb un remot i es detecten canvis al mateix fitxer en tots dos repositoris i es produeix un conflicte. Per resoldre aquest conflicte s'han de fusionar els canvis abans de poder actualitzar els repositoris. Aquesta fusió pot consistir a descartar els canvis d'un dels dos repositoris o editar el codi per incloure els

canvis del fitxer a totes dues bandes. Cal destacar que és possible que un mateix fitxer presenti canvis en molts punts diferents que s'hauran de resoldre per poder donar la fusió per finalitzada.

- **Bloqueig** (lock): alguns sistemes de control de versions en lloc d'utilitzar el sistema de fusions el que fan és bloquejar els fitxers en ús, de manera que només pot haver-hi un sol usuari modificant un fitxer en un moment donat.
- **Directori de treball** (working directory): directori al qual el programador treballarà a partir d'una còpia que haurà fet del repositori en el seu ordinador local.
- **Còpia de treball** (working copy): fa referència a la còpia local dels fitxers que s'han copiat del repositori, que és sobre la qual es fan els canvis (és a dir, s'hi treballa, d'aquí el nom) abans d'afegir aquests canvis al repositori. És emmagatzemada al directori de treball.
- **Tornar a la versió anterior** (revert): descarta tots els canvis produïts a la còpia de treball des de l'última pujada al repositori local.

Classificació dels sistemes de control de versions:

- **Sistemes locals** (en un mismo PC): tiene riesgo puesto que se almacena en un único disco.
- **Sistemes centralitzats** (varios PC)**:** En els sistemes centralitzats hi haurà més d'un programador desenvolupant un projecte en més d'un ordinador. Des dels dos ordinadors es podrà accedir als mateixos arxius de treball i sobre les mateixes versions emmagatzemades.
- **Sistemes distribuïts**: Els sistemes de control de versions distribuïts ofereixen una solució a aquest desavantatge ofert pels sistemes de control de versions centralitzats. Com es pot veure a la figura.7, la solució que ofereixen els sistemes distribuïts és disposar cada ordinador de treball, així com el servidor, d'una còpia de les versions emmagatzemades. Aquesta duplicitat de les versions ofereix una disponibilitat que disminueix moltíssim les possibilitats de no tenir accessibilitat als arxius i a les seves versions.

Operacions bàsiques d'un sistema de control de versions:

Introducción de dades:

- **Importació de dades**: aquesta operació permet dur a terme la primera còpia de seguretat o versionat dels arxius amb els quals es treballarà en local cap al repositori.
- **Pujar** (commit o check in): aquesta operació permet enviar al repositori les dades corresponents als canvis que s'han produït en el servidor local. No es farà una còpia sencera de tota la informació, sinó que només es treballarà amb els arxius que s'hagin modificat en el darrer espai de temps. Cal destacar que no els envia al servidor; els canvis queden emmagatzemats al repositori local, que s'ha de sincronitzar.

Operacións d'exportacións:

- **Baixar** (check-out): amb aquesta operació es podrà tenir accés i descarregar a l'àrea de treball local una versió des d'un repositori local, un repositori remot o una branca diferent.
- **Actualització** (update): aquesta operació permet dur a terme una còpia de seguretat de totes les dades del repositori a l'ordinador client amb què treballarà el programador. Serà una operació que es podrà efectuar de forma manual, quan el programador ho estimi oportú, o de forma automàtica, com en els sistemes distribuïts, on cada vegada que un client accedeix al repositori es fa una còpia completa en local.

Eines de control de versions:

- Team Foundation Server
- CVS - Concurrent Versions System
- Subversion
- Mercurial
- Git

Dipòsits de les eines de control de versions:
Un **dipòsit** és un magatzem de dades on es guardarà tot allò relacionat amb una aplicació informàtica o unes dades determinades d'un determinat projecte.

Un **repositori** és la part principal d'un sistema de control de versions. Són sistemes dissenyats per enregistrar, guardar i gestionar tots els canvis i informacions sobre totes les dades d'un projecte al llarg del temps.

- **Consultar** la darrera versió dels arxius que s'hagin emmagatzemat.
- **Accedir** a la versió d'un determinat dia i comparar-la amb l'actual.
- **Consultar** qui ha modificat un determinat tros de codi i quan va ser modificat.

Problemàtiques dels sistemes de control de versions:
Els sistemes de control de versions han de resoldre certes problemàtiques, com per exemple, com evitar que les accions d'un programador se sobreposin a les d'altres programadors.
Solucións:

- **Compartir** arxius per part de diferents programadors
- **Bloquejar** els arxius utilitzats
- **Fusionar** els arxius modificats

Utilització de Git:
Git és d'un programari de control de versions que utilitza un sistema distribuït i, per conseqüent, cada usuari que clona un repositori obté una còpia completa dels fitxers i l'historial de canvis.

Comandos:
git --version : Ver versión git instalada.
git init : Inicializar/crear repositorio (se crea en el directorio donde se está ubicado). Se creará una carpeta oculta. Se puede acceder a ella tecleando ***cd .git*.**
git status : mostra l'estat actual de la còpia de treball.
git add . : afegeix tots els fitxers al sistema de control de versions.
git add *.html : afegeix tos els fitxers amb extensió "html". També es pot utilizar "*git add <<nombre archivo*".
git rm index.html --cache : Eliminar un arxiu del control de versións.
git commit -m "texto" : Per pujar els fitxers al repositori i començar a enregistrar aquests canvis s'ha d'utilitzar l'ordre commit. El paràmetre -m serveix per afegir un comentari i s'utilitza per afegir informació extra sobre els canvis que inclou la versió (el conjunt de canvis realitzats).
git clone https://github.com/XavierGaro/client-servidor-xat.git : Una altra opció a l'hora de crear repositoris és clonar un repositori existent. Per fer-ho s'utilitza l'ordre git clone, indicant l'adreça del repositori que es vol clonar.

Resumen:

- **git init**: inicialitza un repositori.
- **git add** : afegeix elements de la còpia de treball al control de versions.
- **git rm –cache**: elimina elements del control de versions.
- **git status**: mostra l'estat de la còpia de treball.
- **git commit**: puja els canvis de la còpia de treball sota el control de versions al repositori local.
- **git log**: mostra la llista de versions pujades al repositori local.
- **git clone**: copia un repositori remot, no cal inicialitzar-lo.

NOTA: Configuració de l'adreça de correu a Git → Podeu configurar l'autor i l'adreça de correu utilitzada per Git amb les ordres: **git config –global user.name "Autor"** i **git config –global user.email email@exemple.cat**.

Operacions avançades:

- **git branch**: crea noves branques o llista les branques del repositori → *git branch nova-branca* o si se quiere volver una versión anterior al control de cambios, utilizamos "checkout" pero en lugar de indicar la rama indicamos el indentificador de la versión, por ejemplo, si queremos volver a la versión donde se hizo el paso 2 y su identificador es commit **3f9365181b055c0b956e3bdf97a6e3a37085927f**, la orden que tendríamos que utilizar es: "*git checkout 3f93651*".
- **git checkout**: canvia la còpia de treball a la branca o versió indicada → *git checkout nova-branca*.
- **git diff**: mostra els canvis que s'han afegit en una versió.
- **git log**: mostra la llista de versions per la branca activa.
- **git merge**: fusiona els canvis entre dues branques → *git merge nova-branca*.
- **git tag**: afegeix una etiqueta a una versió → *git tag -a v1.0 -m "Primera versió"*. L'opció -a indica el text de l'etiqueta i l'opció -m permet afegir informació addicional.
- **git show**: mostra informació sobre la versió indicada.
- **git reset –hard**: reverteix els canvis a la còpia de treball.
- **git pull**: puja els canvis del repositori local a un repositori remot.
- **git push**: baixa els canvis d'un repositori remot al repositori local.
- **git remote**: afegeix un repositori remot o llista els repositoris remots enllaçats amb el repositori local.
- **fitxer .gitignore**: permet afegir una llista de fitxers i directoris per excloure del sistema de control de versions.