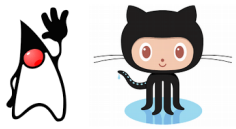
Sistemes de Control de Versions (SCV) i Documentació

Autor: Vicent Vendrell

Índex de continguts

Autor: Vicent Vendrell Tarazona

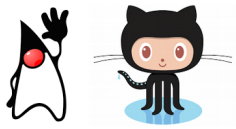
1. Introducció

Un parell d'aspectes importants en la vida d'un projecte de software són per una part la documentació que es genera, i per l'altra com es gestiona la informació o codi que es va creant.

Respecte de la **documentació** que s'ha de generar, el primer que hem de preguntar-nos és què s'ha de documentar i perquè. També hem de veure quins tipus de documents hem de generar, quin format tindran i com podem generar-los. Per últim i depenent del llenguatge de programació que seleccionem, haurem d'utilitzar una sèrie d'eines o unes altres.

Pel que respecta a la gestió de la informació que es crea, el codi font que es desenvolupa i les **versions** que es van generant, haurem d'analitzar quina és la millor manera de gestionar aquesta informació, i veure si podem automatitzar el procés per alliberar de tasques repetitives als desenvolupadors de programari. Per últim haurem d'estudiar la millor manera de gestionar les diferents versions que generem amb la finalitat de poder fer proves amb elles i poder tornar enrere en un moment determinat del nostre projecte.

La documentació que es genera a un projecte i les diferents versions del programari que estem desenvolupant, van a influir en gran manera segons el model de desenvolupament de projecte o cicle de vida del programari que seguim.



2. El cicle de vida del software

El cicle de vida del software, també és conegut com procés de desenvolupament de programari. Consisteix a una metodologia estructurada, aplicada al desenvolupament de programari. Existeixen diversos models on cadascun té un focus diferent segons les activitats que es volen dur a terme. Normalment es basen en processos i l'objectiu és dissenyar un sistema de processos reproduïbles amb la finalitat de millorar la productivitat i la qualitat.

Un model de desenvolupament de programari és una representació abstracta sobre una sèrie de tasques agrupades en un únic procés. Es basen en diversos **paradigmes** com són el tradicional, l'orientat a objectes i el paradigma àgil.

Paradigma tradicional: aquest paradigma es centra a distribuir el projecte per etapes. El problema que presenta és que les etapes no són independents entre elles i qualsevol canvi o error detectat en una fase, implica tornar a fases anteriors per refer-ho tot de nou.

Paradigma orientat a objectes: és un paradigma que es basa en la programació orientada a objectes, per tant apareixen conceptes com classe, anàlisi de requeriments i disseny. Permet la reutilització de programari i es poden utilitzar eines de desenvolupament automàtic.

Paradigma de desenvolupament àgil: és basa en la gestió de processos de forma àgil. La idea principal d'aquest és dividir els processos en parts molt petites on tots els membres del projecte col·laboren desenvolupant el projecte i validant-ho al mateix temps.

En base als paradigmes anteriors, ens podem trobar diversos **models de desenvolupament.**

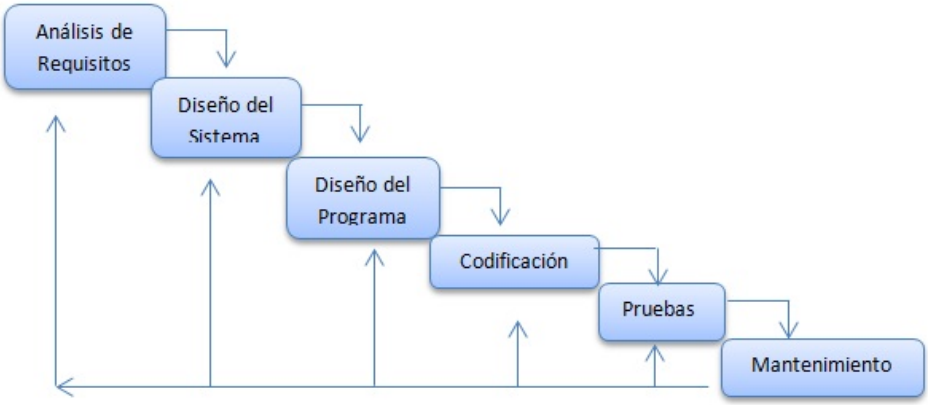
Segons el model de desenvolupament de programari, la documentació que s'ha de generar s'haurà de fer en un moment o altre del projecte, també s'hauran de documentar certs aspectes, que poden ser realment importants per a fases posteriors del projecte. El mateix passa amb el codi del projecte generat. Segons el model seleccionat, haurem de gestionar el codi font desenvolupat d'una manera o altra, depenent de quina sigui la millor manera de gestió, depenent de la tipologia del projecte i el cicle de vida del programari a seguir.

2.1 Models de desenvolupament

2.1.1 Model en cascada

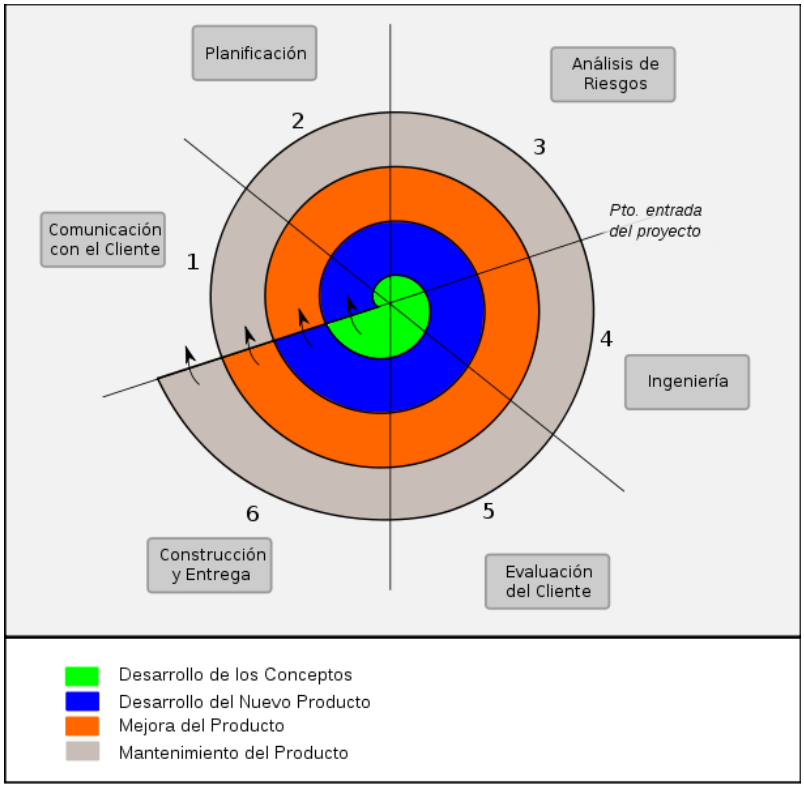
Es basa en el paradigma tradicional on el desenvolupament del projecte es basa en una sèrie de fases consecutives que s'han de complir. Les fases que implica són especificació de requeriments, disseny, implementació, integració, proves, desplegament i manteniment. Cada fase produirà una sèrie d'informació que servirà a la següent fase com dades d'entrada. El problema que presenta és que cada fase ha de ser molt clara i les dades d'entrada i sortida de cada fase han d'estar

perfectament definides. Un altre problema que presenta el model és la falta de flexibilitat, un cop acabada una fase no es aconsellable tornar enrere, perquè sinó el projecte no finalitzaria mai.



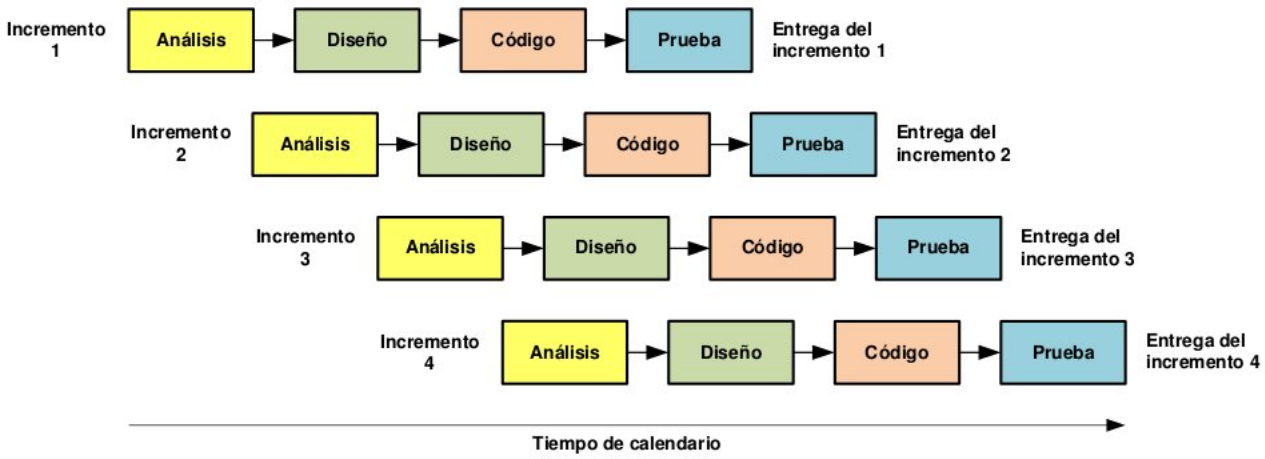
2.1.2 Model en espiral

És un model que es basa en l'anàlisi de riscos del projecte i la repetició d'una sèrie de tasques en diverses iteracions. No es basa en fases com ho fa el model en cascada. Les tasques que s'han de fer són la planificació i definició d'objectius, l'anàlisi de riscos, el desenvolupament i proves, i finalment l'avaluació. Existeixen algunes variants amb altres tasques modificades o ampliades. El problema que presenta aquest model és que es centra en els riscos i açò comporta que els desenvolupadors han de buscar de forma explícita els riscos i analitzar-los de forma exhaustiva perquè el model funcioni. També requereix interacció continua entre client i desenvolupadors.



2.1.3 Model incremental o iteratiu

Aquest model el que tracta de resoldre és el problema que genera el model en cascada. Es basa en un conjunt de tasques agrupades en petites etapes repetitives (iteracions). La idea és començar amb un anàlisi simple de requeriments del projecte i iterativament millorar les versions del projecte. Les fases de les quals consta aquest model són les mateixes que el cicle de vida en cascada. A cada fase es realitzen canvis al disseny i s'afegeixen noves funcionalitats i capacitats al projecte.



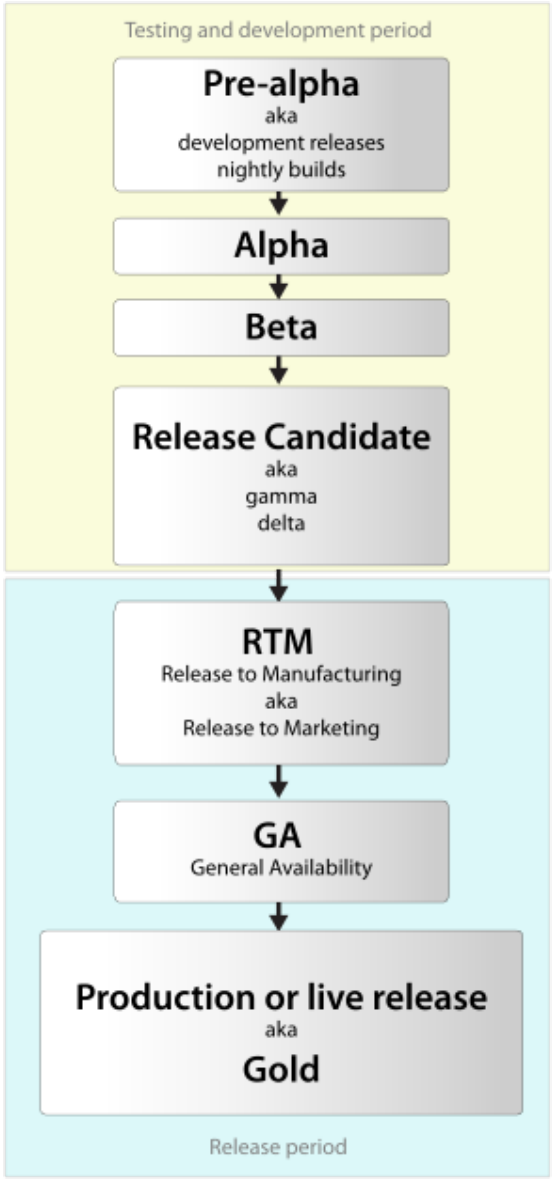
2.1.4 Desenvolupament àgil

És un model que es basa en iteracions. Els requeriments i les solucions aportades evolucionen en el temps segons les necessitats del projecte. El treball desenvolupat es fonamenta en la col·laboració d'equips de treball petits involucrats en un procés on s'han de prendre decisions a curt termini. Cada iteració inclou planificació, anàlisi de requeriments, disseny, codificació, proves i documentació. La finalitat de cada iteració és produir una part del projecte funcional i sense errors de manera que no es necessari tornar enrere per redissenyar el projecte.

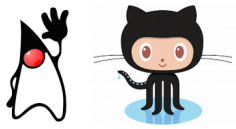
2.2 Versions de programari

Durant la vida d'un projecte podem tindre diverses versions del nostre projecte. Aquestes versions les podem empaquetar per començar a fer proves amb elles. Alguna terminologia que s'utilitza amb les versions de programari són les següents:

1. Pre-Alpha: versió preliminar del producte
2. Alpha: versió amb tots els requisits
3. Beta: Fase de prova, demo i correcció d'errors
    1. Open Beta: un grup gran de gent prova la versió
    2. Closed Beta: un grup reduït i tancat prova la versió
4. Candidata – (release candidate, RC): versió candidata a producte final
5. Versió per a fabricació (release to manufacturing, RTM): versió disponible per a ser llançada al gran públic
6. Disponibilitat general (general availability, GA): versió disponible al gran públic



https://en.wikipedia.org/wiki/Software_release_life_cycle



3. Documentació

Tots els projectes de programari han de disposar d'una documentació consistent, de manera que en qualsevol etapa del seu cicle de vida, començant en la seva fase d'anàlisi i disseny, passant per la seva fase de codificació o implementació en un llenguatge de programació determinat i, fins i tot, en la fase d'explotació, ha d'haver una documentació robusta amb la finalitat de subministrar informació rellevant sobre el codi, funcionalitats de l'aplicació o, fins i tot, limitacions de la mateixa, perquè l'aplicació pugui ser explotada en la seva amplitud i perquè el seu manteniment resulti còmode.

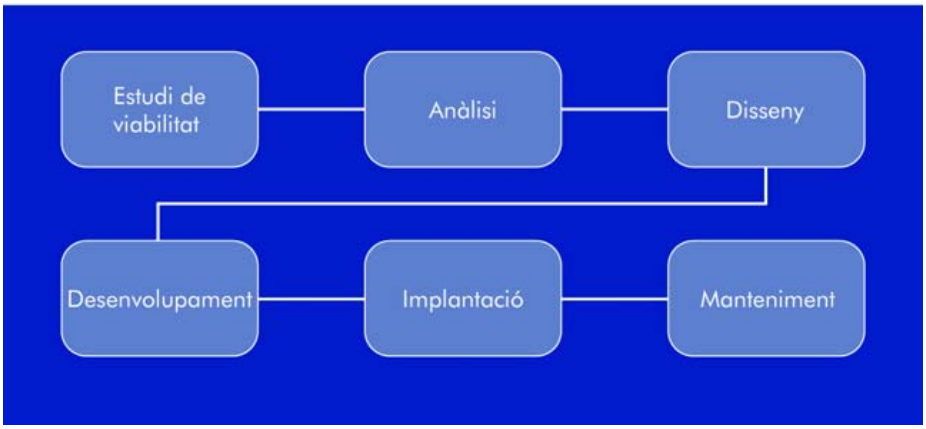
3.1 Introducció a la gestió de projectes

Per a dur a terme un projecte de tecnologies de la informació basat en la utilització d'eines d'ús habitual a Internet (com per exemple el World Wide Web), és necessari seguir un procés que ens porti des de la comprensió de l'abast del problema que volem solucionar fins a la implantació i manteniment de la solució que hàgim triat.

Aquestes fases seran presents, d'una manera o una altra, amb aquests noms o amb d'altres, en qualsevol projecte web, des dels gestionats mitjançant mètodes "clàssics", fins als gestionats com suggereix el conjunt de metodologies conegudes com a àgils.

Hem de conèixer quines **fases** s'han de seguir al llarg de qualsevol projecte. Aquestes s'enumeren a continuació:

- **Estudi de viabilitat**: s'estudiarà en línies generals quins problemes es volen resoldre, quines solucions possibles hi ha i quina és la més adequada.
- **Anàlisi**: es descriurà detalladament el sistema que es vol construir, quins requisits ha de complir i quins usuaris ha de satisfer.
- **Disseny**: es realitzarà el plantejament tecnològic de la solució.
- **Desenvolupament**: es durà a terme la programació, integració, instal·lació, etc. dels diferents subsistemes que componguin el projecte.
- **Implantació**: es passarà el sistema construït a producció, a fi que els usuaris el comencin a utilitzar.
- **Manteniment**: es faran tant les correccions dels possibles errors que puguin sorgir en el sistema implantat com les millores evolutives que es considerin oportunes.



3.2 Què i perquè s'ha de documentar?

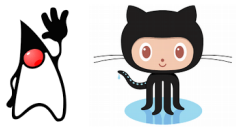
Tal com s'ha comentat a l'apartat anterior, tot projecte sempre va a estar format per una sèrie de fases que hem de seguir per poder dur-lo a terme. Cada projecte que generem tindrà una forma o altra depenent del tipus de metodologia que decidim utilitzar, però en certa manera les fases seran semblants a les exposades.

En tota gestió d'un projecte, cadascuna de les fases està encadenada a una fase posterior, de manera que tota la informació que es genera a una fase, serveix com a informació d'entrada per la següent. Hem de conèixer amb quina informació treballem a cada fase i quina documentació hem de generar per poder donar-li continuïtat i qualitat al projecte.

3.2.1 Estudi de viabilitat

Tracta aspectes econòmics, tècnics, legals i operatius del projecte i la solució que es desenvolupa. En aquesta primera fase es tracta de donar una descripció general del problema que volem resoldre. Per tant hem de documentar:

- Abast del sistema: és una descripció general de les necessitats plantejades pel client en el qual apareixeran els aspectes citats anteriorment.
- Situació actual: es tracta de saber quin és el punt de partida del sistema on es va a desenvolupar el projecte. Un sistema pot ser un negoci ja existent amb unes línies de negoci determinades, o pot ser un sistema que es crea de zero, per exemple. D'aquest sistema hem de fer una descripció general del seu funcionament, elements que el contenen, restriccions, etc.
- Definició dels requisits del sistema: simplement descriurem de forma general quins són els requisits que haurà de complir el projecte que estem desenvolupant.



- Estudi, valoració i selecció de la solució: amb tota la informació obtinguda als punts anteriors, hem d'estudiar com implementar la solució, valorar les diverses solucions que puguen sorgir i decidir quina serà la solució a implementar al nostre projecte. D'aquest punt també sorgiran quins són els requeriments per poder dur a terme el projecte (tecnològics, humans, temporals, riscos...).

3.2.2 Anàlisi

En aquesta fase hem d'especificar detalladament la solució a desenvolupar al nostre projecte. La informació per dur a terme la fase d'anàlisi ens vindrà de la fase anterior (estudi de viabilitat). En aquesta fase hem de documentar:

- Definició del sistema: descripció detallada del sistema, establint com s'ha de comunicar amb els altres i quins són els seus usuaris més representatius.
- Establiment de requisits: s'han de completar els requisits definits a la fase anterior, definint-los i detallant-los al màxim possible.
- Definició d'interfícies d'usuari: definirem els perfils d'usuari, els casos d'ús i la interacció.
- Definició del pla de proves: aquest ens servirà per establir si el sistema compleix els requisits establerts pels usuaris.

3.2.3 Disseny

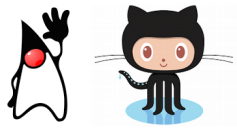
L'objectiu d'aquesta fase és obtenir els models i especificacions que defineixen el projecte a partir de l'anàlisi realitzada a la fase anterior. Les activitats que duguem a terme en aquesta fase ens permetran determinar les especificacions de desenvolupament i integració, i definir l'entorn de proves i implantació necessaris per al seu funcionament correcte.

- Model conceptual i lògic: identificar les parts de que consta el sistema a desenvolupar i quina relació existeix entre ells. El més comú és utilitzar diagrames UML i targetes CRC.
- Estàndards: definició de plantilles i formats dels documents, normatives, estils, idioma, interfícies de l'aplicació...

3.2.4 Desenvolupament

En aquesta fase es tracta de construir de forma ordenada el sistema que hem analitzat i dissenyat.

- Planificació: elaboració d'un diagrama de Gantt amb les tasques a desenvolupar. D'aquestes s'han de saber el seu inici i fi, recursos necessaris, dependències amb altres activitats, etc.



- Codificació: es tracta d'indicar com s'estructura l'aplicació, la distribució dels fitxers, les funcions i les decisions que es prenen. Cada fitxer ha de contenir una capçalera on es detalla informació sobre el contingut, cada classe i funcionalitat. Han d'explicar perquè serveixen i quines són les seves funcionalitats.
- Documentació: en aquest cas es tracta de generar la documentació associada a l'aplicació. Dintre d'aquesta podem trobar manuals d'usuari i d'administrador, manual d'infraestructura, referències a altra documentació, etc.

3.2.5 Implantació

Aquesta fase implica el pas a producció del sistema desenvolupat. Haurem de tenir en compte l'escenari o ecosistema sobre el qual s'implantarà la solució, a qui pot afectar, la planificació temporal, les proves al nou sistema, si s'ha de fer formació als usuaris, etc.

- Document d'implantació: contindrà com a mínim un manual d'instal·lació, requeriments de maquinari i programari, etc.
- Formació: curs a mida segons el tipus d'usuari.
- Proves del sistema: genèriques sobre serveis, sobre la base de dades, sobre els formularis, etc.

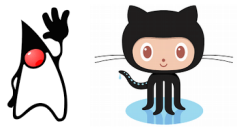
3.2.6 Manteniment

Aquesta és l'última fase del projecte, on s'ha de definir la continuïtat que se li donarà al projecte. Hem de decidir si es crearan noves funcionalitats, si es continua amb alguna altra fase del projecte, si es donarà suport o servei tècnic a l'aplicació, etc.

3.3 Tipus de documentació

Com s'ha estudiat als punts anteriors, durant la vida d'un projecte es pot generar gran quantitat d'informació i a més a més de diversos tipus. Podem tindre:

- Documentació del projecte: aquesta documentació és la que necessitem per poder generar projectes de qualitat. Com s'ha vist, a un projecte s'ha de documentar tot allò que es vol fer, inclús les decisions que s'han de prendre, també requisits i requeriments.
- Documentació tècnica: es tracta de documentació amb cert caràcter especialitzat, on podem trobar detalls del tipus de programari i maquinari que intervé al projecte, especificacions de funcions, funcionalitats concretes, manuals d'instal·lació i manteniment, etc.



- Documentació d'usuari: en aquesta part ens podem trobar tots els manuals d'usuari i d'administrador que s'han de generar, podem incloure també tutorials formatius.

Aquest tipus de documentació es podrà generar bàsicament de dues maneres:

- Documentació manual: És documentació que es redacta mitjançant un editor de text. Exemples d'aquest tipus de documentació són el manual d'usuari o d'instal·lació. Solen incloure imatges per fer més entenedores les explicacions.
- Documentació automàtica: És documentació que es pot generar de forma automàtica mitjançant eines que podem incorporar al nostre projecte. El document típic que es sol generar és la API de la nostra aplicació o descripció de les funcionalitats i llibreries.

3.3.1 Com s'ha de documentar el codi

La forma en la qual afegim comentaris al nostre codi és important. És tan important que abans de començar a inserir comentaris, tot l'equip de desenvolupament s'ha de posar d'acord sobre com van a fer aquesta tasca. Per poder fer açò existeixen el que s'anomena convenció de codi per als llenguatges de programació.

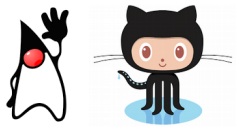
Una convenció de codi, simplement és un recull de bones pràctiques a seguir a l'hora de codificar i documentar el codi dels nostres projectes. Aquestes convencions han de recollir la guia d'estils i bones pràctiques sobre:

- Noms de fitxers i extensions
- Organització dels fitxers
- Indentació o sagnat
- Comentaris i tipus de comentaris
- Declaració de variables, mètodes, classes i fitxers

Al següent web podem trobar les convencions per a documentar amb el llenguatge de programació Java:

http://www.oracle.com/technetwork/java/codeconvtoc-136057.html

http://www.oracle.com/technetwork/java/codeconventions-150003.pdf



3.4 Eines i llenguatges de programació

Les eines de generació de documentació permeten crear una documentació per a programadors (API) o per a usuaris finals a partir de comentaris introduïts en el codi font. Un exemple és Javadoc de Java amb el qual si comentem el nostre codi seguint unes pautes determinades podem obtenir una documentació similar a la de l'API de Java per a les nostres pròpies classes.

Si anem a programar en Java principalment i l'ús que pretenem donar és principalment documentar una API de les nostres classes, Javadoc és l'opció òbvia. Per usos més específics hauríem de buscar una que s'ajusti a les nostres necessitats.

Altres eines de gestió de la documentació automàtica són PHPDocumentor o Doxygen.

EINA	DESCRIPCIÓ	ENTORN / LLENGUATGE
JavaDoc	Genera de forma automàtica la documentació del codi d'un programa en format HTML.	JAVA
PHPDocumentor	Automàticament analitza el codi font PHP i produeix l'API de lectura i documentació del codi font en una varietat de formats.	PHP
Doxygen	És l'eina estàndard per a la generació de documentació de codiescrit en C ++, però també és compatible amb altres llenguatges de programació populars com C, Objective-C, C #, PHP, Java, Python, IDL...	C++, C, Java, Objective-C i Python, entre d'altres.

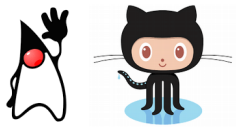
Podem veure una comparativa d'eines de documentació al següent web:

https://en.wikipedia.org/wiki/Comparison_of_documentation_generators

3.4.1 JavaDoc

JavaDoc, és un generador de documentació automàtica creat per SunMicrosystems per al llenguatge Java. Normalment s'utilitza per generar APIs (Aplication Programing Interface). El format de sortida és HTML i sol estar integrat en la majoria de IDEs que existeixen actualment.

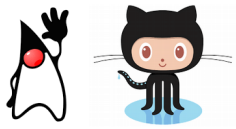
L'objectiu d'aquesta eina, és mantenir al dia la documentació que es va generant als nostres projectes. Al codi font s'escriu la documentació bàsica (classes, mètodes, etc). L'eina extrau aquesta informació del codi font per generar un fitxer HTML.



La informació que ha de recollir JavaDoc l'ha de recollir del codi font, i aquesta informació utilitzarà una nomenclatura especial a base de comentaris i caràcters especials (normalment s'anomenen etiquetes).

Alguns exemples d'etiquetes els podem veure a la següent imatge:

Tag & Parameter	Usage	Applies to	Since
@author *John Smith*	Describes an author.	Class, Interface, Enum	
@version *version*	Provides software version entry. Max one per Class or Interface.	Class, Interface, Enum	
@since *since-text*	Describes when this functionality has first existed.	Class, Interface, Enum, Field, Method	
@see *reference*	Provides a link to other element of documentation.	Class, Interface, Enum, Field, Method	
@param *name description*	Describes a method parameter.	Method	
@return *description*	Describes the return value.	Method	
@exception *classname description* **@throws** *classname description*	Describes an exception that may be thrown from this method.	Method	
@deprecated *description*	Describes an outdated method.	Class, Interface, Enum, Field, Method	
{**@inheritDoc**}	Copies the description from the overridden method.	Overriding Method	1.4.0
{**@link** *reference*}	Link to other symbol.	Class, Interface, Enum, Field, Method	
{**@value** *#STATIC_FIELD*}	Return the value of a static field.	Static Field	1.4.0
{**@code** *literal*}	Formats literal text in the code font. It is equivalent to <code>{@literal}</code>.	Class, Interface, Enum, Field, Method	1.5.0
{**@literal** *literal*}	Denotes literal text. The enclosed text is interpreted as not containing HTML markup or nested javadoc tags.	Class, Interface, Enum, Field, Method	1.5.0



Exemple de classe comentada:

```
/**
 * Constructor de la classe
 * <p>Constructor buit de la classe<p>
 */
public Util (){
}

/**
 * Funció buida que implementa les funcionalitats de la classe Matematiques
 * <p>
 * Des d'aquest mètode utilitzem les funcionalitats que ens proporciona
 * la classe Matemàtiques amb pas de paràmetres d'entrada
 * <p>
 *
 * @param  num1 paràmetre sencer
 * @param  num2 paràmetre sencer
 *
 * @see Matematiques
 */
public void fesMatematiques(int num1, int num2){
    System.out.println(Matematiques.suma(num1,num2));
    System.out.println(Matematiques.resta(num1,num2));
    System.out.println(Matematiques.multiplica(num1,num2));
    System.out.println(Matematiques.divideix(num1,num2));
}
```

Exemple de mètode comentat:

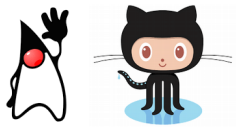
```
/**
 * Mètode per sumar números demanant-los per consola
 *
 * Demana un número a l'usuari i l'emmagatzema a una variable, després
 * demana un segon número i finalment mostra el resultat de l'operació
 * suma per consola.
 */
public static void suma(){
    Scanner lector = new Scanner(System.in);
    System.out.println("donam un número sencer per fer la suma");
    /**
     * Varaible de tipus sencer per realitzar l'operació suma.
     */
    int num1 = lector.nextInt();
    System.out.println("donam un altre número sencer");
    /**
     * Varaible de tipus sencer per realitzar l'operació suma.
     */
    int num2 = lector.nextInt();
    System.out.println("Suma= " + num1+num2);
}
```

A les següents url podem trobar informació sobre el funcionament de JavaDoc.

http://www.oracle.com/technetwork/articles/java/index-137868.html

http://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html

https://en.wikipedia.org/wiki/Javadoc



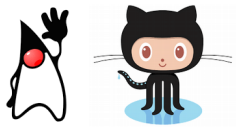
3.4.2 PHPDocumentor

PHPDocumentor ens permet generar automàticament documentació del nostre codi d'una forma molt pareguda a com ho fa JavaDoc. Mitjançant comentaris i etiquetes especials podem definir que fa cada classe, mètode i funció del nostre codi. https://phpdoc.org/

Ens permet generar la documentació des de línia de comandes, des d'una interfície web o des del codi. Els formats de sortida poden ser HTML, PDF o XML

Exemples d'etiquetes:

Tabla de etiquetas	Tipo
@abstract	Especifica que una clase, un método o una variable son abstractos.
@access public|private|protected	Controla la aparición de un elemento en la documentación generada. Si se declara como private, el elemento no aparece a no ser que se genere la documentación de forma específica.
@author	Autor del elemento con la posibilidad de incluir un elemento dentro de <>.
@category nombre	Especifica la categoría a la cual pertenece la entidad documentada dentro del paquete.
@deprecated	Indica que el elemento esta discontinuado. Determina que el elemento es obsoleto y no debería utilizarse ya que puede ser retirado en el futuro cercano.
@example	Se utiliza para incluir una ejemplo.
@final	Determina que el elemento es final y por lo tanto no puede ser sobrescrito o crear subclases de él.
@ignore	Se ignora al elemento.
@internal información_no_publicable	Especifica información que no se incluye en la documentación pública.
@link URL [texto]	Despliega un hiperenlace en la documentación.
@see	Enlaza con la documentación de otro elemento.
@package nombre	Especifica el paquete que agrupa de manera documental las clases y funciones especificadas.
@param tipo1|tipo2|... $nombre_de_variable descripción	Indica el tipo y el nombre del parámetro.
@static	La variable o la función es estática.
@since información de versión	Documenta cuando/en que versión se agregó el elemento.
@return tipo_de_datos	Describe el tipo de retorno de un método o función.
@todo informacion	Documenta cambios que serán realizados en el futuro.
@throws información_de_excepciones	Especifica las posibles excepciones lanzadas por la función o método actual.
@version	Versión del elemento.



El procés de documentació comença amb l'element bàsic de phpDocumentor: un bloc de documentació o DocBlock. Un DockBlock conté tres segments bàsics: descripció curta, descripció llarga i etiquetes.

```
<php
public function isLoggedIn();
  /**
 * Devuelve la información del usuario sobre la cuenta
 *
 * This method is used to retrieve the account corresponding
 * to a given login. <b>Note:</b> it is not required that
 * the user be currently logged in.
 *
 * @access public
 * @param string $user user name of the account
 * @return Account
 */
  public function getAccount($user = '');
}
?>
```

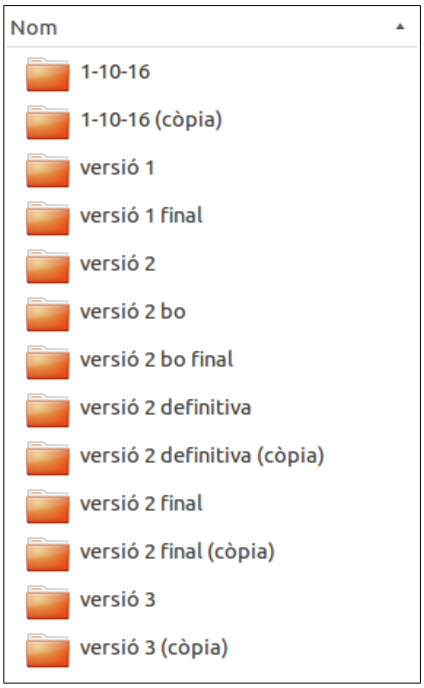
Exemples i ús:

http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/255

https://phpdoc.org/docs/latest/guides/docblocks.html

4. Sistemes de control de versions (SCV)

Quant es treballa amb projectes de programari, el més comú és anar generant codi i al mateix temps fer proves de les funcionalitats que anem desenvolupant, igual que anem modificant el codi font contínuament, afegim comentaris, creem noves funcionalitats i mètodes, provem altres tipus d'algorismes o preparem el codi per fer un accés a una base de dades, creem nous fitxers, etc.



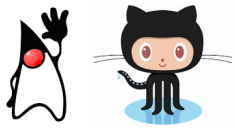
Si tenim intenció de fer un canvi important al nostre projecte, el que podem fer és crear una còpia del projecte en l'estat actual a un altre directori de treball i fer les proves al nou directori. Normalment es controla la versió amb el nom del directori que és la data. D'aquesta manera continuem tenint una versió estable i funcional, i totes les modificacions es fan a una nova versió on podem fer les proves que necessitem. Un cop finalitzem la nova funcionalitat, tindrem una còpia del projecte amb una nova versió estable i funcional però a un directori diferent.

Aquesta metodologia de treball genera uns quants problemes. El primer és que hem d'anar documentat quin directori de treball conté cada versió i quines funcionalitats implementa cada versió. El següent problema que presenta és el volum d'informació que generem o copiem. Realment cada nova versió requereix d'una còpia completa de tot el projecte, on realment estem implementant una petita funcionalitat del global del projecte i en realitat estem copiant una quantitat de informació considerable (mesurada en MB).

Una millora a la gestió de la informació que anem generant al nostre projecte, passa per la utilització d'un sistema de control de versions. Amb aquest tipus d'eines, podem anar guardant les diferents versions a un mateix lloc o directori, inclús podem crear versions per implementar noves funcionalitats sense alterar les funcionalitats confirmades com estables i funcionals.

Entre els avantatges d'utilitzar un sistema de controls de versions podem citar les següents:

- Organitzar les diferents versions de productes
- Recuperar versions antigues i crear-ne de noves a partir de versions existents
- Organitzar desenvolupaments específics dels productes per a determinats clients
- Coordinar diverses persones i distribuir tasques
- Permet escalabilitat, reusabilitat, integritat i productivitat



4.1 Què és un sistema de control de versions.

Un sistema de control de versions, és una aplicació que permet gestionar un rebost d'arxius i les seves diferents versions. Entenem per versions, els canvis realitzats a un arxiu o un conjunt d'arxius durant el temps.

Utilitzen una arquitectura client-servidor on el servidor emmagatzema la versió actual del projecte, i totes les anteriors a mode d'històric.

No només serveix per emmagatzemar diferents versions de codi font del llenguatge de programació que estem utilitzant, sinó que també ens serveix per emmagatzemar qualsevol tipus de fitxer com poden ser imatges, fitxers binaris, fitxers de configuració, etc.

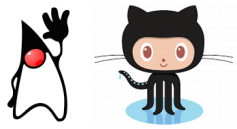
4.2 Conceptes sobre sistemes de control de versions

Amb un sistema de control de versions, realment el que tenim és un directori controlat per una eina on es van guardant els fitxers del nostre projecte i totes les seves versions. Utilitzant aquesta eina, podem extraure una còpia del projecte a un directori de treball, podem fer tots els canvis que necessitem i quant aquests funcionin, podem dir-li al sistema de control de versions que ens dese la nova versió. Normalment el sistema de control de versions ens sol demanar que introduïm un comentari cada cop que volem desar fitxers nous o modificats.

Una altra utilitat important d'aquest tipus d'eines, és que podem obtenir fàcilment qualsevol de les versions anteriors del nostre projecte, veure els comentaris que vam introduir, o inclús comparar distintes versions del mateix fitxers, comprovant les línies que es van modificar.

Per poder comprendre el funcionament d'un sistema de control de versions, hem de conèixer primer una sèrie de conceptes relacionats amb aquests:

- Revisió: és una visió estàtica en el temps de l'estat d'un grup de fitxers i directoris. Té una etiqueta que l'identifica. Sol tindre metadades associades (dades sobre dades o informació sobre dades) com poden ser:
  - Identitat de qui ha fet les modificacions
  - Data i hora en la qual es van emmagatzemar els canvis
  - Raó dels canvis
  - De quina revisió o rama deriva la revisió
  - Paraules clau associades a la revisió



- Còpia de treball: és el conjunt de directoris i arxius controlats pel sistema de control de versions, i que es troba en edició activa. Està associat a una rama de treball concreta.
- Rama de treball: rama o conjunt ordenat de revisions. La revisió principal s'anomena principal o cap (head). Les rames es poden separar i ajuntar segons sigui necessari, formant un grafo de revisions.
- Rebost: lloc on s'emmagatzemen les revisions. Físicament pot ser un arxiu, col·lecció d'arxius, una base de dades, etc. Pot estar emmagatzemat en local o en un servidor remot.
- Conflicte: situació que es dona quant diverses persones fan canvis contradictoris a un mateix fitxer o document. Els sistemes de control de versions tan sols avisen de l'existència del conflicte, no el solucionen. El procés de solució d'un conflicte s'anomena resolució.
- Canvi: modificació d'un arxiu controlat per un sistema de control de versions. Quant es junten diversos canvis, generem una revisió unificada, es diu que s'ha fet una combinació o una integració.
- Pedaç: llista de canvis generada al comparar revisions, i que pot utilitzar-se per reproduir automàticament les modificacions fetes al codi.

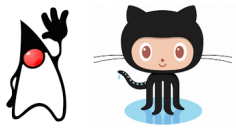
Amb l'utilització de sistemes de control de versions, aconseguim mantindre un rebost amb la informació actualitzada. La forma habitual de treballar consisteix a mantenir una còpia en local i modificar-la. Després actualitzem el rebost. D'aquesta manera no és necessari l'accés continu al rebost.

El procediment d'ús habitual d'un sistema de control de versions segueix el cicle d'operacions següents:

1. Descàrrega inicial dels fitxers (checkout, tan sols es realitza el primer cop que s'utilitzen els fitxers).
2. Modificació de fitxers, aplicant els canvis convenients.
3. Actualització dels fitxers locals (update).
4. Resolució de conflicte (si hi ha, el sistema de control de versions avisa i els usuaris han de resoldre).
5. Actualització dels fitxers al rebost (commit)

Les funcions que ha de poder realitzar un sistema de control de versions són:

- Diversos clients poden obtenir còpies del projecte al mateix temps
- Realitzar canvis als fitxers mantenint un històric de canvis
- Els clients poden comparar diferents versions d'arxius



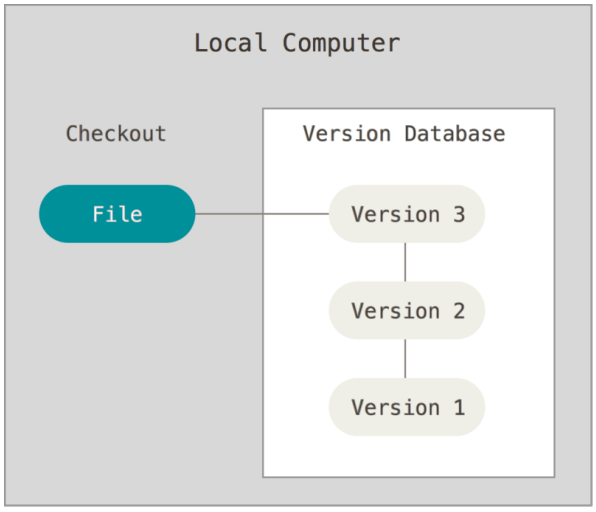
- Unir canvis realitzats per diferents usuaris sobre els mateixos fitxers
- Obtenir una «foto» històrica del projecte tal com es troba en un moment determinat
- Actualitzar una còpia local amb l'última versió que es troba al servidor
- Mantenir distintes rames d'un projecte

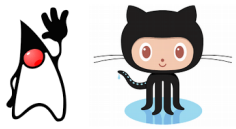
4.3 Tipus de sistemes de control de versions

Encara que els sistemes de control de versions són eines imprescindibles en projectes de certa envergadura i amb diversos desenvolupadors, on el treball en grup permet desenvolupar un projecte i mantenir un únic rebost comú per a tot el grup, també pot ser interessant per a un únic desenvolupador utilitzar aquest tipus d'eines, ja què encara que sigui ell sol qui desenvolupa el projecte, sempre podrà disposar de totes les versions del programa desenvolupat i així tenir controlades les seves versions.

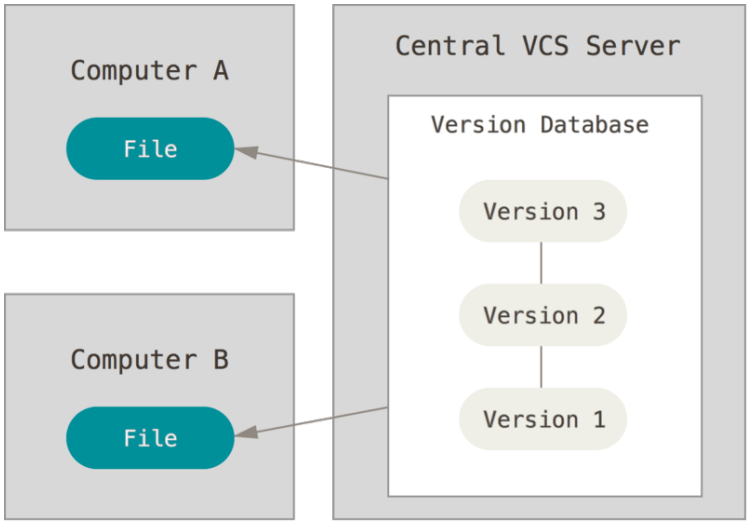
Existeixen diverses maneres de treballar amb sistemes de control de versions, bé siga per la tipologia del projecte o pel tipus de programari a utilitzar. Aquests modes de treball són local, centralitzat o distribuït.

Local: el desenvolupador treballa amb un rebost a la seva màquina local. Aquest rebost pot ser una base de dades, o es pot utilitzar un SCV de qualsevol tipus (centralitzat o distribuït). Realment en aquest model el que li interessa al desenvolupador es tindre un rebost amb les versions que va desenvolupant i només ell tindrà accés. Un exemple d'aplicació es RCS, aquest manté pedaços o patch amb les diferències de les diferents versions. Si volem reconstruir una versió, hem d'agafar la primera versió i ajuntar tots els pedaços generats.

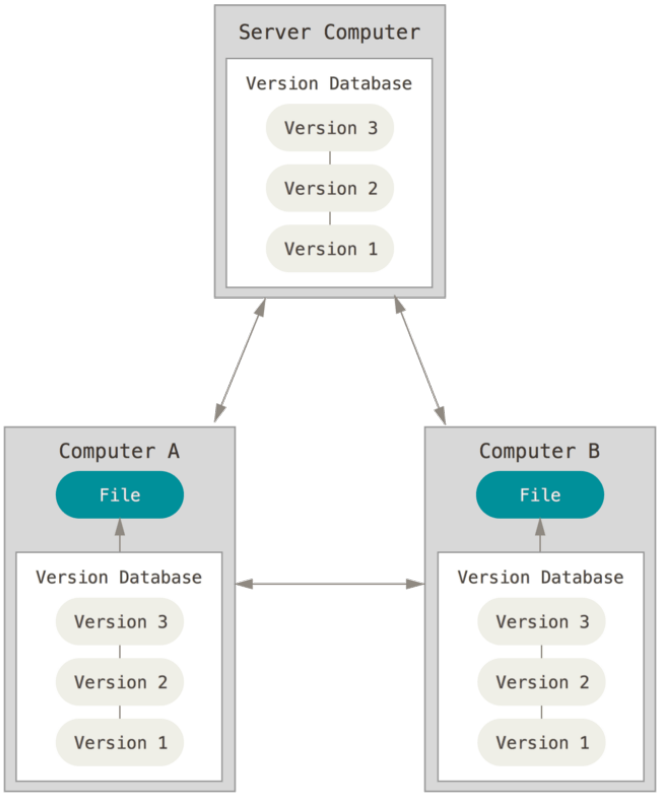




Centralitzat: aquest model és necessari quant un grup de desenvolupadors necessiten accedir a un mateix rebost de dades per dur a terme un projecte. Es tracta de tindre un únic servidor que conté tots els arxius del projecte i les seves respectives versions. Diferents clients descarreguen arxius des del servidor central. Les avantatges que presenta aquest model és que cada desenvolupador estarà treballant a una part del projecte, l'administrador té més control per administrar el projecte i les seves versions. Els inconvenients que presenta el model és que en estar tot centralitzat a un únic servidor, qualsevol fallada de sistema deixarà els desenvolupadors sense poder accedir al projecte, s'ha de dur a terme un control de còpies de seguretat adequat. Exemples de sistemes centralitzats són SVC, Subversion, Perforce, Tortoise i RabbitVCS.



Distribuït: aquest model es basa en la rèplica del projecte a cadascun dels clients o desenvolupadors. D'aquesta manera cada desenvolupador té una còpia del projecte al seu directori de treball. Si el servidor on tenim el projecte cau per una fallada, podem recuperar el projecte amb les còpies que tenen els clients, d'aquesta manera el procés de restauració és molt més senzill. Aquest tipus de sistemes poden treballar bé amb diversos rebosts, de manera que podem dividir un projecte en diversos rebosts i col·laborar diversos grups de persones en cadascun dels rebosts diferents, establint diferents fluxos de treball paral·lels. Exemples de sistemes distribuïts són Git, Mercurial, Bazaar o Darcs.



Model	Avantatges	Inconvenients
Local	• Control a nivell local	• No es pot compartir • Restaurar implica agafar la 1ª versió i tots els pedaços necessaris
Centralitzat	• Informació a un servidor central • Possibilitat de compartir el projecte i les versions • Més control per l'administrador	• Si falla el servidor no es pot accedir al projecte • El projecte s'ha de restaurar de les còpies o backup • Depèn de la velocitat de la xarxa
Distribuït	• Replica el projecte als clients • Si cau el servidor els clients poden continuar treballant	• Presenta bloquejos i conflictes amb fitxers • Major corba d'aprenentatge (més instruccions)

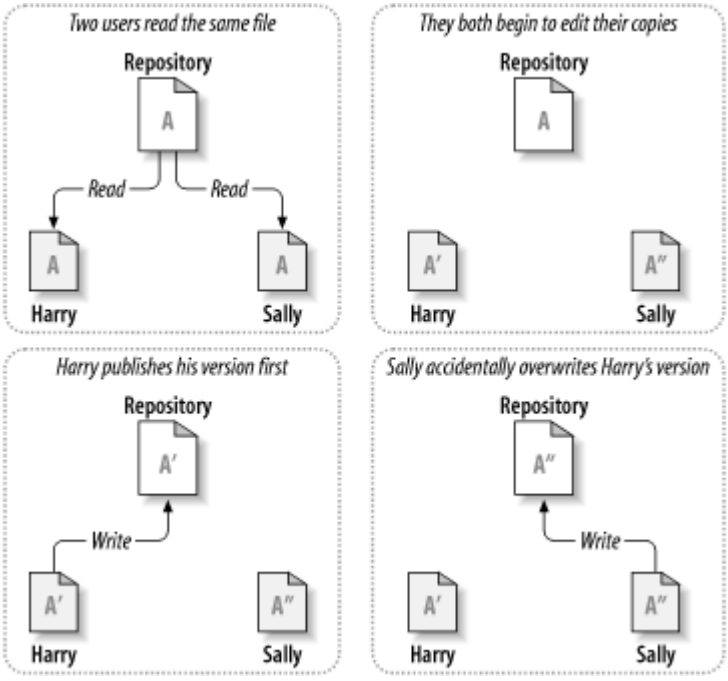
4.4 El problema de compartir fitxers

Un sistema de control de versions ens dona l'avantatge de que podem compartir, mitjançant una xarxa de computador, el nostre projecte i les versions respectives. Aquí es planteja un problema, i és com podem evitar que un usuari reemplaci codi d'un altre desenvolupador quant els dos estan treballant al mateix temps amb el mateix fitxer?

Conflicte: apareix quant dos o més persones treballen sobre la mateixa versió d'un fitxer.

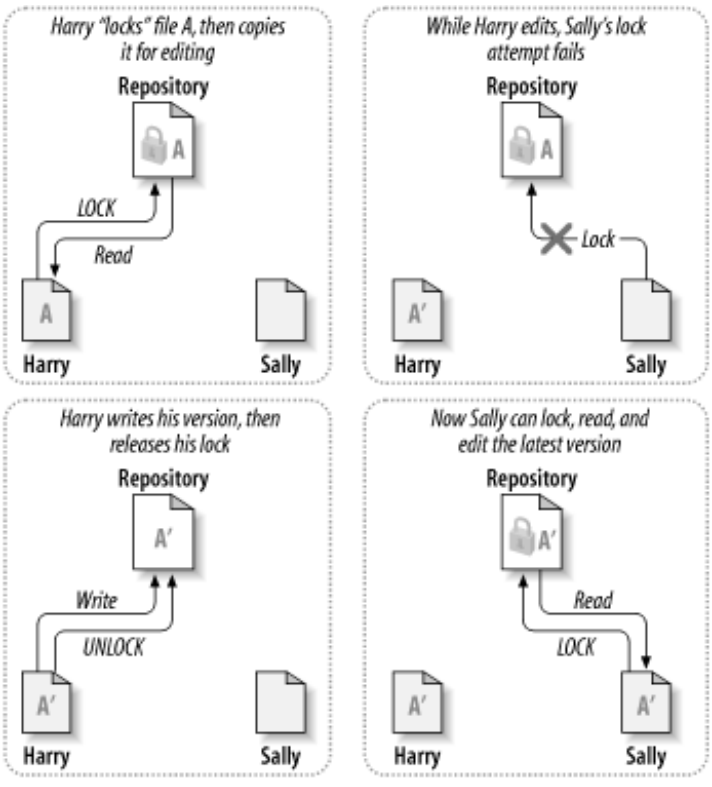
El procés que s'ha de reproduir perquè aparega el conflicte són el següent:

1. Dos usuaris (Harry i Sally) obtenen el mateix fitxer (A).
2. Harry canvia el fitxer i el puja al repositori (A').
3. Sally no actualitza el fitxer després que Harry hagi pujat el fitxer.
4. Sally realitza canvia al mateix fitxer que l'usuari A''.
5. Sally intenta posteriorment pujar el fitxer A'' al repositori.



Solució bloquejar-modificar-desbloquejar

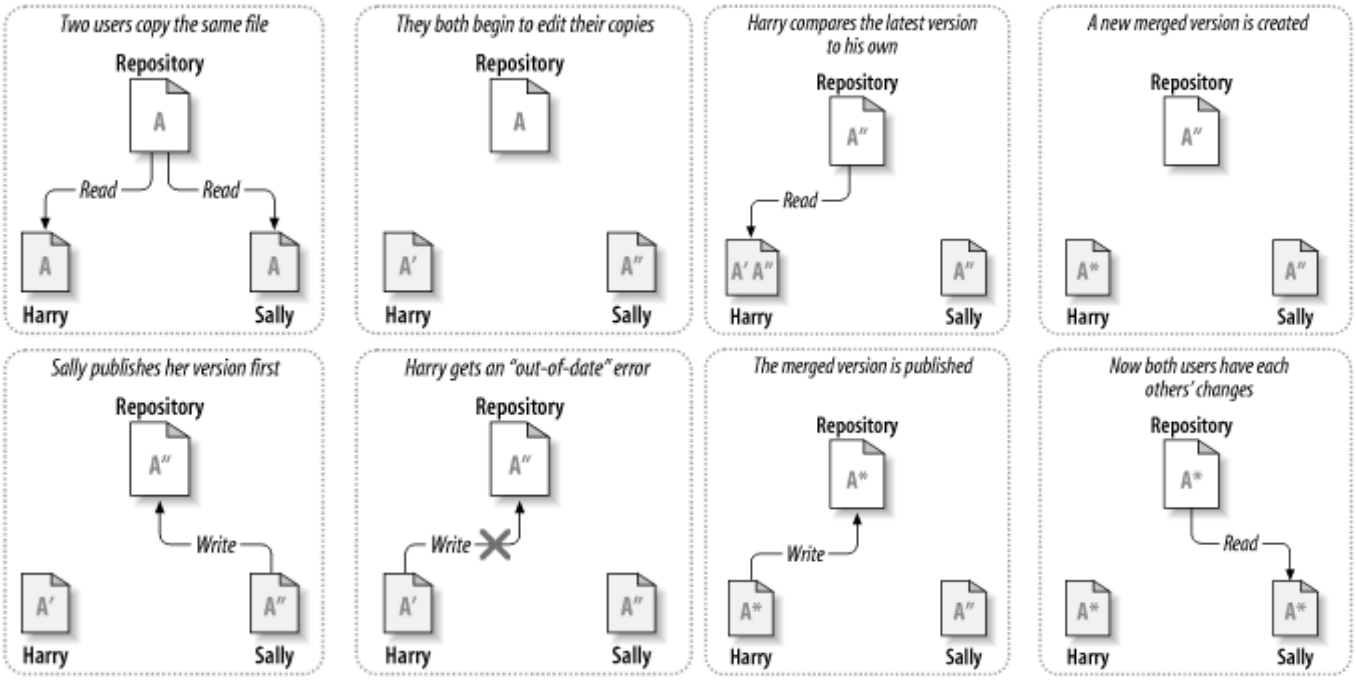
Una solució pot ser la que es presenta a continuació. Es tracta de bloquejar el fitxer amb el qual es va a treballar, de manera que mentre es treballa amb aquest fitxer, cap altre usuari podrà modificar-lo. En acabar de treballar amb el fitxer, hem de desbloquejar el fitxer per deixar-lo lliure i que la resta d'usuaris pugui treballar amb ell.



És una solució molt restrictiva, ja que no permet modificar diferents parts del fitxer a diversos usuaris.

Pot quedar un fitxer bloquejat de forma indefinida.

Solució copiar-modificar-fusionar



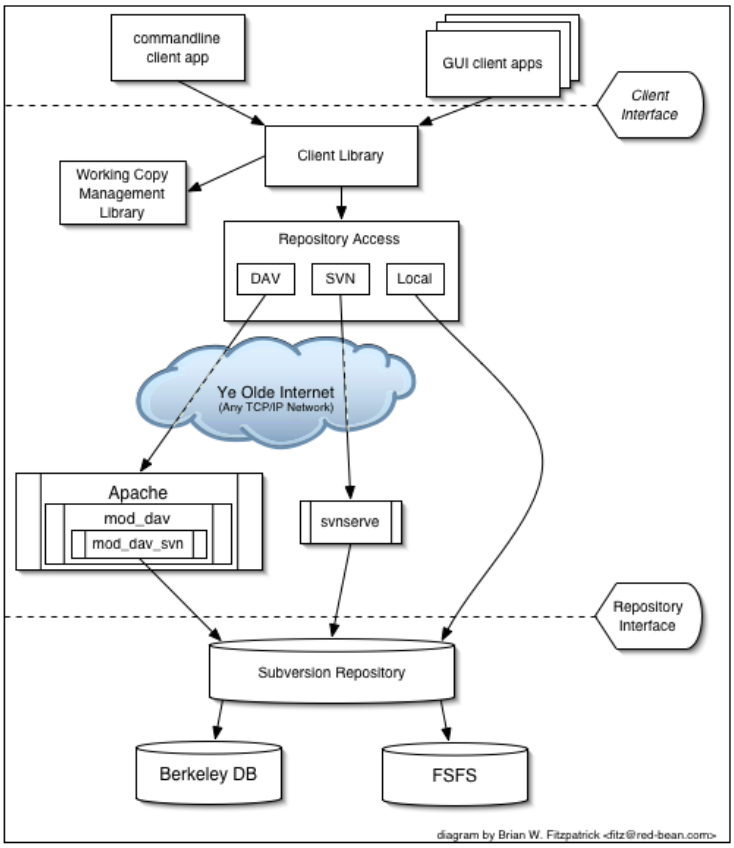
Aquesta solució, és la solució més implementada pel programari actual.

4.5 Subversion

Subversion és un sistema de codi obert/lliure de control de versions (VCS). És a dir, Subversion fa de servidor fitxers i directoris, i controla els canvis fet a ells, en el temps. Això li permet recuperar versions antigues de les seves dades o examinar la història de com van canviar les seves dades.

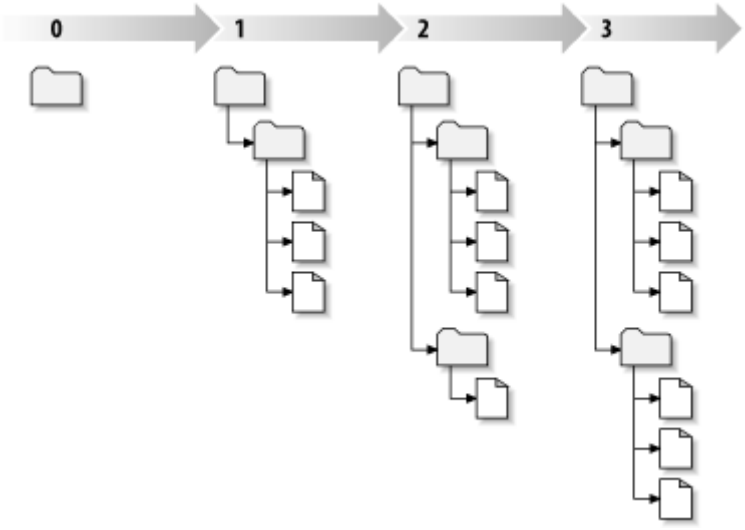
Subversion pot operar a través de les xarxes, el que permet que sigui utilitzat per persones en diferents equips. A cert nivell, la capacitat perquè diverses persones puguen modificar i gestionar el mateix conjunt de dades de les seves respectives ubicacions fomenta la col·laboració. A més el progrés del desenvolupament del projecte ocorre més ràpidament augmentat així la qualitat del mateix. En cas de fer un canvi incorrecte de les dades, simplement podem desfer aquest canvi.

A la següent imatge podem veure els components que utilitza Subversion:



Funcionament de Subversion

Bàsicament Subversion treballa amb repositoris o magatzems de dades, dintre d'aquest utilitza un directori de treball o working directory. Per controlar els diversos canvis que anem fent als nostres fitxers, utilitza revisions. Cada vegada que el repositori accepta un commit, crea un nou estat de l'arbre d'arxius, anomenat revisió. A cada revisió és assignada un nombre natural únic, un més gran que el nombre assignat a la revisió anterior. La revisió inicial de recent creació al repositori es numera 0 i consisteix únicament en un directori arrel buit. Cada revisió és una còpia de tot el projecte actual.



Per a cada fitxer al working copy, Subversion emmagatzema dues peces essencials d'informació :

- Quina versió és el fitxer de treball (això es denomina la revisió de treball de l'arxiu)
- Una marca de temps que ens informa quant es va actualitzar per últim cop la nostra còpia local

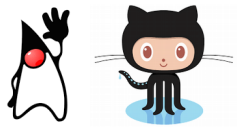
Mitjançant aquesta comunicació amb el servidor, Subversion pot dir quin dels següents quatre estats pot tenir un arxiu:

Sense canvis i actualitzat: l'arxiu no s'ha modificat al directori de treball, i no hi ha canvis al repositori. Un svn commit d'aquest fitxer no farà res, i un svn update del fitxer no farà res.

Localment canviat i actualitzat: l'arxiu ha estat canviat al directori de treball, i no hi ha canvis en aquest arxiu al repositori des de l'última actualització. Hi ha canvis locals que no s'han confirmat en el repositori, així un svn commit del fitxer serà publicar els seus canvis, i un svn update del fitxer no farà res.

Sense canvis i no actualitzat: l'arxiu no ha estat canviat al directori de treball, però ha estat canviat al repositori. L'arxiu ha de ser actualitzat per tal d'actualitzar-lo amb l'última revisió pública. Un svn commit d'aquest fitxer no farà res , i un svn update del fitxer introduirà els canvis a la seva còpia de treball.

Localment canviat i no actualitzat: l'arxiu s'ha canviat tant al directori de treball com al repositori. Un svn commit del fitxer fallarà amb un error "out-of-date". L'arxiu ha de ser actualitzat en primer lloc, una ordre d'actualització de SVN s'intentarà fusionar els canvis públics amb els canvis locals. Si Subversion no pot completar la fusió d'una manera automàtica, deixarà a l'usuari per resoldre el conflicte.



	Directori de treball	Repository	Efecte del Commit	Efecte d'Update
Sense canvis i actualitzat	No canviat	No canviat	Sense efecte	Sense efecte
Canviat localment i actualitzat	Canviat	No canviat	Actualitza Repo	Sense efecte
Sense canvis i des-actualitzat	No canviat	Canviat	Sense efecte	Actualitza directori
Canviat localment i des-actualitzat	Canviat	Canviat	Error per desactualitzat	Intentar fer un merge

4.6 GIT

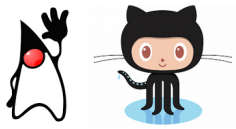
Git és un sistema de control de versions ràpid, està escrit en C i s'ha fet popular sobre tot arran de ser el triat per al nucli de Linux.

Des del seu naixement al 2005, Git ha evolucionat i madurat per ser fàcil d'usar i tot i així, conservar aquestes qualitats inicials. És tremendament ràpid, molt eficient amb grans projectes, i té un increïble sistema de ramificació (branching) per a desenvolupament no lineal.

La principal diferència entre Git i qualsevol altre VCS (Subversion i companyia inclosos) és com Git modela les seves dades. Conceptualment, la majoria dels altres sistemes emmagatzemen la informació com una llista de canvis en els arxius. Aquests sistemes (CVS, Subversion, Perforce, Bazaar, etc.) modelen la informació que emmagatzemen com un conjunt d'arxius i les modificacions fetes sobre cada un d'ells al llarg del temps.

Git no modela ni emmagatzema les seves dades d'aquesta manera, modela les seves dades més com un conjunt d'instantànies d'un mini sistema d'arxius. Cada vegada que confirmes un canvi, o guardes el estat del teu projecte a Git, ell bàsicament fa una "foto" del aspecte de tots els teus arxius en aquest moment, i guarda una referència a aquesta instantània. Per ser eficient, si els arxius no s'han modificat, Git no emmagatzema l'arxiu de nou, només un enllaç a l'arxiu anterior idèntic que ja té emmagatzemat.

Gairebé qualsevol operació és local, la majoria de les operacions en Git només necessiten arxius i recursos locals per operar, per exemple, per navegar per la història del projecte, no es necessita sortir al servidor per obtenir la història i mostrar-la, simplement es llegeix directament de la base de dades local. Això vol dir que es veu la història del projecte gairebé a l'instant. Si cal veure els canvis introduïts entre la versió actual d'un arxiu i aquest arxiu fa un mes, Git pot buscar l'arxiu fa un mes i

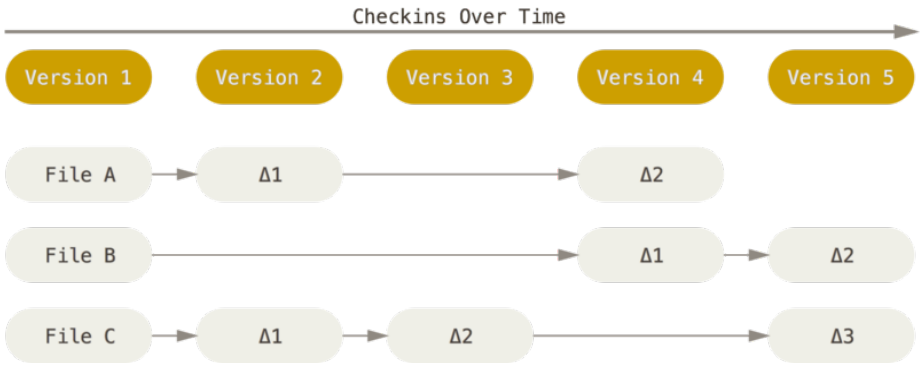


fer un càlcul de diferències localment, en lloc d'haver de demanar-li a un servidor remot que ho faci, o obtenir una versió antiga de l'arxiu del servidor remot i fer-ho de manera local.

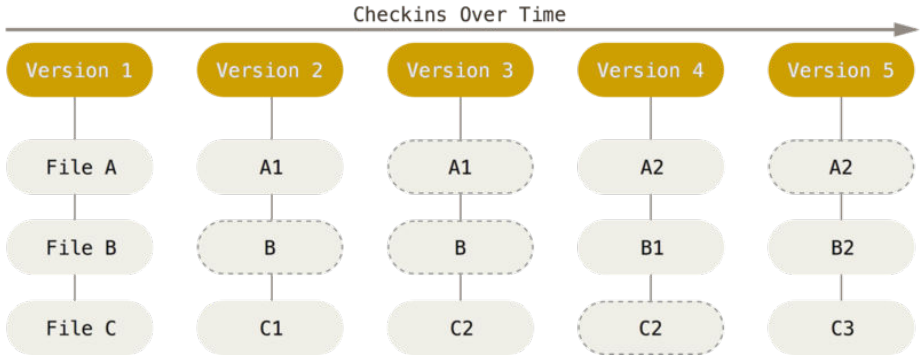
A part de tot això, Git posseeix integritat ja què tot és verificat mitjançant una suma de comprovació abans de ser emmagatzemat, i és identificat a partir d'aquest moment mitjançant aquesta suma. Això significa que és impossible canviar els continguts de qualsevol arxiu o directori sense que Git ho detecti. Com a conseqüència d'això és impossible perdre informació durant la seva transmissió o patir corrupció d'arxius sense que Git sigui capaç de detectar-lo.

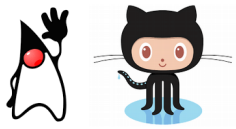
Emmagatzemat d'informació

La principal diferència entre Git i qualsevol altre VCS (incloent Subversion i seus amics) és la forma en la que gestiona les seves dades. Conceptualment, la majoria dels altres sistemes emmagatzemen la informació com una llista de canvis en els arxius. Aquests sistemes (CVS, Subversion, Perforce, Bazaar, etc.) gestionen la informació que emmagatzemen com un conjunt d'arxius i les modificacions fetes a cadascun d'ells a través del temps.



Git no fa servir ni emmagatzema les seves dades d'aquesta forma. Git gestiona les seves dades com un conjunt de còpies instantànies d'un sistema d'arxius miniatura. Cada vegada que confirmes un canvi, o guardes l'estat del teu projecte a Git, ell bàsicament pren una foto de l'aspecte de tots els teus arxius en aquest moment, i guarda una referència a aquesta còpia instantània. Per ser eficient, si els arxius no s'han modificat Git no emmagatzema l'arxiu de nou , sinó un enllaç a l'arxiu anterior idèntic que ja té emmagatzemat. Git gestiona les seves dades com una seqüència de còpies instantànies .





Funcionament de Git

Git té tres estats principals en què es poden trobar els arxius: confirmat (Committed), modificat (modified) i preparat (staged).

- **Confirmat**: significa que les dades estan emmagatzemades de manera segura a la base de dades local.
- **Modificat**: estat en què s'ha modificat l'arxiu però encara no s'ha confirmat a la base de dades.
- **Preparat**: vol dir que s'ha marcat un arxiu modificat en la seva versió actual perquè vagi en la propera confirmació.

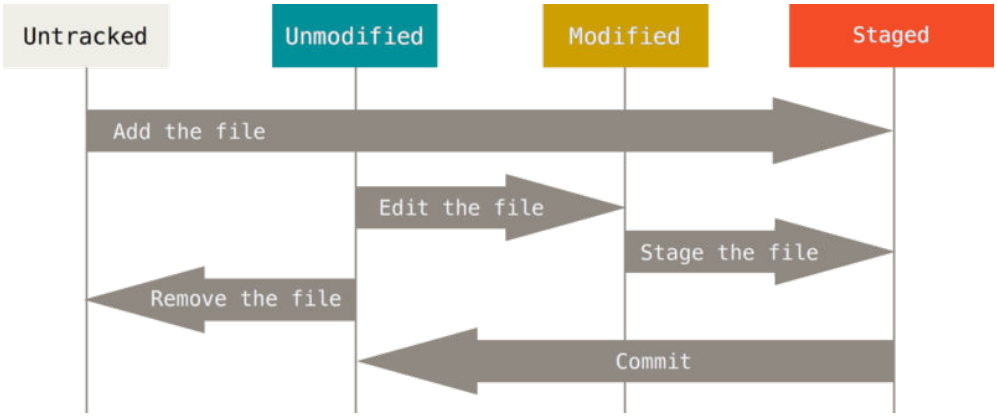
Això ens porta a les tres seccions principals d'un projecte de Git:

- El directori de Git (**Git directory**): Emmagatzema les metadades i la base de dades d'objectes per teu projecte. És la part més important de Git, i és el que es copia quan es clona un repositori des d'un altre ordinador.
- El directori de treball (**working directory)**: És una còpia d'una versió del projecte. Aquests arxius es treuen de la base de dades comprimida en el directori de Git, i es col·loquen en disc perquè es pugui utilitzar o modificar.
- L'àrea de preparació (**staging area**): és un senzill arxiu, generalment contingut en el teu directori de Git, que emmagatzema informació sobre el que va a anar a la propera confirmació. De vegades es denomina índex, però s'està convertint en estàndard el referir-s'hi com l'àrea de preparació.

El flux de treball bàsic en Git consisteix en:

1. Modificar una sèrie d'arxius en el directori de treball.
2. Preparar els arxius, afegint instantànies d'ells a l'àrea de preparació.
3. Confirmar els canvis, pren els arxius tal com estan en l'àrea de preparació, i emmagatzema aquesta instantània de manera permanent en el directori de Git.

Si una versió concreta d'un arxiu està en el directori de Git, es considera confirmada (Committed). Si ha sofert canvis des que es va obtenir del repositori, però ha estat afegida a l'àrea de preparació, està preparada (staged). I si ha sofert canvis des que es va obtenir del repositori, però no s'ha preparat, està modificada (modified).



Seguretat de Git

Git s'enquadra en la categoria dels sistemes de gestió de codi font distribuïda. Amb Git, cada directori de treball local és un repositori complet no dependent, d'accés a un servidor o a la xarxa.

L'accés a cada repositori es realitza a través del protocol de Git, muntat sobre ssh, o utilitzant HTTP, encara que no cal cap servidor web per poder publicar el repositori a la xarxa.

Perquè un repositori públic pugui ser utilitzat d'aquesta manera, cal permetre l'execució de la comanda push. En primer lloc haurem de crear el repositori públic i habilitar algun dels següents accessos:

- Accés directe sistemàticament de fitxers amb permisos d'escriptura per a l'usuari sobre el directori del repositori.
- Accés remot via SSH (consulteu man git-shell) i permisos d'escriptura per a l'usuari sobre el directori del repositori.
- Accés HTTP amb WebDav degudament configurat.
- Accés mitjançant protocol Git amb el servei "receive-pack" activant al dimoni git.

Instal·lació de git al client

sudo apt-get install git

git config --global user.name "el meu nom"

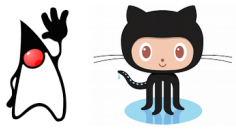
git config --global user.email el_meu_nom@example.com

Exemples d'ús bàsic de Git

Com podem crear un repositori? git init

Com afegim elements al repositori? git add fitxer

Com ens baixem una còpia d'un projecte que està controlat per git? git clone url



Com ens baixem una còpia d'un projecte a un directori en concret? git clone url dir

Com podem saber l'estat dels fitxers del projecte? git status

com podem veure els canvis de fitxers no preparats? git diff

com podem veure els canvis dels fitxers preparats? git diff –staged

Com podem confirmar els canvis del fitxers que tenim preparats afegint un comentari? git commit -m «comentari»

com podem preparar i confirmar fitxers per pujar-los al repositori? git commint -a -m «missatge»

com podem eliminar arxius? git rm fitxer

Com podem renombrar un fitxer? git mv fitxer.orige fitxer.desti

Com podem visualitzaar el historial de confirmacions? git log

Com podem visualitzar els repositoris remots amb els quals treballem? git remote -v

Com podem afegir un repositori remot? git remote add nom url (nom podem referencial-lo més tard)

Com ens baixem el repositori remot sense fusionar amb la còpia local? git fetch nom_repo_desat

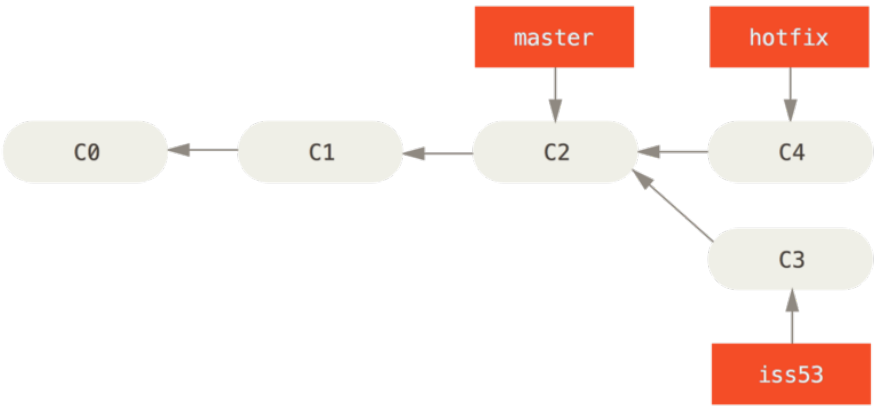
Com ens baixem el repositori remot fusionant-lo amb la còpia local? git pull

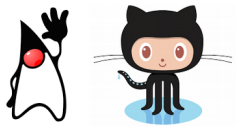
Com podem enviar tots els comits que hem fet a una branca local a un repositori remot? git push url nom_branca

Quines comandes són necessàries per poder crear ramificacions al projecte i poder moure'ns per elles. git branch nom_rama, git checkout nom_rama, git commit -a -m 'comentari'.

git checkout -b és el mateix que fer un branch i després un checkout.

Donada la següent imatge:





Com hem de fer per fusionar la branca hotfix amb la master?

git checkout hotfix

vim index.html (editem el fitxer per modificar)

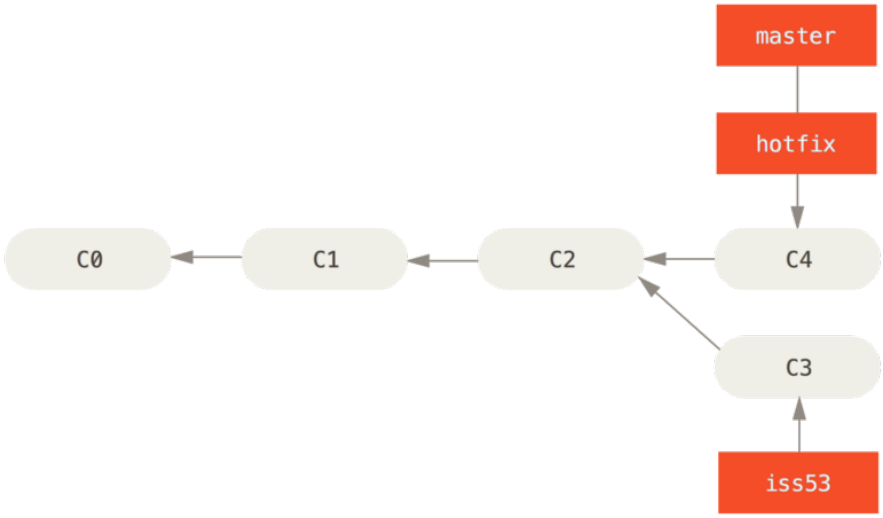
git commit -a -m 'fixed the broken email address'

git checkout master

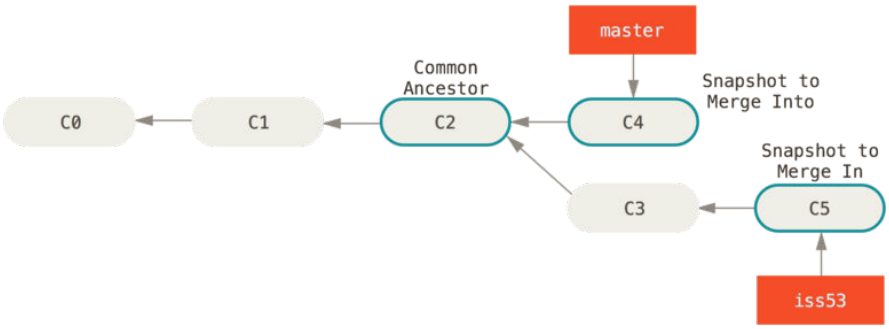
git merge hotfix

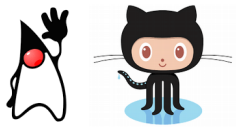
git branch -d hotfix

Resultat: (hotfix no ha d'aparèixer, es deixa per ser més il·lustratiu)



Continuem treballant amb la branca iss53 i finalment volem fusionar-ho amb la branca principal. Tindríem una imatge semblant a la següent:



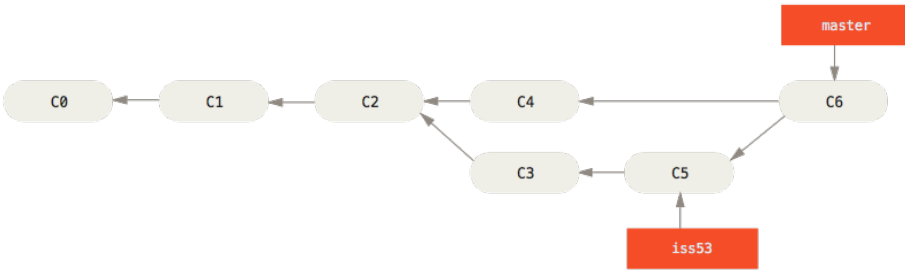


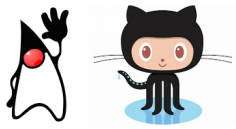
Com hem de fer per fusionar el resultat amb la branca iss53?

git checkout master

git merge iss53

Resultat:





5. Bibliografia i Webgrafia

Gestió de projectes

Projectes Web. Alberto Otero García. UOC febrer 2006.

Cicle de vida del programari

http://es.ccm.net/contents/223-ciclo-de-vida-del-software

https://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software

http://www.hanantek.com/es/modelos-ciclo-vida-software

https://en.wikipedia.org/wiki/Software_release_life_cycle

Convencions de codificació

http://www.oracle.com/technetwork/java/codeconvtoc-136057.html

http://www.oracle.com/technetwork/java/codeconventions-150003.pdf

Eines de documentació automàtica

https://en.wikipedia.org/wiki/Comparison_of_documentation_generators

http://www.oracle.com/technetwork/articles/java/index-137868.html

http://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html

https://en.wikipedia.org/wiki/Javadoc

https://phpdoc.org/

http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/255

https://phpdoc.org/docs/latest/guides/docblocks.html

Sistemes de control de versions

http://chernando.eu/doc/svn/

https://tortoisesvn.net/docs/nightly/TortoiseSVN_es/index.html

https://git-scm.com/book/es/v2

https://git-scm.com/download/gui/linux

https://about.gitlab.com/documentation/