Desenvolupament d'una eina per aplicar BDD en els productes de GTECH

Jordi Pons Llauradó

3/25/2014

Ponent: Ernest Teniente

Departament: Enginyeria de serveis i sistemes d'informació

Director: Jordi Falguera

Empresa: GTECH S.p.A

Titulació: ENGINYERIA INFORMÀTICA

Centre: Facultat d'Informàtica de Barcelona (FIB)

Universitat: Universitat Politècnica de Catalunya(UPC) Barcelona Tech

DADES DEL PROJECTE

Títol del projecte:	Desenvolupament d'una eina per aplicar BDD en els productes de GTECH
Nom de l'alumne:	Jordi Pons Llauradó
Titulació:	Enginyeria en Informàtica
Crèdits:	37.5
Director/Ponent:	Ernest Teniente
Departament:	Enginyeria del Software
Empresa:	GTech

MEMBRES DEL TRIBUNAL (nom i signatura)

President:	Jordi Pradel Miquel
Vocal:	Montserrat Maureso Sánchez
Secretari:	Ernest Teniente Lopez

QUALIFICACIÓ

Qualificació numèrica:
Qualificació descriptiva:

Data:

AGRAIMENTS

Vull donar les gràcies als meus pares pel seu esforç i el seu recolzament, sense ells no hauria estat possible realitzar aquest Projecte de Final de Carrera, també per la seva paciència i comprensió durant tota la carrera. També vull agrair a la Mireia per tots els moments en que ha estat escoltant com es desenvolupava el projecte.

Un altre persona que m'agradaria donar les gràcies és al meu tutor, Ernest Teniente, per la seva orientació i els seus consells els quals m'han estat molt útils.

Per últim, vull agrair a totes les persones de l'oficina de GTECH per la seva col·laboració al llarg de tot el projecte, sobretot al meu director, Jordi Falguera, i a Viktor Farcic per la seva paciència, ajuda i visió de futur que han fet possible dur a terme aquest projecte .

ÍNDEX

1 -Prefaci 10

2 -Introducció 11

2.1 - L'empresa 11

2.1.1 - Metodologia de treball de l'empresa 12

2.2 - El projecte 14

2.3 - Paper realitzat en el projecte 16

2.4 – Metodologia Del projecte 17

2.4.1 - Integració Continua 17

2.4.2 – Test Driven Development (TDD) 18

2.4.3 – Behaviour Driven Development (BDD) 22

3 - Iteracions 30

3.1 - Iteració 1 31

3.1.1 – Anàlisis de requisits 31

3.1.2 Especificació 34

3.1.3 Disseny 43

3.2 - Iteració 2 57

3.2.1 – Anàlisis de requisits 57

3.2.2 Especificació 59

3.2.3 Disseny 65

3.3 - Iteració 3 72

3.3.1 – Anàlisis de requisits 72

3.3.2 Especificació 73

3.3.3 Disseny 76

3.4 - Iteració 4 80

3.4.1 – Anàlisis de requisits 80

3.4.2 Especificació 81

3.4.3 Disseny 83

3.5 - Iteració 5 87
3.5.1 – Anàlisis de requisits 87
3.5.2 Especificació 88
3.5.3 Disseny 90
4 - Implementació 94
4.1 Java 94
4.2 JavaFx 94
4.3 Selenium WebDriver 94
4.4 jBehave 94
4.5 HTML (HyperText Markup Language) 95
4.6 CSS (Cascading Style Sheets) 95
4.7 JavaScript 95
4.8 jQuery 95
4.9 Gradle 96
5 -Tests 97
5.1 Jacoco 97
5.2 JUnit 97
5.3 Mockito 98
6 – Planificacío i estimació econòmica 99
6.1 – Planificació 99
6.1 – estimació econòmica 100
7 - Conclusions 101
7.1 - Objectius assolits 101
7.2 - Futures funcionalitats possibles 102
7.3 - Conclusió personal 103
8 - Bibliografia 105
9 - Glossari 106
10 – Annex 107

10.1 Llista de steps 107
10.2.1 CommonSteps 107
10.2.2 WebSteps 107
10.2.3 DashboardSteps 112
10.2.4 OnceDashboardSteps 113
10.2.5 CommandPromptSteps 113
10.2.6 ShellSteps 115
10.2.7 AlturaSteps 116

1 -PREFACI

Aquest projecte s'ha realitzat dintre de la modalitat "b" (Empresa), concretament en una empresa que s'anomena GTech, empresa proveïdora de tecnologia a empreses de loteria.

La raó principal per la que vaig decidir dur a terme el Projecte de Final de Carrera (PFC) en la modalitat "b" i no en una altra de les diferents modalitats és que des d'un bon principi no volia acabar la carrera i obtenir el títol sense abans haver treballat en una empresa ja que hi ha coses que la universitat no pot arribar a ensenyar-te i que considero que abans d'entrar al mercat laboral com enginyer hauria d'aprendre i, al mateix temps, per acabar de consolidar i ampliar els coneixements obtinguts a la universitat durant tots aquest anys.

Un altre dels motius esta relacionat amb algunes de les últimes assignatures que he realitzat, en les que se'ns va parlar sobre les metodologies Agile a nivell teòric, i en una d'elles es van posar en pràctica, però per falta de temps no vam poder aprofundir tant com m'hauria agradat, és per això que aquest projecte em va semblar engrescador doncs em donava l'oportunitat d'ampliar els coneixements sobre Agile.

2 -INTRODUCCIÓ

2.1 - L'EMPRESA

Gtech és el proveïdor mundial líder de serveis de loteria amb un control de més del 60% en quan a loteria online de tot el món. Dins el marc espanyol s'encarrega de donar suport tecnològic a la ONCE (Organización Nacional de Ciegos Españoles). L'empresa ofereix el disseny del sistema, suport d'operacions, instal·lació i manteniment de terminals, formació especialitzada, suport de marketing, serveis de camp, gestió de línia directa, instal·lació de xarxes de telecomunicacions, gestió i manteniment, i una cartera de jocs molt diversificada per a cada canal de distribució.

Imatge 1: Logo empresa GTech

Un dels objectius principals de GTech és arribar a ser líders en el món com operadors i proveïdors de solucions tecnològiques per als mercats de jocs d'atzar regulats a través de l'orientació al client, productes innovadors i a la presentació de serveis superiors amb un compromís permanent amb els principis de joc responsable.

Tot i ser una agrupació de diferents empreses les quals es van unificar el 2012 en una sola, de cara al client sempre s'ha presentat com una sola empresa per tal de facilitar les comunicacions.

A través de la innovació continua, oferint un rendiment superior i mantenint un enfocament on el client és el centre de totes les activitats de l'empresa, GTech ha aconseguit uns alts nivells de satisfacció dels clients.

L'essència de l'èxit de l'empresa recau en la innovació i l'avantatge competitiu que aquesta innovació dota a l'empresa. Gràcies a la innovació pot respondre a les necessitats dels clients, crear noves oportunitats i definir el futur d'aquesta indústria.

Per poder oferir els millors serveis a els clients i poder ser líders en el sector, GTech disposa d'uns 8.600 empleats repartits en més de 110 països.

A finals del 2012 a tot Europa tenia 305.100 terminals (comptabilitzant terminals de loteria, self-service i de gaming), 425.228 a Amèrica, 15.470 a Africa i 58.700 a Asia i el Pacífic. A més a més dels terminals GTech també proporciona serveis de jocs de loteria governamental arreu del món, a finals del 2012 proporcionava a Europa 700 serveis (agrupant jocs de loteria, interactius i d'impressió com poden ser els "rasca rasca"), 1043 a Amèrica, 43 a Africa i 64 a Asia i el Pacífic. Es van enviar més de 350.000 màquines de "video gaming" a tot el món. Només als EUA es van atendre 152.000 minoristes, es van aconseguir 419 llicències de joc i 13 contractes de Gerents/Operadors privats.

Imatge 2: Mapa de països que el 2012 utilitzaven els serveis de GTech

2.1.1 - METODOLOGIA DE TREBALL DE L'EMPRESA

Avui en dia l'empresa segueix la metodologia Waterfall, metodologia que ordena les etapes del procés de desenvolupament de software de manera que al començament de cada etapa ha d'esperar a la finalització de l'anterior.

Les fases de la metodologia Waterfall són Especificació, seguit del Disseny, un cop s'ha realitzat el disseny es passa a la implementació i el testeig que es realitzen conjuntament en una mateixa fase, posteriorment es realitza el "Customer Acceptance Test (CAT)", un cop finalitzada la fase de CAT es realitza el "Go Live" on s'entrega el producte al client i finalment l'última etapa és la de suport i manteniment.

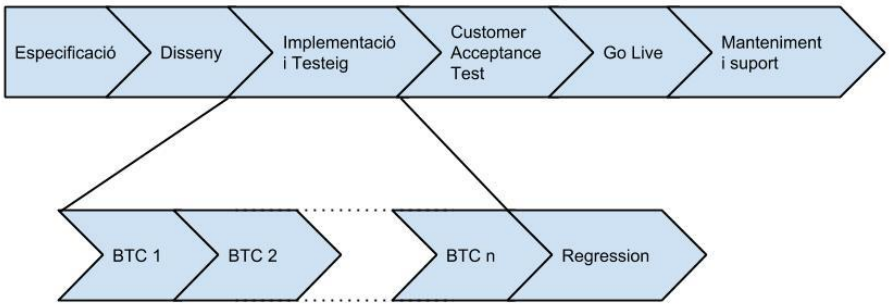


Diagrama 1: Procés de treball de GTech

ETAPA D'ESPECIFICACIÓ

A la fase d'especificació els Business Analysts parlen amb el client per arribar a un acord en les funcionalitats que es volen desenvolupar, en base a això es crea un SRS(Software Requirements Specification) on es descriuen les diferents funcionalitats.

Amb aquest SRS es crea un Prototype per ensenyar al client, el qual ha de validar tant l'aspecte com el comportament, d'aquesta manera si el client vol realitzar algun canvi el cost és molt menor.

ETAPA DE DISSENY

En aqueta etapa es dissenya el producte que el client vol en base al SRS que s'ha generat a l'etapa anterior.

ETAPA D'IMPLEMENTACIÓ I TESTING

El projecte es divideix en petites parts per anar comprovant que les funcionalitats es van fent segon els requisits previstos, aquestes parts se les anomena BTC (Build Test Correct). Al final de cada BTC es fa una "release" o entrega a QA (Quality Assurance) per tal que els enginyers de qualitat puguin testejar les funcionalitats ja entregades.

Un cop acabades les BTC's es fan dos tipus de test més que s'anomenen "System Test" on es prova com funciona el sistema en general incloses les noves funcionalitats (arrencar el dia, transaccions, finalitzar el dia, processos posteriors….) i "Regression Test" on es comprova que les noves funcionalitats no han trencat res del que existia anteriorment.

ETAPA DE CAT

El client valida que les implementacions realitzades són les funcionalitats que va demanar. Des del punt de vista de GTech es dona suport per si hi ha alguna prova que no saben fer o per arreglar algun bug que el client pugui detectar.

GO LIVE

L'etapa de "Go Live" és quan es posa el software en producció, és a dir, quan es fa la instal·lació.

MANTENIMENT I SUPORT

Aquesta etapa dóna suport i manteniment a l'empresa a la que se li ha posat el software en producció.

Tot i seguir el model Waterfall, que s'ha comentat abans, amb l'aparició de les metodologies àgils a l'empresa s'està intentant adoptar algunes de les bones pràctiques d'Agile.

2.2 - EL PROJECTE

És molt important que els productes que desenvolupa GTech funcionin correctament i que hi hagi el mínim d'errors possibles un cop el producte s'ha posat en producció, aquest fet és de vital importància ja que els usuaris dels productes estaran jugant amb diners i un error pot comportar grans pèrdues econòmiques tant a la loteria com a GTech per les condicions contractuals.

Per això totes les implementacions s'estan contínuament testejant de la forma més exhaustiva possible malgrat el cost que comporta és molt elevat. Per altre banda, la majoria dels tests s'estan realitzant de forma manual. Al realitzar-se manualment pot ser que alguna vegada el test no es realitzi exactament igual que un test anterior.

Es per això que es vol realitzar aquest projecte, que té com objectiu aconseguir realitzar proves més exhaustives que es realitzen cada vegada que un empleat realitza un canvi en el codi però al mateix temps que aquestes proves no comportin un augment d'hores dels QA. D'aquesta manera s'aconseguirà saber en tot moment quines funcionalitats fallen i en quin moment han començat a fallar.

A l'hora de realitzar el projecte s'ha definit un llenguatge comú i específic que permeti comunicar-se amb totes les parts involucrades en el projecte, d'aquesta manera es podrà utilitzar el que s'anomena "Specification by example" que s'aplica en mètodes de desenvolupament Àgil.

"Specification by example" consisteix en realitzar la definició de requisits i les proves utilitzant exemples reals en lloc d'afirmacions abstractes. Un dels avantatges d'utilitzar "Specification by example" és la reducció significativa de feedbacks durant el desenvolupament de software, donant lloc a menys feina, més qualitat del producte, disminució dels temps de resposta per canvis de software i una millor alineació de les activitats dels diferents rols involucrats en el desenvolupament.

Un dels motius pel que es vol poder utilitzar "Specification by example" és poder crear un únic document sobre el que s'ha de realitzar i, que aquest serveixi per totes les perspectives.

Actualment els "Business analysts" treballen amb els seus propis documents, els desenvolupadors tenen la seva pròpia documentació i els QA tenen un conjunt de proves funcionals. Això té com a conseqüència una reducció de l'eficàcia de lliurament de programari ja que s'ha d'estar constantment coordinant i sincronitzant els diferents documents.

En canvi, utilitzant "Specification by example" els diferents rols participen en la creació d'una única documentació que capta la comprensió de tothom. Amb els exemples es proporciona claredat i precisió, de manera que aquesta informació es pot utilitzar tant com una especificació, una prova funcional i un cop finalitza el projecte es converteix en un document que explica les funcionalitats existents.

A més a més també es vol realitzar un editor que permeti a tots els stakeholders crear el que s’anomenen stories que a trets generals son els exemples que servirien per crear el document únic.

El llenguatge que es defineixi serà el que s’utilitzi en l’aplicació que es vol realitzar. Aquesta aplicació tindrà un editor de tests que permetrà, a tots els membres de GTech, crear tests d’una forma molt senzilla i per tots els tipus de productes de que disposa. Per altre banda les persones que utilitzin aquesta aplicació també podran executar els tests i veure el seu resultat.

2.3 - PAPER REALITZAT EN EL PROJECTE

Com que GTech te una varietat molt amplia de productes que utilitzen diferents tecnologies i la manera de testejar-les són molt diferents entre elles (no és el mateix testejar una BBDD que un portal web), la part amb la que m'he centrat en aquest projecte és majoritàriament amb la definició d'un framework mitjançant el qual es pot fer el testeig Web i la interfície de l'aplicació, també he participat puntualment amb coses d'Altura (Terminal gràfic en el que es pot comprar loteria, consultar si un tiquet d'un joc ha aconseguit algun premi, etc.).

Pel que fa a la part Web m'he dedicat a dissenyar, implementar, testejar, etc. les diferents funcionalitats que es poden utilitzar en portals web, com per exemple crear un llenguatge pels tests que qualsevol empleat o client pugui entendre, aconseguir identificadors de diferents parts de la web, facilitar la creació de tests d'una forma més ràpida utilitzant el que hem anomenat "Story Telling", etc.

Per poder entendre millor tot el procés que els projectes segueixen a l'empresa, al llarg de tot el projecte, he participat en les diferents fases, gràcies a això m'ha estat més fàcil dissenyar parts de l'aplicació ja que he hagut de realitzar prototips a partir de documents SRS, afegir identificadors en el dashboard de la ONCE els quals els obtenia del prototype i crear test de les funcionalitats creades en el prototype mitjançant els identificadors afegits en el codi.

En quan a la interfície de l'aplicació la dedicació ha estat com en la Web, ajudant a fer la interfície el més "user friendly" possible i afegint-hi noves funcionalitats per als usuaris.

També he participat en la realització de manuals d'usuaris i vídeos per tal de facilitar l'aprenentatge de l'aplicació. Dintre d'aquest àmbit també he ajudat a ensenyar i donar suport de l'aplicació a empleats de Barcelona, la India, Estats Units, etc.

2.4 – METODOLOGIA DEL PROJECTE

Per realitzar aquest projecte s'han utilitzat alguna de les bones pràctiques que marca la metodologia Agile, és per això que el projecte s'ha dividit en cinc iteracions anomenades sprints.

Per saber en tot moment el que s'havia de realitzar en cada un dels diferents sprints, quan s'havia de fer i qui ho realitzava es va crear un product Backlog (un product Backlog conté descripcions prioritzades dels requisits, funcionalitats i errors a solucionar) i per cada sprint el seu corresponent sprint Backlog (priorització de les tasques necessàries per realitzar els requisits assignats al Sprint). Per fer el seguiment del projecte fent us del product Backlog es va utilitzar una eina anomenada Jira, una aplicació web que permet realitzar el seguiment de funcionalitats i el seguiment d'incidències i errors, gràcies a aquesta eina podíem saber que s'estava implementant en tot moment i qui ho estava realitzant.

Al realitzar el projecte amb més membres de l'empresa es va decidir utilitzar Subversion (SVN) com a repositori, es va decidir utilitzar Subversion en comptes d'alguna altre eina pel fet de ser l'eina que usualment s'ha utilitzat a l'empresa. SVN és una eina de software lliure amb llicència d'Apache/BSD. Una propietat que té SVN, com qualsevol controlador de versions, és que permet estar cada membre de l'equip treballant en el seu propi ordinador i quant algú finalitza alguna implementació la pot sincronitzar amb la branca del repositori.

Imatge 3: Logo Subversion

Tal i com s'ha explicat anteriorment l'empresa segueix el model Waterfall però s'està començant la transició cap a algunes bones practiques basades en Agile, per això durant tot el projecte es van utilitzar processos d'Extreme Programming com la integració continuada i el Test Driven Development (TDD).

Un dels objectius que té l'empresa és, seguint un model Agile, mantenir el focus en el valor dels stakeholders al llarg de tot el projecte, per aquest motiu l'eina que s'ha desenvolupat en aquest projecte s'ha realitzat pensant que s'utilitzi seguint un procés àgil anomenat "Behavior Driven Development(BDD)" dissenyat per establir un llenguatge comú entre totes les parts implicades en un projecte i que permeti tenir una documentació viva.

2.4.1 - INTEGRACIÓ CONTINUA

Integració continua és una pràctica de desenvolupament que forma part de les pràctiques Extreme Programming (XP), consisteix en integrar els diferents components del sistema i executar diversos conjunts de test d'un projecte de forma automàtica i el més aviat possible per tal de prevenir els problemes tan aviat com es pugui per així localitzar els bugs més fàcilment.

"Continuous Integration doesn't get rid of bugs, but it does make them dramatically easier to find and remove." **Martin Fowler**

Per realitzar integració continua s'han de seguir els següents principis: mantenir un sol repositori, les builds (s'executen els unit Tests i es crea un executable de l'aplicació) s'han de realitzar de forma automàtica i auto-testejables, les releases també han de ser automàtiques i auto-testejables, l'executable resultant de l'última build realitzada amb èxit ha de ser fàcil d'aconseguir i tothom ha de poder veure que esta passant en tot moment.

Avantatges

- Disminució del temps d'integració.
- Els errors és detecten i s'arreglen contínuament evitant l'acumulació d'aquests els dies pròxims a les releases.
- Disminueix el temps de resolució dels bugs.
- Tot el codi és testejat.
- Feedbacks immediats pels desenvolupadors.
- Reducció dels problemes d'integració i, conseqüentment, disminució dels temps d'entrega de software.

Inconvenients

- Es pot tardar bastant temps en la configuració inicial.

2.4.2 – TEST DRIVEN DEVELOPMENT (TDD)

Test Driven Development és una part fonamental del procés àgil eXtreme Programming (XP) formalitzat per Kent Beck. És un procés de desenvolupament de software basat en la repetició de petits cicles de desenvolupament.

Les fases d'aquests cicles són: el desenvolupador escriu un test en el que es prova una funció o una millora, un cop s'ha escrit el test es passa a escriure la mínima quantitat de codi per tal de que el test passi correctament i, per acabar, es refactoritza el codi per seguir els criteris d'acceptació.

CICLE DEL TDD

En el següent diagrama de flux es mostra el cicle del TDD, en el diagrama els processos s'indiquen mitjançant els quadrants, mentre que els rombes indiquen decisions.

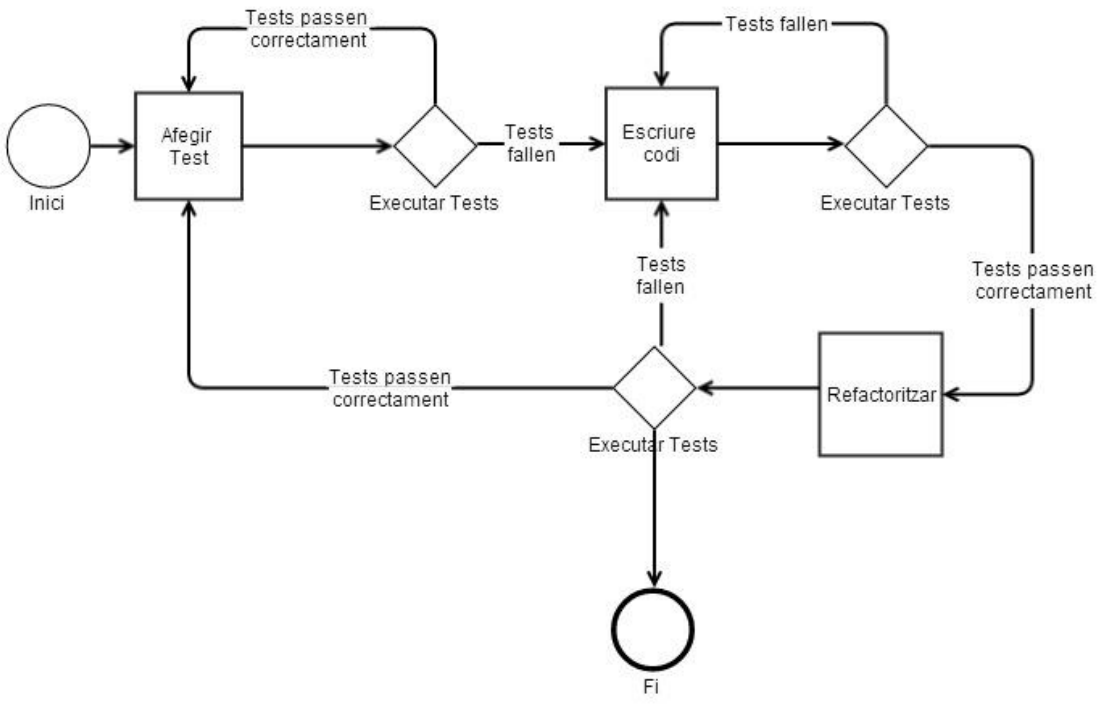


Diagrama 2: Cicle del TDD

AFEGIR UN TEST

Inicialment el test implementat ha de fallar ja que s'escriu el test abans que la implementació, si no és així vol dir que la nova "feature" ja existeix i conseqüentment (en cas que s'hagi seguit el procés correctament) ja existeix algun test que la comprova. També ens permet comprovar que el test fa la seva funció (falla perquè no hi ha implementació)

El fet que el desenvolupador escrigui el test abans de realitzar la implementació fa que el desenvolupador es fixi en els requisits abans d'escriure el codi, en canvi si s'escriu el codi abans pot ser que el desenvolupador no si hagi fixat.

EXECUTAR TOTS ELS TESTS I VEURE SI ALGUN FALLA

Verifica que tots els tests funcionen correctament i que el test que s'ha afegit inicialment falla.

ESCRIURE EL CODI

En aquest pas s'escriu, tal i com hem mencionat anteriorment, la mínima quantitat de codi per tal que el test passi correctament. El codi en qüestió no serà ni elegant ni perfecte però no és un problema ja que posteriorment es millorarà.

EXECUTAR TESTS

Si tots els tests passen satisfactòriament el programador té uns tests que li permeten executar-se unitàriament per comprovar si el requeriment que ha interpretat i implementat s'executa correctament, en cas contrari és necessari tornar al punt anterior.

REFACTORITZAR EL CODI

El codi es millora si és necessari. Fins ara el codi s'ha escrit per tal que passes el test però pot ser que el codi sigui poc llegible o difícil de mantenir. En aquest punt es posa el codi en el lloc on és més lògic que es trobi, també es comprova que no existeixin parts del codi duplicades, en cas que existeixin s'unifiquen. També es canvien els noms de les variables i/o mètodes per noms els quals representin el seu ús fent que el codi sigui autoexplicatiu.

Com a part de la Integració Contínua es tornen a executar tots els tests per estar segur que la refactorització no ha afectat al codi i que, per tant, no s'ha malmés cap funcionalitat.

Per exemple si es volgués crear un programa que donat un string qualsevol volgués girar les paraules (per exemple "Hola" passaria a ser "olaH") es realitzaria de la següent manera:

1- Crear un test que comprovi que donat un string el programa retorna un string.
2- Executar el test.
3- El test falla.
4- Escriure el codi necessari perquè donat un string el programa retorni un string.
5- Executar el test.
6- El test passa correctament.
7- Refactoritzar el codi.
8- Executar el test.
9- El test passa correctament
10- Escriure un test nou que comprovi que donat un string d'una paraula et retorna aquell string girat
11- Executar el test.
12- El test falla.
13- Escriure el codi necessari perquè donat un string d'una paraula agafi aquest string i el retorni girat
14- Executar el test
15- El test passa correctament
16- Refactoritzar el codi.
17- Executar el test.
18- El test passa correctament.
19- ...

L'exemple seguiria el cicle fins que el programa realitzes el que vol el client.

És recomanable tornar a executar tots els tests per estar segur que la refactorització no ha afectat al codi i que, per tant, no s'ha malmés cap funcionalitat.

Tal i com s'ha explicat abans, TDD es basa en la creació de test abans que el codi, d'aquesta manera s'aconsegueix entendre millor els requisits del producte n'assegura la

seva efectivitat i es pot centrar en la qualitat del producte. L'idea que els tests inicialment fallin és que d'aquesta manera s'assegura que el test funciona correctament i és capaç de detectar l'error.

TDD a diferència dels processos de desenvolupament tradicionals disminueix la probabilitat de que puguin aparèixer problemes durant la producció a causa de que una prova no és prou amplia. En els processos tradicionals la fase de testeig es realitza un cop el codi ha estat escrit, fins i tot passat un temps, i com a conseqüència es necessita temps i esforç per tornar endarrere en el codi per tal d'entendre'l i saber perquè està fallant, a més a més, moltes vegades l'equip de testeig comprova la correctesa del software en base a l'execució manual de "test cases" provinents dels requisits i, per tant, no entenen tots els detalls del codi que ha de ser testejat i poden deixar-se de comprovar algunes parts del codi.

Usualment en metodologies clàssiques els tests no estan automatitzats i, al no estar-ho, no es realitzen tests freqüentment o amb molta regularitat això provoca que els tests que es realitzin no es faran sempre de la mateixa manera això pot provocar que un cop arreglat algun problema, la solució pot malmetre una altre part del software però pot ser que la infraestructura de testeig no ho detecti.

BENEFICIS D'UTILITZAR TDD

- Tot el codi és testejat com a mínim per un test, donant al codi un alt nivell de confiança.
- Es triben abans errors en el codi i, per tant, poden ser corregits abans, disminuint els costos alhora de debugar.
- Com que només s'escriu el codi necessari per passar els test, tests automàtics tendeixen a cobrir tot el codi, és a dir, detecten qualsevol canvi inesperat en el comportament del codi, d'aquesta manera es detecten problemes que poden sorgir quan es realitza un canvi posterior en el cicle de desenvolupament el qual altera de forma inesperada alguna funcionalitat.

INCONVENIENTS D'UTILITZAR TDD

- Si els caps de projecte i/o les persones involucrades en el projecte no creuen en TDD aquest no aconseguirà tots els beneficis que ofereix ja que les persones que no hi creuen veuran la creació de tests com una pèrdua de temps.
- Un nombre elevat de tests unitaris que passen correctament poden portar a una falsa sensació de seguretat que pot dur a realitzar menys tests d'integració dels que realment serien necessaris.
- Els tests que estan escrits malament (com per exemple aquells que tenen "hard-coded error strings") tenen un cost de manteniment elevat.

2.4.3 – BEHAVIOUR DRIVEN DEVELOPMENT (BDD)

Behaviour Driven Development (BDD) és un procés àgil dissenyat per mantenir el focus en l'stakeholder durant tot el projecte. El seu creador, Dan North, la va crear després de trobar-se que quan explicava TDD sempre es trobava amb els mateixos problemes i mals entesos, els programadors volien saber per on començar, que testejar i que no, quantes coses testejar en un sol test, com anomenar els tests i com saber perquè un test fallava.

Dan North el 2009 en una conferencia a la "Agile specifications, BDD and testing eXchange" a Londres, va definir BDD de la següent manera:

"BDD IS A SECOND-GENERATION, OUTSIDE-IN, PULL-BASED, MULTIPLE-STAKEHOLDER, MULTIPLE-SCALE, HIGH-AUTOMATION, AGILE METHODOLOGY. IT COMBINES A CYCLE OF INTERACTIONS WITH WELL-DEFINED OUTPUTS, RESULTING IN THE DELIVERY OF WORKING, TESTED SOFTWARE THAT MATTERS"

Tot i canviar l'estructura dels tests el procés que segueix BDD és el mateix que el de TDD.

EN QUE MILLORA BDD LES TÀCTIQUES DE TDD

- Implementa només allò que sigui estrictament necessari
- Descriu els comportaments en una única notació en la que tots els rols puguin fer servir (Business Analysts, testers, desenvolupadors, etc), d'aquesta manera es millora la comunicació entre els diferents rols.
- BDD és mes adequat pels tests funcionals i d'integració.
- El procés de compilació de requisits és molt millor gracies a les narratives i escenaris.
- Millor comunicació amb els clients.

ESTRUCTURA D'UNA STORY

Els tests creats amb BDD se'ls anomena "stories". Aquests estan formats per dues parts, la primera, anomenada "Narrative" que només existeix una per story i la segona, anomenada "Scenario", on en una mateixa Story poden haver-hi molts.

NARRATIVE

És una descripció de la funcionalitat explicada des de la perspectiva de la persona que utilitzarà aquesta funcionalitat. Intenta contestar tres qüestions molt bàsiques que moltes vegades amb els requisits tradicionals es passen per alt. Aquestes qüestions són quin és el benefici o valor que ha de produir (expressat amb la sentència "In order to"), qui ho necessita ("As a") i quina és la funció o l'objectiu ("I want to"). Amb la resposta a aquestes preguntes l'equip pot començar a definir la millor solució, amb la col·laboració dels stakeholders.

```
Narrative:
In Order to have customized experience
As a site visitor
I want to be able to login
```

Només amb la narrativa no s'aconsegueix una descripció completa de la funcionalitat, és necessari haver realitzat el/els escenaris.

Una bona narrativa ha de ser independent per reduir les dependencies i així poder realitzar una planificació millor. Negociable, els detalls s'afegeixen a través de la col·laboració. Valuosa, és a dir que doti de valor al client. Estimable, per poder dur a terme una planificació. Petita, s'ha de poder realitzar per l'equip en, més o menys, una setmana.

SCENARIOS

Els escenaris descriuen les interaccions dels diferents usuaris i del sistema. Estan escrits en un llenguatge que conté el mínim de detall tècnic de manera que qualsevol stakeholder ho pugui entendre.

El conjunt dels escenaris d'una story són el criteri d'acceptació de la narrativa d'aquella story.

Un escenari esta format per una descripció i un conjunt de steps. La descripció de l'escenari no ha de ser molt llarga, més aviat ha de ser una curta explicació del que realitza l'escenari, no ha de contenir detalls i idealment no hauria de tindre més de deu paraules.

Pel que fa als steps, tots comencen amb una paraula que pot ser "Given", "When" o "Then". Aquestes paraules serveixen per identificar si el step és una precondició, en el cas de "Given", si és una acció ("When") o si es vol realitzar una comprovació ("Then"). Després els steps estan formats per sentencies que identifiquen el que realitza l'step en qüestió. A més a més els steps poden acceptar paràmetres els quals s'indiquen amb el símbol "$" seguits d'un nom.

Per exemple per tots els steps destinats a prémer algun element web es podrien crear steps com els següents:

```
When Web user clicks the $id element
When Web user clicks the $id link
When Web user clicks the $id button
```

En aquest steps es pot veure que no fa falta crear un step per cada boto sinó que es pot utilitzar el mateix step substituint el $id per l'identificador del botó.

Si utilitzem la següent Narrativa:

```
Narrative:
In order to have customized experience
As a site visitor
I want to be able to login
```

Amb la narrativa ja escrita, tots els membres de l'equip podrien crear conjuntament els següents escenaris:

- Visitors should be able to register
- Existing users should be able to login
- There should be an option to retrieve password

Una possible implementació inicial de l'escenari "Visitors are able to register" podria ser:

```
[BDD Story]
Scenario: Visitors are able to register

Given visitors is on the home screen
When visitor clicks the register button
Then login screen is loaded
When visitor types his email john.doe@example.com
When visitor types his password mysecretpass
When vistior clicks the register button
Then confirmation text Welcome john.doe@example.com is displayed
```

Un cop creat aquest escenari els membres del projecte poden adonar-se'n que falten algunes precondicions com que per exemple per crear un usuari no pot haver-hi un de loguejat, per tant al inici de l'escenari faría falta un step de tipus "Given", de l'estil: "Given visitor is not logged in". També és possible que després de crear l'escenari inicial s'arribi a la conclusió que el password s'ha de repetir per confirmar que realment és el que l'usuari vol. Tenint en compte aquestes modificacions l'escenari quedaria de la següent manera:

```
[BDD Story]
Scenario: Visitors are able to register

Given visitors is on the home screen
Given visitor is NOT logged in
When visitor clicks the register button
Then login screen is loaded
When visitor types his email john.doe@example.com
When visitor types his password mysecretpass
When visitor types his confirmation password mysecretpass
When vistior clicks the register button
Then confirmation text Welcome john.doe@example.com is displayed
```

Tot i la posterior millora de l'escenari l'equip del projecte podria veure que falten comprovacions per fer, com per exemple, aquest test actualment només serveix per veure que un usuari es pot registrar però no contempla la possibilitat que s'equivoqui al reescriure el password, que el password no tingui la longitud desitjada, que el format del email és correcte, etc

Per solucionar tots els punts anteriors no fa falta realitzar un escenari per cada una de les diferents situacions, utilitzant el que s'anomena "Example table" es pot gestionar fàcilment. Les "Example table" serveixen precisament per poder crear diferents iteracions d'un mateix escenari introduint algunes variacions en alguns valors.

Així doncs l'exemple anterior si es fes us d'una "Example table" quedaria:

```
[BDD Story]
Scenario: Visitors are able to register

Given visitors is on the home screen
Given visitor is NOT logged in
When visitor clicks the register button
Then login screen is loaded
When visitor types his email <email>
When visitor types his password <password>
When visitor types his password <password2>
When vistior clicks the register button
Then confirmation text Welcome <text> is displayed

Examples:
|email            |password|password2|text                          |
|john.doe@exam.com|mysecret|mysecret |Welcome john.doe@example.com |
|john.doe@exam.com|secret  |secret   |Pass must be at least 8 chars|
|john.doe@exam.com|secret  |something|Validation pass  is NOT equal|
|john.doe         |mysecret|mysecret |Email is incorrect           |
```

Amb aquest últim exemple el mateix escenari ens serviria per comprovar els diferents casos de un registre amb èxit, quan hi ha un password incorrecte, quan la validació del password no és correcte i quan s'introdueix malament un email. L'escenari s'executaria en un total de 4 vegades on a cada una s'utilitzaria una fila de la taula "Examples" que és la que identifica la "Example Table". La primera fila de la taula ens serveix per establir el nom de cada columna, així a l'hora d'executar l'escenari quan es trobi un valor encapsulat amb els símbols "<" i ">" buscarà a la taula la cel·la que es troba a la fila que s'està executant en aquell moment i la columna que tingui el mateix nom que el que es troba entre els dos símbols i ho substituirà pel valor que tingui.

Per exemple a l'escenari anterior al afegir l'Example Table s'executaria cuatre vegades. A la primera execució el escenari s'executaria com si fos de la següent manera:

```
Scenario: Visitors are able to register

Given visitors is on the home screen
Given visitor is NOT logged in
When visitor clicks the register button
Then login screen is loaded
When visitor types his email john.doe@exam.com
When visitor types his password mysecret
When visitor types his password mysecret
When vistior clicks the register button
Then confirmation text Welcome Welcome john.doe@example.com is displayed
```

A la segona execució l'escenari s'executaria així:

```
Scenario: Visitors are able to register

Given visitors is on the home screen
Given visitor is NOT logged in
When visitor clicks the register button
Then login screen is loaded
When visitor types his email john.doe@exam.com
When visitor types his password secret
When visitor types his password secret
When vistior clicks the register button
Then confirmation text Welcome Pass must be at least 8 chars is displayed
```

I així seguirien les altres dues execucions.

Com es pot veure, un escenari a l'inici no s'escriu d'una manera perfecte sinó que mitjançant iteracions es va millorant ja que a cada una d'aquestes iteracions apareixen noves qüestions i noves possibilitats que han de ser testejades.

AUTOMATITZACIÓ DE LES STORIES

Gracies a que les stories segueixen unes regles molt clares i es fàcilment identificable quan és una precondició/acció/comprovació, aquestes es poden utilitzar per realitzar tests automatitzats.
Actualment ja existeixen alguns frameworks per realitzar BDD com JBehave (Java), Cucumber (Ruby), RSpec (Ruby), SpecFlow (.Net) o Jasmine (Javascript). Per explicar com automatitzar les stories utilitzaré JBehave ja que era el framework que he utilitzat i per tant és amb el que em resultava més fàcil realitzar els exemples, tot i així el funcionament dels altres frameworks és el mateix, introduint petites variacions.

La manera habitual d'utilitzar BDD és escrivint un codi que comprovi la implementació en funció de la clàusula a comprovar. L'inconvenient de realitzar-ho d'aquesta manera és que tota aquella gent que no tingui coneixements de programació no pot utilitzar-ho, per això, si es vol que qualsevol persona faci tests utilitzant BDD, com és el cas de GTech, una manera és separant els steps del codi. En el cas particular de GTech per aconseguir´ho ha estat necessari passar per tres fases:

1. Crear una llibreria normalitzada de steps.
2. Combinar els steps creats en la fase anterior per convertir-lo en un únic step.
3. Millorar els escenaris amb l'ús de "Example tables".

CREAR UNA LLIBRERIA NORMALITZADA DE STEPS

Per crear aquesta llibreria és necessari separar els escenaris del codi. L'idea de separar el codi dels escenaris és degut a que qualsevol membre d'un projecte que utilitza BDD ha de ser capaç de realitzar escenaris, això vol dir que tant "business stakeholders", desenvolupadors, testers, etc. poden no tenir coneixements de programació però han de poder crear-ne per això és necessari separar el codi dels escenaris.

Les "stories" estan escrites seguint un format de text pla i cada step esta associat a un mètode en el codi. El que han de realitzar els frameworks de BDD és realitzar el "match" entre els steps de les stories i el codi corresponent a cada step.

Per exemple, per fer el match del step "When user clicks the login button" faria falta el següent:

```
[BDD Story]
Scenario: Visitors are able to register
…
When user clicks the login button
…

[Codi en java]
…
@When ("user clicks the $id button")
public final void clickButton(final String id) {
    // Code that clicks the button with specified ID
}
…
```

Una bona pràctica és crear classes separades per cada tipus d'aplicació que s'estigui desenvolupant. Per exemple es podrien crear diferents classes per steps de tipus web, bases de dades, APIs, etc. Aquestes classes poden tindre steps en comú per això és necessari que totes estenguin d'una classe comuna en la que es trobin els steps que poden utilitzar totes les classes. A més a més també es podrien crear classes que estenguin a les classes dels diferents productes, així es podrien crear classes amb steps específics per cada producte.

Per exemple l'estructura de classes a GTech seria la següent:

- CommonSteps
  - WebSteps
    - Dashboard
      - OnceDashboard
    - FlashSteps
  - CommandPromtSteps
    - ShellSteps
      - AlturaSteps
      - EsteSteps
      - HhtSteps
    - TelnetSteps
      - GolsSteps
  - DBSteps

L'estructura de classes CommonSteps contindria els mètodes relacionats amb variables, setups, etc. WebSteps tindria els steps per realitzar accions com fer click en els elements web, validar un text d'un element, etc. A DashboardSteps estarien els steps específics dels dashboards que ofereix GTech (interfícies gràfiques on els usuaris poden administrar un equip o software). CommandPromptSteps contindria steps per realitzar comandes de prompts. ShellSteps s'utilitzaria per utilitzar comandes de Shell. AlturaSteps, EsteSteps, HhtSteps s'utilitzarien per interactuar amb els productes de GTech que s'anomenen Altura, Este i Hht (Handheld Terminal). Per últim DBSTeps serviria per poder interactuar amb bases de dades.

És molt fàcil que a l'hora de crear steps que es creïn diferents steps que realitzen la mateixa acció/verificació, per exemple en una pagina web es pot fer click en un link, un boto, un element, etc. Per realitzar el codi es podria crear un mètode per cada step on a cada un dels mètodes el seu codi seria el mateix que els altres. Aquesta repetició de codi es pot evitar utilitzant "Aliases", d'aquesta manera els diferents steps utilitzen el mateix mètode, eliminant la repetició de codi.

```
[Codi en Java]
@When("Web user clicks the $id element")
@Aliases(values={
 "Web user clicks the $id button",
 "Web user clicks the $id link"
})
public final void clickElement(final String id) {
 //Codi per realitzar un click
}
```

COMBINAR ELS STEPS CREATS EN LA FASE ANTERIOR EN UN ÚNIC STEP

Pot ser que molts escenaris continguin un conjunt de steps exactament iguals, o alguns steps poden arribar a ser irrellevants pel propòsit de l'escenari, però al mateix temps necessaris, com per exemple els steps necessaris per loguejar-se.

```
[BDD story]
Scenario: Visitors are able to register

Given user is on the home screen
When user types john.doe@example.com into the email field
When user types my_secret into the password field
When user clicks the login button
Then confirmation screen is opened
…
```

Aquests conjunt de steps es podrien substituir per:

```
    [BDD story]
    Scenario: Visitors are able to register

    Given user is logged in
    …
```

Per crear el step anterior i poder-lo utilitzar es poden agrupar el conjunt de steps en el que s'anomena composite de la següent manera:

```
[Codi en java]
@Given("user is logged id")
@Composite(steps = {
    "Given user is on the home screen"
    "When user types john.doe@example.com into the email field"
    "When user types my_secret into the password field"
    "When user clicks the login button"
    "Then confirmation screen is openend"
})
```

D'aquesta manera els cinc steps es quedarien agrupats en un de sol.

MILLORAR ELS ESCENARIS AMB L'ÚS DE "EXAMPLE TABLES"

Les "Example table", tal i com s'ha explicat anteriorment, serveixen per poder crear diferents iteracions d'un mateix escenari introduint algunes variacions en alguns valors.

D'aquesta manera no fa falta crear diferents escenaris en els que l'únic que canvia és la comprovació de si un text és correcte o no, marcar un checkbox, etc.

3 - ITERACIONS

El projecte s'ha realitzat en cinc iteracions on cada una d'elles es realitzaven algunes de les funcionalitats introduïdes al Jira.

En cada una de les iteracions es realitzaven els requisits, el disseny, la implementació i els tests. A la memòria en aquest apartat només es trobaran els requisits i el disseny ja que la implementació i els tests han estat molt semblants en totes les iteracions i per aquest motiu estan explicats més endavant en apartats individuals.

Desprès de les tres primeres iteracions es va realitza una release oficial de cara a l'empresa per tal que alguns treballadors comencessin a utilitzar l'eina que s'havia desenvolupat fins el moment i així poder començar a rebre feedbacks de les coses que s'havien de canviar i de les que ja estaven bé.

3.1 - ITERACIÓ 1

3.1.1 – ANÀLISIS DE REQUISITS

En aquesta primera iteració l'objectiu primordial ha estat aconseguir establir un llenguatge que tots els treballadors puguin entendre i, a més a més, que segueixi l'estructura que marca BDD (Narrativa + Escenaris).

Pel que fa a l'establiment d'un llenguatge s'ha decidit diferenciar-los per categories segons el producte que aquests steps testegen i, a més a més, segons si eren steps bàsics o no, d'aquesta manera l'usuari pot filtrar els steps segons si vol steps molt bàsics i genèrics o si vol utilitzar-ne de més específics. Tal i com estableix BDD s'han utilitzat les paraules "Given", "Then" i "When" per les precondicions, accions i postcondicions respectivament.

Pels steps de tipus "Given" s'ha tingut en compte que un usuari abans de realitzar un test voldrà establir algunes regles de configuració, com en quin idioma es troba la web, quina url es troba, mida del navegador, etc. També s'han creat sentències que ajudin a l'hora de veure perquè ha fallat un test, com per exemple, poder guardar una imatge de com es trobava la web en el moment en que el test ha fallat (snapshot). I finalment, també s'han establert steps de com ha d'actuar l'eina a l'hora de dur a terme el test, com per exemple, que cada acció que es realitzi la faci utilitzant un "click" o prement la tecla "Enter".

En els steps de tipus "When" s'ha intentat que aquests abastin el màxim d'accions que un usuari pot realitzar en una web, com per exemple prémer un element (ja sigui un boto, un link, etc), seleccionar una opció d'una llista desplegable, escriure un text en un camp de text, etc.

Finalment, amb els steps de tipus "Then" com estan pensats per dur a terme comprovacions s'han pensat sentencies que comprovin que els elements d'una web tinguin un text determinat o que continguin un fragment de text, consultar si un element esta visible/deshabilitat, consultar els valors d'una cel·la d'una taula, etc.

Pel que fa a la part de l'aplicació s'ha implementat les parts necessàries per poder crear stories utilitzant un editor..

Pel que fa a l'edició de les stories l'aplicació tindrà un mode per defecte en el que els usuaris podran afegir steps fent drag&drop d'una llista desplegable de steps, modificar la story a través de un menú d'edició en el que hi hauran opcions com "Add New Scenario" o "Add Given Stories", etc.

Apart del mode d'edició per defecte s'ha considerat que alguns usuaris poden trobar més còmode escriure les stories ells mateixos a través d'un editor de text el qual es podrà utilitzar si es passa l'editor al mode manual. Per tant, l'aplicació es podrà utilitzar en el mode per defecte o en mode manual per aquells usuaris que vulguin escriure les stories ells mateixos. Si l'usuari utilitza l'editor manual és molt probable que s'equivoqui al escriure algun step, o que el format que segueixi no sigui del tot correcte, per aquest motiu quan un

usuari canviï del mode manual al mode per defecte, l'aplicació, comprovarà si la story és correcte i no conté errors.

En molts casos per no fer stories molt llargues o que moltes d'elles realitzin el mateix en la major part de l'escenari, es voldrà separar en varies stories. Per identificar que una story va abans que una altre, a la story que necessita que s'executi abans ho indicarà a través del que s'anomena "Given Stories".

Per exemple, a les stories de Web que sigui necessari realitzar un login es podrà crear una story que realitzi un login i, desprès, totes les altres stories no hauran de repetir els mateixos steps, simplement hauran d'indicar que tenen com a "Given Story" la story de login.

En el següent conjunt de stories es pot veure com la segona i tercera story tenen com a "Given Story" la primera, així, quan es realitzi una execució de la segona o la tercera story es realitzaran els steps que es troben a login.story.

```
[BDD login.story]
Scenario: User logs in

Given user is on the home screen
When user types john.doe into the user field
When user types my_secret into the password field
When user clicks the login button
Then Web wait for 5 seconds


[BDD story]
Given Story: login.story

Scenario: User returns a ticket

Given user is on the home screen
When Web user selects Servicio de Instantánea from the droplist menu1
When Web user selects Opciones de Sistema from the droplist menu2
When Web user selects Devolución Boletos from the droplist menu3
…

[BDD story]
Given Story: login.story

Scenario: User goes to product management

Given user is on the home screen
When Web user selects OLPM from the droplist menu1
When Web user selects Gestión de Productos from the droplist menu2
…
```

Per clarificar que és el que s'està fent en un step en concret, un usuari pot voler introduir comentaris ja que és possible que amb la descripció de l'escenari no li sigui suficient per clarificar algun punt específic.

Un punt molt important de l'aplicació és la usabilitat, ja que es vol millorar l'eficiència en la creació de tests per això l'aplicació incorporarà una llista amb els steps que pot utilitzar, a més a més, aquesta llista es filtrarà segons el producte que estigui seleccionat a l'aplicació (steps de tipus Web, de BBDD, d'Altura, etc), també es podrà seleccionar si es vol veure tots els steps del tipus que s'hagi seleccionat o només els que siguin més bàsics. Per no haver d'anar escrivint els steps cada vegada, l'usuari podrà arrossegar l'step desitjat de la llista i posar-lo en el punt de l'escenari que desitgi (Drag&Drop).

En relació a l'execució de tests en el cas de stories que contenen steps de tipus web, l'usuari podrà executar les stories o els scenarios a través de l'aplicació, i podrà triar entre els tres navegadors principals (Chrome, Firefox i Internet Explorer), així si l'usuari ho desitja, pot provar que les stories funcionen en diferents navegadors o, com en el cas del Dashboard de la ONCE (interfície gràfica on usuaris de la ONCE poden administrar els seus productes), si la story necessita ser executat des d'un navegador en concret també es pot fer.

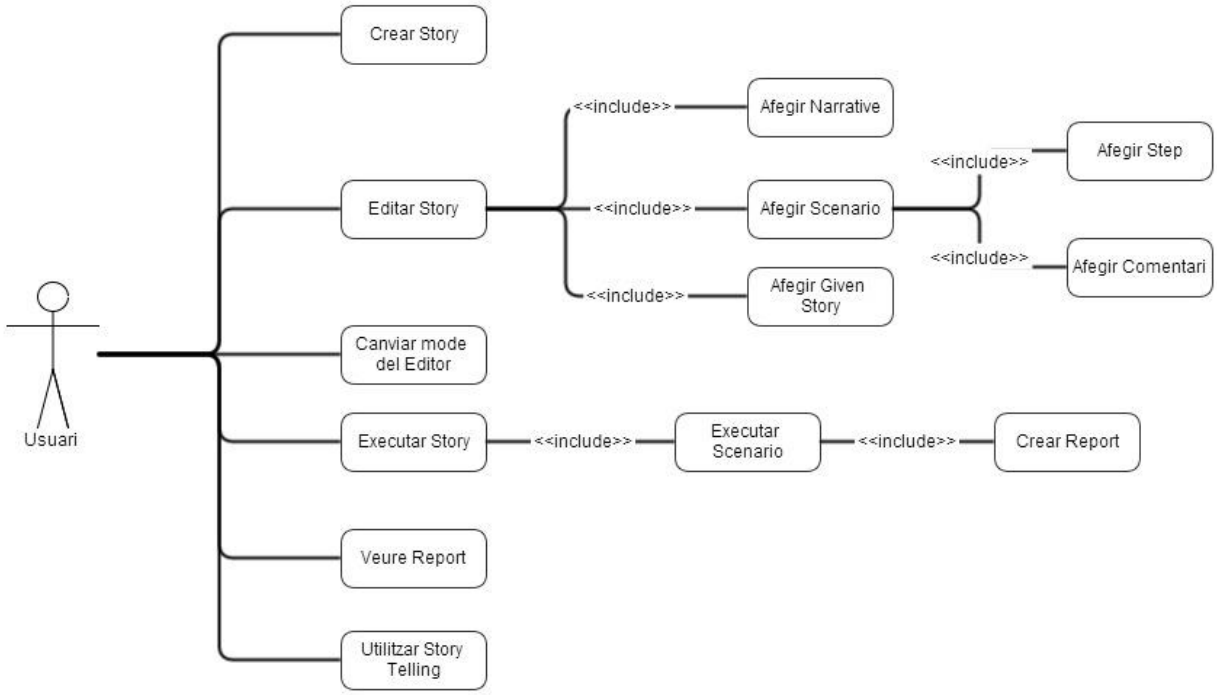
Un cop s'hagi realitzat una execució d'una story, serà útil poder visualitzar un report per comprovar en quin punt s'ha executat satisfactòriament, o en quin punt ha fallat. Per donar més informació, el report indica els steps que han passat satisfactòriament, els que no han passat correctament o, en el cas de que la story tingui un step que no existeixi en la implementació, el marca com a "PENDING".

Per últim, en aquesta iteració, també s'ha realitzat una eina per ajudar a implementar stories de forma més rapida, l'eina l'hem anomenat "Story Telling" i serveix tant per les màquines Altura com per les pàgines Web. Story Telling consisteix en una eina amb la que l'usuari realitza les accions directament en l'aplicació final, i posteriorment l'eina li proposarà els possibles steps que són necessaris per poder crear la story. Per exemple si un usuari vol crear un test en el que s'accedeix a una web, comprovar si el títol de la web és el correcte i desprès loguejar-se utilitzant Story Telling, el que hauria de realitzar és obrir l'aplicació, en el cas de la web, indicar quin navegador vol utilitzar i quina URL s'ha d'obrir, un cop l'usuari es troba en el lloc on vol que l'aplicació comenci, li indica que comenci a gravar i just després podria prémer a l'àrea de text on s'ha d'escriure l'usuari, escriure'l, fer el mateix amb el del password, prémer el botó de "log in" i un cop estigués loguejat indicar-li al Story Telling que ja li pot proposar els steps, un cop li demani li obrirà un popUp i li sortiran els possibles steps que hauria d'afegir a una story per poder realitzar el mateix que acaba de fer.

3.1.2 ESPECIFICACIÓ

En aquest apartat es detalla l'especificació de les funcionalitats que es vol que tingui l'aplicació que es vol crear.

CASOS D'ÚS



Cas d'ús	**Crear Story**
Actors	Usuari
Precondicions	

Flux bàsic:

Precondició:	
Actor	Sistema
1) L'usuari prem el botó "New Story"	
	2) El sistema demana especificar el lloc on es vol guardar la nova story
3) L'usuari especifica el nom de la story i el lloc on vol guardar-la 4) L'usuari prem el botó "Save"	
	5) El sistema obra una nova pestanya amb la story creada
Postcondició:	S'ha creat i guardat una nova story

Fluxes alternatius:

Flux Alternatiu: crear story utilitzant el menú superior

Precondició	
Actor	Sistema
1) L'usuari escull l'opció "New Story" en el menú "File" 3) L'usuari especifica el nom de la story i el lloc on vol guardar-la 4) L'usuari prem el botó "Save"	2) El sistema demana especificar el lloc on es vol guardar la nova story 5) El sistema obra una nova pestanya amb la story creada
Postcondició	S'ha creat i guardat una nova story

Flux Alternatiu: crear story utilitzant el shortcut

Precondició	
Actor	Sistema
1) L'usuari prem els botons "Cntrl"+"N" 3) L'usuari especifica el nom de la story i el lloc on vol guardar-la 4) L'usuari prem el botó "Save"	2) El sistema demana especificar el lloc on es vol guardar la nova story 5) El sistema obra una nova pestanya amb la story creada
Postcondició	S'ha creat i guardat una nova story

Cas d'ús	**Editar story**
Actors	Usuari
Precondicions	1) Es necessita haver creat una story 2) La story ha d'estar oberta en el editor

Flux bàsic:

Precondició:	L'editor es troba en el mode per defecte
Actor	Sistema
1) L'usuari edita la story	
Postcondició:	La story s'ha editat

Fluxes alternatius:

Flux Alternatiu: editar la story manualment

Precondició:	L'editor es troba en el mode manual
Actor	Sistema
1) L'usuari edita la story	
Postcondició:	La story s'ha editat

Cas d'ús	**Afegir narrative**
Actors	Usuari
Precondicions	1) Es necessita haver creat una story 2) La story ha d'estar oberta en el editor

Flux bàsic:

Precondició:	L'editor es troba en el mode per defecte
Actor	Sistema
1) L'usuari escull l'opció "Add Narrative" del menú "Edit" 2) L'usuari indica el "Benefit" que vol posar a la narrative" 3) L'usuari indica el "Rol" que vol posar a la narrative 4) L'usuari indica la "Feature" que vol posar a la narrative 5) L'usuari prem el botó "Ok"	6) El sistema posa la narrative a la part superior de la story que es té seleccionada
Postcondició:	La story conté una narrativa

Fluxes alternatius:

Flux Alternatiu: afegir narrative manualment

Precondició:	L'editor es troba en el mode manual
Actor	Sistema
1) L'usuari escriu tota la narrativa	
Postcondició:	La story conté una narrativa

Cas d'ús	**Afegir Given Story**
Actors	Usuari
Precondicions	1) Es necessita haver creat una story 2) La story ha d'estar oberta en el editor

Flux bàsic:

Precondició:	L'editor es troba en el mode per defecte
Actor	Sistema
1) L'usuari escull l'opció "Add Given Story" del menú "Edit"	2) El sistema introdueix el camp "Given Story" a la story
3) El usuari selecciona la given story	4) El sistema introdueix la ruta de la given story
Postcondició:	La story conté una "Given Story"

Fluxes alternatius:

Flux Alternatiu: afegir given story manualment

Precondició:	L'editor es troba en el mode manual
Actor	Sistema
1) L'usuari escriu la given story	
Postcondició:	S'afegeix una given story a la story

Cas d'ús	**Afegir scenario**
Actors	Usuari
Precondicions	1) Es necessita haver creat una story 2) La story ha d'estar oberta en el editor

Flux bàsic:

Precondició:	L'editor es troba en el mode per defecte
Actor	Sistema
1) L'usuari escull l'opció "Add Scenario" del menú "Edit" 2) L'usuari escriu una breu descripció del que realitza l'escenari	3) El sistema afegeix un escenari a la story que es té seleccionada.
Postcondició:	S'afegeix un escenari a la story

Fluxes alternatius:

Flux Alternatiu: afegir scenario manualment

Precondició:	L'editor es troba en el mode manual
Actor	Sistema
1) L'usuari escriu l'escenari	
Postcondició:	S'afegeix un escenari a la story

Cas d'ús	**Afegir step**
Actors	Usuari
Precondicions	1) Es necessita haver creat una story 2) La story ha d'estar oberta en el editor 3) La story té, com a mínim, un scenario creat

Flux bàsic:

Precondició:	L'editor es troba en el mode per defecte
Actor	Sistema
1) L'usuari tria un step de la llista de possibles steps 2) L'usuari arrastra el step fins a la posició de l'escenari on vol introduir-lo 3) L'usuari canvia tots els paràmetres del step (indicats amb el símbol $) per els valors que desitja	4) El sistema introdueix el step a la posició que l'usuari ha indicat.
Postcondició:	S'afegeix un step a l'escenari

Fluxes alternatius:

Flux Alternatiu: afegir step manualment

Precondició:	L'editor es troba en el mode manual
Actor	Sistema
1) L'usuari escriu el step a dins l'escenari	
Postcondició:	S'afegeix un step a l'escenari

Cas d'ús	**Afegir comentari**
Actors	Usuari
Precondicions	1) Es necessita haver creat una story 2) La story ha d'estar oberta en el editor 3) La story té, com a mínim, un scenario creat

Flux bàsic:

Precondició:	L'editor es troba en el mode per defecte
Actor	Sistema
1) L'usuari selecciona el step on vol que després es posi el comentari 2) L'usuari escull l'opció "Add Comment" del menú "Edit" 3) L'usuari escriu el comentari que vol que aparegui 4) L'usuari prem el botó "Ok"	6) El sistema introdueix el comentari
Postcondició:	S'afegeix un comentari a l'escenari

Fluxes alternatius:

Flux Alternatiu: afegir comentari manualment

Precondició:	L'editor es troba en el mode manual
Actor	Sistema
1) L'usuari escriu el comentari a dins l'escenari	
Postcondició:	S'afegeix un comentari a l'escenari

Cas d'ús	**Canviar mode del editor**
Actors	Usuari
Precondicions	Existeix una story oberta en el editor

Flux bàsic:

Precondició:	L'editor es troba en el mode per defecte
Actor	Sistema
1) L'usuari prem el botó "Toggle Editor"	2) El sistema canvia la vista de mode per defecte a mode manual
Postcondició:	L'editor esta en mode manual

Fluxes alternatius:

Flux Alternatiu: canviar del mode manual al mode per defecte

Precondició:	L'editor es troba en el mode manual
Actor	Sistema
1) L'usuari prem el botó "Toggle Editor"	2) El sistema comprova que tota la story té un format correcta i no conté errors 3) El sistema indica les línies que contenen errors, si en hi ha 4) El sistema canvia la vista de mode manual a mode per defecte
Postcondició:	L'editor esta en mode per defecte

Cas d'ús	**Executar story**
Actors	Usuari
Precondicions	1) Es necessita haver creat una story

Flux bàsic:

Precondició:	
Actor	Sistema
1) L'usuari prem el botó "Save and Run Story"	
	2) El sistema demana els paràmetres que creu necessaris per poder executar la story
3) L'usuari indica quin valor tenen els paràmetres, en cas que no sàpiga que ha de posar en algun dels camps ho consulta prement el botó d'informació. 4) L'usuari prem el botó "Run"	
	5) El sistema executa la story
Postcondició:	La story s'ha executat

Fluxes alternatius:

Flux Alternatiu: executar story a través del menú

Precondició:	
Actor	Sistema
1) L'usuari escull l'opció "Save and Run Story" del menú "File"	
	2) El sistema demana els paràmetres que creu necessaris per poder executar la story
3) L'usuari indica quin valor tenen els paràmetres, en cas que no sàpiga que ha de posar en algun dels camps ho consulta prement el botó d'informació. 4) L'usuari prem el botó "Run"	
	5) El sistema executa la story
Postcondició:	La story s'ha executat

Flux Alternatiu: executar story utilitzant el shortcut

Precondició:	
Actor	Sistema
1) L'usuari prem els botons "Cntrl" + "R"	
	2) El sistema demana els paràmetres que creu necessaris per poder executar la story
3) L'usuari indica quin valor tenen els paràmetres, en cas que no sàpiga que ha de posar en algun dels camps ho consulta prement el botó d'informació. 4) L'usuari prem el botó "Run"	
	5) El sistema executa la story 7) El sistema mostra el report i el va actualitzant a mesura que es va executant la story
Postcondició:	La story s'ha executat

Cas d'ús	**Executar scenario**
Actors	Usuari
Precondicions	1) Es necessita haver creat una story 2) La story ha d'estar oberta en el editor 3) La story té, com a mínim, un scenario creat

Flux bàsic:

Precondició:	El editor es troba en el mode per defecte
Actor	Sistema
1) L'usuari selecciona el scenario 2) L'usuari prem el botó "Run this Scenario"	
	3) El sistema demana els paràmetres que creu necessaris per poder executar l'escenari
4) L'usuari indica quin valor tenen els paràmetres, en cas que no sàpiga que ha de posar en algun dels camps ho consulta prement el botó d'informació. 5) L'usuari prem el botó "Run"	
	6) El sistema executa l'escenari 7) El sistema crea un fitxer HTML on es guarda el report, si ja existeix un report del mateix escenari/story prèviament l'esborra. 8) Per cada step que s'executa el sistema guarda, en el fitxer HTML, el step i si ha passat satisfactòriament o no 9) El sistema mostra el report i el va actualitzant a mesura que es va executant l'escenari
Postcondició:	L'escenari s'ha executat

Cas d'ús	**Veure report**
Actors	Usuari
Precondicions	S'ha executat com a mínim una story/escenari

Flux bàsic:

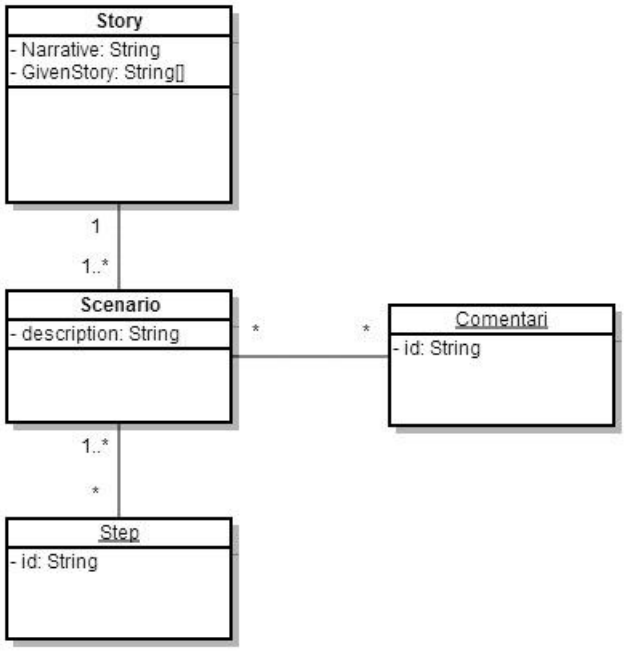
Precondició:	
Actor	Sistema
1) L'usuari tria l'opció "Open Reports" del menú "File"	
	2) El sistema mostra per pantalla el resum de tots els reports
Postcondició:	Apareix un report per pantalla

Cas d'ús	**Utilitzar story telling en la web**
Actors	Sistema
Precondicions	1) Es necessita haver creat una story 2) La story ha d'estar oberta en el editor 3) La story té, com a mínim, un scenario creat

Flux bàsic:

Precondició:	L'editor es troba en el mode per defecte
Actor	Sistema
1) L'usuari selecciona la posició on voldrà que s'afegeixin els steps que proposi la funcionalitat dins la story que té oberta 2) L'usuari tria l'opció "Story Telling" del menú "Product"	
	3) El sistema obra el story telling
4) L'usuari indica la "URL" que vol obrir 5) L'usuari tria quin navegador desitja utilitzar 6) L'usuari prem el botó "Open Browser"	
	7) El sistema obra el navegador a la URL que l'usuari ha indicat
8) L'usuari navega per la web fins el punt on desitja començar a gravar 9) L'usuari prem el botó "Start Recording"	
	10) El sistema comença a guardar les interaccions que el usuari realitza a la web
11) L'usuari interacciona amb al web 12) L'usuari prem el botó "Create Story"	
	13) El sistema mostra per pantalla steps agrupats per blocs (cada bloc correspon a una sola interacció de l'usuari)
14) L'usuari prem el botó "Add" d'un step de la secció, si vol continuar en aquella secció. Prem el botó "Add/Next" si vol afegir el step i passar a la següent secció". O el boto "Next" si vol passar a la següent secció.	
	15) El sistema comprova si el step conté algun paràmetre sense valor (el paràmetre contindrà el símbol $), si en té algun li demana a l'usuari que posi el valor
16) L'usuari introdueix el valor en el paràmetre	
	17) El sistema introdueix el step al scenario 18) Si s'ha premut el botó "Next" o "Add/Next" el sistema passa a la següent secció
19) Si encara hi han seccions es torna al punt 14	
Postcondició:	S'han afegit steps a l'escenari

DIAGRAMA CLASSES D'ESPECIFICACIÓ



3.1.3 DISSENY

Per crear l'aplicació es va decidir seguir el patró Model-Vista-Controlador (MVC) que es tracta d'un patró que separa el model de dades, la interfície d'usuari i la lògica de control. D'aquesta manera si es vulgues realitzar algun canvi en el futur només d'una de les tres capes resultarà més fàcil realitzar el canvi.

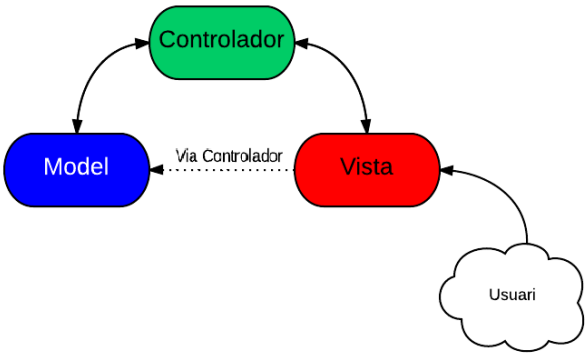
MODEL VISTA CONTROLADOR (MVC)

Aquest patró és un model pensat pel desenvolupament de programació orientada a objectes. El model està composat per tres components, el Model, la Vista i el Controlador tal i com el seu nom indica.

La capa del model és on es troba la lògica de les dades i no conté cap informació sobre la interfície de l'usuari.

La vista està formada per un conjunt de classes que representen els elements en la interfície d'usuari com poden ser botons, pantalles, texts, etc.

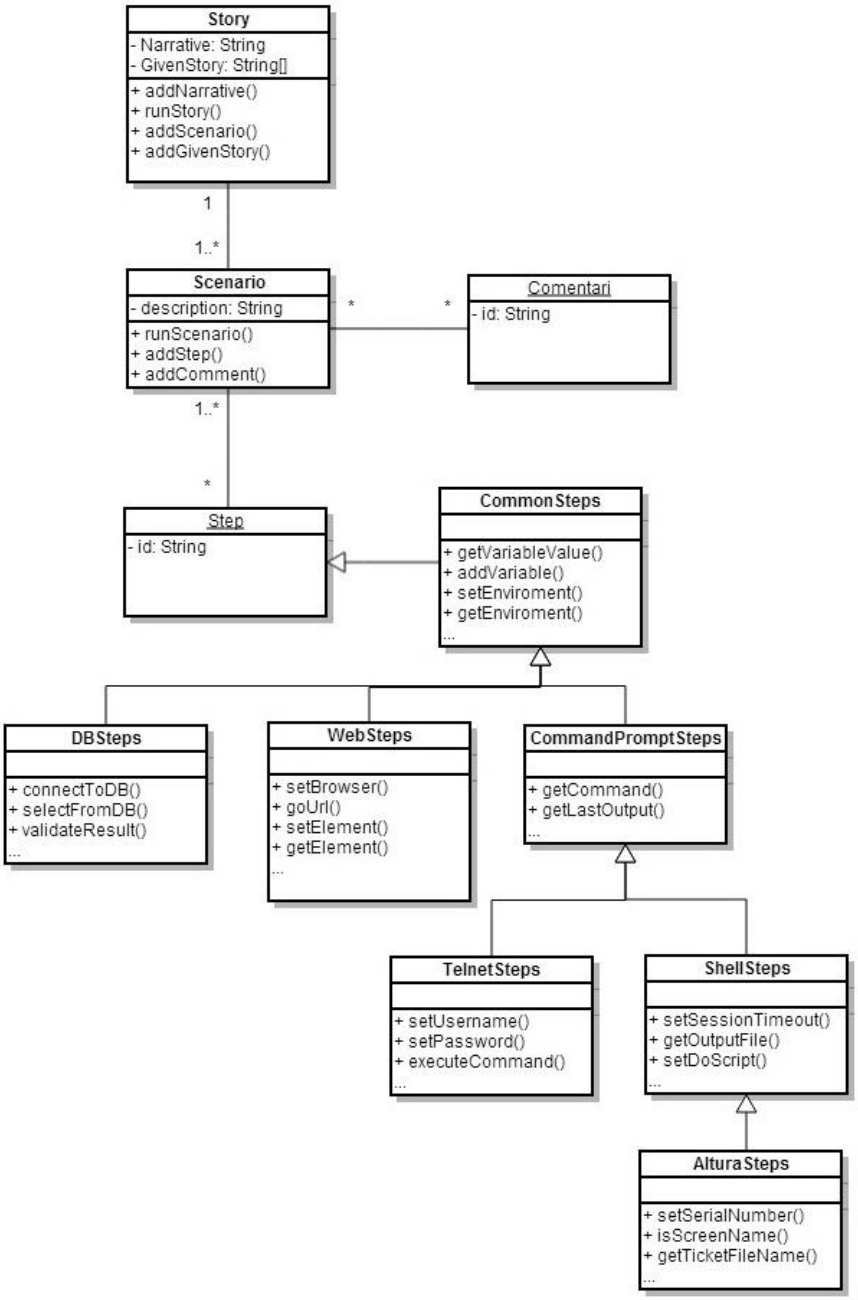
I finalment la capa del Controlador és on es troben totes les classes que es dediquen a connectar la capa del model amb la capa de la vista.



Al ser un patró de capes resulta molt senzill fer canvis gràcies a la seva modularitat i flexibilitat. Si es vulgues canviar totalment la vista només faria falta canviar les classes d'aquesta capa sense tocar res de la capa del Model ja que en cap moment la Vista depèn del Model o viceversa.

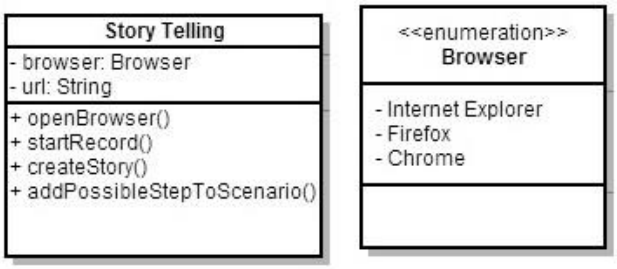
DIAGRAMA DE CLASES I FUNCIONS

El diagrama de classes de la part d'editar una story és el següent:



Com es pot observar la classe Story conté cuatre mètodes que permeten editar i executar una story, els mètodes per editar són "addNarrative", "addScenario", "addGivenStory" i el mètode per executar una story és el "runStory". La classe Scenario igual que la classe Story també conté mètodes per editar i per executar, aquests mètodes són "addStep" i "addComment" els quals et permeten afegir un step i afegir un comentari al escenari i, finalment el mètode "runScenario" que et permet executar l'escenari. Per últim totes les subclasses de Step tenen un mètode per cada clàusula, com que existeixen moltes, no s'han posat totes en el diagrama, si es desitja a l'annex estan totes les clàusules que existeixen per cada una de les classes del diagrama.

Pel que fa al diagrama de classes del story telling seria el següent:

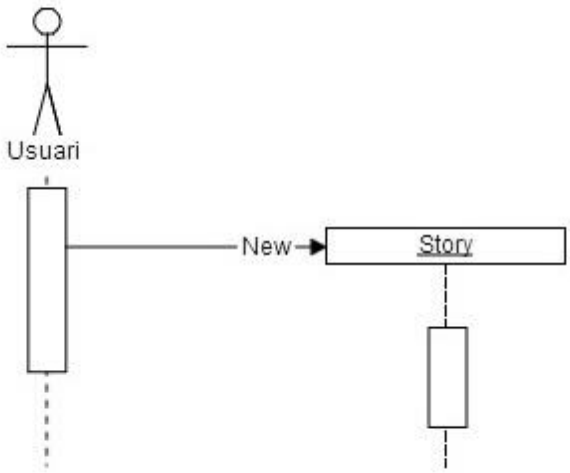


En el diagrama del story telling es pot observar que la classe "Story Telling" conté els mètodes "openBrowser", "startRecord" i "createStory". El mètode "openBrowser" permet obrir un navegador, el mètode "startRecord" s'utilitza per començar a gravar les interaccions que realitzarà l'usuari amb el navegador i, finalment, el mètode "createStory" serveix per processar la informació que s'ha obtingut i proposar els steps.

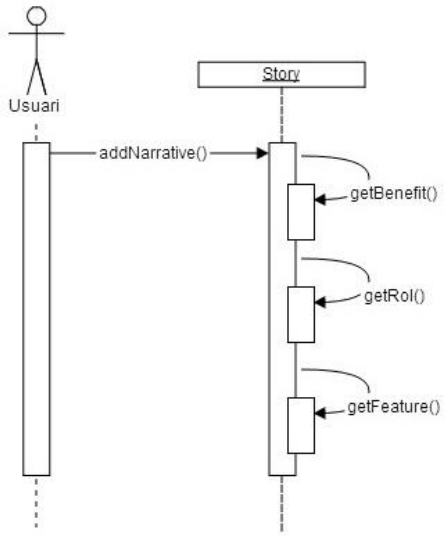
DIAGRAMES DE SEQÜÈNCIA

Els diagrames de seqüència d'aquesta iteració són els següents:

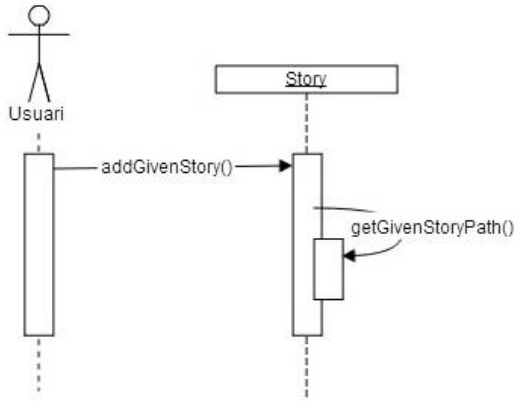
CREAR UNA STORY



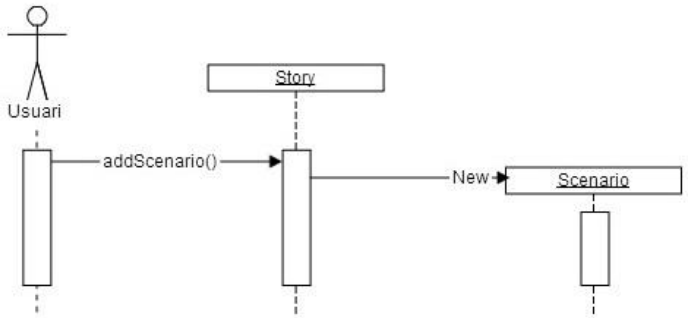
AFEGIR NARRATIVE



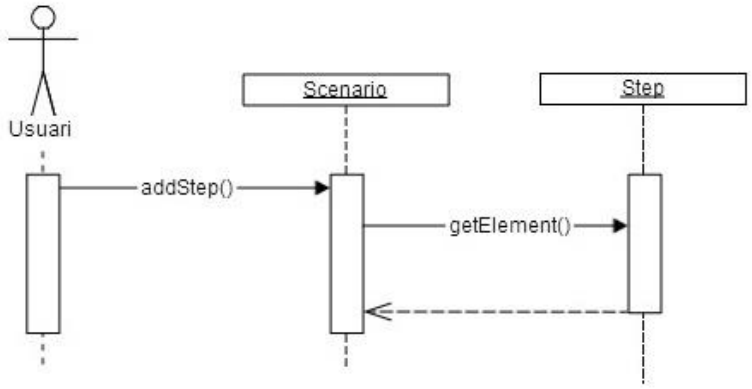
AGEFIR GIVEN STORY



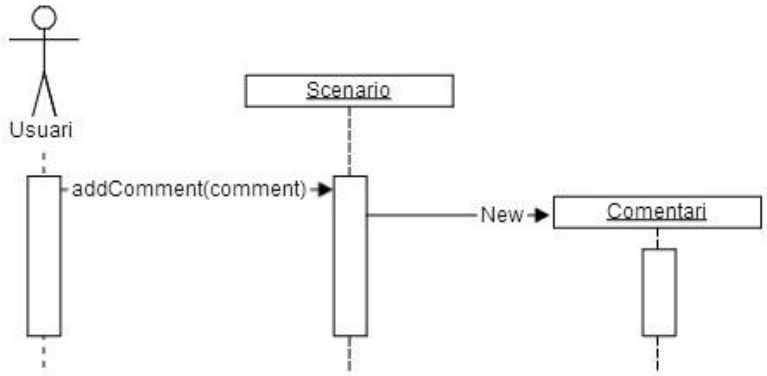
AFEGIR SCENARIO



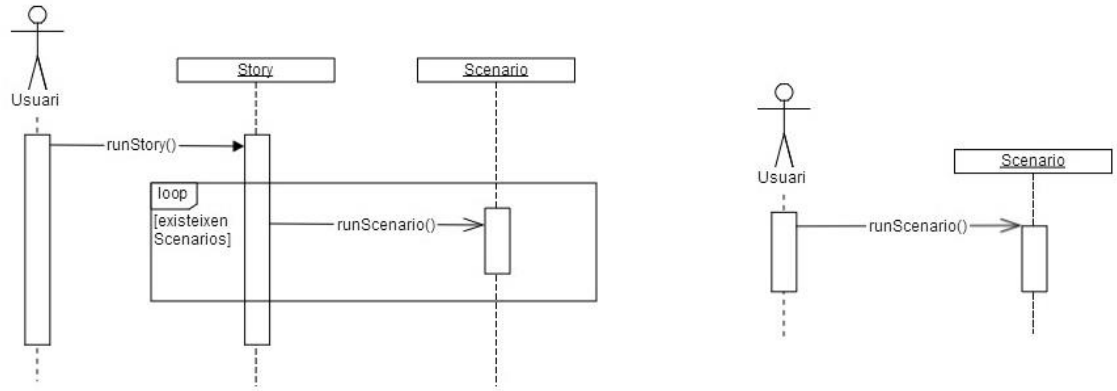
AFEGIR STEP



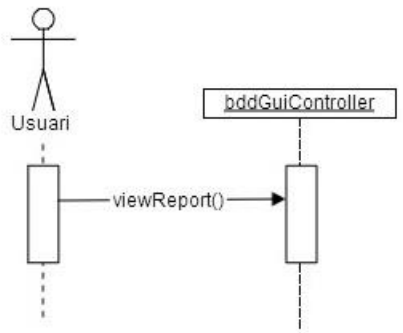
AFEGIR COMENTARI



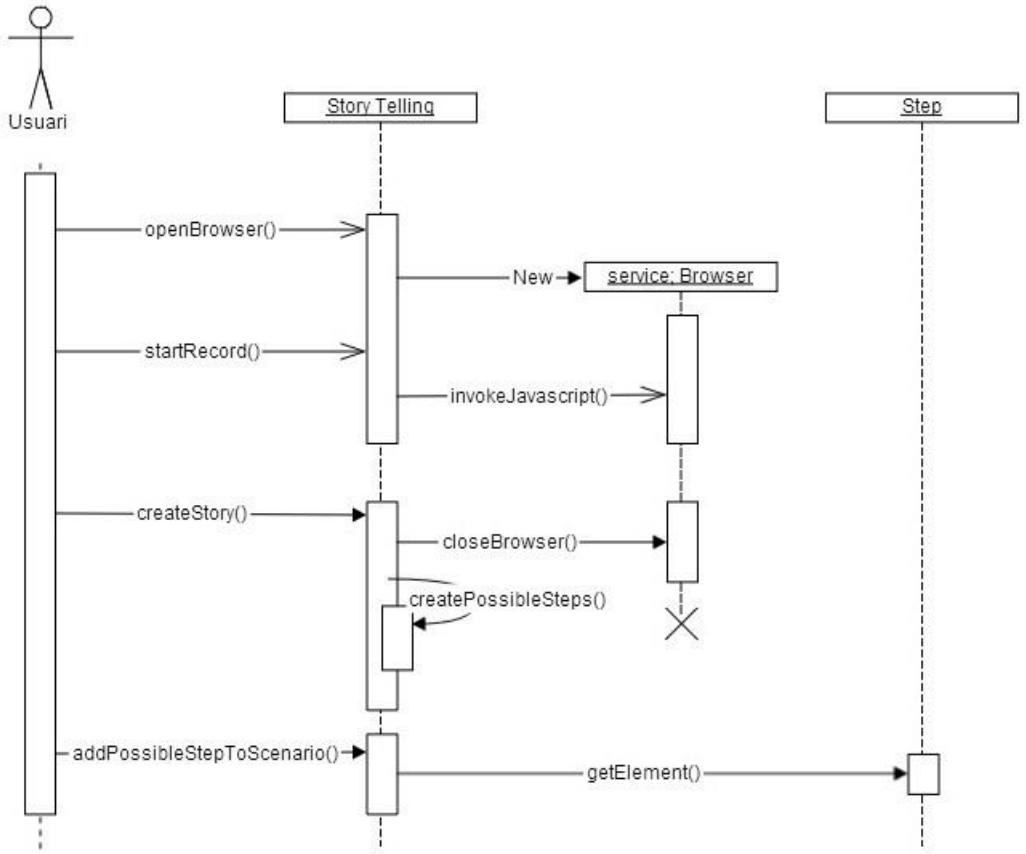
EXECUTAR STORY/SCENARIO



VEURE REPORT

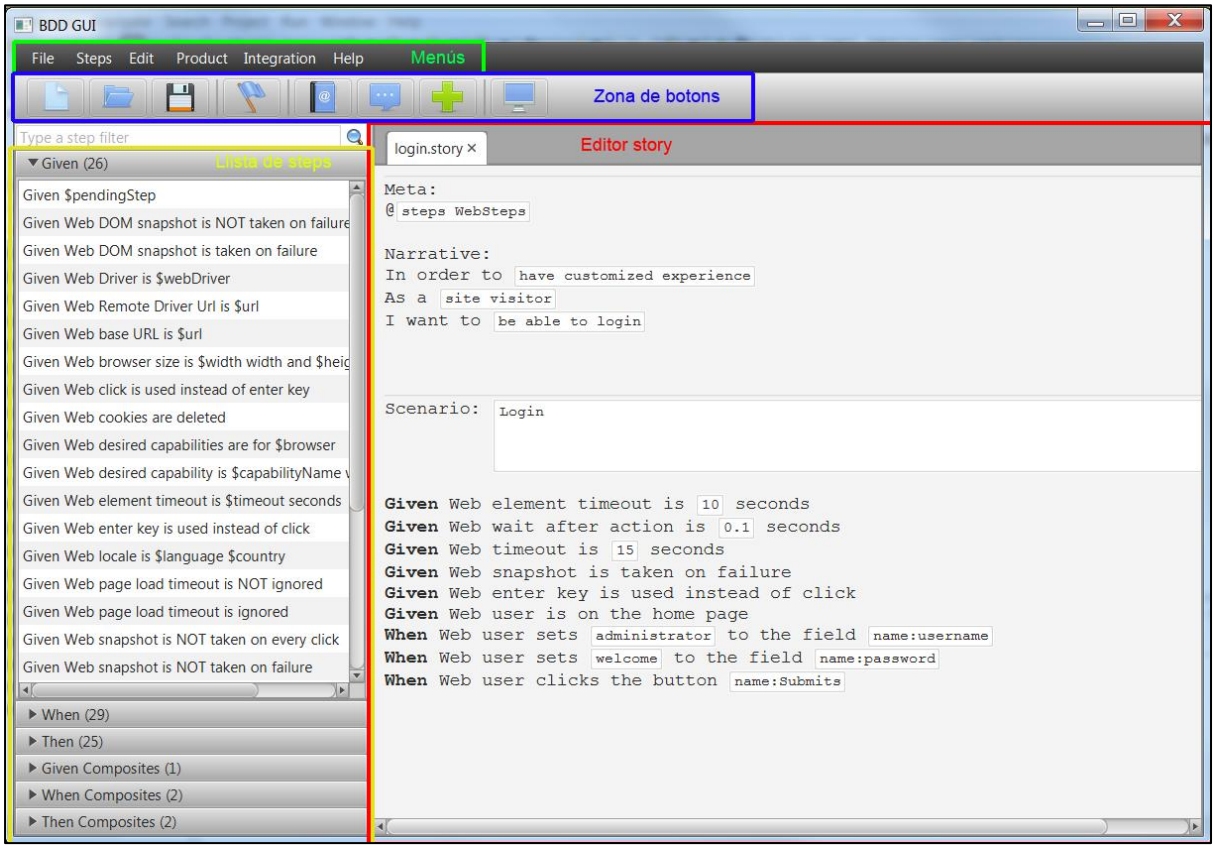


STORY TELLING



STORYBOARD

En aquest apartat es mostrà el storyboard de cada una de les funcionalitats. Però abans d'entrar en cada una d'elles crec convenient posar una imatge de l'aplicació indicant que és cada secció per poder identificar més clarament que és cada cosa



Menús: conjunt de menús que serveixen per accedit a les diferents funcionalitats de l'aplicació.

- **File:** menú on es troben totes les funcionalitats relacionades amb fitxers.
- **Steps:** menú per filtar els steps de la llista de steps
- **Edit:** menú on es troben totes les funcionalitats relacionades amb l'edició de stories i composites.
- **Product:** menú on es troben totes les funcionalitats que utilitzen aplicacions externes, com per exemple el simulador d'un Altura.
- **Integration:** menú per integrar coses externes a BDD, com per exemple obrir un ticket a Jira.
- **Help:** menú on els usuaris poden trobar material d'ajuda.

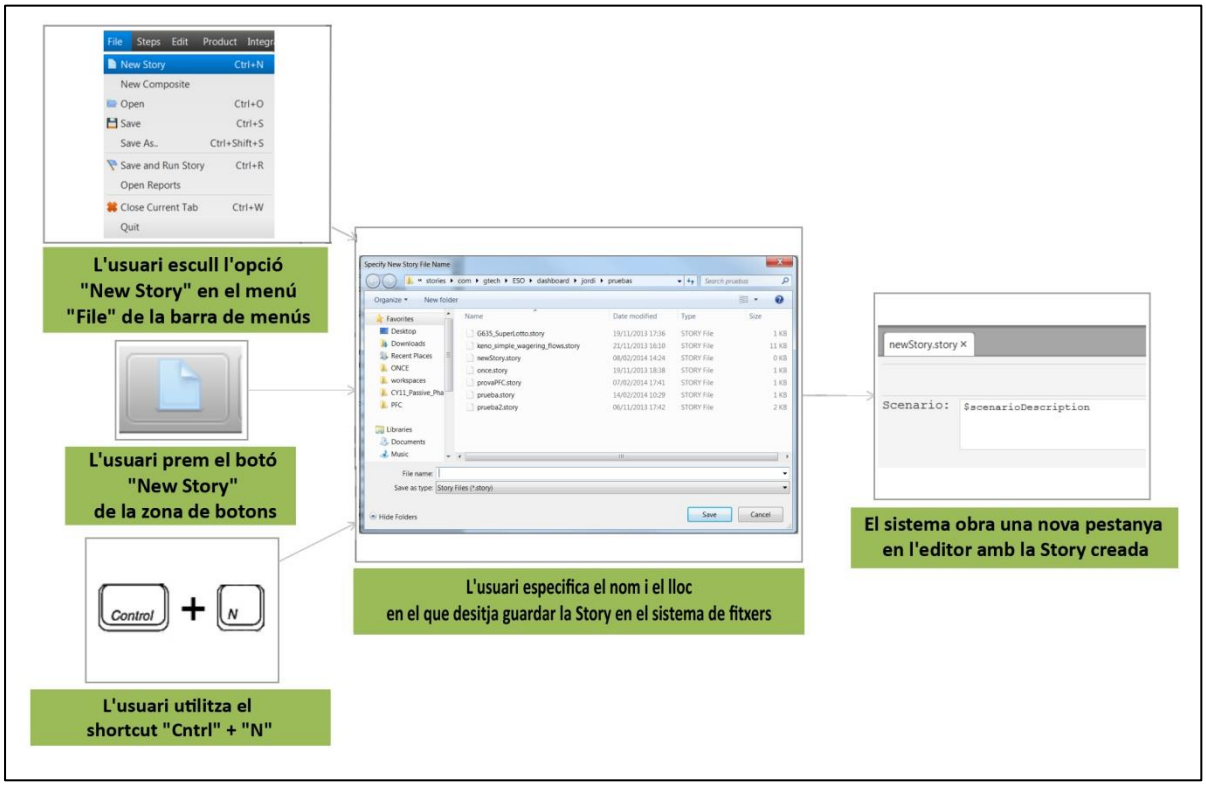
Zona de botons: en aquesta part de l'editor es troben els botons per realitzar les accions que més s'utilitzen en l'aplicació.

- **New Story:** botó per crear una nova story.
- **Open Story:** botó per obrir una story o un composite.
- **Save Story:** botó per guardar el fitxer que es tingui obert a la zona d'editar.
- **Run Story:** botó per executar una story.
- **Add Scenario:** botó per afegir un escenari.
- **Add comment:** botó per afegir un comentari.
- **Add Empty Step:** botó per afegir un step en blanc.
- **Toggle Editor:** botó per canviar el mode de l'editor entre el mode per defecte i el mode manual.

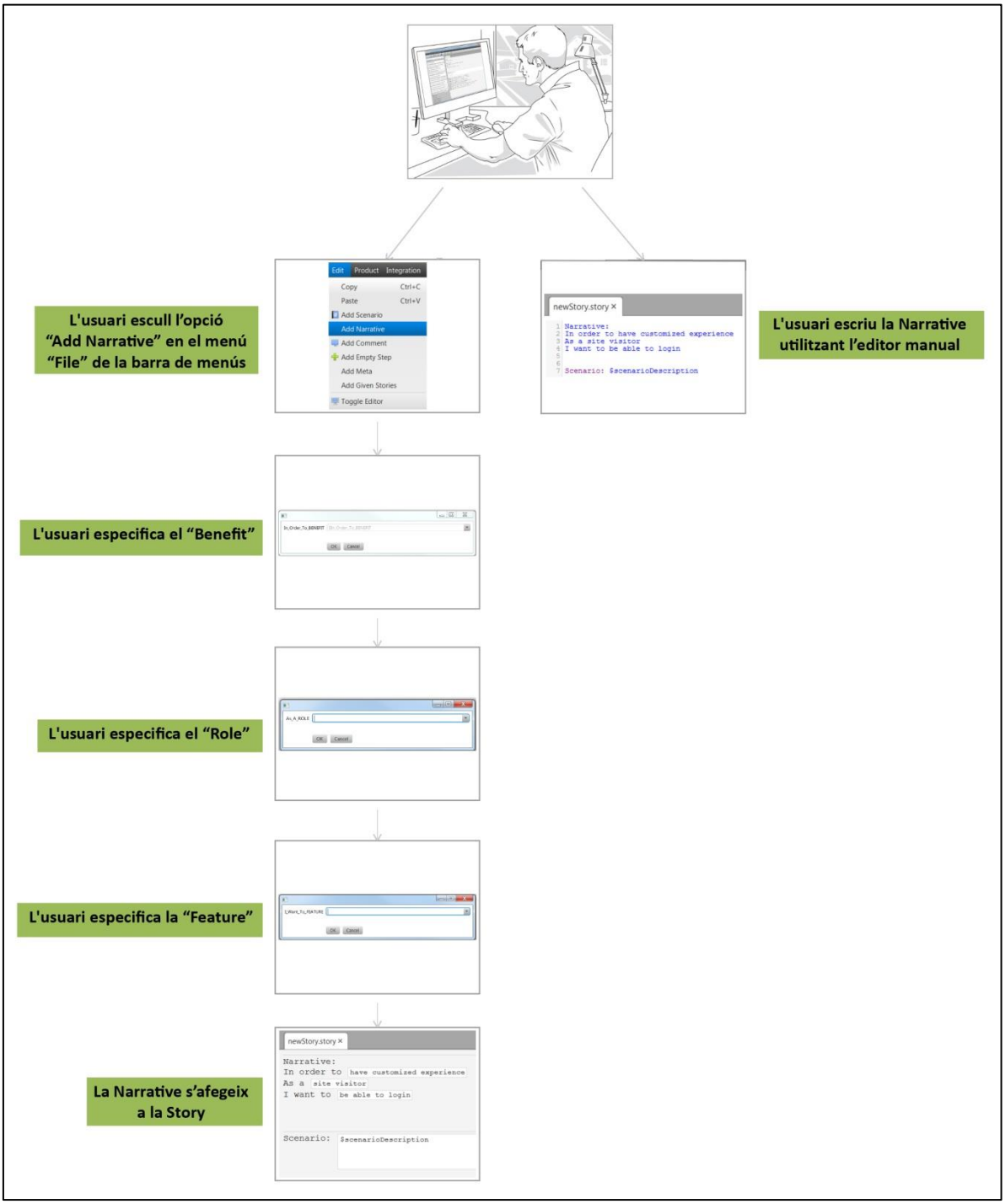
Llista de steps: en aquesta llista desplegable es poden trobar tots els steps del tipus que s'hagi seleccionat en el menú "Steps"

Editor story: zona on es poden editar les stories i els composites.

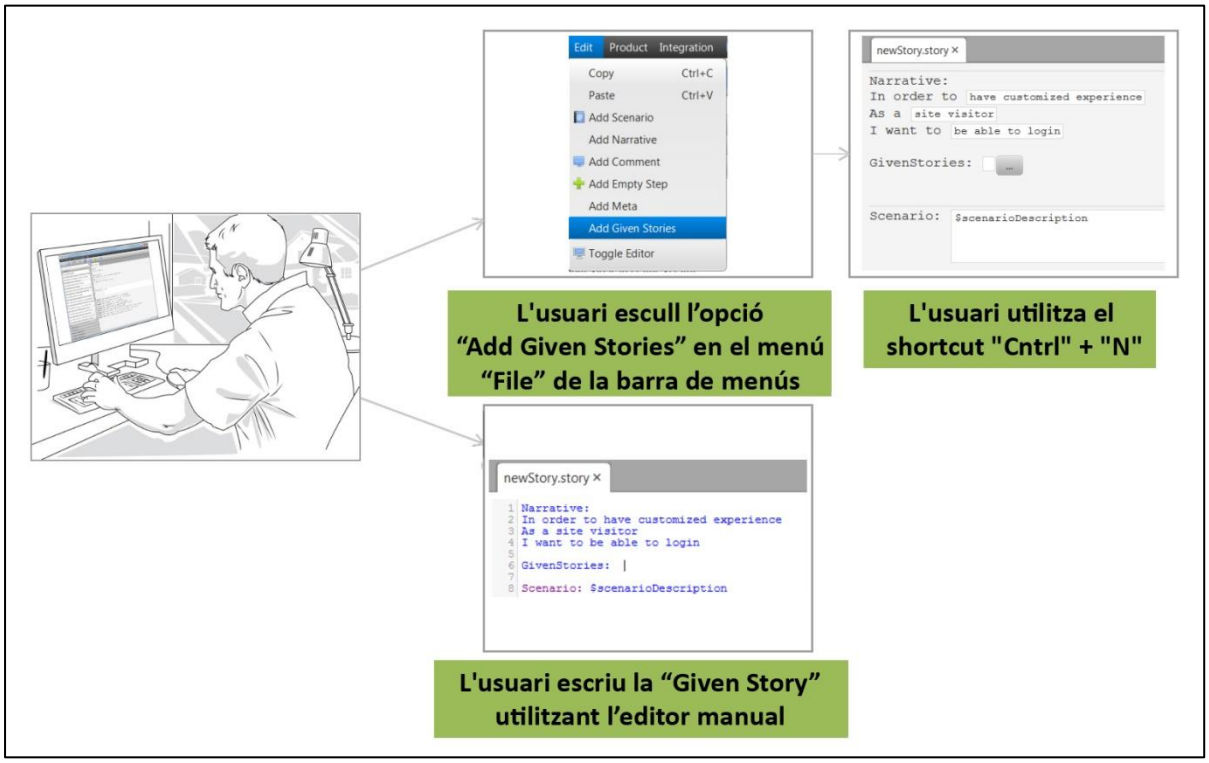
CREAR UNA NOVA STORY



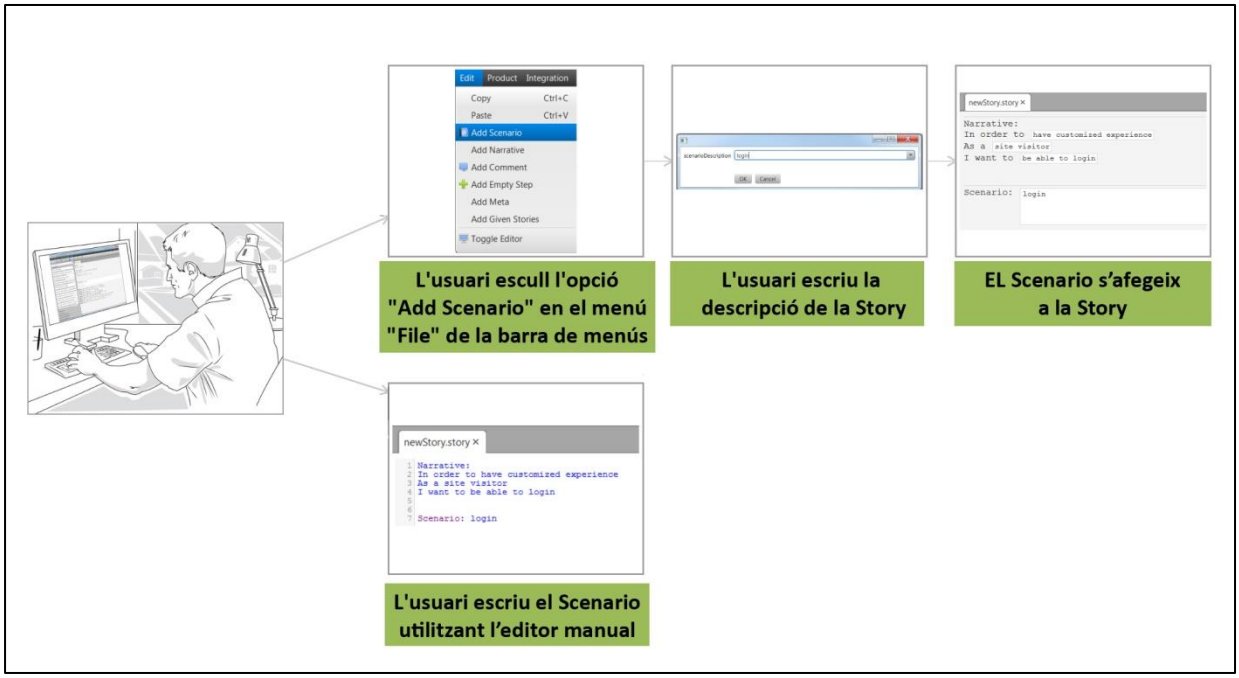
AFEGIR NARRATIVE



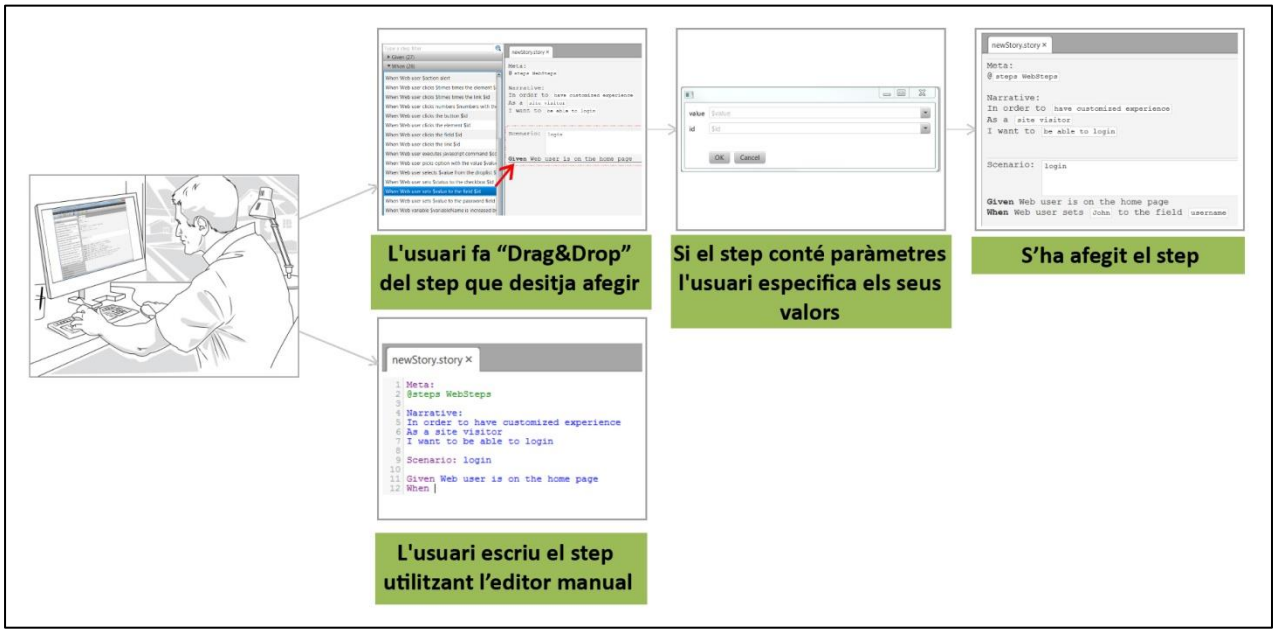
AFEGIR GIVEN STORY



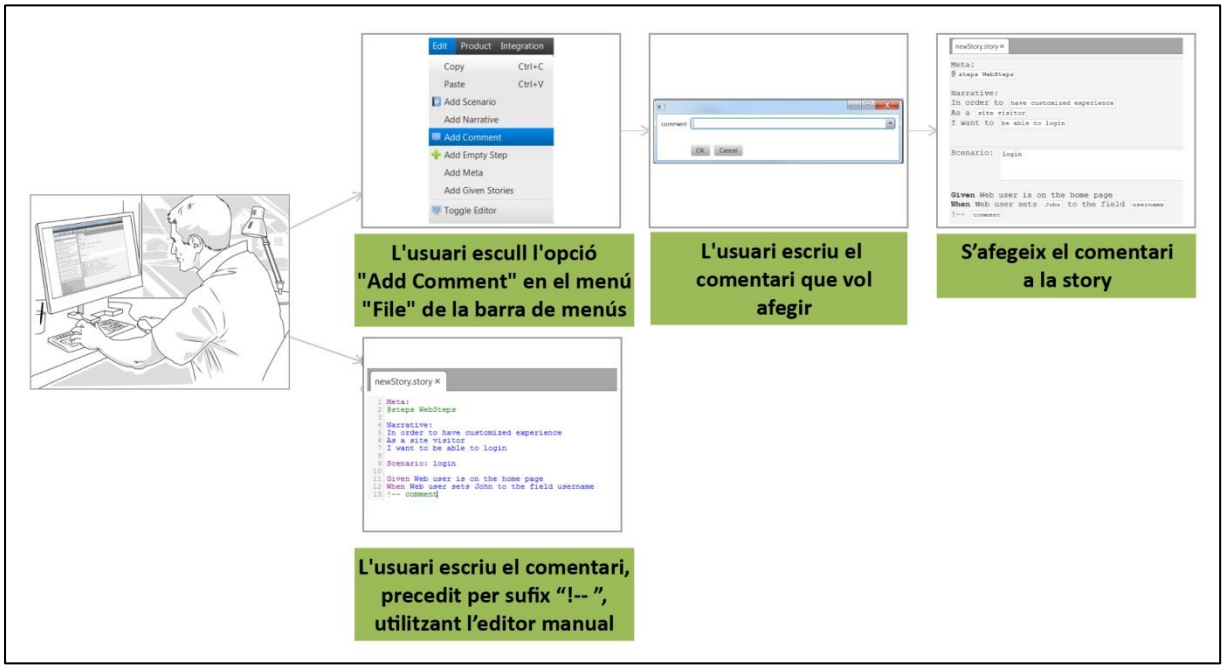
AFEGIR SCENARIO



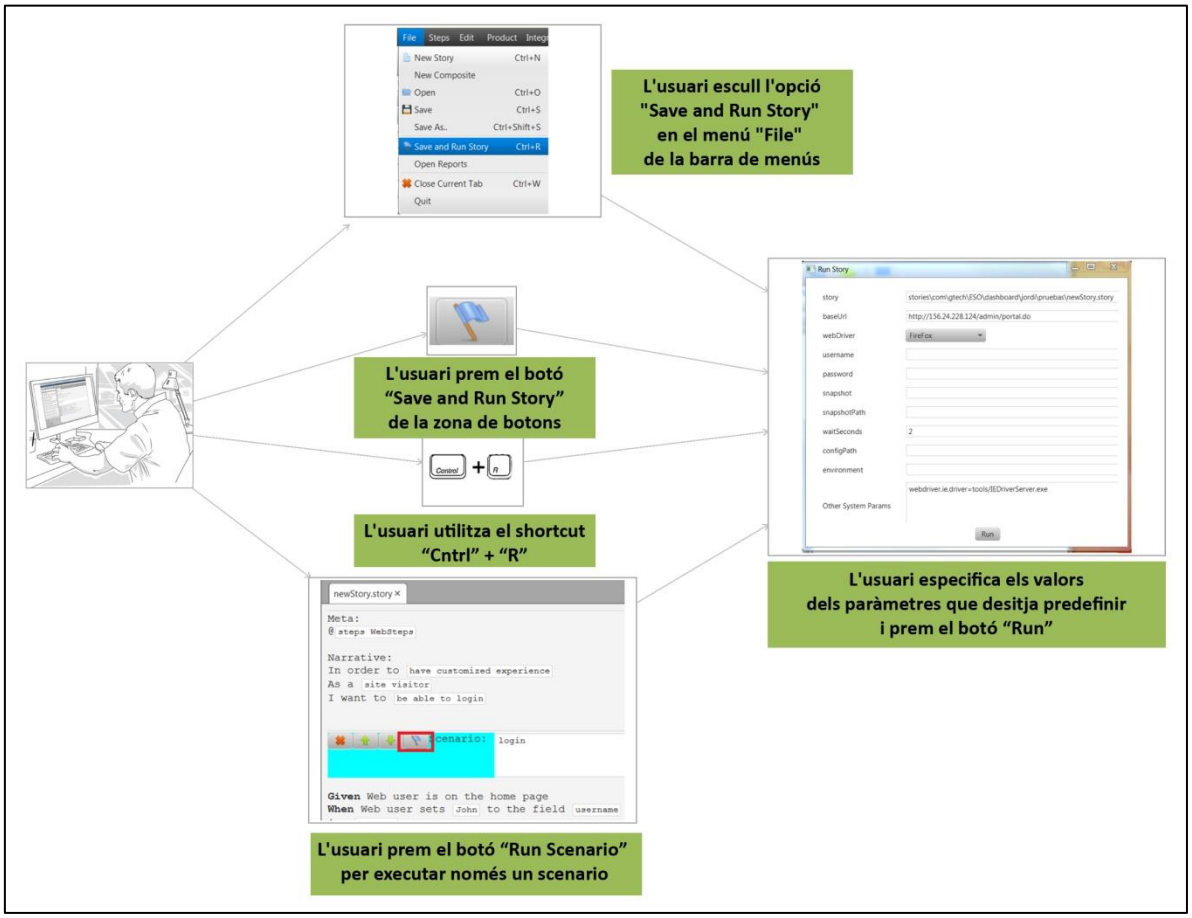
AFEGIR STEP



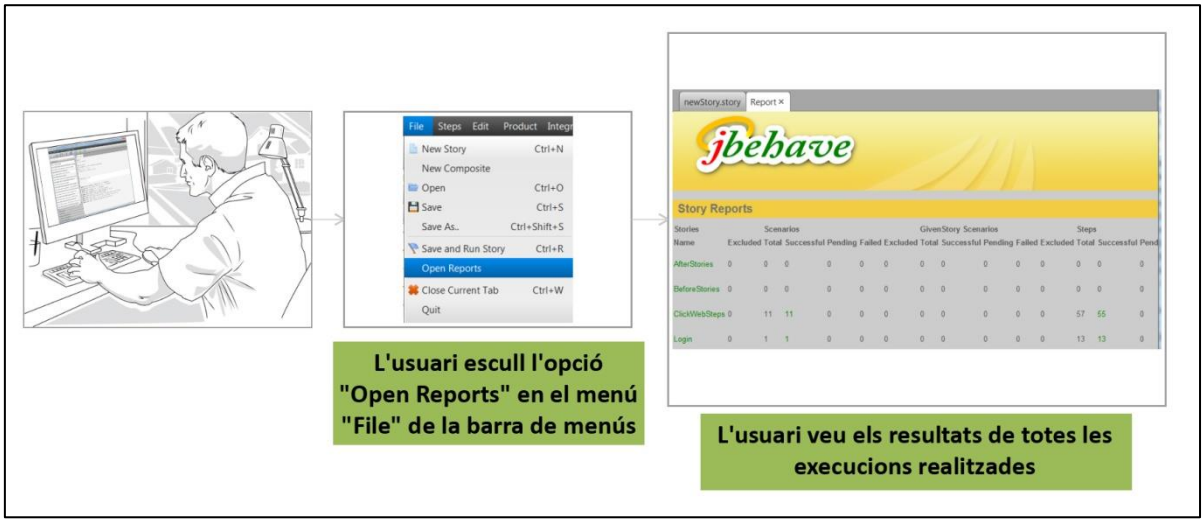
AFAGIR COMENTARI



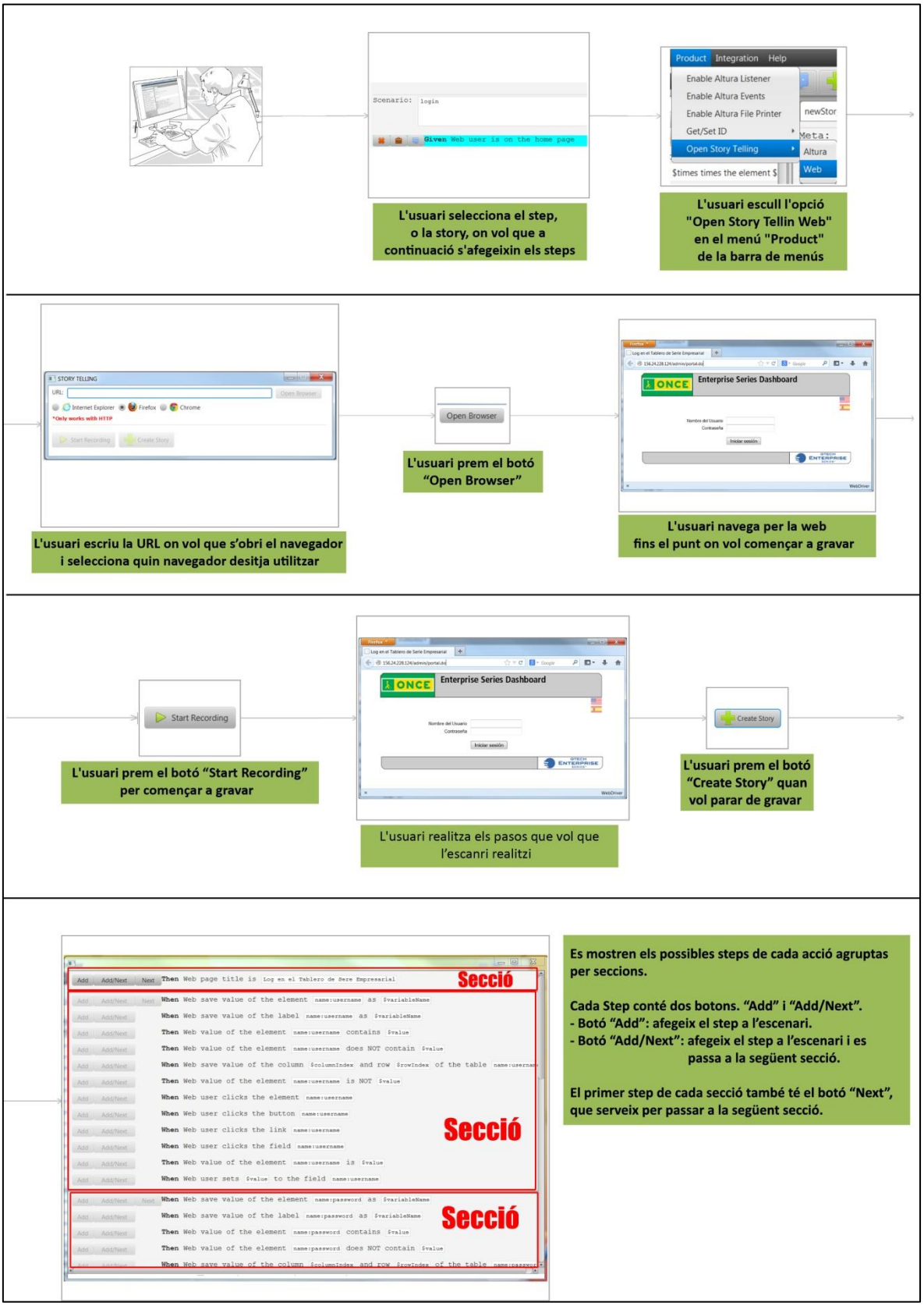
EXECUTAR STORY/SCENARIO



VEURE REPORT



UTILITZAR STORY TELLING



3.2 - ITERACIÓ 2

3.2.1 – ANÀLISIS DE REQUISITS

Per poder reutilitzar escenaris introduint petites variacions es va decidir incorporar en aquesta iteració l'ús de les "Example Tables", així els usuaris no hauran de repetir escenaris que utilitzen els mateixos steps canviant els seus paràmetres, amb un únic escenari i una example table els serà suficient.

En aquesta iteració també s'incorporen els composites els quals s'ha mirat de facilitar-ne el seu ús, per això de cara a l'usuari haurà de realitzar el mateix que si vol crear escenaris. Per crear un composite, l'usuari només haurà d'afegir el step "When $pendingStep" (tant pot ser When com Given o Then) a la posició en la que vol que s'utilitzi, d'aquesta manera s'afegirà el step i se li demanarà que introdueixi el test que vol que tingui el composite, un cop el step es trobi a la story l'usuari haurà de sel·leccionar-l'ho i prémer el boto amb el nom "Convert this step to composite", d'aquesta manera se li obrirà l'editor pel composite en el que l'usuari haurà d'escriure una descripció del què realitza aquell composite, de quin tipus és (Web, Altura, etc) i afegir aquells steps que vol que tingui. Un cop realitzat aquests passos quan l'usuari guardi el composite aquest apareixerà a la llista de steps com si sigues un step com qualsevol altre. També hi haurà la possibilitat de crear el composite des de la barra de menús.

Per exemple si és volgués crear un composite per loguejar-se es podria fer de la següent manera:

L'usuari agafaria el step "When $pendingStep" escriuria "user logs in", i així un cop estigui el composite creat a la llista de steps apareixerà com a "When user logs in", al prémer el botó "Convert this step to composite" com a descripció es podria posar "The user logs on the webside", com a tipus sel·leccionaria "WebSteps" i així quan es filtri la llista de steps, aquest només apareixerà a la llista de WebSteps o a les classes que estenen WebSteps, i com a steps posaria els següents:

```
Steps:
When Web user sets administrator to the field name:username
When Web user sets welcome to the field name:password
When Web user clicks the button name:Submit
Then Web wait for 5 seconds
```

En molts casos, els usuaris voldran utilitzar variables, per això s'han afegit els steps necessaris per poder guardar i consultar valors en variables. En el cas web s'han afegit els steps "When web save value of the element $id as $variableName" i "When web save value of the Date fields with the ID's $dayId $monthId $yearId as $dateVariable" i com a step comú per qualsevol tipus de step que s'estigui utilitzant "Then variable $variableName has value $variableValue". A més a més per facilitar l'ús de variables en qualsevol step, es pot utilitzar una variable escrivint el seu nom precedit pel símbol @, d'aquesta manera quan s'executi el step, aquest abans de realitzar l'acció/condició/comprovació es substituirà pel

valor que tingui la variable. Per exemple si en una variable anomenada "price" tinguéssim el valor "2" i volguéssim comprovar que un producte d'una web té el preu que esta guardat a la variable seria tant senzill com escriure el següent step:

```
Then web value of the element productPrice is @price
```

O si es vulgues guardar el valor del producte es podria fer de la següent manera:

```
When web save value of the element productPrice as price
```

Pel que fa a les variables de tipus data, al guardar-se tant el dia, el mes i l'any a la variable per accedir-hi s'ha d'escriure el símbol "@" seguit del nom de la variable mes ".date", ".month" o ".year" si es vol consultar el dia, el més o l'any.

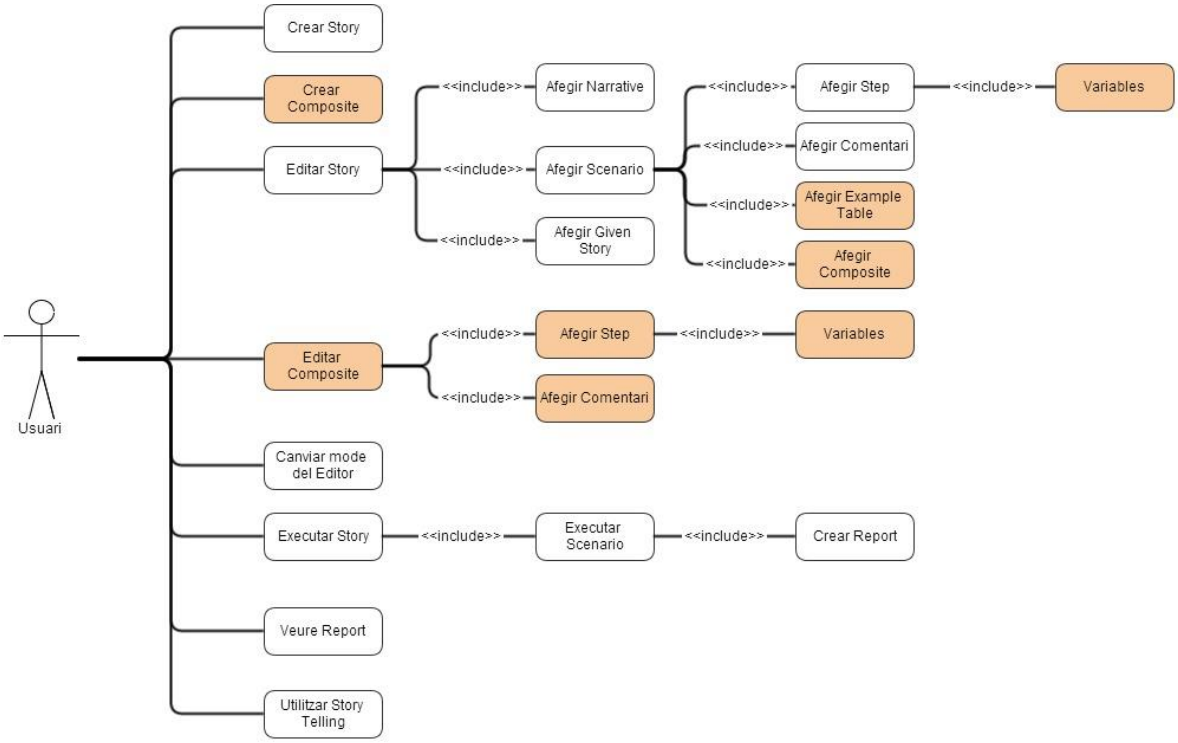
Per altre banda, en molts casos, un escenari pot utilitzar un mateix step varies vegades, per tal d'agilitzar el procés d'afegir steps, si el que es vol és afegir un step que serà igual que un d'existent a la story es podrà utilitzar la funcionalitat de copy&paste.

Finalment per tal de fer més còmode la interpretació del report, els steps que passin satisfactòriament es mostraran de color verd, en canvi aquells que hagin fallat és mostraran de color vermell, i finalment aquells que en el report apareguin com a "Pending" es mostraran de color blau.

3.2.2 ESPECIFICACIÓ

En aquest apartat es detalla l'especificació de les funcionalitats que es van afegir durant la segona iteració

CASOS D'ÚS



Cas d'ús	**Crear Composite**
Actors	Usuari
Precondicions	
Flux bàsic:	

Precondició:	
Actor	Sistema
1) L'usuari escull l'opció "New Composite" en el menú "File"	
	2) El sistema demana on vol guardar el composite i quin nom vol posar en el fitxer
3) L'usuari especifica el lloc on guardar el composite i amb quin nom	
	4) El sistema guarda el composite en el lloc especificat amb el nom que ha posat l'usuari
Postcondició:	Es crea un composite

Cas d'ús	**Afegir composite a un scenario**
Actors	Usuari
Precondicions	1) Es necessita haver creat una story 2) La story ha d'estar oberta en el editor 3) La story té, com a mínim, un scenario creat

Flux bàsic:

Precondició:	L'editor es troba en el mode per defecte
Actor	Sistema
1) L'usuari escull el composite de la llista desplegable de composites 2) L'usuari fa drag&drop a la zona de l'escenari en la que desitja afegir el composite	3) El sistema afegeix el composite a la zona de l'escenari que el usuari desitja
Postcondició:	El scenario conté el composite

Fluxes alternatius:

Flux Alternatiu: afegir composite a un scenari manualment

Precondició:	L'editor es troba en el mode per manual
Actor	Sistema
1) L'usuari escriu el composite dins l'escenari	
Postcondició:	El scenario conté el composite

Cas d'ús	**Editar composite**
Actors	Usuari
Precondicions	Es necessita haver creat el composite

Flux bàsic:

Precondició:	El composite esta afegit a una story que es té oberta a l'editor
Actor	Sistema
1) L'usuari selecciona el composite 2) L'usuari prem el botó "Convert this step to composite" 4) L'usuari edita el composite	3) El sistema obra el composite en l'editor
Postcondició:	S'ha editat el composite

Fluxes alternatius:

Flux Alternatiu: editar composite sense tindre'l afegit en una story

Precondició:	
Actor	Sistema
1) L'usuari obre el composite prement el botó "Open Story" 3) L'usuari tria el composite en el sistema de fitxers 5) L'usuari edita el composite	2) El sistema demana quin arxiu vol obrir 4) El sistema obra el composite en l'editor
Postcondició:	S'ha editat el composite

Flux Alternatiu: editar composite sense tindre'l afegit en una story a través de la barra de menús

Precondició:	
Actor	Sistema
1) L'usuari obre el composite triant l'opció "Open" del menú "File" 3) L'usuari tria el composite en el sistema de fitxers 5) L'usuari edita el composite	2) El sistema demana quin arxiu vol obrir 4) El sistema obra el composite en l'editor
Postcondició:	S'ha editat el composite

Flux Alternatiu: editar composite sense tindre'l afegit en una story a través del shortcut "Cntrl"+"O"

Precondició:	
Actor	Sistema
1) L'usuari obre el composite prement el shortcut "Cntrl+O" 3) L'usuari tria el composite en el sistema de fitxers 5) L'usuari edita el composite	2) El sistema demana quin arxiu vol obrir 4) El sistema obra el composite en l'editor
Postcondició:	S'ha editat el composite

Cas d'ús	**Afegir step al composite**
Actors	Usuari
Precondicions	El composite ha d'estar obert en l'editor

Flux bàsic:

Precondició:	L'editor es troba en el mode per defecte
Actor	Sistema
1) L'usuari escull el step de la llista desplegable de steps 2) L'usuari fa drag&drop a la zona de steps en la que desitja afegir el step	3) El sistema afegeix el step a la zona que de steps que el usuari desitja
Postcondició:	S'ha afegit un step en el composite

Fluxes alternatius:

Flux Alternatiu: afegir step a un composite manualment

Precondició:	L'editor es troba en el mode per manual
Actor	Sistema
1) L'usuari escriu el step dins el composite en el apartat steps	
Postcondició:	S'ha afegit un step en el composite

Cas d'ús	**Afegir comentari al composite**
Actors	Usuari
Precondicions	El composite ha d'estar obert en l'editor

Flux bàsic:

Precondició:	L'editor es troba en el mode per defecte
Actor	Sistema
1) L'usuari selecciona el step on vol que després es posi el comentari 2) L'usuari escull l'opció "Add Comment" del menú "Edit" 3) L'usuari escriu el comentari que vol que aparegui 4) L'usuari prem el botó "Ok"	5) El sistema introdueix el comentari a la posició que l'usuari desitja
Postcondició:	S'ha afegit un comentari

Fluxes alternatius:

Flux Alternatiu: afegir comentari a un composite manualment

Precondició:	L'editor es troba en el mode manual
Actor	Sistema
1) L'usuari escriu el comentari dins el composite en el apartat steps	
Postcondició:	S'ha afegit un comentari

Cas d'ús	**Copiar step**
Actors	Usuari
Precondicions	Hi ha una story o un composite obert en l'editor

Flux bàsic:

Precondició:	El composite/story conté com a mínim un step
Actor	Sistema
1) L'usuari selecciona els step que desitja copiar 2) L'usuari prem el shortcut "Cntrl+C"	3) El sistema es guarda el step
Postcondició:	S'ha copiat el step

Fluxes alternatius:

Flux Alternatiu: copiar step mitjançant la barra de menús

Precondició:	El composite/story conté com a mínim un step
Actor	Sistema
1) L'usuari selecciona els step que desitja copiar 2) L'usuari escull l'opció "Copy" del menú "Edit"	3) El sistema es guarda el step
Postcondició:	S'ha copiat el step

Cas d'ús	**Enganxar step**
Actors	Usuari
Precondicions	1) Existeix una story o un composite obert en l'editor 2) S'ha copiat un step prèviament

Flux bàsic:

Precondició:	
Actor	Sistema
1) L'usuari selecciona els step que desitja posar a continuació el step a enganxar 2) L'usuari prem el shortcut "Cntrl+V"	3) El sistema insereix el step
Postcondició:	S'ha insertat un step

Fluxes alternatius:

Flux Alternatiu: enganxar step mitjançant la barra de menús

Precondició:	
Actor	Sistema
1) L'usuari selecciona els step que desitja posar a continuació el step a enganxar 2) L'usuari escull l'opció "Paste" del menú "Edit"	3) El sistema insereix el step
Postcondició:	S'ha insertat un step

Cas d'ús	**Afegir Example Table**
Actors	Usuari
Precondicions	Existeix una story oberta en l'editor

Flux bàsic:

Precondició:	L'editor es troba en el mode per defecte
Actor	Sistema
1) L'usuari selecciona el camp de text que vol que utilitzi una cel·la de la example table que vol afegir 2) L'usuari escriu entre els símbols "<" i ">" el nom que vol que tingui la columna de la cel·la que vol utilitzar 3) L'usuari prem el botó "ENTER"	4) El sistema introdueix la example table a l'escenari
Postcondició:	S'ha afegit una example table a l'escenari

Fluxes alternatius:

Flux Alternatiu: afegir example table manualment

Precondició:	L'editor es troba en el mode manual
Actor	Sistema
1) L'usuari escriu l'example table a l'escenari	
Postcondició:	S'ha afegit una example table a l'escenari

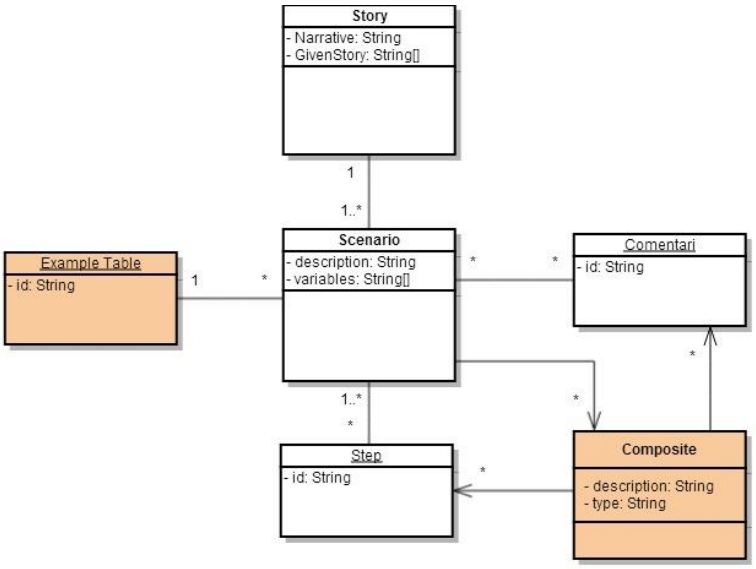
Cas d'ús	**Guardar variable en una story**
Actors	Usuari
Precondicions	Existeix una story o un composite obert en l'editor
Flux bàsic:	

Precondició:	
Actor	Sistema
1) L'usuari afegeix el step "Given variable $variableName is $variable" on $variableName posa el nom que vulgui tindre la variable i a $variable posa el valor que vol que tingui	
Postcondició:	La story conté un step per guardar una variable

Cas d'ús	**Utilitzar variable en una story**
Actors	Usuari
Precondicions	1) Existeix una story o un composite obert en l'editor 2) El sistema té la variable guardada
Flux bàsic:	

Precondició:	
Actor	Sistema
1) L'usuari escriu "@"+ nom de la variable per indicar a la story o el composite que vol que aquell camp de text sigui substituït pel valor que té la variable	
Postcondició:	La story utilitza una variable en un step

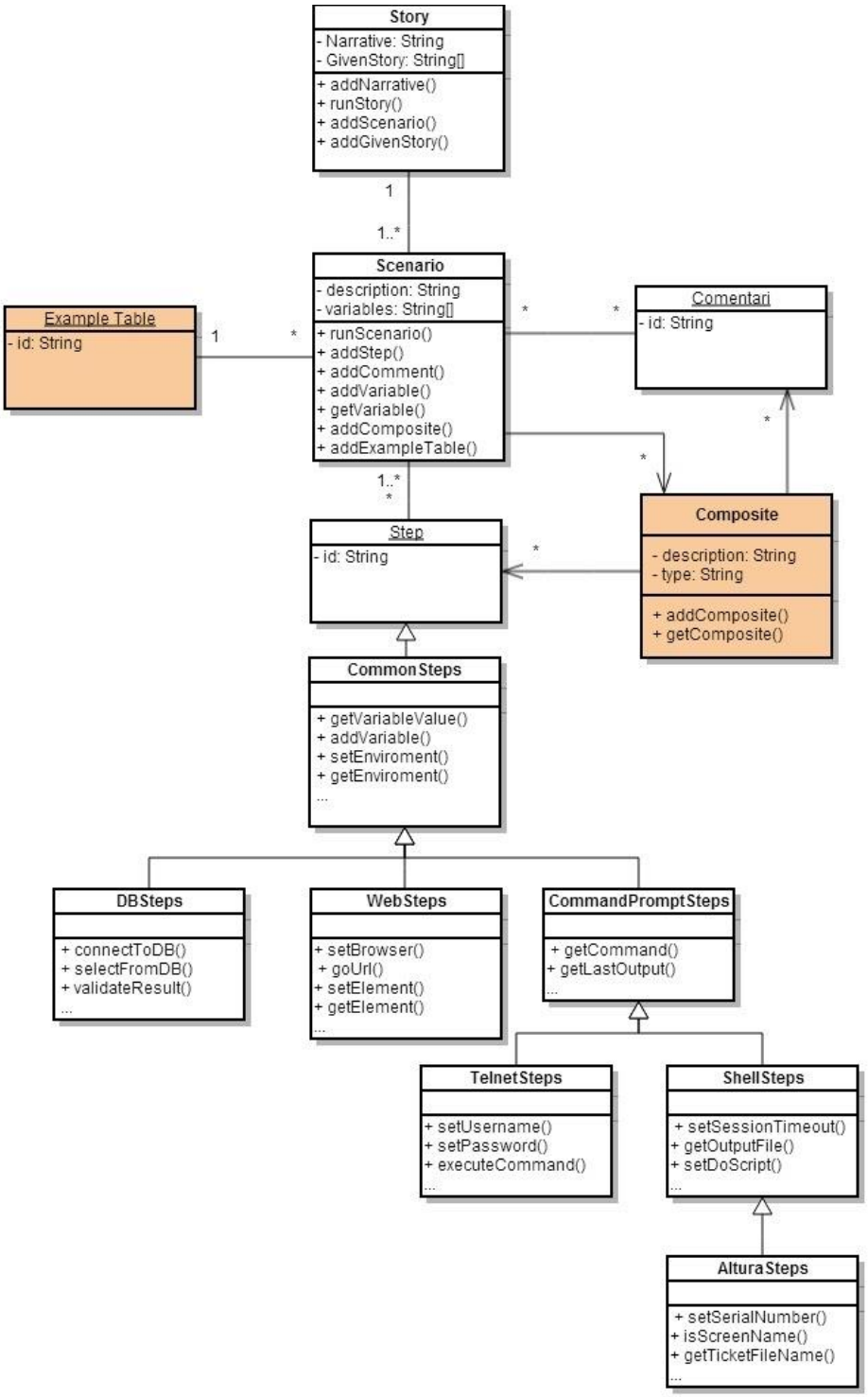
DIAGRAMA CLASSES D'ESPECIFICACIÓ



3.2.3 DISSENY

DIAGRAMA DE CLASES I FUNCIONS

El diagrama de classes afegint les "Example Tables" i els "Composites quedaria de la següent manera:

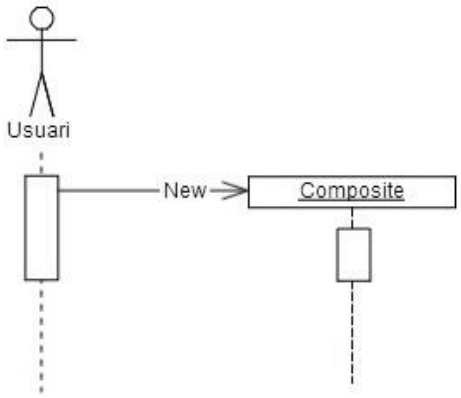


En el diagrama també es pot observar que s'han afegit els mètodes "addVariable" i "getVariable a la classe "Scenario", aquest mètodes permeten afegir i consultar el valor de variables.

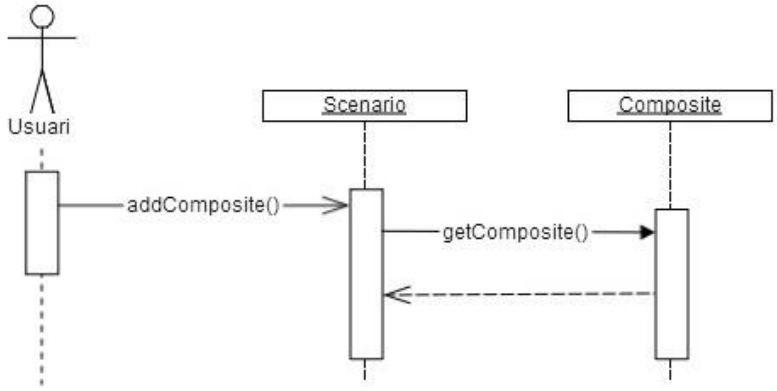
DIAGRAMES DE SEQÜÈNCIA

Els diagrames de seqüència d'aquesta iteració són els següents:

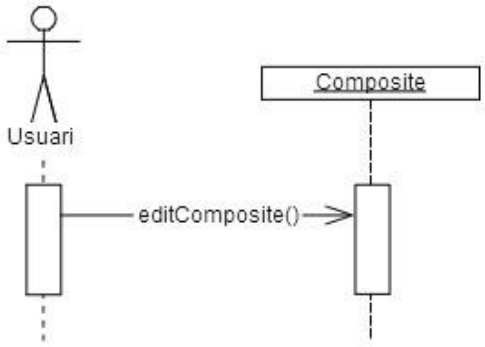
CREAR COMPOSITE



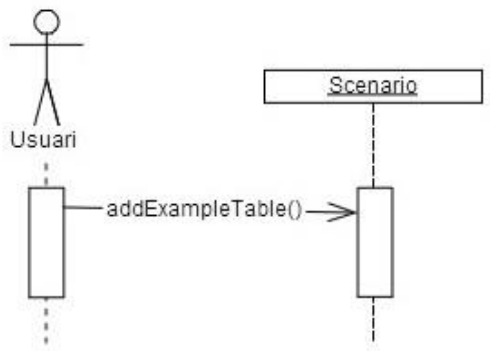
AFEGIR COMPOSITE A UN SCENARIO



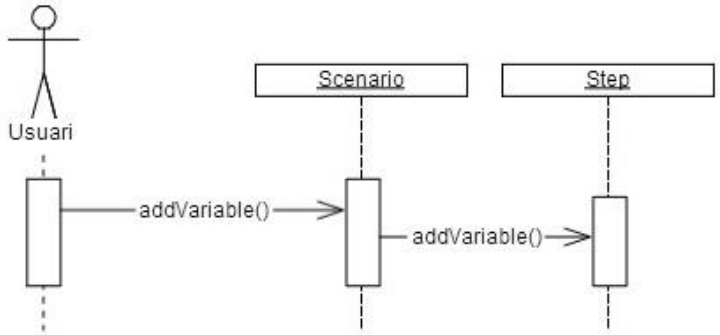
EDITAR COMPOSITE



AFEGIR UNA EXAMPLE TABLE

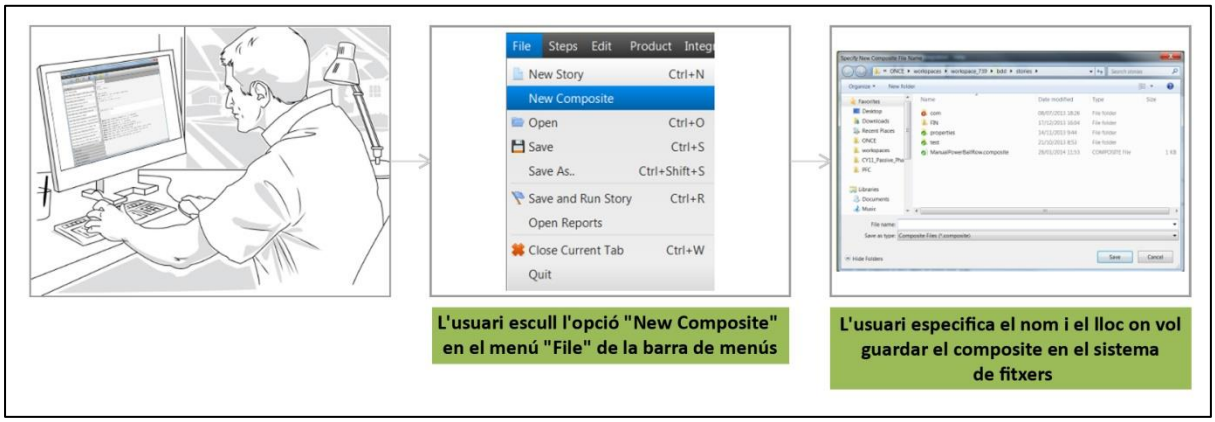


GUARDAR VARIABLE

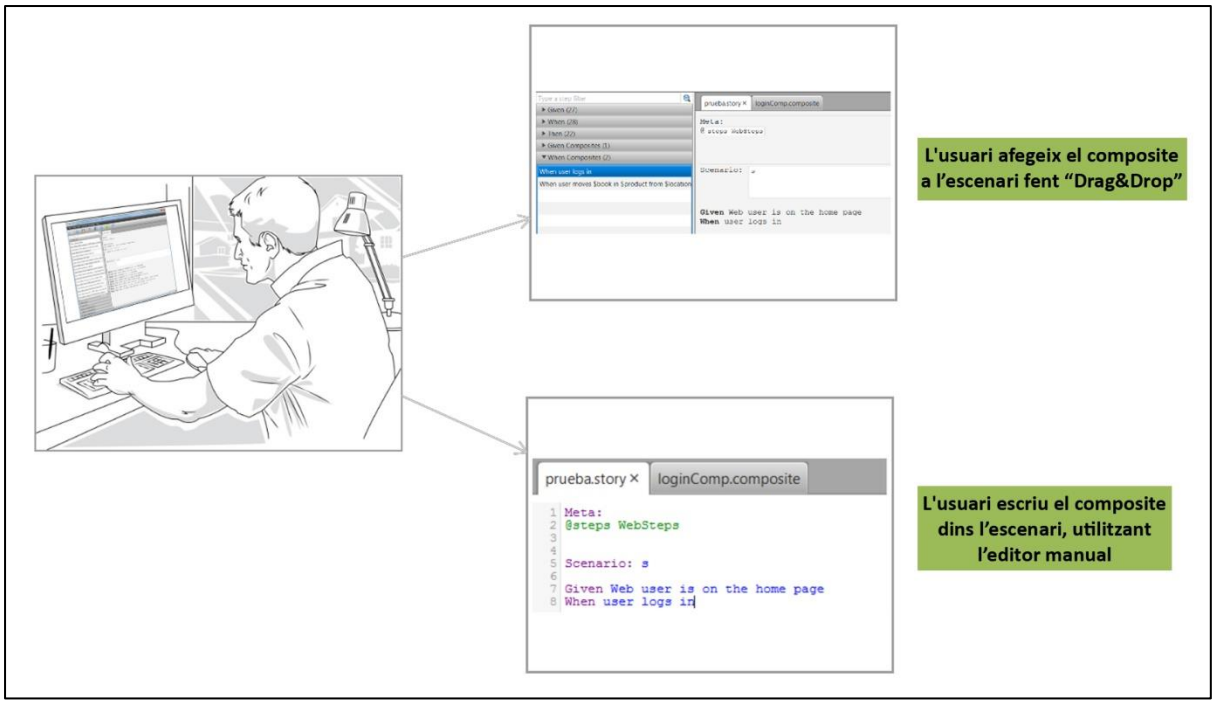


STORYBOARD

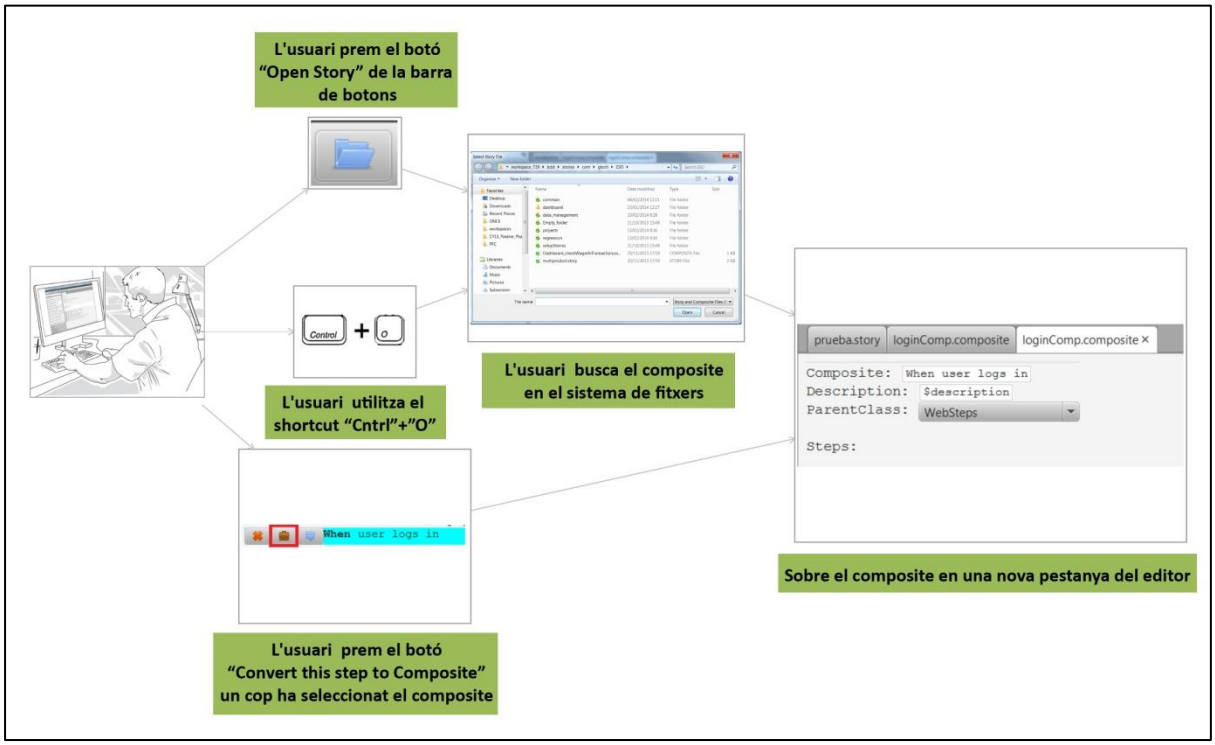
CREAR COMPOSITE



AFEGIR COMPOSITE A UN SCENARIO

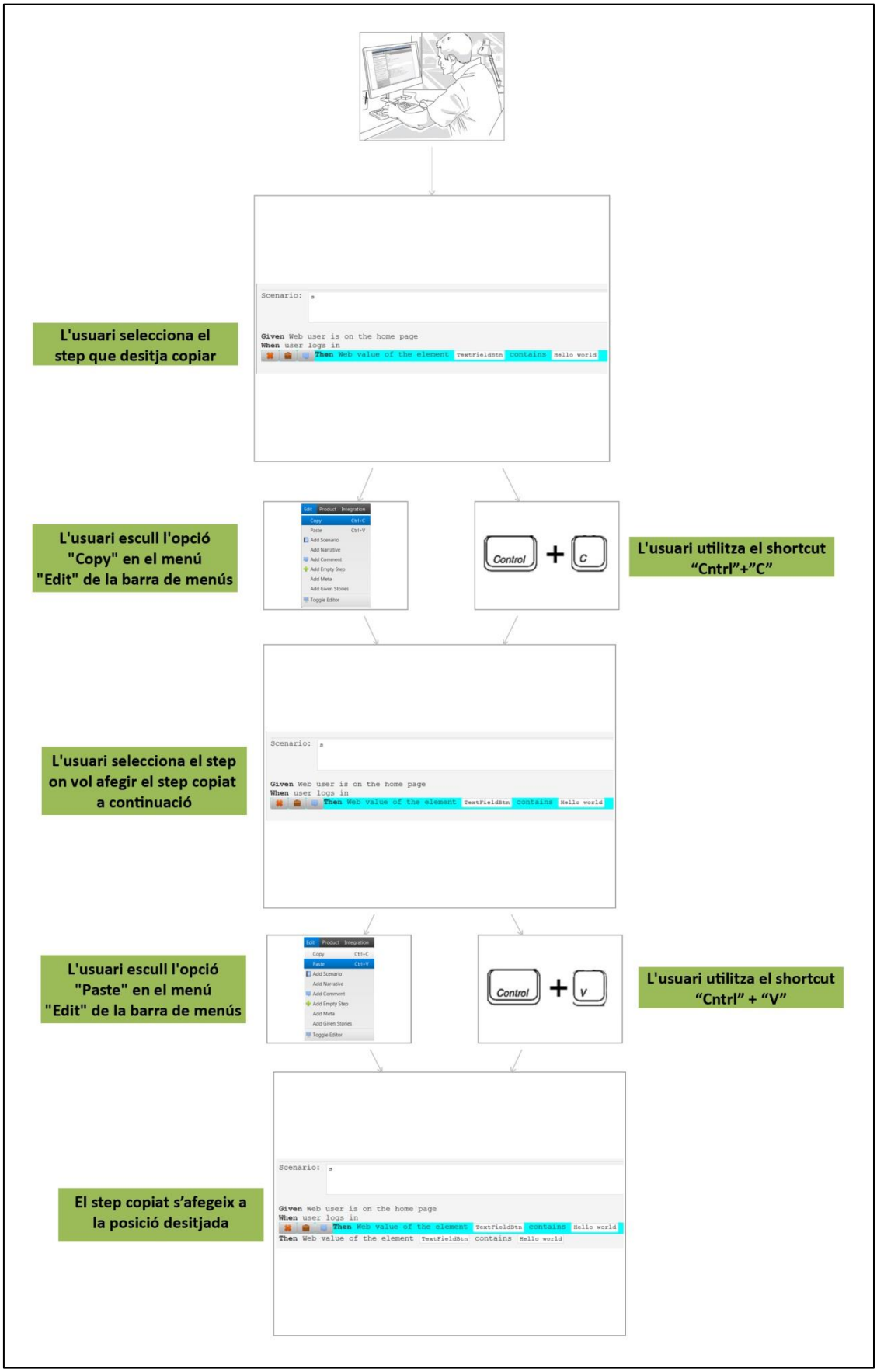


EDITAR COMPOSITE

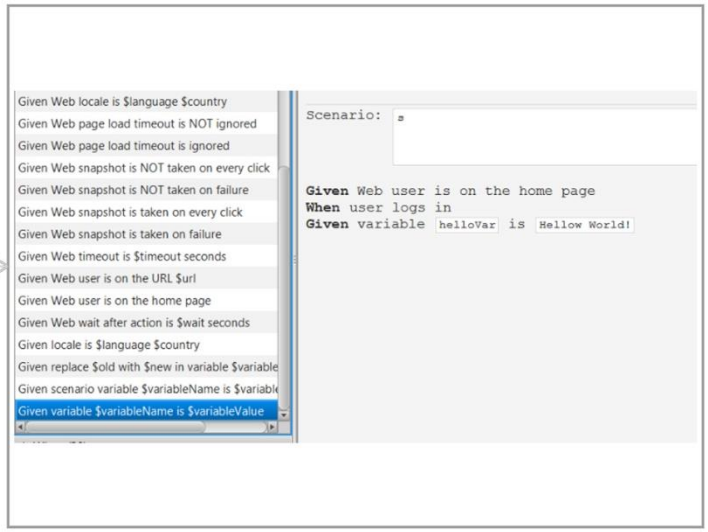


AFEGIR EXAMPLE TABLE

COPY&PASTE STEP



GUARDAR VARIABLE



L'usuari afegeix el step "Given variable $variableName is $variableValue"

UTILITZAR VARIABLE

L'usuari utilitza el valor d'una variable escrivint "@variableName"

3.3 - ITERACIÓ 3

3.3.1 – ANÀLISIS DE REQUISITS

Per aquesta iteració s'han realitzat totes les tasques necessàries per poder realitzar una release oficial de cara a l'empresa. Un punt molt important per poder realitzar la release és preparar l'aplicació perquè en futures releases oficials l'aplicació les detecti i s'actualitzi al obrir-se així els usuaris no hauran d'estar pendents de si hi ha una versió més nova o no, on han d'anar a descargarse-la, etc. L'aplicació cada set dies comprovarà si existeix una nova versió i en cas que existeixi s'actualitzarà automàticament.

Per altre banda s'ha decidit incorporar un botó per poder parar l'execució d'una story.

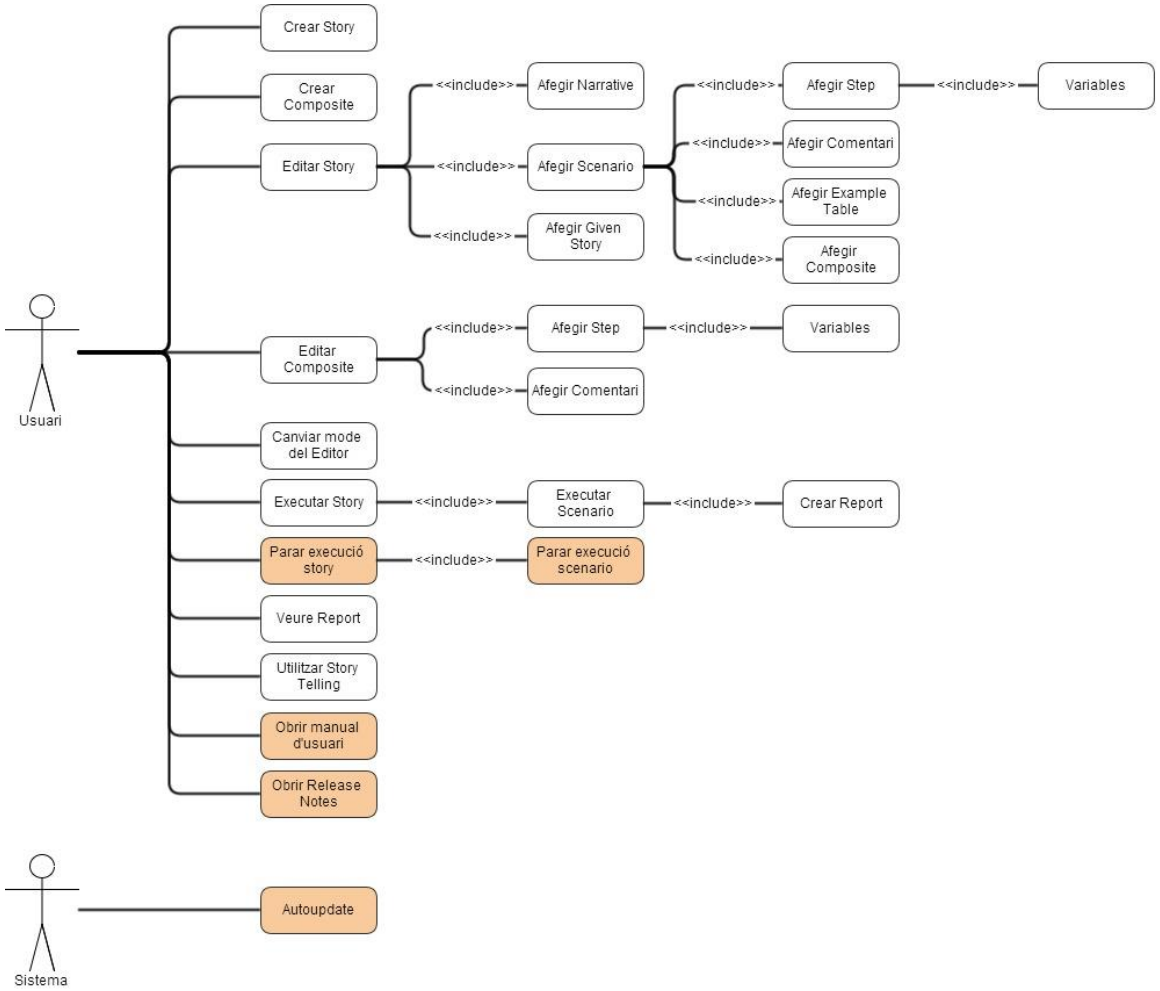
Abans de la release també s'ha de crear un manual d'usuari, ja que és convenient que al fer la release oficial els treballadors de l'empresa puguin obtenir un manual de la forma més senzilla possible, per això s'incorporarà a l'aplicació.

A més a més també es realitzarà la ReleaseNotes la qual es crearà automàticament agafant els tiquets que s'han tancat a Jira i un cop obtinguts aquest tiquets es crearà un document Html el qual es podrà visualitzar a través de l'aplicació.

3.3.2 ESPECIFICACIÓ

En aquest apartat es detalla l'especificació de les funcionalitats que es van afegir durant la segona iteració

CASOS D'ÚS



Cas d'ús	**Parar execució story**
Actors	Usuari
Precondicions	Hi ha una execució en curs
Flux bàsic:	

Precondició:	L'usuari té oberta la pestanya del report
Actor	Sistema
1) L'usuari prem el boto "Stop"	2) El sistema para l'execució
Postcondició:	Finalitza l'execució

Cas d'ús	**Parar execució scenario**
Actors	Sistema
Precondicions	1) Existeix una execució d'una story en curs 2) L'usuari a premut el botó "Stop"

Flux bàsic:

Precondició:	
Actor	Sistema
	1) El sistema para l'execució del scenari que s'està executant en aquell moment
Postcondició:	Finalitza l'execució de la story

Cas d'ús	**Obrir manual d'usuari**
Actors	Usuari
Precondicions	

Flux bàsic:

Precondició:	
Actor	Sistema
1) L'usuari selecciona l'opció "User Manual" del menú "Help"	2) El sistema obra una pestanya a l'editor amb el manual d'usuari
Postcondició:	Es mostra el manual d'usuari

Cas d'ús	**Obrir release notes**
Actors	Usuari
Precondicions	

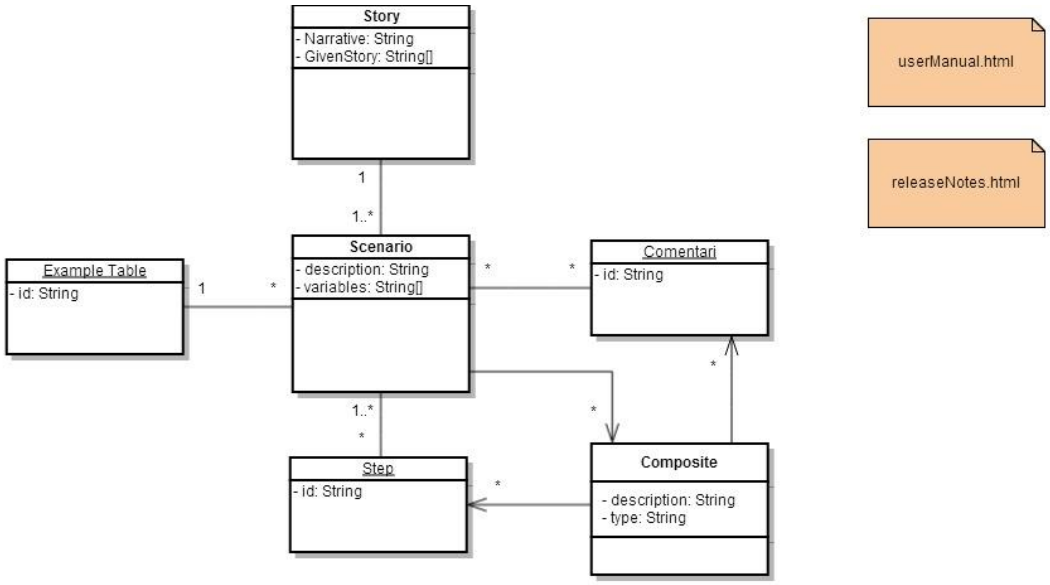
Flux bàsic:

Precondició:	
Actor	Sistema
1) L'usuari selecciona l'opció "Open Release Notes" del menú "Help"	2) El sistema obra una pestanya a l'editor amb la Release Notes
Postcondició:	Es mostra la Release Notes

Cas d'ús	**Autoupdate**
Actors	Sistema
Precondicions	
Flux bàsic:	

Precondició:	Existeix nova versió de l'aplicació
Actor	Sistema
5) L'usuari prem el botó acceptar del missatge	1) El sistema detecta que hi ha una nova versió de l'aplicació 2) El sistema es descarrega la nova versió 3) El sistema instal·la l'aplicació 4) El sistema mostra un missatge a l'usuari indicant que s'ha instal·lat correctament una nova versió.
Postcondició:	S'ha actualitzat l'aplicació

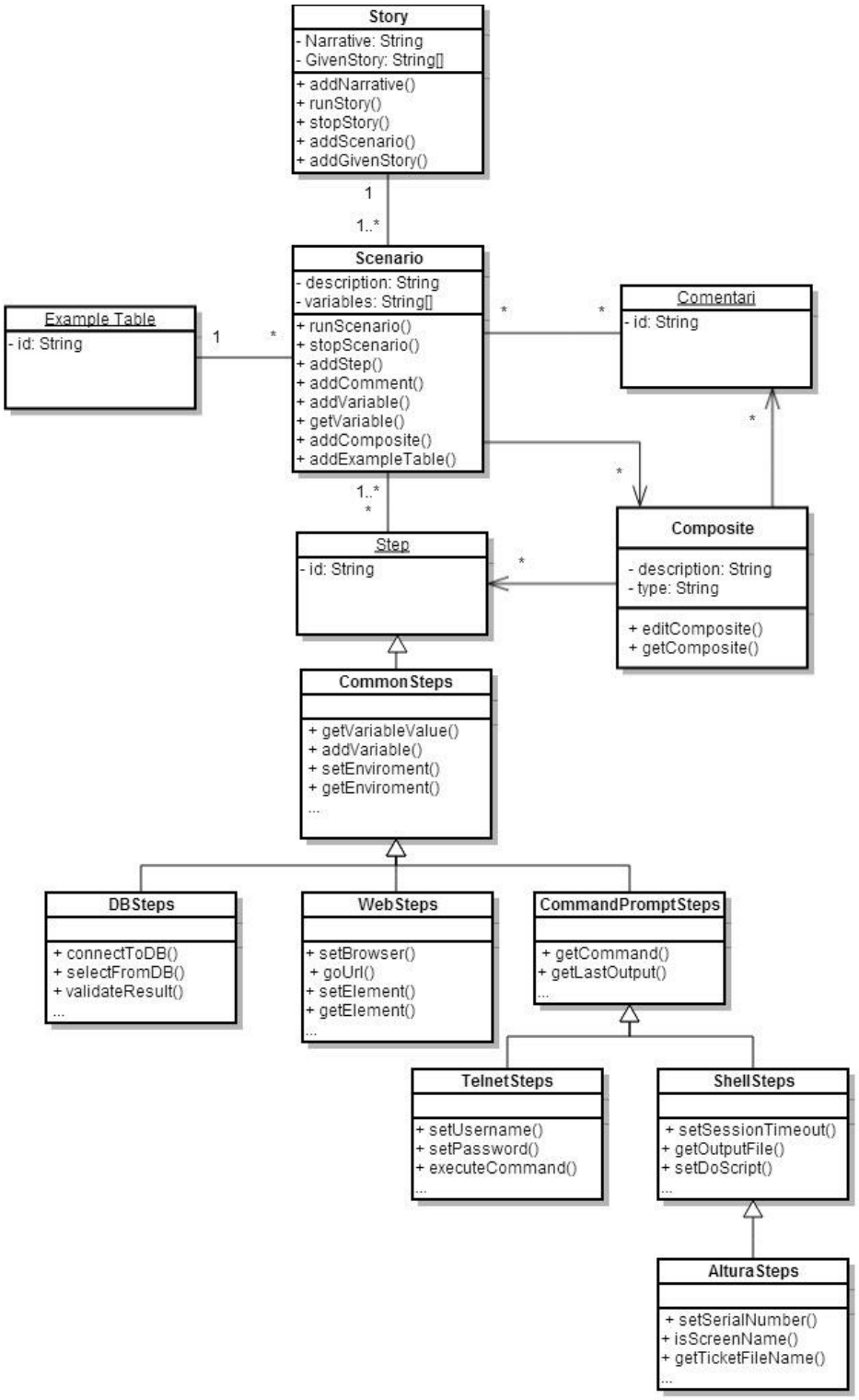
DIAGRAMA CLASSES D'ESPECIFICACIÓ



3.3.3 DISSENY

DIAGRAMA DE CLASES I FUNCIONS

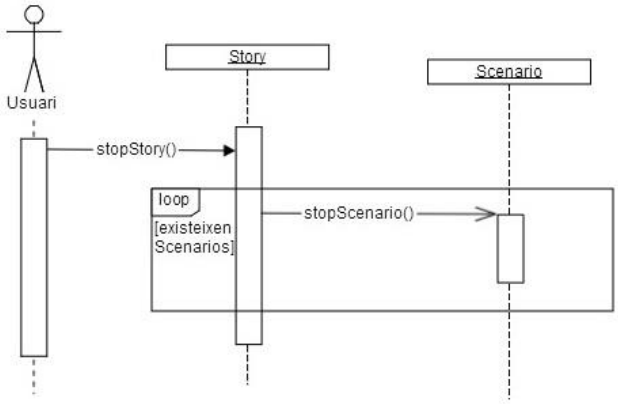
El diagrama d'aquesta iteració se li afegeixen algunes operacions a la classe Story i a la classe Scenario, en concret són les funcions "stopStory" que permet parar l'execució d'una story i "stopScenario" que permet parar l'execució d'un escenari

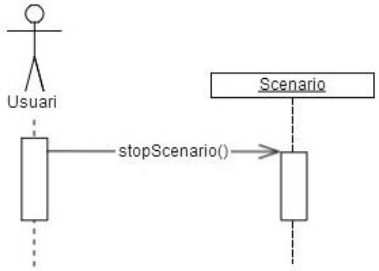


DIAGRAMES DE SEQÜÈNCIA

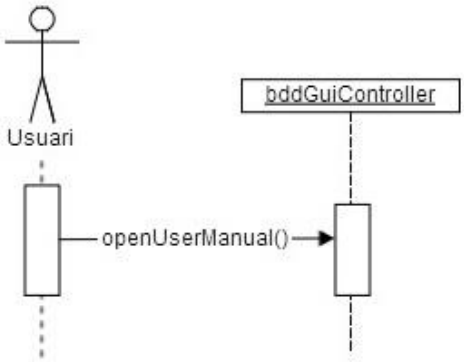
Els diagrames de seqüència d'aquesta iteració són els següents:

PARAR EXECUCIÓ STORY/SCENARIO

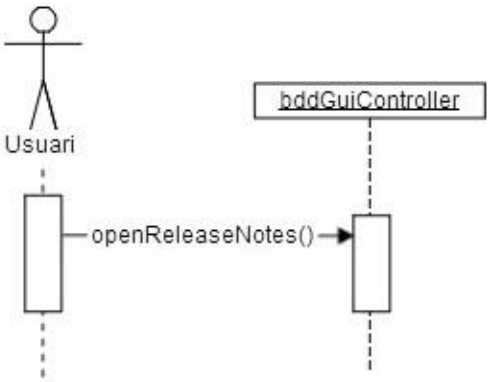




OBRIR MANUAL D'USUARI



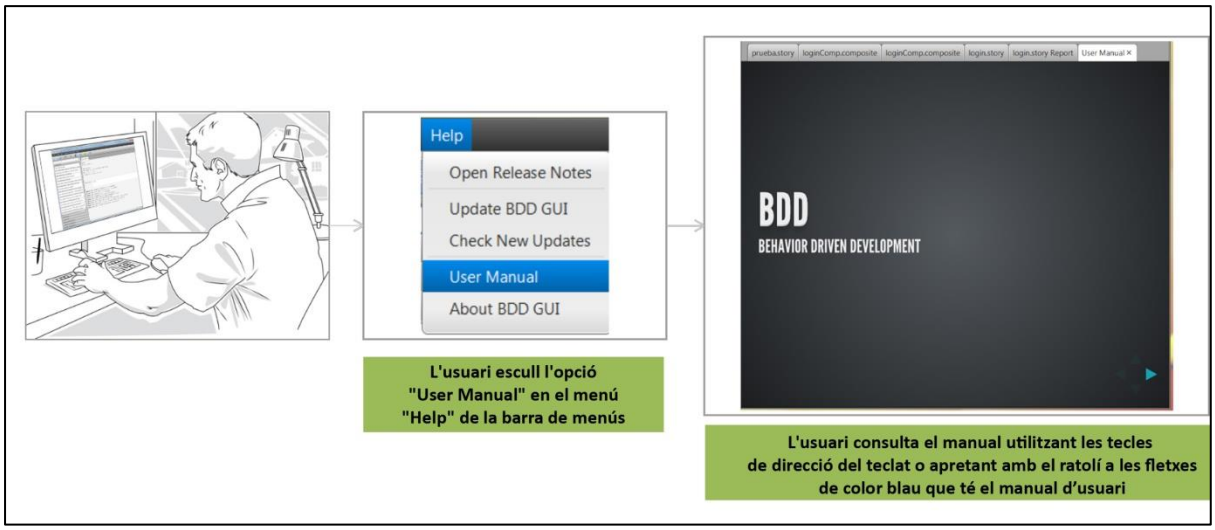
OBRIR RELEASE NOTES



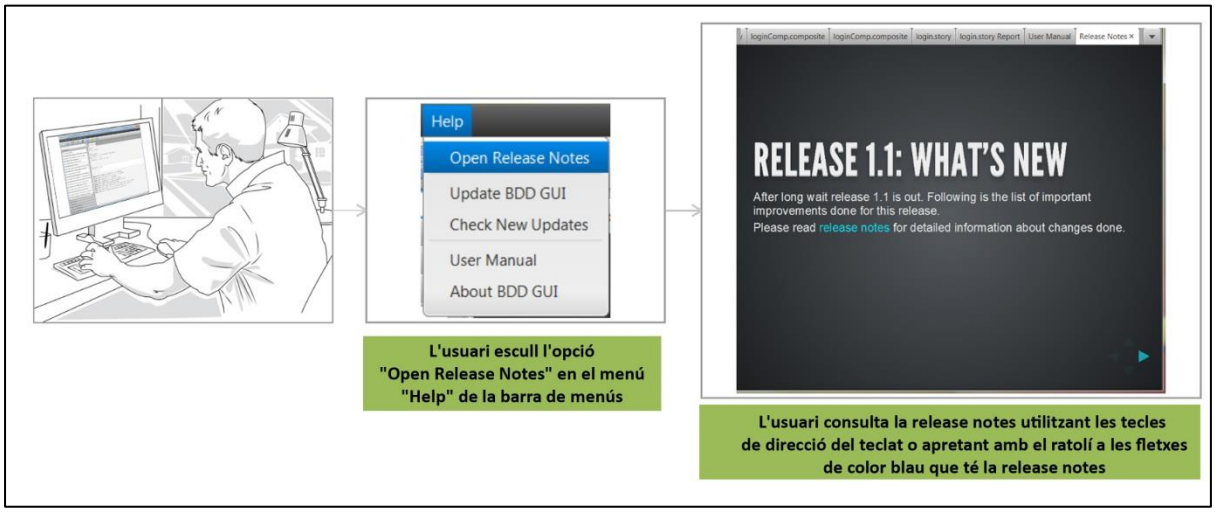
STORYBOARD

PARAR EXECUCIÓ

OBRIR MANUAL D'USUARI



OBRIR RELEASE NOTES



3.4 - ITERACIÓ 4

3.4.1 – ANÀLISIS DE REQUISITS

Desprès d'analitzar les propostes dels usuaris que van començar a utilitzar l'aplicació s'ha decidit afegir tres tipus de step que estenguin WebSteps, aquestes categories se les anomenarà OnceWebSteps, FinWebSteps i DashboardSteps. Les dos primeres serviran per poder filtrar els composites que es creïn ja que les persones de Finlàndia no els interessa veure els steps que utilitza la ONCE, més aviat molesta, i el mateix amb els cas de la ONCE. Per altra banda, el tipus anomenat DashboardSteps serviran per interactuar amb els Dashboards que ofereix GTech.

En relació als steps també ens ha arribat que a molts usuaris preferirien veure els composites per separat dels steps per això a la llista de possibles steps es separarà els steps dels composites.

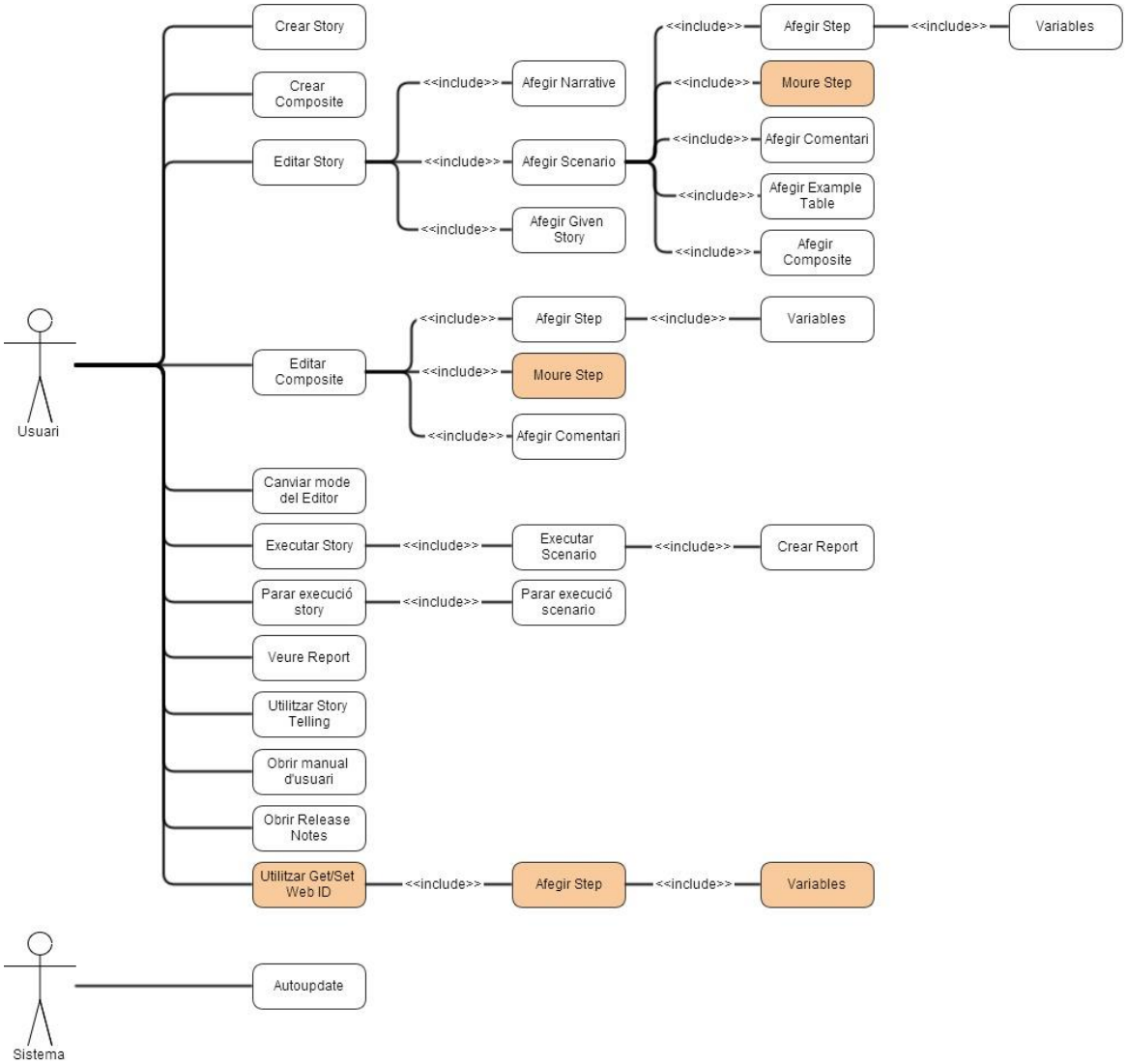
A l'hora de crear stories es va observar que molts dels treballadors els costava trobar com identificar l'identificador d'un element web o d'un element de les màquines Altura, per això s'ha decidit crear una nova funcionalitat amb la qual fent clic sobre l'element et retorna el valor que pot identificar l'element a la story i si es volgués la funcionalitat deixarà guardat aquest valor en una variable.

També s'ha vist que molts usuaris a l'hora de fer drag&drop d'un step s'equivoquen de posició (en general una posició per sota o per sobre de la posició que desitjaven), per això s'incorporarà l'opció de poder moure un step a una altre posició en una story o un composite.

3.4.2 ESPECIFICACIÓ

En aquest apartat es detalla l'especificació de les funcionalitats que es van afegir durant la segona iteració

CASOS D'ÚS



Cas d'ús	**Moure step story/composite**
Actors	Usuari
Precondicions	Hi ha una story/composati obert en l'editor
Flux bàsic:	

Precondició:	L'editor es troba en el mode per defecte
Actor	Sistema
1) L'usuari selecciona el step que desitja moure 2) L'usuari prem el botó primari del ratolí i arrastra el step fins a la posició que desitja posar el step 3) L'usuari deixa de prémer el botó primari	4) El sistema mou el step a la posició que

	desitja l'usuari
Postcondició:	S'ha mogut el step

Cas d'ús	**Consultar identificador amb Get/Set Web Id**
Actors	Usuari
Precondicions	
Flux bàsic:	

Precondició:	
Actor	Sistema
1) L'usuari escull l'opció "Get/Set Web ID" en el menú "Product"	
	2) El sistema mostra la funcionalitat de "Get Set Web ID"
3) L'usuari escriu la URL on vol consultar com identifciar un element 4) L'usuari escull quin navegador desitja utilitzar 5) L'usuari prem el botó "Open Browser"	
	6) El sistema obra el navegador a la URL que l'usuari desitja
6) L'usuari prem en el navegador l'element que desitja consultar l'identificador 7) L'usuari prem el botó "Get element Id"	
	8) El sistema mostra l'identificador de l'element
Postcondició:	S'ha obtingut l'identificador d'un element web

Cas d'ús	**Guardar identificador amb Get/Set Web Id**
Actors	Usuari
Precondicions	L'usuari ha obtingut un identificador amb Get/Set Web ID
Flux bàsic:	

Precondició:	
Actor	Sistema
1) L'usuari escriu el path on es troba la story on vol guardar la variable 2) L'usuari escriu el nom que vol que tingui la variable 3) L'usuari prem el botó "Set ID"	
	4) El sistema guarda la variable a la story que s'ha especificat
Postcondició:	S'ha guardat l'identificador d'un element web en una variable

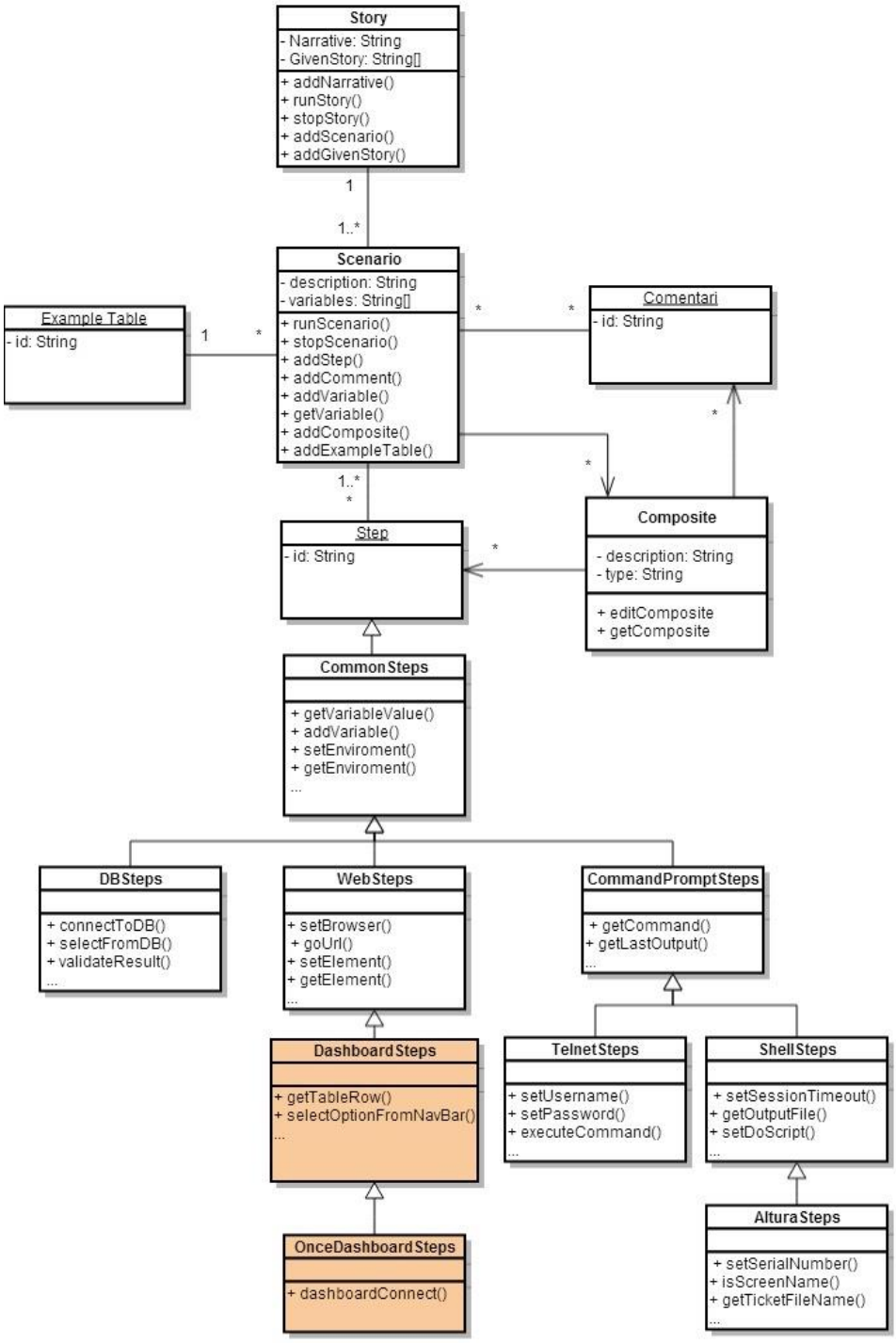
DIAGRAMA CLASSES D'ESPECIFICACIÓ

En aquesta iteració no s'ha modificat l'estructura d'aquest diagrama i s'ha mantingut la de l'anterior iteració

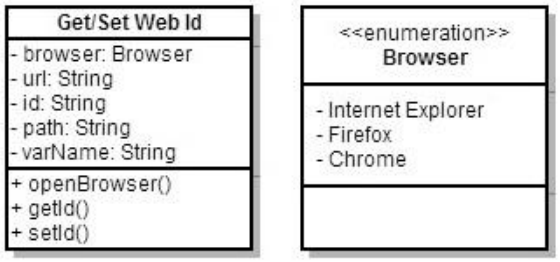
3.4.3 DISSENY

DIAGRAMA DE CLASES I FUNCIONS

En el diagrama de classes s'ha afegit les classes DashboardSteps i OnceDashboardSteps per poder filtar millor els steps.



Pel que fa al diagrama de classes del "Get/Set Web ID" seria el següent:

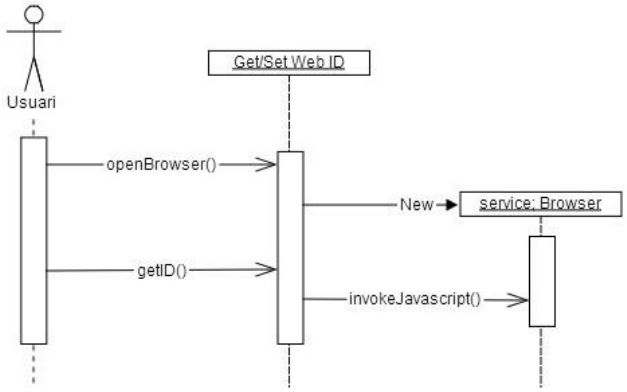


Els mètodes de la classe "Get/Set Web Id" serveixen per poder obrir el navegador ("openBrowser"), consultar l'identificador d'un element ("getId") i, finalment, afegir en una story un step en el que guarda en una variable l'identificador de l'element que s'ha obtingut ("setId").

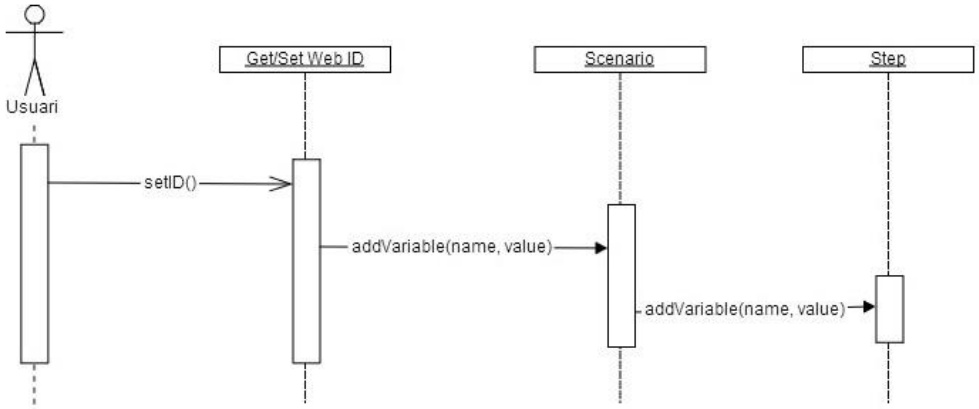
DIAGRAMES DE SEQÜÈNCIA

Els diagrames de seqüència d'aquesta iteració són els següents:

CONSULTAR ID AMB GET/SET WEB ID



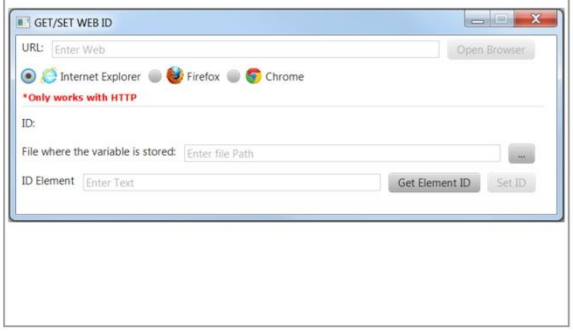
GUARDAR ID AMB GET/SET WEB ID



STORYBOARD

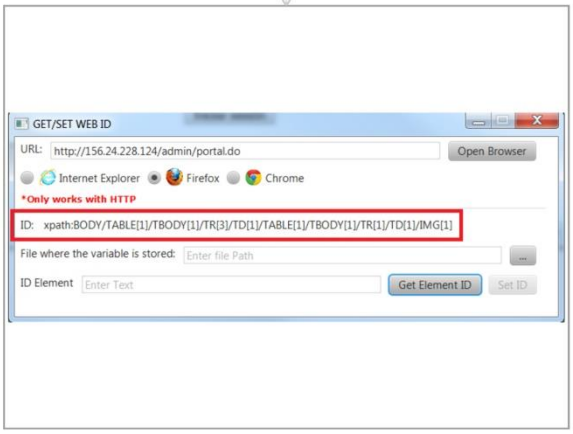
CONSULTAR IDENTIFICADOR AMB GET/SET WEB ID

L'usuari escull l'opció "Get/set Web ID" en el menú "Product" de la barra de menús



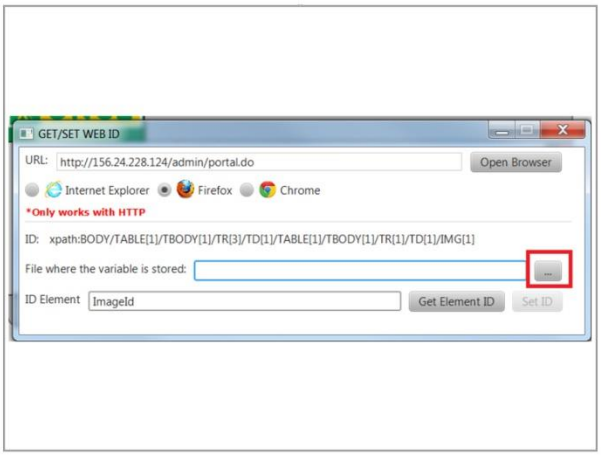
L'usuari indica la URL i el navegador que vol utilitzar per consultar els identificadors i prem el botó "Open Browser"

S'obre el navegador i l'usuari ja pot començar a trobar els identificadors. Per poder trobar l'identificador l'usuari ha de fer click sobre l'element en el que vol consultar el seu identificador

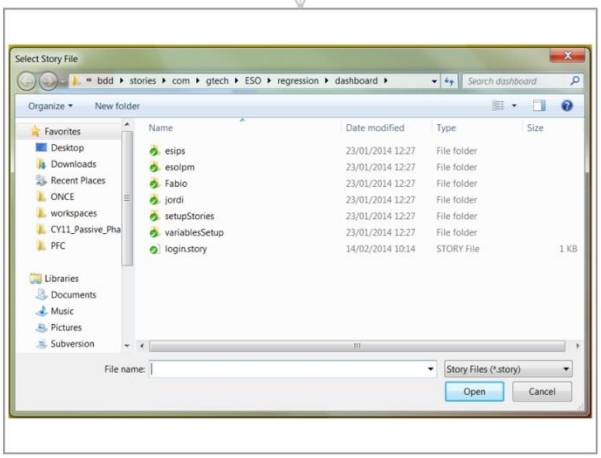


L'usuari prem el botó "Get Element ID" i apareix l'identificador de l'ultim element que s'ha fet click en el navegadaor

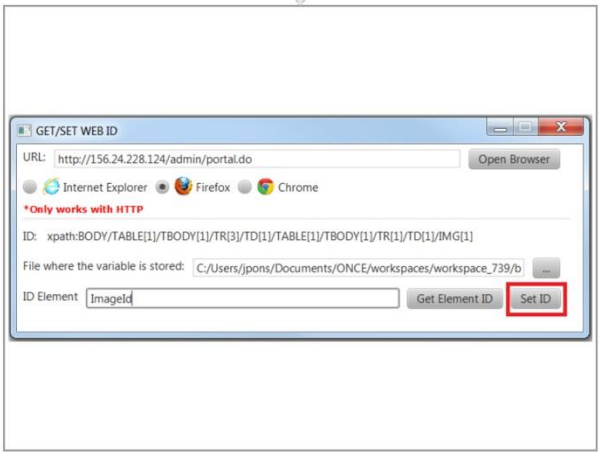
GUARDAR IDENTIFICADOR AMB GET/SET WEB ID



L'usuari prem el botó "..." per especificar on es troba la story on vol guardar la variable amb l'identificador de l'element



L'usuari indica on es troba guardada la story on vol afegir la variable



L'usuari prem el botó "Set ID"

3.5 - ITERACIÓ 5

3.5.1 – ANÀLISIS DE REQUISITS

Per aquesta última iteració s'ha decidit afegir una millora a la funcionalitat de web Story Telling, la millora consisteix en que l'aplicació, quan l'usuari decideix afegir un step, si aquest com a identificador té el "xpath" aleshores l'aplicació li demanarà guardar aquest xpath en una variable per tal de facilitar-ne el seu ús.

A més a més també s'ha vist que molts dels usuaris quan havien d'executar una story no s'havien molt bé que havien de posar a les properties que l'aplicació demana abans de l'execució així que per ajudar els usuaris s'afegiran comentaris de manera que per cada propietat que el "run story" et demani es pugui saber que és el que realment està demanant.

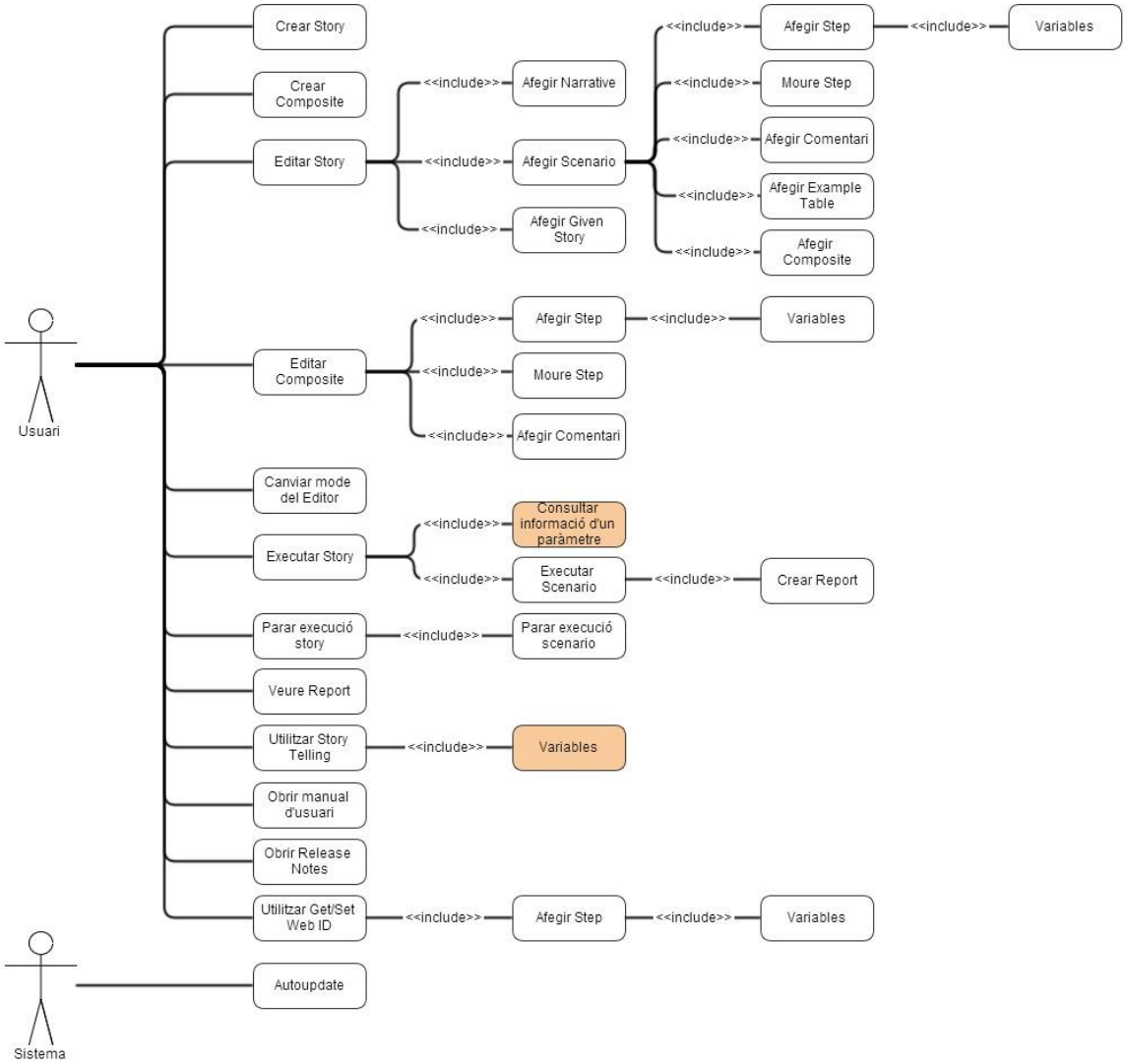
Un altre punt que molts usuaris han coincidit és en el que durant l'execució d'una story s'indiqui quan ha acabat l'execució, així l'usuari no haurà d'intuir quan ha acabat.

Finalment s'incorporarà el navegador Opera per executar stories, ja que en alguns casos les execucions automàtiques es realitzen en servidors Linux i algunes webs, com per exemple el dashboard de la ONCE, només funcionen amb Internet Explorer i Opera.

3.5.2 ESPECIFICACIÓ

En aquest apartat es detalla l'especificació de les funcionalitats que es van afegir durant la segona iteració

CASOS D'ÚS



Cas d'ús	**Story Telling utilitza variables per guardar xpath**
Actors	Usuari
Precondicions	1) El story Telling està mostrant els steps que poden ser afegits a una story 2) L'usuari ha apretat el boto "Add" o "Add Next" d'un dels steps

Flux bàsic:

Precondició:	
Actor	Sistema
3) El usuari especifica el nom que vol que tingui la variable	1) El sistema detecta que hi ha un step amb un xpath 2) El sistema demana a l'usuari que guardi el xpath en una variable

	4) El sistema guarda el xpath en una variable 5) El sistema substitueix pel nom de la variable tots els steps que tinguin aquell xpath
Postcondició:	S'ha guardat una variable amb el xpath

Cas d'ús	**Consultar info paràmetre d'execució**
Actors	Usuari
Precondicions	S'ha obert la pantalla de "Run" on es demana especificar els paràmetres d'execució

Flux bàsic:

Precondició:	
Actor	Sistema
1) L'usuari passa el ratolí per sobre el botó de info situat al costat del camp de text on s'ha d'especificar el valor del paràmetre	2) El sistema mostra la informació sobre aquell paràmetre
Postcondició:	S'ha consultat la informació d'un paràmetre

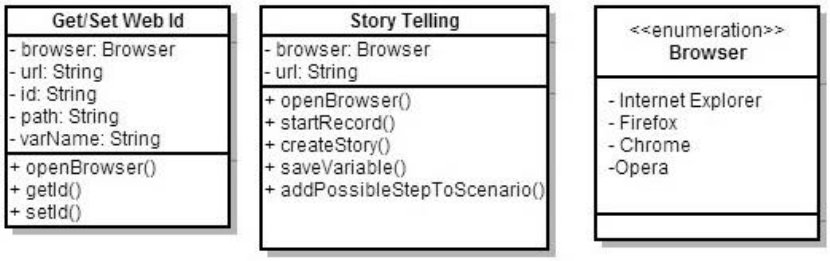
DIAGRAMA CLASSES D'ESPECIFICACIÓ

En aquesta iteració no s'ha modificat l'estructura d'aquest diagrama i s'ha mantingut la mateixa que la tercera iteració.

3.5.3 DISSENY

DIAGRAMA DE CLASES I FUNCIONS

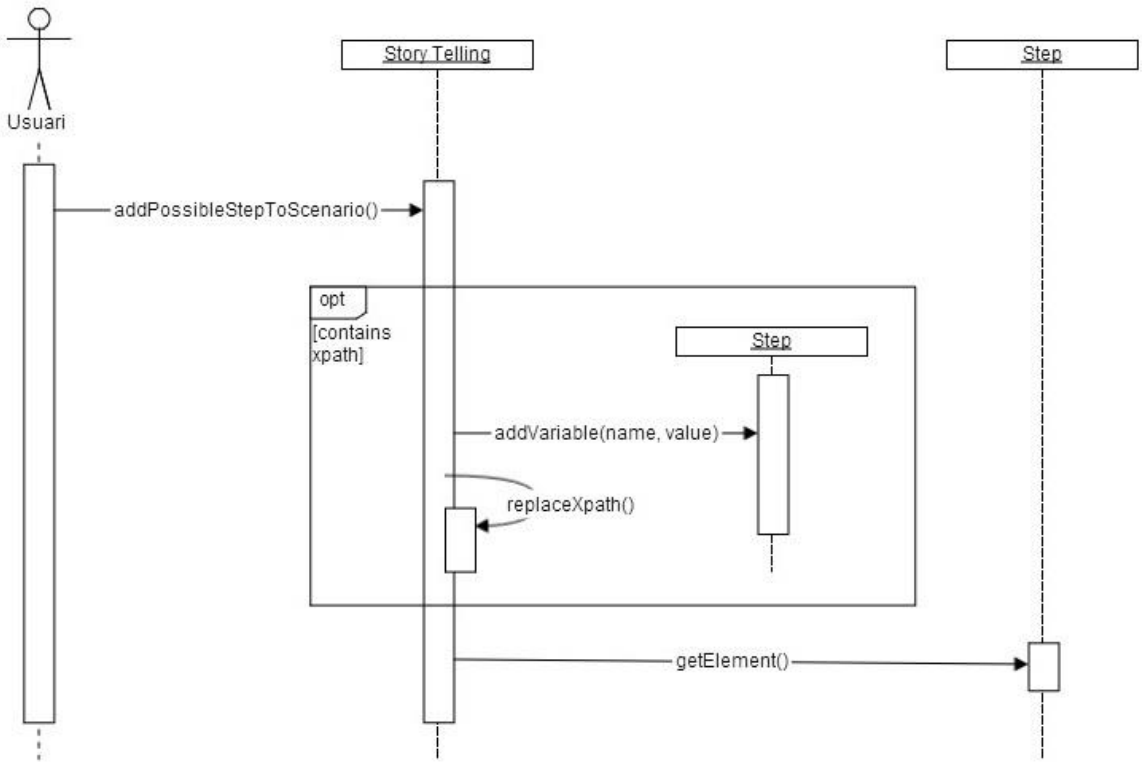
El diagrama de classes en aquesta iteració no es veu massa afectat ja que l'únic que canvia és que en el "enumeration" s'ha afegit l'Opera com a tipus de navegador i a la classe "Story Telling" s'ha afegit un mètode per poder guardar un xpath en una variable, aquest mètode se l'ha anomenat "saveVariable".



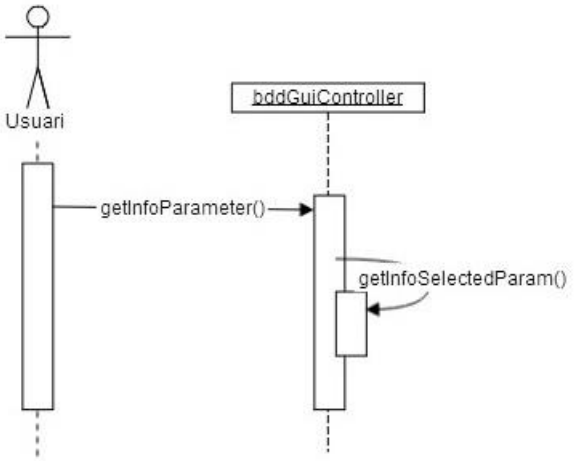
DIAGRAMES DE SEQÜÈNCIA

Els diagrames de seqüència d'aquesta iteració són els següents:

STORY TELLING UTILITZA VARIABLES PER GUARDAR XPATH

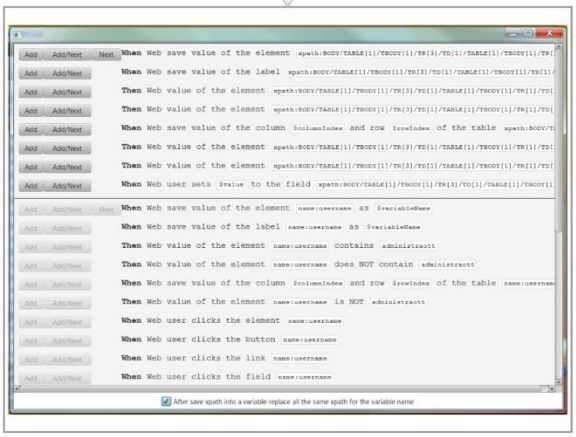


CONSULTAR INFO PARÀMETRE D'EXECUCIÓ

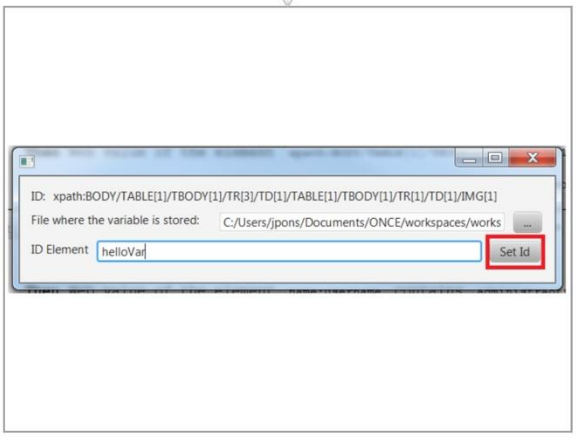


STORYBOARD

STORY TELLING UTILITZAR VARIABLES PER GUARDAR XPATH



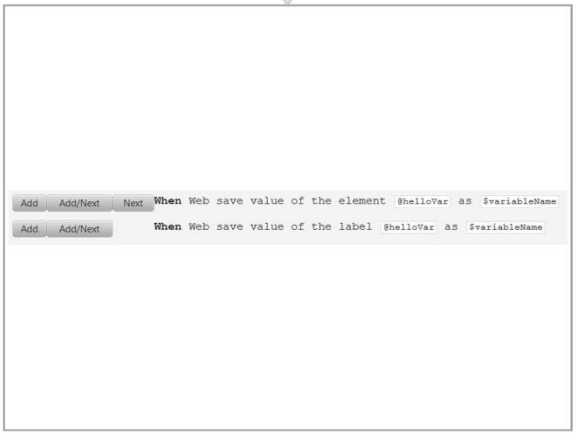
L'usuari prem el boto "Add" o "Add/Next".
El sistema detecte que un dels paràmetres del step conté un xpath



L'usuari especifica on es troba la story on vol afegir la variable amb el xpath.

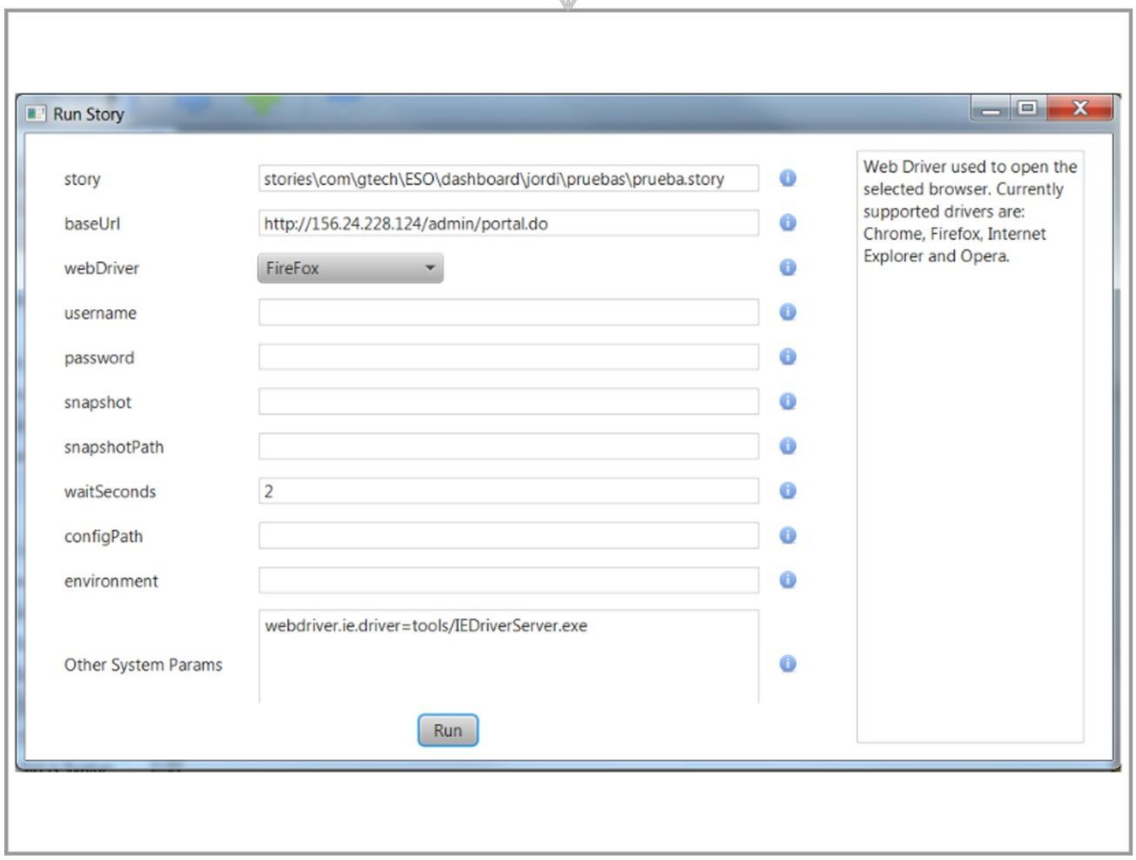
L'usuari especifica el nom de la variable

L'usuari prem el botó "Set ID"



Tots els steps que contenien el mateix xpath aquests són substituïtspel nom de la variable precedida pel símbol "@"

CONSULTAR INFO PARÀMETRE D'EXECUCIÓ



L'usuari per consultar l'utilitat de cada paràmetre passa el ratolí per sobre del símbol d'informació que apareix al costat de cada paràmetre

4 - IMPLEMENTACIÓ

La major part d'aquest projecte s'ha realitzat amb Java, tot I així a l'hora d'implementar algunes de les funcionalitats s'han utilitzat altres tecnologies que ens eren més convenients ja que ens facilitaven la feina o no es podien realitzar amb Java.
En aquesta part s'explicaran les tecnologies utilitzades durant tota la implementació i on s'han utilitzat.

4.1 JAVA

Java és un llenguatge de programació orientat a objectes, interpretat, senzill, robust ja que s'ha escrit pensant en poder verificar errors, i que permet l'execució de múltiples fils d'execució o executar diverses tasques de forma simultània i dinàmica.

Imatge 4: Logo Java

L'aplicació es va realitzar amb Java principalment per dues raons, la primera és per les característiques que té Java com l'orientació a objectes o poder realitzar diverses tasques de forma simultània i la segona, per ser un llenguatge que disposa de moltes llibreries on, per exemple, podem trobar la llibreria de Selenium.

4.2 JAVAFX

JavaFx és una llibreria de Java que intenta substituir Swing com a llibreria estàndard d'interfícies d'usuari.

Per crear la part de la GUI de l'aplicació es va utilitzar JavaFX ja que existeix un programa anomenat JavaFX Scene Builder que ens permetia crear interfícies d'una forma rapida i al mateix temps el programa s'encarregava de generar un controlador cosa que ens facilitava la feina a l'hora de seguir el MVC.

4.3 SELENIUM WEBDRIVER

Selenium WebDriver és una llibreria que ens permet realitzar comandes a un navegador i aconseguir informació. No és necessari utilitzar un servidor ja que utilitza una instancia d'un navegador i la controla.

Imatge 5: Logo Selenium

En el projecte es va utilitzar aquest framework en totes aquelles funcionalitats en les que es requeria interactuar amb pàgines web.

4.4 JBEHAVE

JBehave és un framework per Behaviour-Driven Develpment. Aquest framework permet crear, executar escenaris i veure el resultat de l'execució.

Imatge 6: Logo jBehave

4.5 HTML (HYPERTEXT MARKUP LANGUAGE)

HTML és un llenguatge que combina dades i etiquetes per estructurar textos i relacionar-los en forma d'hipertext. Es conegut per fer-se servir per crear pàgines web.

Aquest llenguatge s'ha utilitzat per crear el manual d'usuari i la release notes ja que d'aquesta manera els usuaris poden consultar el contingut utilitzant l'aplicació o si ho desitgen a través d'un navegador.

4.6 CSS (CASCADING STYLE SHEETS)

CSS és un llenguatge de fulls d'estil utilitzat per descriure la semàntica de presentació (l'aspecte i format) d'un document escrit en un llenguatge de marques. La seva aplicació més comuna és dissenyar pàgines web escrites en HTML però el llenguatge també pot ser aplicat a qualsevol classe de document XML.

En el nostre cas, CSS s'ha utilitzat pel mateix que l'HTML i, a més a més, un cop es generava un report d'una story executada (on es crea un document xml) aquest se li donava estil utilitzant CSS per veure més clarament que havia passat en l'execució.

4.7 JAVASCRIPT

JavaScript és un llenguatge interpretat que es defineix com orientat a objectes, basat en prototips, imperatiu, i dinàmic. És conegut sobretot pel seu ús en pàgines web.

Es va decidir utilitzar JavaScript per poder implementar les funcionalitats de "Web Story Telling" i "Web Get/Set Id" ja que per poder aconseguir informació dels elements d'una web era necessari inserir "listeners" a tots els elements de la web i així, quan l'usuari cliqui un element o escrigui en un camp de text, podem recopilar tota la 'informació que ens sigui necessària.

4.8 JQUERY

JQuery és un framework de Javascript que permet simplificar la manera d'interactuar amb els documents HTML, manipular l'arbre DOM, gestionar esdeveniments, desenvolupar animacions i afegir interacció amb la tecnologia AJAX en pàgines web. Ofereix una sèrie de funcionalitats basades en JavaScript que d'una altre manera requeririen molt més codi, és a dir, amb les funcions d'aquest framework s'aconsegueixen bons resultats en menys temps i espai.

Imatge 7: Logo jQuery

Es va decidir utilitzar jQuery en les mateixes funcionalitats que JavaScript, de fet la raó que ens va dur a utilitzar jQuery en els JavaScripts que s'injectaven va ser que amb aquest framework injectar els "listeners" es fa de la mateixa manera per tots els navegadors, en canvi si es realitzava amb JavaScript es necessitava comprovar si amb el navegador es pot realitzar unes crides o s'havien d'utilitzar unes altres, a més a més tal i com s'ha comentat gràcies a l'ús de jQuery ens va permetre aconseguir la informació amb més bons resultats i en menys temps.

4.9 GRADLE

Gradle és una eina d'automatització de projectes que es basa en els conceptes d'Apache Ant i Apache Maven e introdueix un llenguatge específic de domini basat en Groovy en lloc de la forma més tradicional de XML que declara la configuració del projecte.

Imatge 8: Logo Gradle

A diferencia d'Apache Maven, que defineix els cicles de vida, i Apache Ant, on s'invoquen els "targets", Gradle utilitza un graf acíclic (Directed acyclic graph "DAG") per determinar l'ordre en què les tasques es poden executar.

En el nostre cas gradle es va utilitzar per realitzar les builds automàtiques des de Jenkins (software d'integració continua open source).

5 -TESTS

Al llarg de tot el projecte s'han realitzat unit tests ja que ens servien per comprovar que les funcionalitats que s'implementaven funcionaven correctament i no només això sinó que quan s'implementava una nova funcionalitat o es retocava alguna part de la implementació els unit tests ens servien per veure que els canvis en el codi no provocaven un error en altres funcionalitats.

A més a més en el cas particular dels steps de tipus Web, per veure que realment funcionaven i que el llenguatge que utilitzaven era correcte, a part de les stories que van anar generant diferents treballadors de GTech, també vaig dedicar-me a crear stories del Dasboard de la ONCE per testejar que els steps realment funcionaven correctament, d'aquesta manera podia comprovar personalment si el llenguatge de cada step era l'adequat o feia falta canviar-lo.

Per realitzar els unit Test i saber en tot moment el Test Coverage s'han utilitzat diferents tecnologies, com Jacoco, jUnit, etc.

Pel que fa al Test Coverage, com que el càlcul es realitzava al fer-se una build, i les builds es realitzaven a Jenkins, les stories del Dashboard de la ONCE no s'utilitzaven ja que pot ser que en un moment donat la web no funcioni per algun motiu extern a la nostra aplicació i aleshores al realitzar el test de les stories ens donarien un error, per això s'han creat stories que comproven el bon funcionament dels steps els quals es testejaven utilitzant un fitxer html, d'aquesta manera ens assegurem de que si hi ha algun error, aquest és degut a que un test no passa correctament i que l'error no ve donat per culpa d'un mal funcionament del servidor on esta allotjada la web
.

5.1 JACOCO

Jacoco és una eina open source per mesurar i informar del "code coverage" d'un projecte escrit en Java. Gràcies a aquesta eina podíem saber en tot moment el percentatge de codi que teníem testejat, si existia codi mort, si algun mètode tenia una branca que no s'estava testejant, etc.

5.2 JUNIT

JUnit és una biblioteca per realitzar proves unitàries en aplicacions implementades amb el llenguatge de programació Java. Permet executar les classes Java de manera controlada, de forma que es pot avaluar si el funcionament de cadascun dels mètodes de la classe es comporta com s'espera. La seva funció és, donat un valor d'entrada, analitza el valor de retorn que s'espera, si la classe compleix amb l'especificació, aleshores JUnit indica que el mètode de la classe ha passat la prova; en cas contrari indica que ha fallat.

5.3 MOCKITO

Mockito és un framework open source per java que permet crear "mock objects" (objectes que imiten el comportament d'objectes reals de forma controlada) en els unit tests.

Imatge 9: Logo Mockito

Una de les raons que ens va dur a utilitzar mockito és que altres frameworks no funcionaven bé amb Jacoco ja que els unit tests que utilitzaven altres frameworks Jacoco no els comptabilitzava.

6 – PLANIFICACÍO I ESTIMACIÓ ECONÒMICA

6.1 – PLANIFICACIÓ

A continuació es detalla la planificació temporal del projecte. La planificació s'ha dividit en iteracions i cada una d'aquestes iteracións s'ha desglossat per tasques a realitzar.

Tasca	Hores
Iteració 1	**210h**
Crear una story	60h
Afegir Narrative	15h
Afegir Given Story	15h
Afegir Scenario	15h
Afegir Step	10h
Afegir Comentari	5h
Executar Story/Scenario	10h
Veure report	10h
Utilitzar Story Telling	60h
Meetings	10h
Iteració 2	**90h**
Composite	20h
Copy&Paste steps	10h
Afegir Example Table	20h
Guardar variable	15h
Utilitzar variable	15h
Meetings	10h
Iteració 3	**165h**
Parar execució	35h
Manual d'usuari	50h
Video tutorials	15h
Release Notes	45h
Meetings	20h
Iteració 4	**80h**
Knowledge transfer	30h
Recopilar consells/crítiques	15h
Get/Set Web ID	30h
Meetings	5h
Iteració 5	**75h**
Knowledge transfer	15h
Recopilar consells/crítiques	8h
Story telling utilitzant variables	30h
Consultar info paràmetre execució	20h
Meetings	2h
Hores Totals:	**620h**

6.1 – ESTIMACIÓ ECONÒMICA

Per calcular el cost econòmic del projecte, a més del cost d'hores de personal qualificat, en el que s'ha posat un cost de 70€/hora (per raons de privacitat el cost no és el que GTech posa, sinó que és un preu estimat), també cal incloure el cost del maquinari necessari, que en el cas d'aquest projecte només ha estat un ordinador per poder dur a terme l'implementació ja que el software que s'ha utilitzat ha estat tot openSource.

Per tant, sumant els costos de personal i material surt un cost total de:

Concepte	Preu
Cost d'hores de personal (70€/h)	43.400€
Ordinador	800€
Total	**44.200€**

7 - CONCLUSIONS

7.1 - OBJECTIUS ASSOLITS

El primer objectiu que es volia assolir i que a més era el que tenia més importància era el d'aconseguir un llenguatge específic que permetés la comunicació entre les diferents parts implicades en el procés d'entrega de software, aquest objectiu s'ha assolit i ha tingut una gran acollida com es pot veure reflexat amb les 650 stories creades en tres mesos, on el conjunt d'aquestes stories es tenia un total de 1,931 escenaris que comptabilitzant les example tables feien un total de 108,376 iteracions, el que equivaldria a 108,376 Test Cases de QA.

Pel que fa a l'aplicació també s'han aconseguit els seus objectius, per exemple a EEUU s'està utilitzant a 24 estats, els projectes de Finlàndia també estan aplicant BDD utilitzant l'eina i en el cas de la ONCE també s'està utilitzant BDD per realitzar els tests del seu dashboard i de les màquines Altura.

En el cas particular de Finlandia els clients comencen a mostrar un gran interès amb aquesta nova manera de realitzar l'especificació i com a conseqüència en projectes nous es comencen a realitzar workshops on es creen stories amb la col·laboració del client.

A nivell d'eficiència en la creació de test, aquesta s'ha vist augmentada, només fa falta veure la quantitat de stories que s'han creat amb molt poc temps.

També ha augmentat la freqüència d'execució dels tests, ja que els projectes en els que s'aplica BDD, es configura el Jenkins perquè aquest executi les stories cada vegada que es realitza un canvi en el codi del projecte, això vol dir que cada vegada que es realitza una build s'està testejant que la implementació és correcta.

7.2 - FUTURES FUNCIONALITATS POSSIBLES

Tot i que a dia d'avui ja s'han implementat moltes funcionalitats de l'aplicació encara existeixen coses per fer.

Una de les funcionalitats que es té pensat realitzar en un futur és la de millorar el "Story Telling" per tal de que no només es pugui utilitzar en maquines Altura o en web sinó que també funcioni amb els prototypes que es creen utilitzant el programa Axure. Tot i que encara falta una mica de recerca per saber com realitzar aquesta part, ja es té una lleugera noció de com es podria arribar a fer. Aquesta millora ajudaria a poder crear els tests abans de la implementació i assegurar-se que el test funciona.

També s'ha decidit incorporar nous tipus de step, concretament steps que estendran els steps del dashboard però que seran específics de cada producte com ESIPS, ESMS, etc.

Una possible nova funcionalitat és la de poder utilitzar l'aplicació a través d'una web però de moment s'ha descartat ja que això representaria tota la implementació de la vista.

També s'ha decidit fer que el projecte passi a ser opensource, d'aquesta manera es proporcionarà una eina per fer BDD a la comunitat. Tenint en compte que actualment les eines que es proporcionen per fer BDD es necessita tenir uns alts coneixements tècnics, al proporcionar aquesta aplicació es podrà ampliar el nombre de persones que podran fer BDD ja que no farà falta tenir els coneixements que actualment es necessiten.

7.3 - CONCLUSIÓ PERSONAL

Personalment crec que aquest projecte ha estat un èxit tant per l'empresa com a nivell personal.

Per l'empresa perquè un cop l'aplicació va estar suficientment avançada com perquè els usuaris comencessin a realitzar les stories aquesta va tindre una molt bona acollida, molts dels que van utilitzar-la es van involucrar de manera que el feedback que es va rebre sempre va ser constructiu cosa que va ajudar a l'hora de realitzar canvis i millores.

També considero que ha estat un èxit per l'empresa ja que s'ha pogut comprovar que la corba d'aprenentatge dels usuaris no ha estat molt elevada, si que cal remarcar que les persones que tenen nocions de programació els costa una mica més que la resta pel fet que quan creen una story pensen en realitzar-la com si creessin un programa i al no poder realitzar condicionals (if, else if, else) ni tampoc bucles (a excepció dels que pots arribar a crear gràcies a les example tables) els costa veure la manera de realitzar el test, tot i així al cap d'un temps d'estar creant stories s'adaptaven i en alguns casos trobaven útil el fet de que no existissin ni condicionals ni bucles.

A nivell personal també crec que ha estat un èxit ja que tot i tindre unes bones expectatives amb el projecte, aquestes s'han superat. No només per haver aconseguit acabar-lo en el temps que s'havia pensat i haver realitzat totes les funcionalitats que s'esperaven realitzar sinó perquè ha estat una gran experiència, i durant tot el projecte he passat grans moments i anècdotes que de ben segur recordaré.

Al principi quan em van explicar el que es volia realitzar vaig veure que hi havia molta feina i que el temps que es tenia no era massa però al final ha resultat possible crear l'aplicació i no només això sinó que hem pogut crear tutorials de vídeo, dedicar temps a explicar a la gent com utilitzar-la, millorar implementacions perquè siguessin més eficients, etc.

Un dels punts que crec que m'ha ajudat molt en aquest projecte ha estat en l'organització del treball a realitzar. Al començament del projecte no sabia molt bé per on començar, que realitzar primer i que no, però a mesura que el projecte va anar avançant vaig adonar-me'n que feia les coses d'una manera molt més estructurada.

També ha estat un èxit pel que fa a aprenentatge ja que a l'implementar l'aplicació ha estat necessari aprendre llenguatges que tot i que els coneixia no els havia utilitzat massa i també m'ha servit per aprendre a realitzar implementacions que al inici del projecte no sabia com fer-ho. Un punt a remarcar i que em va cridar l'atenció va ser quan es va decidir utilitzar alguns dels patrons que ens havien ensenyat durant la carrera, em va cridar l'atenció pel fet que els coneixia a nivell teòric però no a nivell d'implementació.

A més a més també m'ha servit per acabar d'entendre, consolidar i fins i tot descobrir moltes coses d'Agile i que, com he explicat al inici de la memòria, no em va ser possible aprendre amb profunditat durant la carrera.

Un cop acabat el projecte veig que per acabar de consolidar els coneixements obtinguts a la Universitat triar GTech i, en concret aquest projecte, va ser una molt bona elecció.

8 - BIBLIOGRAFIA

Documentació sobre jBehave: http://jbehave.org/

Documentació sobre BDD: http://dannorth.net/introducing-bdd/

Blog sobre temes d'Agile i programació: http://technologyconversations.wordpress.com/

Documentació sobre Jenkins: http://jenkins-ci.org/

Manual d'usuari de javaFX Scene Builder:
http://docs.oracle.com/javafx/scenebuilder/1/user_guide/jsbpub-user_guide.htm

Web de Mockito: https://code.google.com/p/mockito/

Web de Selenium: http://docs.seleniumhq.org/

Documentació sobre Gradle: http://www.gradle.org/

9 - GLOSSARI

Altura: Terminal gràfic en el que es pot comprar loteria, consultar si un tiquet d'un joc ha aconseguit algun premi, etc.

Composite: clàusula que agrupa un conjunt d'un o més steps en un de sol per facilitar la comprensió d'un scenario.

Dashboard: interfície gràfica on l'usuari pot administrar un equip o software.

Framework: estructura conceptual i tecnològica de suport definit, normalment amb artefactes o mòduls de software concrets, que poden servir de base per l'organització i el desenvolupament de software. Típicament, pot incloure suport de programes, biblioteques i un llenguatge interpretat, entre altres eines, per ajudar a desenvolupar i unir els diferents components d'un projecte.

Graphical User Interface (GUI): interfície d'usuari que utilitza elements gràfics que permeten interactuar de forma molt més intuïtiva amb un sistema informàtic.

Llicència de joc: autorització concedida per una autoritat reguladora que permet a l'empresa dur a terme una determinada acció en aquesta jurisdicció, normalment col·laboració o manteniment dels jocs.

Quality Assurance (QA): persones encarregades de testejar els projectes per tal que s'adaptin als requisits i tinguin la qualitat que es demana.

Release Notes: document suplementari que s'entrega al client cada vegada que un bug es corregeix o e3s fa una millora en el producte.

Software Requirements Specification (SRS): document que conté una descripció completa del comportament del sistema que ha de ser implementat. Inclou un conjunt de casos d'ús que descriuen les interaccions que els usuaris tindran amb el software.

Steps: clàusula que identifica una precondició, una acció o una comprovació en un escenari.

10 – ANNEX

10.1 LLISTA DE STEPS

10.2.1 COMMONSTEPS

Step	Type	Description
Given locale is $language $country	ALL	
Given variable $variableName is $variableValue	Basic	Saves value as variable that can be used later.
When $type script $scriptText is executed	All	Executes custom made script. Available Java class is CommonSteps referenced as steps. Currently supported scripting type is Groovy. If scriptText ends with file name prefix (i.e. test.groovy), the step shall load the script from the file.If file name is specified without the full path, the step shall search for the specified file inside stories directory and all subdirectories. *** Do NOT use this step inside stories. Place it inside a composite step. ***
When save random string with length $length as $variableName	All	Saves random string as internal variable that can be used later.

10.2.2 WEBSTEPS

This step collection also includes steps from the following collections: **CommonSteps**

Step	Type	Description
Given Web timeout is $timeout seconds	All	Sets step timeout.
When Web save value of the element $id as $variableName	All	Saves value of the label as internal variable that can be used later. Element references (id, name, className, link and xpath) can be specified as variables. Element references should be separated from value by colon (:). If value starts with @, it will be considered variable and replaced with the correct value. Value can describe type of finder by starting with "id:", "name:", "classname:", "linkText:" or "xpath:". Additional processing can be performed using functions. Following functions are available: .substr([START_INDEX], [END_INDEX]) i.e. If value of variable is abc123def456 and variable name is testVariable.substr(1, 4), testVariable would receive value bc1
Given Web snapshot is taken on every click	All	Snapshot shall be taken before every click.
Given Web snapshot is NOT	All	Snapshot will NOT be taken before every click.

taken on every click		
When Web snapshot is taken	All	Takes snapshot.
Given Web user is on the home page	Basic	Loads home page. URL is set through @ref setBaseUrl.
Then Web page title is $title	Basic	Verifies that page title is correct.
Given Web base URL is $url	Basic	Sets the base URL.
Given Web user is on the URL $url	All	Loads specified URL.
Given Web cookies are deleted	All	Deletes all cookies.
Given Web Driver is $webDriver	Basic	Sets the Web Driver. Currently supported drivers are: InternetExplorer, FireFox, Chrome, RemoteWebDriver.
Given Web wait after action is $wait seconds	All	Sets wait period after an action.
Then Web wait for $seconds seconds	All	Waits until the execution of next step.
When Web user selects $value from the droplist $id	Basic	Selects value from the droplist. Element references (id, name, className, link and xpath) can be specified as variables. Element references should be separated from value by colon (:). If value starts with @, it will be considered variable and replaced with the correct value. Value can describe type of finder by starting with "id:", "name:", "classname:", "linkText:" or "xpath:".
Then Web selected value of the droplist $id is $value	Basic	Checks that the value from the droplist is correct. Element references (id, name, className, link and xpath) can be specified as variables. Element references should be separated from value by colon (:). If value starts with @, it will be considered variable and replaced with the correct value. Value can describe type of finder by starting with "id:", "name:", "classname:", "linkText:" or "xpath:".
When Web user clicks the element $id	Basic	Clicks the element. Multiple IDs should be separated by space character. Element references (id, name, className, link and xpath) can be specified as variables. Element references should be separated from value by colon (:). If value starts with @, it will be considered variable and replaced with the correct value. Value can describe type of finder by starting with "id:", "name:", "classname:", "linkText:" or "xpath:".
When Web user $action alert	All	ACCEPT or DISMISS alert popup.
When Web user clicks $times times the link $id	Basic	Clicks the element. Element references (id, name, className, link and xpath) can be specified as variables. Element references should be separated from value by colon (:). If value starts with @, it will be considered variable and replaced with the correct value. Value can describe type of finder by starting with "id:", "name:", "classname:", "linkText:" or "xpath:".
Then Web value of the element $id is $value	Basic	Checks that the value of the element is correct.If you want to use a variable you need to write @+variableName, in the case of type variables Date you must specify which attribute

		you want (day, month, year or date of the year), so you need to write @+ variableName+.+[day | month | year | dayOfYear]. Summary: @+variableName(+.+attribute). Element references (id, name, className, link and xpath) can be specified as variables. Element references should be separated from value by colon (:). If value starts with @, it will be considered variable and replaced with the correct value. Value can describe type of finder by starting with "id:", "name:", "classname:", "linkText:" or "xpath:".
Then Web value of the element $id is NOT $value	Basic	Checks that the value of the element is NOT the same as specified value. If you want to use a variable you need to write @+variableName, in the case of type variables Date you must specify which attribute you want (day, month, year or date of the year), so you need to write @+ variableName+.+[day | month | year | dayOfYear]. Summary: @+variableName(+.+attribute). Element references (id, name, className, link and xpath) can be specified as variables. Element references should be separated from value by colon (:). If value starts with @, it will be considered variable and replaced with the correct value. Value can describe type of finder by starting with "id:", "name:", "classname:", "linkText:" or "xpath:".
When Web user sets $value to the field $id	Basic	Sets that the value of the field. If value is ENTER, enter key will be pressed. If you want to use a variable you need to write @+variableName, in the case of type variables Date you must specify which attribute you want (day, month, year or date of the year), so you need to write @+ variableName+.+[day | month | year | dayOfYear]. Summary: @+variableName(+.+attribute). Element references (id, name, className, link and xpath) can be specified as variables. Element references should be separated from value by colon (:). If value starts with @, it will be considered variable and replaced with the correct value. Value can describe type of finder by starting with "id:", "name:", "classname:", "linkText:" or "xpath:".
When Web user sets $status to the checkbox $id	Basic	Sets that the value of the checkbox is checked (CHECKED) or not (NOT CHECKED). If you want to use a variable you need to write @+variableName, in the case of type variables Date you must specify which attribute you want (day, month, year or date of the year), so you need to write @+ variableName+.+[day | month | year | dayOfYear]. Summary: @+variableName(+.+attribute). Element references (id, name, className, link and xpath) can be specified as variables. Element references should be separated from value by colon (:). If value starts with @, it will be considered variable and replaced with the correct value. Value can describe type of finder by starting with "id:", "name:", "classname:", "linkText:" or "xpath:".
When Web switch to frame $frame	All	Switches context to a specified frame.
Then Web switch to new window	All	
Given Web locale is $language $country	All	

Given Web enter key is used instead of click	All	Enter key shall be used instead of left mouse button.
Given Web click is used instead of enter key	All	Left mouse button shall be used instead of enter key.
Given Web browser size is $width width and $height height	All	Sets browser size.
Given Web Remote Driver Url is $url	All	Sets Remote Web Driver URL. See https://saucelabs.com/docs/browsers for more info.
Given Web desired capabilities are for $browser	All	Sets capabilities. See https://saucelabs.com/docs/browsers for more info.
Given Web desired capability is $capabilityName with value $value	All	Sets capability. See https://saucelabs.com/docs/browsers for more info. If capabilityName or value are set to NULL, capability shall not be added.
Given Web page load timeout is ignored	All	Page load timeout exception is ignored.
Given Web page load timeout is NOT ignored	All	Page load timeout exception is NOT ignored.
Given Web element timeout is $timeout seconds	All	Sets step timeout.
When Web user clicks the link $id $times times	All	Clicks the element multiple times. If times parameter is set to 0, nothing shall be executed.
When Web user clicks numbers $numbers with the base ID $baseId	Basic	Clicks elements for each specified number. IDs that are clicked are generated using baseId and each number. i.e. If step is "When Web user clicks numbers 1 2 3 4 with the base ID some_id_", ID that shall be clicked are: some_id_1 some_id_2 some_id_3 some_id_4
When Web save values of the Date fields with the IDs $dayOfMonthId $monthId $yearId as $dateVariable	All	Saves value of the fields as internal variable that can be used later. Element references (id, name, className, link and xpath) can be specified as variables. Element references should be separated from value by colon (:). If value starts with @, it will be considered variable and replaced with the correct value. Value can describe type of finder by starting with "id:", "name:", "classname:", "linkText:" or "xpath:".
Then Web label $id does NOT exist	Basic	Verifies that the label does NOT exist. Element references (id, name, className, link and xpath) can be specified as variables. Element references should be separated from value by colon (:). If value starts with @, it will be considered variable and replaced with the correct value. Value can describe type of finder by starting with "id:", "name:", "classname:", "linkText:" or "xpath:".
Then Web label $id does exist	Basic	Verifies that the label does exist. Element references (id, name, className, link and xpath) can be specified as variables. Element references should be separated from value by colon (:). If value starts with @, it will be considered

		variable and replaced with the correct value. Value can describe type of finder by starting with "id:", "name:", "classname:", "linkText:" or "xpath:".
Then Web value of the element $id contains $value	Basic	Verifies that the element contains correct value. If you want to use a variable you need to write @+variableName, in the case of type variables Date you must specify which attribute you want (day, month, year or date of the year), so you need to write @+ variableName+.+[day | month | year | dayOfYear]. Summary: @+variableName(+.+attribute). Element references (id, name, className, link and xpath) can be specified as variables. Element references should be separated from value by colon (:). If value starts with @, it will be considered variable and replaced with the correct value. Value can describe type of finder by starting with "id:", "name:", "classname:", "linkText:" or "xpath:".
Then Web value of the column $columnIndex and row $rowIndex of the table $id is $value	Basic	Verifies that the column has correct value. Check empty cell value = NULL. If you want to use a variable you need to write @+variableName, in the case of type variables Date you must specify which attribute you want (day, month, year or date of the year), so you need to write @+ variableName+.+[day | month | year | dayOfYear]. Summary: @+variableName(+.+attribute).
Then Web value of the column $columnIndex and row $rowIndex of the table $id contains $value	Basic	Verifies that the column contains correct value. If you want to use a variable you need to write @+variableName, in the case of type variables Date you must specify which attribute you want (day, month, year or date of the year), so you need to write @+ variableName+.+[day | month | year | dayOfYear]. Summary: @+variableName(+.+attribute).
Then Web element $id is in status $status	Basic	Verifies whether the Web element is in the specified status. Currently supported statuses are: enabled disabled visible invisible Element references (id, name, className, link and xpath) can be specified as variables. Element references should be separated from value by colon (:). If value starts with @, it will be considered variable and replaced with the correct value. Value can describe type of finder by starting with "id:", "name:", "classname:", "linkText:" or "xpath:".
Then Web checkbox $id is $status	Basic	Verifies checkbox $id is $status (status can be CHECKED or NOT CHECKED)
When Web user sets $value to the password field $id	All	Sets that the value of the field. Element references (id, name, className, link and xpath) can be specified as variables. Element references should be separated from value by colon (:). If value starts with @, it will be considered variable and replaced with the correct value. Value can describe type of finder by starting with "id:", "name:", "classname:", "linkText:" or "xpath:".
Given Web snapshot is taken on failure	All	Snapshot will be taken when scenario fails.
Given Web snapshot is NOT taken on failure	All	Snapshot will NOT be taken when scenario fails.
When Web switch to default frame	All	Switches context to the default (main) frame.
When Web $type	All	Executes custom made script. Available Java class is

script $scriptText is executed		WebSteps referenced as steps. Currently supported scripting type is Groovy. If scriptText ends with file name prefix (i.e. test.groovy), the step shall load the script from the file.If file name is specified without the full path, the step shall search for the specified file inside stories directory and all subdirectories. *** Do NOT use this step inside stories. Place it inside a composite step. ***
Then Web Driver is closed	All	Web Driver is closed

10.2.3 DASHBOARDSTEPS

This step collection also includes steps from the following collections: **WebSteps**, **CommonSteps**

Step	Type	Description
Then Dashboard table $table has row $row	Basic	Verify if exists one row in the table that values matches. Each cell must be encapsulated with the | symbol. If the content of a cell is indifferent must write NULL. Example: 123||NULL|321 If you don't want to check all the cells, you have to write until the last one that needs to be checked. Example: a table with 7 columns if you only want to check the first three or only the third, you would need to write the three columns "123|NULL|321" in both cases
When Dashboard column $column from the table $table that has row $row is clicked	Basic	User the the element in the column that matches the elements of the row. Each cell must be encapsulated with the | symbol. If the content of a cell is indifferent must write NULL. Example: 123||NULL|321 If you don't want to check all the cells, you have to write until the last one that needs to be checked. Example: a table with 7 columns if you only want to check the first three or only the third, you would need to write the three columns "123|NULL|321" in both cases
Then Dashboard table $table has NO row $row	Basic	Verify if doesn't exist one row in the table that values matches. Each cell must be encapsulated with the | symbol. If the content of a cell is indifferent must write NULL. Example: 123||NULL|321 If you don't want to check all the cells, you have to write until the last one that needs to be checked. Example: a table with 7 columns if you only want to check the first three or only the third, you would need to write the three columns "123|NULL|321" in both cases

10.2.4 ONCEDASHBOARDSTEPS

This step collection also includes steps from the following collections: **DashboardSteps**, **WebSteps**, **CommonSteps**

Step	Type	Description
Given ONCE Dashboard login using InternetExplorer	Basic	Performs a Dashboard login for ONCE using InternetExplorer.
When User access to Instant Services > System Preferences > $menu3Option	Basic	Access to options for System Preferences under Instant Services
When user moves $book in $product from $location with id $locationID to $newLocation with id $newLocationID	Basic	User move a specific book in "inventory Status" from one location to another one

10.2.5 COMMANDPROMPTSTEPS

This step collection also includes steps from the following collections: **CommonSteps**

Step	Type	Description
Given command prompt Username is $username	All	Sets username used to establish SSH connection. If system argument -Dusername is specified this step can be skipped.
Given command prompt Password is $password	All	Sets password used to establish SSH connection. If system argument -Dpassword is specified this step can be skipped.
Given command prompt Port is $port	All	Sets Port used to establish SSH connection. If system argument -Dport is specified this step can be skipped.
Then command prompt last command output contains $text	Basic	Verifies that the output of the last executed command contains specified text. If $text is set as NULL, this step is skipped.
Then command prompt last command output ends with $text	All	Verifies that the output of the last executed command ends with specified text. If $text is set as NULL, this step is skipped.
Then command prompt last command output regex contains $regEx	All	Verifies that the output of the last executed command contains specified regular expression.For more information on regular expressions check http://docs.oracle.com/javase/tutorial/essential/regex/char_classes.html
Then command prompt last command output regex variables in line $lineNumber are $regEx	All	Verifies that the output of the last executed command contains specified regular expression within given line number. Result of the regular expression is stored inside a list for later use. List of stored variables is incremental within one scenario. For more information on regular expressions check

		http://docs.oracle.com/javase/tutorial/essential/regex/char_classes.html
Then command prompt last command output does NOT contain $text	Basic	Verifies that the output of the last executed command does NOT contain specified text
Then command prompt wait for $seconds seconds	All	Waits until the execution of next step.
Given command prompt wait time is $wait seconds	All	Specifies amount of seconds during which the test will wait before starting the execution of the next step. Wait period can be specified through the waitSeconds system property.
Given command prompt steps are running in dry-run mode	All	Specifies amount of seconds during which the test will wait for the screen to load. This step can be specified using system property dryRun.
Given command prompt IP is $ip	All	Sets IP used to establish SSH connection.If system argument -Dip is specified this step can be skipped.
When command prompt new variable $variable is added	All	Adds new variable to the list.
When command prompt variable number $variableIndex is converted to date using following format $format	All	Converts variable with specified index to date using specified Java SimpleDateFormat.
Given command prompt prefix command is $prefixCommandValue	All	Sets prefix command that shall be added to the command. For example, if prefix command is "go oltp;" and the command itself is "/eso_release/deploy/host/hdeploy"following shell command shall be executed "go oltp; /eso_release/deploy/host/hdeploy"
Given command prompt default error keyword $keyword is added	All	Adds error keyword that shall be verified after the execution of each command.
Given command prompt default error keywords are NOT used	All	
Given command prompt credentials are reloaded	All	Reloads credentials.
Then command prompt last command output regex variables are $regEx	All	Verifies that the output of the last executed command contains specified regular expression. Result of the regular expression is stored inside a list for later use. List of stored variables is incremental within one scenario. For more information on regular expressions check http://docs.oracle.com/javase/tutorial/essential/regex/char_classes.html
When command prompt variable number $variableIndex is incremented by $increment	All	Increment variable with specified index.
When command prompt variable number $variableIndex is incremented by $increment with the limit of $limit	All	Increment last variable.

When command prompt last variable is incremented by $increment with the limit of $limit	All	Increment last variable.
When command prompt last variable is incremented by $increment	All	Increment last variable.
Given command prompt locale is $language $country	All	
Given property prefix is $prefix	All	Set property prefix. Example: If property prefix is este, ip property shall be esteIp.
Given command prompt property prefix is $prefix	All	

10.2.6 SHELLSTEPS

This step collection also includes steps from the following collections: **CommonSteps**, **CommandPromptSteps**

Step	Type	Description
Given Shell session timeout is $seconds seconds	All	Specifies amount of seconds the system should wait between clicking buttons.
Then Shell repeat last command $times times	All	Repeat last command.
When Shell command $newCommand is executed	Basic	Executes specified command. Execution of this step creates new SSH session using @ref setIp, @ref setUsername and @ref setPassword specified with Given steps. If prefix command is specified through @ref setPrefixCommand it shall be added before the actual command. If error keywords are specified through @ref setErrorKeywordList verification shall be performed after the command is executed. If $newCommand is set as NULL, this step is skipped.
When Shell do $action is executed	All	Script specified in @ref setDoScript shall be executed with one parameter $action.
When Shell do $action with $paramList is executed	Basic	Script specified in @ref setDoScript shall be executed with first parameter $action following with the rest of parameters. paramList must contain at least one parameter. Word "and" can NOT be used as part of argument. If argument has more than one word, whole argument must be enclosed within "" (i.e. loto_numbers "01 02 03 04 05 06 07") Multiple parameters must be divided by " and " (i.e. draw_number 34 and verbose). Example: When Shell do backup with cdc 3445 and label Yet another backup and verbose is executed
When Shell command $newCommand with $params is executed	All	Executes specified command with parameters. Execution of this step creates new SSH session using @ref setIp, @ref setUsername and @ref setPassword specified with Given steps. If prefix command is specified through @ref setPrefixCommand it shall be added before the actual

		command. If error keywords are specified through @ref setErrorKeywordList verification shall be performed after the command is executed. If $newCommand is set as NULL, this step is skipped.
Then Shell repeat last command for $seconds seconds or until output contains $text	All	Verifies that the output of the last executed command contains specified text. If $text is set as NULL, this step is skipped.
Given Shell do script is $script	Basic	Specifies script that will be used by @ref executeDoScript and @ref executeDoScriptWithParam
Given Shell output file is $file	All	Outputs results to the specified file.
Then Shell repeat last $count commands $times times	All	Repeat last commands.
Then Shell repeat all commands $times times	All	Repeat all commands.
When Shell command $newCommand is executed with variables	All	Similar to @ref executeCommand except that the Java style variables (String.format) within the command are replaced with values accumulated using @ref checkLastCommandOutputLineRegEx and @ref checkLastCommandOutputRegEx.

10.2.7 ALTURASTEPS

This step collection also includes steps from the following collections: **ShellSteps**, **CommandPromptSteps**, **CommonSteps**

Step	Type	Description
Given Altura standalone mode is $mode	All	Specifies whether Altura terminal is in the standalone mode.
When Altura reads barcode $value	All	Simulates barcode reader using hex value
When Altura serial number is entered	Basic	Puts serial number obtained previously. Buttons must be named as numbers (i.e. 7)
Given Altura serial number format is $format	All	Specifies serial number format expressed as regular expression.
Then Altura save the value of the serial number from the label with the ID $id	All	Saves value of the serial number from the label as internal variable that can be used later.
Then Altura save the value of the serial number from the label with the ID $id in line $line	All	Saves value of the serial number from the label in specified line as internal variable that can be used later.
Then Altura increment value of the date by $days days	All	Increments value of the date by specified number of days.
Given Altura missing element failures are NOT ignored	All	Disable errors in steps caused by a missing element (not found or not visible). This property is reset on every scenario
Given Altura wait between buttons click is $wait seconds	All	Specifies amount of seconds the system should wait between clicking buttons.

Given Altura ticket file name is $fileName	All	Defines file name format of printed tickets. Default value is ticket*.txt.
Then Altura screen name is $screenName in $number seconds	All	Waits for the terminal to reach the specified screen name in a specified number of seconds.
When Altura save value of the label with the ID $id as variable $variableName	All	Saves value of the label as internal variable that can be used later.
When Altura save value of the label with the ID $id as variable using RegEx $regEx	All	Saves value of the label as internal variable that can be used later. Regular expressions are used to evaluate variable.
When Altura command $command is executed	All	Executes Altura commands.
Given Altura snapshot is taken on failure	All	Snapshot will be taken when scenario fails.
Given Altura snapshot is NOT taken on failure	All	Snapshot will NOT be taken when scenario fails.
Given Altura snapshot is displayed	All	Snapshot will be displayed in a separate window.
Given Altura snapshot is NOT displayed	All	Snapshot will NOT be displayed in a separate window
Given Altura snapshot is taken on every command	All	Snapshot will be taken before every action.
Given{Altura snapshot is NOT taken on every command	All	Snapshot will NOT be taken before every action.
When Altura teller number is entered	Basic	Puts teller number obtained from system property tellerNumber.
When Altura teller password is entered	Basic	Puts teller password obtained from system property tellerPassword.
When Altura single buttons $buttons are clicked	Basic	Executes PUSH command separately for each button. For example, if buttons is Button1 Button2 Button3, following will be executed: PUSH Button1 PUSH Button2 PUSH Button3
Then Altura result contains $result	Basic	Verifies that the output of the last executed command contains specified text. If $text is set as NULL, this step is skipped.
Then Altura result is $result	Basic	Verifies that the output of the last executed command ends with specified text. If $text is set as NULL, this step is skipped.
Then Altura label $id contains $text	Basic	Verifies that the value of the label is correct. If $text is set as NULL, this step is skipped.
Then Altura label $id does NOT contain $text	Basic	Verifies that the value of the label is correct. If $text is set as NULL, this step is skipped.
Then Altura last printed ticket contains $text	Basic	Verifies that the value of the print is correct. If $text is set as NULL, this step is skipped. If $text is set as EMPTY, this step will verify that the specified line is empty.
Then Altura last printed ticket contains $text in line $line	Basic	Verifies that the value of the print is correct in specified line. If $text is set as NULL, this step is skipped. If $text is set as EMPTY, this step will verify that the specified line is empty.
Then Altura last printed ticket contains date	Basic	Verifies that the date of the printed ticket is correct.
Given Altura property $key is	All	Adds global properties

added with value $value		
When Altura reads last wager barcode	All	Reads last wager
When Altura save last wager barcode as variable	All	Puts last wager as variable
When Altura save last wager barcode to the file $fileName	All	Saves wager to the specified file
When Altura reads wager barcode from variable $number	All	Reads wager from the variable
When Altura reads wager barcode from file $fileName	All	Reads wager from the file
When Altura reads betslips $betslip	All	Reads betslips
Then Altura global property $property contains $value	All	
Given Altura is on the $id game screen	Basic	Following steps are executed: Goes to the home screen Verifies that the screen name is correct Clicks the game button Verifies that the screen name is correct. Following naming conventions must be followed: Home screen must be name HomeScreen Game button must be named [GAME]Button. [GAME] should be substituted with the name of the game (i.e. KenoButton) Game screen must be named [GAME]Screen. [GAME] should be substituted with the name of the game (i.e. KenoScreen)
Given Altura missing element failures are ignored	All	Disable errors in steps caused by a missing element (not found or not visible). This property is reset on every scenario
When Gemini hardware sensor $sensorAction is simulated	Basic	Performs a hardware sensor simulation on the Gemini (LVM) pc app ONLY! The sensorAction can be any of the actions listed in the pc app hardware sensor menu. Example: Main Door Open, Main Door Closed, etc.
Then Altura label $id is $text	Basic	Verifies that the value of the label is correct. If $text is set as NULL, this step is skipped. If $text is set as EMPTY, this step will verify that the specified line is empty.
Given Altura date format is $format using locale $locale	All	Specifies date format.
Then Altura save the value of the date from the label with the ID $id	All	Saves value of the date from the label as internal variable that can be used later.
Given Altura text printer is enabled	All	Switches to text printer.
Given Altura Java path is $path	All	Sets the Java path. Default value is /usr/local/java/jre/bin/java.
Then Altura screen name is $screenName	Basic	Waits for the terminal to reach the specified screen name. After time specified with @ref setWait expires test fails.
When Altura snapshot is taken	All	Takes snapshot.
Given Altura terminal is on the home screen	Basic	Moves terminal to the home screen.
When Altura button $id is	Basic	Executes PUSH command. If NULL or empty

clicked		string, this step shall not be executed.
When Altura number of credits are increased by $amount	Basic	Adds credits to Self Service (Gemini, Gamepoint, etc) terminal.
Then Altura button $id contains $text	Basic	Verifies that the value of the button is correct. If $text is set as NULL, this step is skipped.
Then Altura label $id contains date	Basic	Verifies that the value of the label contains date.
Then Altura wait for $wait seconds or until label $id contains $text	Basic	Verifies that the value of the label is correct. If $text is set as NULL, this step is skipped. Verification is repeated until the outcome is successfull or it times out.
Then Altura label $id contains $text in line $line	Basic	Verifies that the value of the label is correct in specified line.If $text is set as NULL, this step is skipped. If $text is set as EMPTY, this step will verify that the specified line is empty.
Then Altura button $id is in status $status	All	Verifies whether the Altura button is in the specified status. Currently supported statuses are: enabled disabled visible hidden