TESTING FOR PPC

Andrea Giuliani – 194352 – p90.cc@hotmail.it
Gabriele Di Rocco – 195037 – squall_l@live.it
Daniele Evangelista – 151125 – vandaniel83@libero.it
Valerio Piccioni – 223000 - vesparo@gmail.com

zioTesting diretly from zioFrank

INDICE

0. TESTING TEAM

Testing is not a random activity and requires a precise focus of the team on the campaign set . In this chapter of the preface we think it's important to make a nod to the composition of the team that has worked to carry out this work.

0.1 ORIGINE

Il team autore di questo documento e del lavoro di Testing cui fa riferimento è un sottoinsieme del team zioFrank precedentemente costituito per portare termine la fase di sviluppo del software PPC.
Essendo oggetto di questa campagna lo stesso software PPC e dovendo eseguire il testing in varie nature il team ha dovuto cercare di estraniarsi dal precedente elaborato.
Accettiamo quindi il paradosso che zioFrank sia diverso da zioTesting e procediamo al lavoro.

0.2 ENTRY POINT

Il punto di partenza della campagna è insieme all'analisi del prodotto da testare, senza dubbio uno degli aspetti cruciali del lavoro svolto, in quanto fondamentale per l'ottimizzazione delle risorse.
Il testing è di per se un operazione troppo spesso postposta e/o dalle aziende a causa dei costi elevati che comporta, poterla avviare da un punto di partenza più avanzato potrebbe costituire un notevole vantaggio.
Entrando nel merito del caso specifico, il team zioTesting ha ricevuto come base di partenza l'output elaborato da un membro del team di sviluppo (Vincenzo Battisti) avente operato BBTesting parziale sul software PPC.
L'idea iniziale era di sfruttare tale lavoro svolto e completarlo con una parte ad essa complementare, tuttavia ciò non è stato possibile in quanto le due campagne di testing proponevano quadrature dell'applicativo troppo differenti. Quella proposta dal nostro collega ignorava totalmente una grossa fetta del software collocando l'attenzione su una partizione troppo incompleta del programma e pertanto sebbene si possa ritenere valido il suo lavoro siamo costretti a non poterlo validare come punto di partenza per una campagna più approfondita quale quella che ci è stata richiesta.

0.3 RIPARTIZIONE E SOTTOGRUPPI

Dapprima il team ha deciso di operare su due fronti distinti per poi riunire i lavori svolti separatamente, tale scelta è stata subito riconsiderata vista la necessità di formarsi a livello accademico e di raggiungere quindi una visione di insieme comune che fosse comprensiva sia di WBT che di BBT.

Il lavoro pertanto è stato portato avanti da questo macroteam di 4 persone che di volta in volta si è preoccupato di riassegnare giornalmente nuovi compiti al singolo individuo o a sottoinsiemi del team in modo da favorire il lavoro in parallelo e soprattutto l'apprendimento utile a colmare il vuoto teorico sugli argomenti trattati.

1. PROGRAM DESCRIPTION

Please describe here the source code you want to use for testing purposes. The description shall be in terms of how the program behaves and a brief description of its classes.

1.1 PROGRAM BEHAVIOR

Micron commissiona a zioFrank lo sviluppo di un software capace di adempiere alcune funzioni di supporto alla catena di montaggio dell'azienda.
In questa sede ci occuperemo di testare appunto il suddetto software (che porta il nome di PPC).

Il profilo utente che verrà messo a contatto con l'app è quello di un utente business, formato a livello aziendale all'uso del programma, pertanto riteniamo superfluo eseguire dispendiosi test sulla user experience, sappiamo inoltre che in fase di sviluppo il team zioFrank ha già speso energie in tal senso.

Analizzando la documentazione fornita con l'eseguibile emerge come alcuni punti siano stati gestiti dal team zioFrank semplicemente istruendo l'utente ad un uso corretto del software, ma qualora qualcosa dovesse andare storto, la risposta del programma sarebbe accettabile? Le casistiche estreme sono gestite come ci si aspetta? Il codice è ben strutturato?
Descriviamone le caratteristiche principali di PPC:

- Il software deve connettersi ad un DB per poter funzionare
- Il software deve permettere all'utente di configurare la connessione con il DB opportuno
- Il software deve permettere di creare file .csv contenenti alberi
- Il software deve permettere l'upload degli stessi alberi su DB
- Il software deve permettere dei calcoli sugli attributi degli alberi
- Il software deve permettere l'aggiunta di nuovi attributi

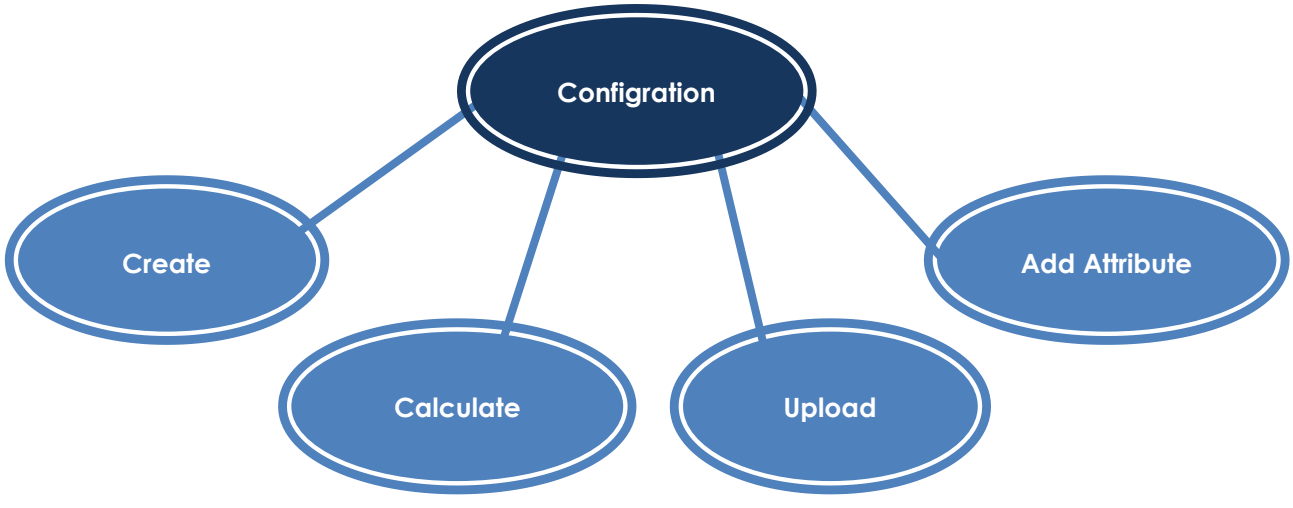
Chiaramente tale funzionamento è più accuratamente esplicitato nella documentazione che il team di sviluppo zioFrank ha prodotto e consegnato insieme all'eseguibile.
Alleghiamo pertanto, oltre all'eseguibile, anche la suddetta documentazione della quale si è tenuto estremamente conto durante la campagna di testing effettuata.
In questa sede ci dedicheremo quindi ad esplicitare solo sommariamente il funzionamento dell'applicativo ponendo l'accento invece sulle parti del software che riteniamo essere al centro del nostro interesse come tester, garantendo tuttavia la più ampia copertura del programma.
Il lavoro documentato in queste pagine prende come input di partenza l'elaborato di un collega (Vincenzo Battisti) e mira a migliorarne la qualità e la copertura.

1.2 Program structure

L'architettura dell'applicativo è quella di una WPF implementata in codice XAML e C#.
Suddividiamo in due macroaree la struttura del software:

- Un interfaccia grafica GUI
- Un ENGINE dedito a soddisfare gli aspetti funzionali dell'applicativo

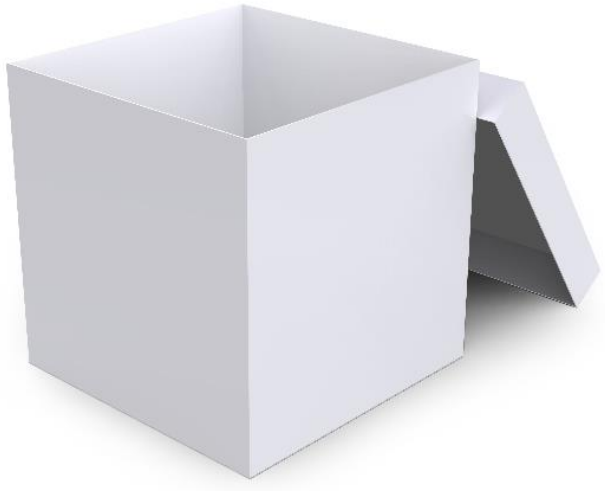
Mettiamo al centro della nostra campagna di testing la seconda macroarea in quanto, la GUI è stata realizzata con costrutti standard delle WPF e pertanto riteniamo possa risultare di scarso interesse investirvi risorse.
L'engine al contrario della GUI è responsabile delle funzionalità cardine del sistema e su di esso vogliamo concentrarci, complice anche il fatto che il motore con le funzionalità principali, possa essere interrogato anche da sistemi esterni indipendenti dalla GUI.
Ma come è fatto L'engine?



Nel grafo stilizzato qui sopra mettiamo in mostra come qualsiasi funzionalità del sistema sia di fatto dipendente dal fatto che la macchina sia o meno configurata. Questo rende la funzione "Configuration" propedeutica a tutte le alter; è evidente che un fallimento evidenziato in fase di configurazione comprometterebbe tutto il funzionamento del sistema.

Sappiamo inoltre che nessuna funzionalità del sistema è indipendente dalla connessione a DB pertanto eventuali guasti sul Server dove esso è salvato comporterebbero l'inutilizzabilità dell'intero software.

White Box



DISCLAIMER

La formazione del team circa l'esecuzione di WBTesting si circoscrive ad un ambito teorico della disciplina stessa, nonostante l'enorme sforzo di compensare anche la faccia pratica della medaglia, il team ha dovuto fronteggiare talune difficoltà legate alle tecnologie in gioco.
Non è stato infatti possibile trovare supporto sulle tecnologie precedentemente imposte da micron al team di sviluppo di PPC, ne un tool che rilasciasse licenze free per automatizzare tale procedimento MC/DC su C#.

Questa parte di lavoro si propone quindi, come suggerito in ultima istanza dal committente, di impostare solamente l'attività teorica di MC/DC senza portare a termine l'esecuzione dei casi di test.

2. WHITE BOX TESTING (OPTION 1)

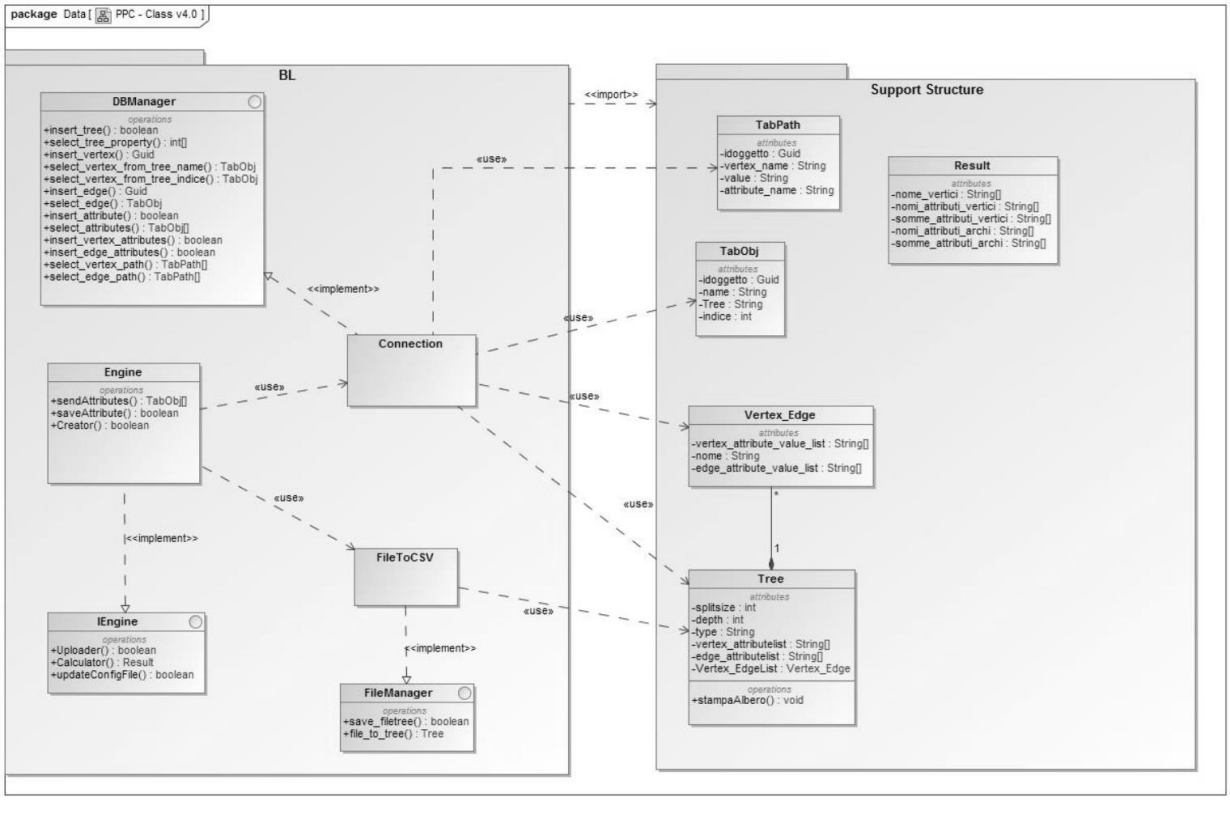
For those of you selecting Option 1 (white-box Testing) you shall provide here a
i) much more elaborated description of the program structure
ii) a detailed description of the coverage criteria used for selecting test cases
iii) a list of test cases you generated
iv) information on which amount of coverage you are going to achieve
v) the execution of the test cases and report on the results.

2.1 DESCRIPTION OF THE PROGRAM CODE

In this section, please provide a more detailed description of the program structure, in terms of its classes, complexity, etc.

Come già detto nei capitoli precedenti di questo documento, non prenderemo in analisi per questa operazione di testing le classi del codice ritenute più o meno standard. Trascureremo quindi tute le classi importate da librerie esterne o riguardanti il codice dell'interfaccia grafica, ritenuto di importanza marginale al fine del risultato che ci aspettiamo di ottenere.
Le tecnologie che sono dietro all'implementazione del programma sono:
- .NET framework
- Microsoft SQL
- XAML
- C#

Il tipo di applicazione sviluppata è un applicazione WPF.
Importiamo di seguito il Class Diagram del codice sorgente con relativo estratto di documentazione.



<< Nell'ultima fase del lavoro, prettamente di natura implementativa, si sono evidenziati nel dettaglio tutti gli aspetti ambigui residui, comportando una continua ristrutturazione del class diagram.

La stesura del codice ha condizionato questa continua rianalisi del diagram, mettendo alla prova la capacità da parte del team di coniugare flessibilità nelle modifiche e coerenza con ciò che era stato progettato precedentemente.

Questa metodologia si è mostrata necessaria dal momento che l'esperienza del team non permetteva una progettazione pienamente consapevole, maturata solamente al raggiungimento della fase implementativa.

Inoltre una rivisitazione del class risulta essere necessaria dal momento che il dettaglio raggiunto in questa fase è molto più a basso livello rispetto alla precedente.

La suddivisione dei due package principali BL e SUPPORT STRUCTURE permane nonostante alcune rilevanti modifiche, segno che il livello di dettaglio raggiunto nelle fasi precedenti ha effettivamente facilitato flessibilità e coerenza del sistema.

Nel package BL la classe Wrapper con le sue aggregazioni è stata sostituita dalla classe Engine che oltre a svolgere le azioni principali (create, upload e calculate) si occupa del passaggio degli attributi alla GUI e del salvataggio del tipo di file.

Sempre all'interno del package BL si è palesata la necessità di tre interfacce (DBManager, FileManager ed IEngine) che oltre a gestire i file albero prodotti, le connessioni a database e garantire l'accesso da parte di un sistema esterno all'engine, svolgono le funzioni del package Utility, rimosso poiché rivelatosi poco adatto al compito. Il PPC.dataset contenuto nel suddetto package realizzava le connessioni utilizzando una matching del DB in locale, caratteristica non contemplata nel nostro sistema. I dataset richiedevano maggiori risorse in termini di spazio, basandosi comunque su SQL commands. Si è optato dunque per l'utilizzo diretto di quest'ultimi.

In particolare, come richiesto da specifica, l'IEngine non da accesso alla funzione di creazione di alberi, ma solamente a quelle di upload e calcolo.

Per quanto riguarda il package Support Structure, persiste la composizione tra la classe Vertex_Edge e la classe Tree, mentre ulteriori classi di supporto quali TabPath e TabObj sono utilizzate dalle classe Connection rispettivamente per le query relative alla calculate path e per le select sulle tabelle Tree e Vertex_Edge.

La classe Result gestisce gli output della calculate path. >>

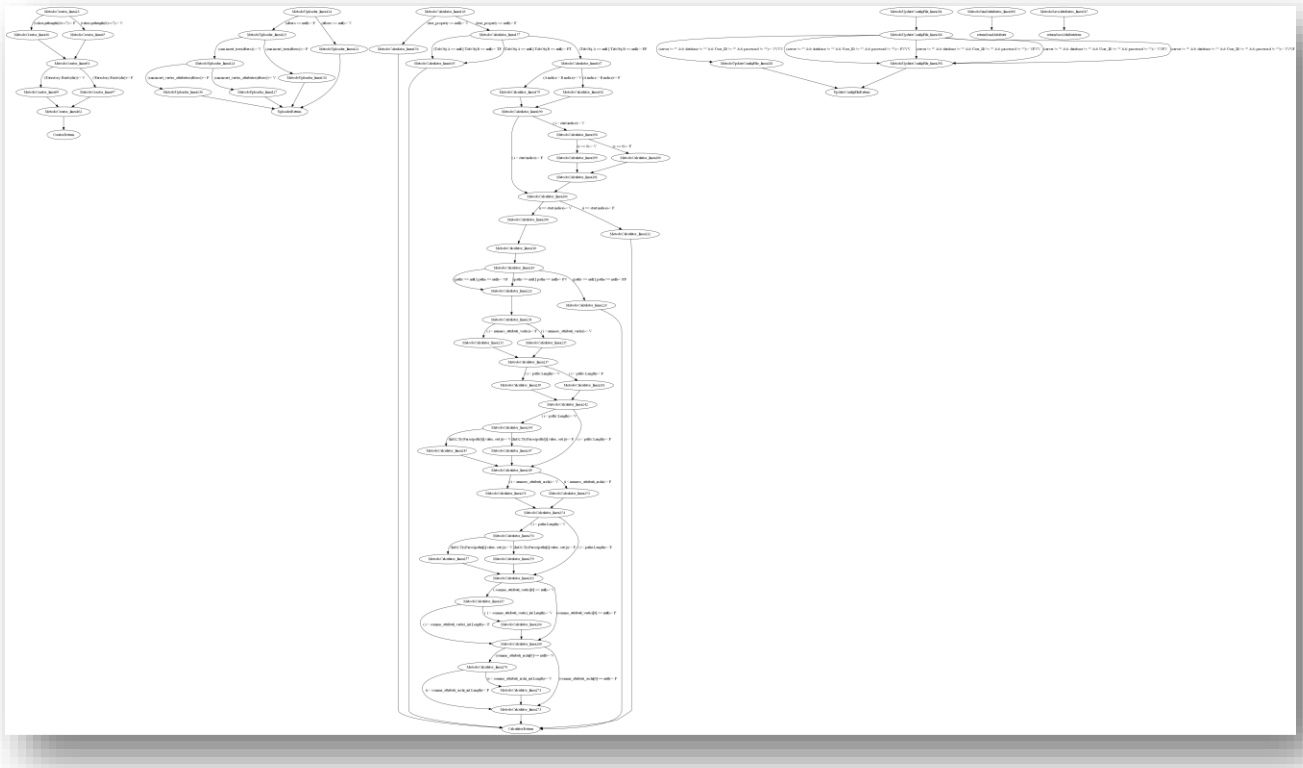
2.2 APPLICATION OF THE COVERAGE

In this section you may want to use an automated program to generate the program flow graph, such as Control Flow Graph Factory in Eclipse or any other available online (to check what available for C#).

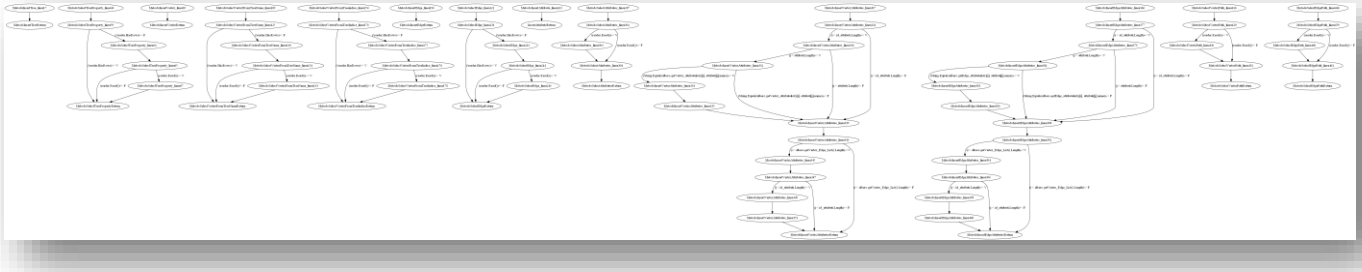
Dopo un attenta analisi del codice e delle esecuzioni attese dallo stesso, abbiamo provveduto ad individuare un grafo per la copertura del codice che rispetti lo standard MC/DC (letteralmente Modified Condition/Decision Coverage). Questo processo che dovrebbe essere automatizzato in qualche modo è stato eseguito a mano poiché Visual Studio non mette a disposizione tool MC/DC e gli unici compatibili online con C# come Testwell CTC++ sono a pagamento e non sono disponibili versioni trial a meno di contatto preventivo con l azienda produttrice. Abbiamo quindi dapprima scandagliato ogni riga di codice delle nostre classi dell engine cercando le guardie da controllare. A tal fine ci è venuto incontro l'utilizzo del tool MC/DC generator, che presa in pasto una guardia nel codice genera la relativa tabella di verità . Alcune classi come quelle di supporto o strutturali non contengono grafi significativi poiché i metodi non contengono guardie e la maggioranza dei metodi sono di tipo {get; set;}. Dunque riporteremo solo i grafi delle classi Engine, Connection e FileToCSV che sono le nostre classi chiavi.Successivamente con l'ausilio del tool GraphViz siamo stati in grado di disegnare i grafi MC/DC. Alleghiamo le immagini anche in file separati all 'interno di una cartella per permetterne una migliore leggibilità viste le dimensioni.

Di seguito i grafi dei metodi di ogni classe:

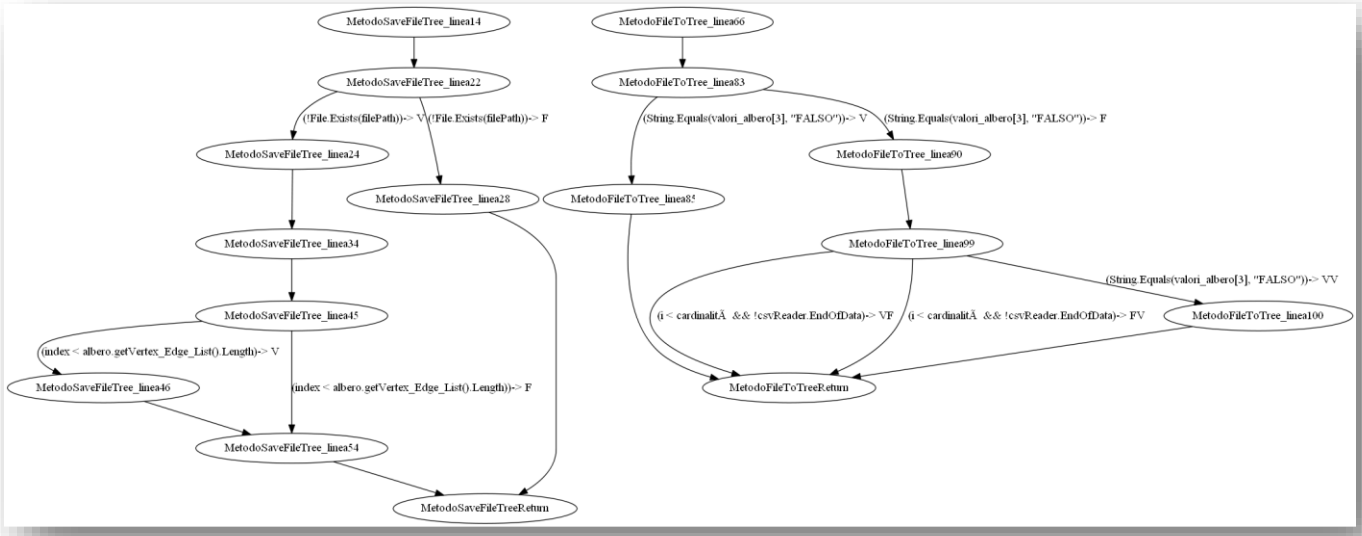
CLASSE ENGINE



CLASSE CONNECTION



CLASSE FILETOCSV



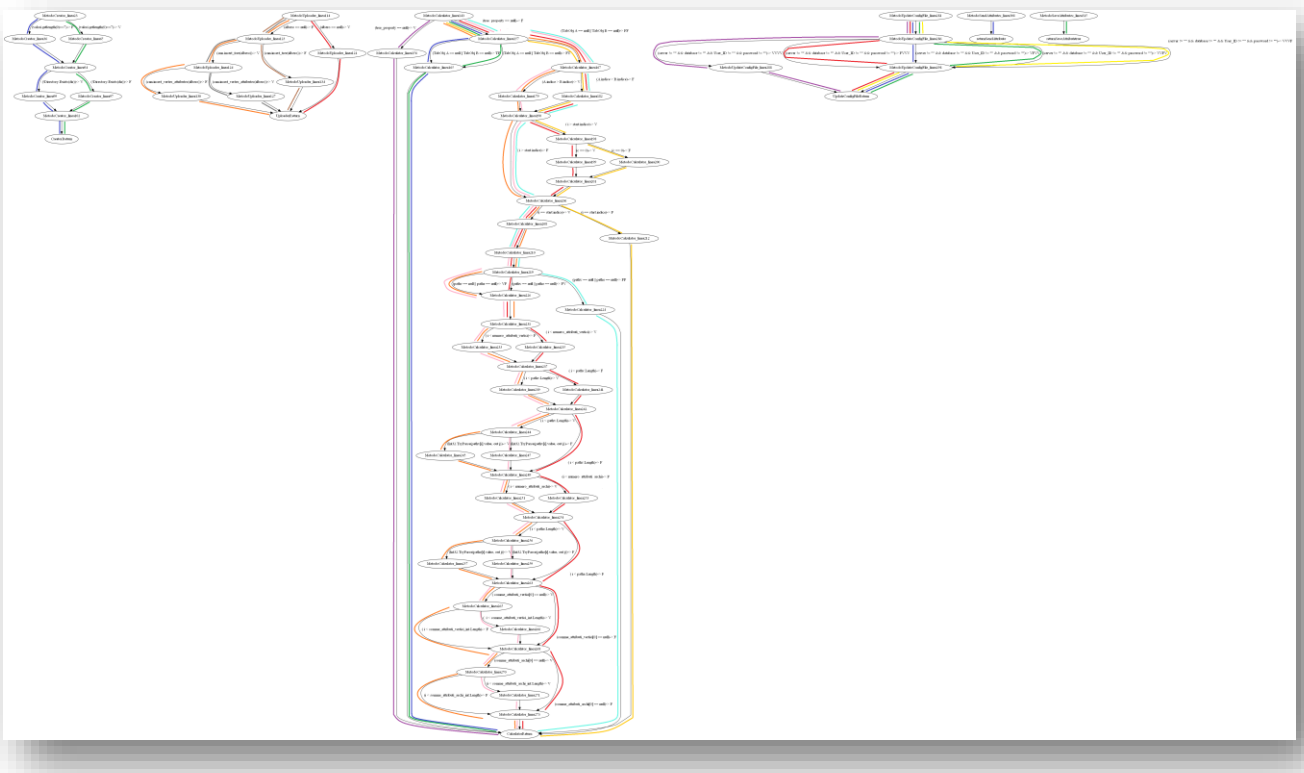
2.3 LIST OF GENERATED TEST CASES

Per creare dei test case per la copertura dei grafi il Team ha scelto un determinato numero di test che coprono la totalità degli archi (il 100% in una casistica ideale per ovvie ragioni non sarà possibile da raggiungere, specialmente nei casi in cui si tratti di archi che dipendono da guardie di cicli for e guardie di tabelle mc e non da input).

Dunque ci aspettiamo che diversi casi di test non siano eseguibili e dunque il coverage realmente raggiungibile sia inferiore. Successivamente il team avrebbe dovuto cercare altri percorsi del grafo (test) possibili da eseguire per ottenere un adeguata copertura.
Non potendo eseguire tali test nella pratica a causa della mancanza di un tool, limitiamo l inserimento di alcuni possibili percorsi che garantirebbero la copertura totale come esempio di tentativo di coverage .

Troviamo utile anche in questo caso un esposizione grafica con dei colori dei testcase con immagini (sempre per favorire la leggibilità, troviamo tali immagini allegate anche in file separati in allegato).

Nelle pagine che seguono mostriamo e documentiamo tali test case (ci riserviamo di snellire il testo da descrizioni di casi troppo banali).



Metodo Creator

blu = Test[valori.getlength (0) != 7, (La creazione della directory va a buon fine)]
verde = Test[valori.getlength (0) ==7, (La creazione della directory fallisce)]

Metodo Uploader

Arancione= Test[albero!=null,(il metodo insertTree va a buon fine),(il metodo InserVertexAttribute fallisce)]
Grigio= Test[albero!=null, ,(il metodo insertTree va a buon fine),(il metodo InserVertexAttribute va a buon fine)]
Marroncino= Test[albero!=null, ,(il metodo insertTree fallisce),(il metodo InserVertexAttribute fallisce)]
Rosso= Test[albero==null]

Metodo Calculator

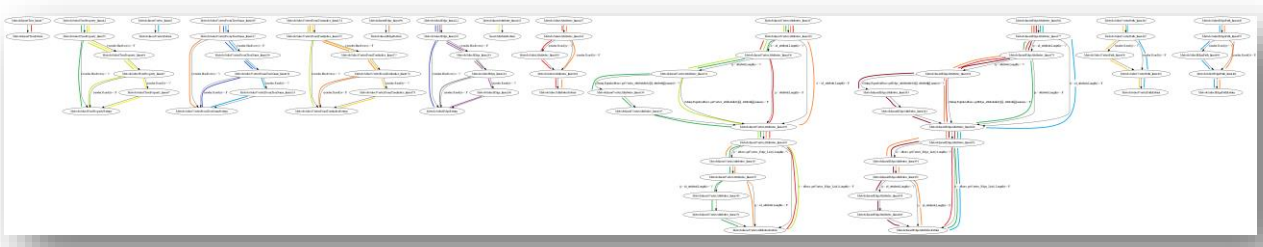
Viola= Test[Tree_property==null]
Arancione= Test[Tree_property!=null, (TabObj A!=null ,TabObj B!=null),A.indice>B.indice, i<=start.indice, i== start.indice, (pathv==null, pathe!=null),i>=numero_attributi_vertici, i< pathv.length,(pathv[i].value è convertibile ad intero),i< numero_attributi_archi,i<pathe.length, (pathe[i].value è convertibile ad intero), somme_attributi_vertici[0]==null, i>=somme_attributi_vertici_int.length, somme_attributi_archi[0]==null, i>=somme_attributi_archi _int.length]
Giallo= Test[Tree_property!=null, (TabObj A!=null ,TabObj B!=null),A.indice<=B.indice, i>start.indice, c!=0, i!= start.indice]
verde = Test[Tree_property!=null, (TabObj A!=null ,TabObj B==null)]
blu = Test[Tree_property!=null, (TabObj A==null ,TabObj B!=null)]]
Rosso= Test[Tree_property!=null, (TabObj A!=null ,TabObj B!=null),A.indice<=B.indice, i>start.indice, c==0, i== start.indice, (pathv!=null, pathe==null),i<numero_attributi_vertici, i>=pathv.length, i>= numero_attributi_archi,i>=pathe.length, somme_attributi_vertici[0]!=null, somme_attributi_archi[0]!=null]

Rosa= Test[Tree_property!=null, (TabObj A!=null ,TabObj B!=null),A.indice>B.indice, i<=start.indice, i== start.indice, (pathv==null, pathe!=null),i>=numero_attributi_vertici, i< pathv.length,(pathv[i].value non è convertibile ad intero),i< numero_attributi_archi,i<pathe.length, (pathe[i].value non è convertibile ad intero), somme_attributi_vertici[0]==null, i<somme_attributi_vertici_int.length, somme_attributi_archi[0]==null, i<somme_attributi_archi _int.length]
Celeste= Test[Tree_property!=null, (TabObj A!=null ,TabObj B!=null),A.indice<=B.indice, i<=start.indice, i== start.indice, (pathv!=null, pathe!=null)]

Metodo Update Config File

Viola= Test[server != “”, database!=””,User_ID=!””,password!=””]
Rosso= Test[server == “”, database!=””,User_ID=!””,password!=””]
blu = Test[server != “”, database==””,User_ID=!””,password!=””]
Giallo= Test[server != “”, database!=””,User_ID==””,password!=””]
verde = Test[server != “”, database!=””,User_ID!=””,password==””]

TESTCASE CONNECTION



Metodo Select Tree Property

verde = Test[(il reader non ha trovato valori)]
Giallo= Test[(il reader ha trovato valori),(il reader ha ancora valori da leggere)]
verde chiaro= Test[(il reader ha trovato valori),(il reader non ha più valori da leggere)]

Metodo Select Vertex From Tree Name

Arancione=Test[(il reader non ha trovato valori)]
Celeste=Test[(il reader ha trovato valori),(il reader ha ancora valori da leggere)]
blu =Test[(il reader ha trovato valori),(il reader non ha più valori da leggere)]

Metodo Select Vertex From Tree Indice

verde chiaro= Test[(il reader non ha trovato valori)]
Arancione=Test[(il reader ha trovato valori),(il reader ha ancora valori da leggere)]
Giallo=Test[(il reader ha trovato valori),(il reader non ha più valori da leggere)]

Metodo Select Edge

blu =Test[(il reader non ha trovato valori)]
Grigio=Test[(il reader ha trovato valori),(il reader ha ancora valori da leggere)]
Viola=Test[(il reader ha trovato valori),(il reader non ha più valori da leggere)]

Metodo Select Attributes

Rosso=Test[](il reader ha ancora valori da leggere)]
Arancione=Test[(il reader non ha più valori da leggere)]

Metodo Select Vertex Path

Giallo=Test[](il reader ha ancora valori da leggere)]
blu =Test[(il reader non ha più valori da leggere)]

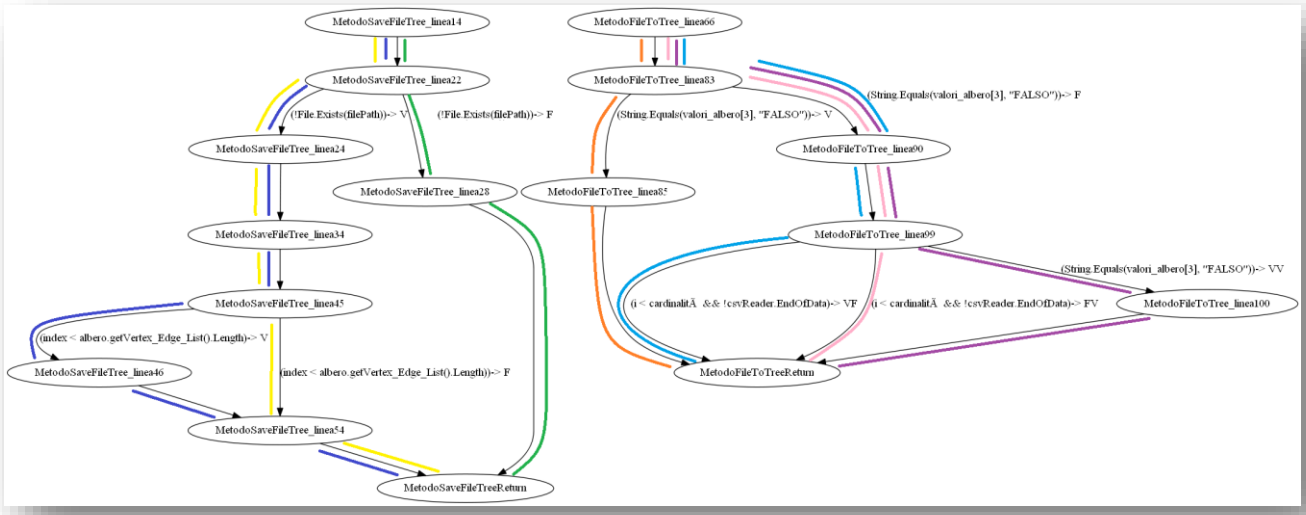
Metodo Select Edge Path

blu =Test[](il reader ha ancora valori da leggere)]
Arancione=Test[(il reader non ha più valori da leggere)]

Metodo Insert Vertex Attribute

Arancione=Test[i>=id_attributi.length, i<albero.getVertex_Edge_List().length,j>=id_attributi.length]
Rosso=Test[i<id_attributi.length,j>=attributi.length, ,i>=albero.getVertex_Edge_List().length]
verde chiaro= Test[i<id_attributi.length,j<attributi.length,
albero.getVertex_attributelist()[i]!=attributi[j].name ,i>=albero.getVertex_Edge_List().length]
verde = Test[i<id_attributi.length,j<attributi.length,
albero.getVertex_attributelist()[i]==attributi[j].name
,i<albero.getVertex_Edge_List().length,j<id_attributi.length]

TESTCASE FILETOCSV



Metodo Save File Tree

blu = Test[(il file non esiste), index< albero.getVertex_Edge_list().length]
Giallo= Test[(il file non esiste), index>= albero.getVertex_Edge_list().length]
verde = Test[(il file non esiste)]

Metodo File To Tree

Arancione= Test[valori_albero[3]=="Falso"]
Celeste= Test[valori_albero[3]!="Falso", i <cardinalità, (il lettore di file CSV ha raggiunto la fine del file in lettura)]
Rosa= Test[valori_albero[3]!="Falso", i >=cardinalità, (il lettore di file CSV non ha raggiunto la fine del file in lettura)]
Viola= Test[valori_albero[3]!="Falso", i <cardinalità, (il lettore di file CSV non ha raggiunto la fine del file in lettura)]

Black Box

DISCLAIMER

Per ragioni accademiche il team di testing che è andato ad eseguire il Black Box testing sull'app PPC è lo stesso team che si è occupato dello sviluppo e dell'implementazione della stessa, pertanto si è andati contro allo spirito del testing a scatola nera.
In fede alla coerenza di tale metodologia, il team si è riservato di simulare una condizione di ignoranza del codice.

3. BLACK-BOX TESTING (OPTION 2)

For those of you selecting Option 2 (black-box Testing) you shall provide here a
i) much more elaborated description of the program behavior, with a functional specification you are going to use to test the system
ii) a detailed description of the Category Partition method application to your system
iii) a list of test specifications you generated
iv) a list of generated test cases
v) the execution of the test cases and report on the results.

3.1 SPECIFICATION OF THE PROGRAM BEHAVIOR

In this section, please provide a detailed specification of the program behavior, to be used for testing purposes.

Agganciandoci a quanto detto precedentemente circa il comportamento atteso dal Software estraiamo nel dettaglio, dalla specifica e dall'interfaccia proposta dall'applicativo, quello che è il comportamento ATTESO del programma in esecuzione. Astraendoci completamente dai dettagli implementativi.
Sappiamo che l'utente che si porrà davanti a PPC dovrà prima di tutto configurare i parametri di connessione al DB.
Successivamente ci aspettiamo molteplici utilizzi DISTINTI:
Creazione di nuovi file contenenti alberi tramite compilazione di un apposita Form, aggiunta di nuovi attributi al DB, upload degli stessi alberi su DB ed interrogazione del DB al fine di eseguire calcoli sugli attributi degli stessi alberi inseriti. Risulta evidente che sebbene non vi sia nessun obbligo per l'utente di svolgere queste 3 operazioni esattamente in sequenza, non avrebbe senso tentare di interrogare il DB circa un albero che non è mai stato inserito, al contempo non avrebbe senso tentare di eseguire l'upload di un albero il cui file non è mai stato generato. Da questo e dai numeri raccolti nella documentazione del software riusciamo ad intuire una proiezione sui dati di utilizzo del software stesso.

A QUESTO PUNTO IL GRUPPO DI TESTING STUDIA UN PO' IL SUO "NEMICO"...

Memori dei numeri e delle stime che Micron ha fornito al team di sviluppo dell'applicativo, indichiamo con Ax.funzione gli accessi dell'utente a quella determinata funzionalità e stabiliamo alcuni vincoli:

1. Non può essere eseguito l'upload di un file se prima il file non è stato creato [Ax.Upload <= Ax.Create]
2. Non può essere eseguita un interrogazione su un albero che non è stato caricato su DB tuttavia, una volta eseguito tale upload ci si aspetta che le interrogazioni siano ben più di una per ogni albero caricato, altrimenti il software perderebbe di significato [Ax.Calculate > (Ax.Upload || Ax.Create)]
3. E' inverosimile credere che si acceda a PPC per svolgere una sola azione ma è credibile supporre che la stringa di configurazione venga settata una sola volta per ogni lancio dell'eseguibile [(Ax.Calculate + Ax.Upload + Ax.Create) > Ax.Configuration]
4. Il numero di attributi in gioco è talmente piccolo (5 o 6) che la funzionalità di Aggiunta di un nuovo attributo risulta quasi una formalità tanto saranno rare le volte in cui verrà chiamata in gioco [Ax.NewAttr = epsilon]

Possiamo quindi tradurre tali dati in una stima percentuale degli accessi per ogni funzione:

- 1% - Configuration
- 0,1% - NewAttr
- 9,9% - Create
- 9% - Upload
- 79% - Calculate

!!! ATTENZIONE !!!
Tali dati rappresentano una STIMA molto approssimativa di quelle che potranno essere le volte in cui un utente "cada" nell'esecuzione di una funzione piuttosto che di un'altra, NON stiamo assolutamente stabilendo una gerarchia di importanza.
Tuttavia se sappiamo che un utente tende statisticamente a "stressare" di più la funzione Calculate sappiamo anche che aumenterà la molteplicità degli input e che sarà più probabile che da tale funzione emergano fallimenti.

3.2 APPLICATION OF THE CATEGORY PARTITION METHOD

&&

3.3 LIST OF GENERATED TEST SPECIFICATIONS

In this section please describe how you applied the Category Partition method to the selected program. This section is the core of this assignment, so be sure to make it informative.

Si intende testare le funzionalità descritte nel paragrafo precedente tramite l'utilizzo del **Category Partition Method**, un metodo sistematico per la selezione di dati per il test, che parte dall'analisi delle specifiche del sistema e produce test script, attraverso una precisa serie di step di decomposizione:

- **Fase 1**: Identificare le Independently Testable Features (ITF);
- **Fase 2**: Identificare categorie tra le unità funzionali;
- **Fase 3**: Partizionare le categorie in scelte;
- **Fase 4**: Identificare vincoli tra le scelte;
- **Fase 5**: Produrre/Valutare una specifica per i test case;
- **Fase 6**: Generare test cases dalla specifica.

L'intero processo verrà adeguato alla sintassi del tool utilizzato per l'automazione del passo 6 (**TSL Generator**).

La presente documentazione si propone come un testo autosufficiente per la comprensione dell'elaborato, tuttavia basando il nostro lavoro sull'operato degli sviluppatori del tool TLS GENERATOR è doveroso citare il materiale da cui abbiamo attinto e da cui sarebbe eventualmente possibile un approfondimento.

Material: https://github.com/alexorso/tslgenerator

Manual: https://github.com/alexorso/tslgenerator/blob/master/Docs/TSLgenerator-manual.txt

LA PRIMA FASE È STATA QUINDI QUELLA DI IDENTIFICARE LE ITF DEL NOSTRO SISTEMA.

La nostra analisi ci ha portato ad identificarne 5 di cui per altro abbiamo ampiamente parlato fino ad ora.

Cercheremo di fare un piccolo riassunto per affrontare la lettura dei prossimi paragrafi con più agilità:

1. Configuration
2. Create Tree
3. Upload Tree
4. Calculate Path
5. Add new Attribute

La prima funzione, come sappiamo è quella tramite la quale riusciamo a configurare la stringa di connessione al DB. Poiché nessuna delle altre funzionalità ha senso senza una corretta connessione al DB, capiamo da soli quanto sia il caso di testarla indipendentemente ed in combo con le altre.

La seconda funzione è la responsabile della creazione del file .csv che successivamente verrà editato ed uppato. E' importante verificare che tale funzioni generi file coerenti con le aspettative affinchè anche le funzioni successive non siano inficiate da eventuali malfunzionamenti di Create Tree.

La terza ITF è quella che riguarda il caricamento di un albero su database. Essa dipende sia dagli input inseriti da tastiera che dalle condizioni del file che si tenta di uppare, si tenterà di testare ogni possibile condizione in modo tale da individuare eventuali gravi fallimenti.

La quarta funzione è ovviamente quella di interrogazione del DB per l'esecuzione di calcoli sugli alberi, trovare Fail su questa funzionalità potrebbe voler significare far morire la maggior parte delle esecuzioni del programma. E' molto importante che si arrivi ad eseguire questa operazione sicuri di aver pulito il più possibile il software da bug.

L'ultima ITF è quella, molto poco utilizzata ma che potrebbe generare fallimenti delicati sul DB, di aggiunta di nuovi attributi. I controlli su di essa non sono molti e sono abbastanza standard.

A QUESTO PUNTO IDENTIFICHIAMO LE CATEGORIE E DIAMONE UNA BREVE DESCRIZIONE, SEMPRE SEGUENDO LA SINTASSI SUGGERITA DAL TOOL TSL:

```
String category:
        zero length. [error]
        one or more char. [property settled]
        spaces. [single]
        special char. [single]
```

Innanzitutto suddividiamo la categoria di tutte le possibili stringhe: esse possono essere vuote, avere uno o più caratteri, contenere spazi oppure caratteri speciali.

Ovviamente, la proprietà associata alla stringa vuota sarà errore; in caso contrario sarà settled.

La presenza di eventuali spazi o caratteri speciali costituirà casi da valutare singolarmente, non in combinazione dunque con altre ITF.

CONFIGURATION:

```
        Strings Depth content:
                all match with data on server. [if settled] [property connection]
```

La configurazione con il server avviente ogni volta che c'è scambio di dati. Se viene riscontrata la guardia settled, è associata la proprietà connection.

CREATE TREE:

```
        Strings Depth content:
                spaces. [error]
                char.   [error]
                lesser than two(negative too). [error]
                two. [if settled] [property MinDepth] [else] [error]
                lesser than logarithm on base splitsize of (2000000-1)multiplied for
                splitsize, then minus one. [if settled] [property MediumDepth]
                [else][error]
                nineteen. [if settled] [property MaxDepth] [else] [error]
                more than nineteen. [if settled] [property OverDepth] [else] [error]
```

Nella creazione di un albero identifichiamo le possibili categorie di input della stringa contenuta nella Depth.

Spazi e caratteri ovviamente non sono ammessi, generando errore.

Da documentazione sappiamo anche che non possono essere inseriti nemmeno valori inferiori a due (alberi di grandezza negativa non hanno senso, e alberi con un unico elemento, ossia la

radice, non avrebbe senso nel contesto produttivo.

Il valore "2" è ritenuto il minimo possibile associato alla Depth; una Depth statisticamente ritenuta media viene ricavata tramite un'espressione particolare, mentre il valore massimo

previsto non deve superare il valore "19"

```
String Splitsize content:
        spaces. [error]
        char. [error]
        zero.[error]
        two. [if settled && OverDepth] [error] [else] [property MinSplitsize]
        between two and one thousand. [if settled && (OverDepth ||
        MaxDepth)][error] [else] [property MediumSplitsize]
        one thousand. [if settled && !MinDepth] [error] [else] [property
        MaxSplitsize, single]
```

Discorso analogo può essere fatto per la SplitSize, con guardie opposte a quelle della Depth, per ovvi motivi di combinazione delle due proprietà associate ad un albero.

Infatti, i vincoli riguardanti la grandezza massima di un albero impongono che i valori di SplitSize e Depth siano inversamente proporzionali (per rientrare nel limite dei 2000000 di nodi

massimi per albero).

Da notare i valori medi e massimo della SplitSize dunque in combinazione con valori medi e minimi della Depth al fine di preservare il vincolo sul numero massimo di nodi generabile.

```
String Type Content:
        unmatch with data on server. [if settled][property typeOk][else][error]
```

Il type di albero deve ovviamente essere "nuovo", ossia non presente sul DB e nel filesystem, altrimenti lo scenario è associato a errore.

```
Lists of attributes length:
        zero length. [error]
        only one attribute selected. [if settled && connection] [property
        OneAttr, single] [else] [error]
        special attribute NOattribute selected. [if settled && !OneAttr &&
        connection] [property NoAttr, single] [else] [error]
        greater than one.[if settled && !NoAttr && !OneAttr && connection]
        [property MoreAttr]
```

In caso di lista attributi vuota ovviamente è associata la proprietà errore.

In caso di singolo attributo selezionato, occorre verificare che venga settato e ci sia connessione, per associare singolo attributo, da valutarsi singolarmente. L'alternativa è

l'associazione di errore.

Lo scenario di selezione di NOattribute, per andare a buon fine, esige la combinazione dei casi selezionato, nessun attributo e connessione attiva.

Per numero di attributi superiore a uno, le guardie assicurano l'assenza della condizione "nessun attributo selezionato"

```
Radioboxes Rulegenerations content:
    default value. [if settled] [property Default] [else] [error]
    random value.  [if settled && !Default][property RangeOnly][else][error]
```

Il Radioboxes ovviamente distingue gli scenari default e random, verificando opportunamente le condizioni Default e non Default, essendo i due scenari mutualmente esclusivi.

```
String Ranges value:
    zero. [if settled && RangeOnly][property RangeZeroValue,
    single][else][error]
    greater than zero. [if settled && RangeOnly] [property
    RangeGreaterThanZeroValue][else] [error]
    lesser than zero. [if settled && RangeOnly] [property
    RangeLessThanZeroValue, single] [else] [error]
```

La stringa Ranges che può assumere valori ==0, >0 e <0, è opportunamente controllata dalle guardie che ne verificano la correttezza.

UPLOAD TREE:

```
String Filename content:
    match with existing file. [error]
    unmatch with existing file. [if settled && connection] [property
    UnmatchWithFile] [else] [error]
    match with existing tree on DB. [if settled] [error]
    unmatch with existing tree on DB. [if settled && connection] [property
    UnmatchWithTree] [else] [error]
```

Nell'Upload Tree, come prima cosa, viene controllata l'esistenza di un file esistente con lo stesso nome dell'albero da inserire.

In seguito, al fine di assicurare la corretta esecuzione, viene controllato che si tratti di salvare un albero su filesystem o di caricare i dati di un file su database, a patto

che anche su DB non esista un albero con lo stesso nome.

CALCULATE PATH:

Strings content:

match with data on DB. [if settled && connection] [property matchWithData] [else] [error]

unmatch with data on DB. [if settled] [error]

vertex A branch differs from vertex B branch. [if settled] [error]

vertex B preceedes vertex A. [if settled && connection] [property BpreceedesA] [else] [error]

Viene controllato in primis che esista un albero che corrisponda al valore della stringa inserita; in seguito viene controllato che i due vertici di riferimento per il calcolo

esistano e siano connessi tramite path di edge.

Come ultimo controllo, si valuta la precedenza dei due vertici l'uno rispetto all'altro.

ADD ATTRIBUTE:

String AttrName content:

unmatch with data on DB. [if settled && connection] [property AttrUnmatchWithData, single] [else] [error]

L'Add Attribute verifica l'esistenza di un attributo già esistente su DB.

3.4 GENERATED TEST CASES

Al fine di non compromettere la leggibilità del documento l'intera lista dei test case generati dal tool possiamo trovarla in allegato al documento nel file ***spec.tsl***

3.5 TEST CASE EXECUTION REPORT

Prendendo in analisi I test case generate il team ha speso una buona quantità di tempo nell'analisi di quelli che potevano essere i casi più interessanti da testare.

Sappiamo infatti che un buon tester non è quello che testa tutti i casi possibili, ma è quello che riesce a trovare un maggior numero di fallimenti con un minor dispendio di risorse possibile.

Il team zioTesting ha scelto di selezionare un 10% dei casi possibili per poi eseguirli e registrare un report di seguito tabellato.

Test Case #ID	Risultato Atteso	Risultato Riscontrato	Esito
1	Error	Error	Ok
2	Succ	Succ	Ok
3	Succ	Succ	Ok
6	Error	Crash	Fail
12	Error	Crash	Fail
13	Error	Succ	Fail
17	Error	Succ	Fail
18	Error	Succ	Fail
51	Error	Error	Ok
74	Error	Error	Ok
86	Error	Error	Ok
96	Error	Error	Ok
101	Succ	Succ	Ok
143	Error	Error	Ok
150	Succ	Succ	Ok
159	Error	Error	Ok
164	Succ	Succ	Ok
172	Succ	Succ	Ok
184	Succ	Succ	Ok
187	Succ	Succ	Ok
198	Succ	Succ	Ok
206	Succ	Succ	Ok
227	Error	Error	Ok
232	Succ	Succ	Ok
255	Succ	Succ	Ok

Test Case #ID	Risultato Atteso	Risultato Riscontrato	Esito
275	Error	Error	Ok
282	Succ	Succ	Ok
288	Succ	Succ	Ok
318	Succ	Succ	Ok

Dal report dei casi di testing eseguiti emergono dati importanti.

Come prevedibile, grazie anche alla legge di Pareto, pur avendo sparato le nostre cartucce su un po' tutta la superficie dell'eseguibile, una porzione di software che si dimostra vulnerabile ad un fallimento, tipicamente poi si rivela vulnerabile in quella zona a numerosi altri fallimenti. Questo si evince anche dalla tabella qui in alto.

Altro dettaglio importante è che possiamo stabilire una sorta di gerarchia tra i fallimenti trovati (ovviamente tale gerarchia ha una validità puramente indicativa poiché non dimostra assolutamente come un bug sia concretamente più urgente da correggere di un altro) notiamo come infatti il programma alcune volte si limiti a bloccare un operazione che doveva andare a buon fine, mentre altre volte vada letteralmente in crash, quello che tuttavia ci preoccupa sono le operazioni che ci aspettavamo terminassero in errore e che invece riescono ad essere portate a buon fine (vedi Test case 13, 17 e 18); abbiamo ragione di credere che tali casistiche potrebbero arrecare seri danni alla macchina su cui il software stia girando.

Il team di testing si riserva pertanto di render presente questo dato al team di sviluppo che probabilmente dovrà spendere del tempo sul codice prima di mettere PPC sul mercato.

Ironia della sorte vuole che in questo caso il team di testing ed il team di sviluppo siano composti dallo stesso pool.