Manutenzione e Reverse Engineering

Riferimenti

- Sommerville, Ingegneria del Software, 8a ed., Capitolo 21

Ulteriori Letture Raccomandate

- Grady Booch, Nine Things you can do with old software, IEEE Software, Sept/Oct 2008
- Canfora, Di Penta “Frontiers of Reverse Engineering: a Conceptual Model”, FOSM08 – IEEE Comp. Soc. 2008

La Manutenzione del software- generalità

- Qualsiasi software, dopo il rilascio della prima release, avrà bisogno di essere modificato.
- I sistemi software sono, per loro natura, sistemi evolutivi che cambiano durante la loro vita.
- Ciò deriva dal fatto che, durante la vita del sistema, le caratteristiche che lo definiscono cambiano, cambiando sia le esigenze di chi usa il software, sia dell'ambiente del mondo reale in cui il sistema opera.
  - Più I requisiti del sistema sono instabili, o specificano un problema in maniera incompleta ed approssimativa, più il sistema avrà bisogno di cambiare.

- L'evoluzione di un software è dunque inevitabile!

Motivazioni per il cambiamento

- Errori possono essere individuati e devono essere corretti;
- Il dominio del software può evolvere;
- Nuovi requisiti possono emergere dopo il rilascio;
- Nuove tecnologie hardware e software possono affermarsi nel frattempo;
- Può essere necessario migliorare la qualità del software (ad esempio l'affidabilità o le performance).

L’importanza dell’evoluzione

- Le organizzazioni proprietarie hanno fatto grossi investimenti per I loro sistemi software, che sono risorse critiche!
- Per preservare il valore di tali risorse, I sistemi devono necessariamente cambiare ed evolvere.
- La manutenzione è un’attività costosa e la maggior parte del budget speso per il software è in genere speso per la sua manutenzione, piuttosto che per il suo sviluppo.

I costi della manutenzione

- Un tipico progetto di sviluppo software mediamente dura tra 1 e 2 anni, mentre…

- La durata del periodo di manutenzione può variare tra 5 e 6 anni [1]
  - Più della metà dei costi di un progetto software sono spesi per la manutenzione
  - Recenti survey riportano la regola dell' 80-20, ossia 80% di sforzo speso per la manutenzione e 20% per lo sviluppo.

- [1] Parikh and Zvegintzov, Tutorial on Software Maintenance, IEEE, 1993

Evoluzione o Declino?

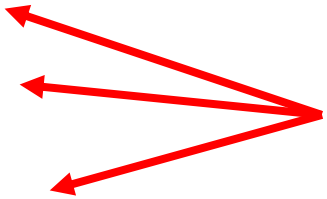
- Fino a che punto si può continuare a far evolvere un sistema software?
- Quando si deve decidere di gettare il vecchio sistema e sostituirlo con uno nuovo?

- Alcune domande da porsi:
  - Il costo di manutenzione è troppo alto?
  - L'affidabilità del sistema è inaccettabile?
  - Non si riesce più ad adattare il software in tempi accettabili?
  - Le prestazioni sono inaccettabili?
  - Le funzionalità del sistema sono poco utili?
  - Ci sono altri sistemi che fanno lo stesso lavoro meglio, più velocemente ed economicamente?
  - Il costo di manutenzione dell'hardware è diventato tale da giustificare la sostituzione con nuovo hardware?

Leggi dell'evoluzione del software

- Proposte da Lehman e Belady a partire dal 1976, in seguito a studi empirici basati inizialmente sull'osservazione dell'evoluzione di 4 versioni successive di un S.O. IBM
  - Modifiche continue
  - Complessità crescente
  - Evoluzione dei grandi sistemi
  - Stabilità organizzativa
  - Conservazione della familiarità
  - Crescita continua
  - Qualità deteriorata
  - Sistema feedback

leggi iniziali



leggi più recenti

Le Leggi di Lehman (1974)


Manny Lehman
*1925 †2010

- Modifiche continue
  - Un sistema deve necessariamente cambiare, o diventera progressivamente inutile.
- Complessità crescente
  - Quando un sistema viene modificato, la sua struttura si deteriora: per evitare ciò bisogna investire sulla manutenzione preventiva.
- Evoluzione dei grandi sistemi
  - I grandi sistemi hanno una loro dinamica che è caratterizzata da comportamenti e trends regolari che possono essere usati per fare predizioni: in particolare, la dimensione, il tempo fra due release successive, il numero di errori rilevati sono approssimativamente invarianti per ogni release.

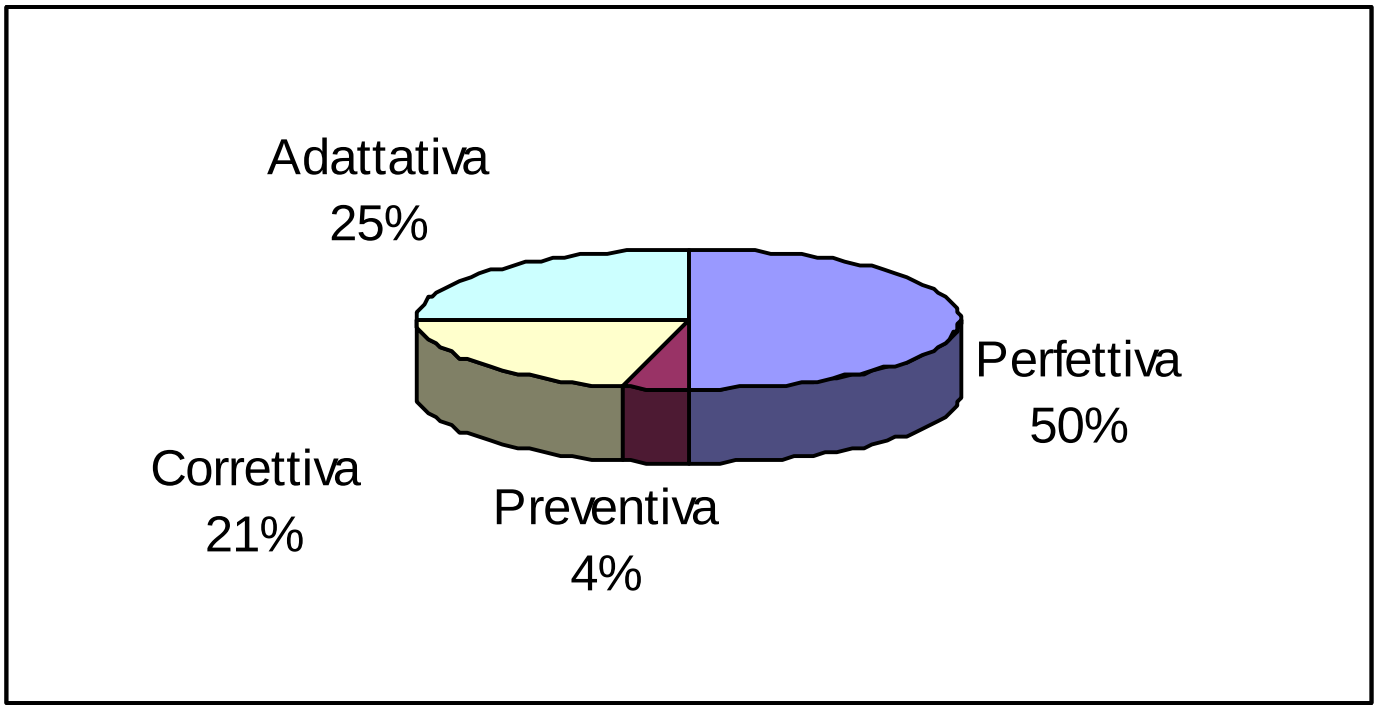
Le Leggi di Lehman (1978-1991-1996)

- Stabilità organizzativa
  - Raggiunto lo stato ‘saturo’, non ci possono essere variazioni significative nella produttività dello staff
- Conservazione della familiarità
  - Le modifiche incrementali per ogni release sono approssimativamente costanti
- Crescita continua
  - Le funzionalità offerte dai sistemi devono crescere continuamente, per la soddisfazione degli utenti
- Qualità deteriorata
  - La qualità dei sistemi si deteriora se non si interviene con adattamenti ai nuovi ambienti operativi
- Sistema feedback
  - I processi evolutivi incorporano sistemi di feedback utili al miglioramento dei prodotti

Tipi di Manutenzione del software

- Classificazione degli interventi di manutenzione
  - Manutenzione correttiva
    - Modifiche per correggere difetti
  - Manutenzione adattativa
    - Modifiche per adattare il software a cambiamenti dell'ambiente operativo (hardware, software di base, interfacce, organizzazione, legislazione, ecc.)
  - Manutenzione perfettiva o evolutiva
    - Estensione dei requisiti funzionali, o migliorie di requisiti non funzionali in risposta a richieste dell'utente
  - Manutenzione preventiva
    - Modifiche che rendono più semplici le correzioni, gli adattamenti e le migliorie
- Manutenzione di emergenza
    - Manutenzione correttiva non programmata, necessaria a mantenere il sistema in funzione

Distribuzione dello sforzo di manutenzione [1]



[1] Lientz, Swanson, Problems in application software maintenance, 1981 Communication of the ACM

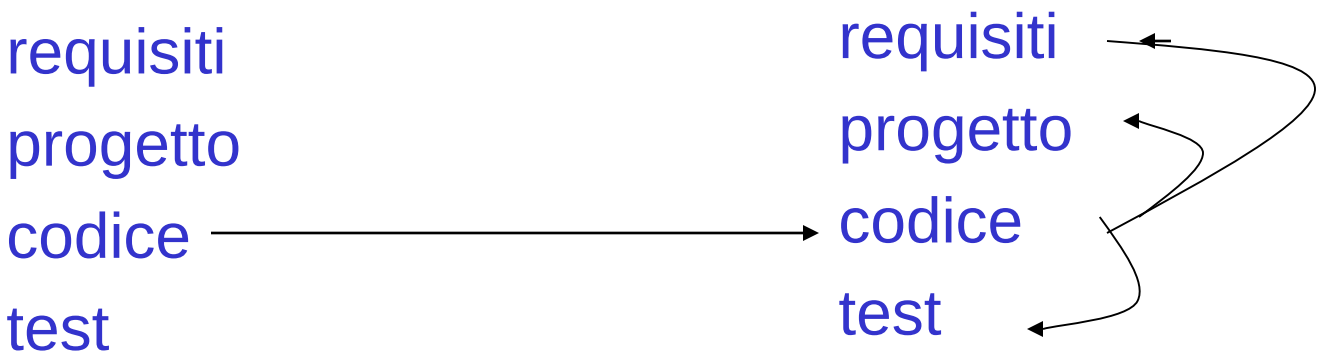
Manutenzione “d’urgenza”

- In alcuni casi le richieste di manutenzione devono essere soddisfatte rapidamente:
  - Se un difetto serio deve essere riparato;
  - Se modifiche dell’ambiente operativo causano effetti collaterali imprevisti sull’operatività del sistema;
  - Se è necessario l’adeguamento urgente a seguito di cambiamenti imprevisti (es. Ambiente o mercato)

- In questo caso, le fasi di analisi e progetto della modifica potrebbero non essere eseguite, implementando direttamente il cambiamento.

Processo di riparazione d'urgenza (quick- fix model)

Vecchio sistema **Nuovo sistema**



- Con tale approccio, la qualità complessiva del software si ridurrà.
- Utile pianificare interventi di aggiornamento della documentazione e/o di miglioramento della qualità del software.

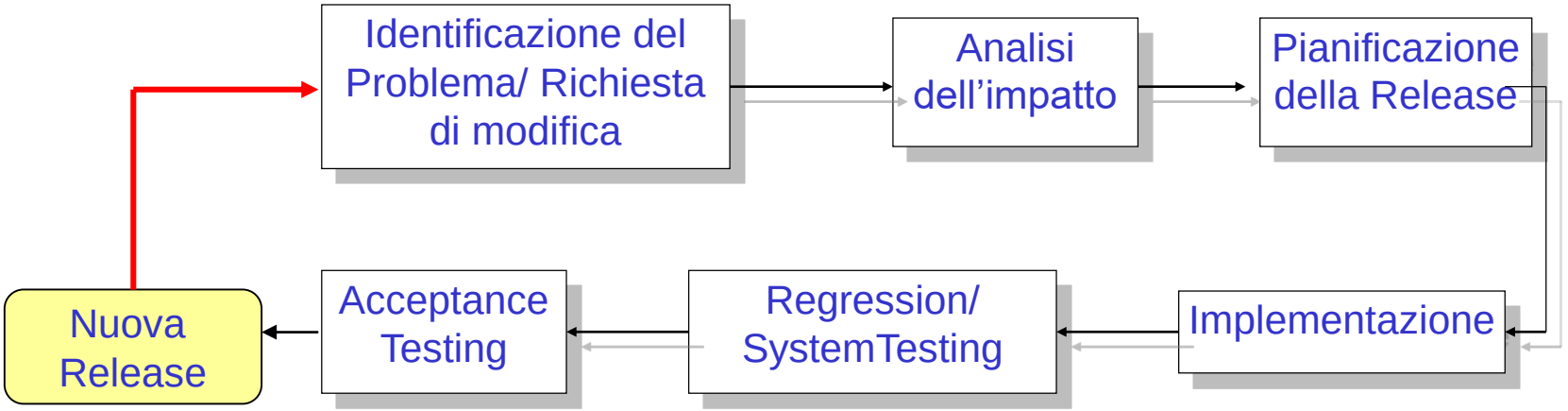
Problemi della manutenzione

- In gran parte dipendono dalla mancanza di controllo e disciplina nelle fasi di analisi e progetto del Ciclo di Vita del Software
- Alcuni fattori tecnici:
  - difficoltà nel comprendere un programma scritto da altri
  - mancanza di documentazione completa/ consistente
  - software non progettato per modifiche future
  - difficoltà nel tradurre una richiesta di modifica di funzionamento del sistema in una modifica del software
  - valutazione dell'impatto di ciascuna modifica sull'intero sistema
  - la necessità di ritestare il sistema dopo le modifiche
  - la gestione della configurazione del software

Fattori di Costo della manutenzione

- Stabilità del team
  - I costi di manutenzione si riducono se lo stesso staff si occupa della manutenzione per lungo tempo
- Responsabilità contrattuale
  - Sviluppo e manutenzione sono talvolta appaltati ad aziende diverse, cosicchè chi sviluppa non ha interesse a semplificare il lavoro di chi effettuerà la manutenzione
- Capacità dello staff
  - Chi si occupa della manutenzione potrebbe non avere lo stesso livello di esperienza e pratica di chi lo ha sviluppato
- Età e struttura del programma
  - Gli interventi di manutenzione tendono a far deteriorare la qualità del software e quindi a rendere più difficili tutti gli interventi di manutenzione necessari
    - v. anche G. Visaggio, “Aging of a legacy system: symptoms and remedies”, Journal of Software Maintenance, Wiley

Il processo di evoluzione del software



Gli interventi per ‘Ringiovanire’ il software

- Sono finalizzati a migliorare la manutenibilità di un software ormai deteriorato dagli interventi di manutenzione subiti.
- Diverse tipi di intervento possibili:

- Ridocumentazione
- Restructuring (o Refactoring)
- Reengineering
- Reverse Engineering

Reengineering

- It is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form [*Chikofsky and Cross*]

- La reingegnerizzazione (reengineering) è un'attività di re-implementazione di un sistema software svolta per migliorare la manutenibilità di un software esistente.

- Essa può comprendere:
  - Ridocumentazione, Ristrutturazione, Refactoring e riscrittura di parte del software (o anche di tutto) senza modificare l'insieme di funzionalità che esso realizza

Obiettivi del Reengineering

- *Modularizzazione del sistema*
  - Suddivisione di un sistema monolitico in parti da riusare separatamente
- *Miglioramento delle Performance*
  - Migliorare le prestazioni di un sistema esistente
- *Migrazione (o Porting) verso altre Piattaforme*
  - Necessità di localizzare I componenti dipendenti dalla piattaforma
- *Estrazione del progetto*
  - Per migliorare maintainability, portability, etc.
- Migrazione verso una *nuova Tecnologia*
  - quali nuove caratteristiche di un linguaggio, standards, librerie, etc.

Ridocumentazione

- Consiste in una **analisi statica** del codice sorgente (attraverso appositi strumenti) al fine di produrre documentazione del sistema. Si analizzano:
  - usi delle variabili, chiamate fra componenti, path del flusso di controllo, dimensioni dei componenti, parametri di chiamate, .. Per capire cosa fa il codice e come lo fa.

- Gli output di una attività di ridocumentazione possono essere:
  - grafi delle chiamate,tabelle delle interfacce delle funzioni, dizionari dati, diagrammi del data-flow o control-flow, pseudo-codice, cross-reference fra componenti o variabili
  - Tali output si possono usare per verificare se il software ha bisogno di ristrutturazione.

Ridocumentazione

- Un approccio diverso alla ridocumentazione può passare per l'analisi dinamica
  - Si eseguono scenari dei casi d'uso dell'applicazione

- Per analisi dinamica si possono ottenere:
  - Modelli dell'interfaccia utente
  - Modelli dell'interazione tra i componenti
  - Documentazione per l'utente finale
    - Tutorial, …

Standard di documentazione del codice

- Tra gli innumerevoli standard di documentazione esistenti, verrà presentato Javadoc
  - Ideato per Java
  - Affiancato da un tool (javadoc) in grado di generare manualistica a partire dall'analisi della documentazione presente nel codice sorgente
  - Generalizzabile a qualsiasi altro linguaggio
- Come tool generante, oltre a javadoc, verranno mostrate le potenzialità di Doxygen

Esempio Javadoc

Inizio commento per Javadoc

URL diventerà un link nella documentazione HTML

Breve riassunto dello scopo e dei parametri del metodo (Description Block)

Nuova linea nell'HTML risultante

Parametri

Valore di ritorno

Riferimento

Codice del metodo

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the base location of the image
 * @param  name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
public Image getImage(URL url, String name) {
        try {
            return getImage(new URL(url, name));
        } catch (MalformedURLException e) {
            return null;
        }
}
```

HTML risultante

getImage

```
public Image getImage(URL url,
                      String name)
```

Returns an `Image` object that can then be painted on the screen. The `url` argument must specify an absolute `URL`. The `name` argument is a specifier that is relative to the `url` argument.

This method always returns immediately, whether or not the image exists. When this applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

Parameters:

`url` - an absolute URL giving the base location of the image

`name` - the location of the image, relative to the `url` argument

Returns:

the image at the specified URL

See Also:

`Image`

Principali Tag Javadoc

@author *name-text*
@deprecated *deprecated-text*
{@code *text***}**
{@docRoot}
@exception *class-name description*
{@inheritDoc}
{@link *package.class#member label***}**
{@linkplain *package.class#member label***}**
{@literal *text***}**
@param *parameter-name description*
@return *description*
@see *reference*
@serial *field-description* | include | exclude
@serialField *field-name field-type field-description*
@serialData *data-description*
@since *since-text*
@throws *class-name description*
{@value *package.class#field***}**
@version *version-text*

- **La documentazione di riferimento del “linguaggio” javadoc e dei modi di utilizzo del tool corrispondente è disponibile all’indirizzo:**

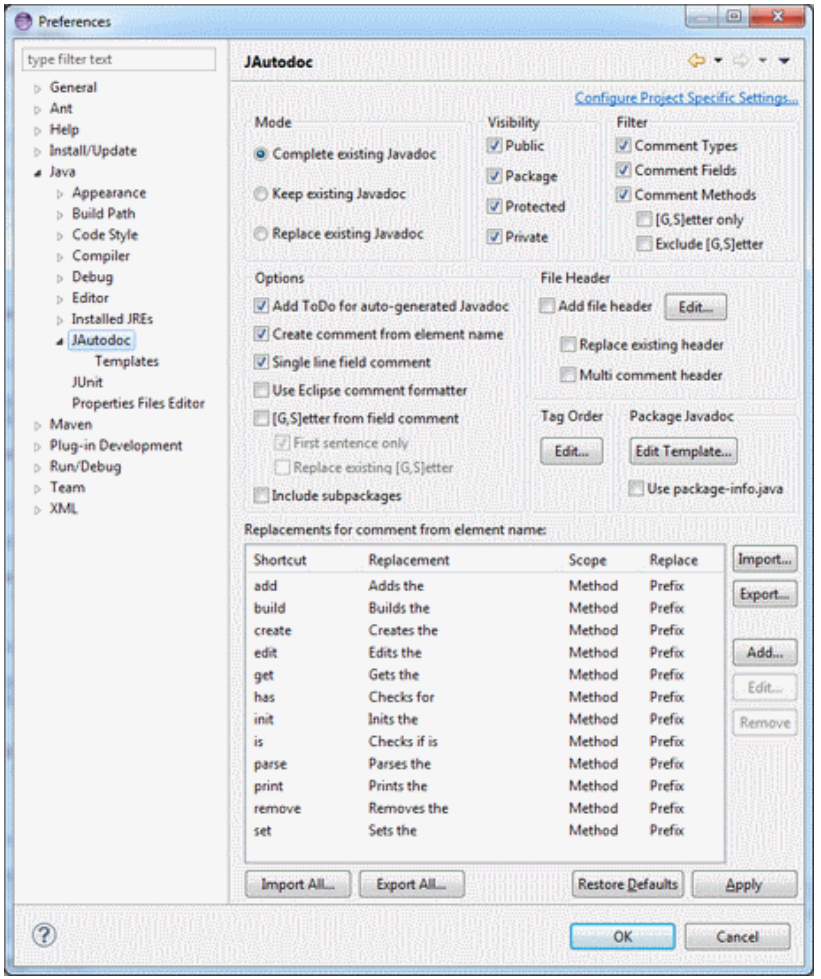
  http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/javadoc.html#tags

- Esistono strumenti in grado di valutare la completezza della documentazione
  - Checkdoc della Sun: http://java.sun.com/j2se/javadoc/doccheck/

- **Il tool javadoc, richiamabile da linea di comando, genera un insieme di file in puro HTML (con l’utilizzo di frames), altamente navigabili**
  - **Non è prevista, viceversa, la generazione di un unico documento, che possa costituire lo scheletro di un manuale utente**

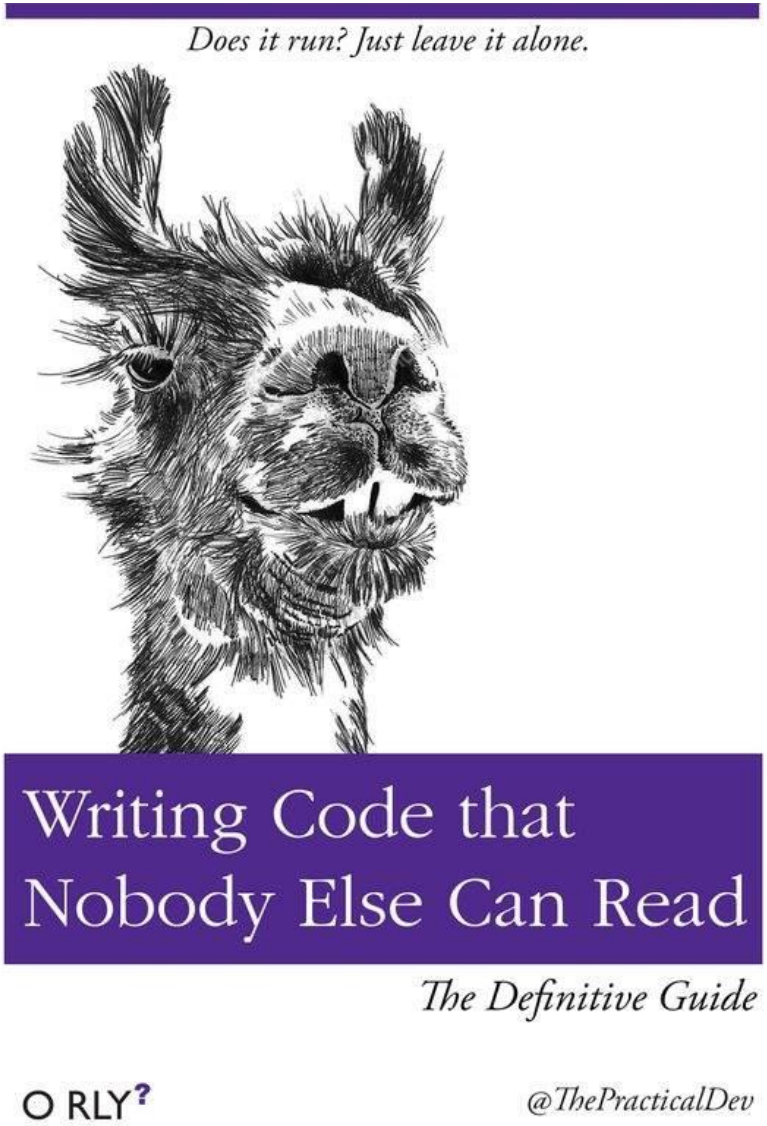
JAutoDoc

- JAutoDoc è uno strumento, integrato in Eclipse, che genera automaticamente scheletri di configurazione, da completare
  - http://jautodoc.sourceforge.net/
- E' in grado anche di riparare documentazioni incomplete



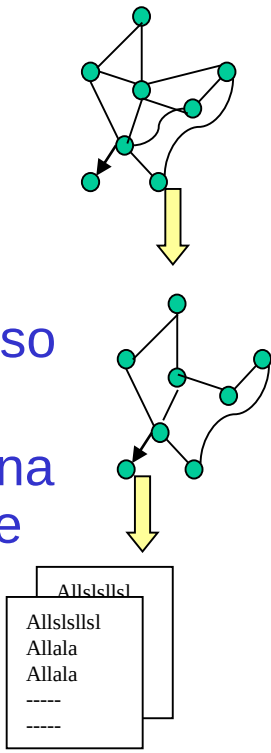
Doxygen

- Sistema di documentazione utilizzabile per programmi scritti in C++, C, Java, ed altri
  - https://sourceforge.net/projects/doxygen
  - Genera documentazione in HTML e anche un manuale di riferimento completo in RTF (MS-Word), PostScript, PDF, chm o man Unix, direttamente a partire dalle informazioni reperibili nel codice sorgente.
  - Doxygen estrae i commenti scritti in formato simile a quello di Javascript e inoltre la struttura dei costrutti principali ricavabili dall'analisi statica del codice, comprese associazioni, ereditarietà, …
  - Doxygen genera automaticamente alcuni diagrammi, tra cui grafi delle dipendenze, diagrammi di ereditarietà, diagrammi di collaborazione ed altro, secondo il formato DOT
    - http://www.research.att.com/sw/tools/graphviz/


Does it run? Just leave it alone.
Writing Code that Nobody Else Can Read
The Definitive Guide
O RLY?
@ThePracticalDev

Restructuring

- Attività che trasforma il codice esistente in codice equivalente dal punto di vista funzionale, ma migliorato dal punto di vista della sua qualità.
- In genere si esegue in tre passi:
  1. Analisi statica del codice per ottenerne una rappresentazione interna (es. call graph, control-flow graph…)
  2. Semplificazione della rappresentazione interna attraverso tecniche di trasformazione automatiche.
  3. La nuova rappresentazione viene usata per generare una versione strutturata (migliorata) ed equivalente al codice originario.



Refactoring

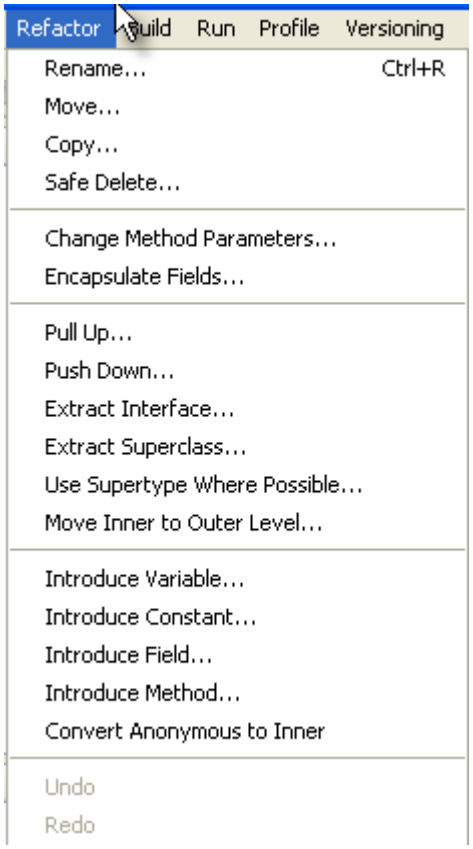
- Anche i sistemi object-oriented sono soggetti al deteriorarmento e diventano legacy!
- Il refactoring è un insieme di tecniche usabili per migliorare il codice ed il design di sistemi object-oriented.
  - Martin Fowler è autore del libro “Refactoring: Improving the Design of Existing Code” (1999)dove presenta più di 70 pattern per eseguire il refactoring.
- Tali tecniche cercano di eliminare i cosiddetti *Bad Smells* dal codice.

Esempi di Bad Smells nel codice

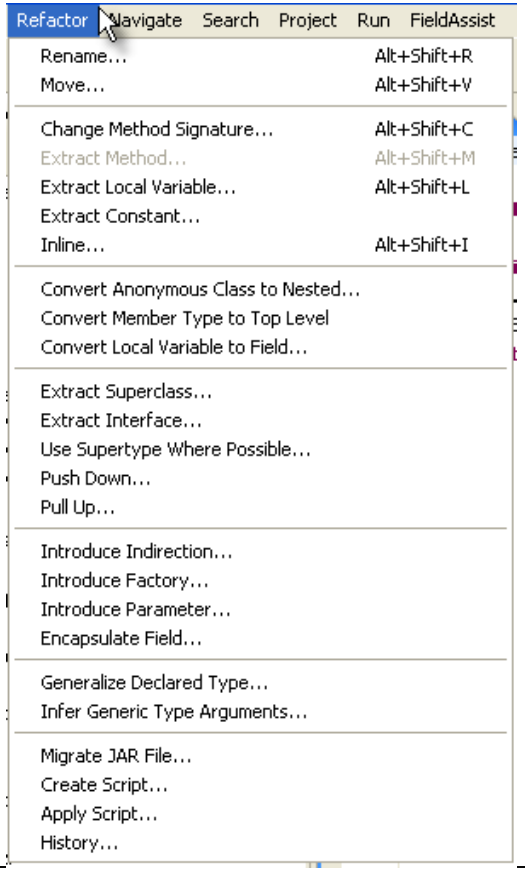
Problema (Bad Smell)	**Pattern di Refactoring applicabile**
Codice duplicato	*Extract Method*
Metodi troppo lunghi	*Extract Method*
Classi troppo grandi	*Extract Class, Extract Sub-Class, Extract Interface*
Lunghe liste di parametri di metodi	*Replace parameter with Method*
Cambiamenti divergenti in una classe (una classe è soggetta a cambiamenti per tanti motivi)	*Extract Class*
…	…

Esempi di Tool per il Refactoring

Net Beans Refactoring



Eclipse Refactoring



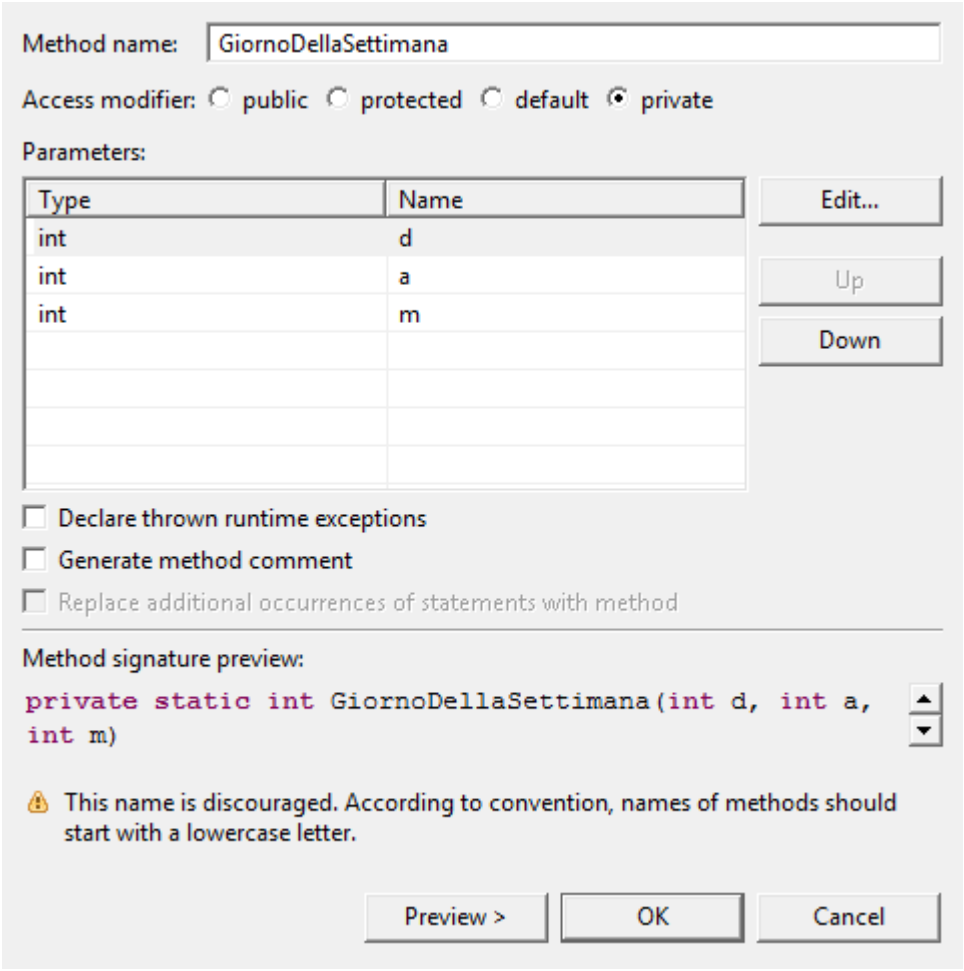
Esempi di Refactoring: extract method

- Extract method
  - Seleziona un pezzo di codice
  - Imposta nome e parametri
  - Sostituisci

```
if ((m==4 || m==6 || m==9 || m==11) && d>30)
    return "Errore";

if (m<=2)
 {
  m = m + 12;
  a--;
 };
 int f1 = a / 4;
 int f2 = a / 100;
 int f3 = a / 400;
 int f4 = (int) (2 * m + (.6 * (m + 1)));
 int f5 = a + d + 1;
 int x = f1 - f2 + f3 + f4 + f5;
 int k = x / 7;
 int n = x - k * 7;

 if (n==1)
     return "Lunedi";
```

Esempi di Refactoring: extract method

- Extract method
  - Seleziona un pezzo di codice
  - Imposta nome e parametri
  - Sostituisci



Esempi di Refactoring: extract method

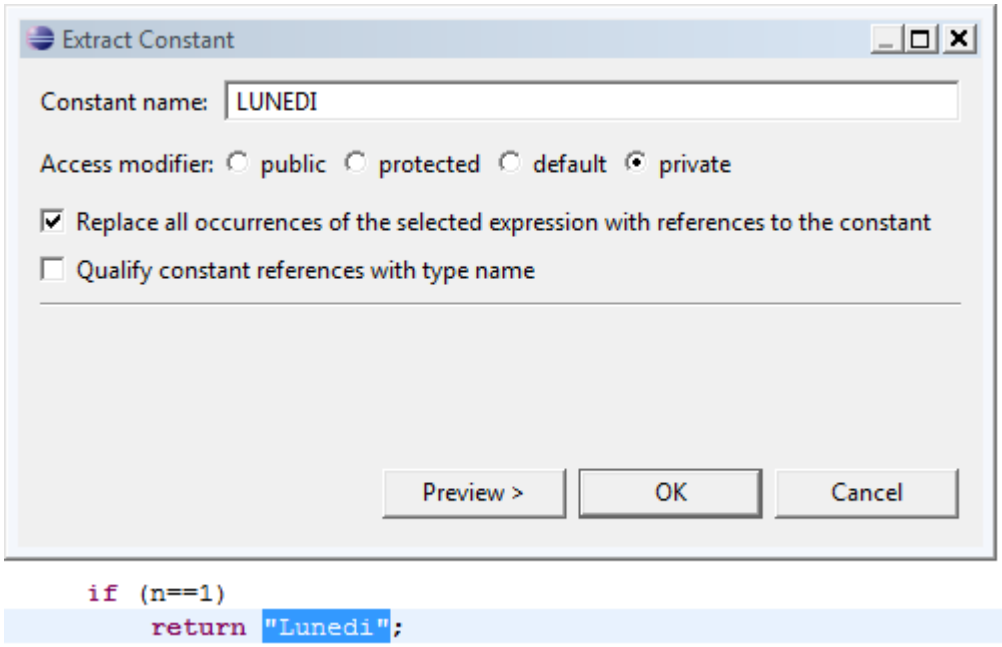
- Extract method
  - Seleziona un pezzo di codice
  - Imposta nome e parametri
  - Sostituisci

```
int n = giornoDellaSettimana(d, a, m);
```

```
private static int giornoDellaSettimana(int d,
int a, int m) {
if (m<=2)
 {
  m = m + 12;
  a--;
 };
 int f1 = a / 4;
 int f2 = a / 100;
 int f3 = a / 400;
 int f4 = (int) (2 * m + (.6 * (m + 1)));
 int f5 = a + d + 1;
 int x = f1 - f2 + f3 + f4 + f5;
 int k = x / 7;
 int n = x - k * 7;
return n;
}
```

Esempi di Refactoring: extract constant

- Extract constant:
  - Seleziona un valore costante
  - Converti in costante (tutte le sue occorrenze)



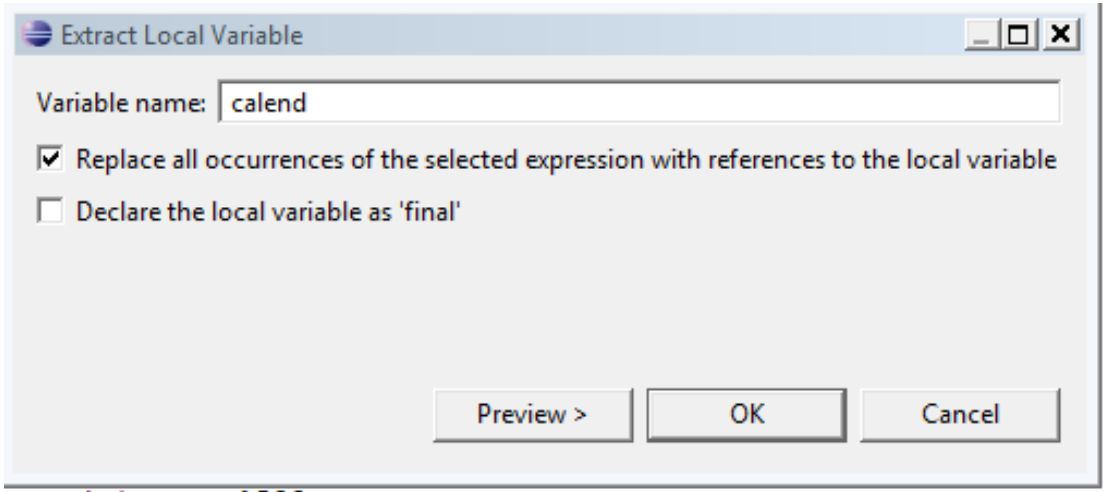
Esempi di Refactoring: extract constant

- Extract constant:
  - Seleziona un valore costante
  - Converti in costante (tutte le sue occorrenze)

```
private static final String
LUNEDI = "Lunedi";

…

 if (n==1)

 return LUNEDI;
```

Esempi di Refactoring: extract local variable

- Extract local variable:
  - Seleziona un'espressione che ha un valore
  - Converti l'espressione in una variabile (tutte le sue occorrenze)

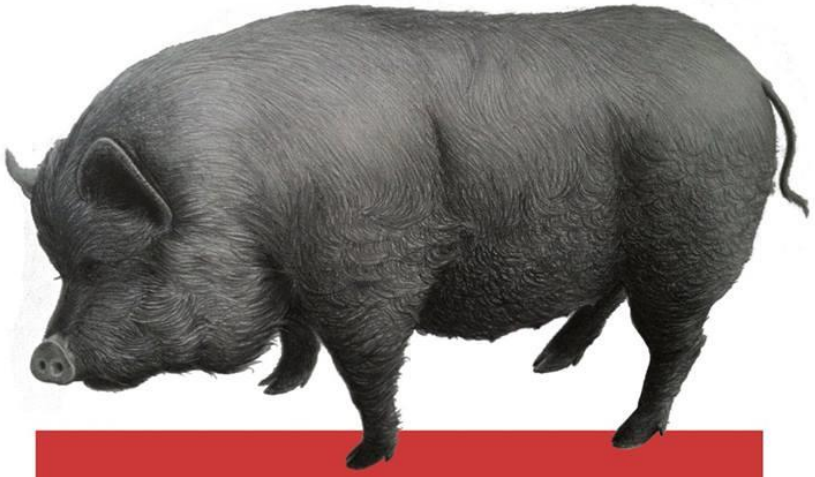


Esempi di Refactoring: extract local variable

- Extract local variable:
  - Seleziona un'espressione che ha un valore
  - Converti l'espressione in una variabile (tutte le sue occorrenze)

```
String calend =
calend(giorno,mese,anno);

System.out.println(calend);
```

Code written by some stranger on the internet is always perfect



Taking on Needless Dependencies

Fragile Development Guide

O RLY?

@ThePracticalDev

CodePro Analytix

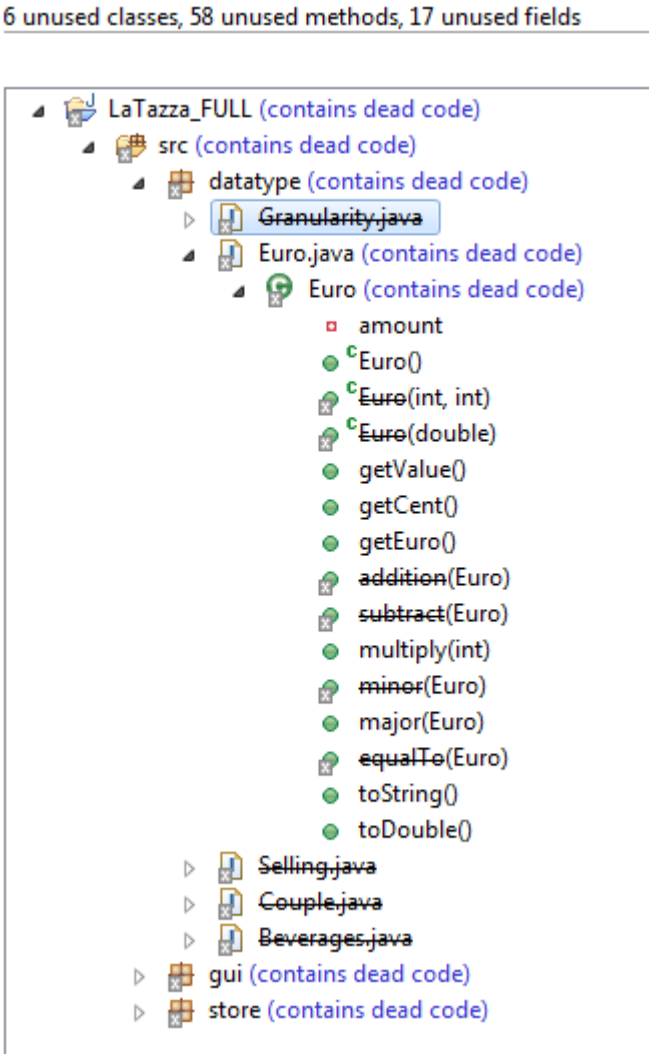
- Plug-in multifunzionale per Eclipse offerto da Google
  - https://developers.google.com/java-dev-tools/codepro/doc/
  - Scaricabile direttamente da:
    - http://dl.google.com/eclipse/inst/codepro/latest/3.7
    - Aggiornato ad Eclipse Indigo; dovrebbe funzionare anche per le versioni successive
- Tutorial e documentazione accessibili da:
  - https://developers.google.com/java-dev-tools/codepro/doc/

CodePro Analytix Dead Code Analysis

- CodePro cerca, tramite analisi statica, di verificare quale parte del codice non sia raggiungibile (codice morto)
  - Le indicazioni di CodePro non possono essere considerate né complete né sempre valide, a causa di possibili utilizzi del codice tramite chiamate dinamicamente generate
- Per eseguire l'analisi del Dead Code è sufficiente eseguire *Find Dead Code* nel menu contestuale *Code Pro Tools*

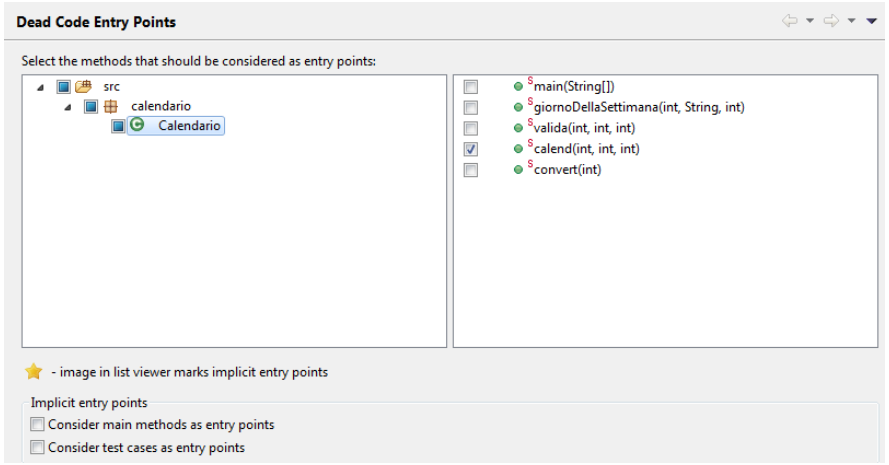
Utilizzo di Find Dead Code

- Selezionare (da proprietà del progetto/*Dead Code Entry Point*) i punti di accesso dell'applicazione (di solito il main)
- Avviare *Find Dead Code*
- Osservare i risultati nella view *Dead Code*
  - Le classi/metodi/attributi barrati sono inutilizzabili (secondo quest'analisi)



Utilizzo di Find Dead Code

- La ricerca del Dead Code non funziona bene in caso di codice generato dinamicamente
  - Un caso tipico è rappresentato dal codice ascoltatore di eventi utente
    - In questo caso, è opportuno considerare le classi che rispondono agli ascoltatori come ulteriori Entry Point
    - Per conoscere il Dead Code interno ai metodi, può essere opportuno segnare ogni metodo come possibile entry point



Reverse Engineering

- È un'attività che consente di ottenere specifiche e informazioni sul design di un sistema a partire dal suo codice, attraverso processi di estrazione ed astrazione di informazioni.

- La definizione di **Chikofsky and Cross**, 1990 [1]:
- *Analyzing a subject system*
  - *to identify the system's components and their inter-relationships and*
  - *to create representations of the system in another form or at a higher level of abstraction*
- *A two-steps process*
  - *information extraction*
  - *view abstraction*

[1] **Chikofsky and Cross,** Reverse engineering and design recovery: A taxonomy. *IEEE Software*,1990

Reverse Engineering Goals

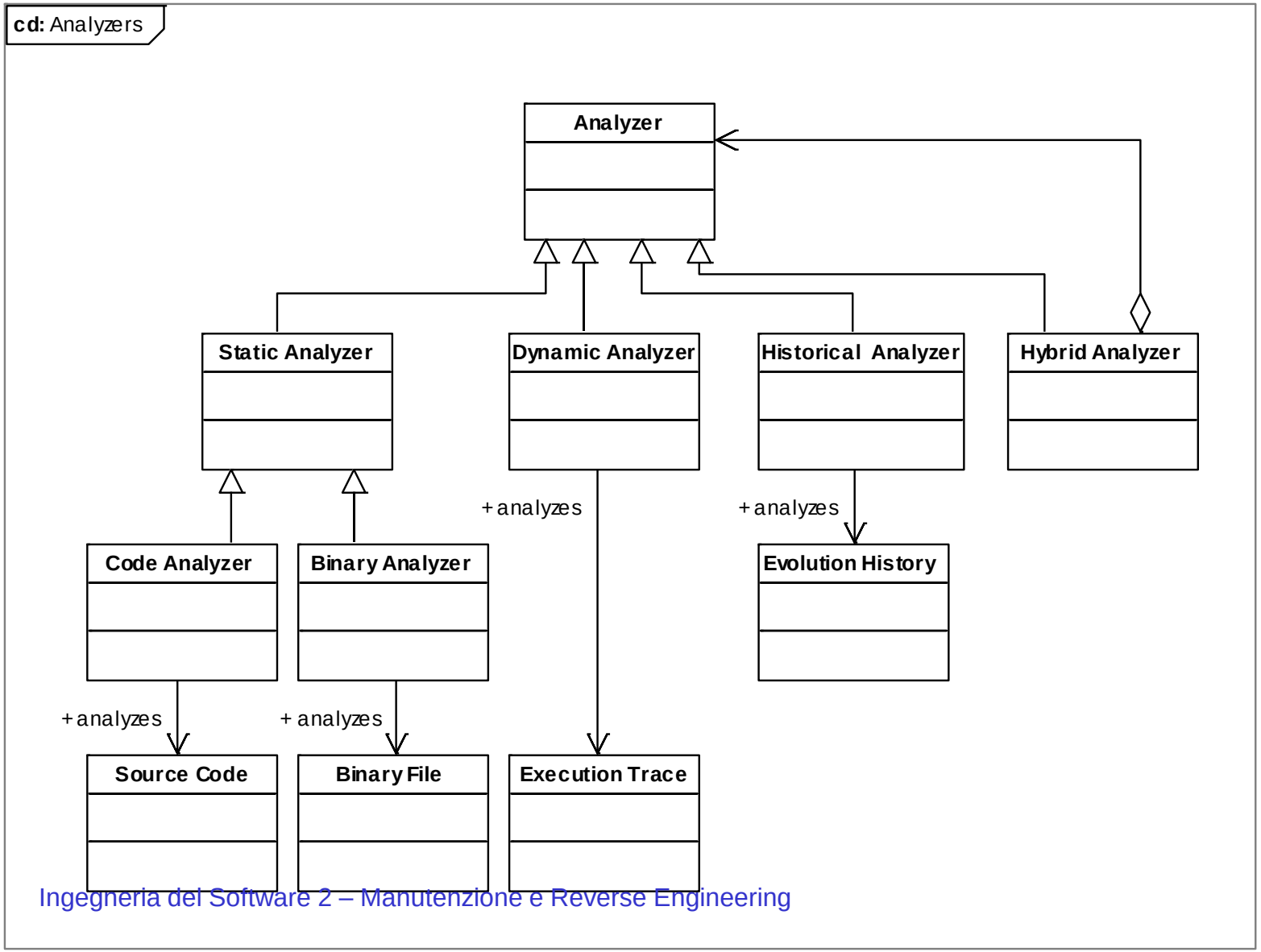
- Reverse engineering tasks are performed to satisfy reverse engineering goals:
  - Change goals: produce a renewed product
    - e.g. through migration, modernization, modularization
  - Understanding goals: increase the current understanding of a software product
    - Produce high level views by means of an Abstractor
    - Can start from source code, but also from binary
    - Examples of targets: recovering architectures, design, traceability links, building metric views

Canfora, Di Penta "Frontiers of Reverse Engineering: a Conceptual Model", FOSM08 – IEEE Comp. Soc. 2008

Fasi di Estrazione ed Astrazione

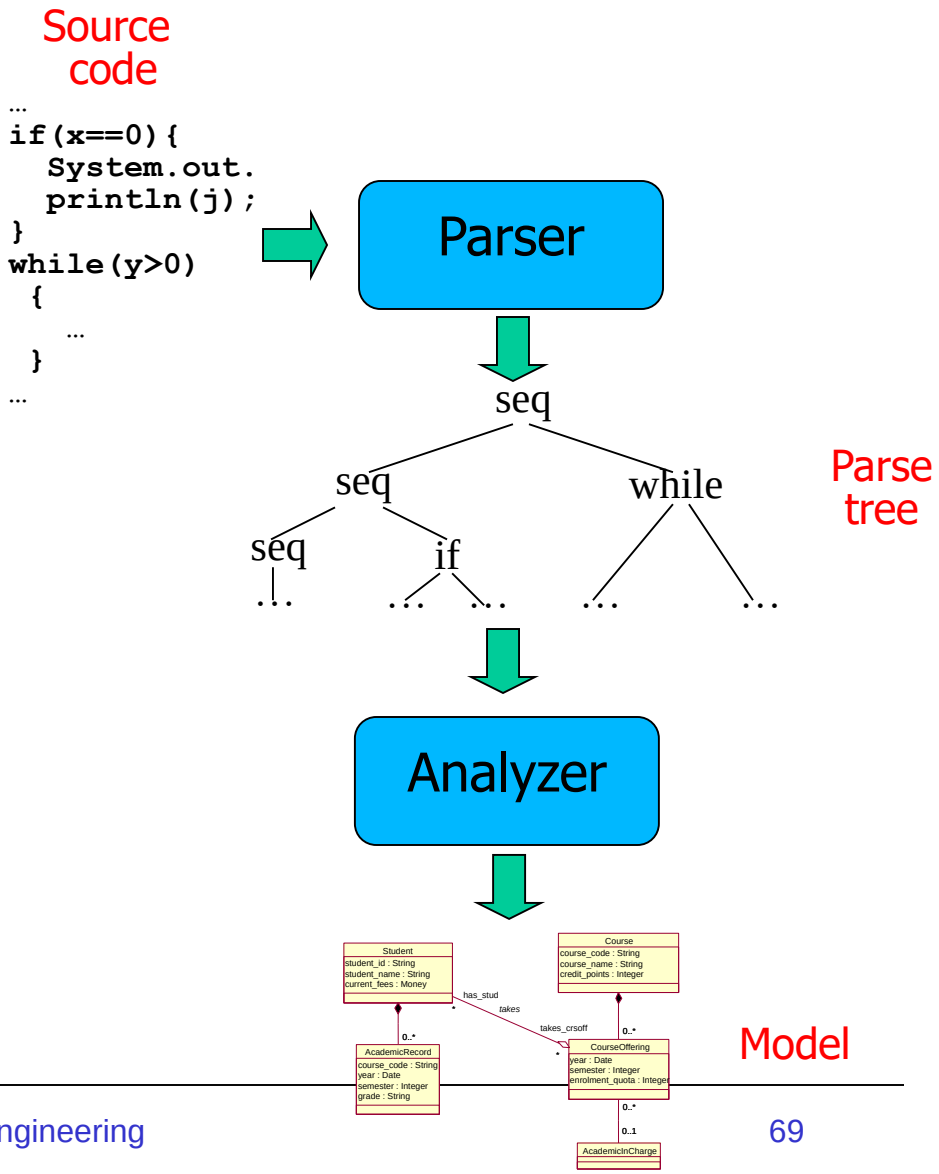
- Estrazione
  - Analisi del codice o di altri artifatti software, allo scopo di ottenere informazioni relative al sistema analizzato.
  - Particolarmente utili sono quelli strumenti in grado di estrarre informazioni da un codice sorgente qualsiasi, nota che sia la grammatica del linguaggio di programmazione (ad esempio JavaCC o Antlr)
- Astrazione
  - Si esaminano le informazioni estratte e si cercano di astrarre diagrammi, o viste, ad un più alto livello di astrazione (es.: diagrammi di progetto, architetturali, del dominio dei dati)
  - I processi di astrazione non sono completamente automatizzabili poichè necessitano di conoscenza ed esperienza umana

Tassonomia degli analizzatori



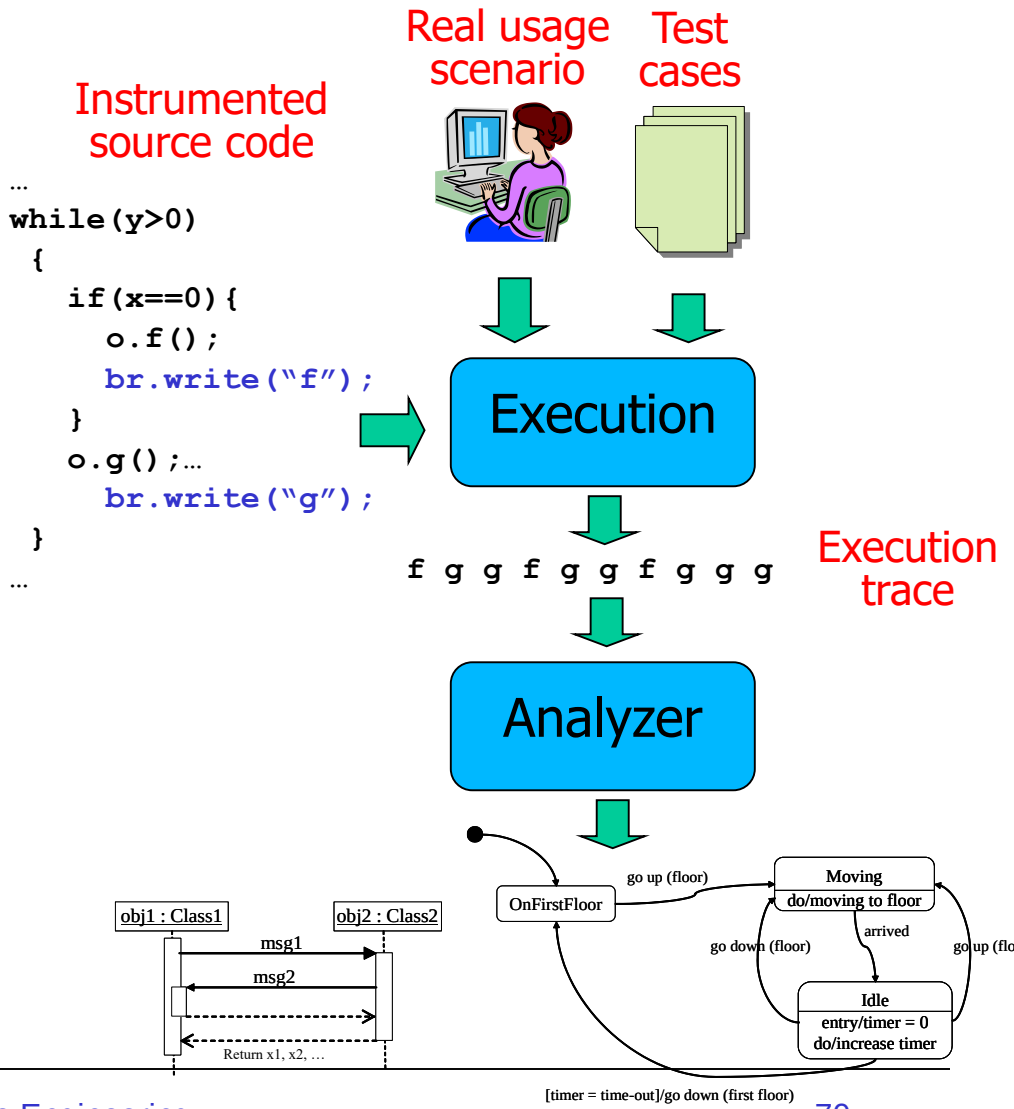
Static analysis

- Source code or other artifacts are parsed and then a model is built
  - Island grammars or code transformations as alternatives
- Relatively fast and cheap ☺
- Complete ☺
- Difficult to capture behavioral/dynamic models
  - State machines
  - Sequence diagrams
  - Detect behavioral patterns
- Intrinsic difficulties increasing with highly dynamic systems ☹
  - Pointer analysis
  - Polymorphism
  - Ultra-late binding



Dynamic analysis

- System exercised by means of
  - test cases
  - realistic usage scenario
- Collecting execution traces
  - Code instrumentation
  - Patching virtual machines
- Able to deal with dynamicity ☺
- Useful to extract dynamic models ☺
- Code must be compilable 😐
- The quality of the models built depends on the data used for execution 😐
- Might be incomplete ☹
- Might be expensive ☹



Dynamic models

Historical analysis

- Static and dynamic analysis do not capture information such as:
  - How does an artifact change during the time?
  - When was it changed?
  - Why was it changed?
    - New requirements, bug-fixing, refactoring, re-documentation…
  - Who changed it?
  - What artifacts changed together?

Problemi indecibili nel Reverse Engineering

- non è possibile, a partire dal solo codice, astrarre ***il*** *progetto* dal quale esso è stato prodotto
  - non è invece indecidibile il problema di astrarre un progetto coerente con il codice;
- non è possibile, a partire dal solo programma oggetto, astrarre ***il*** *programma* sorgente dal quale esso è stato prodotto
  - non è invece indecidibile il problema di astrarre un programma sorgente che generi il dato programma oggetto.

Problemi del Reverse Engineering

- Il processo di produzione del software è costellato di *pozzi* nei quali si perde parte della conoscenza: non tutta la conoscenza ed esperienza messa in campo dall'ingegnere del software in una fase di produzione (ad es. progettazione) viene in qualche forma rappresentata nello stesso prodotto di fase (progetto) o in quello delle fasi successive (ad es. codice).

- Questo comporta che ai problemi di indecidibilità si aggiungono quelli dovuti alla perdita di conoscenza che richiedono, per la realizzazione completa di un'astrazione, l'**aggiunta di conoscenza** ed esperienza da parte dell'ingegnere del software (almeno per ora).

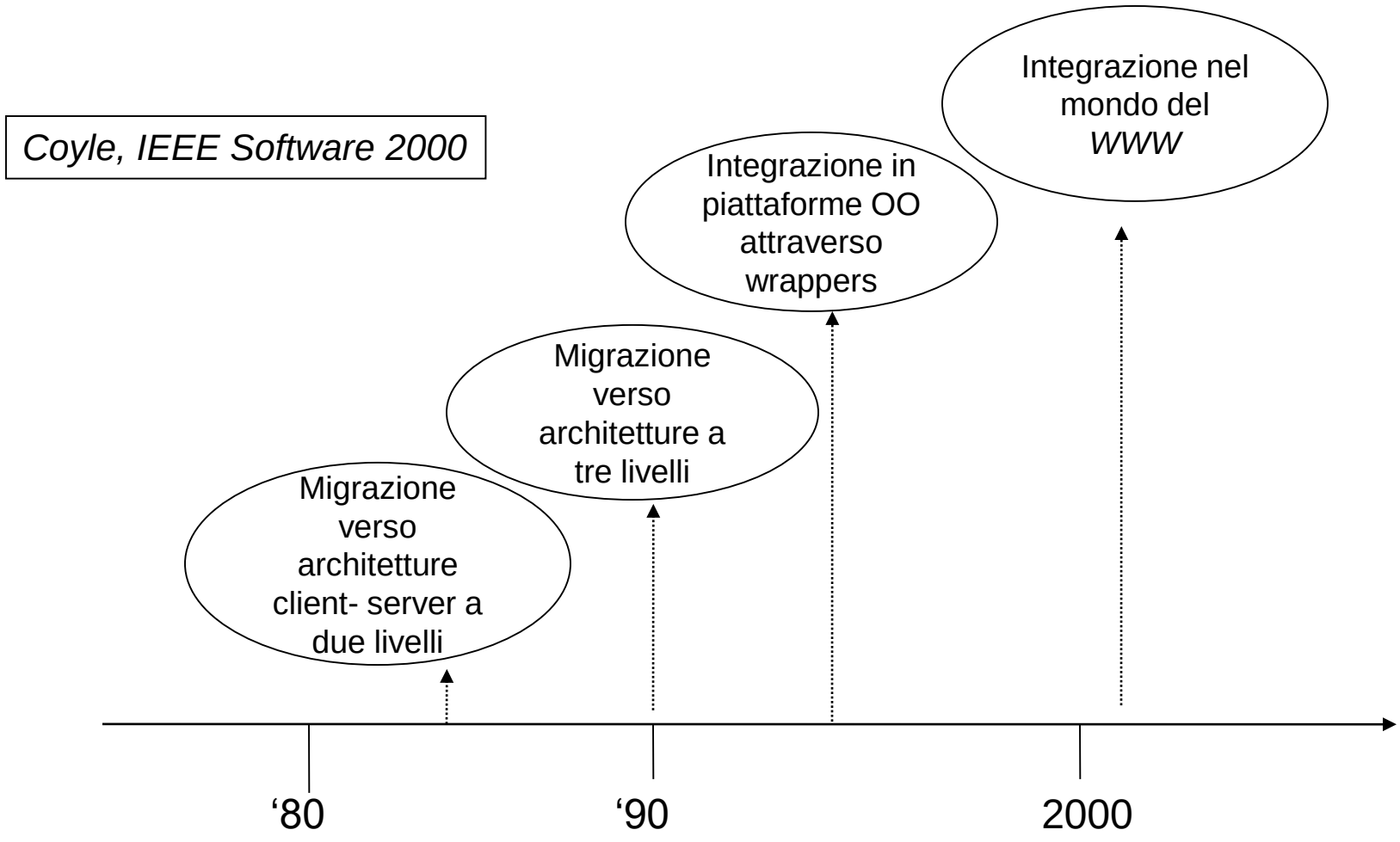
- Il Reverse Engineering non è, quindi, un'attività completamente automatizzabile!

Appendice

Il problema dei Legacy Systems

- Un sistema legacy (“ereditato”)
  - é spesso vecchio (10 anni o più di vita);
  - è di grandi dimensioni (centinaia di migliaia di linee di codice)
  - è scritto in assembler o in un linguaggio di vecchia generazione
  - è stato probabilmente sviluppato prima che si diffondessero i moderni principi dell’ingegneria del software
  - La manutenzione é stata svolta in modo da seguire le modifiche nei requisiti, aumentando così l’entropia (il disordine) del sistema
  - La manutenzione risulta ormai difficile e costosa
  - Realizza funzionalità cruciali e irrinunciabili per l’organizzazione
  - Contiene anni di esperienza accumulata nell’ambito del dominio specifico del problema

La gestione dei Sistemi Legacy: due decenni di strategie



Una recente analisi di strategie proposta da Grady Booch

G. Booch, Nine things you can do with old software, IEEE Software, Sept 2008

- Abbandonarlo
  - quando il suo valore economico si è esaurito, o lo sforzo di risviluppo non è eccessivo
- Regalarlo
  - Se non serve più, si può cederlo a qualche comunità open-source dove potrà ancora tornare utile a qualcuno
- Ignorarlo
  - Se è abbastanza stabile, e fa qualcosa di utile, si può continuare a usarlo senza però modificarlo.

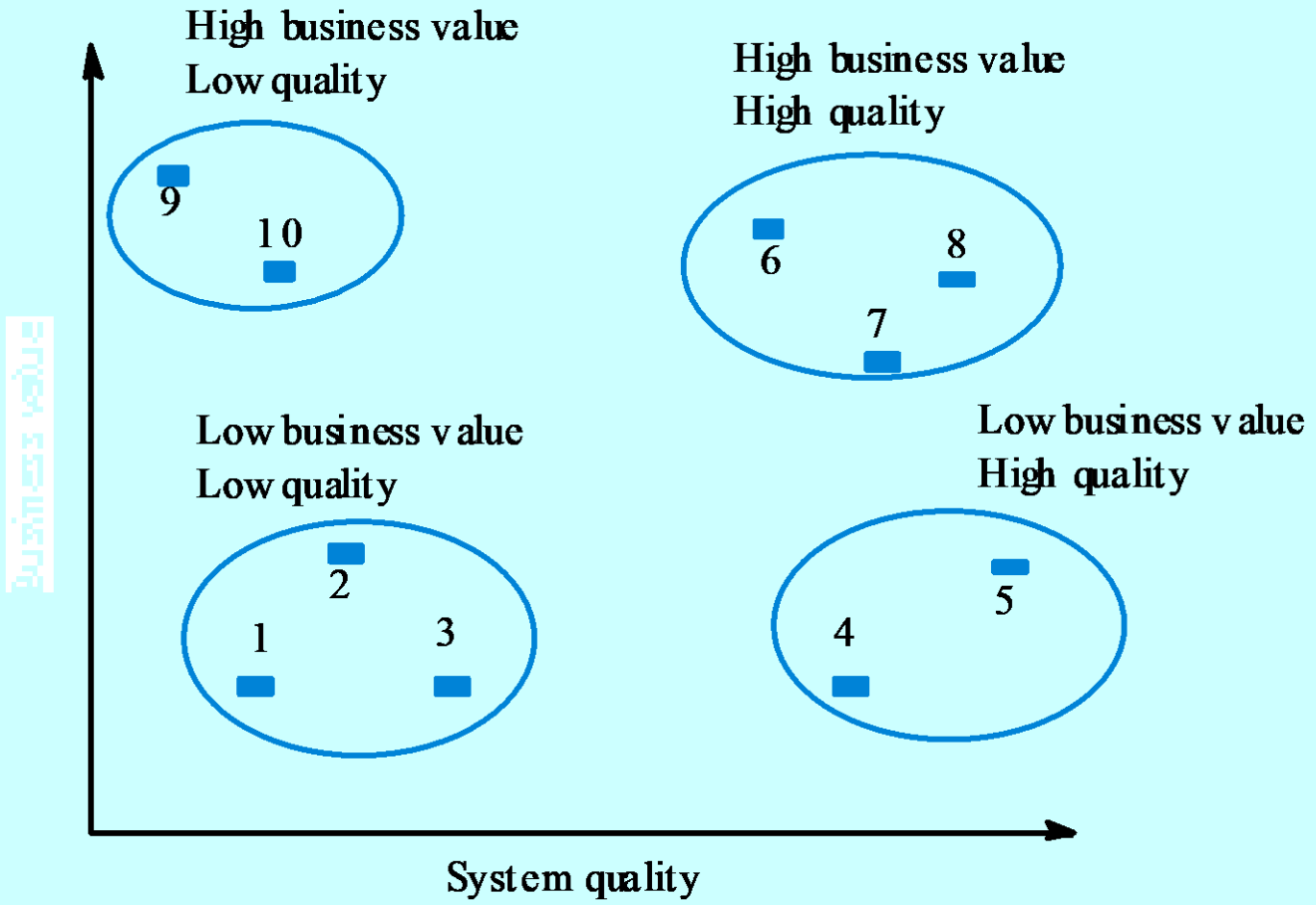
Nove cose da poter fare con software legacy

- Farlo sopravvivere
  - Quando l'hardware su cui gira non è più supportato (e non si dispone del codice sorgente per portarlo in nuove piattaforme), si fa sopravvivere o cercando vecchio hardware da cannibalizzare, o usando emulatori di piattaforma.
- Riscriverlo
  - Quando la manutenzione è troppo costosa, o il sistema è troppo fragile, si può riscriverlo (ma sapendo che ottenere un sistema funzionalmente equivalente al primo è impossibile, e che bisognerà probabilmente convincere gli utenti ad accettare qualche cambiamento)
- Farlo fruttare (in qualche modo)
  - Cercare parti del sistema da conservare (algoritmi, pattern, astrazioni…) perché ancora utili, ed usarle come una base di conoscenza per un nuovo sviluppo

Nove cose da poter fare con software legacy

- Wrapping
  - Usare tecniche di wrapping per integrarlo in nuove piattaforme (quali SOA)
- Trasformarlo
  - È la strategia più difficile e va praticata per mantenere il sistema in condizioni ottimali, se si prevede di continuare ad usarlo a lungo termine: va dal semplice refactoring, alla trasformazione architetturale.
- Preservarlo
  - Anche il software antico può avere un valore storico (es. vecchi sistemi operativi, o videogiochi) e culturale da preservare (magari in un Museo della Storia dei Computer)

System quality and business value



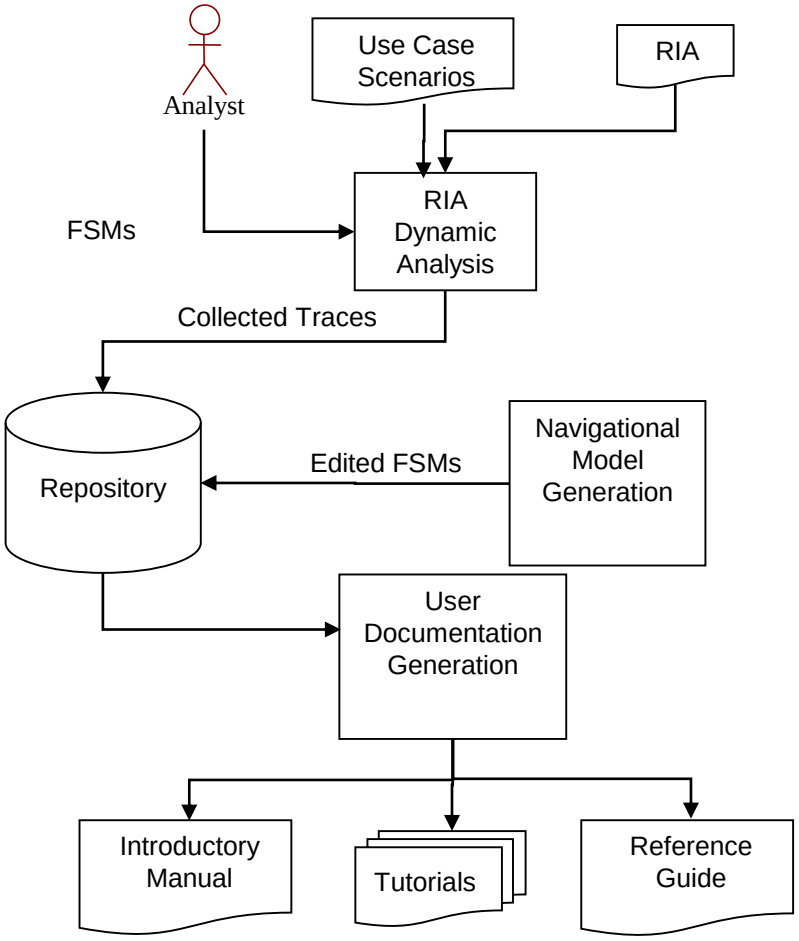
Legacy system categories

1. Low quality, low business value
   - Dovrebbe essere abbandonato
2. Low-quality, high-business value
   - Realizza funzionali importante ma è costoso mantenerlo. Dovrebbe essere reingegnerizzato in modo da rendere le future (necessarie) operazioni di manutenzione più agevoli ed efficaci (cioè finire nel quarto caso)
3. High-quality, low-business value
   - Si può decidere sia di abbandonarlo (in quanto poco importante), sia di rimpiazzarlo con COTS (se realizza qualcosa di generale, indipendente dal dominio specifico) oppure mantenerlo (dato che i costi di manutenzione saranno limitati)
4. High-quality, high business value
   - Su di esso si eseguono le operazioni di manutenzione
     - In questo modo, però, la qualità diventerà via via più bassa, fino a finire nel secondo caso

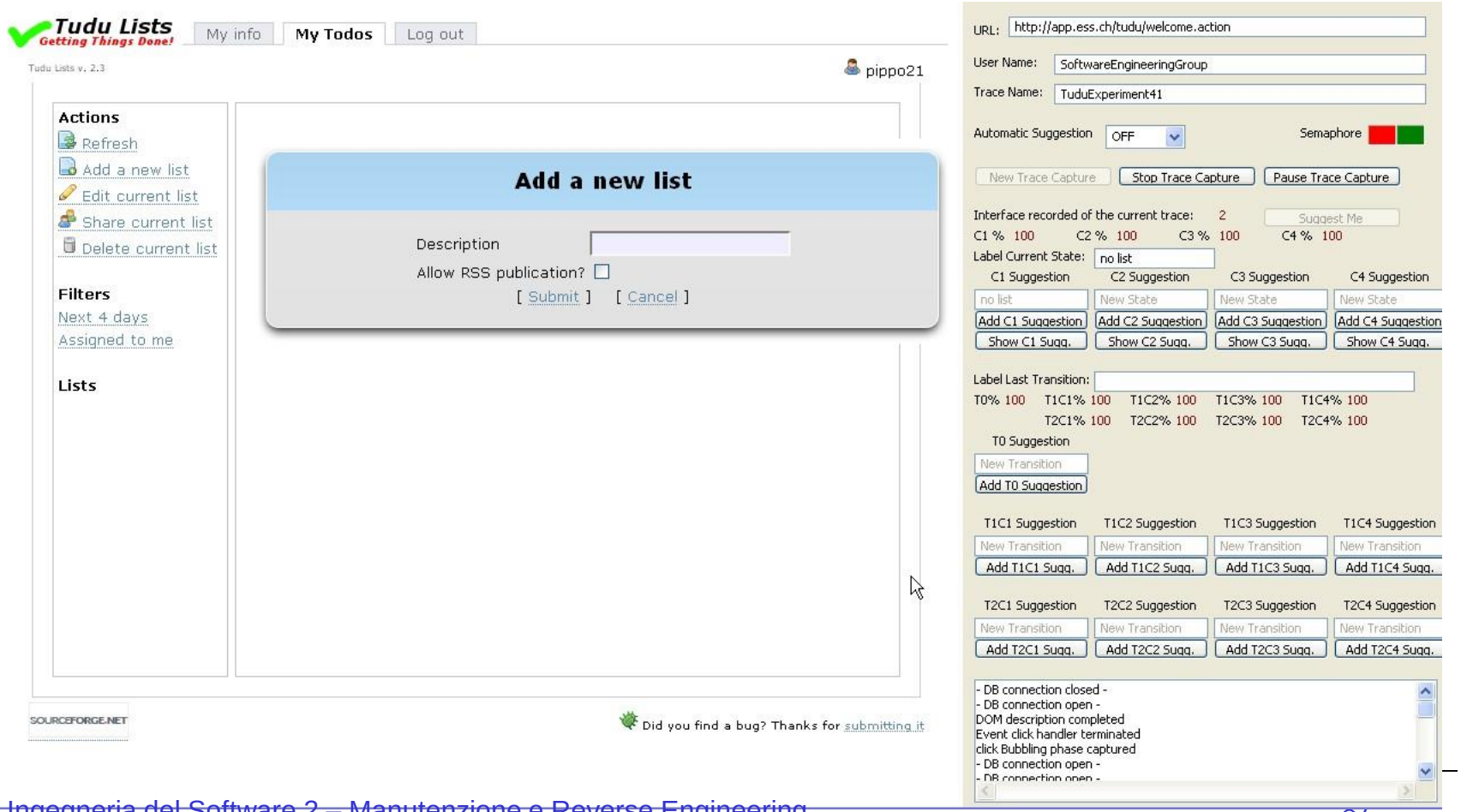
Appendice

Esempio: generazione automatica di tutorial di applicazioni Web

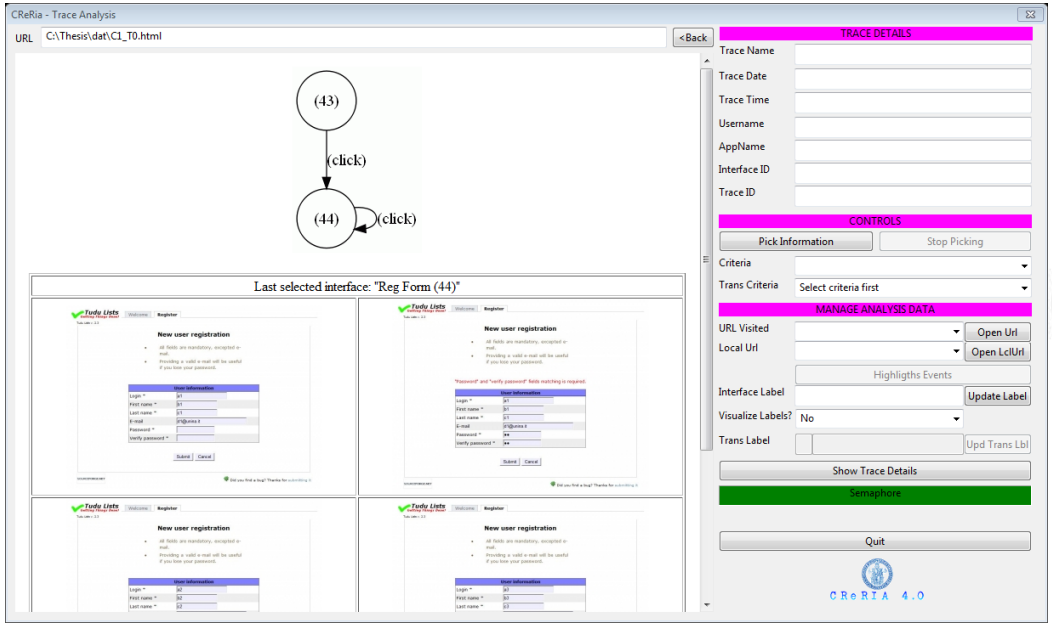
- Tre fasi:
  - Web Application Dynamic Analyis
  - Generazione del Navigation Graph
  - Generazione della End User Documentation

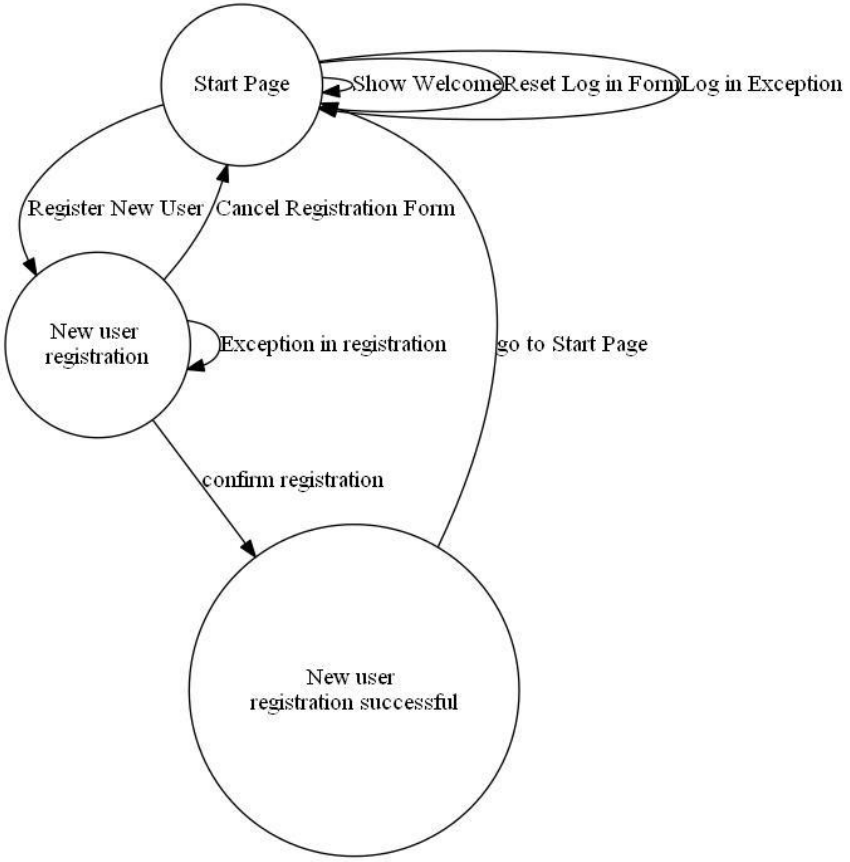


Web Application Dynamic Analysis

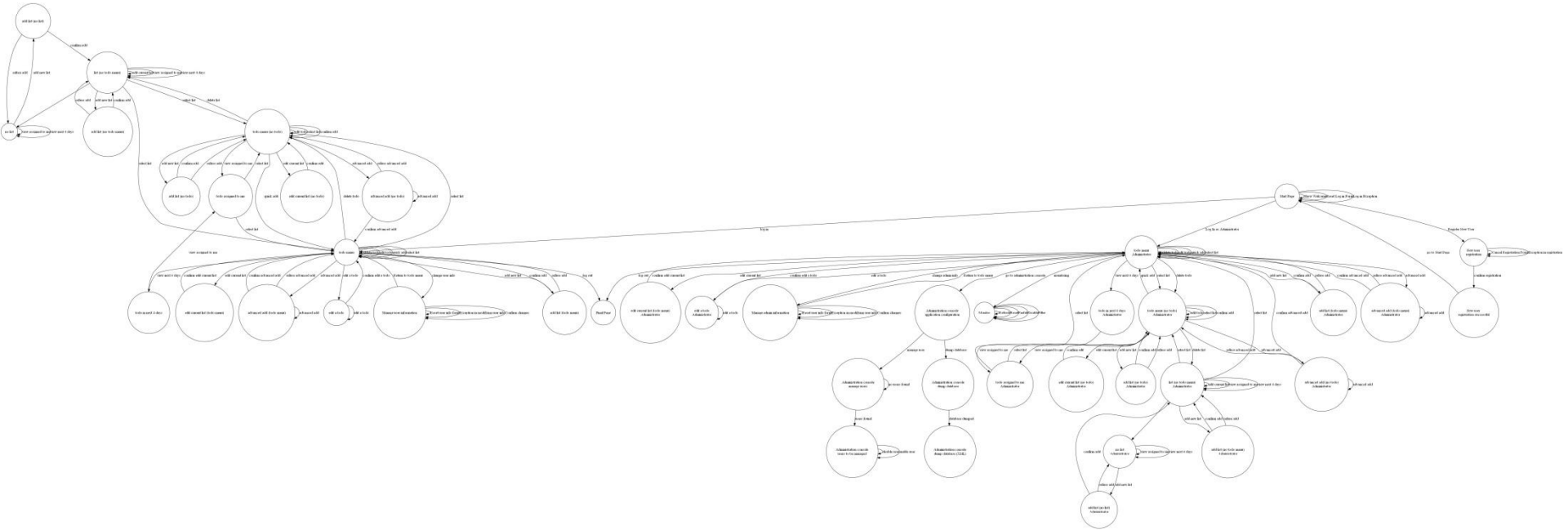


Navigation Graph Generation



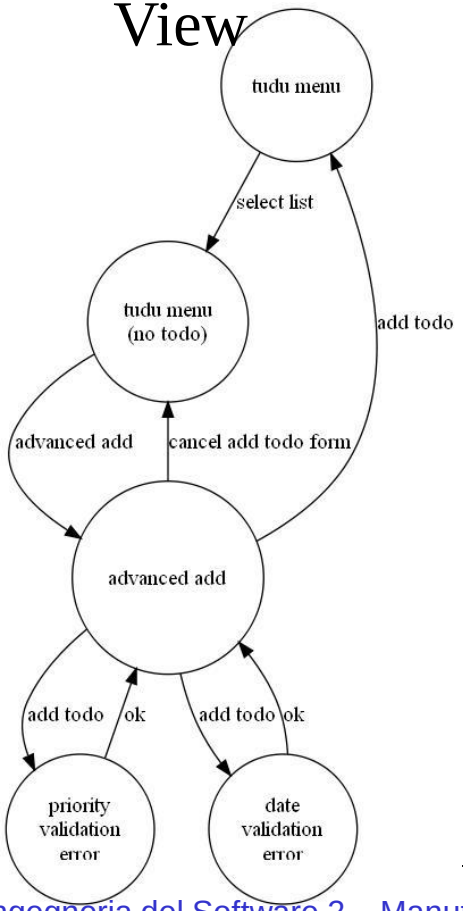


Overall Navigation Graph

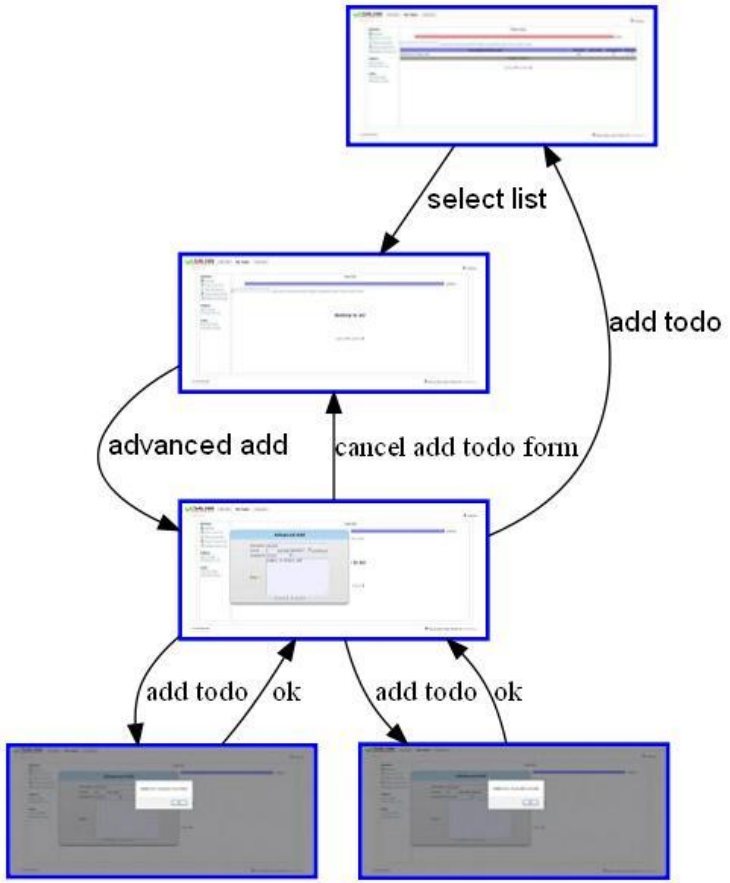


Esempi di scenari di navigazione

Classic Graph View



Graph view with Screenshots

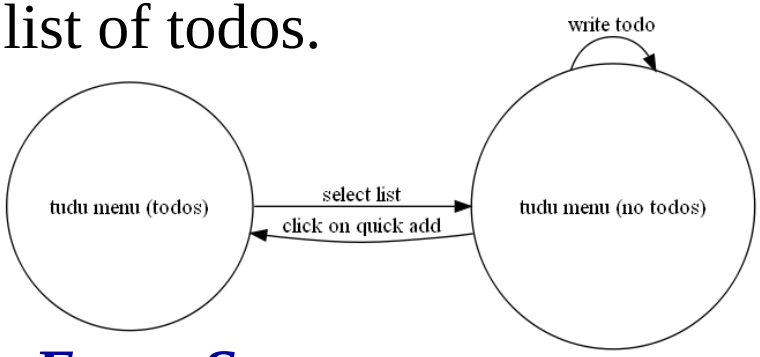


Clickable nodes

Esempio di descrizione di uno scenario

Quick Add

In this scenario a logged user can insert a tudu in a selected list of todos.



Event Sequences

Related Regular Scenarios

Advanced Add

Related Exception Scenarios

Insertion of Wrong Data

Scenario Tutorial example (1/4)

Tutorial: ***Advanced Add***

Actor: Logged User, Administrator
Use case: Advanced Add
Scenario Description: Insertion of a todo in a todo list by specifying a description, a positive level of priority, a future due date, the user to which the todo is assigned and an annotation

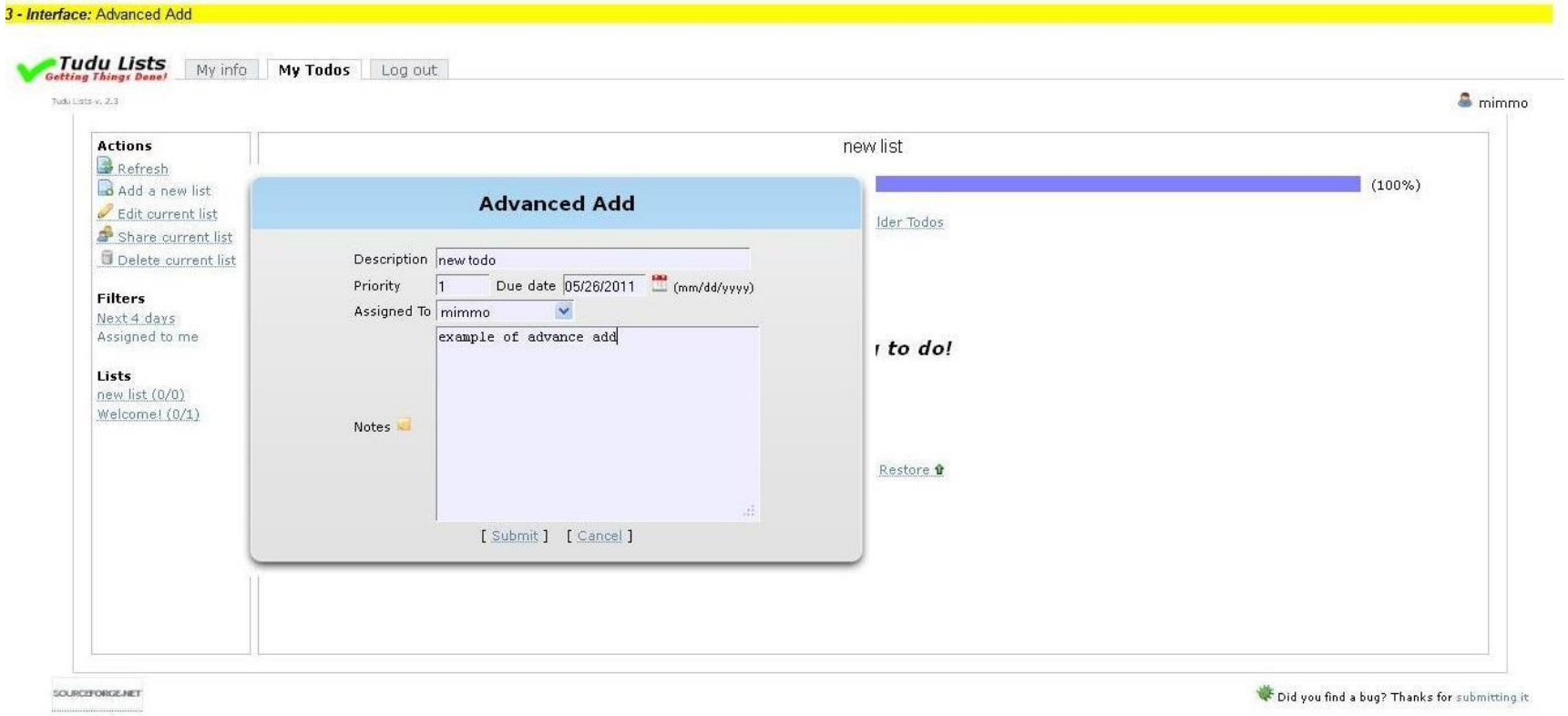
1 - Interface: tudu menu

Transition: select list
Event: click on link "st (0/0)"

Scenario Tutorial example (2/4)

Scenario Tutorial example (3/4)

3 - *Interface:* Advanced Add



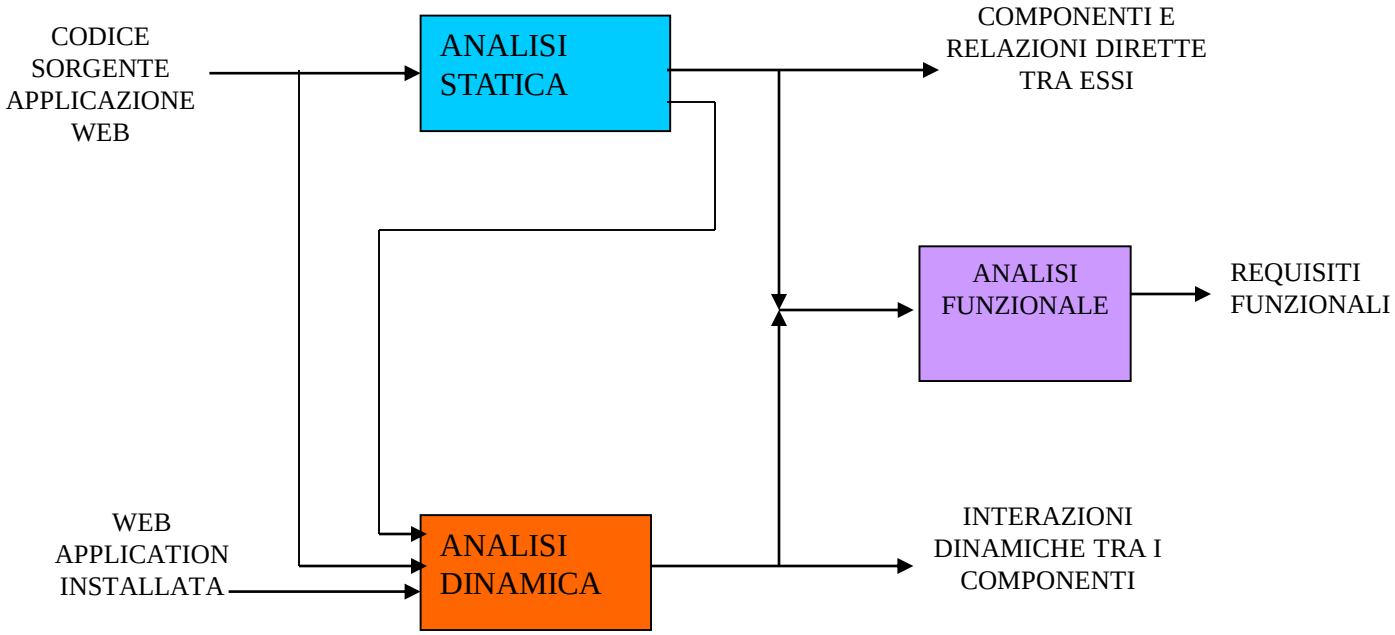
Transition: add todo
Input:
insert *'new todo'* in *description Text field*
insert *'1'* in *priority Text field*

insert *'05/26/2011'* in *dueDate Text field*
select *'mimmo'* in *assignedUser Select field*
insert *'example of advanced add'* in *notes Text area*
Event: click on link "Submit"

Scenario Tutorial example (4/4)

Appendice

Esempio (d'annata):
Reverse Engineering di applicazioni web



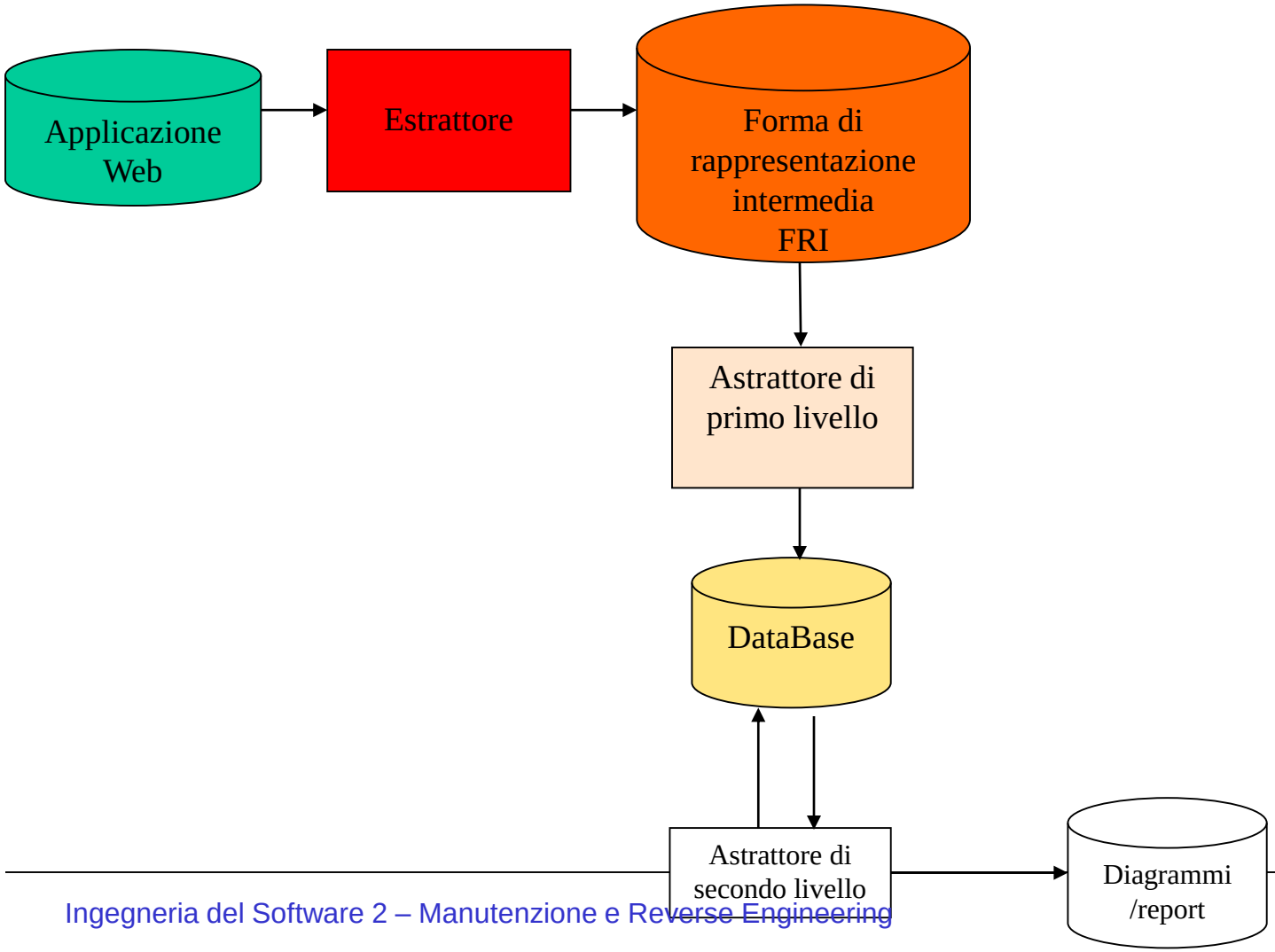
Analisi statica vs Analisi Dinamica

- L'analisi statica:
  - Deve essere effettuata necessariamente sul codice sorgente
    - Possibile solo per li sviluppatori dell'applicazione
  - Estrae solo un sottoinsieme delle informazioni
    - Ad esempio, nei software object oriented non può estrarre gli oggetti istanziati dinamicamente (e nemmeno le eventuali classi dichiarate a tempo di esecuzione)
  - Non va a modificare il codice sorgente

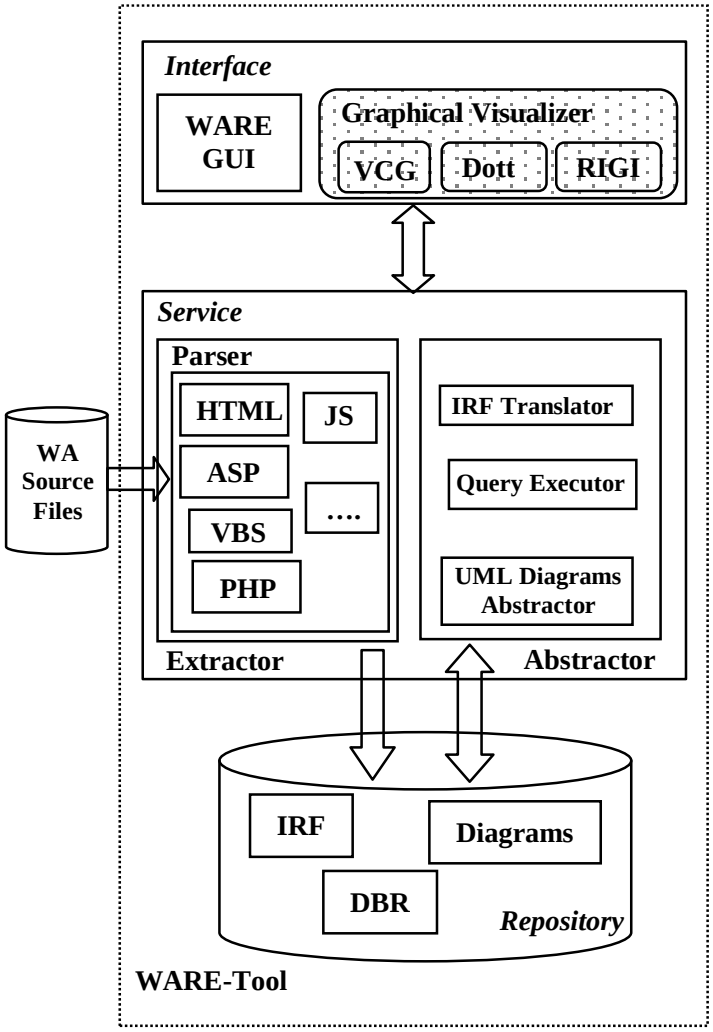
- L'analisi dinamica:
  - Può essere effettuata anche dagli utenti dell'applicazione
  - Potenzialmente può estrarre tutte le interazioni che vengono in essere durante l'esecuzione del software
  - Necessita di sonde da inserire nel codice (per esportare dati)
  - Non ha una terminazione
    - Dovrebbe riferirsi ad un insieme "significativo" di esecuzioni dell'applicazione
      - Ad esempio una test suite che soddisfi un certo criterio di copertura

Un tool di supporto all'analisi statica:

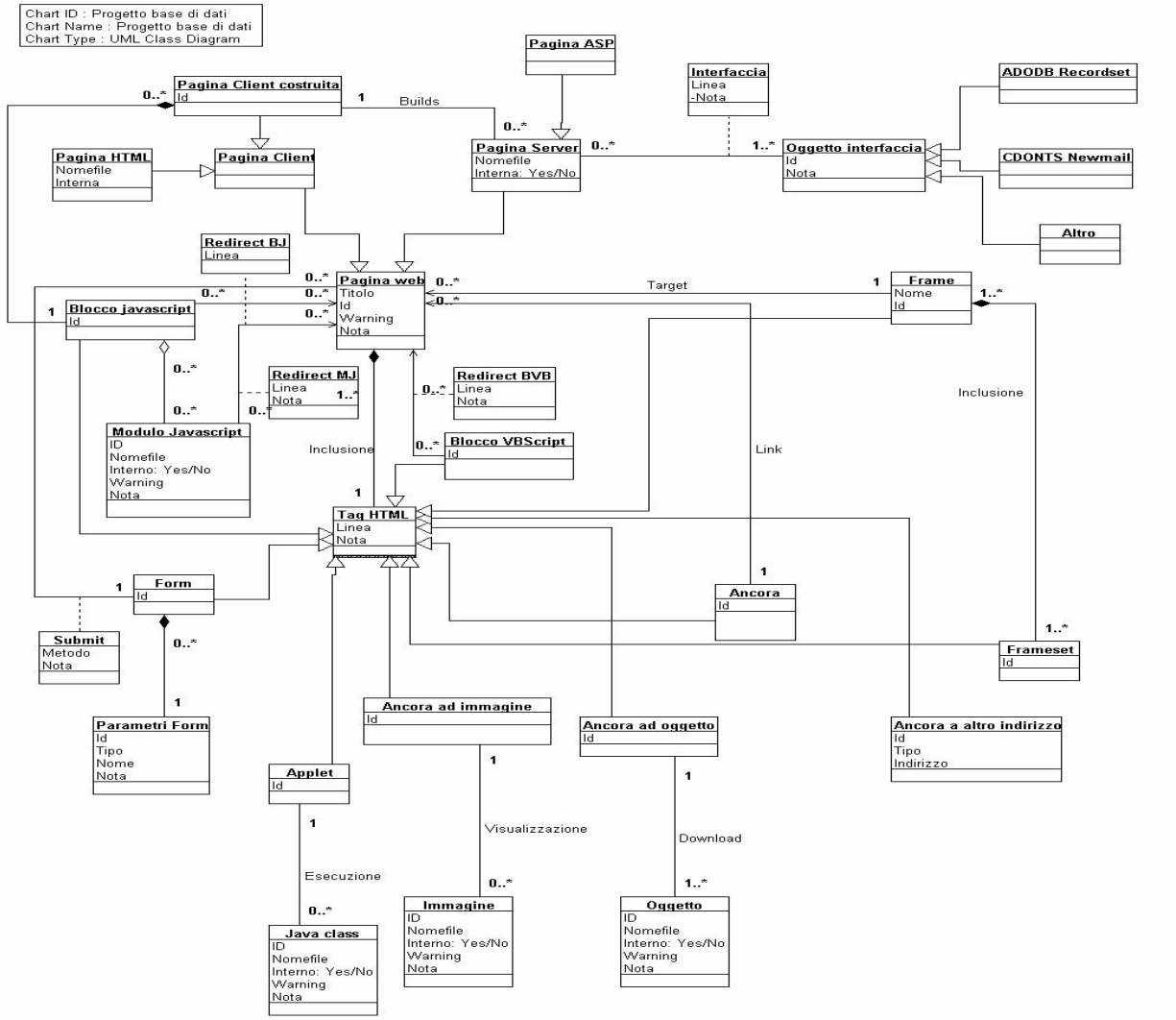
Flusso dei dati



Vista architetturale



Sviluppo del tool estrattore: Analisi delle informazioni



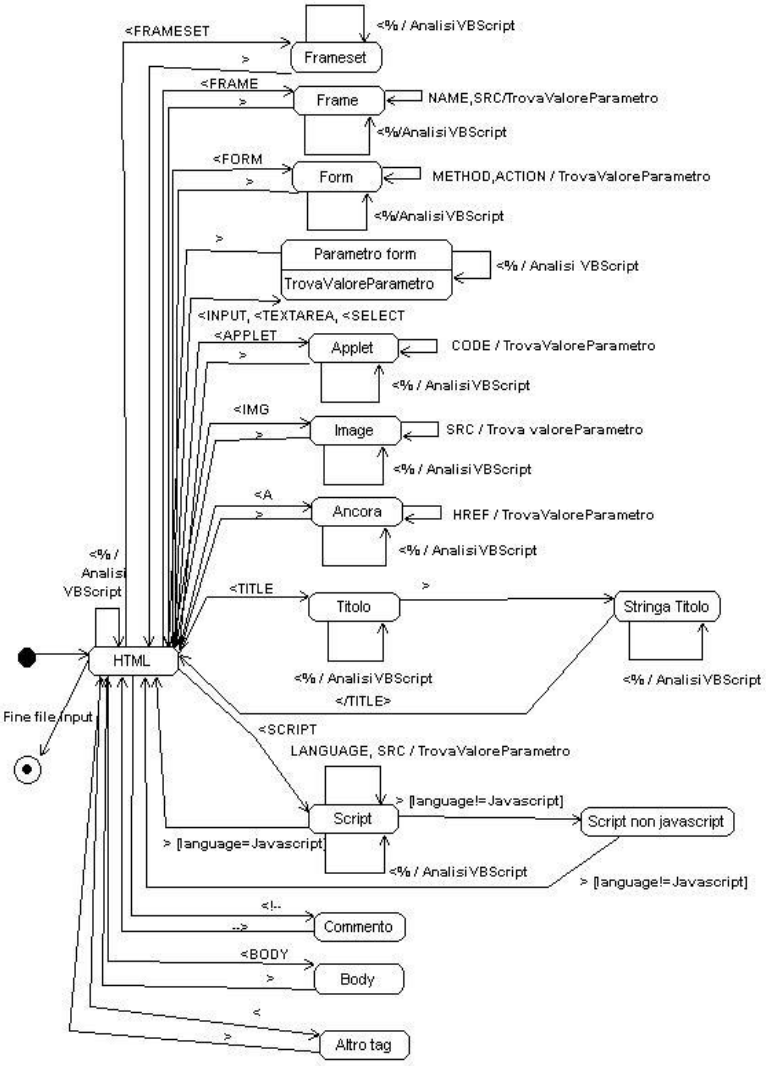
Sviluppo del tool estrattore:
Definizione della forma di rappresentazione intermedia

- Censimento di tutte le espressioni del codice sorgente che devono essere riconosciute: (tag HTML - comandi Javascript -comandi VBScript - eventi di apertura e chiusura file)
- Per ognuna di queste espressioni viene definita la sintassi del tag corrispondente nella forma di rappresentazione intermedia
- Per ogni tag della forma di rappresentazione intermedia viene definita la semantica

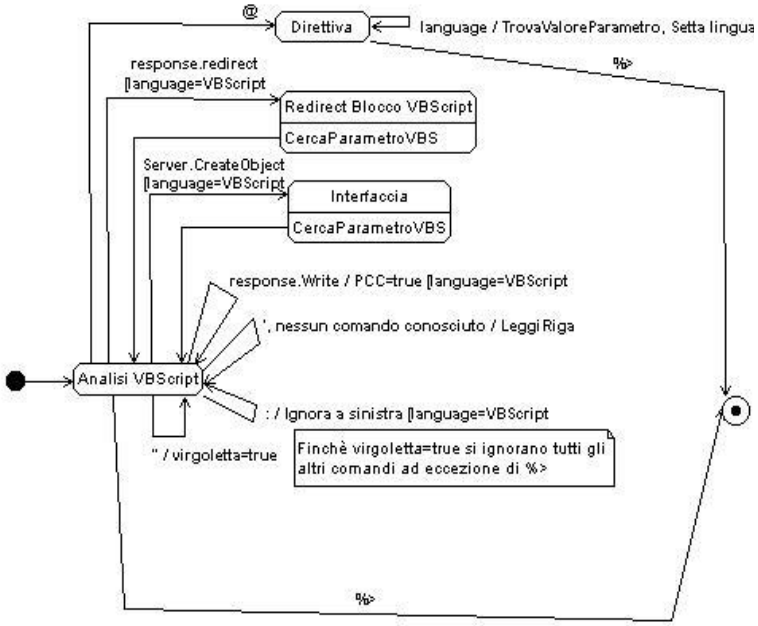
```
<APERTURA>
        <NOMEFILE="\index.htm">
</APERTURA>
<TITOLO>
        <TITOLO="Giuridea - Forum e Laboratorio
Giuridico">
</TITOLO>
<APERTURA BLOCCO JAVASCRIPT>
        <LINEA=22>
</APERTURA BLOCCO JAVASCRIPT>
<CHIUSURA BLOCCO JAVASCRIPT>
<IMMAGINE>
        <LINEA=50>
        <NOMEFILE="images/title.jpg">
</IMMAGINE>
<ANCORA>
        <LINEA=51>
        <NOMEFILE="Archivio/Archivio.htm">
</ANCORA>
```

Sviluppo del tool estrattore: Implementazione

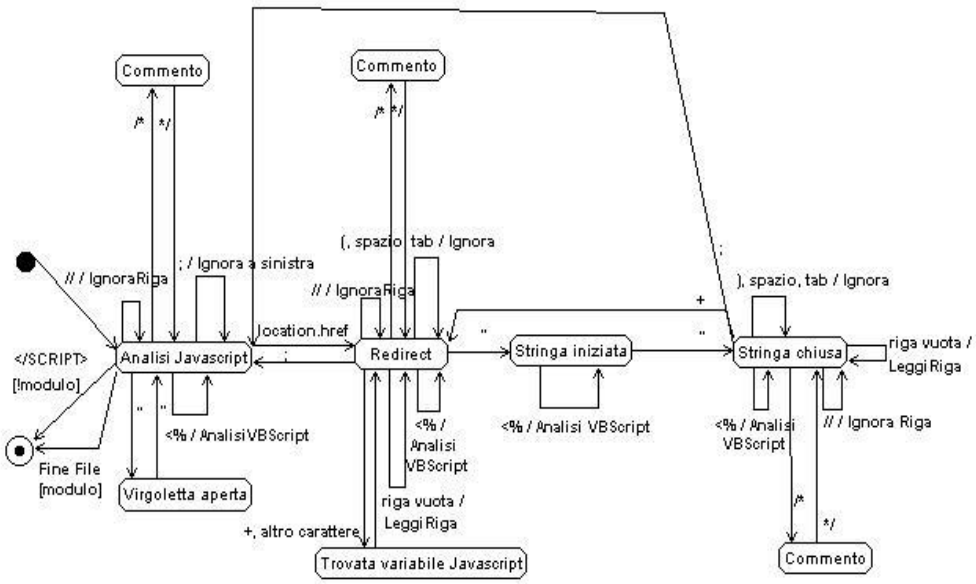
Statechart raffigurante l'automa riconoscitore di tag HTML



Sviluppo del tool estrattore: Implementazione



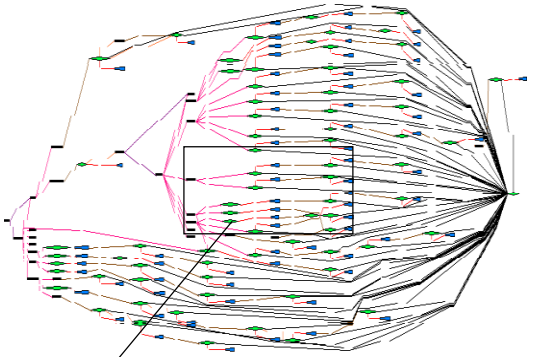
Automa riconoscitore di parole chiave VBScript

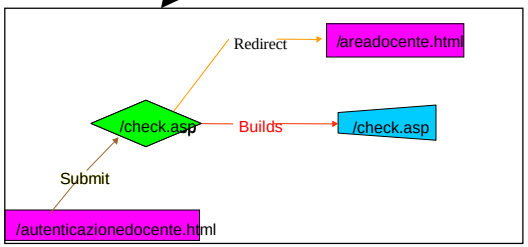


Automa riconoscitore di parole chiave Javascript

Tool astrattore

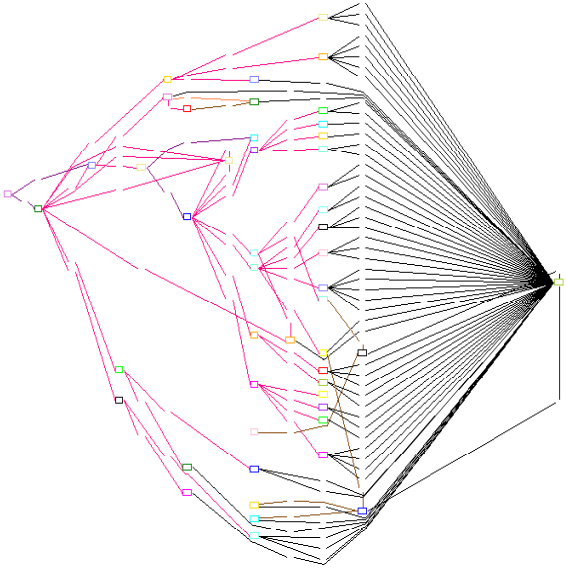
Diagramma delle pagine

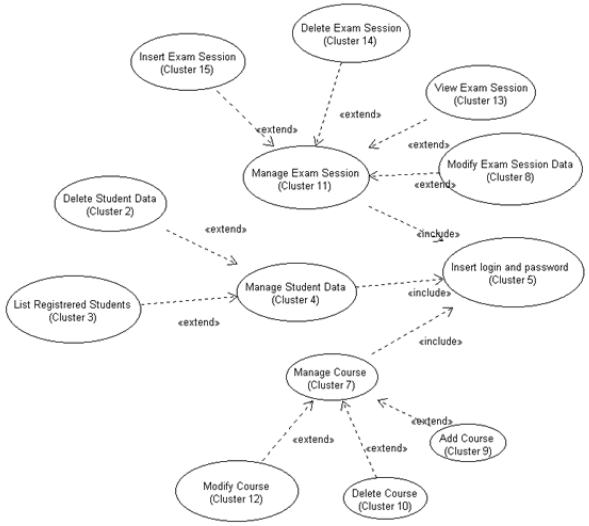




Euristiche per il raggruppamento delle pagine

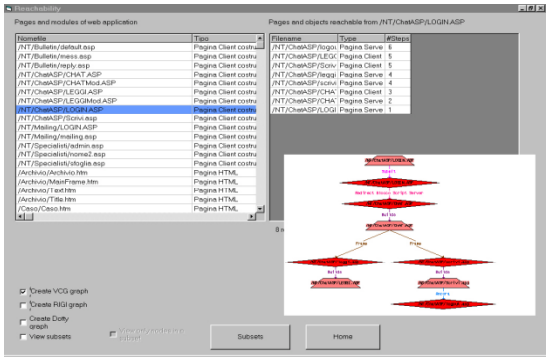
Diagramma dei gruppi di pagine correlate



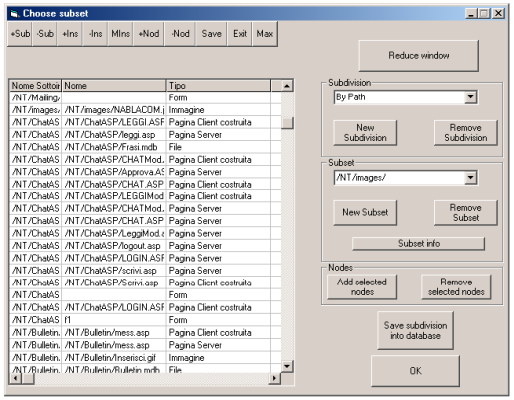


Validazione dei gruppi e loro interpretazione come casi d'uso

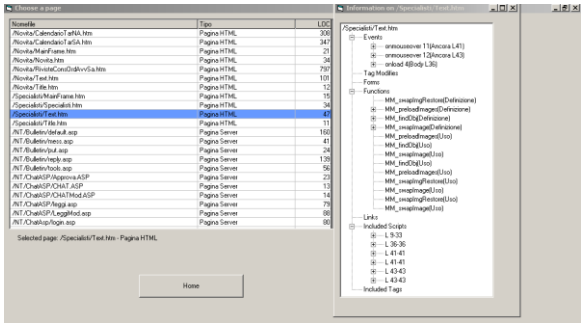
Tool Visualizzatore



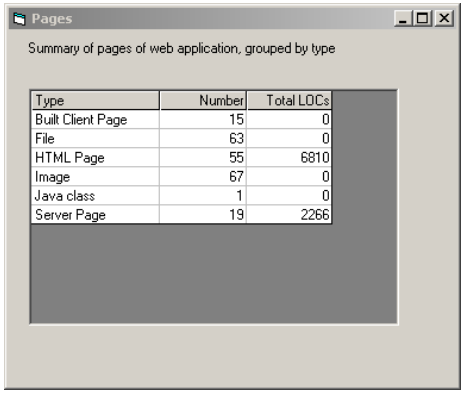
WA components reachable from a selected component



Definition of sub sets of WA components



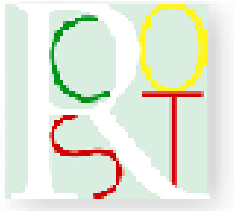
Page list and information extracted from a client page



Form showing some WA metrics calculated by WARE

Appendice

Esempio:
Migrating Interactive Legacy Systems To Web Services



Porfirio Tramontana

Anna Rita Fasolino

Giovanni Frattolillo

Dipartimento di Informatica e Sistemistica

University of Naples Federico II, Italy

Gerardo Canfora

RCOST – Research Centre on Software Technology

University of Sannio, Benevento, Italy

Three basic questions …

1. What to expose as a Web Service?
2. When the migration is convenient?
   - G. Lewis, E. Morris and D. Smith have approached this question in the yesterday tutorial and in the previous talk …
   - S. Tilley, J. Gerdes, T. Hamilton, S. Huang, H. Muller, K. Wong also outline the challenges inherent in migrating to Web services
3. Which approaches for the migration?
   - Sneed and Sneed present a tool supported process to make accessible selected sections of legacy code as Web Services;
   - E. Stroulia, M. El-Ramly, P. Sorenson propose methods based on the analysis of screen features and on the tracing of user interactions to reverse engineering interfaces of an interactive legacy system in order to support the migration

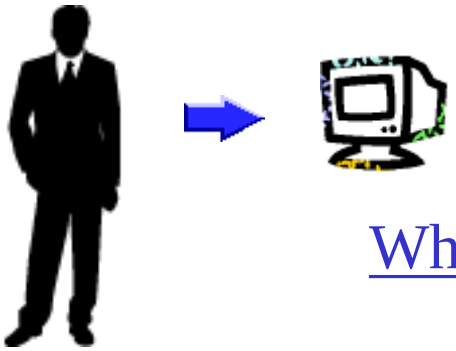
A specific problem:
the migration of interactive legacy system functionalities toward Web Services

Comparing Interaction paradigms…

Form based Interactive Systems

Users query the system by inputting data and sending commands, by interacting with the user interface.

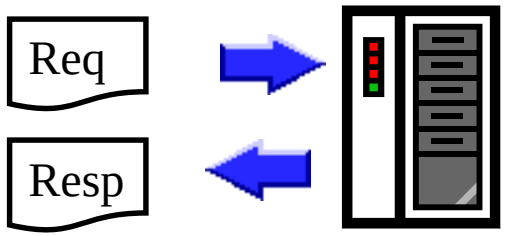
System answers by producing a response screen, containing output values and new input fields and command buttons



Web Services

A Client party invokes a service implemented by a provider party, using a request message.

The provider processes the request and sends a response message with the obtained results.

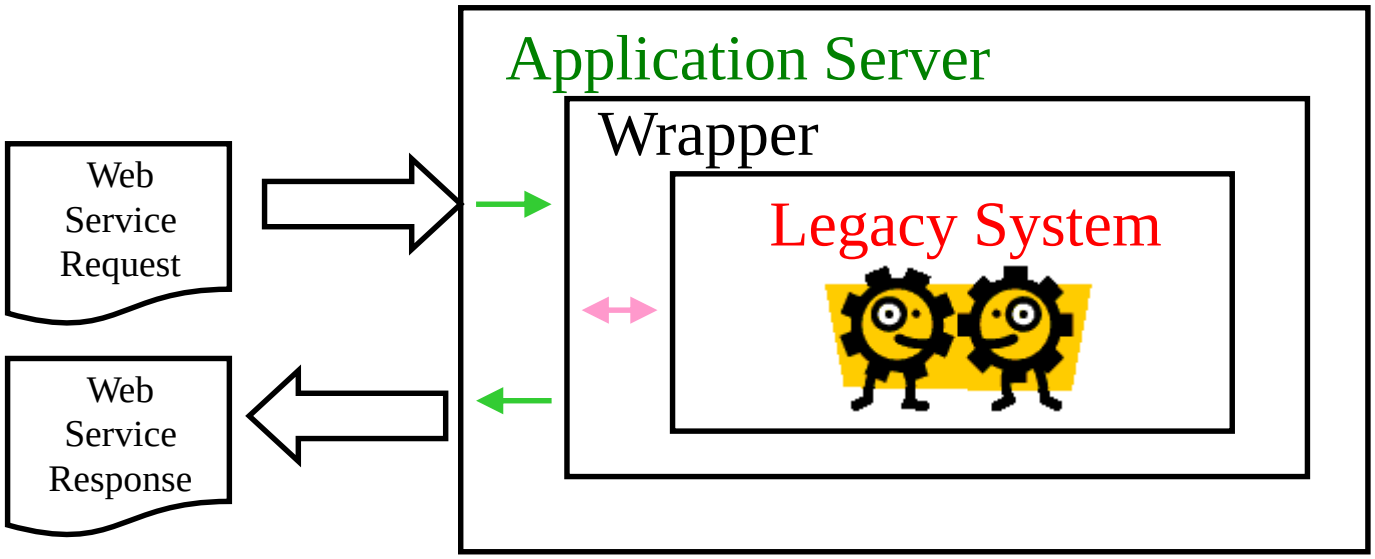


Which approaches for the migration?

Wrapping

The Wrapper

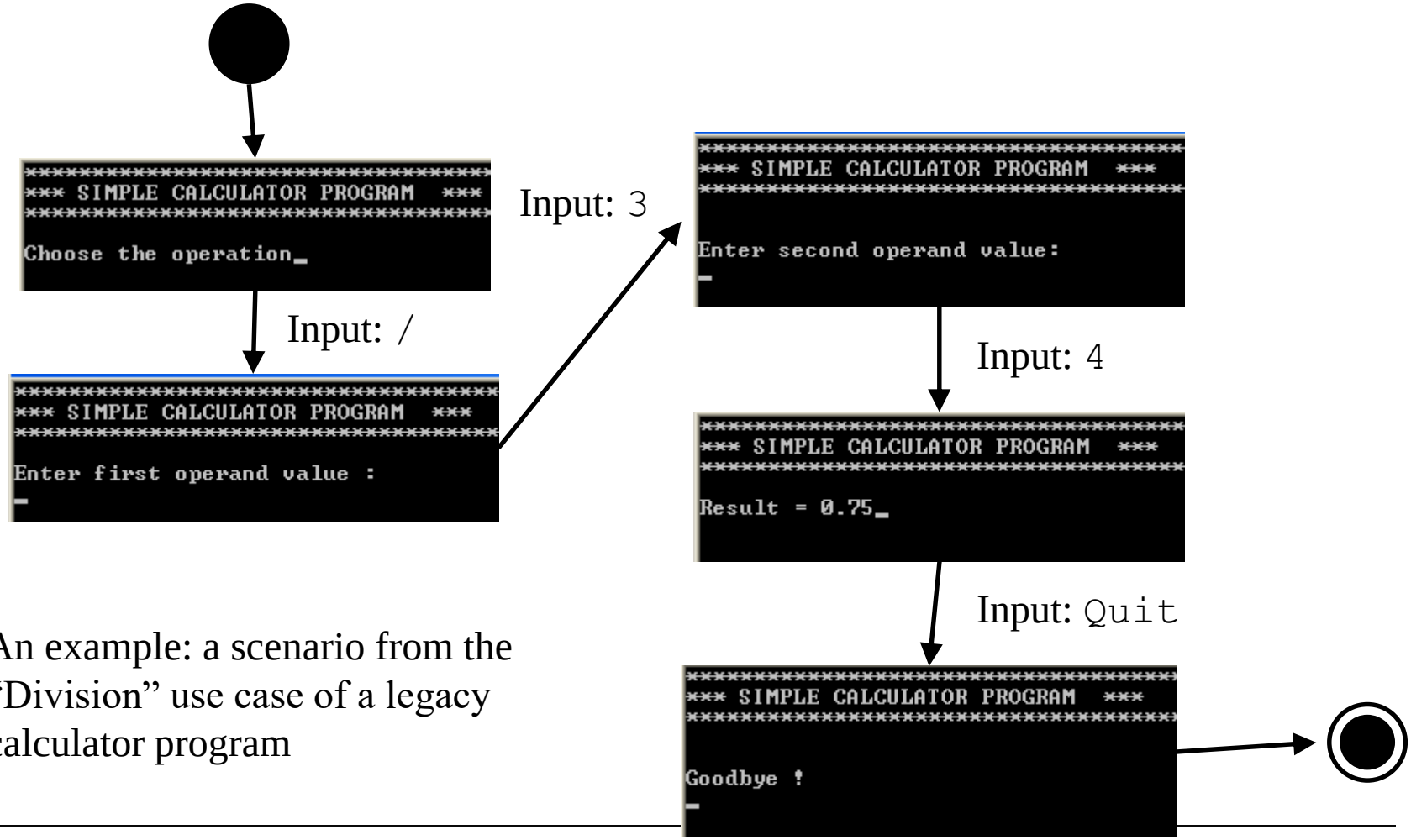
- The goal of the wrapper is to drive the legacy system during the execution of each possible interaction scenario associated with the use case to migrate, by providing it with the needed flow of data and commands.
- The wrapped legacy system use case is accessible as a Web Service



A key requirement of the Wrapper

- The wrapper must be reusable for migrating different use cases, so…
- The wrapper behavior requested for each use case will not be embedded in the wrapper…
- But it will be separately specified for each use case
- A key question: obtaining for each use case a complete model of the interaction between the legacy system and the user
- A Reverse engineering problem!

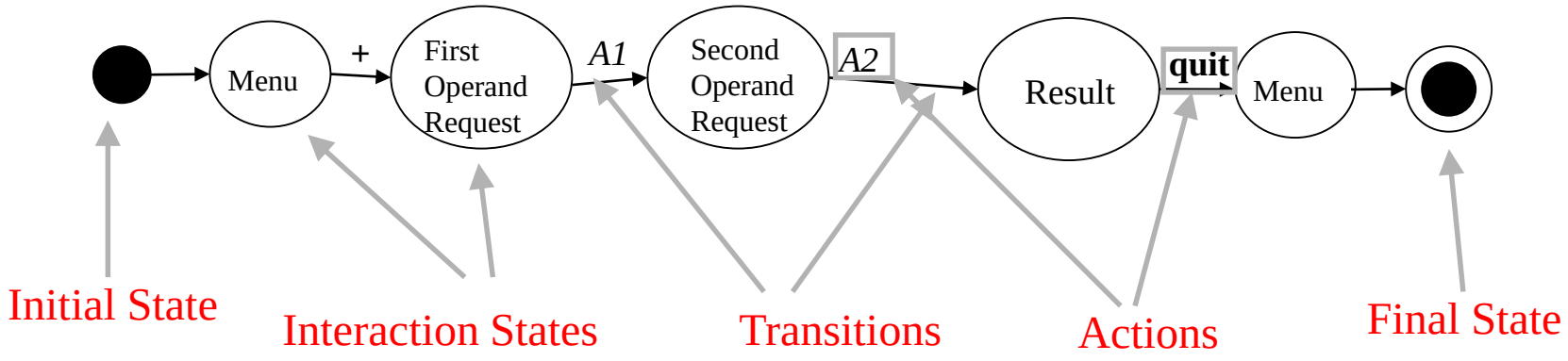
Modelling Interactions between User and Legacy System



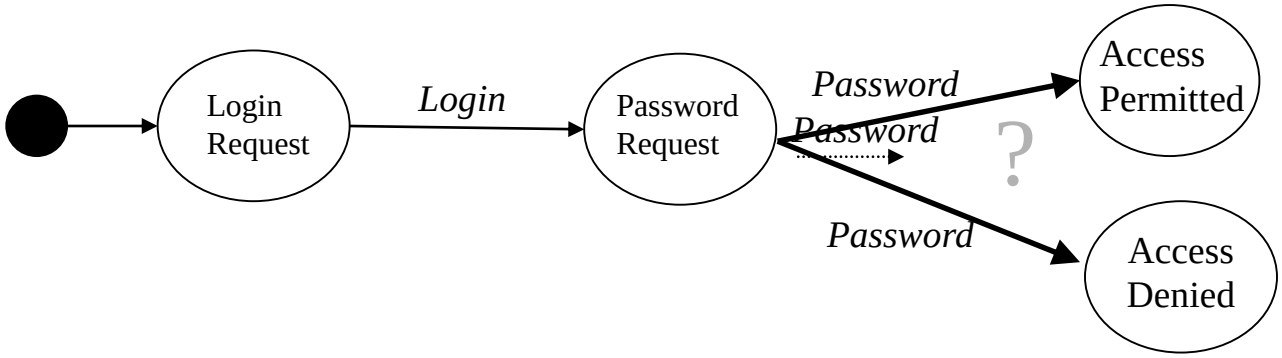
An example: a scenario from the "Division" use case of a legacy calculator program

The Model of the Interaction

- A Finite State Automaton FSA= (S, T, A, Sin, Sfin) where:
  - S is the set of Interaction States,
  - A is the set of Actions performed by the user when an Interaction State occurs,
  - T is the set of Transitions between states,
  - Sin and Sfin are the Initial and Final states of the interaction.



A problem

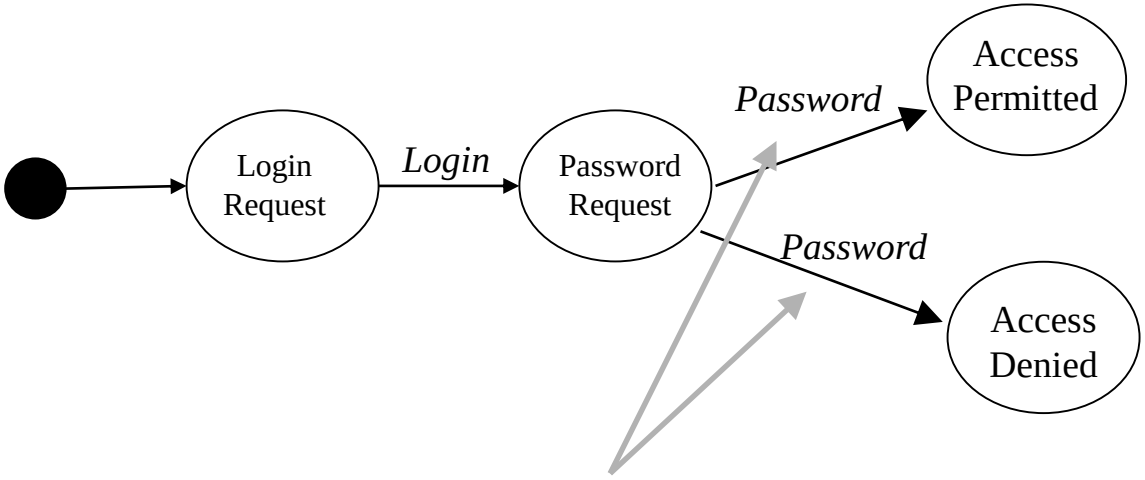


What the next State?

- The next state depends on the internal logic or on the internal state of the legacy system.
- A solution: Non Deterministic Finite State Automata
  - a Non Deterministic Finite State Automaton (NFA) is a finite state machine where for each pair of state and input symbol there may be several possible next states

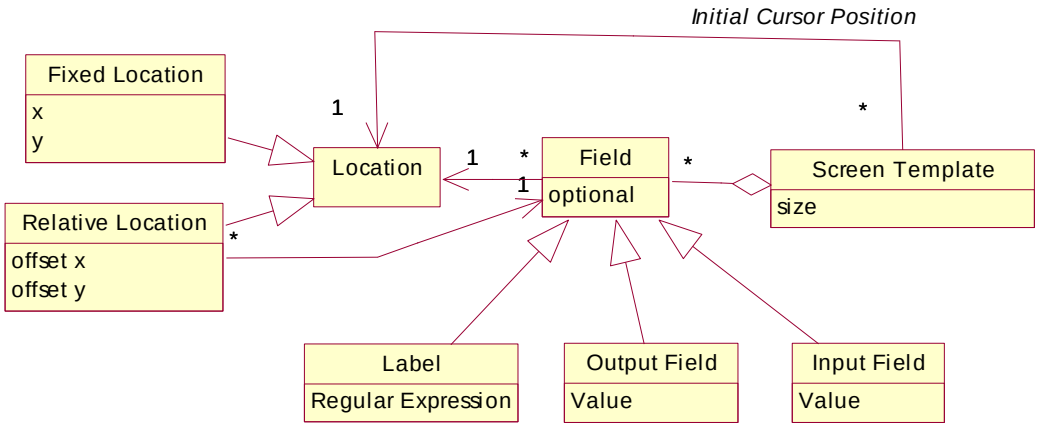
Another Wrapper Requirement

- The wrapper must know the list of the possible Next States of a given State
  - Possible successors of `Password Request` State are `Access Permitted` and `Access Denied` states
- The wrapper must be able to identify the current state on the basis of the returned screen
  - Wrapper must discriminate among `Access Permitted` screen and `Access Denied` screen



Screen Templates

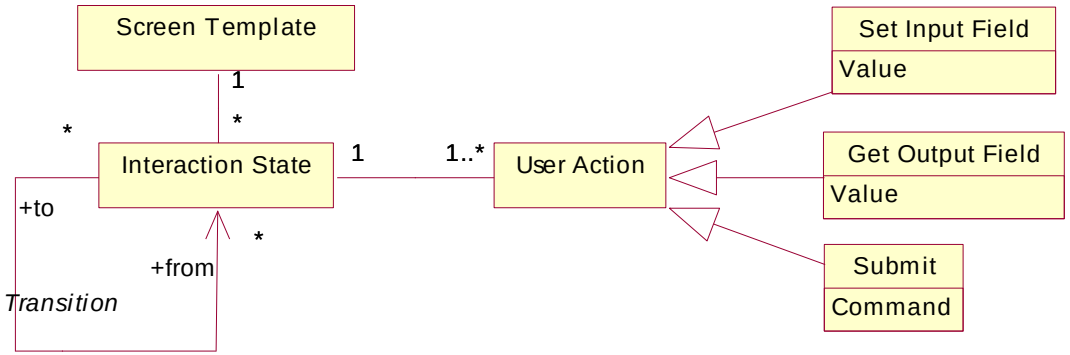
- A description of Legacy Screen is needed for the identification: Screen Templates
  - A Screen Template is a collection of Fields:
    - Labels;
    - Input Fields;
    - Output Fields;
  - Each field has a Location on the Screen. Location may be defined as a:
    - Fixed Location, i.e. coordinates of the field;
    - Relative Location, i.e. distance from another field.



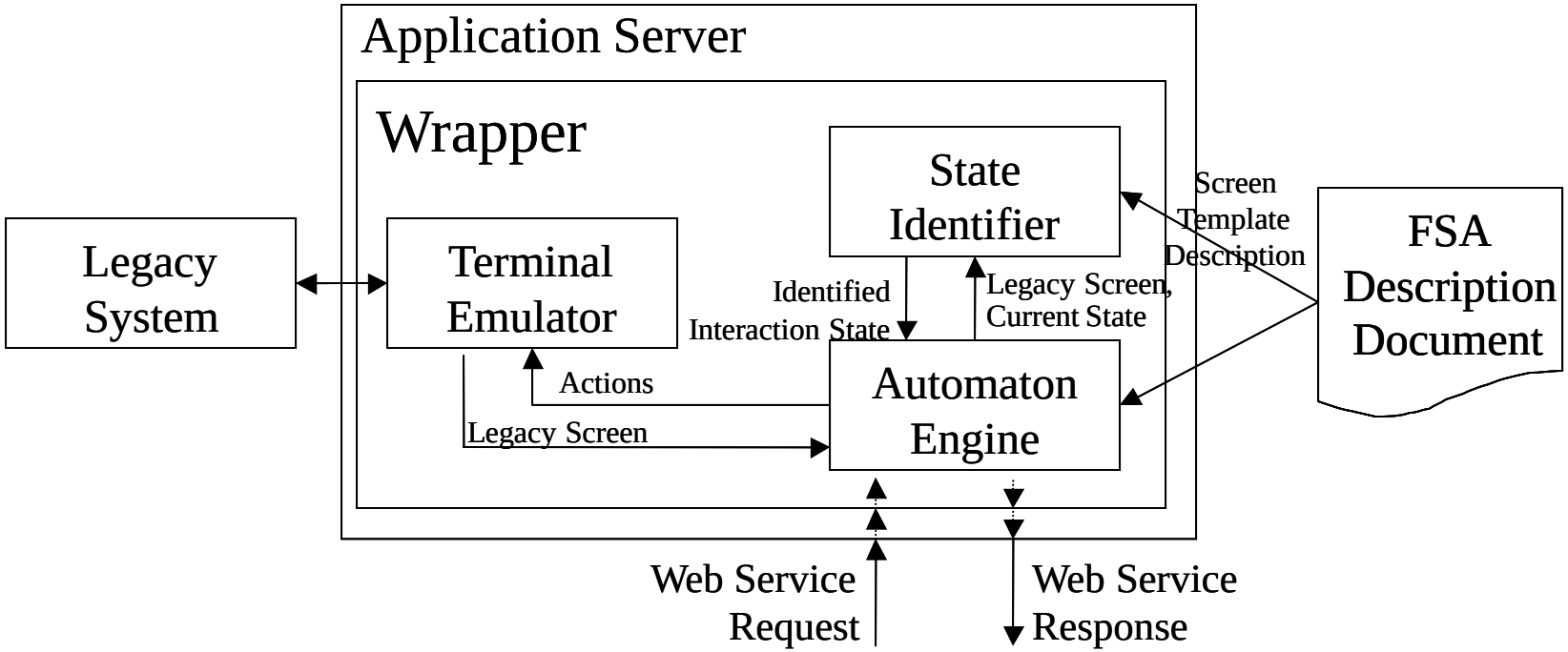
Characterising Interaction States

- An Interaction State is characterised by a Screen Template and a set of actions to perform on its fields, causing transitions to other Interaction States

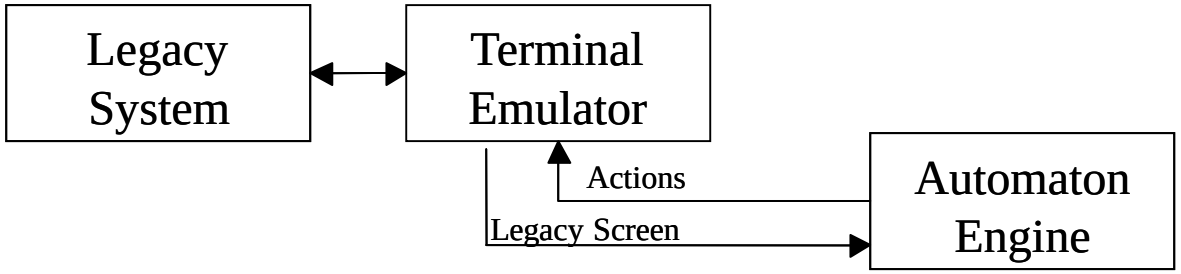
- User Actions may be:
  - Set Input Field Actions
  - Get Output Field Actions
  - Submit Command Actions



Wrapper Architecture



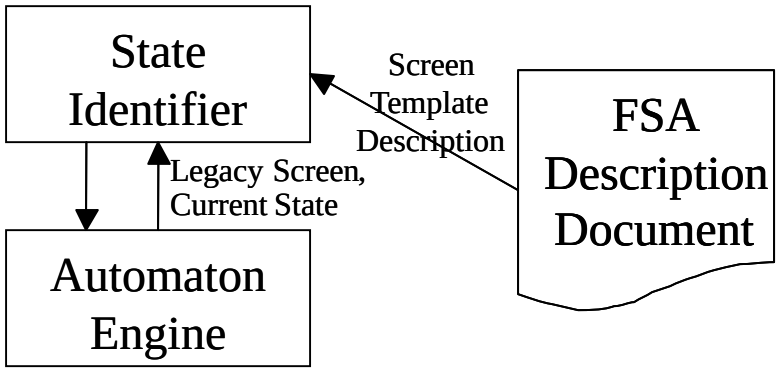
Terminal Emulator



- The Terminal Emulator component is responsible for the dialogue between the Wrapper and the Legacy System terminal
  - Different implementations of the Terminal Emulator are needed for different Legacy System Terminals
    - Stream Oriented terminals;
    - Block oriented terminals;
    - Web Applications.

State Identifier

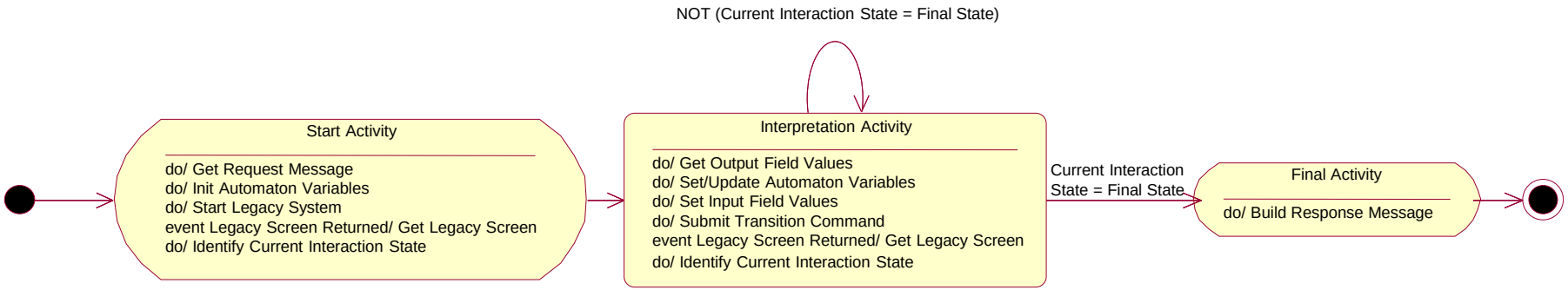
- The State Identifier component is responsible for the identification of the Interaction State reached by the Legacy System



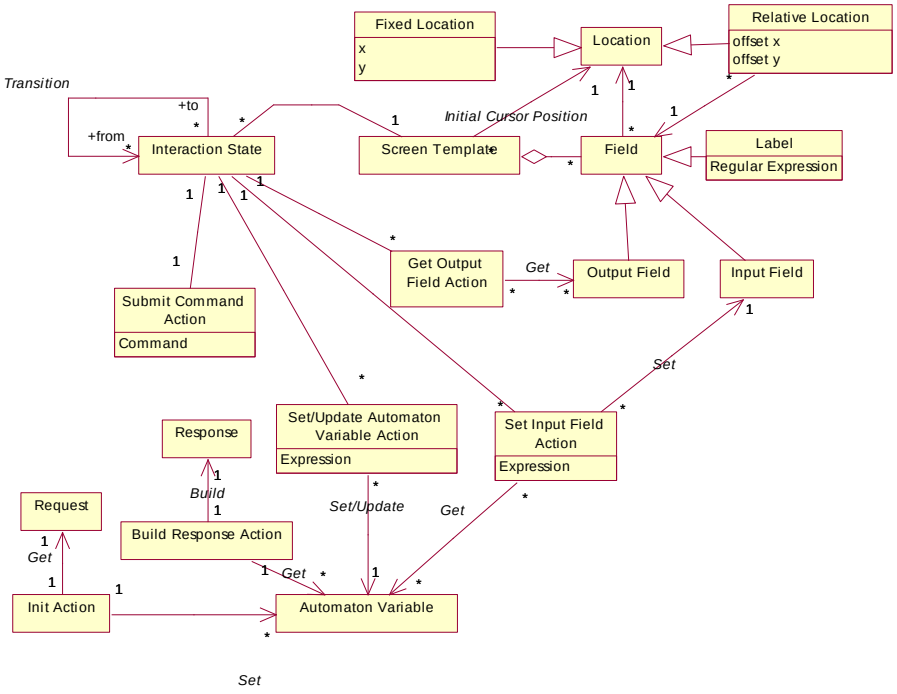
  - It has to match the current screen of the legacy system with the Screen Templates associated with potentially reachable Interaction States
  - The Screen Templates descriptions are part of the Automaton Description Document
- Moreover, the State Identifier
  - localises Labels
  - localises Input Fields
  - Localises Output Fields and read their values

Automaton Engine

- The Automaton Engine is responsible for interpreting the FSA associated with a given service offered by the legacy system. It:
  - Sends commands to and receives screens from the Terminal Emulator
  - Queries the State Identifier about the identification of the Current Interaction State
  - Interprets the request message received from the application server
  - Builds the response message and sends it to the application server
  - Manages Automaton Variables (i.e. temporary variables needed to save intermediate results of the execution of the Automaton)



Finite State Automaton Description Document



Automaton Description Document UML model

```
<automa>
 <automa-states>

….

 <state id="go_to_header" type="automa"
screen="PineGoToHeaderScreen">
       <description>State
       </description>
       <layout>
        <location x="1" y="2"/>
        <size width="8" height="2"/>
       </layout>
       <actions>
          <set-fields-action>
           <field ref="prompt">
              <data ref="/root/header"/>
            </field>
          </set-fields-action>
       </actions>
       <next-states>
          <next-state ref="bad_header">
       </next-state>
          <next-state ref="header">
      </next-state>
      </next-states>
     </state>
```

```
<screen id="GoToHeader">
 <size width="80" height="25"/>
 <simple-field id="GoToHeaderId" optional="false"
input="false">
    <fixed-location>
       <point x="0" y="22"/>
    </fixed-location>
    <content pattern="Message number to jump to"
length="25"/>
 </simple-field>
 <simple-field id="prompt" optional="true"
input="true">
      <fixed-location>
         <point x="25" y="22"/>
       </fixed-location>
      <content pattern="" length="5"/>
    <focus order="1">
      <advance-key id="ENTER"/>
    </focus>
 </simple-field>
 <caret-location>
  <fixed-location>
     <point x="28" y="22"/>
  </fixed-location>
 </caret-location>
</screen>

…

</automa-states>
</automa>
```

An excerpt of an Automaton Description Document

A Case study

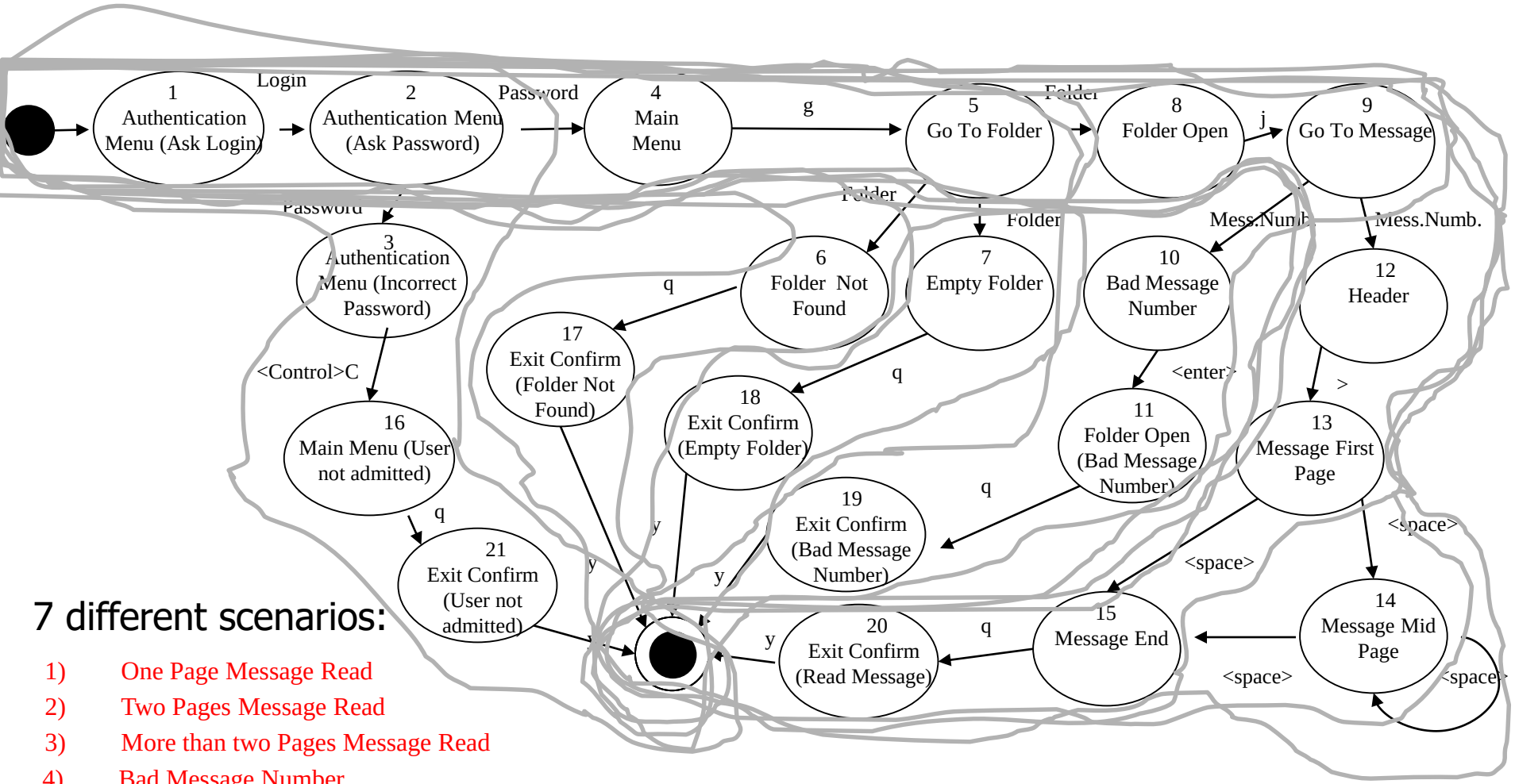
- A migration case study has been carried out according to a process including the following phases:
  - Identification, i.e. reverse engineering of the interaction model;
  - Design, i.e. defining the FSA describing the Wrapper behaviour;
  - Implementation, i.e. realisation of the XML FSA Description Document;
  - Web service deploy, i.e. deployment of the wrapper in the context of an application server;
  - Validation, i.e. testing of the scenarios of the migrated use case

A Case Study

- Legacy system: Pine (ver. 4.64)
  - client mail software, that allows a user to read, compose and manage e-mail messages from an existing message box.
- Pine is a form based legacy system based on *stream oriented* terminals.
  - Usually, Pine is accessible via the Telnet protocol.
- We submitted to the migration process the *Get Message* use case that allows the owner of a mailbox to get the text of a specific e-mail message contained in a specific mailbox folder.

Use case:	Get Message
Preconditions	None
Input	Login, Password, Folder, Message Number
Output	Date, From, To, cc, Subject, Body, Exception
Postconditions	None

The Automaton: graphical view



7 different scenarios:

1) One Page Message Read
2) Two Pages Message Read
3) More than two Pages Message Read
4) Bad Message Number
5) Empty Folder
6) Folder Not Found
7) Incorrect Password

The Automaton: a tabular specification

Interaction State ID	Interaction State Description	Actions	Submit Command	Next State
START		**Init** (*Login, Password, Folder, Message Number*)		1
1	Authentication Menu (Ask Login)	**Set Input Field**: *Login*	<Enter>	2
2	Authentication Menu (Ask Password)	**Set Input Field**: *Password*	<Enter>	3,4
3	Authentication Menu (Incorrect Password)	**Set Automaton Variable**: *Exception* = "Incorrect Login and Password"	<Control>C	16
4	Main Menu		g	5
5	Go To Folder	**Set Input Field**: *Folder*	<Enter>	6,7,8
6	Folder Not Found	**Set Automaton Variable**: *Exception* = "Folder not found"	q	17
7	Empty Folder	**Set Automaton Variable**: *Exception* = "No messages in the folder"	q	18
8	Folder Open		j	9
9	Go To Message	**Set Input Field**: *Message Number*	<Enter>	10,12
10	Bad Message Number	**Set Automaton Variable**: *Exception* = "Incorrect Message Number"	<Enter>	11
11	Folder Open (Message not found)		q	19
12	Header		>	13
13	Message First Page	**Get Output Fields**: (*Date, From, To, Cc, subject,, Body*); **Set Automaton Variables**: (Output: (*Date, From, To, Cc, subject, Body*))	<Space>	14,15
14	Message Mid Page	**Get Output Field**: *Body* **Update Automaton Variable**: *Body* = *Body* + Output:*Body*	<Space>	14,15
15	Message End	**Get Output Field**: *Body* **Update Automaton Variable**: *Body* = *Body* + Output:*Body*	q	20
16	Main Menu (User not admitted)		q	21
17	Exit Confirm (Folder Not Found)		y	END
18	Exit Confirm (Empty Folder)		y	END
19	Exit Confirm (No Message)		y	END
20	Exit Confirm (Read Message)		y	END
21	Exit Confirm (User not admitted)		y	END
END		**Build Response**: (*Date, From, To, Cc, Subject, Body, Exception*)		

Testing Strategy

- A Test Suite comprehending 7 Test Cases had been selected in order to cover the 7 linear independent paths individuated on the FSA

TC#	TC Description	Interaction State Sequence
1	One Page Message Read	S-1-2-4-5-8-9-12-13-15-20-E
2	Two Pages Message Read	S-1-2-4-5-6-9-12-13-14-15-20-E
3	More Than Two Pages Message Read	S-1-2-4-5-8-9-12-13-14-14-15-16-21-E
4	Bad Message Number	S-1-2-4-5-8-9-10-11-19-E
5	Empty Folder	S-1-2-4-5-7-18-E
6	Folder Not Found	S-1-2-4-5-6-17-E
7	Incorrect Password	S-1-2-3-16-17-E

  - We noticed that all the 7 scenarios of the migrated use case had been covered by the selected Test Suite