



UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTA DI SCIENZE MATEMATICHE, FISICHE E NATURALI
COMUNICAZIONE DIGITALE

MODERNIZZAZIONE DI SISTEMI LEGACY

Dalle mappe BMS alle pagine Web

Il laureando:

Francesca Madeddu
Matr. 756354

Relatori:

Prof. Lorenzo Capra
Dott. Fabrizio Annatelli

Aprile 2013

RIASSUNTO

La maggior parte dei sistemi informatici attualmente considerati *mission critical* sono sistemi legacy realizzati e mantenuti tramite l'utilizzo di tecnologie ormai obsolete. Oggi esistono più di 200 miliardi di linee di Cobol equivalente all'80% del codice utilizzato a livello mondiale e sono in esercizio applicazioni mainframe per un valore di 2 miliardi di dollari, con le quali vengono gestite circa il 70% di tutte le transazioni eseguite a livello mondiale. Le applicazioni legacy gestiscono ogni aspetto della nostra vita: dai semafori stradali ai telefoni cellulari. Queste applicazioni portano con sé molti problemi. Sono difficili da mantenere e migliorare a causa della carente conoscenza interna del sistema; rendono difficile l'innovazione poiché la maggior parte delle risorse è impegnata nell'attività di manutenzione; hanno costi di gestione molto alti, dovuti sia a questioni prettamente tecnologiche che di risorse umane: è sempre più difficile trovare personale con le competenze adatte; infine sono generalmente caratterizzati da un alto livello di complessità. Oggi le aziende hanno l'esigenza di restare competitive nella recessione del mercato il che significa incrementare i servizi senza aumentare i costi e perseguire l'agilità per permettere di stare al passo con i processi di business: ciò risulta estremamente difficile quando si deve fare affidamento su un'infrastruttura IT datata. In questo contesto assume particolare importanza la modernizzazione, cioè l'applicazione di quelle tecniche che permettono di intervenire su un sistema per adeguarlo ai requisiti attuali.

Scopo di questo lavoro è stato mostrare caratteristiche, vantaggi e limiti di un particolare strumento di modernizzazione di tipo black box, che non indaga cioè il funzionamento interno del sistema. L'applicazione su cui la modernizzazione è stata realizzata è scritta in Cobol, è in esecuzione su ambiente CICS mainframe ed è accessibile tramite terminale 3270. Nonostante l'applicazione sia ancora soddisfacente da un punto di vista funzionale, essa presenta diverse problematiche in termini di usabilità, accessibilità ed efficienza operativa. Infatti anche se i programmi di emulazione consentono agli operatori di eseguire l'applicazione su un normale computer, rimangono comunque i problemi di usabilità legati all'utilizzo della vecchia interfaccia testuale, che rende le operazioni di inserimento, accesso e ricerca dei dati difficoltose e caotiche. Per questi motivi il revamping dell'applicazione, cioè la creazione di una nuova interfaccia grafica, è risultata essere la scelta di modernizzazione più appropriata. A tale scopo è stato utilizzato uno strumento di sviluppo agile per il disegno e l'implementazione di applicazioni web, WebRatio, il quale fornisce un ambiente integrato basato sul linguaggio WebML; il suo approccio model

driven permette di velocizzare e semplificare in modo considerevole la fase di implementazione. Al fine di rendere WebRatio uno strumento specializzato nel revamping è stato integrato con una particolare libreria, la w4bms, che implementa il concetto di screen scraping, cioè un mapping tra le mappe BMS (le “schermate” dell’applicazione legacy) e le pagine web dell’applicazione modernizzata.

Questo tipo di modernizzazione ha portato a risultati estremamente positivi. Per quanto riguarda la fase di sviluppo ha reso possibile il riutilizzo di gran parte dell’applicazione legacy e ha quindi permesso di mantenere bassi i costi e limitato il rischio. Il fatto poi che WebRatio sia uno strumento di modellazione di facile utilizzo ha consentito al personale abituato a lavorare con tecnologie legacy di conoscere e imparare nuovi strumenti tipici del web. I miglioramenti in termini di usabilità sono stati dimostrati tramite l’utilizzo di opportuni test con utenti. Tali miglioramenti si sono tradotti in una maggiore efficienza ed efficacia nell’utilizzo da parte dell’utente finale, oltre che ad un più alto livello di soddisfazione e quindi di produttività. Infine la realizzazione dell’interfaccia web ha permesso all’applicazione di essere utilizzabile attraverso differenti strumenti e da persone affette da disabilità percettive o cognitive. Durante la fase di modernizzazione non sono mancati i problemi, legati sia al tipo di modernizzazione che allo strumento. I primi dipendono dal fatto che il revamping non permette di modificare la logica interna dell’applicazione e quindi i problemi che facevano parte della logica interna dell’applicazione legacy sono ancora presenti in quella modernizzata. I secondi riguardano invece i limiti in termini di personalizzazione che è possibile effettuare. La forza di WebRatio e della w4bms sta nel fatto che essi sono strumenti basati su una modellazione concettuale che permette di gestire un certo numero di situazioni: grazie a ciò si hanno tempi di sviluppo brevi, costi bassi e rischio limitato. Questo però significa che i requisiti dell’applicazione che non ricadono dentro allo standard previsto dallo strumento sono difficili da gestire ed estremamente costosi poiché richiedono una personalizzazione su misura.

In conclusione l’accoppiata WebRatio w4bms è risultata essere un ottimo strumento per la modernizzazione di applicazioni legacy 3270 nel caso in cui i problemi siano legati esclusivamente all’usabilità e all’accessibilità e non si rendano necessarie particolari personalizzazioni.

CONTENTS

I	MODERNIZZAZIONE DI SISTEMI LEGACY	1
1	INTRODUZIONE	3
1.1	Il problema	4
1.2	Le domande di ricerca	4
1.3	Organizzazione dei contenuti	5
2	EVOLUZIONE DEL SOFTWARE	7
2.1	Manutenzione del software secondo lo standard ISO/IEC 14764	8
2.1.1	Tipi di manutenzione	8
2.2	Evoluzione di una sistema secondo Seacord ed al.	10
2.2.1	Manutenzione	11
2.2.2	Modernizzazione	12
2.2.3	Sostituzione	12
2.3	Le leggi di Lehman	12
2.4	I costi dell'evoluzione	14
2.5	Contesto attuale	16
3	SISTEMI LEGACY	19
3.1	Persistenza dei sistemi legacy	19
3.1.1	Criticità	20
3.1.2	Sintesi dei problemi	21
3.2	I sistemi legacy oggi	22
3.2.1	Modernizzazione come opportunità	22
3.3	Grandi aziende e sistemi legacy	23
3.3.1	Mainframe	23
3.3.2	Caratteristiche e prestazioni	24
3.3.3	Il mondo mainframe	24
3.3.4	I costi	26
4	MODERNIZZAZIONE	29
4.1	Sfide della modernizzazione	29
4.2	Approcci alla modernizzazione	30
4.2.1	White-box	31
4.2.2	Black-box	31
4.2.3	Altre soluzioni	32
4.3	Tecniche di modernizzazione	32
4.3.1	Reingegnerizzazione	32
4.3.2	Considerazioni finali	36
4.4	Risk Managed Modernization	37
4.4.1	Analisi del portfolio	38
4.4.2	Identificazione degli stakeholder	39

4.4.3	Comprensione dei requisiti	39
4.4.4	Creazione del caso di business	40
4.4.5	Comprendere il sistema legacy e la tecnologia software esistente	40
4.4.6	Valutare la tecnologia	40
4.4.7	Definire il target dell'architettura	40
4.4.8	Definire la strategia di modernizzazione	40
4.4.9	Stima delle risorse per la strategia di modernizzazione	41
II	MODERNIZZAZIONE IN PRATICA	43
5	UN APPROCCIO BLACK-BOX: WEBRATIO	45
5.1	Lo strumento: WebRatio	45
5.1.1	La metodologia agile	45
5.1.2	La Model Driven Architecture	46
5.2	Il linguaggio: WebML	48
5.2.1	Obiettivi e vantaggi	49
5.3	Il modello concettuale WebML	49
5.3.1	Il modello dei dati	51
5.3.2	Il modello dell'ipertesto	53
5.3.3	Il modello di presentazione	57
6	APPLICAZIONI LEGACY 3270: SARA	59
6.1	SARA Legacy	59
6.1.1	Struttura dell'applicazione	59
6.1.2	Logica di navigazione	60
6.2	Analisi euristica	62
6.2.1	Problemi riscontrati	62
6.2.2	Risultato dell'analisi	66
6.2.3	Conclusioni	69
7	WEBRATIO E LA W4BMS	71
7.1	La w4bms	71
7.2	Le unit customizzate	72
7.2.1	IdentifyMap Unit	73
7.3	Progetto in pratica	76
7.3.1	Modello dei dati	77
7.3.2	Modello dell'ipertesto	78
7.3.3	Modello di presentazione	79
7.4	Criticità della modernizzazione	80
8	L'APPLICAZIONE MODERNIZZATA: SARA WEB	83
8.1	La nuova interfaccia	83
8.1.1	Login	83
8.1.2	Pagina di ricerca	85
8.1.3	Lista risultati	87
8.1.4	Dettaglio	88
8.2	Miglioramenti	89

8.2.1	Navigazione	89
8.2.2	Aspetto grafico	90
9	ANALISI DI USABILITÀ E ACCESSIBILITÀ DI SARA WEB.	93
9.1	Analisi usabilità	93
9.1.1	Statistiche dell'esecuzione dei test utente	93
9.1.2	Statistiche dei questionari	94
9.2	Analisi dell'accessibilità	95
9.2.1	L'Italia e la legge Stanca	95
9.2.2	Web Content Accessibility Guidelines 2.0	96
9.2.3	SARA Web e accessibilità	98
9.3	Conclusioni	100
10	CONCLUSIONI	101
III	APPENDICE	105
A	ANALISI EURISTICA	107
A.1	Risultati dell'analisi euristica	109
B	ESPERIMENTO CON UTENTI	111
B.O.1	Risultati test usabilità Sara Legacy	114
	BIBLIOGRAPHY	119

LIST OF FIGURES

Figure 1	Classificazione delle azioni di manutenzione	8
Figure 2	Distribuzione per categorie delle modifiche attuate sul sistema.	9
Figure 3	Evoluzione di un sistema secondo Seacord et al.	10
Figure 4	Evoluzione di un software attraverso le fasi di implementazione, mantenimento, modernizzazione e sostituzione.	10
Figure 5	Distribuzione dei costi per attività	14
Figure 6	Tempo speso per attività durante la fase evolutiva del sistema	15
Figure 7	Macchine Mainframe	23
Figure 8	Definizione di ambienti	25
Figure 9	Un esempio di terminale 3270	26
Figure 10	Strategie di reingegnerizzazione	6 30
Figure 11	Revamping[45]	34
Figure 12	Screen scraping sea [7]	35
Figure 13	Diagramma Unified Modeling Language (UML) delle attività	37
Figure 14	Grafo per l'analisi del portfolio	38
Figure 15	Differenti approcci al processo di sviluppo del software	47
Figure 16	Il modello concettuale di WebML	50
Figure 17	Segmento di uno schema dei dati di un applicazione web per la gestione di film	52
Figure 18	Modularizzazione di viste basata su aree e pagine	54
Figure 19	Esempi di link automatico e di trasporto	56
Figure 20	Logica di navigazione dell'applicazione: login e selezione procedura	61
Figure 21	Logica di navigazione dell'applicazione: ricerca, lista e dettaglio	61
Figure 22	Distribuzione problemi usabilità per categoria	64
Figure 23	Distribuzione problemi usabilità per problema	65
Figure 24	Distribuzione problemi usabilità per tipologia e sezione d'intervento	66
Figure 25	Modello di Hutchins, Holland e Norman	67
Figure 26	Screen scraping di applicazione legacy grazie alla w4bms	71
Figure 27	Macroarchitettura della w4bms	72
Figure 28	Notazione grafica delle custom unit della w4bms	72

Figure 29	La struttura dei files di una <i>custom unit</i> all'interno dell'albero di WebRatio	73
Figure 30	Custom unit	76
Figure 31	Entità volatili	77
Figure 32	Modulo WebRatio: selezione della procedure ISMA	78
Figure 33	Modulo WebRatio: procedura di ricerca ISMA	79
Figure 34	Login dell'applicazione legacy	83
Figure 35	Login dell'applicazione modernizzata	85
Figure 36	Sezione di ricerca della procedura INGG	85
Figure 37	Miglioramenti dell'applicazione modernizzata	86
Figure 38	Dettaglio menù tendina	87
Figure 39	Lista risultati	87
Figure 40	Dettaglio	89
Figure 41	Gestione addizionale dell'errore	91
Figure 42	Risultati medi dei test utente	93
Figure 43	Domanda n° 2 del questionario: <i>E' molto difficile muoversi all'interno dell'applicazione-sito</i>	95
Figure 44	Alternativa testuale alle immagini	98
Figure 45	Contestualizzazione dell'informazione	99
Figure 46	Materiale di lavoro	108
Figure 47	Materiale di lavoro	113

LIST OF TABLES

Table 1	Le cinque <i>content units</i> predefinite in WebML	55
Table 2	<i>Operation Units</i> di WebML	57
Table 3	Accessibilità: linee guida WCAG 2.0	97

LISTINGS

Listing 1	Output.template	74
Listing 2	Logic.template	74
Listing 3	BMSIdentifyMapUnitService	75
Listing 4	Codice per l'interpretazione della mappa BMS	77
Listing 5	Script Groovy per lo split della stringa	80

ACRONIMI

CICS	Customer Information Control System
MDA	Model Driven Architecture
WebML	Web Modelling Language
CASE	Computer-Aided Software Engineering
UML	Unified Modeling Language
E/R	Entity/Relationship
XML	eXtensible Markup Language
MVC	Model View Controller
JSP	JavaServer Pages
HTML	HyperText Markup Language
SARA	Soluzione Assegni RA
CPU	Central Processing Unit
BMS	Basic Map Supporting
WAI	Web Accessibility Initiative
W3C	World Wide Web Consortium
WCAG	Web Content Accessibility Guidelines
GUI	Grafic User Interface
UI	User Interface
TUI	Text User Interface
COTS	Commercial Of The Shelf
ASP	Application Service Provider
ERP	Enterprise Resource Planning
SSH	Secure SHell
MIPS	Million Instructions Per Second

MVS	Multiple Virtual Storage
TP	Transaction Processing
API	Application Programming Interface
URL	Uniform Resource Locator
RMM	Risk Managed Modernization
QoS	Quality of Services
PIM	Platform Independent Model
PSM	Platform Specific Model
MDD	Model Driven Development
MDA	Model Driven Architecture
OMG	Object Management Group
JSTL	Java Standard Tag Library

Part I

MODERNIZZAZIONE DI SISTEMI LEGACY

INTRODUZIONE

Chiunque sia vissuto intorno alla fine degli anni '90 ricorderà la grande paura che aleggiava attorno alla presunta catastrofe del millennium bug. Il problema nasceva dal fatto che in molti sistemi informatici l'anno veniva rappresentato con due sole cifre e quindi con il passaggio al nuovo millennio i calcolatori avrebbero pensato di essere nel 1900 a causa della contrazione in 00, portando a conseguenze imprevedibili.

Le radici di questo problema affondano negli anni 60 quando uno dei maggiori problemi nell'ambito IT era quello della scarsa capacità di memorizzazione dei dati. Uno dei metodi più utilizzati per risparmiare memoria era quello di accorciare la lunghezza dei campi delle date nei programmi: ciò veniva fatto appunto mediante la contrazione dell'anno e permetteva di risparmiare 2 byte per occorrenza. Di per sé un tale risparmio potrebbe sembrare insignificante ma bisogna pensare che i campi con le date erano molto frequenti e all'interno della stessa applicazione potevano ricorrere centinaia se non addirittura migliaia di volte. Si pensi ad esempio ad un file che contiene la lista di impiegati di una grande azienda e supponiamo che per ogni impiegato si voglia memorizzare l'informazione circa la data di nascita, di assunzione, quella del previsto pensionamento e di eventuali promozioni. O ancora si pensi all'industria delle automobili e a quella aerospaziale dove è necessario tenere traccia delle milioni di parti associate ad un particolare modello di macchina o di aeroplano: ogni modello ha dei campi che riguardano la data di costruzione, di installazione, di durata, di sostituzione e molti altri ancora. Si capisce quindi perché una riduzione di soli 2 byte per data significhi nella realtà pratica un risparmio ben più grande.

Durante quegli anni i programmatori non si preoccupavano delle conseguenze dovute all'utilizzo di tale sintassi poiché di certo non si aspettavano che i loro programmi sarebbero stati ancora in esecuzione 20, 30 o addirittura 40 anni dopo. Negli anni '60, quando questi programmi venivano sviluppati, l'industria dell'IT era ancora agli albori e da allora il mondo dell'informatica è rapidamente mutato: sono nati nuovi metodi di sviluppo del software come la programmazione strutturata, i database relazionali, i linguaggi di quarta generazione e la programmazione a oggetti. Si pensava che attraverso questi nuovi strumenti si sarebbero potuti implementare nuovi programmi, più efficienti e funzionali, che sarebbero andati a sostituire quelli vecchi. Quello che non si era ancora realizzato era che le applicazioni sarebbero cresciute così tanto in complessità, grandezza e importanza da renderne complicata se non impossibile la sostituzione. Il

risultato di ciò fu che un gran numero dei programmi sviluppati negli anni 60 era ancora in esecuzione alla fine degli anni 90.

Tutto ciò rappresentava il fulcro del problema quando, con l'arrivo del nuovo millennio, l'anno 2000 sarebbe stato interpretato dalla maggior parte dei sistemi legacy in maniera erranea, causando black out o altri risultati imprevedibili. Fu attuato quindi un massivo sforzo di correzione. Alcune industrie come banche e compagnie aeree spesero diversi anni per correggere il loro software e per garantire il corretto funzionamento dei sistemi entro il 1 gennaio del nuovo anno. Si stima che i costi per la gestione del millennium bug, tra prevenzione e correzione postuma, furono molto alti e si aggirarono intorno ai 300 miliardi di dollari.

Ci furono comunque anche dei lati positivi. In questo periodo, ad esempio, molte aziende furono costrette a stilare un inventario delle applicazioni su cui dovevano essere attuate le correzioni: durante questa fase si accorsero che molte di queste non supportavano più le attività di business e addirittura non venivano più utilizzate e fu quindi possibile eliminarle dagli asset aziendali permettendo una notevole riduzione dei costi.

1.1 IL PROBLEMA

Il caso illustrato non rappresenta un'eccezione ma è altamente esemplificativo della situazione in cui si trovano oggi la maggior parte delle aziende: sono costantemente spinte da diversi fattori ad intervenire sui propri sistemi che però sono diventati negli anni complessi, difficili da mantenere e impossibili da modificare. Nella maggior parte dei casi anche la sostituzione è diventata critica poiché tali sistemi racchiudono il *core business* dell'impresa, cioè la conoscenza e l'esperienza accumulata nel corso degli anni, e una sostituzione rischierebbe di comprometterne la conservazione.

E' proprio a causa di questi motivi che negli anni sono stati introdotti diversi approcci di modernizzazione, che servono a supportare il cambiamento di questi grandi sistemi laddove per diverse ragioni non è possibile una sostituzione e la semplice manutenzione non è più sufficiente.

1.2 LE DOMANDE DI RICERCA

Scopo di questa tesi è illustrare in quale modo oggi sia possibile risolvere i problemi legati ai sistemi legacy. Le domande a cui si cercherà di trovare una risposta saranno quindi:

- quand'è che un sistema non può più essere mantenuto e deve essere modernizzato o sostituito?
- quali sono i tipi di modernizzazione disponibili e come si può scegliere quello corretto per un dato sistema?

- quali sono gli strumenti a disposizione per la modernizzazione?
- come è possibile valutare se la modernizzazione ha raggiunto il suo scopo, e il sistema è stato effettivamente rinnovato in termini di efficienza?

1.3 ORGANIZZAZIONE DEI CONTENUTI

La prima parte della tesi si occuperà di illustrare il ciclo di vita dei prodotti software e dei sistemi, con particolare attenzione alle fasi che seguono il rilascio degli stessi sul mercato. Si analizzeranno nel dettaglio le situazioni in cui si rende necessario apportare delle modifiche, catalogandole a seconda dell'invasività come parti dei processi di manutenzione, modernizzazione e sostituzione. Si spiegherà quindi cosa caratterizza un sistema legacy, perché diventa tale e in che modo la modernizzazione ne risolve i problemi.

Nella parte pratica si illustrerà il funzionamento di uno strumento capace di implementare il revamping, un particolare tipo di modernizzazione black-box in cui si agisce solo sull'interfaccia del sistema. Infine si mostreranno i risultati del lavoro attraverso l'analisi dell'usabilità e dell'accessibilità, che metteranno in risalto i miglioramenti dell'applicazione web modernizzata.

Di seguito si riporta la suddivisione degli argomenti per capitolo.

PRIMO CAPITOLO si spiegheranno le dinamiche evolutive del software dal momento in cui viene rilasciato per la prima volta. Si spiegheranno quindi quali sono i motivi per cui si interviene sulle applicazioni e quali sono i tipi di modifiche possibili nel contesto dell'ingegneria del software. In particolare si vedranno i processi di manutenzione, di modernizzazione e di sostituzione nel contesto dell'attuale letteratura.

SECONDO CAPITOLO si descriveranno i sistemi legacy: si spiegherà perché sono così importanti, perché è così difficile mantenerli e se ne delineerà in breve la struttura.

TERZO CAPITOLO si spiegherà in modo approfondito cos'è la modernizzazione, quali sono le diverse tecniche con cui è possibile attuarla, come e quando può essere applicata ai sistemi legacy e in che modo ne risolve i problemi. Si metteranno quindi in evidenza pregi e difetti delle diverse soluzioni.

QUARTO CAPITOLO descriverà un particolare strumento di modernizzazione di tipo black-box, WebRatio, con particolare enfasi sul linguaggio di modellazione su cui esso è basato: WebML.

QUINTO CAPITOLO presenterà una famiglia di applicazioni di tipo legacy, quelle Customer Information Control System (CICS) a interfaccia testuale accessibili mediante terminali 3270. Su un esempio di tale applicazione, Soluzione Assegni RA (SARA) sarà effettuata l'analisi euristica allo scopo di analizzare le problematiche in termini di usabilità che rendono necessaria la modernizzazione.

SESTO CAPITOLO si introduce la libreria w4bms, necessaria alla customizzazione di WebRatio per la realizzazione effettiva della modernizzazione. Si mostrerà nel dettaglio cosa significa, in pratica, modernizzare con WebRatio.

SETTIMO CAPITOLO mostrerà i risultati ottenuti in termini di interfaccia grafica con la modernizzazione. Si metteranno in evidenza i miglioramenti sia in termini di logica di navigazione che di layout.

OTTAVO CAPITOLO attraverso il confronto in termini di usabilità tra la nuova e la vecchia applicazione si mostreranno i risultati ottenuti mediante la modernizzazione. Inoltre si mostrerà come la nuova applicazione web risponda anche ai requisiti di accessibilità previsti dalle Web Content Accessibility Guidelines (WCAG).

NONO CAPITOLO sarà dedicato alle considerazioni conclusive.

Agli albori dell'informatica il problema principale della programmazione era come mettere insieme una sequenza di istruzioni per fare in modo che il calcolatore producesse qualcosa di utile. Con la diminuzione dei prezzi dei computer e la loro diffusione, un numero sempre maggiore di persone iniziò a utilizzarlo; nei tardi anni Cinquanta furono inventati i linguaggi ad alto livello per rendere più facile comunicare con le macchine, ma la programmazione restava ancora un compito individuale. Pochi anni dopo programmare divenne una professione: una persona poteva chiedere al programmatore di scrivere un programma invece di realizzarlo per conto proprio. Questo introdusse per la prima volta una separazione tra utente e computer: l'utente specificava cosa voleva ottenere da una data applicazione utilizzando un linguaggio diverso dalla notazione di programmazione; il programmatore leggeva quindi tale specifica e la traduceva in un insieme ben preciso di istruzioni macchina. A partire dagli anni Sessanta ci furono i primi tentativi di creare grandi sistemi complessi; in questo periodo si comprese che i problemi riscontrati nella creazione di grandi sistemi software non riguardavano la pura e semplice capacità di programmazione; spesso erano i problemi che dovevano essere risolti a non essere ben compresi. Le persone che lavoravano sul progetto infatti spendevano molto più tempo per comunicare tra di loro piuttosto che per scrivere codice. In questo periodo si consolidò l'idea che il problema della costruzione di un software dovessero essere affrontato nello stesso modo adottato dagli ingegneri per costruire sistemi grandi e complessi; lo scopo era vedere il software finale come un prodotto complesso e la sua creazione come un lavoro ingegneristico [47]. Tale approccio richiedeva management, organizzazione, strumenti, teorie e metodologie tecniche.

Nacque quindi l'idea del ciclo di vita del software: si dice che il software ha un ciclo di vita composto da varie fasi, a ciascuna delle quali è associato lo sviluppo di una parte del sistema ¹. L'obiettivo di una divisione simile è di permettere di definire degli stadi intermedi che permettano la validazione dello sviluppo, cioè la conformità del software secondo i bisogni espressi, e la verifica del processo di sviluppo, cioè l'adeguatezza dei modelli attuati. Il ciclo di vita permette di rilevare gli errori il prima possibile e quindi di controllare la qualità del software, dei tempi di realizzazione e i costi associati.

¹ Il ciclo di vita del software comprende alcune fasi fondamentali, la cui esecuzione cambia a seconda del modello di riferimento: analisi, progettazione, implementazione, collaudo, rilascio e manutenzione.

La fase di evoluzione deriva dalla consapevolezza che anche dopo il rilascio in produzione i sistemi devono essere costantemente modificati. I motivi sono molteplici: cambiamenti nella logica di business potrebbero generare nuovi requisiti, alcune parti del software potrebbero richiedere la correzione degli errori scoperti durante la fase operativa o infine potrebbe essere necessario l'adattamento verso una nuova piattaforma per migliorare le prestazioni o altre caratteristiche non funzionali. Lo sviluppo del software, dunque, non si ferma all'atto della consegna, ma continua durante tutta la vita del sistema [47].

2.1 MANUTENZIONE DEL SOFTWARE SECONDO LO STANDARD ISO/IEC 14764

Secondo la definizione ripresa dallo standard ISO/IEC 14764 [8] la manutenzione è quel processo che include tutte le modifiche effettuate sul sistema che prendono atto dopo la consegna. Lo scopo di tali modifiche è migliorare alcuni attributi qualitativi, come performance e affidabilità, oppure adattare il sistema a nuovi ambienti. Le ragioni che possono far nascere la necessità di intervenire sul sistema sono molte e possono essere divise in correzioni o miglioramenti come mostrato in figura.

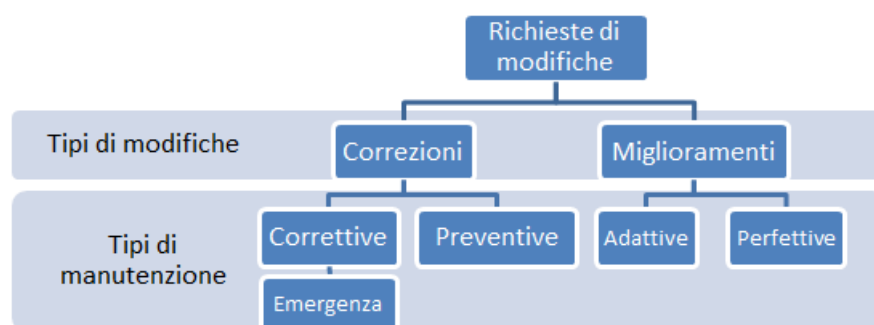


Figure 1: Classificazione delle azioni di manutenzione

Una correzione è connessa alla scoperta di errori o potenziali *bug*, quindi il suo scopo è quello di permettere un corretto funzionamento del sistema. Il miglioramento invece nasce dall'esigenza di estensione, implementando ad esempio nuova funzionalità, al fine di stare al passo con i crescenti bisogni di business.

2.1.1 Tipi di manutenzione

La manutenzione può essere quindi classificata in base al tipo di modifica che si rende necessaria come [44]:

- **correttiva**: si riferisce alle modifiche che sono effettuate per correggere gli errori trovati dopo l'installazione del sistema o quando non sono stati soddisfatti tutti i requisiti stabiliti in fase di analisi;
- **di emergenza**: questo tipo di manutenzione è generalmente collegata ai cambiamenti correttivi. Consiste in una correzione immediata, effettuata subito dopo la scoperta di un errore, in modo da permettere che il sistema resti operativo;
- **preventiva**: si riferisce alle modifiche introdotte per prevenire potenziali problemi nel futuro. Un esempio di azione preventiva consiste nell'aumento della manutenibilità e dell'affidabilità;
- **adattiva**: riguarda le modifiche che sono causate da un cambio dei requisiti del sistema. Per esempio la necessità di utilizzare differenti piattaforme hardware o il bisogno di integrazione con altre applicazioni;
- **perfettiva**: sono le modifiche introdotte dopo la consegna al cliente. Servono per trovare i difetti nascosti nel sistema, prima che possano diventare veri e propri problemi. Includono inoltre le modifiche che migliorano la documentazione del software o attributi come performance e usabilità.

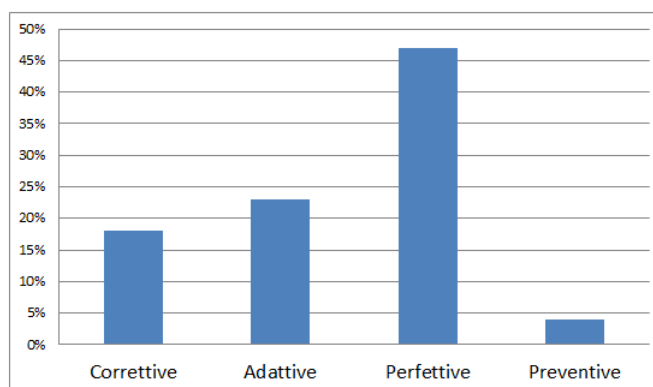


Figure 2: Distribuzione per categorie delle modifiche attuate sul sistema.

Le modifiche che sono parte della manutenzione correttiva e preventiva sono classificate come correzioni, perché il loro ruolo è prevenire bug o risolvere quelli già esistenti. Le modifiche di emergenza fanno parte della manutenzione correttiva perché tali richieste nella maggior parte dei casi sono causate da errori critici che devono essere risolti nell'immediato o il sistema potrebbe non funzionare correttamente. Le azioni attuate all'interno della manutenzione adattiva e perfettiva invece sono classificate come miglioramenti, perché influenzano la crescita e lo sviluppo del sistema.

2.2 EVOLUZIONE DI UNA SISTEMA SECONDO SEACORD ED AL.

Un'altra definizione di manutenzione arriva dal libro *"Modernizing Legacy System: Software Technologies, Engineering Processes, and Business Practices"* nel quale essa è vista come uno stadio del processo evolutivo del sistema [46]. Secondo gli autori l'evoluzione può essere divisa in tre differenti fasi: manutenzione, modernizzazione e sostituzione. Una dettagliata tassonomia è mostrata in figura.

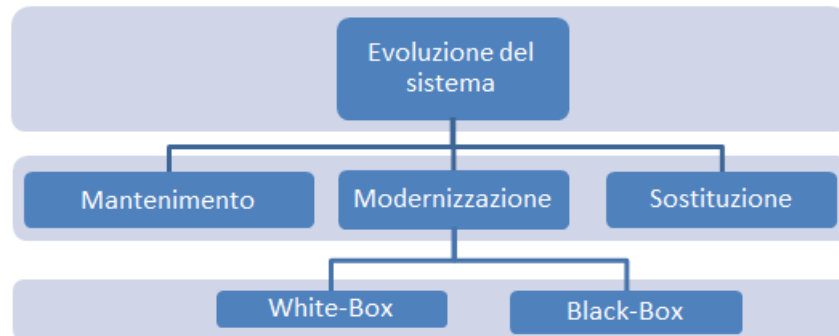


Figure 3: Evoluzione di un sistema secondo Seacord et al.

Le attività evolutive che prendono atto durante il ciclo di vita del software sono molteplici e rispecchiano generalmente la necessità di tenere il sistema aggiornato rispetto alle necessità di business.

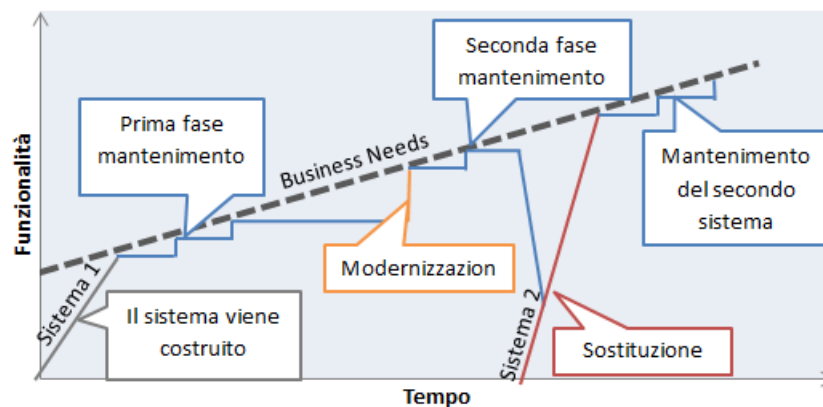


Figure 4: Evoluzione di un software attraverso le fasi di implementazione, mantenimento, modernizzazione e sostituzione.

La linea tratteggiata nella figura rappresenta i crescenti bisogni di business mentre la linea continua rappresenta le funzionalità offerte dal sistema. Ripetute manutenzioni del sistema supportano i bisogni di business

per un certo periodo di tempo, ma man mano che il sistema diventa sempre più datato la manutenzione non riesce più a stare al passo con le nuove richieste di business. A questo punto è richiesto un maggiore sforzo sia in termini di tempo che di funzionalità rispetto a quelli necessari per la semplice manutenzione. In questo caso sono possibili due scelte: se il sistema non è diventato completamente ingestibile e obsoleto può essere rinnovato attraverso una modernizzazione mentre se non riesce più a soddisfare gli obiettivi di business deve essere sostituito [21]. Determinare la categoria dell'attività evolutiva che è più adatta al sistema in un certo momento del ciclo di vita non è semplice. Le organizzazioni che hanno un budget limitato per la manutenzione e per l'aggiornamento dei sistemi ereditati, devono decidere come ottenere il miglior profitto dai loro investimenti. Per prendere la decisione corretta è necessario conoscere bene il sistema e analizzare le conseguenze di ogni azione al fine di identificare la strategia più adatta per farlo evolvere [54].

2.2.1 *Manutenzione*

La manutenzione è un processo incrementale e iterativo durante il quale il sistema viene modificato dopo la consegna [46]. Le modifiche fatte al software possono essere relativamente semplici, come i cambiamenti atti a correggere errori di programmazione o di progettazione, o molto più complesse; è il caso delle modifiche necessarie a correggere errori di specifica o per adattare il sistema a nuovi requisiti. I cambiamenti sono implementati agendo sui componenti esistenti del sistema e, dove necessario, aggiungendone di nuovi. La manutenzione richiesta per supportare l'evoluzione del sistema ha delle limitazioni [21]: il vantaggio competitivo derivato dall'adozione di nuove tecnologie è seriamente limitato: il net-centric computing² o l'intervento sulle Grafic User Interface (GUI), ad esempio, non sono considerate come operazioni di manutenzione; il costo per la manutenzione del software cresce col tempo; trovare esperti di tecnologie datate diventa sempre più difficile e costoso; inoltre spesso l'impatto complessivo è più grande della somma delle piccole modifiche a causa dell'erosione dell'integrità concettuale del sistema. Questa opzione dovrebbe essere scelta quando il sistema è ancora necessario all'azienda, è sufficientemente stabile e gli utenti richiedono un numero relativamente basso modifiche.

² Il network centric computing è un modello di riferimento per il futuro e fa riferimento ad un'architettura dove gli attuali pc, particolarmente appesantiti da pacchetti software spesso inutilizzati, vengano sostituiti da macchine intelligenti con una dotazione di software minimale e una grande capacità di connessione alla Rete. Tale modello che vede il suo sviluppo embrionale nelle applicazioni Application Service Provider (ASP) permetterebbe a privati e aziende di ridurre gli investimenti in software[18].

2.2.2 Modernizzazione

La modernizzazione è un'alternativa alla sostituzione che permette di minimizzare il rischio. A seconda dell'approccio con cui si pone rispetto all'analisi e alla conoscenza del sistema viene generalmente catalogata come *black-box* o *white-box*. Nel primo caso si opera una modernizzazione senza indagare il funzionamento interno del sistema, ma se ne considera solo il comportamento, cioè l'interfaccia esterna. Viceversa nel secondo caso è prevista un'approfondita fase di analisi al fine di comprendere le dinamiche interne del sistema. La modernizzazione è meno costosa rispetto alla sostituzione, permette la conservazione della conoscenza di business e la facile reintegrazione del personale permettendo comunque l'adeguamento rispetto ai nuovi requisiti migliorando quindi l'agilità complessiva del sistema. Per contro pone limiti maggiori dal momento che, per quanto ci si sforzi di migliorare e risolvere le problematiche esistenti, restano i vincoli dovuti alle precedenti implementazioni. La modernizzazione non risolve infatti il problema principale legato all'evoluzione del software, quello della crescente complessità.

2.2.3 Sostituzione

La sostituzione è appropriata per quei sistemi legacy che non riescono a tenere il passo con i bisogni di business e per i quali la modernizzazione non è possibile o fattibile rispetto ai costi; tale approccio deve comunque essere scelto con cura poiché porta con sé un alto numero di rischi. La sostituzione è generalmente usata per i sistemi che sono poco documentati, datati o non estensibili. Esistono due tipi di approcci: quello *big-bang* [24] prevede che il sistema sia installato tutto in una volta e quello incrementale prevede che il sistema sia installato modulo per modulo. La sostituzione incrementale può prendere la forma di una serie di modernizzazioni di tipo *white-box*. La sostituzione implica la riscrittura del sistema da zero e un utilizzo di energie molto intensivo. Inoltre le risorse IT allocate per la manutenzione potrebbero non essere familiari con le tecnologie moderne utilizzate per la costruzione dei nuovi sistemi. Infine la sostituzione richiede fasi di testing molto lunghe e accurate. Infatti i sistemi legacy sono ben testati, incapsulano l'esperienza di business e non ci sono garanzie che il nuovo sistema sia funzionante e robusto come quello vecchio [46].

2.3 LE LEGGI DI LEHMAN

La maggior parte del lavoro svolto nello studio dell'evoluzione del software è stata fatta da Lehman e Belady che proposero una serie di leggi. [46].

Secondo la prima legge la manutenzione di un sistema è un processo inevitabile: quando l'ambiente del sistema cambia, emergono nuovi requisiti e il sistema deve essere modificato; l'introduzione del sistema modificato nell'ambiente causa una serie di cambiamenti nell'ambiente stesso e quindi il processo evolutivo si ripete.

La seconda legge sostiene che, quando il sistema viene modificato, la sua struttura si deteriora; l'unico modo per evitare che questo accada è investire nella manutenzione preventiva per migliorare la struttura del software senza aggiungervi funzionalità; ovviamente questo comporta costi aggiuntivi, oltre a quelli per l'implementazione delle modifiche necessarie al sistema.

La terza legge sostiene che i grandi sistemi hanno una loro dinamica, stabilita nei primi stadi del processo di sviluppo, che determina le tendenze complessive del processo di manutenzione e limita il numero delle possibili modifiche. Una volta che un sistema oltrepassa una determinata dimensione minima, diventa più difficile modificarlo: a causa della sua grandezza e della sua complessità, il esso risulta difficile da comprendere ed è più probabile che i programmatori sbaglino e vi introducano errori. Pertanto fare piccoli cambiamenti permette di mantenere l'affidabilità del sistema, viceversa un grande cambiamento introdurrà probabilmente molti nuovi errori che limiteranno le modifiche utili che sarà possibile implementare nelle nuove versioni del sistema.

Secondo la quarta legge nella maggior parte dei grandi progetti di programmazione si lavora in quello che viene definito come *stato saturo*, ovvero un cambiamento alle risorse dello staff ha effetti impercettibili sull'evoluzione a lungo termine del sistema.

La quinta legge riguarda le modifiche incrementali legate alle *release* del sistema. Aggiungere nuove funzionalità introduce inevitabilmente nuovi errori di conseguenza maggiori saranno le funzionalità aggiunte ad ogni release maggiori saranno anche gli errori introdotti. E' quindi necessario tenere in considerazione alcuni aspetti: da un lato è sconsigliato introdurre un gran numero di funzionalità in una versione successiva; dall'altro, quando ciò è strettamente necessario, è consigliato far seguire il rilascio da una serie di aggiornamenti il cui unico scopo non è un ulteriore aumento delle funzionalità bensì la correzione degli errori. Infine è necessario che l'azienda tenga conto del costo dovuto alla correzione di tali errori nel momento in cui stanziava il budget.

La sesta e la settima legge sono simili e dicono essenzialmente che gli utenti del software diventano sempre più insoddisfatti se non c'è manutenzione e se non si aggiungono nuove funzionalità.

Le osservazioni di Lehman dovrebbero essere sempre prese in considerazione quando si pianificano i processi di manutenzione. Potrebbe comunque succedere che motivi economici o aziendali nasca l'esigenza di ignorarle: per ragioni di marketing, ad esempio, può essere necessario fare

diverse modifiche al sistema in una singola release. Ogni caso va valutato e fondo ed è fondamentale essere consapevoli di quali siano le conseguenze derivate dall'adozione di ogni possibile strategia [15].

2.4 I COSTI DELL'EVOLUZIONE

Gli investimenti aziendali riguardano in larga parte la gestione di software esistente piuttosto che lo sviluppo del nuovo: studi statistici dimostrano che il costo dell'evoluzione di un progetto copre dal 50% fino all'80% del costo complessivo [39]. Da notare che in generale gli interventi dipendono più dalla nascita di nuovi requisiti che dalla riparazione dei guasti. Un'importante ragione per cui i costi di manutenzione sono alti è che aggiungere funzionalità quando un sistema è operativo è più costoso che implementare la stessa funzionalità durante lo sviluppo.

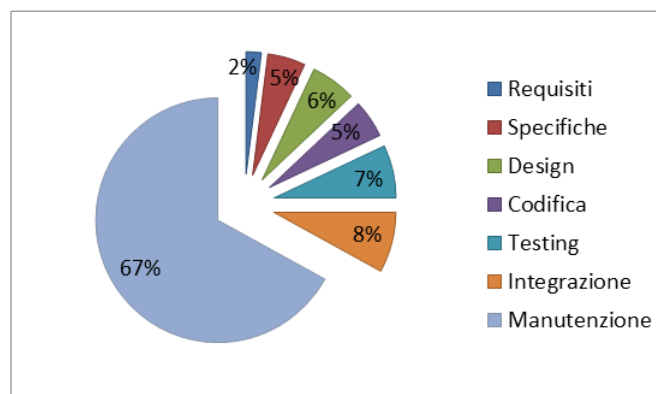


Figure 5: Distribuzione dei costi per attività

Esistono alcuni fattori chiave che distinguono lo sviluppo dalla manutenzione aumentandone i costi [47]:

- **stabilità del team:** dopo la consegna di un sistema è normale sciogliere il team di sviluppo per permettere alle persone di lavorare su nuovi progetti. Il nuovo team o i singoli responsabili della manutenzione possono non comprendere il sistema o il background delle decisioni di progettazione. Buona parte dello sforzo durante il processo di manutenzione avviene ancor prima di potervi implementare le modifiche e riguarda la comprensione del sistema;
- **analisi e documentazione:** il codice sorgente e i commenti sono gli artefatti più importanti per la comprensione del sistema. Come già detto uno dei problemi maggiori quando si ha a che fare con sistemi datati riguarda la loro comprensione strutturale: è chiaro che questo compito è reso ancora più difficile dall'assenza di una docu-

mentazione affidabile e di qualità. Nel grafico è mostrato il tempo dedicato all'analisi durante la fase evolutiva del sistema.

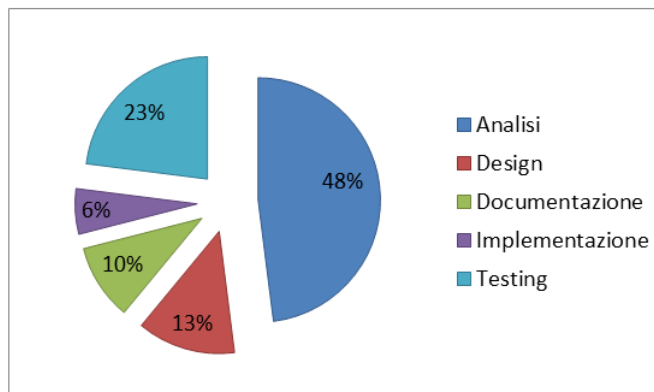


Figure 6: Tempo speso per attività durante la fase evolutiva del sistema

- responsabilità contrattuali: il contratto per la manutenzione del sistema di solito è separato da quello di sviluppo perché può essere stipulato con un'altra compagnia piuttosto che con lo sviluppatore iniziale del sistema; questo fattore, insieme alla mancanza di stabilità nel team, determina l'assenza di incentivi a scrivere il software in modo che sia poi facile da mantenere: se i programmatori possono prendersela comoda al fine di ridurre gli sforzi necessari all'implementazione è quasi certo che lo faranno anche se questo significa aumentare i costi legati alle modifiche successive;
- capacità dello staff: lo staff di manutenzione spesso è relativamente inesperto e ha poca familiarità con il dominio di applicazione; inoltre i vecchi sistemi sono tipicamente scritti in linguaggi di programmazione obsoleti di cui le risorse possono non avere esperienza e quindi deve essere speso del tempo per lo studio e per la comprensione dell'infrastruttura. Infine la manutenzione è poco considerata dagli ingegneri del software: sembra che sia un processo meno impegnativo rispetto alla fase di sviluppo e viene spesso assegnato allo staff più giovane;
- età e struttura del programma: come visto nella seconda legge di Lehman, mentre i programmi invecchiano, la loro struttura tende a deteriorarsi, rendendoli più difficili da comprendere e modificare. Alcuni sistemi sono stati sviluppati senza le moderne tecniche di ingegneria del software, spesso sono mal strutturati e forse sono stati ottimizzati più per l'efficienza che per la comprensibilità; la documentazione può essere andata persa o essere inconsistente; infine i vecchi sistemi possono non essere stati soggetti alla gestione della

configurazione, quindi spesso si perde tempo cercando le versioni giuste delle componenti che è necessario modificare.

I primi quattro problemi derivano dal fatto che molte organizzazioni considerano lo sviluppo e la manutenzione attività separate: la manutenzione è vista come attività di seconda classe e non c'è incentivo a investire soldi durante lo sviluppo per ridurre i costi delle modifiche successive. L'unica soluzione a lungo termine consiste nel prendere consapevolezza del fatto che i sistemi non hanno generalmente una vita definita ma continuano ad evolvere in modo continuo nel tempo, per cui una maggiore attenzione va rivolta alla progettazione preventiva.

2.5 CONTESTO ATTUALE

Il processo di evoluzione del software e in particolare quello di modernizzazione è contestualizzabile oggi in tre diversi ambiti:

- grandi imprese: sono le grandi aziende, come banche, istituzioni e pubbliche amministrazioni, che fanno affidamento su un gran numero di sistemi legacy per la propria sopravvivenza. In questo contesto predominano i *mainframe* tipicamente utilizzati per applicazioni critiche per l'elaborazione ad alte prestazioni e affidabilità di grandi moli di dati: ne sono un esempio quelle in gioco nelle transazioni finanziarie, nei censimenti, nelle statistiche delle industrie, nelle applicazioni Enterprise Resource Planning (ERP) e nei sistemi di stampa delle banconote.
- medie imprese: anch'esse fanno un'uso intensivo di infrastrutture datate per gestire quei servizi che sono cruciali per l'azienda. In questo scenario predominano i *minicomputer*: si tratta di computer più piccoli ed economicamente abbordabili rispetto ai mainframe. Questi sistemi, denominati *midrange systems*, vengono utilizzati specialmente nell'ambito gestionale per l'elaborazione di dati aziendali o con la funzione di server. Sistemi di questo tipo sono infatti ampiamente utilizzati come sistemi dipartimentali o per aziende medie o piccole. Tra i più diffusi e conosciuti si ricordano gli IBM *as/400* oggi conosciuti come *iSeries*.
- applicazioni ad uso privato: sono quelle applicazioni caratterizzate da un'interfaccia testuale quali *PuTTY*³.

Nei primi due ambiti la modernizzazione oggi ha un ruolo cruciale. Sono molte le aziende che spinte dalla necessità di incrementare i propri guadagni

³ PuTTY è un client Secure SHell (SSH), Telnet ed rlogin combinato con un emulatore di terminale

a parità di spese vedono nella modernizzazione una strategia per ottimizzare le proprie infrastrutture e migliorare l'agilità al fine di rispondere meglio alle esigenze di business in costante cambiamento. Per quanto riguarda il contesto privato invece la modernizzazione non trova una grande diffusione. Questo è dovuto al fatto che, a differenza delle grandi e medie imprese, i vantaggi derivanti da una modernizzazione sarebbero minimi rispetto allo sforzo richiesto per la sua attuazione. Per questo motivo nei prossimi capitoli si darà particolare attenzione solo alla modernizzazione di sistemi legacy tipici delle grandi imprese.

Al giorno d'oggi esiste un enorme numero di sistemi legacy sparsi per il mondo ed esistono altrettante definizioni di cosa un sistema legacy sia. Tutte le definizioni sono comunque d'accordo nella definizione di *legacy* come una sorta di ereditarietà che porta con se un certo valore ma anche l'idea di vecchio e obsoleto. In questo contesto la definizione usata da Tromp e Hoffman fornisce un buon punto di partenza: *"un sistema legacy è un sistema che è stato disegnato, implementato e installato in un ambiente radicalmente differente da quello imposto dalla corrente strategia It"* [52]. Per contestualizzare meglio la definizione bisognerebbe evidenziare la prima logica conseguenza e cioè che i sistemi legacy non riescono più a supportare la corrente strategia di business. Un sistema legacy è comunque qualcosa di più ampio e con questo termine ci si può riferire ad una qualsiasi applicazione software esistente che continua a fornire servizi vitali ad un'azienda a dispetto dell'età e delle tecnologie hardware usate [9]. Sono un esempio i mainframe, le applicazioni in esecuzione su di essi e il linguaggio di programmazione in cui sono state scritte. Da notare che il termine legacy non ha necessariamente a che fare con la dimensione o perfino con l'età del sistema; si potrebbe anche pensare che ogni applicazioni sviluppata con tecnologie non correnti sia legacy per enfatizzare il fatto che non ci si deve limitare a pensare ai sistemi legacy come a *"dinosauri in via d'estinzione"* Lucca [28]. Qualcuno addirittura dice che *"il codice scritto ieri, oggi è codice legacy"* [46].

3.1 PERSISTENZA DEI SISTEMI LEGACY

Esistono diversi motivi per cui è necessario tenere in vista questi sistemi:

- il costo per il redesign può essere proibitivo a causa delle dimensioni, della complessità o perché si ha a che fare con un sistema monolitico e modernizzarlo significherebbe andare a rivoluzionare anche i processi di business dell'organizzazione;
- il sistema fornisce ancora funzionalità *core* o di importanza critica che non possono essere estrapolate dai servizi e il costo per il design di un nuovo sistema dello stesso livello è troppo alto; nel corso degli anni l'organizzazione ha investito sia economicamente che a livello di capitale intellettuale e non può limitarsi a dismettere i sistemi così costruiti; il sistema è considerato come un deposito delle funzionalità di business dell'organizzazione che non sono esplicitamente doc-

umentate altrove [41]. Come dice Bisbal *“sostituire un sistema legacy influenza non solo il costo dello sviluppo del software ma anche la spesa necessaria a riscoprire tutta la conoscenza circa le regole e i processi di business incorporati nel sistema”*[12];

- il funzionamento del sistema non è più ben conosciuto perché i progettisti originari non sono più nell'organizzazione e la documentazione è di scarsa qualità o è addirittura andata persa; il sistema lavora bene e il proprietario non ha sufficienti ragioni per cambiarlo. In altre parole, creare e reimparare un nuovo sistema avrebbe costi proibitivi in tempo e denaro.

3.1.1 Criticità

I motivi per cui i sistemi legacy sono ancora diffusi sul mercato sono gli stessi per cui sono così difficili da dismettere. Spesso vengono eseguiti su hardware obsoleti e lenti le cui componenti sono sempre più difficili da trovare [10]. Questi sistemi sono anche difficili da mantenere e migliorare perché c'è una mancanza di conoscenza interna del sistema: la documentazione è inadeguata, i manuali sono andati perduti negli anni, creando una situazione in cui non è rimasto nessuno con la conoscenza necessaria per poter gestire la manutenzione e le eventuali azioni evolutive. Esistono diversi motivi per cui la documentazione diventa inadeguata: ad esempio in molti casi, quando il codice sorgente è modificato per correggere un bug o aggiungere un miglioramento, le specifiche e la scrittura dei documenti non sono aggiornati. Altre volte la documentazione viene cambiata per riflettere nuove funzionalità richieste prima ancora che tali funzionalità siano effettivamente aggiunte e magari non verranno mai effettivamente implementate. Il risultato è che gli sviluppatori imparano a non fidarsi e a non usare niente altro che il codice sorgente, rendendo il software meno affidabile e molto più difficile da capire ed evolvere[40].

Quando si perde l'equilibrio tra dimensione tecnica e di business, il termine legacy può essere visto come la distanza tra i bisogni di business e le capacità tecniche [31]. Una volta che si fa coincidere il concetto di legacy con questo gap si capisce perché, come detto precedentemente, anche i nuovi sistemi possano facilmente essere considerati datati: sono tali semplicemente perché non sono più capaci di rispondere alle necessità di business. E' chiaro che il software legacy è un asset fondamentale per le compagnie che lo posseggono, sia per gli investimenti spesi su di essi che per la conoscenza che essi racchiudono. Inoltre, nonostante la loro complessità e l'evidente difficoltà evolutiva e forse proprio a causa di questi fattori, tali sistemi sono difficili da imitare: proprio per questo motivo possono essere visti come una fonte di vantaggio competitivo.

Le applicazioni legacy odierne, come detto prima, sono il risultato degli investimenti di capitali passati. Il valore degli investimenti tende però a diminuire nel tempo man mano che il contesto di business e tecnologico cambiano. Questo può accadere, ad esempio, quando l'infrastruttura alla base è sostituita, è richiesto l'accesso web o il peso dei cambiamenti nell'applicazione e la mancanza di *know-how* disponibile rendono impossibile continuare il miglioramento. Man mano che gli investimenti sull'applicazione tendono a diminuire, il costo del mantenimento cresce, addirittura fino a superare il costo di una eventuale sostituzione a meno di una qualche emulazione o compatibilità retroattiva che permettono al vecchio software di essere eseguito su nuovo hardware [38]. Esempi tipici sono i sistemi che gestiscono gli account bancari o il traffico aereo: sono spesso scritti in linguaggi vecchi come il Cobol, fanno affidamento su infrastrutture realizzate ad-hoc e le organizzazioni non possono sostituirli senza incontrare enormi problemi di sicurezza. In sostanza le organizzazioni non possono permettersi di dismetterli e allo stesso tempo non possono permettersi l'aggiornamento.

Decidere di modernizzare le applicazioni legacy è una decisione che porta sia rischi che vantaggi. Da alcuni punti vista sembrerebbe più facile fare affidamento su quello che appare stabile e sperare che sia adeguato a supportare il business attuale, almeno nel breve termine. Il problema è che spesso queste applicazioni costituiscono il *core business* dell'azienda ed aspettare troppo tempo può diventare fatale. Si rischia infatti di ritrovarsi in una situazione in cui il gap tra tecnologica e obiettivi di business cresce talmente tanto che diventa impossibile colmarlo.

Un altro aspetto critico relativo ai sistemi legacy è legato alla capacità di innovare. Innovazione e produttività vanno di pari passo: solo quando gli sviluppatori sono in grado di lavorare in modo produttivo piuttosto che preoccuparsi della manutenzione è possibile trovare spazio e risorse per innovare. Tuttavia come menzionato precedentemente il 75% delle risorse IT oggi è impegnata nel mantenimento di applicazioni esistenti, quindi solo il restante 33% ha effettivamente la possibilità di innovare [30]. Per favorire l'innovazione al posto della manutenzione sono possibili tre strade: accelerare il ciclo di vita dello sviluppo del software, rendere l'applicazione esistente più flessibile o aumentare il riutilizzo del codice.

3.1.2 Sintesi dei problemi

I problemi legati al mondo dei sistemi legacy possono essere quindi ricondotti a [23]:

- mancanza di flessibilità: l'applicazione delle modifiche richiede troppo tempo o risulta addirittura impossibile;

- **manutenibilità:** le attività di manutenzione ordinaria sono molto costose a causa della documentazione di scarsa qualità, della difficoltà nel capire il sistema e della mancanza di personale con le giuste competenze;
- **accessibilità:** spesso è difficile rendere il sistema disponibile ai clienti a causa della scarsa flessibilità;
- **costo delle operazioni:** i sistemi legacy hanno costi molto alti dovuti sia al mantenimento delle infrastrutture che a quello delle licenze;
- **interdipendenza delle applicazioni con l'infrastruttura:** tipicamente risulta impossibile aggiornarne una senza che le modifiche vadano ad impattare anche sull'altra.
- **stare al passo con gli obiettivi di business:** il sistema è poco agile e non riesce a stare al passo con le nuove funzionalità richieste;
- **scarsa capacità di investire nell'innovazione** a causa delle risorse impiegate nel mantenimento.

3.2 I SISTEMI LEGACY OGGI

Di fondamentale importanza risulta capire quale sia l'effettivo impatto dei sistemi legacy nell'attuale situazione di mercato. Si è visto che i sistemi informatici dell'azienda sono asset critici che incorporano tutta la conoscenza chiave acquisita durante la vita dell'organizzazione. In molti modi i sistemi informatici sono per le aziende quello che il cervello è per la specie animale: una massa complessa e poco capita sulla quale l'organismo fa affidamento per la propria esistenza [21]. Le odierne esigenze delle aziende sono: restare competitive nella recessione del mercato; incrementare i servizi senza aumentare i costi; perseguire l'agilità per permettere di stare al passo con i processi di business.

3.2.1 *Modernizzazione come opportunità*

Le aziende, al di là della crisi economica e del contesto generale, sono costantemente alla ricerca di strumenti per semplificare l'infrastruttura IT, abbatterne i costi, senza però perdere la capacità di sviluppare prodotti e servizi innovativi. Raggiungere questi obiettivi quando si fa affidamento su sistemi datati per soddisfare i bisogni primari del business è tutt'altro che facile. Proprio per questo motivo sono molte le aziende che intravedono una possibile soluzione ai propri problemi nella modernizzazione. L'evoluzione del software assume quindi un ruolo chiave: i sistemi legacy non possono essere un peso ma devono diventare uno strumento per trarre

vantaggio. Prima di capire come funziona la modernizzazione e perché possa essere la strategia vincente è necessario capire quale sia lo scenario in cui si trovano la maggior parte delle grandi aziende oggi.

3.3 GRANDI AZIENDE E SISTEMI LEGACY

3.3.1 *Mainframe*

Il termine mainframe proviene dalla preistoria dell'informatica e veniva utilizzato per indicare il grosso assemblato in cui si trovavano il processore centrale e tutti i dispositivi in entrata e in uscita. Successivamente il termine venne anche utilizzato per distinguere gli elaboratori di livello superiore da quelli di potenza inferiore, che erano caratterizzati da dimensioni ridotte. Ai giorni nostri con il termine mainframe ci si riferisce sostanzialmente ai sistemi *System z9 della IBM*, che discendono in linea diretta dal *System/360*. I mainframe attuali si differenziano dagli altri sistemi non solo per la velocità con cui sono in grado di eseguire una funzione, in quanto questa dipende solitamente dalle impostazioni dell'orologio interno, quanto per il loro design interno, basato sulla sicurezza, sull'affidabilità, sul perfetto bilanciamento delle performance e soprattutto sulla grande compatibilità con programmi di vecchia data, caratteristica che assicura la massima protezione degli investimenti realizzati dalle aziende nell'acquisto di software. Si tratta di macchine longeve e affidabili che possono funzionare ininterrottamente per anni e anni e che permettono di essere controllate ed eventualmente riparate senza dover essere fermate, possibilità particolarmente importante per tutte quelle applicazioni dove un'interruzione del servizio si tradurrebbe in una ingente perdita economica. Non per niente a queste macchine è stata anche assegnata la sigla [3] per la descrizione delle sue caratteristiche principali.

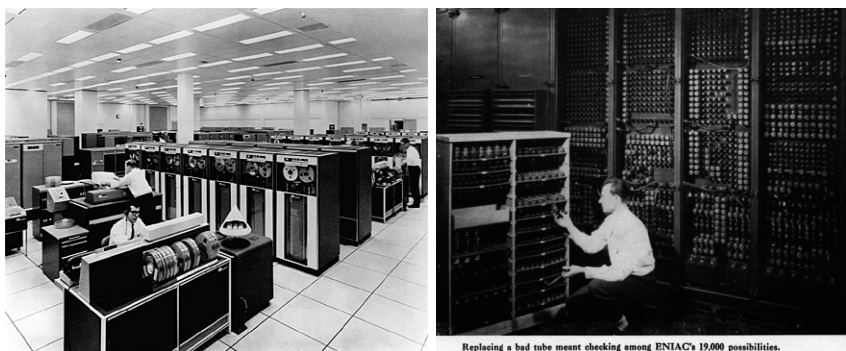


Figure 7: Macchine Mainframe

3.3.2 Caratteristiche e prestazioni

I mainframe sono macchine in grado di eseguire e in certi casi anche ospitare diversi sistemi operativi grazie all'utilizzo della tecnologia della virtualizzazione¹. Un solo mainframe, infatti, è in grado di svolgere il lavoro di centinaia di server di dimensioni inferiori, con conseguente riduzione dei costi per l'amministrazione e per la gestione, offrendo all'azienda un sistema maggiormente scalabile e decisamente più affidabile, come non sarebbe possibile ottenere con l'impiego dei server distribuiti. Tuttavia nonostante non siano assolutamente in grado di eguagliare i mainframe in quanto a livello di efficienza, sono tanti i server UNIX di livello superiore che offrono la tecnologia del partizionamento logico e sono tanti anche i produttori che cercano di promuovere l'utilizzo della virtualizzazione cercando di assimilare all'interno dei propri progetti le basi da cui si muovono i mainframe. I mainframe sono in grado di aumentare le proprie potenzialità elaborative in modo granulare e dinamico. Tra le caratteristiche uniche proprie dei mainframe c'è anche la capacità di elaborazione senza errori. Ad esempio, i System z9 ripetono ogni indicazione ricevuta due volte e comparando i risultati ottenuti, e se dovessero individuare una discordanza cercano di individuare il malfunzionamento e quando lo trovano attivano automaticamente un'unità di riserva. Questa modalità operativa viene chiamata *lock-stepping* in quanto tutti e due i processori compiono la stessa operazione in modo simultaneo. Da sempre, la potenza di calcolo di un mainframe viene calcolata utilizzando come unità di misura il Million Instructions Per Second (**MIPS**), ossia le milioni di istruzioni per secondo. Al giorno d'oggi si continua a utilizzare questa unità di misura non tanto per il calcolo delle istruzioni eseguite quanto per riferirsi alla potenza di elaborazione della macchina, che si calcola tenendo come riferimento un determinato carico di lavoro eseguito in un ambiente senza alcun tipo di vincolo strutturale.

3.3.3 Il mondo mainframe

La figura mostra uno spaccato del mondo mainframe che permette di discuterne i vari componenti. L'Multiple Virtual Storage (**MVS**) è un sistema operativo dei mainframe². Pur essendoci un unico processore, il sistema dà l'impressione, ai livelli software sovrastanti, che siano in corso più pro-

¹ Il termine virtualizzazione si riferisce alla possibilità di astrarre le componenti hardware, cioè fisiche, degli elaboratori al fine di renderle disponibili al software in forma di risorsa virtuale. Uno dei principali vantaggi è la razionalizzazione e l'ottimizzazione delle risorse hardware grazie ai meccanismi di distribuzione delle risorse disponibili di una piattaforma fisica.

² L'MVS era il sistema operativo più comunemente utilizzato sui computer mainframe di IBM System/370 e System/390

cessi ognuno operante sul proprio processore e con la propria memoria virtuale; ogni processo, su cui può essere eseguito un qualsiasi ambiente operativo o pacchetto software, tipicamente il **CICS**, dispone quindi di una copia virtuale della macchina sottostante. **CICS** è un ambiente operativo con funzionalità di Transaction Processing (**TP**) Monitor.

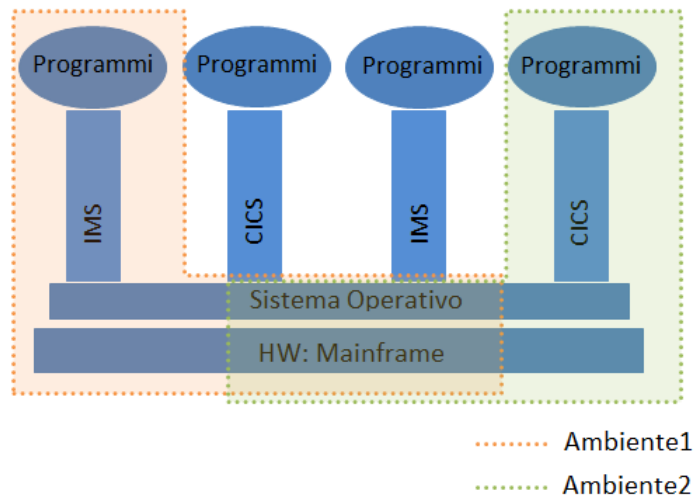


Figure 8: Definizione di ambienti

E' un grosso programma che si appoggia al sottostante **MVS** e su di esso appoggiano i programmi applicativi che vivono al suo interno, e per i quali il **CICS** costituisce a sua volta il sistema operativo: esso offre infatti anche servizi anche di basso livello, per cui quando il programma deve interagire con il sistema operativo, in realtà invoca i suoi servizi intermedi. E' proprio grazie al **CICS** che ogni programma può essere progettato e scritto come se fosse l'unico in esecuzione, poiché esso maschera tutte le problematiche di multitasking, multithreading ed accesso alle risorse condivise. Il **TP** monitor è un programma che monitora le transazioni passandole da uno stadio di un processo a un altro; esso assicura che le transazioni vadano a buon fine e intraprende opportune azioni in caso di errore. I **TP** monitor sono particolarmente utili nell'architettura *three-tier* per il *load balancing* poiché le transazioni possono essere gestite e invi server. Per quanto riguarda le funzionalità di **TP** Monitor, le operazioni vengono raggruppate in Logical Unit of Work ed il **CICS** si occupa degli aspetti di *commit/rollback* interagendo con i sistemi di database sottostanti, generalmente DB2. Tipicamente la transazione **CICS** viene avviata da un terminale 3270, ma esistono Application Programming Interface (**API**) per accedere alle funzionalità **CICS** anche da altre piattaforme remote.



Figure 9: Un esempio di terminale 3270

Con ambiente invece ci si riferisce a mainframe con il sistema operativo **MVS** e un **TP** Monitor sul quale si appoggiano i programmi applicativi che vivono in quell'ambiente. Quindi su una stessa macchina si possono avere molti ambienti del tutto indipendenti tra di loro, che all'esterno sembrano essere funzionalmente e logicamente equivalenti ad una moltitudine di macchine.

3.3.4 I costi

Ancora oggi molte organizzazioni a causa delle qualità che i mainframe offrono, come robustezza, affidabilità e sicurezza, decidono di sviluppare la parte backend dei loro sistemi informativi su tali macchine. I mainframe, nelle loro varianti più moderne vengono quindi ancora acquistati ed utilizzati da una fascia di organizzazioni che gestiscono tipicamente una enorme mole di dati.

Per capire quale sia la pervasività dei sistemi legacy e qual'è l'impatto di tali sistemi sul mercato basta pensare alle sole applicazioni scritte in linguaggio Cobol. Esistono oggi più di 200 miliardi di linee di Cobol, un dato che equivale all'80% del codice attivamente utilizzato a livello mondiale. Stando ai dati pubblicati da Butler Group [1](#), a livello mondiale, attualmente sono in esercizio applicazioni mainframe per un valore di 2 miliardi di dollari, con le quali vengono gestite circa il 70% di tutte le transazioni eseguite a livello mondiale. Il 60-80% delle imprese mondiali basa tutt'oggi le proprie attività di business su applicazioni che usano il linguaggio Cobol. Anche se non ne siamo consapevoli oggi ogni singolo aspetto della nostra vita è gestito da queste applicazioni, dai semafori stradali ai registratori di cassa, fino ai telefoni cellulari e alle prenotazioni di viaggio online. In breve, senza il Cobol incontreremmo davvero grandi difficoltà anche per la più ovvia delle operazioni [\[42\]](#).

Se da una parte il mantenimento di tali sistemi assorbe una fetta consistente degli investimenti, tutte le aziende si pongono il problema della compatibilità del costo di possesso dei mainframe rispetto alla costante e continua richiesta di riduzione delle spese. Dal punto di vista tecnologico, oggi, lo scenario di riferimento tipico di una media-grande organizzazione è il seguente: presenza di sistemi eterogenei con grandissima diffusione di

ambienti mainframe sui quali si trovano installate applicazioni scritte in linguaggio Cobol, su cui ricade l'attenzione in virtù di progetti di consolidamento, virtualizzazione, standardizzazione e ridisegno delle architetture in ottica *service oriented*.

Come dimostrano diverse ricerche uno dei maggiori problemi che riguarda la manutenzione delle applicazioni su mainframe è la conoscenza. Oggi la maggior parte dei Cio delle più grandi aziende è seriamente preoccupata per l'uscita progressiva dal mondo del lavoro di esperti di mainframe. Questo evento espone infatti le aziende a un aumento dei costi e dei rischi legati al business. Gli sviluppatori esperti oggi sono quindi un asset di business critico: ciò spiega il perché circa metà (43%) delle spese operative dei mainframe riguarda i loro stipendi. Sempre per Compuware [2] la perdita di competenze porterà ad aumentare i costi, poiché sviluppatori inesperti avranno bisogno di più tempo per comprendere le applicazioni mainframe. Inoltre apprendimento significa maggiore probabilità di errori e perdita di ricavi a causa delle interruzioni dell'applicazione mainframe. Per capire meglio questo aspetto basta pensare ad un altro dato, sempre ricavato da Compuware, secondo il quale si stima che un minuto di interruzione dell'applicazione mainframe potrebbe costare ad un'azienda la perdita di 14.000 dollari. Inoltre il costo stimato per la riscrittura dei programmi Cobol si aggira sui 25 dollari a linea. Dato che esistono oltre 200 milioni di queste linee, è facile capire che si tratterebbe di una spesa troppo elevata da sostenere per le aziende, senza contare l'interruzione delle operazioni derivante da lunghe strategie di riscrittura

MODERNIZZAZIONE

La modernizzazione è quell'insieme di modifiche effettuate sul sistema che permettono allo stesso di rimanere allineato con i mutevoli requisiti di business. Essa coinvolge cambiamenti più radicali rispetto a quelli del mantenimento, ma conserva una significativa parte del sistema. I cambiamenti spesso includono la ristrutturazione del sistema, importanti miglioramenti funzionali o l'inserimento di nuovi attributi. La modernizzazione quindi è usata quando i sistemi legacy richiedono cambiamenti importanti, ma hanno ancora un valore di business che deve essere preservato.

4.1 SFIDE DELLA MODERNIZZAZIONE

Le sfide della modernizzazione riguardano:

- **Obiettivi del business:** oggi le aziende sono costantemente soggette a pressioni di natura economica, spesso lo scopo principale del reparto IT è capire come sia possibile assolvere alle stesse funzioni ad un costo sempre minore. La modernizzazione può essere un ottimo modo per raggiungere tale obiettivo, ma deve essere valutata all'interno di questo contesto. Se la modernizzazione implica maggiore costo ci sono buone probabilità che non verrà neppure considerata dal management.
- **Complessità:** dimensioni ed ereditarietà inscrutabile sono tipiche dei sistemi legacy. E' necessario ridurle dove possibile e gestirle quando non eliminabili; la complessità è il più grande limite della modernizzazione.
- **Integrazione:** se all'interno dei propri sistemi si integrano componenti commerciali di cui periodicamente vengono rilasciati aggiornamenti, è necessario decidere come gestire l'integrazione. Spesso si decide di non integrare gli aggiornamenti onde evitare di rendere instabile un sistema, a meno che tali aggiornamenti non risolvano problemi altrimenti impossibili da eliminare.
- **Rischi:** la fase di identificazione dei rischi è una parte fondamentale del processo di modernizzazione. Prima applicare una determinata strategia è bene avere chiaro quali sono le problematiche che si potranno verificare.

A seconda di come questi problemi vengono risolti e superati si potranno ottenere diversi benefici [5]: una migliore soddisfazione dell'utente poichè ridefinire il sistema aiuta a servire i clienti in maniera più efficiente; l'ottimizzazione della produzione e dell'efficienza operativa poichè i miglioramenti del sistema portano all'aumento giornaliero del volume di business e favoriscono quindi una più alta produttività per ogni impiegato; una maggiore agilità, quindi la capacità di adattarsi alle esigenze di mercato in maniera più rapida e la capacità di consegnare il prodotto in tempi più brevi grazie all'ottimizzazione del processo di business; la riduzione dei costi di investimento e del tempo di sviluppo dovuti all'agilità e alla riutilizzabilità delle componenti; un aumento del ROI giustificato dal volume di business crescente e dalla capacità di essere sul mercato in tempi brevi, dalla riduzione dei costi per gli investimenti e dalla maggiore produttività per dipendente; il miglioramento degli standard di business; la riduzione dell'errore manuale e dipendenze dovute all'automatizzazione dei processi di business; infine la naturale crescita del profitto, poichè un alto volume di business aumenta la produttività.

4.2 APPROCCI ALLA MODERNIZZAZIONE

Quando si decide di modernizzare un sistema sono possibili diversi approcci riconducibili a quattro azioni di base cioè trasformare, sostituire, riscrivere o riutilizzare [23].

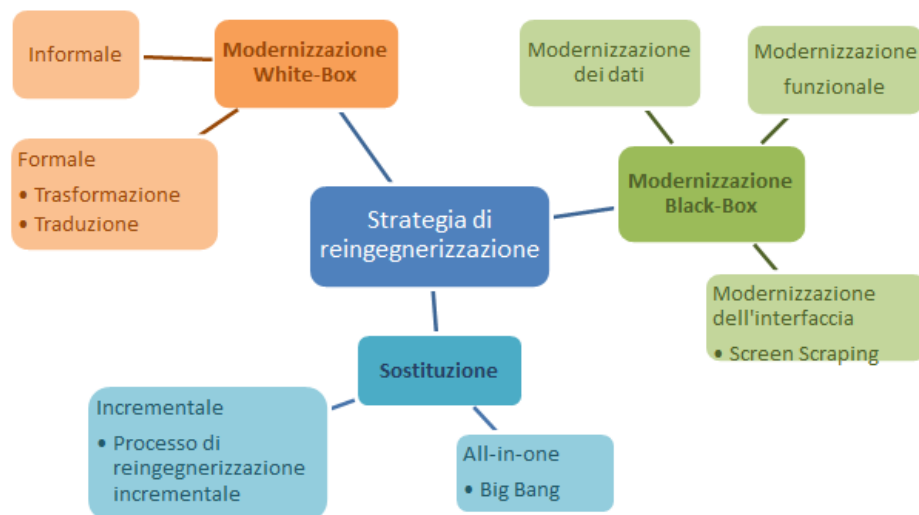


Figure 10: Strategie di reingegnerizzazione 6

La differenza tra questi approcci sta nel diverso sforzo richiesto per la comprensione del sistema. Quando è necessaria una conoscenza approfondita del funzionamento interno si parla di modernizzazione white-box; se

invece è sufficiente comprendere le interfacce, allora si parla di modernizzazione black-box.

4.2.1 *White-box*

E' l'approccio che si utilizza quando si trasformano o si riscrivono i sistemi e richiede un periodo iniziale di *reverse engineering* necessario all'acquisizione della conoscenza sul funzionamento interno. Durante questa fase si identificano le componenti del sistema e le relazioni tra le parti, in seguito alla quale è prodotta una rappresentazione astratta a un livello superiore [21]. La comprensione del programma è la principale forma di reverse engineering: riguarda la modellizzazione del dominio e l'estrazione delle informazioni dal codice utilizzando appositi meccanismi. La creazione di astrazione aiuta nella comprensione della sottostante struttura. Analizzare e capire un vecchio sistema è difficile perché nel tempo esso tende a collassare dentro alla sua stessa complessità. Anche se per quanto riguarda il reverse engineering recentemente sono stati compiuti dei passi avanti, tale pratica resta ancora rischiosa a causa della complessità e perché richiede un utilizzo intensivo delle risorse. Dopo che il codice è stato analizzato e compreso, si pensa ad una ristrutturazione del sistema o del codice stesso. La ristrutturazione del software può essere pensata come una trasformazione da una rappresentazione a un'altra allo stesso livello di astrazione dove il comportamento esterno viene preservato. Questa trasformazione è tipicamente usata per il miglioramento di alcuni attributi del sistema come manutenibilità e performance. Un esempio di modernizzazione white-box è la reingegnerizzazione. Essa è definita come "*l'analisi e la modifica di un sistema al fine di ricostruirlo in una nuova forma*" 17.

4.2.2 *Black-box*

E' una modernizzazione che si basa sul concetto di riuso. Come prima cosa si esaminano input e output del sistema all'interno del suo contesto operativo al fine di comprenderne le interfacce. Anche se acquisire la conoscenza circa il comportamento del sistema non è un lavoro semplice, resta comunque un approccio meno impegnativo rispetto a quello white-box [21]. Questo tipo di modernizzazione è spesso basata sul concetto di wrapping: si racchiude il sistema legacy all'interno di uno strato software che ne nasconde la complessità permettendogli di mostrare all'esterno solo l'interfaccia modernizzata. Il wrapping è usato per risolvere il problema del disallineamento tra le interfacce del sistema legacy e le interfacce che sono richieste alla fine dell'opera di modernizzazione. Teoricamente il wrapping è quindi una reingegnerizzazione a scatola chiusa durante la quale viene analizzato solo il comportamento visibile del vecchio sistema senza

alcuna conoscenza del suo funzionamento interno. Sfortunatamente questa soluzione non è sempre facilmente praticabile e spesso è richiesta almeno una parziale comprensione della struttura interna del sistema.

4.2.3 Altre soluzioni

Fino ad ora si è visto come applicare la modernizzazione attraverso riscrittura, riuso e trasformazione. L'ultima possibilità è quella di sostituire un sistema: non tutti però sono d'accordo nel considerare la sostituzione come un tipo estremo di modernizzazione e la considerano piuttosto una fase del processo evolutivo insieme al mantenimento e alla modernizzazione [36].

4.3 TECNICHE DI MODERNIZZAZIONE

Quando si decide di modernizzare ad alto livello esistono due scelte possibili [9]:

- *application engineering*: si sviluppa un nuovo sistema informatico dal nulla;
- *application reengineering*: per la progettazione del nuovo sistema si riutilizza qualcosa di già esistente.

4.3.1 Reingegnerizzazione

La reingegnerizzazione è una forma di modernizzazione definita da Tittley and Smith come *“la trasformazione sistematica di un sistema al fine di realizzare miglioramenti nella qualità delle operazioni, nelle capacità del sistema, nelle funzionalità, nelle performance, nella possibilità di evolvere a minor costo e nella riduzione del rischio per il cliente”* [51]. Tale approccio alla modernizzazione non è più economico della semplice manutenzione né efficace come la sostituzione; tuttavia resta uno degli approcci più pragmatici. E' infatti più efficace della semplice manutenzione ma meno rischiosa della sostituzione. Ci sono differenti tipi di strategie di reingegnerizzazione. Possono prendere la forma di [46]:

- *retargeting*;
- *revamping*;
- utilizzo di componenti commerciali;
- traduzione del codice sorgente;
- trasformazione funzionale.

4.3.1.1 Retargeting

Il retargeting è “il processo ingegneristico di trasformazione, rehosting o di porting di un sistema esistente in una nuova configurazione” [26] dove nuova configurazione significa una nuova piattaforma hardware, un nuovo sistema operativo o una piattaforma Computer-Aided Software Engineering (CASE). Qualche volta adottare piattaforme CASE non richiede cambiamenti nel sistema esistente, questi casi non sono trattati come azioni di retargeting. Nella nuova piattaforma sia logica di business che i dati dell'applicazione legacy rimangono intatti. Il retargeting di solito è motivato da un alto costo di mantenimento delle vecchie piattaforme e ovviamente da tutti i benefit che le soluzioni moderne possono portare [46].

4.3.1.2 Revamping

Il termine revamping è una forma comune di reingegnerizzazione in cui viene modificata solo l'interfaccia User Interface (UI) [45]. E' applicata quando il sistema funziona in maniera soddisfacente ma resta l'esigenza di offrire all'utente finale una migliore interazione. Quest'obiettivo si traduce in un miglioramento dell'interfaccia: non più quella a caratteri tipica dei terminali 3270, ma una GUI o meglio ancora un'interfaccia web accessibile da un browser. In molti casi infatti basta “mettere il sistema su Internet” [34] per ottenere un notevole miglioramento per il business dell'organizzazione. Ad esempio l'azienda privata può ottenere benefici economici in quanto capace di raggiungere nuovi clienti; per la pubblica amministrazione invece i vantaggi si possono tradurre in una maggiore offerta di servizi al cittadino. La tecnologia oggi offre una serie di strumenti di middleware, spesso proprietari, con cui diventa quasi immediato offrire i propri servizi sul web, utilizzando il vecchio sistema nascosto sotto alla nuova interfaccia; si tratta degli *screen scraper* e dei linguaggi di scripting. Queste tecnologie non richiedono alcun intervento sul sistema legacy e quindi sono applicabili a tutti i tipi di sistemi, anche quelli monolitici, ma soffrono di problemi di prestazioni. Chiaramente una scelta migliore è quella di sostituire l'interfaccia utente, riscrivendola, ma questo è possibile solo nel caso di sistemi altamente decomponibili. Esistono comunque dei tool che, partendo dal codice sorgente della vecchia interfaccia, aiutano nel processo di reverse engineering e di riscrittura della nuova GUI. Nella pratica si adotta spesso un approccio misto per cui in un primo momento, attraverso tecniche di screen scraping, viene sviluppata una nuova interfaccia grafica, che è resa disponibile in tempi molto brevi. Questa fase è utile anche per verificarne l'usabilità e l'accettabilità da parte dell'utente e funge quindi da prototipo. Nel frattempo si comincia la riscrittura della nuova interfaccia al termine della quale la precedente verrà ritirata, classificandola quindi come software usa e getta.

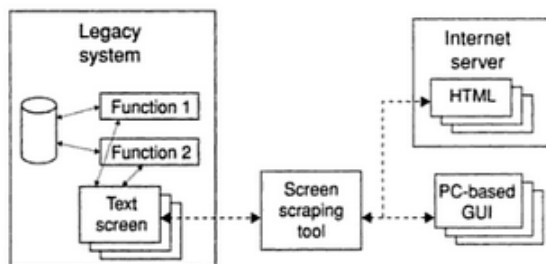


Figure 11: Revamping[45]

SCREEN SCRAPING Come visto tra i metodi di revamping uno dei più utilizzati è sicuramente lo screen scraping, un approccio alla modernizzazione di tipo black-box, che consiste nell'incapsulare le vecchie interfacce basate sul testo Text User Interface (TUI) all'interno di una nuova GUI. Questo approccio è relativamente semplice e rende la modernizzazione ben visibile. Tuttavia sotto alla nuova interfaccia grafica persiste la vecchia, quindi aggiungere nuove funzionalità o il successivo mantenimento resta molto difficile, dato che durante la modernizzazione non è stata migliorata l'estensibilità del sistema [4].

L'interfaccia utente è la parte più visibile del sistema; modernizzarla significa comunque apportare grandi miglioramenti all'usabilità e all'accessibilità, il che è sicuramente apprezzato dagli utenti finali. Lo screen scraping è un'estensione dell'emulazione di terminale: esso permette alle applicazioni client di simulare la pressione dei tasti e la lettura/scrittura di posizioni specifiche dello schermo, fornendo delle API per tali funzionalità. Quindi il browser web, attraverso CGI! (CGI!)¹, o il nuovo programma con la sua GUI chiamano lo screen scraper che a sua volta invoca l'interfaccia utente del vecchio sistema [9].

Questi tool generano automaticamente le nuove schermate a partire dalle vecchie. Dalla prospettiva del sistema legacy l'interazione con l'utente attraverso la nuova interfaccia grafica è indistinguibile rispetto quella che vede l'utente compilare i campi della schermata legacy. Dalla prospettiva dell'utilizzatore finale la modernizzazione ha successo dal momento in cui fornisce una nuova interfaccia grafica

¹ CGI! una tecnologia standard usata dai web server per interfacciarsi con applicazioni esterne generando contenuti web dinamici. Ogni volta che un client richiede al web server un Uniform Resource Locator (URL) corrispondente ad un documento in puro HyperText Markup Language (HTML) gli viene restituito un documento statico; se l'URL corrisponde invece ad un programma CGI!, il server lo esegue in tempo reale, generando dinamicamente informazioni per l'utente[25].

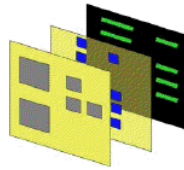


Figure 12: Screen scraping sea [7]

moderna e decisamente migliorata in termini di usabilità e accessibilità. Purtroppo, dalla prospettiva IT, il nuovo sistema è poco flessibile e difficile da mantenere esattamente come il vecchio. Questo approccio quindi è consigliato per sistemi stabili in cui il principale obiettivo è migliorare l'usabilità.

4.3.1.3 *Utilizzo di componenti commerciali*

Questo tipo di reingegnerizzazione consiste nella sostituzione di una parte del sistema legacy con componenti hardware e/o software disponibili sul mercato conosciuti come Commercial Of The Shelf (COTS). Un grande vantaggio di questa soluzione è la riduzione della quantità di codice che richiederà la manutenzione nel futuro. Uno svantaggio invece consiste nell'impossibilità di riutilizzare la logica di business. In ogni caso, quando si calcolano i costi di questa soluzione non bisogna trascurare le spese aggiuntive che non sono direttamente legate prezzo di componenti e licenze, bensì allo sforzo necessario per l'estrazione delle parti del sistema che devono essere sostituite oltre che al codice *collante* che permetterà la cooperazione tra le vecchie e le nuove componenti .

4.3.1.4 *Traduzione del codice sorgente*

Un'altra tecnica di reingegnerizzazione usata comunemente è la traduzione del codice. In pratica si tratta di “una trasformazione del codice sorgente da un linguaggio ad un altro, o da una versione di un linguaggio a un'altra” [26]. Può essere motivata ad esempio da una mancanza di compatibilità tra la piattaforma di partenza e quella di destinazione dato che il vecchio linguaggio potrebbe non essere supportato dalla nuova piattaforma. E' una delle tecniche di reingegnerizzazione più rischiose a causa dei costi relativamente alti di realizzazione, spesso sottostimati, specialmente in relazione ai risultati che si possono ottenere [46]. Una possibile soluzione che consente di contenere i costi è rendere la migrazione automatizzata: la migrazione automatizzata è di natura algoritmica e ha quindi il vantaggio di non richiedere l'utilizzo di intelligenza umana nel processo di trasformazione. In ogni caso la traduzione del codice sorgente non è facile:

se non eseguita correttamente si potrebbe finire con l'ottenere semplicemente un nuovo sistema che sembra esattamente uguale al precedente ma riscritto con un differente linguaggio, senza cioè quegli elementi specifici della nuova piattaforma che permetterebbero un miglioramento della qualità.

4.3.1.5 *Trasformazione funzionale*

La trasformazione funzionale è una forma di reingegnerizzazione in cui anche la struttura del programma viene modernizzata. Per esempio il codice procedurale può essere trasformato in codice orientato agli oggetti, le applicazioni possono essere divise in moduli o gli spazi di archiviazione dei dati potrebbero essere cambiati. La trasformazione funzionale può essere divisa in tre categorie: miglioramenti alla struttura del programma, modularizzazione del programma e reingegnerizzazione dei dati.

4.3.2 *Considerazioni finali*

I diversi approcci presentati non sono in contrapposizione tra di loro e l'applicazione di uno non esclude automaticamente l'utilizzo anche parziale degli altri [9]. Inoltre può capitare che si cominci con uno e si passi nel tempo ad uno differente; ad esempio non è escluso che un'organizzazione, per soddisfare esigenze immediate, inizi ad esporre alcuni servizi sul web attraverso semplici wrapper d'accesso o una migrazione delle interfacce utente, successivamente passi a sviluppare wrapper ad oggetti ed infine decida di migrare completamente l'implementazione del proprio sistema informativo, ormai nascosta dietro ad un modello ad oggetti stabile, dal mondo mainframe ad un'architettura differente. Oppure che un'organizzazione, il cui obiettivo originario era una migrazione, decida invece dopo aver implementato un gateway di accontentarsi di questo incapsulamento del sistema dietro ad una nuova interfaccia, realizzando quindi alla fine un wrapping del proprio sistema. Spesso i concetti di migrazione e di integrazione possono diventare interscambiabili, dato che è evidente la similitudine, non solo concettuale ma anche tecnologica e della relativa offerta di mercato, tra un wrapper ed un gateway. C'è da tener presente che nessun approccio è completo, nel senso che nessuno può essere applicato con successo a tutte le categorie di sistemi legacy. Ad esempio l'approccio dell'integrazione con il wrapping ha maggior senso nel contesto di un'architettura ad oggetti distribuiti, mentre altri approcci potrebbero essere più convenienti per sistemi isolati e non troppo estesi. La scelta di un approccio non è affatto banale e solleva molti problemi che devono essere attentamente vagliati, non ultimo l'effettivo supporto del mercato.

4.4 RISK MANAGED MODERNIZATION

Il risk management è quel processo che riguarda l'identificazione, l'analisi, il monitoraggio e la gestione dei rischi di uno specifico progetto [50]; non si tratta di un'idea nuova ma fa parte del processo di sviluppo a spirale sviluppato da Boehm². La figura mostra un diagramma delle attività UML dell'approccio Risk Managed Modernization (RMM) [35]. Gli ovali nel diagramma rappresentano le attività, le frecce le transizioni tra le attività. Le barre di sincronizzazione orizzontali richiedono il completamento delle precedenti attività prima che le nuove possano avere inizio. La fine del processo è rappresentata da un piano integrato di modernizzazione [46].

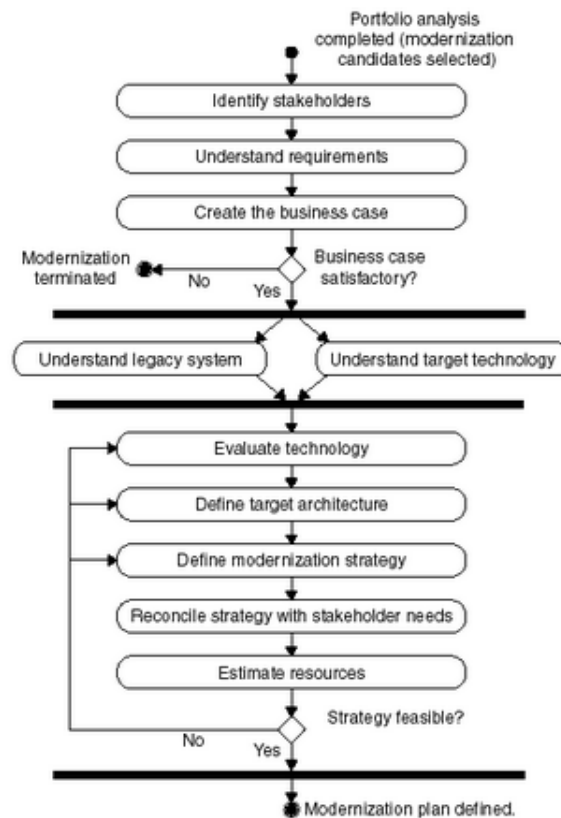


Figure 13: Diagramma UML delle attività

² Nel modello a spirale, il software viene sviluppato in una sequenza di versioni crescenti mediante la discussione di molteplici regioni che caratterizzano ogni giro della spirale. La spirale ha il pregio di considerare tutto il ciclo di vita; oltre ad arrivare alla consegna del software, permette di strutturare e programmare anche l'attività successiva all'installazione[47].

4.4.1 Analisi del portfolio

L'analisi del portfolio stabilisce per un insieme di sistemi le misure tecniche di qualità e di valore di business e valuta questo insieme rispetto alle misure. La qualità tecnica è la misura di bontà di un'applicazione o di un sistema rispetto ad un insieme di criteri tecnici. Esempi di criteri per la qualità sono la frequenza di rilascio di nuove release, la facilità di effettuare cambiamenti, l'affidabilità dell'hardware e del software, l'organizzazione dell'infrastruttura e le performance del sistema. Il valore di business invece misura l'importanza del sistema o dell'applicazione per l'organizzazione. Esempi di criteri di valore di business sono il profitto, il livello di utilizzo, il numero di obiettivi soddisfatti, la soddisfazione dell'utente e il valore delle informazioni memorizzate nel sistema[55]. I sistemi legacy quindi devono essere valutati rispetto a queste misure e posizionati in un grafo come mostrato in figura.

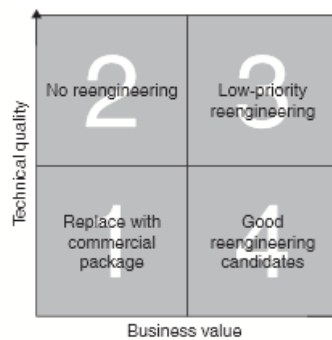


Figure 14: Grafo per l'analisi del portfolio

I riquadri di valutazione dividono i sistemi in:

- bassa qualità, basso valore;
- alta qualità tecnica, basso valore economico;
- alta qualità, alto valore economico;
- bassa qualità, alto valore economico.

Da notare che la stesura del grafico da origine ad una distribuzione in cui i prodotti aziendali sono identificati da bolle; l'individuazione della strategia corretta da applicare per ogni prodotto quindi non è univoca. In pratica questo significa che per la modernizzazione di un software potrebbe essere necessaria la cooperazione sinergica di approcci differenti. Dopo aver risolto le ambiguità e concluso quindi l'analisi preliminare del portfolio, bisogna tenere conto di tutte le altre problematiche che riguardano

l'organizzazione e le risorse; tali problematiche infatti possono far crescere notevolmente il rischio. Tipicamente esse includono la politica, i costi, le scadenze, la disponibilità dello staff, le skill tecniche richieste per la modernizzazione e il training necessario agli utenti per usare i sistemi modernizzati. Un volta che si è analizzato il portfolio e si sono identificati gli altri possibili problemi si scelgono i sistemi da modernizzare assegnando ad ognuno una certa priorità. A questo punto si può scegliere la giusta strategia di modernizzazione.

4.4.2 *Identificazione degli stakeholder*

Gli stakeholder sono gli sviluppatori, i tester, i manutentori, gli amministratori di sistema, i clienti, i vendor, gli sponsor, gli utenti, gli architetti, cioè persone che ricoprono un determinato ruolo all'interno del progetto. Queste persone dovranno in ultimo giudicare il risultato e l'impatto del progetto di modernizzazione. Durante questa fase è importante ottenere il loro supporto e il loro accordo. Questo può essere difficile perché gli stakeholder hanno diversi interessi e di conseguenza vedono anche il rischio in maniera differente.

4.4.3 *Comprensione dei requisiti*

Durante lo sviluppo del software e l'attuazione della modernizzazione la definizione dei requisiti potrebbe essere un task difficile. In generale i requisiti possono essere divisi nelle seguenti categorie:

- utente: sono le capacità che devono essere fornite dal sistema; questi requisiti sono spesso espressi in forma di task o di attività che devono essere supportati dal sistema;
- sistema: descrivono le capacità del sistema e il sistema stesso;
- vincoli: includono le decisioni che sono già state prese come l'interazione con altri sistemi, lo sviluppo di standard e i costi;
- non funzionali: comprendono le proprietà comportamentali come la performance, l'usabilità e la sicurezza del sistema.

Selezionare i requisiti in modo accurato chiarisce cosa ci aspetta dal sistema, riducendo i rischi. Molti progetti falliscono proprio per una mancanza di rigore nel processo di selezione. Usare l'aiuto degli stakeholder per validare i requisiti può aiutare ad assicurarsi che il sistema finale sarà effettivamente come ci si aspetta.

4.4.4 *Creazione del caso di business*

Un caso di business è un documento che supporta il processo decisionale e la pianificazione; poiché i progetti di modernizzazione richiedono molto lavoro e altrettante risorse finanziarie, il caso di business può aiutare a convincere il management che il progetto è economicamente possibile. Pertanto è spesso la chiave per ottenere sia l'approvazione che i fondi. Per la creazione di un buon caso di business è fondamentale capire a fondo i requisiti degli stakeholder.

4.4.5 *Comprendere il sistema legacy e la tecnologia software esistente*

Capire sia il sistema che il contesto è essenziale per il successo di ogni modernizzazione. La sfida è conquistare questa competenza in modo efficiente, economico e puntuale; generalmente si utilizzano tecniche come il reverse engineering, la comprensione del programma e la ricostruzione dell'architettura. Per quanto riguarda la comprensione della tecnologia esistono tre classi di informazioni che sono di interesse per la modernizzazione: le tecnologie usate per costruire il sistema, compresi linguaggi e database utilizzati, le tecnologie moderne disponibili e le tecnologie offerte dai vendor di sistemi legacy.

4.4.6 *Valutare la tecnologia*

Una volta comprese le tecnologie disponibili e le loro capacità, si possono fare dei paragoni. Se esiste una sovrapposizione di capacità è necessario vedere se le tecnologie risolvono lo stesso tipo di problema con una differente qualità di servizio. Nel progetto di modernizzazione infatti si potrebbe includere più di una tecnologia che soddisfa lo stesso scopo ma fornisce un differente Quality of Services (QoS) per la modernizzazione.

4.4.7 *Definire il target dell'architettura*

L'architettura target rappresenta l'architettura del sistema che si vorrebbe ottenere, fornendo la visione tecnica per la modernizzazione. Deve essere descritta in modo che supporti una adeguata comunicazione tra gli stakeholder e quindi richiede una descrizione che usi differenti punti di vista con differenti livelli di granularità e specificità.

4.4.8 *Definire la strategia di modernizzazione*

La modernizzazione dei sistemi è complessa e multistrato; poiché i sistemi legacy spesso supportano operazioni critiche per la maggior parte

delle aziende, agire sull'intero sistema in una volta risulta impossibile. Come risultato si ha che i sistemi legacy sono tipicamente modernizzati in maniera incrementale [34]: inizialmente il sistema è composto interamente da codice legacy ma ogni volta che un nuovo incremento è completato la percentuale di tale codice diminuisce. La migrazione assicura che il sistema rimanga funzionante durante il processo di modernizzazione [45]: una strategia efficace definisce la trasformazione dall'architettura del sistema legacy all'architettura del sistema modernizzato. Durante il periodo di modernizzazione le tecnologie possono cambiare, ed è richiesta l'acquisizione di conoscenza addizionale circa il sistema esistente; la modernizzazione deve accomodare anche questi cambiamenti. Al fine di soddisfare questi requisiti la strategia deve minimizzare il costo di sviluppo e di deploy, supportare delle scadenze predicibili, mantenere la qualità del prodotto, minimizzare i rischi, incontrare le aspettative sulle performance e mantenere la complessità ad un livello accettabile.

4.4.9 *Stima delle risorse per la strategia di modernizzazione*

L'ultimo passo del RMM è stimare i costi della modernizzazione. Una volta completato questo task si ha una conoscenza approfondita del sistema e delle tecnologie di modernizzazione, dell'architettura target e della strategia di modernizzazione. Basandosi su queste informazioni il management deve determinare se la strategia suggerita è fattibile in termini di risorse e vincoli. In caso affermativo si mette in atto la modernizzazione, in caso contrario deve essere fornita una spiegazione.

Part II

MODERNIZZAZIONE IN PRATICA

UN APPROCCIO BLACK-BOX: WEBRATIO

Come si è detto precedentemente i possibili approcci alla modernizzazione sono suddivisibili in due macro categorie: white e black-box. Di seguito si presenta uno strumento per l'implementazione della modernizzazione black-box.

5.1 LO STRUMENTO: WEBRATIO

WebRatio è uno strumento commerciale per il disegno e l'implementazione di applicazioni web caratterizzato da un approccio agile e Model Driven Architecture (MDA)[48]. Tale strumento, se opportunamente personalizzato, permette il revamping delle applicazioni legacy, cioè quel tipo di reingegnerizzazione in cui si interviene sulla sola interfaccia[45]: questo approccio è utile quando il sistema funziona in maniera soddisfacente ma resta l'esigenza di offrire all'utente finale una migliore interazione. In particolare WebRatio crea per la nuova applicazione non una GUI ma un'interfaccia web, permettendo quindi un miglioramento anche in termini di accesso oltre che di usabilità.

Nei prossimi capitoli si mostrerà come WebRatio, se adeguatamente integrato da opportune librerie, sia in grado di modernizzare una particolare famiglia di applicazioni caratterizzate da interfaccia testuale e accesso tramite terminale 3270.

5.1.1 La metodologia agile

Per metodologia agile si intende un particolare metodo per lo sviluppo del software che coinvolge quanto più possibile il committente, ottenendo in tal modo una elevata reattività alle sue richieste.

Un processo di sviluppo agile, secondo il *Manifesto per lo Sviluppo Agile di Software*¹ è tale se:

- si focalizza sugli individui partecipanti al progetto e sulla loro interazione piuttosto che sui processi e gli strumenti;

¹ La formalizzazione dei principi su cui si basano le metodologie agili è stata oggetto del lavoro di un gruppo di progettisti software e guru dell'informatica che si sono prontamente riuniti nell'*Agile Alliance*. Il documento finale di questo lavoro è stato poi sottoscritto da un gruppo di questi professionisti, molti dei quali hanno anche sviluppato alcune delle metodologie leggere più famose. Fonte [<http://agilemanifesto.org/iso/it/>].

- antepone il buon funzionamento del software alla produzione di documentazione esaustiva;
- si basa sulla collaborazione con il cliente piuttosto che sulla negoziazione di un contratto;
- offre una buona risposta ai cambiamenti piuttosto che limitarsi a seguire i piani.

WebRatio si configura come strumento di supporto alla metodologia agile in quanto permette una reazione efficace ai cambiamenti, la comunicazione efficiente tra tutti gli stakeholder e l'assorbimento del cliente nel team di sviluppo. Tale strumento prevede infatti consegne incrementali e frequenti: ogni iterazione è un piccolo progetto a sé stante alla fine della quale il team deve rivalutare le priorità di progetto.

5.1.2 *La Model Driven Architecture*

La cosiddetta *model driven architecture* rappresenta un nuovo approccio per lo sviluppo software definito dall'Object Management Group (OMG)². Lo sviluppo di applicazioni segue una filosofia del tutto differente rispetto a quello tradizionale: ci si concentra sulla generazione di modelli e sulla loro trasformazione, in un processo che termina con la creazione del codice vero e proprio del software finale. La specifica dei sistemi software, quindi, risulta del tutto indipendente da uno specifico linguaggio di programmazione: le applicazioni vengono generate in maniera del tutto indipendente dalla tecnologia su cui verranno eseguite Mellor [29].

5.1.2.1 *I modelli MDA*

L'idea che sta alla base di questo principio è che il modello diventa il codice sorgente del sistema da cui, in seguito, vengono semplicemente generati gli eseguibili. In questo modo i modelli possono ricoprire differenti livelli di astrazione, dai diagrammi concettuali nello spazio del problema fino ai modelli dettagliati a basso livello specifici di una determinata piattaforma. In generale il *model driven development* permette di generare applicazioni a partire da modelli formali Kleppe et al. [27].

Il processo inizia con la creazione di un modello indipendente dalla piattaforma (Platform Independent Model (PIM)) che viene poi trasformato in un modello specifico di una determinata piattaforma (Platform Specific Model (PSM)), infine tradotto in codice sorgente. Si noti come, in realtà, questi due modelli rappresentino modi differenti di descrivere lo stesso sistema.

² (<http://www.omg.org>)

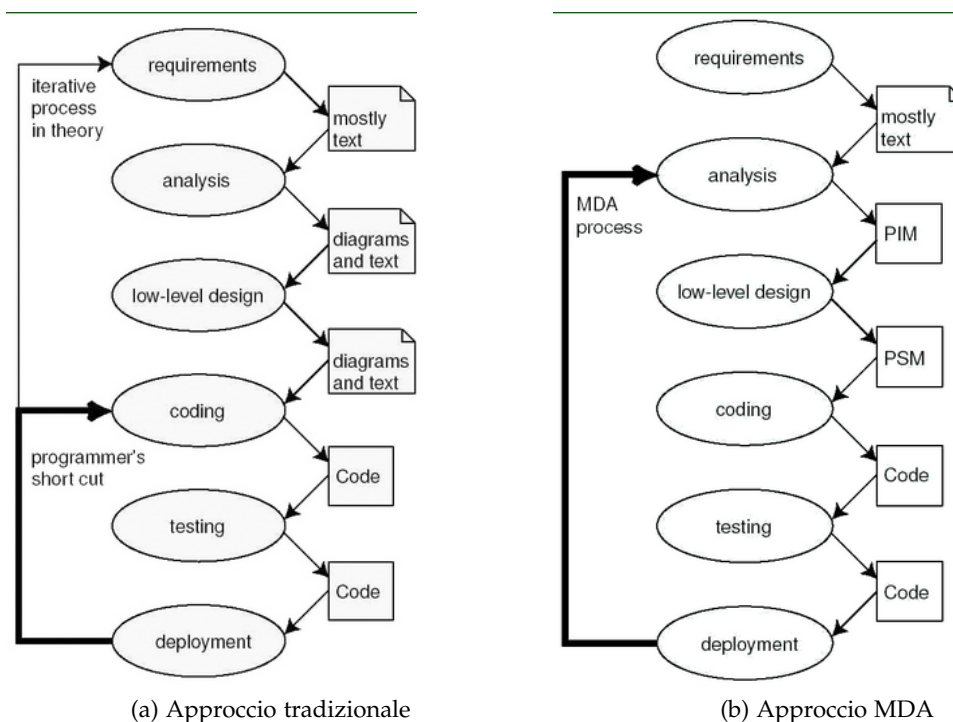


Figure 15: Differenti approcci al processo di sviluppo del software

- il **PIM** è una rappresentazione della logica di business e del comportamento dell'applicazione e non contiene alcun dettagli relativo alla tecnologia utilizzata per la sua implementazione;
- il **PSM** è una rappresentazione di più basso livello del sistema, del tutto dipendente dalla piattaforma tecnologica a cui si riferisce.

Pertanto per implementare un **PIM** su di una specifica piattaforma è necessario uno strumento capace di trasformarlo in un **PSM**: esso deve conoscere la tecnologia di destinazione ed essere in grado di tradurre gli artefatti contenuti nel **PIM** in componenti specifici del **PSM**. Ad esempio un singolo oggetto della logica di business, quindi facente parte del **PIM**, può essere trasformato in una tabella di una base dati o in un *entity bean* a seconda della tipologia di **PSM** scelta.

Il **PIM** di WebRatio è specificato utilizzando WebML, un linguaggio creato specificatamente per la modellazione di applicazioni web, mentre tutte le restanti logiche che portano alla creazione del **PSM** sono orientate alla creazione di artefatti tipici della piattaforma di riferimento di WebRatio: la *Java Enterprise Edition*. In particolare per l'accesso ai dati viene utilizzato il framework *Hibernate*, la logica di business è implementata utilizzando *servolet* e *java bean* mentre il front-end utilizza pagine *JavaServer Pages (JSP)* per la presentazione dei contenuti. Questa netta divisione tra i diversi strati segue il paradigma *Model View Controller (MVC)*, tipico del framework *Struts* ampiamente sfruttato da WebRatio.

Per permettere la generazione automatica del codice a partire da un modello, quest'ultimo deve seguire una sintassi e una semantica ben definite: WebRatio si basa sul linguaggio WebML che verrà discusso nel dettaglio nei prossimi paragrafi.

5.1.2.2 Vantaggi

L'adozione dell'approccio MDA porta con se numerosi vantaggi. Lo sviluppatore può concentrarsi sulle attività di analisi e progettazione avendo chiaro il dominio del problema mentre le problematiche relative a tutti gli aspetti legati alla fase implementativa possono essere trascurati. Si ottiene così una separazione tra gli aspetti di progettazione e quelli implementativi che rischiano di distogliere l'attenzione dal tema principale che ha spinto alla creazione del software. Inoltre permette di migliorare la produttività del processo di sviluppo del software, riducendone costi e tempi. Grazie alla notevole diminuzione dello sforzo implementativo è infatti possibile effettuare rilasci in un periodo di tempo di gran lunga inferiore rispetto ai metodi tradizionali; anche le eventuali attività di prototipazione vengono notevolmente semplificate. Il processo di sviluppo risulta maggiormente strutturato, favorendo la divisione dei compiti all'interno del gruppo di lavoro a vantaggio dell'efficienza. Infine il fatto che MDA preveda una generazione delle applicazioni in modo del tutto indipendente dalla tecnologia su cui verranno eseguite, preserva gli investimenti effettuati per la costruzione dei sistemi durante l'evoluzione tecnologica. Si può dire quindi che l'utilizzo di questo approccio permetta di migliorare la qualità delle applicazioni, il grado di riutilizzo e l'efficienza di sviluppo.

Il successo dell'approccio Model Driven, inoltre, è strettamente associato alla sua usabilità pratica: se la creazione del modello fosse più problematica dell'implementazione delle funzionalità tramite codice, difficilmente gli sviluppatori sarebbero interessati al passaggio verso questa filosofia di sviluppo. Questo aspetto risulta particolarmente importante nel mondo legacy, in quanto permette alle risorse IT abituate a lavorare con tecnologie datate di lavorare per la produzione e lo sviluppo di applicazioni moderne come quelle web.

5.2 IL LINGUAGGIO: WEBML

WebML è una notazione visuale proposta in sede internazionale da un gruppo di ricercatori di Milano per la specifica della composizione e della navigazione di applicazioni per il Web e utilizza un processo di sviluppo di tipo MDA/Model Driven Development (MDD). In particolare lo scopo di WebML è supportare il design e l'implementazione delle cosiddette applicazioni web *data intensive*, cioè quelle applicazioni per il web il cui scopo

principale è gestire e pubblicare grandi quantità di dati [19]. Ne sono un esempio:

- commercio: cataloghi elettronici, e-mail, aste elettroniche;
- contenuto: quotidiani online, biblioteche elettroniche, siti istituzionali;
- servizi: e-banking, prenotazioni, monitoraggio degli ordini;
- comunità: portali tematici, forum.

Per raggiungere questo scopo WebML utilizza modelli dei dati concettuali già esistenti mentre propone una notazione originale per esprimere le caratteristiche della navigazione e della composizione dell'interfaccia ipertestuale.

5.2.1 *Obiettivi e vantaggi*

Fin dalla sua creazione WebML è stato pensato per supportare al meglio il processo di sviluppo delle applicazioni orientate al Web e si basa quindi su dei principi fondamentali che mirano a semplificare il lavoro degli ingegneri del software e degli sviluppatori:[19]:

- espressività: il modello deve essere capace di descrivere le applicazioni web ad un livello di complessità paragonabile a quello di sistemi sviluppati manualmente;
- facilità d'uso: il modello deve essere semplice da imparare per quegli sviluppatori non esperti di ingegneria del software; a tale scopo esso prevede un numero minimo di concetti che permettono la generazione del codice;
- implementabilità: il modello deve contenere informazioni sufficienti per permettere la generazione del codice di tutti i livelli di un'applicazione web dinamica e il generatore di codice deve produrre codice ottimizzato.

5.3 IL MODELLO CONCETTUALE WEBML

La modellazione concettuale consiste nella definizione di schemi concettuali che esprimano l'organizzazione dell'applicazione ad un alto livello di astrazione e in modo indipendente dai dettagli di implementazione [37]. Il modello concettuale di WebML è così diviso[19]:

- Modello strutturale: esprime il contenuto dell'applicazione web in termini di entità e relazioni. WebML non propone un altro linguaggio per la modellazione dei dati ma è compatibile con le classiche

notazione Entity/Relationship (E/R) e con il diagramma delle classi UML.

- Modello dell'ipertesto: tale modello utilizza un approccio piuttosto differente dalle precedenti proposte. Anziché definire delle primitive per la rappresentazione di tutte le possibili combinazioni per organizzare l'interfaccia ipertestuale, l'idea centrale è quella di focalizzarsi su un numero minimo di concetti, i quali possono essere composti in modi ben definiti per ottenere le diverse configurazioni adatte a soddisfare i requisiti dell'applicazione. Ogni ipertesto definisce una vista la cui descrizione consiste a sua volta in due sottomodelli:
 - Modello di composizione: specifica quali pagine compongono l'ipertesto e quali *unit* compongono la pagina;
 - Modello di navigazione: specifica come le pagine e le *unit* sono collegate all'interno dell'ipertesto.
- Modello di presentazione: specifica il *layout* grafico dell'applicazione.

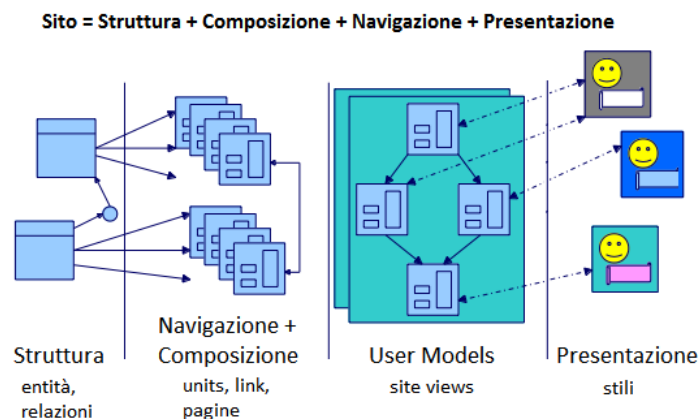


Figure 16: Il modello concettuale di WebML

Una caratteristica distintiva di WebML è l'assenza di un sotto-modello separato specifico per la logica di business. Essa è inclusa in parte nel modello strutturale, sotto forma di specifiche di derivazione dei dati, e in parte nel modello di ipertesto, sotto forma di *content* e *operation unit*. Questa scelta ha permesso la semplificazione del modello al prezzo di due comportamenti non attesi:

- la proliferazione di *custom unit* per includere logiche di business non standard;

- la crescita di complessità del modello di presentazione che è stato utilizzato per catturare anche il comportamento dell'applicazione lato client, non esprimibile secondo altre modalità.

Un'ulteriore semplificazione risiede nel fatto che non esiste un modello architetturale esplicito. Le risorse fisiche come sorgenti di dati, application server e *web services* non sono modellate in un diagramma ma sono semplicemente dichiarate come risorse del progetto all'interno degli strumenti di sviluppo e associate agli elementi del modello che le utilizzano.

Un modello WebML presenta un livello di astrazione molto più alto rispetto ad una rappresentazione orientata agli oggetti: i vari concetti di un modello solitamente sono mappati su differenti artefatti software, che risiedono spesso in diversi livelli. Ad esempio una pagina WebML, le sue unit e i suoi link, sono una rappresentazione compatta di molteplici artefatti come: il codice che si occupa dell'estrazione dei dati nel livello dei dati, gli oggetti che memorizzano i contenuti all'interno del livello di business o di presentazione e i tag delle pagine web che traducono il contenuto degli oggetti in markup. La decisione di mantenere un alto livello di astrazione ha aumentato la compattezza del modello a discapito del suo realismo. Infatti agli occhi di uno sviluppatore il modello WebML di una pagina unisce elementi che in realtà si trovano in strati differenti dell'applicazione. Ad ogni modo se lo scopo della modellazione è la generazione del codice, la possibilità di creare un modello completo in modo semplice deve prevalere sul realismo della sua rappresentazione. Dato che anche un progetto di piccole dimensioni può tipicamente comprendere decine di pagine contenenti centinaia di componenti è semplice immaginare come l'innalzamento del livello di astrazione sia cruciale per ottenere un certo grado di usabilità.

Un aspetto non ovvio della progettazione basata sui modelli, che distingue WebML dai tradizionali metodi orientati agli oggetti, è l'assenza di un modello separato per la rappresentazione del comportamento dell'applicazione. Tali aspetti sono incorporati nel modello dell'ipertesto che presenta una semantica operativa fissa per tutte le applicazioni WebML e quindi non deve essere specificata esplicitamente dal progettista. In pratica il modello dell'ipertesto è rappresentato da una macchina a stati finiti in cui gli eventi catturano le interazioni da parte dell'utente e le transizioni descrivono la propagazione della computazione da un componente all'altro; il flusso di controllo è espresso per mezzo di condizioni sulle transizioni.

5.3.1 Il modello dei dati

Un modello dei dati è un insieme di meccanismi di astrazione per definire una base di dati con associato un insieme predefinito di operatori e di vincoli di integrità. In sostanza si definisce un modello dei dati per rispondere a domande come[47]:

- quali sono gli oggetti informativi da pubblicare tramite l'applicazione?
- quali sono le proprietà che li caratterizzano?
- in che modo i vari oggetti sono correlati?

La modellazione e la gestione dei dati è una delle discipline più diffuse e consolidate del mondo IT e quindi esistono già numerosi linguaggi di modellazione e linee guida: per questa ragione WebML non ne propone una nuova, piuttosto sfrutta il già esistente modello [E/R](#).

Gli elementi fondamentali del modello dei dati di WebML sono le entità, definite come contenitori di elementi di dati, e le relazioni cioè le connessioni semantiche tra entità. Le entità hanno nomi e proprietà, chiamate attributi, con un tipo associato. Le entità possono essere organizzate in gerarchie generalizzate e le relazioni possono essere ristrette e vincolate attraverso le cardinalità. Nel design delle applicazioni web è spesso richiesto di calcolare il valore di qualche attributo o relazione di un'entità a partire dal valore di qualche altro elemento dello schema. Attributi e relazioni così definiti sono chiamati derivati. Gli attributi e le relazioni derivate possono essere denotati aggiungendo il carattere / prima del nome, e le loro regole di computazione possono essere specificate come espressioni logiche aggiunte alla dichiarazione dell'attributo o della relazione. [13].

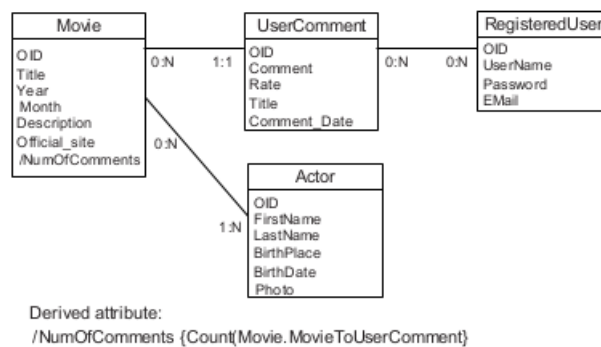


Figure 17: Segmento di uno schema dei dati di un'applicazione web per la gestione di film

Le primitive del modello dei dati sono:

- entità: una classe di oggetti nel dominio dell'applicazione;
- attributo: una proprietà di una entità;
- relazione: una connessione tra entità;
- gerarchia IS-A: costruito utilizzato per classificare o raggruppare.

5.3.2 Il modello dell'ipertesto

Il modello dell'ipertesto permette la definizione dell'interfaccia di *front-end* che è mostrata all'utente tramite il browser. La sua definizione risponde a domande come[19]:

- in che modo l'utente deve fruire del contenuto pubblicato tramite il sito?
- quali sono le pagine nell'ipertesto tramite cui l'utente può accedere ai contenuti?
- quale informazione deve essere pubblicata in ogni pagina?
- in che modo i nodi dell'ipertesto sono collegati tra loro?

Essa permette la definizione delle pagine e della loro organizzazione interna in termini di componenti il cui scopo è esporre il contenuto (*content unit*). Tale modello prevede anche il concetto di link tra la pagine e tra le *content unit* definendo la logica di navigazione. Le componenti possono anche specificare operazioni (*operation unit*), come la modifica del contenuto o le procedure di login/logout di un utente.

La struttura modulare di un applicazione di *front-end* è quindi definita in termini di viste, aree, pagine e *content unit*. Una vista è un particolare ipertesto, disegnato per incontrare specifici requisiti. Esso consiste di aree, le sezioni principali dell'ipertesto, che comprendono ricorsivamente altre sottoaree o pagine. Le pagine sono l'effettivo contenitore di informazioni mostrato all'utente. Diverse viste possono essere definite sopra lo stesso schema dei dati per soddisfare i bisogni di differenti gruppi di utenti o per rendere il contenuto fruibile da diversi dispositivi.

Le primitive del modello dell'ipertesto sono:

- Viste
- Aree
- Pagine
- Unit
- Link

5.3.2.1 Aree e Pagine

La figura mostra un esempio di organizzazione di pagine e aree in una vista. Una pagina è un contenitore di una o più *unit* di contenuto mostrate all'utente contemporaneamente mentre le aree sono insiemi di pagine logicamente omogenee.

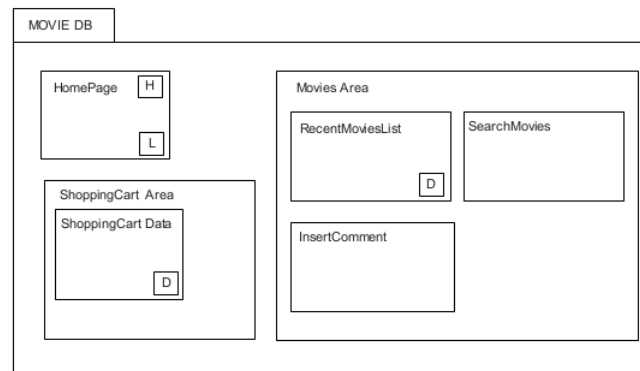


Figure 18: Modularizzazione di viste basata su aree e pagine

Nell'esempio la vista è composta da un homepage, che è la prima pagina a cui l'utente accede quando entra nell'applicazione; essa comprende differenti aree (*ShoppingCart*, *HomePage*, *Movies Area*). Le pagine sono caratterizzate da alcune proprietà che possono evidenziarne l'importanza. In particolare le pagine all'interno di un'area o di una vista possono essere di tre tipi:

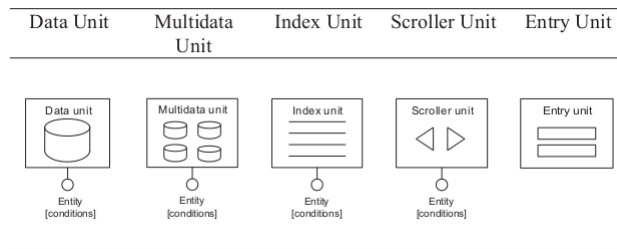
- home page (denotata dalla "h") è la pagina di default dopo che l'utente si logga e deve essere unica;
- la default page (denotata da una "d") è quella presentata quando si accede ad una particolare area e deve essere unica all'interno dell'area;
- landmark page (denotata da una "l") è raggiungibile da tutte le altre pagine.

5.3.2.2 Viste

WebML permette per una stessa applicazione la definizione di più modalità di navigazione dei dati tramite il concetto di vista; essa è quindi un modulo che definisce una modalità di accesso ai dati. Ad esempio tramite più viste è possibile distinguere l'utente non registrato dall'amministratore, fornendo una visione del sito completamente diversa anche se basata sugli stessi dati. Le viste possono inoltre venire utilizzate per definire una modalità di navigazione specificatamente pensata per un client, come un normale browser su PC, smartphone o browser a sintesi vocale.

5.3.2.3 Composizione delle pagine e Content Units

Una *content unit* in WebML è l'elemento atomico per la pubblicazione dell'informazione. Le *unit* rappresentano una o più istanze di un'entità

Table 1: Le cinque *content units* predefinite in WebML

dello schema strutturale, tipicamente selezionate attraverso query sugli attributi dell'entità o sulle relazioni. WebML prevede cinque *content unit* di default:

- la *Data Unit* pubblica l'informazione circa la singola istanza un entità;
- la *Multidata Unit* è simile alla precedente ma pubblica l'informazione circa un insieme di istanze;
- la *Index Unit* mostra una lista di un insieme di istanze di entità abilitando la selezione di una di esse;
- la *Scroller Unit* permette di definire il browsing in un insieme di oggetti: visualizza link al primo, al precedente, al successivo e all'ultimo oggetto dell'insieme;
- la *Entry Unit* non espone il contenuto a partire da elementi dello schema dei dati, ma pubblica un form per acquisire valori di input dall'utente.

Data, *Multidata*, *Index*, e la *Scroller Unit* prevedono un'entità sorgente e un selettore. L'entità sorgente è il nome dell'entità dalla quale il contenuto della *unit* è recuperato; il selettore è il predicato usato per determinare gli attuali oggetti dell'entità sorgente che contribuiscono a popolare il contenuto della unit. Ogni *unit* può avere parametri di input e output: i parametri in input sono necessari per calcolare la *unit* e sono i parametri richiesti dal selettore; i parametri in output possono essere utilizzati per la computazione di una o più *unit* che dipendono dalla *unit* corrente.

5.3.2.4 *Link*

I link tracciano delle connessioni orientate tra due *unit*, una sorgente e una di destinazione, la cui presentazione corrisponde ad ancore o bottoni di tipo *submit*. Essi permettono all'utente di navigare tra i diversi nodi dell'ipertesto, trasportano informazione di contesto e come effetto

collaterale attivano una computazione. Il contesto è trasportato dai link attraverso l'uso di parametri su essi definiti: un parametro sui link ha un nome e il suo contenuto è un attributo della *unit* sorgente del link.

I link possono essere definiti *automatici* quando passano il contesto alla *unit* di destinazione immediatamente dopo la visualizzazione della *unit* sorgente, senza la necessità di un intervento dell'utente; successivamente, l'utente può cambiare il contesto passato, scegliendo un oggetto differente tramite l'ancora che rappresenta il link.

In alcuni casi i link sono usati solo per passare informazione contestuale, in questo caso sono chiamati *link di trasporto* (si indicano con una linea tratteggiata) per evidenziare che tali link abilitano solo il passaggio e non l'interazione; in questo caso l'utente non può cambiare il contesto trasportato dal link poiché esso non è visualizzato tramite un ancora.

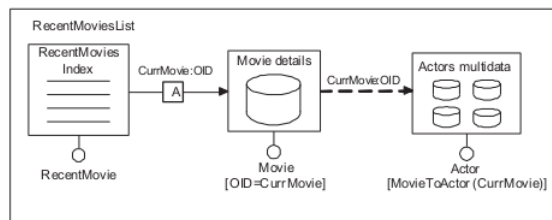


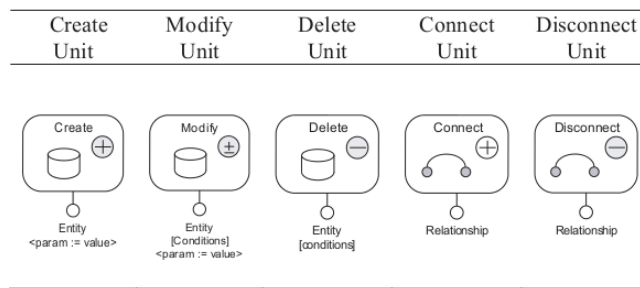
Figure 19: Esempi di link automatico e di trasporto

5.3.2.5 Parametri globali

In alcuni casi l'informazione contestuale non è trasferita da punto a punto durante la navigazione ma può essere impostata come globalmente disponibile a tutte le pagine di una vista. Questo è possibile attraverso i parametri globali che astraggono la nozione di dati persistenti di sessione. I parametri possono essere definiti attraverso la *set unit* e consumati attraverso la *get unit*.

5.3.2.6 Operation Units

Le *operation units* sono componenti che si occupano dell'esecuzione della logica di business, non pubblicano contenuto e per questo motivo sono posizionate all'esterno delle pagine. Come le *content unit* le *operation unit* hanno una fonte (entità o relazione) e un selettore, possono ricevere parametri dai link in input e possono fornire valori che saranno usati dai parametri dei link di output. Uno dei link di input è il link di attivazione (quello scatenato dall'interazione dell'utente) mentre gli altri trasportano solo informazione contestuale e parametri, come ad esempio l'identificatore di qualche oggetto coinvolto nell'operazione.

Table 2: *Operation Units* di WebML

I link di output possono essere di trasporto o di tipo OK/KO: il primo gestisce la navigazione nel caso in cui l'operazione abbia avuto successo, il secondo nel caso in cui l'operazione fallisca. Infine il risultato dell'esecuzione di un'operazione può essere mostrato in pagina attraverso appropriate *content unit* come ad esempio una *data* o *multidata unit* definita su gli oggetti aggiornati dall'operazione.

Un ipertesto WebML quindi può essere descritto essenzialmente come un grafo composto da componenti parametrici, connesso da link, in cui alcuni componenti stessi pubblicano il contenuto e sono inclusi all'interno delle pagine, altri compiono le azioni che fanno parte della logica di business e sono innescati dai link uscenti dalle pagine.

5.3.3 Il modello di presentazione

Il modello di presentazione è costituito da molteplici *template* di *layout* che descrivono l'aspetto visuale dell'intera applicazione. Ogni *template* può essere associato ad un componente del modello dell'ipertesto: esistono quindi *template* specifici per le aree, pagine, unit e per tutti gli altri elementi che contribuiscono alla generazione dell'aspetto finale dell'applicazione.

APPLICAZIONI LEGACY 3270: SARA

Nel precedente capitolo è stato introdotto lo strumento di modernizzazione ma si è detto poco circa le applicazioni che tale strumento permette di modernizzare. Nei prossimi paragrafi si parlerà della famiglia di applicazioni legacy CICS sviluppate su mainframe e accessibili tramite terminale 3270, se ne effettuerà un'analisi in termini di usabilità, e si spiegherà perché la modernizzazione si rende necessaria. L'analisi e la seguente modernizzazione verrà effettuata a titolo esemplificativo su una particolare applicazione: SARA.

6.1 SARA LEGACY

SARA è un'applicazione utilizzata in ambito bancario per la gestione degli assegni tra le diverse filiali. E' scritta in Cobol, è in esecuzione su mainframe in ambiente CICS ed è accessibile mediante emulatore di terminale 3270.

6.1.1 Struttura dell'applicazione

Il 3270 è un terminale definito di tipo *block-oriented* o *block-mode* poichè comunica con il suo host in blocchi di dati, in opposizione con il terminale orientato ai caratteri che comunica con il suo host un carattere alla volta; a causa del colore del testo questi terminali sono noti informalmente come green screen. In una tipica applicazione CICS l'host manda al terminale una maschera preformattata contenente sia dati statici che dinamici: nel data stream quindi sia il testo che i comandi sono interlacciati permettendo che l'intera schermata sia stampata a video come una singola operazione di output [43]. In questi dispositivi il concetto di formattazione permette di dividere lo schermo in gruppi di celle di caratteri contigui per i quali possono essere impostati alcuni attributi come colore, evidenziazione, set di caratteri e protezione dalla modifica. Un attributo occupa una locazione fisica sullo schermo che determina anche l'inizio e la fine di un campo, inteso come la sottosezione separata dello schermo che è indirizzabile. Il terminale, utilizzando una tecnica conosciuta come *read modified*, può anche comunicare con il mainframe inviando solo i campi che sono stati effettivamente modificati tralasciando quelli rimasti inalterati e i campi statici: questa tecnica migliora il *throughput* della Central Processing Unit (CPU) e minimizza la quantità di dati trasmessi. In principio infatti il 3270 era stato pensato per

le connessione con i mainframe da postazioni remote utilizzando la tecnologia disponibile agli inizi degli anni '70 e i principali obiettivi erano minimizzare il carico di dati trasmessi e la diminuzione della frequenza di interrupt al mainframe. I terminali block-oriented possono anche apparire all'utente più scattanti, specialmente nel caso di connessioni lente, poiché la modifica all'interno di un campo è fatta localmente, cioè nel terminale stesso, anziché dipendere dall'*echoing* dell'host.

Lo sviluppo software oggi è in molti modi ritornato all'approccio del 3270, quando cioè tutte funzionalità dell'applicazione erano centralizzate. Con l'avvento del PC l'idea era di invocare il sistema centrale solo quando non era assolutamente possibile fare diversamente e tutto il lavoro veniva svolto localmente dalla singola macchina; oggi, nell'era del web, l'applicazione sta tornando ad essere di nuovo fortemente centralizzata e solo le funzionalità tecniche sono distribuite sul PC.

6.1.1.1 *Le mappe BMS*

Basic Map Supporting (BMS) è un'interfaccia tra le applicazioni su CICS e i terminali; essa permette di separare le questioni relative alla visualizzazione del contenuto dalla quelle inerenti alla pura programmazione dell'applicazione. In particolare BMS interpreta i comandi di output dei programmi in maniera generalizzata e indipendente dal dispositivo e genera *data stream* specifici dipendenti dal dispositivo: si occupa quindi di trasformare il *data stream* in arrivo in un formato accettabile per l'applicazione. Le mappe sono utilizzate da BMS per gestire la formattazione dei campi da visualizzare nell'interfaccia testuale.

6.1.2 *Logica di navigazione*

In generale le azioni che l'utente può compiere sull'interfaccia del terminale 3270 sono estremamente vincolate; è permesso il solo editing dei campi di testo mentre la navigazione è gestita con opportuni comandi che si possono considerare standard: ad esempio il tasto funzione ENTER conferma, PF7 e PF8 procedono rispettivamente indietro e avanti in una eventuale paginazione, F1 serve per la consultazione delle funzioni di help e via dicendo.

L'applicazione SARA è caratterizzata da una logica di navigazione abbastanza semplice ed è riassunta nei diagrammi degli stati riportati in figura. Tale logica è stata dedotta grazie ad un lavoro di reverse engineering dovuto al fatto che la documentazione legacy non disponeva delle informazioni necessarie. Per richiamare l'applicazione si avvia l'emulatore di terminale, si digita il nome dell'applicazione (SA01), si pulisce la schermata dal testo (CTRL+C) e si seleziona un particolare ambiente CICS (in

questo caso il D). A questo punto ci si trova davanti alla schermata di login dell'applicazione. Questa avviene mediante tre step successivi:

1. Inserimento utente, password e istituto (ENTER);
2. L'applicazione comunica quale connessione e quale terminale utilizzare; l'utente deve compilare i campi corrispondenti e confermare la scelta (PF4);
3. L'applicazione comunica l'avvenuta connessione e l'utente può effettivamente loggarsi confermando con il comando ENTER.

A questo punto ci si troverà davanti ad un menù che elenca le possibili funzioni. Per la modernizzazione di **SARA** ci si è concentrati sulle cinque funzioni più utilizzate. Per accedervi è necessario inserire in un campo della mappa il corrispondente codice della procedura, in questo caso: INGG, IANG, IATR, IRIM e ISMA. La conferma è effettuata sempre mediante tasto ENTER.

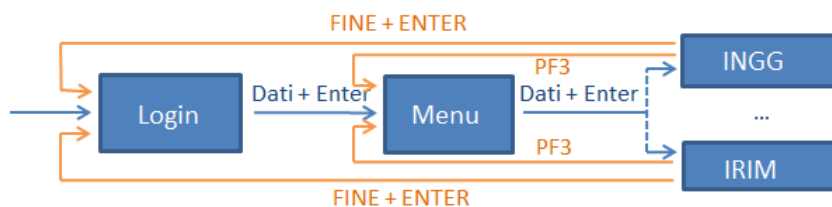


Figure 20: Logica di navigazione dell'applicazione: login e selezione procedura

Le cinque funzioni sono piuttosto modulari. Per ognuna di esse si accede ad una maschera con dei campi preformattati la quale permette di specificare alcuni parametri per la ricerca degli assegni. Nel caso in cui tale ricerca abbia successo viene proposta una lista di risultati; per scorrere tra essi si utilizzano i tasti funzione PF7 (avanti) e PF8 (indietro). Infine è possibile visualizzare il dettaglio di un assegno inserendo la lettera "s" sulla riga corrispondente all'elemento e confermando con ENTER. Nel caso in cui la ricerca produca come risultato un solo assegno ne è mostrato direttamente il dettaglio.

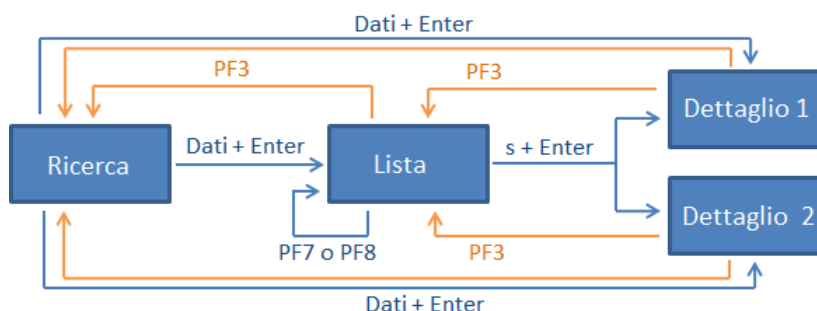


Figure 21: Logica di navigazione dell'applicazione: ricerca, lista e dettaglio

I problemi legati a questo tipo di applicazione riguardano principalmente l'usabilità e l'accessibilità più che l'efficienza in termini di prestazioni tecnologiche. Al fine di capire e individuare tali problematiche è stata effettuata un'approfondita analisi dell'usabilità, il cui scopo è individuare i punti critici su cui è necessario intervenire.

6.2 ANALISI EURISTICA

La valutazione euristica è una tecnica sistematica di ispezione in cui alcuni valutatori esaminano il sistema e giudicano quanto è adeguato, individuando i problemi di usabilità che esso presenta. Nello studio dell'usabilità si va ad osservare il coinvolgimento dei fattori umani nel calcolo e nella comunicazione digitale [14]. L'analisi effettuata sul sistema tiene conto delle cinque dimensioni dell'usabilità previste da Nielsen[32]:

- apprendibilità: facilità nell'apprendere il comportamento del sistema;
- efficienza d'uso: il sistema deve essere efficiente da usare, cosicché quando l'utente lo ha appreso, sia possibile un alto livello di produttività;
- flessibilità: il sistema deve essere flessibile rispetto alla molteplicità di modalità di interazione oltre che facile da ricordare;
- robustezza: il sistema deve indurre ad un basso tasso di problemi, facili da correggere e con un minimo impatto sul costo;
- soddisfazione: il sistema deve essere gratificante da usare.

6.2.1 Problemi riscontrati

I dettagli circa l'organizzazione e l'esecuzione dell'analisi euristica sono riportati nell'[Appendix A](#). Di seguito sono presentati solo alcuni esempi dei problemi riscontrati durante l'utilizzo dell'applicazione:

- I messaggi di stato non hanno caratteristiche che richiamano l'attenzione (come il colore) ed è facile trascurarli confondendoli con il testo del testo;
- Le associazioni relative ai colori non sempre sono significative e coerenti. In particolare i colori utilizzati sono :
 - blu: nome mappa, spiegazione comandi, data, nome utente, filiale
 - verde: per i campi
 - bianco: il resto

- Non è chiaro come ricorrere ad una funzione di *help* sia per le funzionalità dell'applicazione che per i comandi;
- Il codice degli stati non sono descrittivi e conducono l'utente all'errore (ad esempio l'assegno circolare è identificato dal codice 0 mentre quello bancario dal codice 1);
- Ogni volta che l'utente deve cercare la descrizione di un codice deve richiamare l'apposita funzione;
- Non è chiaro in che modo debba essere effettuata la selezione di un elemento della lista;
- La formattazione non aiuta la comprensione visuale: secondo la legge della Gestalt¹ della vicinanza sono possibili inferenze scorrette;

Di seguito si riporta una categorizzazione dei risultati ottenuti durante l'analisi euristica dell'applicazione legacy.

6.2.1.1 *Problemi per categoria*

- layout: sono i problemi che compromettono la struttura grafica e l'architettura del sistema;
- navigazione: sono i problemi riguardanti l'orientamento e la contestualizzazione tra le sezioni del sistema;
- contenuto: sono i problemi legati alla carenza, alla non completezza e alla difficoltà di raggiungimento delle informazioni;
- funzionalità mancanti o errate: sono i problemi dovuti alla mancanza di funzioni che agevolerebbero l'utente nell'utilizzo del sistema.

Come si evince dal grafico la maggior parte dei problemi dell'applicazione riguardano le funzionalità mancate o erranti. L'applicazione risulta spesso difficile da usare a causa dei vincoli imposti dalla struttura delle applicazioni 3270, in particolare l'impossibilità di gestire in maniera elastica l'interfaccia. Un esempio eclatante consiste nella corrispondenza codice-descrizione. Si supponga che un utente voglia ricercare un certo tipo di assegno negoziato per un lasso di tempo. Per ragioni di spazio il campo tipo deve essere rappresentato in una forma sintetica, un codice, che non è però assolutamente indicativo e descrittivo per l'utente e la cui memorizzazione diventa quindi molto difficile e cognitivamente onerosa.

¹ La psicologia della Gestalt, detta anche psicologia della forma, è una corrente psicologica riguardante la percezione e l'esperienza che nacque e si sviluppò agli inizi del XX secolo in Germania. La legge della vicinanza afferma che gli elementi del campo percettivo vengono uniti in forme con tanta maggiore coesione quanto minore è la distanza tra di loro [14].

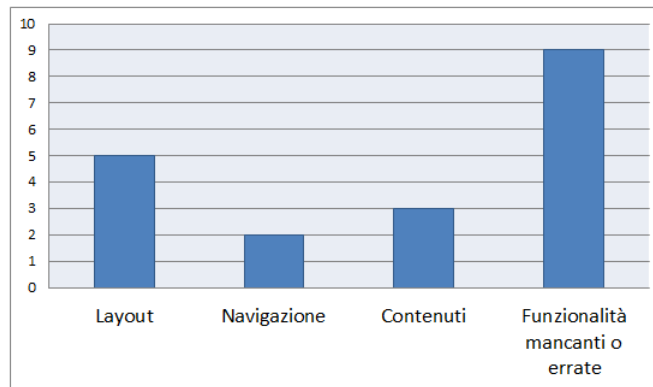


Figure 22: Distribuzione problemi usabilità per categoria

Per le stesse ragioni anche il layout dell'applicazione può considerarsi critico: l'utente non è agevolato nell'interazione con l'applicazione poiché non possono essere utilizzati strumenti finalizzati a questo scopo: si pensi al widget del calendario nel caso di selezioni date, checkbox, o menù a tendina che sono di solito presenti nelle applicazioni web.

6.2.1.2 *Problemi per principio di Nielsen*

Nielsen ha individuato dieci principi all'interno dei quali possono esser inseriti i problemi di usabilità. Essi sono[32]:

1. Far vedere lo stato del sistema
2. Adeguare il sistema al mondo reale
3. Controllo dell'utente e libertà
4. Assicurare consistenza
5. Riconoscimento piuttosto che aiuto della memoria
6. Assicurare flessibilità ed efficienza d'uso
7. Visualizzare tutte e sole le informazioni necessarie
8. Prevenire gli errori
9. Permettere all'utente di correggere gli errori e non solo rilevarli
10. Fornire aiuto e documentazione

Nella figura è mostrata la distribuzione dei problemi riscontrati sull'applicazione rispetto a tali principi: risulta chiaro che gli errori più gravi riguardano la mancata assistenza all'utente. Le applicazioni 3270 sono state pensate

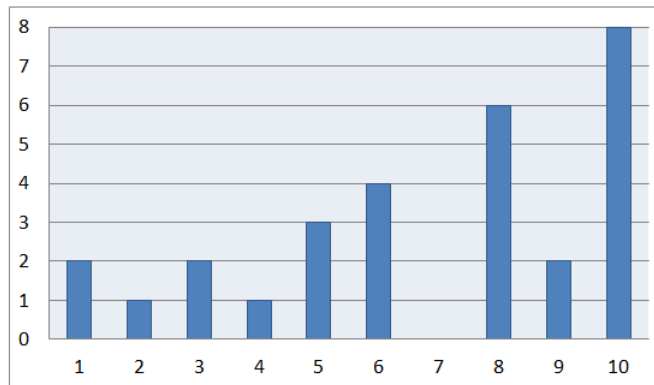


Figure 23: Distribuzione problemi usabilità per problema

quando ancora non si parlava di usabilità, e quindi prevedevano un apposito addestramento per l'utilizzo. Per questo motivo l'utente inesperto cade spesso in errore e non riesce a trovare la documentazione anche quando tecnicamente disponibile. Manca inoltre un qualsiasi aiuto per la prevenzione all'errore. Anche l'utente che sa utilizzare l'interfaccia non è comunque agevolato nel suo utilizzo: l'applicazione non permette libertà di interazione, non esonera dallo sforzo cognitivo dovuto alla memorizzazione di informazioni (come il significato dei codici), non è flessibile (non sono previste alternative o scorciatoie per l'esecuzione dello stesso compito) e di conseguenza non permette l'efficienza d'uso.

6.2.1.3 Problemi per tipologia e sezione di intervento

I problemi dell'applicazione sono:

- percettivi: gli errori di percezione riguardano in egual misura grafica, programmazione e architettura. Essi dipendono dal fatto che l'utente durante l'atto percettivo e quindi ancora prima di qualsiasi elaborazione cognitiva, valuta erroneamente un qualche aspetto dell'interfaccia. Esempi di problemi percettivi riguardano l'errata interpretazione di un campo a causa del colore: si è visto infatti che i colori dell'applicazione a volte sono usati in maniera incoerente. Come spiega la legge della Gestalt della somiglianza², questo può portare alla generazione di inferenze scorrette: l'utente potrebbe pensare che i campi funzione e utente abbiano una qualche relazione semantica poiché entrambi colorati di verde mentre nella realtà tra i due campi non esiste alcuna relazione.

² La legge della somiglianza afferma che gli elementi vengono uniti in forme con tanta maggior coesione quanto maggiore è la loro somiglianza [14].

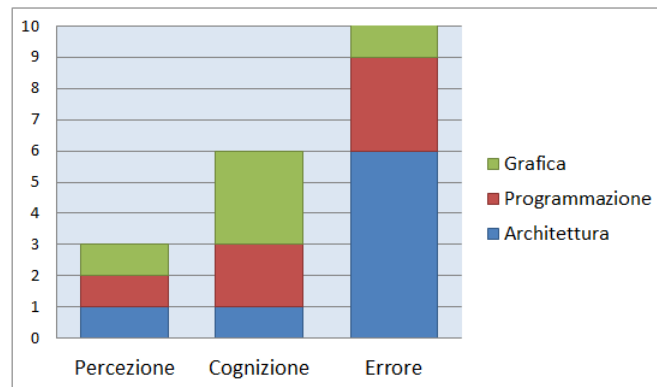


Figure 24: Distribuzione problemi usabilità per tipologia e sezione d'intervento

- **cognitivi:** in questo caso i problemi sono principalmente legati all'aspetto grafico dell'applicazione e in parte anche a problemi di programmazione. I problemi cognitivi sono legati allo sforzo eccessivo richiesto all'utente per l'elaborazione e la realizzazione di normali compiti. Un esempio è rappresentato dal fatto che all'utente è richiesto di imparare a memoria tutti i comandi per poter agire in maniera efficace ed efficiente.
- **errore:** i problemi in questo sono legati soprattutto alla grafica.

6.2.2 Risultato dell'analisi

L'analisi euristica ha permesso di individuare i principali problemi di usabilità presenti nell'applicazione. Essi sono numerosi, spesso gravi e questo pregiudica la corretta interazione con l'utente.

I problemi nascono dal fatto che l'applicazione è stata sviluppata con lo stesso layout grafico, architettura e tecnologie disponibili negli anni '70, un periodo in cui le limitazioni tecnologiche non permettevano di concentrarsi su problematiche relative all'usabilità. Analizzando i risultati e confrontando i grafici ottenuti, si può vedere come la distribuzione dei problemi per categoria sia abbastanza uniforme per quanto riguarda layout, navigazione e contenuti, mentre i problemi dovuti a funzionalità mancanti o errate siano ancora più consistenti. Infatti i principi per il controllo e la prevenzione dell'errore, per l'efficienza e la flessibilità sembrano essere stati completamente ignorati. Il risultato è un'esperienza d'uso priva di difficoltà, cognitivamente impegnativa e generalmente frustrante.

L'applicazione può dunque considerarsi:

- Efficace: in quanto l'utente riesce in generale a portare a termine i compiti assegnati, anche se con un notevole sforzo cognitivo e alta probabilità di errore in caso di utente inesperto.
- Non efficiente: è necessario uno sforzo considerevole per prendere familiarità con l'applicazione. Inoltre, anche una volta che si è imparato l'utilizzo, molte attività restano lunghe e noiose. Si pensi all'individuazione della descrizione dei codici: o la si ricorda o ogni volta è necessario richiamare un'apposita funzione di traduzione (di cui a sua volta è necessario ricordare il nome per poter essere invocata nel campo di procedura).
- Non soddisfacente: l'utente oggi è abituato a relazionarsi con interfacce piacevoli da usare e gratificanti, cosa che un'interfaccia testuale non può sicuramente offrire.

6.2.2.1 Applicazioni 3270 e il modello di Hutchins, Holland e Norman

Al fine di individuare il vero problema di [SARA Legacy](#), ma in generale di tutte le applicazioni CICS caratterizzate da interfaccia testuale come quelle accessibili da terminale 3270, è utile considerare il modello di Hutchins, Holland e Norman, un modello dell'azione nell'interazione uomo-computer. Tale modello deriva dalla teoria del controllo delle azioni di Norman e Shallice [22] e tenta di rendere conto delle fasi previste durante l'utilizzo di strumenti, nonché delle difficoltà di passaggio da una fase all'altra dove tale passaggio è visto come distanza. Le fasi del modello sono sette, una per gli obiettivi, tre per l'esecuzione e tre per la valutazione[16].

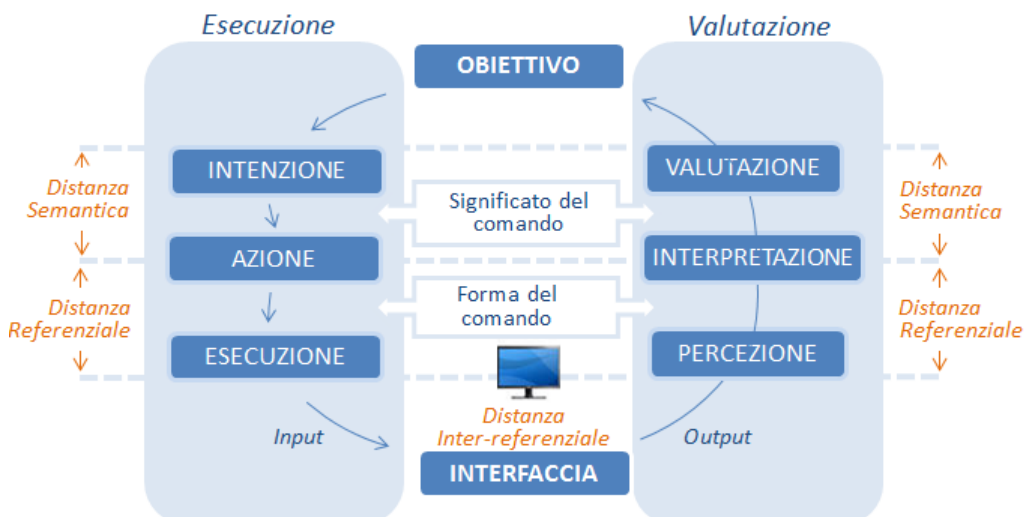


Figure 25: Modello di Hutchins, Holland e Norman

Per contestualizzare il modello si prenda un possibile esempio di interazione con l'applicazione [SARA](#). Si immagini di avere la necessità di cercare lo stato degli assegni tratti di un certo periodo di tempo: a questo punto è stato definito l'*obiettivo*, cioè il recupero di informazioni mediante l'applicazione; tale obiettivo deve comunque essere tradotto nelle *intenzioni* che definiscono le azioni appropriate nel mondo: prendere il mouse, cliccare sull'icona dell'emulatore, digitare gli opportuni comandi e via dicendo. Si deve anche specificare come muovere la mia mano, come muovere il mouse correttamente e come spingere il suo bottone controllando il puntatore sullo schermo, come effettuare una pressione sui tasti e via dicendo. L'obiettivo deve quindi essere tradotto in specifiche intenzioni, che a loro volta devono essere tradotte in specifiche sequenze di azioni, azioni che controllano gli apparati percettivo-motori. E' da notare inoltre che l'obiettivo può essere soddisfatto attraverso altre sequenze di azioni e intenzioni: ad esempio è possibile inserire il testo nei campi di ricerca spostandosi da uno all'altro con il mouse o con il tasto di tabulazione.

Le distanze tra le varie fasi del modello dell'azione si riferiscono proprio al modo con cui si realizza più o meno prontamente il passaggio da una fase ad un'altra. Più specificamente la *distanza semantica* per ciò che concerne l'azione riguarda la relazione fra le intenzioni dell'utente ed il significato dei comandi che è possibile eseguire sull'interfaccia; tale distanza è funzione della facilità con cui l'interfaccia fornisce mezzi e strumenti per esprimere le intenzioni dell'utente. Sul lato della valutazione la distanza semantica si riferisce all'elaborazione che è richiesta all'utente per determinare se le condizioni di soddisfacimento delle proprie intenzioni sono state realizzate. Le applicazioni 3270 sono caratterizzate da elevate distanze semantiche sia sul lato dell'esecuzione che sul lato della valutazione. Questo comporta che l'intenzione di avere, per esempio, la descrizione corrispondente ad un certo codice, venga tradotta in una serie articolata di comandi dalla sintassi rigida che devono essere recuperati dalla memoria a lungo termine ed eseguiti. Il risultato poi può essere valutato solo operando ulteriori trasformazioni mentali sugli effetti realmente prodotti o eseguendo la serie di operazioni necessaria per ottenere il risultato finale e solo allora valutarne il risultato.

Un esempio eclatante di distanza semantica è riscontrabile nei comandi che l'applicazione mette a disposizione. Oggi infatti l'utente è abituato a interfacce grafiche intuitive e che richiamano metafore prese dal mondo reale, in modo da immergere l'esperienza in un ambiente familiare. I tasti funzione dell'applicazione, che identificano i comandi che essa offre, non sono per niente descrittivi o semanticamente chiari. Ad esempio la distanza semantica tra l'intenzione di scorrere tra i risultati della lista e l'azione che permette di compiere tale intenzione, cioè la digitazione del tasto PF8, è molto elevata e non è in alcun modo deducibile o inferibile dal contesto a meno di un'opportuna descrizione del comando.

Vi è infine la distanza *inter-referenziale*, che riguarda la relazione fra le forme di input e quelle di output presenti nel corso dell'interazione con l'applicazione. Questa distanza è massima quando le due forme sono completamente distinte. Nelle interfacce grafiche un'icona, ad esempio, rappresenta un output del sistema ma è nello stesso tempo oggetto di possibili azioni dell'utente ed i risultati di queste azioni costituiscono a loro volta nuovi input per il sistema. Nelle interfacce testuali come quella delle applicazioni 3270 input e l'output non solo sono operativamente differenti ma spesso hanno un lessico differente.

Ci si potrebbe chiedere come mai oggi ancora molti utenti siano legati ad interfacce legacy come quelle a linee di comando, a tal punto da preferirle alle interfacce grafiche. La risposta viene fornita dal processo di automizzazione del controllo dei processi cognitivi [20]. In base a tale processo con la pratica si riduce progressivamente il numero di informazioni sotto controllo attentivo necessarie ad attivare le conoscenze rilevanti per il compito in atto. Attraverso la pratica questi utenti sviluppano processi di attivazione di conoscenza che fanno sì che le loro intenzioni relative a specifici compiti vengano formulate dapprima nei termini stessi dei comandi possibili sull'interfaccia (scorrere tra i risultati di una lista significa automaticamente interagire con i tasti funzione PF7 e PF8) ed in seguito che la sola definizione dell'obiettivo sia sufficiente ad evocare il *pattern motorio* adeguato.

Si potrebbe erroneamente pensare che la differenza tra uno strumento che riduce le distanze cognitive e uno che le mantiene sia solo una questione di abitudine. Tale conclusione non considera le opportunità che uno strumento può offrire. Infatti le distanze cognitive sono una misura dei vincoli di uno strumento e per rendersi conto della differenza che esiste tra utenti che operano con strumenti a distanza cognitiva ridotta rispetto a quelli che operano con strumenti a distanza cognitiva elevata, è sufficiente ascoltare il gergo che si sviluppa nell'interazione. Più questo è vicino alla sintassi, quindi ai vincoli dello strumento meno svolge la sua funzione di strumento cognitivo[33]. Inoltre la rapidità con cui degrada l'abilità di un utente quando ci si allontana dalle attività routinarie è decisamente superiore per gli strumenti con distanze elevate. Una ulteriore conferma è data dalla difficoltà di recuperare l'abilità nell'uso di uno strumento se non lo si è usato per lungo tempo: laddove la distanza semantica è considerevole, il riappropriarsi dell'uso dello strumento e quindi il ricordare sintassi e sequenze di azioni, diventa un compito estremamente oneroso.

6.2.3 Conclusioni

Per tutti i motivi fin qui visti le applicazioni 3270 risultano essere ottime candidate in termini di modernizzazione. Tali applicazioni risultano an-

cora efficaci ed efficienti da un punto di vista tecnologico e computazionale ma hanno la necessità di una completa rivisitazione dell'interfaccia utente in termini di accessibilità e usabilità. In particolare le questioni critiche da risolvere riguardano:

- diminuzione dello sforzo cognitivo legato alla memorizzazione di codici, funzioni e in generale tutte quelle informazioni che non sono di per sé esplicative;
- maggiori spiegazioni relative alle azioni che l'utente può compiere e una migliore individuazione dei comandi: in generale le azioni riguardano sia la logica di navigazione (come spostarsi da una pagina all'altra o consultare l'help di errore) che le funzioni specifiche dell'applicazione (quali operazioni è possibile effettuare sulla lista dei risultati?)
- chiarire lo stato del sistema;
- pervenienza, correzione dell'errore ed eventuale *recovery* da una situazione di errore.

In sostanza è necessario intervenire per la riduzione drastica delle distanze semantiche sia in fase di valutazione che di esecuzione e della distanza inter-referenziale.

Poiché le sezioni di intervento riguardano quasi esclusivamente l'interfaccia grafica dell'applicazione, e non interessa in alcun modo intervenire sugli aspetti funzionali, la tecnica di modernizzazione più appropriata risulta essere quella di tipo black-box. Inoltre poiché l'applicazione è caratterizzata da un'interfaccia testuale la conclusione più logica è quella di passare ad un'interfaccia grafica: la modernizzazione appropriata risulta quindi essere il *revamping* dell'applicazione attraverso opportune tecniche di *screen scraping*.

Nel [Chapter 5](#) si è introdotto WebRatio, uno strumento per la creazione di applicazioni web basato sul linguaggio Web Modelling Language ([WebML](#)); in questo senso esso non può essere considerato un vero e proprio strumento di modernizzazione, in quanto la creazione di un applicativo web rappresenterebbe piuttosto un esempio di sostituzione. Nei prossimi paragrafi si vedrà come, grazie all'uso di opportune librerie, sia possibile rendere WebRatio uno strumento specializzato nel revamping delle applicazioni CICS appena discusse.

7.1 LA W₄BMS

La *w₄bms* è un insieme di librerie che integrate con WebRatio implementano le tecniche di screen scraping che permettono di modernizzare applicazioni esistenti attraverso il mapping delle vecchie interfacce testuali in nuove e moderne interfacce grafiche.



Figure 26: Screen scraping di applicazione legacy grazie alla w₄bms

La combinazione WebRatio-w₄bms è caratterizzata dalle seguenti proprietà:

- **flessibilità:** è possibile mantenere il flusso operativo originario o intervenire per apportare delle modifiche. Si può considerare a titolo di esempio il controllo dell'errore: si è visto che per alcuni campi dell'applicazione l'input non è validato in maniera soddisfacente. Per questi campi può essere aggiunto un controllo preventivo nell'interfaccia web, mentre gli altri possono essere lasciati invariati. Un altro esempio è nella logica di navigazione: tale procedura può essere semplificata eliminando quegli step che sono facilmente automatizzabili e che non richiedono l'interazione con l'utente. Questo significa che è possibile la coesistenza tra la vecchia applicazione 3270 e la nuova versione web-based che possono tranquillamente rimanere attive in modo contemporaneo.

- **automatismo:** le componenti software necessarie vengono generate automaticamente partendo dalla definizione originaria della schermata, cioè delle mappe [BMS](#)
- **integrazione:** è uno strumento di facile utilizzo per lo sviluppatore, basato su ambiente di sviluppo Eclipse. Essendo uno strumento di modellazione grafico è adatto anche ai non addetti ai lavori. Questa caratteristica è molto importante, poiché permette anche al personale abituato alle tecnologie legacy di interfacciarsi con il mondo web.

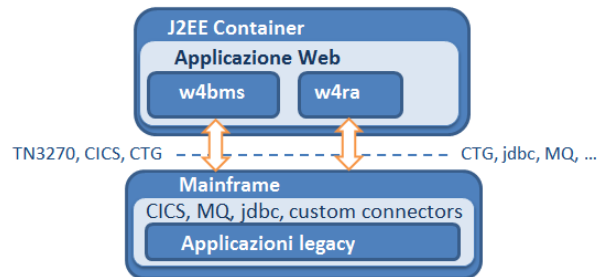


Figure 27: Macroarchitettura della w4bms

- **soddisfazione dell'utente ed efficienza:** con uno sforzo e un costo minimo w4bms fa mapping 1:1 delle pagine 3270 permettendo l'integrazione di elementi grafici intuitivi come *combo box*, *check-box* e *radio button*, che permettono una migliore usabilità.
- **accessibilità:** la predisposizione nativa dell'ambiente di sviluppo e l'utilizzo di template dedicati ai dispositivi mobile, rende immediato l'utilizzo dei vecchi servizi sui dispositivi più recenti come smartphone e tablet oltre che ai dispositivi di supporto per chi affetto da disabilità cognitive e percettive come strumenti di sintesi vocale e screen reader.

7.2 LE UNIT CUSTOMIZZATE

La *w4bms* consiste di una serie di unit customizzate che permettono l'interazione con l'applicazione legacy:



Figure 28: Notazione grafica delle custom unit della w4bms

- *connect unit*: gestisce la connessione con l'applicazione CICS;
- *identifyMap unit*: identifica la mappa corrente;
- *readMap unit*: legge la mappa corrente popolandolo un entità ad essa associata con i rispettivi campi;
- *sendCommand unit*: simula l'invio di un comando (PF3, PF4, ENTER ecc)
- *writeMap unit*: scrive sui campi della mappa corrente.

A titolo esemplificativo si mostrerà l'*IdentifyMap Unit*, addetta alla gestione della scrittura della mappa.

7.2.1 *IdentifyMap Unit*

Esplorando tra i package disponibili su WebRatio si trovano le *unit custom*. La cartella comprende:

- una cartella con le immagini che descrivono la unit nel modello concettuale;
- una cartella che definisce i template di input e output logico;
- un file eXtensible Markup Language (XML) che ne descrive la struttura.

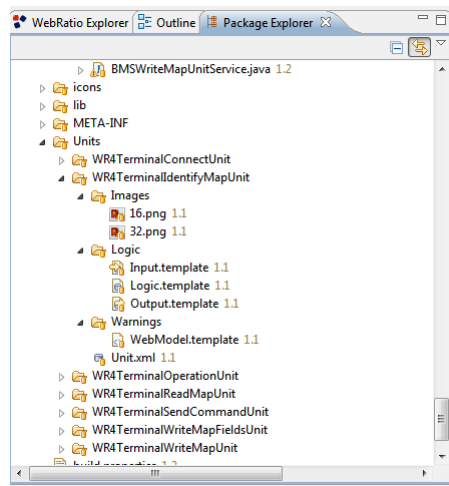


Figure 29: La struttura dei files di una *custom unit* all'interno dell'albero di WebRatio

7.2.1.1 *Logica*

L'input e l'output dinamico prodotto dalle istanze delle unità richiede una coppia di templates scritti in *Groovy* chiamati `Input.template` e `Output.template`

memorizzati nella sotto cartella Logic della cartella della unit. Lo scopo di questi template è quello di produrre due frammenti XML che conterranno rispettivamente i parametri di input e di output della unit.

Il file `Output.template` specifica i parametri di output: ognuno di essi ha un nome (`name`), un tipo (`tipo`) una etichetta (`label`) e se il parametro è obbligatorio o meno. L'attributo `tipo` è molto importante perché, se usato correttamente, permette di abilitare la funzione di agganciamento automatico quando si passano le informazioni su un link in uscita dalla unit personalizzata: la logica utilizzata è tale da agganciare due parametri se hanno lo stesso tipo di dato. Nel caso della `IdentifyMap Unit` ci si aspetta che essa produca obbligatoriamente in uscita il nome di una mappa la cui etichetta è `Map Name`

Listing 1: `Output.template`

```
#?delimiters <%,%>,<%=,%>
<% setXMLOutput() %>
  <OutputParameters>
    <OutputParameter name="mapName" type="" label="Map Name"
      mandatory="true"/>
  </OutputParameters>
```

Il file di input è assolutamente speculare al file di input e valgono le stesse considerazioni già fatte.

Il file `Logic.template` specifica le proprietà statiche dell'unità. Il file contiene un elemento `<Descriptor>` che descrive il comportamento e l'aspetto delle diverse proprietà. Gli elementi figli di `<Descriptor>` contengono i valori delle proprietà disponibili nel quadro proprietà dell'unità. Tipicamente le proprietà sono già state definite nella pagina dei sottoelementi del file `unit.xml`, il seguente codice non fa altro che serializzare le proprietà definite.

Listing 2: `Logic.template`

```
#?delimiters <%,%>,<%=,%>
<% setXMLOutput() %>
<Descriptor service="it.racomputer.wr4terminal.unit.bms.
  BMSIdentifyMapUnitService">
  <MapIdentifierClass><%=unit["mapIdentifierClass"]%></
    MapIdentifierClass>
  <% for (prop in unit.selectNodes("Property")){ %>
    <Property name="<%=prop["name"]%>" value="<%=prop["value"]%>"/>
  <% } %>
  <SessionName><%=unit["sessionName"]%></SessionName>
</Descriptor>
```

Come si vede nel listato il template richiama la classe `BMSIdentifyMapUnitService`. Si tratta della classe Java che gestisce le funzioni per l'identificazione della

mappa [BMS](#). In particolare ad ogni unit corrisponde una classe Java che implementa un metodo `execute()`, lanciato in esecuzione quando la unit è richiamata.

IL CODICE JAVA La creazione di una nuova unit richiede la creazione del servizio Java che gestisce la logica del componente. In particolare durante il processo di creazione della nuova unità il servizio viene aggiunto nel package di default (`com.WebRatio.unit.custom.nomeunit` se non specificato diversamente). La classe referenziata nel file `Logic.template` viene creata automaticamente nella directory `src` del progetto unit.

Listing 3: BMSIdentifyMapUnitService

```
import ...;
public class BMSIdentifyMapUnitService extends AbstractBMSUnitService
    implements RTXOperationUnitService {
    private IRW3270MapIdentifier mapIdentifier;
    private Map<String, String> props;
    private RTXManager manager;
```

La classe implementa `AbstractBMSUnitService`. Tale classe fornisce alcuni metodi per la gestione della sessione e della connessione. Il campo `mapIdentifier` è di tipo `IRW3270MapIdentifier`. Si tratta di un'interfaccia che dichiara il metodo `getMapName` il quale deve restituire una stringa che definisce il nome corrente della mappa [BMS](#).

```
public Object execute(Map operationContext, Map sessionContext) throws
    RTXException {
    logDebug("execute()");
    ExtendedOperationUnitBean bean = new ExtendedOperationUnitBean();
    RW3270 terminal = getConnection(sessionContext);
    try {
        String mapName = mapIdentifier.getMapName(terminal, props,
            manager);
        bean.put("mapName", mapName);
        logDebug("mapName=" + mapName);
        bean.setResultCode(RTXConstants.SUCCESS_CODE);
    } catch (WR4TerminalException e) {
        logError("Map identification failed", e);
        bean.setResultCode(RTXConstants.ERROR_CODE);
    }
    logDebug("ResultCode=" + bean.getResultCode());
    return bean;
}
```

Oltre alle varie funzioni di debugging il metodo inizializza un bean con il valore della mappa. Da notare che il nome del campo corrisponde al parametro di output specificato nel template di output, cioè `mapName`. Inoltre setta il `resultCode` a "success" o a "error" in base al fatto che

l'operazione abbia successo o meno. Questa impostazione permetterà alla operation unit di decidere se indirizzare la navigazione verso un OK o un KO link dopo la sua esecuzione. L'esecuzione della unit ritorna infine il bean opportunamente valorizzato.

7.2.1.2 Descrittore XML

Questo è il file più importante della nuova unit e gestisce tutti i parametri principali. E' un file [XML](#), ma WebRatio mette a disposizione un interfaccia grafica, suddivisa in diverse sottopagine, che guida nella scelta delle varie opzioni.

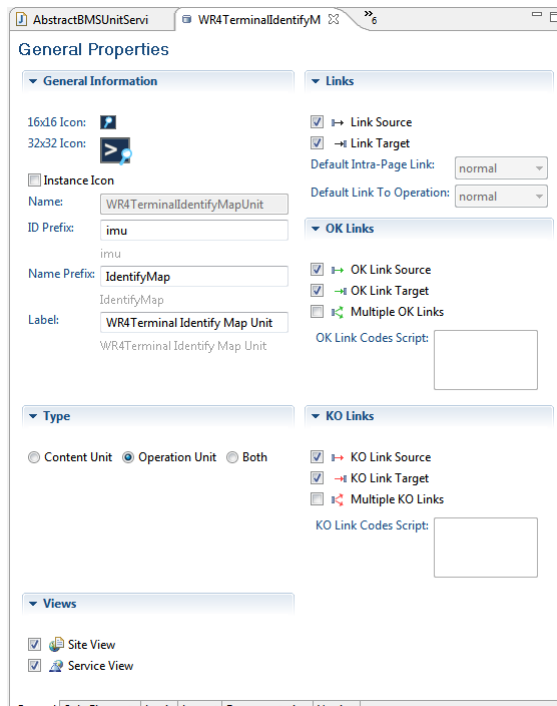


Figure 30: Custom unit

La pagina generale ad esempio ha un duplice scopo: contiene una sezione per impostare le informazioni generali dell'unità come il nome, il tipo, e le viste dove può essere inserita e specifica se la unit può essere di origine e di destinazione di uno o più link. La IdentifyMap Unit ad esempio è una operation unit, e può essere sia sorgente che destinazione di link di tipo OK, KO e di trasporto.

7.3 PROGETTO IN PRATICA

Nei prossimi paragrafi si illustreranno alcune parti dell'implementazione pratica del progetto.

7.3.1 Modello dei dati

L'applicazione non utilizza un database di appoggio poiché come visto, si limita a mettere in pratica quello che è lo *screen scraping*. In sostanza vengono invocati dei servizi i quali leggono mappe BMS, ne estrapolano il contenuto, eventualmente lo manipolano e infine rendono visibile in qualche forma esterna. Di conseguenza, dal punto di vista del modello dei dati, le uniche entità necessarie sono volatili e servono alla memorizzazione dei campi delle mappe.

SAM0SS	SAM800	SAM0PN_SA03_list
oid: integer	oid: integer	abi: string
ctrn: string	ctrn: string	cab: string
zapp: string	zapp: string	numero: string
cist: string	cist: string	tipo: string
copr: string	copr: string	importo: string
dday: string	dday: string	fil/conto cedente: string
otim: string	otim: string	stato: string
cpcc: string	cpcc: string	zriga: string
zpnu: string	zpnu: string	sriga: string
cconses: string	cconses: string	oid: integer
cpnu: string	cpnu: string	
cter: string	cter: string	
znpa: string	znpa: string	
npag: string	npag: string	
zqpa: string	zqpa: string	
qpag: string	qpag: string	
kkkk0601: string	kkkk0408: string	

Figure 31: Entità volatili

In particolare ogni entità è popolata a partire dalla mappa corrispondente: per ognuna esiste infatti un particolare file che ne identifica la struttura specificando in che modo recuperare attributi ed etichette.

Ad esempio per l'entità mostrata in figura, la SAM0SS viene richiamato il seguente codice:

Listing 4: Codice per l'interpretazione della mappa BMS

```

SAM0SS  DFHMSD  TYPE=&SYSPARM,
                                TIOAPFX=YES,
                                STORAGE=AUTO,
                                MODE=INOUT
SAM0SS  DFHMDI  SIZE=(24,80)
CTRN    DFHMDF  POS=(01,01),
                                LENGTH=08,
                                ATTRB=(ASKIP,FSET)
CIST    DFHMDF  POS=(01,43),
                                LENGTH=05,
                                ATTRB=(ASKIP,FSET)

```

La sintassi è abbastanza esplicativa. Si prenda ad esempio l'attributo CIST (abbreviazione per codice istituto); esso è recuperato dalla mappa a partire dalla riga 1 in posizione 43, ed è di lunghezza 5. Gli attributi riguardano le particolari caratteristiche del campo, ad esempio ASKIP gestisce l'autoskip del cursore durante la compilazione dei campi.

7.3.2 Modello dell'ipertesto

L'applicazione si compone di una sola vista poiché allo stato attuale non è prevista differenziazione in base ai servizi. Per ragioni di ottimizzazione spaziale WebRatio offre il concetto di modulo: si tratta di una componente che racchiude al suo interno un insieme di istanze di altre componenti.

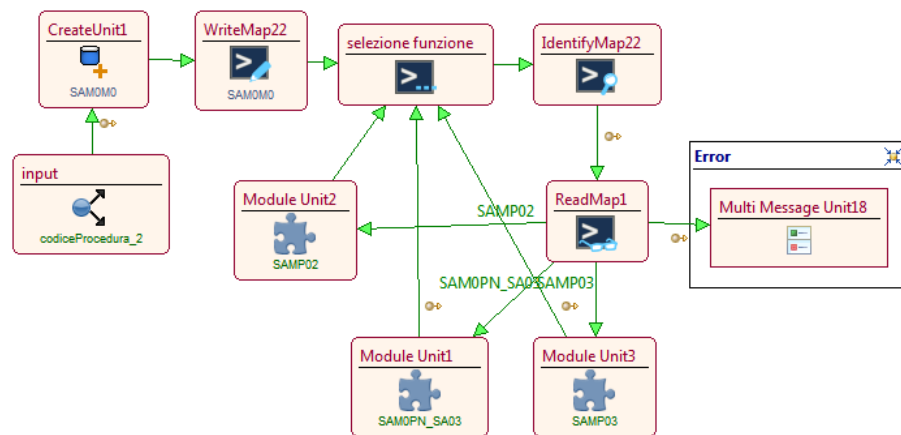


Figure 32: Modulo WebRatio: selezione della procedure ISMA

In figura è mostrata la fase di richiamo di una delle funzioni messe a disposizione da SARA. L'*input collector*¹ porta con sé un parametro il quale identifica la procedura da richiamare (per esempio la ISMA, cioè quella per la gestione degli assegni smarriti). Viene quindi creata un'entità che andrà a contenere attributi e campi della rispettiva mappa alla quale viene passato il codice della procedura al fine di valorizzare in modo opportuno il campo corrispondente (che in questo caso è ISMA). La mappa viene quindi valorizzata con il contenuto dell'entità appena creata e su di essa è eseguito il comando di conferma (la unit *selezioneFunzione* viene cioè inizializzata con il comando ENTER). A questo punto l'applicazione 3270 restituirà una nuova mappa che deve essere identificata e quindi letta. La *readMap* è preimpostata per leggere tre particolari mappe, quelle relative alla procedura in esecuzione (la ISMA). In particolare in caso di mappa SAMP02 si passerà al modulo relativo alla ricerca, in caso di mappa SAMOPN_SA03 al

¹ L'*input collector* è una unit che insieme all'*OK* e *KO collector*, gestisce il passaggio dei parametri in ingresso e in uscita da un modulo.

espressioni del linguaggio Groovy, componenti lato client oppure tag personalizzati), che rappresentano il codice con cui verrà generato l'aspetto finale dei componenti. Esempi tipici sono i template associati alle pagine e alle unit, che ne definiscono la resa visuale. Al momento della generazione dell'applicazione vera e propria, tutta la gerarchia dei template dei singoli componenti viene combinata per generare il layout finale della singola pagina. Nel caso di WebRatio, il risultato è una pagina il cui contenuto è costituito da tag e scriptlet tipici delle tecnologie [JSP](#) e Java Standard Tag Library ([JSTL](#))

WebRatio utilizza alcuni tag e *placeholder* personalizzati per permettere l'associazione tra gli elementi visuali definiti nel template stesso e le informazioni provenienti dal modello WebML. Ad esempio, tramite il tag *Value*, tipico di una content unit, è possibile definire in quale punto del template l'effettivo valore di un determinato attributo di una entità verrà visualizzato. In questo modo, si ottiene un forte disaccoppiamento tra gli elementi che concorrono alla creazione dell'interfaccia vera e propria e la logica applicativa che fornisce i dati veri e propri.

7.4 CRITICITÀ DELLA MODERNIZZAZIONE

La modernizzazione è stata effettuata in maniera efficace ed efficiente grazie alle *custom unit* fornite dalla *w4bms*. Tuttavia esistono ancora delle funzionalità che potrebbero essere integrate nella libreria e che permetterebbero uno sviluppo ancora più veloce. Un esempio è la gestione delle liste: si è visto nel modello dei dati che esiste una corrispondenza 1:1 tra i campi della mappa [BMS](#) e i campi dell'entità volatile corrispondente. Questo avviene anche per le liste: ogni lista è mappata in un'entità ed ogni elemento della lista corrisponde ad una stringa. Tale stringa non è a sua volta mappata in una entità che permette di estrapolarne i valori degli attributi: per fare ciò è necessario effettuare delle substring con l'utilizzo di opportuni script Groovy. Poiché non è dato sapere a priori dove inizia e finisce ogni campo questo si traduce in un lavoro lungo e noioso possibile solo grazie all'uso costante di debugging.

Nello script vengono definiti come parametri di input le righe `riga16`, `riga17`, `riga18` che conterranno le rispettive righe dell'interfaccia testuale dell'applicazione legacy (quindi tre elementi della lista). Le righe quindi non sono altro che campi del bean opportunamente valorizzato a partire dalla mappa corrispondente. Vengono inoltre dichiarate le variabili `stato`, `fil_conto_cedente` e `tipo` definiti come array che conterranno i valori dei campi estrapolati dalle stringhe. Poniamo quindi che la lista contenga tre assegni bancari, identificati dal codice 1, l'array `tipo` conterrà tre elementi stringa valorizzati a `[1,1,1]`.

Listing 5: Script Groovy per lo split della stringa

```

//inputs=RIGA16|RIGA17|RIGA18
//outputs=stato|fil_conto_cedente|tipo
def arrayRighe = [RIGA16|RIGA17|RIGA18]
def stato = []
def fil_conto_cedente = []
def tipo = []
def count = 0 for (riga in arrayRighe){
  if(riga instanceof java.lang.Object[]){
    riga = riga[0]
  }
  if (riga != null && riga instanceof java.lang.String){
    if (!"".equals(riga.trim())){
      stato.add(StringUtils.substring(riga,65))
      fil_conto_cedente.add(StringUtils.substring(riga,46,65))
      importo.add(StringUtils.substring(riga,27,45))
      count=count+1
    }
  }
}
return[
  "stato":stato.toArray(),
  "fil_conto_cedente":fil_conto_cedente.toArray(),
  "importo":importo.toArray(),
]

```

Il ciclo `for` itera le righe passate in input allo script e su ognuna di esse effettua una `substring` il cui risultato verrà memorizzato nella corrispondente variabile di output.

Un altro esempio di problematica non attualmente gestita dalla *w4bms* è la gestione della navigazione. Lo *screen scraping* di solito prevede una corrispondenza 1:1 tra pagina web e mappa bms e consiste in modifiche al layout grafico piuttosto che alla logica di navigazione. Questo accade non perchè tali modifiche non siano tecnicamente e tecnologicamente possibili ma perchè si scontrano con gli obiettivi di una modernizzazione di tipo *black-box*: essa prevede costi di sviluppo bassi sia in termini di tempo che di risorse. Intervenire sulla logica interna dell'applicazione comporta proprio questo: l'aumento dei costi.

Le criticità appena illustrate non sono quindi insormontabili e sono tecnicamente risolvibili attraverso l'implementazione di nuove custom unit in grado di gestire tali situazioni. Tuttavia è bene ricordare che è sempre opportuno, prima di procedere in tale direzione, effettuare un'attenta analisi e comparazione tra i benefici e costi di sviluppo.

L'APPLICAZIONE MODERNIZZATA: SARA WEB

8.1 LA NUOVA INTERFACCIA

La realizzazione dell'interfaccia web mostra un'applicazione completamente rinnovata. Nei prossimi paragrafi verrà illustrato un confronto tra la nuova e la vecchia interfaccia seguendo l'ordine della navigazione logica dell'applicazione. Verranno quindi illustrati i miglioramenti che la modernizzazione ha reso possibili.

8.1.1 *Login*

La parte di login è stata completamente rivisitata. In particolare è stato l'unico caso, insieme alla gestione della paginazione, e all'interrogazione per il recupero dell'informazione descrittiva dei codici, in cui è stata necessaria anche una modifica rispetto alla logica di navigazione. I tre step necessari per l'autenticazione sono mostrati in figura.

Figure 34: Login dell'applicazione legacy

```

SA01      SOLUZIONE ASSEgni RA      99999 *LOGIN*      14/02/13 17:02:10
SA$1      MENU MAPPA INIZIALE      SA$1 N092

USERID.....: _____      PASSWORD.....: 
ISTITUTO.....: _____      LINGUAGGIO....: I
STAMPANTE.....: _____

TIPO OPERAZIONE...: A      CONNESSIONE...: ____
TERMINALE.....: N092

ENT=AVANTI

4B LUTN092 004/029
  
```

(a) Step 1. Inserimento di nome utente, password e istituto. Per procedere è necessario il comando di conferma ENTER.

```

SA01      SOLUZIONE ASSEGNI RA      99999 *LOGIN*      14/02/13 17:04:15
SA$1      MENU MAPPA INIZIALE      SA$1 N092

USERID.....: RASVI14      PASSWORD.....:
ISTITUTO.....: 00001      LINGUAGGIO.....: I
STAMPANTE.....: _____

TIPO OPERAZIONE...: █      CONNESSIONE...: _____
TERMINALE.....: _____

ENT=CICLA      P04=CONF.CLO

CONNESSIONE 2008 APERTA PER UTENTE RASVI14 SU TERMINALE N017 .
4B          LUTN092          008/022

```

- (b) Step 2. L'applicazione comunica terminale e connessione sul quale l'utente può operare. E' necessario inserire tali dati e confermare, questa volta mediante il tasto PF4.

```

SA01      SOLUZIONE ASSEGNI RA      99999 *LOGIN*      14/02/13 17:05:29
SA$1      MENU MAPPA INIZIALE      SA$1 N092

USERID.....: RASVI14      PASSWORD.....:
ISTITUTO.....: 00001      LINGUAGGIO.....: I
STAMPANTE.....: _____

TIPO OPERAZIONE...: A      CONNESSIONE...: 2008
TERMINALE.....: N092

ENT=AVANTI

CONNESSIONE 2008 CHIUSA PER UTENTE RASVI14 E TERMINALE N017 .
4B          LUTN092          002/002

```

- (c) Step 3. L'applicazione da comunicazione dell'avvenuta connessione. E' possibile procedere attraverso il tasto di conferma ENTER.

E' evidente come queste operazioni risultino ripetitive e poco chiare per l'utente, dal momento in cui si tratta principalmente di prendere i dati restituiti dall'applicazione e reinserirli nelle schermate successive. Questi passaggi sono stati quindi automatizzati dalla nuova applicazione in modo completamente trasparente per l'utente, riducendo il numero di passaggi necessari da tre a due. Inoltre l'utente non è più obbligato a ricopiare i dati forniti dall'applicazione, ma si limiterà a dare conferma circa la connessione resa disponibile.

Figure 35: Login dell'applicazione modernizzata

HOME

Istituto 00001 - Istituto Tramite - 03069 ▼

Utente

Login

(a) Step 1. Inserimento del nome utente e dell'istituto.

HOME

Connessione aperta. Continuare?

Avanti Annulla

(b) Step 2. Conferma di connessione. Nel caso in cui si decida di annullare l'operazione si tornerà alla schermata di login, in caso contrario si accederà in maniera diretta all'applicazione.

8.1.2 Pagina di ricerca

Nella seguente figura viene mostrato il confronto tra le pagine di ricerca. Il miglioramento grafico e in termini di usabilità è immediatamente visibile.

SA00 SOLUZIONE ASSEGNI RA 00001 RASVI14 14/02/13 17:06:18
 INGG INQUIRY NEGOZIATI DEL GIORNO 2008A SA02 N092

ISTITUTO : 00001

FILIALE NEGOZIATRICE : _____

NUMERO ASSEGNO / ABI / CAB : _____ CRA : _____

FILIALE CEDENTE / CONTO : _____

DATA OPERAZIONE DA : 14/02/2013 A : 14/02/2013

OPERATORE : _____

STATO (?) : _____

P10=REFRESH ENT=CONFERMA

4B LUTN092 006/032

(a) Maschera di ricerca legacy

(b) Maschera ricerca modernizzata

La modernizzazione ha permesso infatti di inserire una serie di miglioramenti come i radio button e i calendari per i campi data.

Figure 37: Miglioramenti dell'applicazione modernizzata

(a) Dettaglio data

(b) Dettaglio check box

8.1.2.1 La funzione di mapping

Il problema principale risolto in questa sezione riguarda la descrizione dei campi i cui valori sono rappresentati da codici. Come visto nei precedenti capitoli sono numerosi i punti dell'applicazione in cui per ragioni di spazio limitato i campi devono essere valorizzati mediante codici piuttosto che con descrizioni esplicative. Ad esempio per l'assegno bancario si usa il codice 0 per quello circolare il codice 1. Chiaramente questo approccio comporta la richiesta di uno sforzo cognitivo esagerato per l'utente, se non addirittura il rischio di fallimento nell'esecuzione di un task a causa del fatto che egli non riesce a ricordare il codice corretto. A tal scopo

l'applicazione legacy offre una funzione di interrogazione che permette di recuperare l'informazione descrittiva associata a ciascun codice. Tale operazione però richiede diversi passaggi: è chiaro come questo fatto impatti enormemente sull'efficienza dei task svolti.

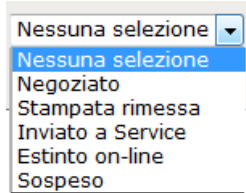


Figure 38: Dettaglio menù tendina

Nell'applicazione modernizzata si è tenuto conto di questo problema ed è stata realizzata una funzione che in modo trasparente per l'utente recupera le informazioni descrittive associate ai codici e le dispone in un menù a tendina. In questo modo l'utente è finalmente esonerato dal compito di ricordare una grande quantità di informazioni e dall'obbligo di interrompere ogni volta il task in esecuzione per l'interrogazione della funzione di traduzione.

8.1.3 Lista risultati

Nella seguente figura viene mostrata la lista contenente i risultati della ricerca. La pagina è divisa in due sezioni, la superiore è riassuntiva circa i parametri utilizzati nella ricerca, la seconda mostra la lista con i risultati. Per ogni occorrenza è possibile visualizzare il dettaglio. Anche la parte dei criteri di ricerca è stata in parte modificata, resa più leggibile ed è stata aggiunta informazione in caso di parametri di ricerca vuoti.

SA00 INGG	SOLUZIONE ASSEgni RA LISTA NEGOZIATI DEL GIORNO				00001 RASVI14 2008A SA03 N092	14/02/13 17:07:13 PAG 1 DI 364		
FIL NEG :					DATA DA :	14/02/2004 A :	14/02/2006	
NUMERO/ABI/CAB :					CRA :	STATO :		
FIL/CONTO CEDENTE :					OPERATORE :			
TOT. NUM. ASSEgni :					4.000	TOT. IMP. ASSEgni :	353.269.779,50	
	ABI	CAB	NUMERO	TIPO	IMPORTO	FIL/CONTO	CEDENTE	STATO
-	00001	00001	0000000001	0	1,00	00001	000000049563	03
-	00002	00002	0000000002	1	1,00	00001	000000049563	03
-	00003	00003	0000000003	1	1,00	00001	000000049563	03
-	00004	00004	0000000004	1	1,00	00001	000000049563	03
-	00005	00005	0000000005	1	1,00	00001	000000049563	03
-	00001	00001	0000000101	0	1,00	00001	000000049563	03
-	00002	00002	0000000102	1	1,00	00001	000000049563	03
-	00003	00003	0000000103	1	1,00	00001	000000049563	03
-	00004	00004	0000000104	1	1,00	00001	000000049563	03
-	00005	00005	0000000105	1	1,00	00001	000000049563	03
-	00006	00006	0000000106	1	1,00	00001	000000049563	03
ENT=CONFERMA P08=AVANTI								
4B					LUTN092	011/002		

(a) Lista risultati applicazione legacy

Infine anche in questa sezione è stata utilizzata la funzione per il mapping tra i campi valorizzati con codici e la relativa descrizione.

The screenshot shows the 'Negozianti del Giorno' application interface. At the top, there are navigation tabs: 'Negozianti del Giorno', 'Negozianti', 'Tratti', 'Rimesse', 'Smarriti', and 'Logout'. Below the tabs is a header bar with the text 'HOME - Negozianti del Giorno'. Underneath is a 'Parametri di ricerca' section with the following fields: 'FILIALE NEGOZIATRICE: ----', 'OPERATORE: ----', 'STATO: ----', 'NUMERO/ABI/CAB: ----', 'CRA: ----', 'DATA DA: 28/01/2004', 'A: 28/01/2006', 'TOT. NUMERO ASSEGNI: 4.000', and 'TOT. IMPORTO ASSEGNI: 353.269.779,50'. Below the search parameters is a 'Risultati' section containing a table with the following columns: 'ABI', 'CAB', 'Numero', 'Tipo', 'Importo', 'Filiale/Conto Cedente', and 'Stato'. The table contains 15 rows of data. At the bottom of the table is a button labeled 'Indietro'. Below the table, there are navigation arrows and the text '2 di 364'.

ABI	CAB	Numero	Tipo	Importo	Filiale/Conto Cedente	Stato
00007	00007	0000000107	Circolare	1,00	00001 000000049563	Inviato a Service
00008	00008	0000000108	Circolare	1,00	00001 000000049563	Inviato a Service
00009	00009	0000000109	Circolare	1,00	00001 000000049563	Inviato a Service
00001	00001	0000000171	Bancario	1,00	00001 000000049563	Inviato a Service
00002	00002	0000000172	Circolare	1,00	00001 000000049563	Inviato a Service
00003	00003	0000000173	Circolare	1,00	00001 000000049563	Inviato a Service
00001	00001	0000000551	Bancario	1,00	00001 000000049563	Inviato a Service
00002	00002	0000000552	Circolare	1,00	00001 000000049563	Inviato a Service
00003	00003	0000000553	Circolare	1,00	00001 000000049563	Inviato a Service
00004	00004	0000000554	Circolare	1,00	00001 000000049563	Inviato a Service
00005	00005	0000000555	Circolare	1,00	00001 000000049563	Inviato a Service

(b) Lista risultati applicazione modernizzata

8.1.3.1 Paginazione

Un'altra parte ostica per la riguarda la paginazione tra i risultati di una ricerca. L'applicazione legacy prevede di poter andare avanti di una pagina alla volta. E' chiaro che questo risulta piuttosto scomodo quando i risultati sono nell'ordine delle decine o addirittura delle centinaia. Per questo motivo nell'applicazione modernizzata sono stati inseriti dei pulsanti che permettono di raggiungere direttamente la prima e l'ultima pagina; quando l'utente clicca sull'icona che porta all'ultima pagina dei risultati quello che avviene in maniera trasparente è che l'applicazione richiama ricorsivamente il tasto funzione PF7 o PF8 che gestiscono la paginazione avanti e indietro, fino a raggiungere la pagina desiderata.

8.1.4 Dettaglio

Infine è mostrata in figura la parte di dettaglio che, a meno del layout, è stata fondamentalmente lasciata invariata.

Figure 40: Dettaglio

```

SA00      SOLUZIONE ASSEgni RA      00001 RASVI14      14/02/13 17:08:12
NGG      DETTAGLIO NEGOZIATI DEL GIORNO 2008A SA04 N092

ABI/CAB TRATTARIA      : 00001 / 00001
NUMERO ASSEGNO         : 0000000001 TIPO      : 0 BANCARIO
IMPORTO                : 1,00 DIVISA      : EUR
FILIALE NEGOZIATRICE   : 00100
NUMERO OPERAZIONE      : 
DATA DI NEGOZIAZIONE   : 27-04-2005 OPERATORE : SAPBNSR1
DATA EMISSIONE         : 21-04-2005 TIPO NEGOZIAZIONE : MANUALE
                        : DATA VALUTA VERSAMENTO: 27-04-2005

FILIALE/ C/C CEDENTE   : 00001 000000049563
FILIALE/ C/C TRAENZA   : 

STATO                  : 03
NUMERO RIMESSA         : 00000000000010173 DATA RIMESSA : 18-03-2005
IMPORTO RIMESSA        : 0,00
LUOGO ARCHIVIAZIONE    : CRA :

P03=RITORNO P10=INF.TRAT P11=INF.OPER ENT=INVIO

4B LUTN092 002/002

```

(a) Dettaglio applicazione legacy

Negoziati del Giorno	Negoziati	Tratti	Rimesse	Smarriti	Logout
HOME - Negoziati del Giorno					
Dettaglio					
ABI/CAB TRATTARIA:	00009 00009	FILIALE NEGOZIATRICE:	00100		
NUMERO ASSEGNO:	0000000109	TIPO:	CIRCOLARE		
IMPORTO:	1,00	DIVISA:	EUR		
NUMERO OPERAZIONE:		OPERATORE:	SAPBNSR1		
DATA DI NEGOZIAZIONE:	27-04-2005	TIPO NEGOZIAZIONE:	MANUALE		
DATA EMISSIONE:	21-04-2005	DATA VALUTA VERSAMENTO:	27-04-2005		
FILIALE C/C CEDENTE:	00001 000000049563	FILIALE C/C TRAENTE:			
STATO:	INVIATO A SERVICE	CRA:			
NUMERO RIMESSA:	00000000000010173	DATA RIMESSA:	18-03-2005		
IMPORTO RIMESSA:	0,00	LUOGO ARCHIVIAZIONE:			
Indietro					

(b) Dettaglio applicazione web

8.2 MIGLIORAMENTI

I miglioramenti apportati all'applicazione possono quindi essere riassunti in termini di navigazione e di aspetto grafico.

8.2.1 Navigazione

I miglioramenti della navigazione riguardano quei cambiamenti in cui è stata alterata la corrispondenza 1:1 tra mappa BMS e pagina web in maniera trasparente per l'utente finale. Tali miglioramenti sono:

- il login: gli step obbligatori sono stati ridotti da tre a due e sono necessarie meno interazioni;
- paginazione: non è più obbligatorio spostarsi una pagina alla volta poichè grazie alle apposite icone è possibile arrivare in maniera più efficiente alla pagina desiderata;
- traduzione dei codici: l'utente non è obbligato ad utilizzare l'apposita funzione per la traduzione dei codici poichè essi sono già presentati in pagina in una forma descrittiva.

8.2.2 *Aspetto grafico*

I miglioramenti grafici realizzati nell'applicazione sono:

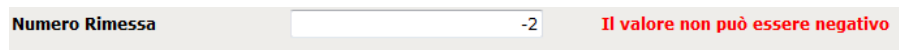
MENÙ A TENDINA I menù a tendina permettono di non dover ricordare ogni volta la corrispondenza tra codice e significato o di doverla cercare con l'apposita funzione. Inoltre limitano la possibilità di interazione prevenendo la possibilità di errore.

MENÙ DI NAVIGAZIONE L'applicazione legacy non disponeva di un vero e proprio menù di navigazione, era solo possibile richiamare le procedure desiderate inserendone il nome nel campo opportuno e confermando con il comando ENTER. L'applicazione modernizzata ha invece un menù le cui voci identificano le procedure, che possono quindi essere richiamate con un solo click e nessuno sforzo cognitivo.

COMANDI Il significato dei comandi dell'applicazione legacy deve essere spesso memorizzato poichè nella pagina non c'è spazio sufficiente per la sua spiegazione; inoltre tali comandi non sono esplicativi nella loro forma: non c'è alcuna corrispondenza semantica tra il tasto funzione PF8 e il significato "avanti". Per questo motivo nell'applicazione modernizzata tutti i comandi sono stati sostituiti da icone e link descrittivi: ad esempio il tasto avanti e indietro nella paginazione sono delle icone a forma di freccia e la selezione di un elemento di una lista è un'icona a forma di lente di ingrandimento.

ETICHETTE DEI CAMPI Le etichette dei campi dell'applicazione legacy non sempre sono descrittive. Questo, come la maggior parte dei problemi dell'applicazione, dipende dai vincoli di spazio dell'interfaccia testuale. Nell'applicazione modernizzata tutti i nomi sono quindi riportati in maniera estesa.

CONTROLLO AGGIUNTIVO PER L'ERRORE In fase di analisi euristica si è visto che alcuni campi della maschera di ricerca avevano dei problemi per quanto riguarda la parte di validazione; ad esempio l'applicazione non verificava che la data "da" fosse antecedente rispetto alla data "a" o l'inserimento di valori non validi. Per questi campi è stata introdotta una validazione precedente rispetto al richiamo dell'applicazione legacy e quindi l'applicazione web risulta maggiormente robusta.



The image shows a web form with a label "Numero Rimessa" on the left. To its right is a text input field containing the value "-2". To the right of the input field, there is a red error message: "Il valore non può essere negativo".

Figure 41: Gestione addizionale dell'errore

La gestione dell'errore è inoltre migliorata grazie alle modifiche appena descritte: i vincoli introdotti da menù a tendina, checkbox e menù.

ANALISI DI USABILITÀ E ACCESSIBILITÀ DI SARA WEB.

L'analisi dell'accessibilità e dell'usabilità sono due strumenti chiave per comprendere i problemi legati all'interazione con l'applicazione. In questo contesto l'analisi si rivela utile per comprendere se la modernizzazione abbia ottenuto i risultati previsti o se invece la nuova applicazione risulta ancora affetta dagli stessi problemi di quella legacy.

9.1 ANALISI USABILITÀ

Per l'analisi dell'usabilità sono stati effettuati dei test con utenti che prevedevano l'esecuzione di un determinato numero di task e la successiva compilazione di un questionario. Per i dettagli sulla preparazione, esecuzione e risultati dei test si rimanda all'[Appendix B](#). Nei prossimi paragrafi ci si limiterà a fornire un'analisi dei risultati di tali esperimenti.

9.1.1 Statistiche dell'esecuzione dei test utente

Nella tabella vengono riportati i valori medi dei dati rilevati durante l'esperimento con gli utenti.

Figure 42: Risultati medi dei test utente

Test di compito per SARA Legacy	Tempo medio in sec	Num. medio interazioni	Risposte sbagliate	Utenti arresi
Effettua login	105	3	/	/
Verifica tramite l'inquiry negoziati del giorno quanti sono gli assegni trattati per il periodo di tempo 2002/04	100	6	2	/
Per gli assegni negoziati vedere il dettaglio di un assegno con procedura stanza (date 2002/04)	50	6	/	/
Per gli assegni tratti cercare quelli con date comprese tra 2002/04 caratterizzati dallo stato anomalo	100	6	2	/
Individuare tra le rimesse la data contabile e la data rimessa di un assegno negoziato batch	110	7	3	/
Per gli assegni smarriti tra il 2002/04 verifica il numero di assegni con tipo di procedura check truncation	90	5	/	/
Effettua il logout	7	1	/	/

(a) Il caso dell'applicazione legacy

Test di compito per SARA Web	Tempo medio in sec	Numero medio click	Risposte sbagliate	Utenti arresi
Effettua login	7	2	/	/
Verifica tramite l'inquiry negoziati del giorno quanti sono gli assegni trattati per il periodo di tempo 2002/04	30	1	/	/
Per gli assegni negoziati vedere il dettaglio di un assegno con procedura stanza (date 2002/04)	50	6	/	/
Per gli assegni tratti cercare quelli con date comprese tra 2002/04 caratterizzati dallo stato anomalo	25	2	/	/
Individuare tra le rimesse la data contabile e la data rimessa di un assegno negoziato batch	30	2	/	/
Per gli assegni smarriti tra il 2002/04 verifica il numero di assegni con tipo di procedura check truncation	10	1	/	/
Effettua il logout	3	1	/	/

(b) Il caso dell'applicazione web

La grande differenza in termini di efficienza risulta evidente fin dal primo task: effettuare il login. Grazie alla modernizzazione il tempo medio richiesto per l'esecuzione del task è passato da 105 a 10 secondi. Tale proporzione, anche se in maniera meno eclatante, è valida per quasi tutti gli altri compiti. Questo è conseguenza del fatto che sono necessarie meno interazioni (click o esecuzione di comandi) per l'esecuzione di un determinato task e che ogni interazione risulta semplificata dall'intuitività dell'interfaccia grafica. Questo ha permesso anche una diminuzione delle risposte sbagliate: un esempio è l'utente che esegue il tasto funzione non corretto poiché gli è difficile ricordare la funzione corrispondente.

E' da notare che la maggior parte degli utenti che hanno eseguito il test sono utenti con una certa familiarità con le interfacce testuali tipiche del 3270. Questa scelta è motivata dal fatto che l'utente nuovo a questa interfaccia, senza una spiegazione introduttiva, sarebbe impossibilitato all'interazione e un confronto tra i risultati dei due esperimenti risulterebbe poco significativo. D'altra parte bisogna tenere conto del fatto che tutti gli utenti hanno oggi una certa esperienza in fatto di interfacce web mentre quasi nessun ricorda più e ha dimestichezza con le vecchie interfacce testuali.

9.1.2 Statistiche dei questionari

Anche i questionari fatti compilare agli utenti al termine dell'esecuzione dei task hanno mostrato un livello di soddisfazione molto più alto per l'applicazione modernizzata. Per quanto riguarda la logica di navigazione ad esempio gli utenti che dichiaravano "E' molto difficile muoversi all'interno dell'applicazione" sono passati dal 70% allo 0%.

Figure 43: Domanda n° 2 del questionario: *E' molto difficile muoversi all'interno dell'applicazione-sito*

Molto d'accordo	70%	Molto d'accordo	0%
D'accordo	20%	D'accordo	0%
Così così	10%	Così così	10%
Disaccordo	0%	Disaccordo	90%
Molto disaccordo	0%	Molto disaccordo	0%

(a) SARA Legacy
(b) SARA Web

In generale gli utenti che utilizzavano l'applicazione modernizzata hanno trovato che la navigazione risultasse più facile, con la conseguente impressione di avere un maggiore controllo circa le possibilità di interazione. Hanno notato miglioramenti anche in termini di efficienza ed efficacia, e i compiti assegnati risultavano di più facile esecuzione. Mentre la maggior parte degli utenti ha sentito la necessità di maggiore documentazione e spiegazioni per l'applicazione legacy, questa esigenza è risultata completamente assente per l'applicazione modernizzata: in questo caso essi hanno dichiarato che l'applicazione risultava intuitiva e chiara fin dal primo utilizzo. Infine hanno mostrato di gradire maggiormente l'aspetto grafico.

9.2 ANALISI DELL'ACCESSIBILITÀ

Tim Berners-Lee, inventore del Web ha detto: *“La forza del Web sta nella sua universalità. L'accesso da parte di chiunque, indipendentemente dalle disabilità, ne è un aspetto essenziale [11]”*. Tale definizione porta con sé due degli elementi fondamentali dell'accessibilità:

- l'attenzione ai problemi di accesso al web dei disabili;
- l'attenzione a garantire l'universalità dell'accesso, ovvero a non escludere nessuno: non solo i disabili in senso stretto, ma anche chi soffre di disabilità temporanee, chi ha attrezzature obsolete, chi usa sistemi poco comuni, chi dispone di connessioni particolarmente lente.

9.2.1 L'Italia e la legge Stanca

In Italia l'accessibilità è regolata dalla *Legge 4 gennaio 2004, n. 4, Disposizioni per favorire l'accesso dei soggetti disabili agli strumenti informatici [49]*, nota come legge Stanca. Essa contiene le seguenti definizioni:

ACCESSIBILITÀ: intesa come la capacità dei sistemi informatici, nelle forme e nei limiti consentiti dalle conoscenze tecnologiche, di erogare servizi

e fornire informazioni fruibili, senza discriminazioni, anche da parte di coloro che a causa di disabilità necessitano di tecnologie assistive o configurazioni particolari;

TECNOLOGIE ASSISTIVE: cioè gli strumenti e le soluzioni tecniche, hardware e software, che permettono alla persona disabile, superando o riducendo le condizioni di svantaggio, di accedere alle informazioni e ai servizi erogati dai sistemi informatici.

La Legge Stanca nasce per riconoscere e tutelare il diritto di ogni persona ad accedere a tutte le fonti di informazione e ai relativi servizi, inclusi quelli che si articolano attraverso gli strumenti informatici e telematici. La legge Stanca è rivolta agli enti pubblici economici, alle aziende private concessionarie di servizi pubblici, alle aziende municipalizzate regionali, agli enti di assistenza e di riabilitazione pubblici, alle aziende di trasporto e di telecomunicazione a prevalente partecipazione di capitale pubblico e alle aziende appaltatrici di servizi informatici e sancisce la necessità di allineare gli strumenti informatici, incluso il web, alle disposizioni in merito ai principi di accessibilità da essa dettati nelle linee guida pubblicate nel luglio 2005. La modernizzazione dell'applicazione risulta in questo senso estremamente in portante poiché permette di allinearsi facilmente alle disposizioni di legge attualmente in vigore.

Dato che in sede internazionale le raccomandazioni standard sono proposte sono dal Web Accessibility Initiative (WAI)¹ e ad esse stesse si ispira la legge Stanca, di seguito se ne propone una panoramica contestualizzandola rispetto all'applicazione modernizzata.

9.2.2 *Web Content Accessibility Guidelines 2.0*

Le WCAG 2.0[53] definiscono come rendere accessibile il contenuto web alle persone affette da disabilità, dove per disabilità si intende quella visiva, uditiva, fisica, del parlato, cognitiva, linguistica, dell'apprendimento e disabilità neurologiche. Sebbene queste linee guida coprano una vasta gamma di necessità, non sono in grado di soddisfare le esigenze delle persone con tutti i tipi, i gradi, e le combinazioni di disabilità. Queste linee guida rendono inoltre i contenuti web utilizzabili più facilmente da persone in età avanzata, con capacità mutevoli a causa dell'invecchiamento, e spesso migliorano l'usabilità per tutti gli utenti in generale. L'accessibilità del web non dipende solamente dal contenuto accessibile ma anche dall'accessibilità dei browser web e degli altri programmi utente.

¹ La WAI è un'iniziativa del World Wide Web Consortium (W3C) finalizzata a migliorare l'accessibilità del Web rispetto alle persone affette da un qualche tipo di disabilità. A tale scopo sono state proposte alcune linee guida: le WCAG 1.0 nel 1999, poi aggiornate in una nuova versione, le WCAG 2.0 nel 2008.

In tabella sono riportate le linee guida suddivise per principio di appartenenza

Accessibilità: linee guida WCAG 2.0	
1 Percebibile	1 Alternative testuali: Fornire alternative testuali per qualsiasi contenuto non di testo in modo che questo possa essere trasformato in altre forme fruibili secondo le necessità degli utenti come stampa a caratteri ingranditi, Braille, sintesi vocale, simboli o un linguaggio più semplice.
	2 Adattabile: Creare contenuti che possano essere rappresentati in modalità differenti (ad esempio, con layout più semplici), senza perdere informazioni o la struttura.
	3 Fornire alternative per i tipi di media temporizzati.
	4 Distinguibile: Rendere più semplice agli utenti la visione e l'ascolto dei contenuti, separando i contenuti in primo piano dallo sfondo.
2 Utilizzabile	1 Accessibile da tastiera: Rendere disponibili tutte le funzionalità tramite tastiera.
	2 Fornire agli utenti tempo sufficiente per leggere ed utilizzare i contenuti.
	3 Convulsioni: Non sviluppare contenuti che possano causare attacchi epilettici.
	4 Navigabile: Fornire delle funzionalità di supporto all'utente per navigare, trovare contenuti e determinare la propria posizione.
3 Comprensibile	1 Leggibile: Rendere il testo leggibile e comprensibile.
	2 Prevedibile: Creare pagine Web che appaiano e che siano prevedibili.
	3 Assistenza nell'inserimento: Aiutare gli utenti ad evitare gli errori ed agevolarli nella loro correzione.
4 Robusto	1 Compatibile: Garantire la massima compatibilità con i programmi utente attuali e futuri, comprese le tecnologie assistive.

Table 3: Accessibilità: linee guida WCAG 2.0

Di seguito è riportata un'analisi dell'applicazione modernizzata basata su tali linee guida

9.2.3 SARA Web e accessibilità

L'applicazione non prevede contenuti audio, tuttavia per quanto riguarda l'informazione visiva come le immagini è sempre presente un'alternativa testuale equivalente (*principio 1.1*). Ad esempio le icone cliccabili del dettaglio e quella di paginazione riportano un tooltip descrittivo, e tale testo è informativo circa l'azione possibile attraverso l'interazione. A livello di implementazione questo significa che per tutti gli elementi IMG, INPUT hanno l'attributo alt opportunamente valorizzato.

Figure 44: Alternativa testuale alle immagini



I contenuti della pagina sono adattabili alle differenti modalità di interazione e visualizzazione (*principio 1.2*): è quindi possibile l'accesso alle diverse funzionalità indipendente dai dispositivi usati. L'applicazione prevede la possibilità di operare da tastiera, mouse e comandi vocali. Sono state usate caratteristiche che permettono di attivare gli elementi della pagina attraverso una molteplicità di dispositivi di input (*principio 2.1*); in particolare si è visto che l'applicazione permette l'interazione anche solo tramite tastiera; questo risulta essere sufficiente, nella maggior parte dei casi, per garantire l'accessibilità anche mediante input vocale o interfaccia a linea di comando.

L'informazione non è mai veicolata solo attraverso l'uso del colore e le combinazioni tra i colori dello sfondo e primo piano forniscono sempre sufficiente contrasto (*principio 1.4*). Per la verifica sono stati utilizzati opportuni plug-in per browser atti alla validazione della pagina ².

Viene sempre fornita informazione per la contestualizzazione e l'orientamento, al fine di aiutare gli utenti a comprendere i contenuti (*principio 2.4*). Ne sono un esempio la voce di menù che cambia colore per identificare la sezione corrente in cui ci si trova, lo sfondo alternato delle righe della tabella per permettere una migliore comprensione visiva, e il colore di

² Per la validazione dei requisiti è stata utilizzata l'*accessibility toolbar* integrata nel browser Mozilla Firefox. Per ulteriori informazioni si veda <http://firefox.cita.uiuc.edu/>

HOME - Negoziati del Giorno

Parametri di ricerca

FILIALE NEGOZIATRICE: ---- OPERATORE: ---- STATO: ----
 NUMERO/ABI/CAB: ---- CRA: ----
 DATA DA: 29/03/2004 A: 29/03/2006
 TOT. NUMERO ASSEGGI: 4.000 TOT. IMPORTO ASSEGGI: 353.269.779,50

ABI	CAB	Numero	Tipo	Importo	Filiale/Conto Cedente	Stato
00001	00001	0000000001	Bancario	1,00	00001 0000000049563	Inviato a Service
00002	00002	0000000002	Circolare	1,00	00001 0000000049563	Inviato a Service
00003	00003	0000000003	Circolare	1,00	00001 0000000049563	Inviato a Service
00004	00004	0000000004	Circolare	1,00	00001 0000000049563	Inviato a Service
00005	00005	0000000005	Circolare	1,00	00001 0000000049563	Inviato a Service
00001	00001	0000000101	Bancario	1,00	00001 0000000049563	Inviato a Service
00002	00002	0000000102	Circolare	1,00	00001 0000000049563	Inviato a Service
00003	00003	0000000103	Circolare	1,00	00001 0000000049563	Inviato a Service
00004	00004	0000000104	Circolare	1,00	00001 0000000049563	Inviato a Service
00005	00005	0000000105	Circolare	1,00	00001 0000000049563	Inviato a Service
00006	00006	0000000106	Circolare	1,00	00001 0000000049563	Inviato a Service

Indietro

Figure 45: Contestualizzazione dell'informazione

selezione per la riga evidenziata quando in corrispondenza dell'evento *on-mouseover*.

Per quanto riguarda l'utilizzo delle nuove tecnologie, nella loro applicazione si è sempre tenuto conto della retrocompatibilità con i browser più vecchi (*principio 4.1*). Questo è vero per l'utilizzo di linguaggi di scripting quali JavaScript e jQuery. In particolare è stata verificata la retrocompatibilità con i browser: *Internet Explorer* (dalla versione 6), *Chrome 16*, *Firefox 9*, *Opera 11.60* e *Safari 5.1.2*.

Il linguaggio naturale utilizzato è sempre opportunamente dichiarato e non vengono utilizzate abbreviazioni. Questo fattore risulta molto importante poiché cambiamenti di lingua non dichiarati e abbreviazioni possono risultare indecifrabili per la lettura da parte dei dispositivi di sintesi vocale.

Sono state utilizzate soluzioni provvisorie in modo che le tecnologie assistive e i browser più vecchi possano operare correttamente; in particolare non vengono mai aperte nuove finestre e non è mai cambiato il *focus* da una finestra all'altra senza che l'utente ne sia informato; inoltre nel caso in cui le tecnologie assistive non siano in grado di differenziare collegamenti adiacenti ma distinti, tali collegamenti sono sempre separati da opportuni spazi.

L'inserimento di dati è assistito e facilitato (*principio 3.3*) dagli strumenti messi a disposizione quali menù a tendina, radio button e link. Tali strumenti aiutano anche nella prevenzione dell'errore poiché vincolano l'interazione non consentendo all'utente di effettuare una scelta sbagliata.

Infine la pagina è disposta in maniera chiara e le varie sezioni sono altamente modulari rendendo la navigazione prevedibile; l'interazione è possibile tramite icone esplicative (ad esempio l'immagine di dettaglio) e *button* ben identificabili: i meccanismi di navigazione utilizzati sono quindi semplici, intuitivi e chiari (*principio 3.2*).

9.2.3.1 Criticità

Le criticità dell'applicazione in termini di accessibilità riguardano la disposizione degli elementi nella pagina. Il livello di presentazione di WebRatio prevede di strutturare la pagina in una griglia principale all'interno della quale si collocano altre griglie oppure righe e colonne, quindi celle. Questa esigenza nasce dal fatto che tale strumento deve offrire un approccio quanto più modulare e adattabile possibile. La soluzione della griglia supporta tali caratteristiche poiché in questo modo ogni elemento può essere posizionato e personalizzato in maniera appropriata. Tuttavia questo tipo di gestione si traduce nell'utilizzo di tabelle per l'impaginazione dei contenuti. Questo approccio può rendere difficile, per l'utente che utilizza software specialistico, la comprensione dell'organizzazione della pagina o la navigazione. In sostanza non si è tenuto conto della differenza tra contenuto, struttura e presentazione.

9.3 CONCLUSIONI

Grazie ai test con gli utenti è stato possibile mettere in evidenza i miglioramenti dell'applicazione modernizzata in termini di usabilità, intesa come efficacia efficienza e soddisfazione nell'utilizzo dell'applicazione stessa. I risultati sono stati soddisfacenti e rivelano un'applicazione migliorata sotto tutti e tre gli aspetti.

Con la nuova applicazione infatti gli utenti riescono a svolgere i compiti assegnati in maniera:

- efficace: si rendono necessarie un numero limitato di interazioni e la probabilità di errore è diminuita;
- efficiente: i task sono eseguibili in pochi passi, con uno scarso sforzo cognitivo e in breve tempo;
- soddisfacente: dal momento che i compiti risultano facilmente realizzabili.

L'applicazione inoltre permette l'accesso universale, poiché è sufficiente essere collegati alla rete e disporre di un browser web. Inoltre anche i requisiti di accessibilità risultano rispettati in quanto, come si è visto dall'analisi, durante la fase di progettazione si è tenuto conto delle [WCAG](#). Questo si traduce nella possibilità di rendere l'applicazione disponibile anche per quelle persone affette da un qualche tipo di disabilità visiva, uditiva e/o cognitiva.

CONCLUSIONI

I problemi legati ai sistemi legacy rivestono un ruolo cruciale specialmente per le aziende che oggi hanno l'esigenza di restare competitive nella recessione del mercato. Esse devono riuscire a incrementare i servizi senza aumentare i costi, e perseguire l'agilità per permettere di stare al passo con i processi di business; ciò è possibile solo mediante un adeguamento della propria infrastruttura IT in modo che questa possa continuare a supportare i requisiti di business in costante evoluzione. I problemi legati ai sistemi legacy possono riguardare la logica interna così come lo strato più superficiale, cioè la loro interfaccia. E' il caso delle applicazioni CICS accessibili da terminale 3270, caratterizzate da un'interfaccia testuale: tipicamente queste applicazioni sono ancora efficaci in termini funzionali ma presentano diverse problematiche che riguardano usabilità, accessibilità ed efficienza operativa. Non si deve pensare che tali applicazioni siano relegate a ricoprire ruoli marginali all'interno delle infrastrutture IT aziendali: l'applicazione modernizzata in questo lavoro, ad esempio, è addetta alla gestione degli assegni tra le diverse filiali. Questo significa che ogni volta che si effettua un acquisto tramite un assegno o che ci si reca in banca per la sua riscossione, la gestione dello stesso sarà gestita attraverso l'applicazione in questione. Il problema pratico che ogni filiale si trova ad affrontare consiste nel fatto che affinché i propri dipendenti siano in grado di utilizzarla devono essere sottoposti ad un opportuno addestramento: ciò si traduce in costi monetari, per l'organizzazione di corsi formativi, e temporali, poiché i dipendenti non possono essere subito operativi. Inoltre, anche quando ne hanno finalmente imparato l'utilizzo, restano tutti i problemi legati alla scarsa usabilità e quindi efficienza che l'applicazione è in grado di offrire.

Per questo tipo di applicazioni la modernizzazione di tipo black-box è decisamente la più appropriata in quanto permette di intervenire sul sistema tralasciando i dettagli circa il suo funzionamento interno. WebRatio, in questo contesto, si è rivelato un ottimo strumento: i vantaggi che esso porta con sé sono molteplici. Innanzitutto ha permesso di riutilizzare gran parte dell'applicazione legacy, il che si traduce in costi più bassi, tempi di sviluppo minori e rischio di insuccesso limitato. E' infatti da ricordare che uno dei problemi critici riguardo alla modernizzazione dei sistemi legacy risiede nel fatto che tali applicazioni racchiudono il core business aziendale, nonché la conoscenza e l'esperienza accumulata durante i decenni di utilizzo. Agire in maniera invasiva su tali applicazioni porta con sé il rischio di intaccare quel core dell'applicazione così cruciale per l'azienda poiché è quello che ne determina il vantaggio competitivo.

Inoltre l'approccio model driven che caratterizza WebRatio fa sì che esso sia di facile utilizzo anche per i non esperti nel settore permettendo il riciclo della conoscenza anche in termini di personale. Un altro dei fattori critici della modernizzazione risiede nel fatto che esiste oggi un certo divario delle conoscenze del personale: da un lato ci sono le persone che lavorano nel mondo dell'IT dagli anni '70, conoscono molto bene i sistemi legacy ma sanno ben poco riguardo le nuove tecnologie; dall'altro ci sono le persone figlie delle nuove tecnologie che ignorano completamente il funzionamento dei sistemi legacy. Entrambi non hanno in generale le competenze per poter modernizzare: i primi non conoscono le tecnologie con cui poterlo fare, i secondi hanno teoricamente gli strumenti ma non capiscono a fondo il sistema su cui bisognerebbe intervenire. WebRatio, con il suo approccio model driven, permette di creare un ponte tra i due tipi di conoscenza. Un altro vantaggio di questo tipo di modernizzazione è la sua automaticità. La libreria w4bms, come visto, permette l'automazione dello screen scarping: in pratica lo sviluppatore in maniera pressoché automatica può mappare le vecchie schermate dell'applicazione legacy nelle nuove pagine dell'applicazione web.

I risultati della modernizzazione in termini di usabilità sono risultati conformi se non superiori alle aspettative. L'analisi dei risultati ottenuti mediante i test con utenti hanno infatti dimostrato che l'applicazione web risulta notevolmente migliorata sia in termini di efficacia che di efficienza. L'utilizzatore finale infatti riesce a svolgere le normali operazioni in tempi più brevi e con un minor numero di interazioni. Questo significa maggiore soddisfazione e inevitabilmente anche una maggiore produttività. L'intuitività nell'utilizzo dell'applicazione modernizzata inoltre la rende "pronta per l'utilizzo". Questo significa che l'utente non deve più essere sottoposto a formazione, come accadeva per le applicazioni legacy, e può essere subito operativo. Anche questo aspetto risulta di particolare importanza e si traduce ancora una volta in una riduzione dei costi per l'azienda. La modernizzazione ha permesso anche un notevole miglioramento in termini di accessibilità; oltre che essere raggiungibile via web ora l'applicazione è utilizzabile anche da parte di tutte quelle persone che sono affette da particolari disabilità cognitive e percettive. Questo è possibile grazie all'utilizzo di template che rispettino le [WCAG](#).

I problemi riscontrati durante la modernizzazione sono legati sia al tipo di modernizzazione scelto che allo strumento. I primi dipendono dal fatto che il revamping non permette di modificare la logica interna dell'applicazione e quindi i problemi che facevano parte del core dell'applicazione legacy sono ancora presenti in quella modernizzata. I secondi riguardano invece i limiti in termini di personalizzazione che è possibile effettuare. La forza di WebRatio e della w4bms sta nel fatto che essi sono strumenti basati su una modellazione concettuale che permette di gestire un certo numero di situazioni: grazie a ciò si hanno tempi di sviluppo brevi, costi bassi e ris-

chio limitato. Questo però significa che i requisiti dell'applicazione che non ricadono dentro allo standard previsto dallo strumento sono difficili da gestire ed estremamente costosi poiché richiedono una personalizzazione su misura.

La w4bms è comunque uno strumento flessibile in grado di soddisfare le necessità di business in costante evoluzione: nel caso in cui nascessero nuovi requisiti è sempre possibile sviluppare nuove unit in grado di soddisfarli. In questo senso è possibile affermare che il tool di modernizzazione non può essere considerato uno strumento finito, e sono possibili future revisioni per l'implementazione di nuove funzionalità che permettano di gestire nuovi requisiti. E' ovviamente opportuno valutare ogni volta il bilancio tra i benefici che tale implementazione porterebbe rispetto ai costi richiesti.

In conclusione l'accoppiata WebRatio w4bms è risultata essere un ottimo strumento per la modernizzazione di applicazioni legacy 3270 nel caso in cui i problemi siano legati esclusivamente all'usabilità e all'accessibilità e non si rendano necessarie particolari personalizzazioni.

Part III

APPENDICE

ANALISI EURISTICA

L'analisi euristica è stata svolta da un valutatore, il che secondo le ricerche di Nielsen dovrebbe essere sufficiente a trovare circa un terzo dei problemi di usabilità totali.

Valutatori

Il valutatore si è calato nei panni di un utente esperto. Egli è un uomo o una donna che rientra in una fascia di età compresa tra i 20 e 25 anni, ha una buona conoscenza della lingua inglese (è capace sia di comprendere testi scritti che di sostenere una conversazione). Ha ottime conoscenze informatiche: sa usare tutti i principali programmi applicativi, ha familiarità con i diversi sistemi operativi, conosce più linguaggi di programmazione (sia ad oggetti che procedurali) ed ha in generale una buona comprensione sul funzionamento delle nuove tecnologie. Egli inoltre è un esperto dell'interazione uomo-macchina: sa progettare e valutare interfacce, la loro usabilità e accessibilità, il loro impatto sugli utenti ed ha una buona conoscenza degli utenti stessi.

Profili utente dei valutatori

Il valutatore ha messo a disposizione degli utenti i seguenti dispositivi HW:

Ambiente di test

- Notebook Asus, Intel Core i5, 8GB RAM, 15.4"

Durante la fase di valutazione si sono inoltre appoggiati all'utilizzo dei seguenti SW:

- Sistemi operativi utilizzati: Microsoft Windows 7
- Altro: emulatore di terminale 3270.

Infine per la raccolta dei dati durante l'esperimento è stato utilizzato il seguente foglio di lavoro:

Materiale di lavoro

Esso è suddiviso in più parti e comprende:

- Dati del valutatore e sistema HW e SW di riferimento;
- Principi di riferimento per l'analisi;
- Dimensioni di riferimento per i principi utilizzati;
- Foglio per l'annotazione degli errori riscontrati.

Figure 46: Materiale di lavoro

Analisi Euristica di SARA Legacy

Data:	/ /
Durata:	
Nome Valutatore:	Francesca Madeddu
Browser Utilizzato:	Explorer 7
SO Utilizzato:	Windows Seven
Sistema da valutare:	SARA Legacy

DIMENSIONI DELL'USABILITÀ

Apprendibilità	facilità nell'apprendere il comportamento del sistema.
Efficienza d'uso	il sistema deve essere efficiente da usare, cosicché quando l'utente lo ha appreso, sia possibile un alto livello di produttività.
Flessibilità	rispetto alla molteplicità di modalità di interazione; facile da ricordare.
Robustezza	il sistema deve indurre ad un basso tasso di problemi, facili da correggere con minimo impatto sul costo.
Soddisfazione	il sistema deve essere gratificante da usare.

PRINCIPI DI RIFERIMENTO

1	Far vedere lo stato del sistema
2	Adeguare il sistema al mondo reale
3	Controllo dell'utente e libertà
4	Assicurare consistenza
5	Riconoscimento piuttosto che aiuto della memoria
6	Assicurare flessibilità ed efficienza d'uso
7	Visualizzare tutte e sole le informazioni necessarie
8	Prevenire gli errori
9	Permettere all'utente di correggere gli errori e non solo rilevarli
10	Fornire aiuto e documentazione

(a) Dati anagrafici e principi di riferimento

Errori trovati in fase di analisi

Errore #	
Mappa:	Area:
Principio violato:	Dimensione dell'errore:
Descrizione:	

Errore #	
Mappa:	Area:
Principio violato:	Dimensione dell'errore:
Descrizione:	

Errore #	
Mappa:	Area:
Principio violato:	Dimensione dell'errore:
Descrizione:	

Errore #	
Mappa:	Area:
Principio violato:	Dimensione dell'errore:
Descrizione:	

(b) Foglio per le annotazioni

A.1 RISULTATI DELL'ANALISI EURISTICA

Ad ogni problema è stato assegnato un livello di gravità da 1 a 3.

1. lieve: il problema, seppur presente, non pregiudica l'interazione tra sistema e utente
2. medio: il problema causa frustrazione, ma non pregiudica l'interazione tra sistema e utente
3. molto grave: il problema pregiudica un'interazione corretta tra sistema e utente

	Problema	Principio Violato	Area	Gravità problema
Tutte le sezioni (Problematiche comuni)				
1	I messaggi di stato non hanno caratteristiche che richiamano l'attenzione (come il colore) ed è facile trascurarli confondendoli con il testo del testo.	1	Soddisfazione	2
2	Le associazioni relative ai colori non sempre sono significative e coerenti. In particolare i colori utilizzati sono <ul style="list-style-type: none"> - Blu: nome mappa, spiegazione comandi, data, nome utente, filiale - Verde: per i campi - Bianco: il resto 	4	Apprendibilità Robustezza	1
3	Quando l'applicazione va in errore i messaggi sono poco significativi	1-10	Robustezza, Soddisfazione	2
4	Non è chiaro come ricorrere ad una funzione di help sia per le funzionalità dell'applicazione che per i comandi	10	Apprendibilità Robustezza	3
Login				
1	E' necessario ricordare i comandi di pulizia dello schermo (CTRL+C) e di login dell'applicazione (SA00 + Enter)	5-10	Flessibilità Soddisfazione	2
2	Il meccanismo di login è complicato e poco intuitivo	5-10	Robustezza	1
Ricerca				
1	Il codici degli stati non sono descrittivi e conducono l'utente all'errore.	8-9	Robustezza	2
2	Ogni volta che l'utente deve cercare la descrizione di un codice deve richiamare l'apposita funzione	6	Efficienza d'uso	2
3	Non è possibile sapere a priori se i campi devono essere valorizzati con "descrizioni" piuttosto che con codici"	8-10	Robustezza, Flessibilità	1
4	E' permessa la ricerca di range di date/importi che genera errore una volta effettuata la ricerca	8	Efficienza	1
5	Mancano adeguati controlli sui campi in fase di compilazione	8	Robustezza	1
6	Se vengono compilati due campi in modo scorretto la segnalazione di errore riguarda solo uno dei due	8-9	Robustezza	1

Lista				
1	Non è chiaro in che modo debba essere effettuata la selezione di un elemento della lista	8-10	Robustezza, Soddifazione	3
2	E' necessario ricordare la corrispondenza codice/descrizione per l'interpretazione del risultato	5-6	Efficienza d'uso	2
3	La formattazione non aiuta la comprensione visuale (secondo la legge della Gestalt della vicinanza sono possibili inferenze scorrette)	10	Soddifazione, Robustezza	1
4	Una volta effettuata una ricerca i "parametri di ricerca" non sono più editabili	6	Efficienza d'uso	1
5	L'utente è obbligato a scorrere tra i risultati della lista una pagina alla volta	3	Flessibilità	1

Dettaglio				
1	E' necessario ricordare la corrispondenza codice descrizione	5-6	Efficienza d'uso	2
2	A causa della dimensione fissa dei campi (la mappa è 23x80 char) le etichette sono "troncate" e non sempre descrittive	2	Robustezza, Soddifazione	1

Problemi rilevati nella parte di Logout

Logout				
1	Non è indicato come fare il logout dell'applicazione	3-10	Soddifazione	2

ESPERIMENTO CON UTENTI

Per l'esperimento con gli utenti sono stati scelti sei differenti test di compito: agli utenti è stato chiesto di svolgere compiti specifici per esercitare le funzioni del sistema usate più frequentemente.

Durante l'esperimento il valutatore si preoccupa di rilevare le quattro misure fondamentali:

Misure di valutazione

- il tempo in secondi speso per effettuare ogni compito
- il numero di click dell'utente
- le risposte esatte e quelle sbagliate
- utente arreso.

Durante l'esperimento sono stati coinvolti 20 utenti con le seguenti caratteristiche:

Profilo utente

- età: 22-56
- lingua madre: italiano
- conoscenza lingua inglese: da nessuna a buona.
- titolo di studio: da licenza media a laurea specialistica
- conoscenza del PC: da scarsa a professionale

I quattro valutatori hanno messo a disposizione degli utenti i seguenti dispositivi HW:

Ambiente di test

- Notebook Asus, Inter Core i5, 8GB RAM, 15.4"

Durante la fase di valutazione si sono inoltre appoggiati all'utilizzo dei seguenti SW:

- Sistemi operativi utilizzati: Microsoft Windows 7
- Browser utilizzati: Internet Explorer 8.
- Altro: emulatore di terminale 3270.

Sono stati assegnati sei diversi compiti, uno per ogni utente che ha partecipato all'esperimento:

Task assegnati

- Effettua il login.

- Verifica tramite l'inquiry negoziati del giorno quanti sono gli assegni trattati per il periodo di tempo dal 2004 al 2006
- Per gli assegni negoziati vedere il dettaglio di un assegno con procedura stanza (date 2002/04).
- Per gli assegni tratti cercare quelli con date 2002/04 caratterizzati dallo stato "anomalo".
- Individuare tra le rimesse la data contabile e la data rimessa di un assegno negoziato batch (date 2002/04).
- Per gli assegni smarriti tra il 2004 e il 2004 verificare il numero di assegni con tipo di procedura check truncation.
- Effettua il logout.

Materiale di lavoro

Per la raccolta dei dati durante l'esperimento è stato utilizzato il seguente foglio di lavoro:

Test e Questionari per l'analisi dell'usabilità

Data: / /

Nome Valutatore: Francesca Madeddu

Piattaforma di supporto: Emulatore di terminale 3270

SO Utilizzato: Windows Seven

Sistema da valutare: SARA Legacy

PROFILO UTENTE

Nome: _____

Cognome: _____

Titolo di studio: _____

Usa il computer per lavoro: ☐ sì ☐ no ☐ poco

• se Sì quante ore al giorno: _____

Usa il computer nel tempo libero: ☐ sì ☐ no ☐ poco

• se Sì quante ore al giorno? _____

Usa il computer per navigare sul web ☐ sì ☐ no ☐ poco

• se Sì quante ore al giorno? _____

TEST USABILITA': TASK

Eseguire i seguenti test di compito:	Tempo (s)	(*) Numero interazioni	Risposte sbagliate	Utente arreso
Effettua il login.				
Verifica tramite l'inquiry negoziati del giorno quanti sono gli assegni trattati per il periodo di tempo dal 2004 al 2006				
Per gli assegni negoziati vedere il dettaglio di un assegno con procedura stanza (con date di riferimento 2002-2004).				
Per gli assegni tratti cercare quelli con date comprese dal 2002 al 2004 caratterizzati dallo stato "anomalo".				
Individuare tra le rimesse la data contabile e la data rimessa di un assegno negoziato batch (con date di riferimento 2002-2004).				
Per gli assegni smarriti tra il 2004 e il 2004 verificare il numero di assegni con tipo di procedura check truncation.				
Effettua il logout				

QUESTIONARIO: Pt.1					
<i>Esprimere un grado di preferenza per le seguenti affermazioni.</i>					
	Molto d'accordo				Disaccordo
L'applicazione è molto interessante	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
E' difficile muoversi all'interno dell'applicazione.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Posso facilmente trovare quello che voglio all'interno dell'applicazione.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
L'applicazione mi pare caratterizzato da una logica ben definita.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
L'applicazione ha bisogno di più spiegazioni preliminari.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Le schermate di questa applicazione sono molto attraenti.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Quando navigo all'interno dell'applicazione sento che ne ho il controllo.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Questa applicazione è troppo lenta.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
L'applicazione mi aiuta a trovare quello che sto cercando.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Imparare a trovare il mio modo per navigare questa applicazione è problematico.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

QUESTIONARIO: Pt.2					
<i>Esprimere un grado di preferenza per le seguenti affermazioni.</i>					
	Molto d'accordo				Disaccordo
Usare questa applicazione non mi piace.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Mentre uso questa applicazione sento che il mio lavoro è efficiente.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
E' difficile dire se l'applicazione ha quello che voglio.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Usare per la prima volta l'applicazione è semplice.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
L'applicazione ha caratteristiche fastidiose.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ricordarmi dove sono, all'interno dell'applicazione, è difficile.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Usare questa applicazione è una perdita di tempo.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Quando clicco sugli oggetti dell'applicazione ottengo quello che mi aspetto.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

(b) Questionario a risposta multipla

Figure 47: Materiale di lavoro

Esso è suddiviso in più parti e comprende:

- Un questionario che permette l'individuazione del profilo utente (età, conoscenza informatiche, titolo di studio, ecc.) - Una tabella usata dal valutatore per identificare il compito assegnato e per raccogliere le misure dell'esperimento (numero di click, tempo totale, ecc.)
- Un questionario di 20 domande riguardante l'esperienza d'uso. Prevede risposte che permettono all'utente di esprimere accordo o disaccordo in merito alla domanda posta. ¹
- Uno spazio per eventuali commenti del valutatore durante l'esecuzione dell'esperimento

¹ Fonte delle domande: <http://www.wammi.com/questionnaire.html>

- Una tabella finale riassuntiva che raccoglie i dati ottenuti da ciascun valutatore nei sei esperimenti.

B.O.1 Risultati test usabilità Sara Legacy

L'applicazione è molto interessante

Molto d'accordo	0%	Molto d'accordo	0%
D'accordo	23%	D'accordo	60%
Così così	50%	Così così	30%
Disaccordo	27%	Disaccordo	10%
Molto disaccordo	0%	Molto disaccordo	0%

(a) Applicazione legacy

(b) Applicazione modernizzata

E' difficile muoversi all'interno dell'applicazione.

Molto d'accordo	70%	Molto d'accordo	0%
D'accordo	20%	D'accordo	0%
Così così	10%	Così così	10%
Disaccordo	0%	Disaccordo	90%
Molto disaccordo	0%	Molto disaccordo	0%

(a) Applicazione legacy

(b) Applicazione modernizzata

Posso facilmente trovare quello che voglio all'interno dell'applicazione.

Molto d'accordo	10%	Molto d'accordo	0%
D'accordo	50%	D'accordo	40%
Così così	25%	Così così	50%
Disaccordo	15%	Disaccordo	1%
Molto disaccordo	0%	Molto disaccordo	0%

(a) Applicazione legacy

(b) Applicazione modernizzata

L'applicazione mi pare caratterizzato da una logica ben definita.

Molto d'accordo	20%	Molto d'accordo	10%
D'accordo	70%	D'accordo	90%
Così così	10%	Così così	0%
Disaccordo	0%	Disaccordo	0%
Molto disaccordo	0%	Molto disaccordo	0%

(a) Applicazione legacy

(b) Applicazione modernizzata

L'applicazione ha bisogno di più spiegazioni preliminari.

Molto d'accordo	90%	Molto d'accordo	0%
D'accordo	10%	D'accordo	0%
Così così	0%	Così così	20%
Disaccordo	0%	Disaccordo	80%
Molto disaccordo	0%	Molto disaccordo	0%

(a) Applicazione legacy

(b) Applicazione modernizzata

Le schermate di questa applicazione sono molto attraenti.

Molto d'accordo	0%	Molto d'accordo	0%
D'accordo	0%	D'accordo	50%
Così così	20%	Così così	50%
Disaccordo	80%	Disaccordo	0%
Molto disaccordo	0%	Molto disaccordo	0%

(a) Applicazione legacy

(b) Applicazione modernizzata

Quando navigo all'interno dell'applicazione sento che ne ho il controllo.

Molto d'accordo	0%	Molto d'accordo	10%
D'accordo	0%	D'accordo	80%
Così così	20%	Così così	10%
Disaccordo	70%	Disaccordo	0%
Molto disaccordo	10%	Molto disaccordo	0%

(a) Applicazione legacy

(b) Applicazione modernizzata

Questa applicazione è troppo lenta.

Molto d'accordo	0%	Molto d'accordo	0%
D'accordo	0%	D'accordo	10%
Così così	10%	Così così	30%
Disaccordo	90%	Disaccordo	60%
Molto disaccordo	0%	Molto disaccordo	0%

(a) Applicazione legacy

(b) Applicazione modernizzata

L'applicazione mi aiuta a trovare quello che sto cercando.

Molto d'accordo	10%	Molto d'accordo	0%
D'accordo	0%	D'accordo	60%
Così così	0%	Così così	40%
Disaccordo	70%	Disaccordo	0%
Molto disaccordo	20%	Molto disaccordo	0%

(a) Applicazione legacy

(b) Applicazione modernizzata

Imparare a trovare il mio modo per navigare questa applicazione è problematico.

Molto d'accordo	0%	Molto d'accordo	0%
D'accordo	15%	D'accordo	10%
Così così	50%	Così così	20%
Disaccordo	20%	Disaccordo	60%
Molto disaccordo	10%	Molto disaccordo	10%

(a) Applicazione legacy

(b) Applicazione modernizzata

Usare questa applicazione non mi piace. Risultati test usabilità di SARA Legacy

Molto d'accordo	10%	Molto d'accordo	0%
D'accordo	15%	D'accordo	0%
Così così	50%	Così così	30%
Disaccordo	20%	Disaccordo	60%
Molto disaccordo	5%	Molto disaccordo	10%

(a) Applicazione legacy

(b) Applicazione modernizzata

*Mentre uso questa
applicazione sento
che il mio lavoro è
efficiente.*

Molto d'accordo	0%
D'accordo	0%
Così così	45%
Disaccordo	50%
Molto disaccordo	5%

(a) Applicazione legacy

Molto d'accordo	10%
D'accordo	80%
Così così	5%
Disaccordo	5%
Molto disaccordo	0%

(b) Applicazione modernizzata

*E' difficile dire se
l'applicazione ha
quello che voglio.*

Molto d'accordo	5%
D'accordo	0%
Così così	40%
Disaccordo	50%
Molto disaccordo	5%

(a) Applicazione legacy

Molto d'accordo	0%
D'accordo	30%
Così così	50%
Disaccordo	20%
Molto disaccordo	0%

(b) Applicazione modernizzata

*Usare per la prima
volta l'applicazione è
semplice.*

Molto d'accordo	0%
D'accordo	0%
Così così	0%
Disaccordo	0%
Molto disaccordo	100%

(a) Applicazione legacy

Molto d'accordo	10%
D'accordo	80%
Così così	10%
Disaccordo	0%
Molto disaccordo	0%

(b) Applicazione modernizzata

*L'applicazione ha
caratteristiche
fastidiose.*

Molto d'accordo	5%
D'accordo	70%
Così così	20%
Disaccordo	0%
Molto disaccordo	5%

(a) Applicazione legacy

Molto d'accordo	0%
D'accordo	0%
Così così	10%
Disaccordo	80%
Molto disaccordo	10%

(b) Applicazione modernizzata

*Ricordarmi dove
sono, all'interno
dell'applicazione, è
difficile.*

Molto d'accordo	0%
D'accordo	0%
Così così	10%
Disaccordo	70%
Molto disaccordo	20%

(a) Applicazione legacy

Molto d'accordo	0%
D'accordo	0%
Così così	45%
Disaccordo	55%
Molto disaccordo	0%

(b) Applicazione modernizzata

*Usare questa
applicazione è una
perdita di tempo.*

Molto d'accordo	0%
D'accordo	0%
Così così	45%
Disaccordo	55%
Molto disaccordo	0%

(a) Applicazione legacy

Molto d'accordo	0%
D'accordo	5%
Così così	10%
Disaccordo	60%
Molto disaccordo	25%

(b) Applicazione modernizzata

*Quando clicco sugli
oggetti
dell'applicazione
ottengo quello che
mi aspetto.*

Molto d'accordo	0%	Molto d'accordo	5%
D'accordo	30%	D'accordo	90%
Così così	60%	Così così	5%
Disaccordo	10%	Disaccordo	0%
Molto disaccordo	0%	Molto disaccordo	0%

(a) Applicazione legacy

(b) Applicazione modernizzata

*All'interno
dell'applicazione
ogni cosa è facile da
capire.*

Molto d'accordo	0%	Molto d'accordo	0%
D'accordo	0%	D'accordo	70%
Così così	40%	Così così	20%
Disaccordo	60%	Disaccordo	10%
Molto disaccordo	0%	Molto disaccordo	0%

(a) Applicazione legacy

(b) Applicazione modernizzata

BIBLIOGRAPHY

- [1] URL <http://www.butlergroup.ie/>. Available on line. (Citato a pagina 26.)
- [2] URL <http://it.compuware.com/>. Available on line. (Citato a pagina 27.)
- [3] *Reliable computer systems : design and evaluation*. A. K. Peters, 1998. (Citato a pagina 23.)
- [4] Screen scraping: a wolf in sheep's clothin. *Communications of the ACM*, 2004. (Citato a pagina 34.)
- [5] Strategic approach to modernize your legacy systems and wreck the business bottlenecks. 2006. (Citato a pagina 30.)
- [6] Enterprise legacy modernization (elm). 2007. (Citato a pagina viii and 30.)
- [7] Top five myths of screen-scraping. 2008. (Citato a pagina viii and 35.)
- [8] ISO/IEC 14764. Software Engineering, Software life cycle processes, Maintenance. page 1, 2006. (Citato a pagina 8.)
- [9] D. E. et al Ansuni. *Sistemi Informativi: sistemi distributi*. FrancoAngeli, Milano, 2001. (Citato a pagina 19, 32, 34, and 36.)
- [10] K Bennet. Legacy systems: Coping with success. *Communications of the ACM*, (12):19–23, 1995. (Citato a pagina 20.)
- [11] Tim Berners-Lee. URL <http://www.w3.org/WAI/>. Available on line. (Citato a pagina 95.)
- [12] D. Wu B. Bisbal, J. Lawless and J. Grimson. Legacy information system migration: A brief review of problems, solutions and research issues. *Communications of the ACM*, (16):33–11, 1999. (Citato a pagina 20.)
- [13] G. et al. Booch. *The Unified Modeling Language User Guide*. Pearson Education, 2th edition, 2005. (Citato a pagina 52.)
- [14] G. Brajnik and E. Toppano. *Creare siti web multimediali. Fondamenti per l'analisi e la progettazione*. Pearson, 2007. (Citato a pagina 62, 63, and 65.)
- [15] M. Jazayeri e D. Mandrioli C. Ghezzi. *Ingegneria del Software*. Pearson Education Italia, 2nd edition, 2004. (Citato a pagina 14.)

- [16] J.M. Carroll. *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science*. Morgan Kaufmann series in interactive technologies. Elsevier Science, 2003. (Citato a pagina 67.)
- [17] E. Chikofsky and J Cross. Reverse engineering and design recovery: A taxonomy. *Communications of the ACM*, 7(1):13–18, 1990. (Citato a pagina 31.)
- [18] B.C. Cole. *The Emergence of Net-Centric Computing: Network Computers, Internet Appliances, and Connected PCs*. Prentice Hall, 1999. (Citato a pagina 11.)
- [19] Politecnico di Milano. URL <http://www.webml.org/>. Available on line. (Citato a pagina 49 and 53.)
- [20] S. Di Nuovo. *La valutazione dell'attenzione. Dalla ricerca sperimentale ai contesti applicativi*. Serie di psicologia. FrancoAngeli, 2006. (Citato a pagina 69.)
- [21] S. Cornella-Dorda et al. *A Survey of Legacy System Modernization Approaches*. Carnegie Mellon University, 2000. (Citato a pagina 11, 22, and 31.)
- [22] M.S. Gazzaniga and E. Bizzi. *The New cognitive neurosciences*. Bradford books. A Bradford Book, 2000. (Citato a pagina 67.)
- [23] D Good. Legacy transformation. 2002. (Citato a pagina 21 and 30.)
- [24] C Guerindon. *Continuous Flow Manufacturing: Quality in Design and Processes*. Marcel Dekker Inc, New York, 1995. (Citato a pagina 12.)
- [25] S. Gundavaram. *CGI Programming on the World Wide Web*. Nutshell Handbook. O'Reilly Vlg. GmbH & Company, 1996. (Citato a pagina 34.)
- [26] R. E Johnson. Software reengineering assessment handbook, us department of defense. 1997. (Citato a pagina 33 and 35.)
- [27] A.G. Kleppe, J.B. Warmer, and W. Bast. *Mda Explained, the Model Driven Architecture: Practice and Promise*. The Addison-Wesley Object Technology Series. ADDISON WESLEY Publishing Company Incorporated, 2003. (Citato a pagina 46.)
- [28] G. D Lucca. *Legacy systems*. 2003. (Citato a pagina 19.)
- [29] S.J. Mellor. *Mda Distilled: Principles of Model-Driven Architecture*. The Addison-Wesley Object Technology Series. ADDISON WESLEY Publishing Company Incorporated, 2004. (Citato a pagina 46.)

- [30] Microsoft. The business value of legacy modernization. *Communications of the ACM*, 2007. (Citato a pagina 21.)
- [31] E Mitleton-Kelly. *It Legacy Systems: Enabling environments that reduce the legacy problem, a complexity perspective*. Wiley, Edinburg Gate, 8th edition, 2004. (Citato a pagina 20.)
- [32] J. Nielsen, H. Loranger, and W. Vannini. *Web Usability 2.0*. Apogeo, 2006. (Citato a pagina 62 and 64.)
- [33] D.A. Norman and B. Parrella. *Il computer invisibile. La tecnologia migliore è quella che non si vede*. Apogeo Saggi. Apogeo, 2005. (Citato a pagina 69.)
- [34] R. M. Olsem. *An Incremental Approach to Software Systems Re-engineering*. John Wiley and Sons, 1998. (Citato a pagina 33 and 41.)
- [35] L. D. Olson and D. D Wu. *Enterprise Risk Management*. World Scientific Publishing, Edinburg Gate, 2008. (Citato a pagina 37.)
- [36] S. Plakosh, D. Hissam and K Wallnau. Into the black box: A case study in obtaining visibility into commercial software. *cmu/sei-99-tn-010*. (Citato a pagina 32.)
- [37] J. et al. Preece. *Interaction Design*. Apogeo, 2th edition, 2004. (Citato a pagina 49.)
- [38] S. Price and G Waite. Oracle forms to soa: A case study in modernization. 2008. (Citato a pagina 21.)
- [39] Schach R. *Software Engineering*. McGraw-Hill, Bostin, 4th edition, 1999. (Citato a pagina 14.)
- [40] S Reiss. Incremental maintenance of software artifacts. (32):9, 2006. (Citato a pagina 20.)
- [41] F. Ricca and P Tonella. *Software systems reengineering: limits and capabilities*. Mondo Digitale, Edinburg Gate, 8th edition, 2006. (Citato a pagina 20.)
- [42] A Rodger. Cobol, continuing to drive value in the 21th century. 2008. (Citato a pagina 26.)
- [43] J. Russell and R. Cohn. *Ibm 3270*. Book on Demand, 2012. (Citato a pagina 59.)
- [44] K. A. Saleh. *Software Engineering*. Ross Publishing Inc, 2009. (Citato a pagina 8.)

- [45] R. C. et al Seacord. Legacy system modernization strategies, software engineering institute. 2001. (Citato a pagina [viii](#), [33](#), [34](#), [41](#), and [45](#).)
- [46] Lewis G A Seacord R C, Plackosh D. *Modernizing Legacy System: software technologies, engineering processes, and business practices*. Addison-Wesley Professional, 2003. (Citato a pagina [10](#), [11](#), [12](#), [19](#), [32](#), [33](#), [35](#), and [37](#).)
- [47] Ian Sommerville. *Software Engineering*. Pearson Education Limited, Edinburg Gate, 8th edition, 2007. (Citato a pagina [7](#), [8](#), [14](#), [37](#), and [51](#).)
- [48] Web Models s.r.l. WebRatio. URL <http://www.webratio.com/>. Available on line. (Citato a pagina [45](#).)
- [49] Lucio Stanca. Disposizioni per favorire l'accesso dei soggetti disabili agli strumenti informatici, 2004. URL <http://www.camera.it/parlam/leggi/040041.htm>. Available on line. (Citato a pagina [95](#).)
- [50] M Sunderkotter. Software engineering risk management. 2004. (Citato a pagina [37](#).)
- [51] S. R. Tilley and D. B Smith. Perspectives on legacy system reengineering. *Communications of the ACM*, 1995. (Citato a pagina [32](#).)
- [52] H. Tromp and G Hoffman. Evolution of legacy systems : strategic and technological issues, based on a case study published in proceedings of elisa. *International Workshop on Evolution of Large-scale Industrial Software Applications*, 2003. (Citato a pagina [19](#).)
- [53] W3C. Wcag 2.0, 2008. URL <http://www.w3.org/TR/WCAG/>. Available on line. (Citato a pagina [96](#).)
- [54] I. Warren and J. Ransom. Renaissance: A method to support software system evolution. *Computer Software and Applications Conference, Annual International*. (Citato a pagina [11](#).)
- [55] G Withey. Investment analysis of software assets for product lines. 1996. (Citato a pagina [38](#).)