Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica

Relazione finale di tirocinio

La creazione del valore. Un approccio agile alla trasformazione dell'IT

Laureando: Riccardo Birello

Relatore: Prof. Michele Moro

Azienda ospitante: Larus Business Automation s.r.l.

Tutor aziendale: Dott. Alessandro Polo

Anno Accademico: 2010/2011

Indice

Parte I

Sommario

Sommario

La seguente tesi rappresenta il frutto dell'attività di tirocinio svolta presso l'azienda Larus Business Automation s.r.l. di Mestre (VE).

Scopo della tesi è quello di descrivere l'adozione di metodologie agili nell'ambito di un percorso di trasformazione della struttura IT di una grande azienda, percorso nel quale Larus Business Automation è stata - ed è - coinvolta come partner tecnico (Technical Coordinator).

In particolare, vengono esaminati gli aspetti inerenti all'erogazione di valore che qualificano l'IT quale componente abilitante del business e si è cercato di dimostrare come - per dare risposta alla rapida e continua evoluzione di quest'ultimo - anche le metodologie di sviluppo del software siano cambiate, sino alla formulazione del modello agile. Attraverso l'analisi di un caso reale di applicazione di tale modello, si è tentato di valutarne limiti e punti di forza e, soprattutto, di capire le ragioni di quell'efficacia - ed efficienza - che lo hanno reso un punto di riferimento nel mondo dell'informatica professionale.

Il testo è articolato in due parti. La prima vuole fornire - sotto un profilo "teorico" - una panoramica delle principali metodologie di sviluppo software, da quando lo stesso concetto è nato sino ad oggi.

La seconda tratta, invece, del caso di studio reale sopra citato.

Parte II

Introduzione

1 Origine

All'inizio dell'informatica, lo sviluppo di software era un'attività caotica caratterizzata dal cosiddetto approccio *code and fix*. Questo termine si può tradurre come una programmazione priva di organizzazione e di direttive riguardante il controllo dei tempi, dei costi e della qualità, il cui obiettivo è quello di capire approssimativamente quale sarà la risposta finale del software richiesta e di tentare ripetutamente di generare codice e correggere gli eventuali errori. Cose inimmaginabili al giorno d'oggi visto l'andamento del mercato e l'altissima concorrenza nel settore.

L'approccio *code and fix* funziona bene soltanto per software di piccole dimensioni, ma al crescere del sistema diventa molto difficile da gestire; basti pensare alle difficoltà che si potrebbero riscontrare nell'apportare modifiche o nell'aggiungere funzionalità, mantenendo la stessa qualità. Si sarebbe infatti costretti ad impiegare la maggior parte del tempo nella correzione piuttosto che nell'implementazione delle stesse funzioni.

Nell'ottica di un'azienda ci si accorge subito che i costi legati alla correzione e, quindi, non all'evoluzione del software, diventano elevati e, a volte, insostenibili.

Gli esperti di informatica si accorsero della necessità di apportare delle modifiche nel processo di implementazione di software, favorendo così la nascita dell'ingegneria del software.

1.1 Ingegneria del software

Qualunque *industria* manifatturiera ha un modello per la produzione dei beni che consente di pianificare le proprie attività e le risorse necessarie, oltre a prevedere e controllare i costi del processo e la qualità dei prodotti. Senza un modello, questo non sarebbe possibile.

I primi esperti informatici proposero allora delle metodologie per risolvere questi problemi in modo da dare una linea guida agli sviluppatori per rendere più efficiente lo sviluppo di nuovi software.

Si inizia così a parlare di una vera e propria disciplina: l'**ingegneria del software** che si occupa dei processi produttivi e delle metodologie di sviluppo per realizzare sistemi software. Essa ha come obiettivo l'evoluzione dello sviluppo del software sia da un punto di vista tecnologico (per esempio attraverso la definizione di nuovi linguaggi di programmazione più performanti) sia da un punto di vista metodologico (per esempio il perfezionamento dei modelli di ciclo di vita del software), facendo particolare attenzione ai costi per applicare la conoscenza scientifica. [4]

> *"L'ingegneria del software è l'istituzione e l'impiego di principi ingegneristici ben fondati, allo scopo di ottenere in modo economico software affidabile ed efficiente su macchine vere."*
>
> *Fritz Bauer*

1.1.1 Ciclo di vita del software

Con l'ingegneria del software, si portò alla luce il concetto di **ciclo di vita del software** (figura 1), che possiamo definire come una descrizione della *produzione industriale* del software, dalla sua concezione iniziale fino al suo sviluppo completo, al suo rilascio e alla sua successiva evoluzione.

Quando si realizza un prodotto è importante svolgere una serie di passi prevedibili, una sorta di percorso guidato che aiuti ad ottenere risultati di alta qualità nel tempo prefissato (**On Target, On Time, On Budget**).

Con il termine ciclo di vita del software si indica il modo con cui una metodologia o un modello scompone l'attività di realizzazione di prodotti software in sotto-attività fra loro coordinate e comprende generalmente almeno le seguenti attività:

- **Definizione degli obiettivi**, ovvero le finalità del progetto.
- **Analisi delle esigenze e della fattibilità**, cioè l'espressione, la raccolta e la formalizzazione dei bisogni del richiedente (il cliente) e dell'insieme dei limiti.
- **Concezione generale**, che riguarda l'elaborazione delle specifiche dell'architettura generale del software.
- **Concezione dettagliata**, che intende definire precisamente ogni sotto-insieme del software.
- **Codifica e Test unitari**, ossia la traduzione in un linguaggio di programmazione delle funzionalità definite in fase di concezione e la verifica che ogni sotto-insieme del software sia implementato conformemente alle specifiche.
- **Integrazione e Verifica**, con l'obiettivo di assicurare l'interfacciamento dei diversi elementi (moduli) del software e la verifica della conformità del software alle specifiche iniziali.
- **Rilascio**, cioè la consegna del prodotto al cliente.
- **Manutenzione**, che comprende tutte le azioni correttive (manutenzione correttiva) ed evolutive (manutenzione evolutiva) al software.

La sequenza e la presenza di ognuna di queste attività nel ciclo di vita dipende dalla scelta di un modello di ciclo di vita tra il cliente e la squadra di sviluppo.

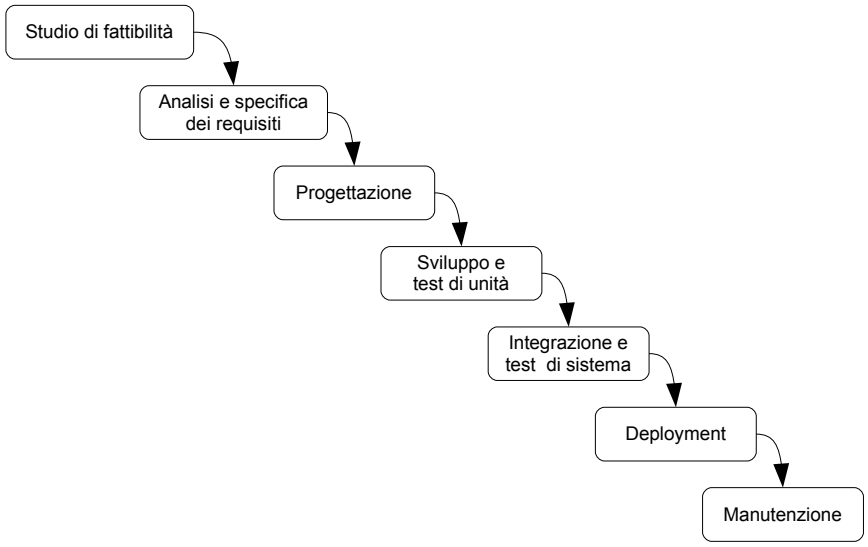

Figura 1: Il ciclo di vita del software.

L'origine di questa divisione proviene dalla constatazione che gli errori hanno un costo tanto più elevato quanto la loro rilevazione avviene tardi nel processo di realizzazione. Il ciclo di vita permette di rilevare gli errori il prima possibile e quindi di controllare la qualità del software, i tempi di realizzazione ed i costi associati. È importante introdurre stabilità, controllo e organizzazione in un'attività che, se lasciata libera, può diventare piuttosto caotica e dispendiosa in termini economici.

1.1.2 Metodologia vs Modello

Nei paragrafi precedenti sono stati usati due termini diversi, i quali possono risultare sinonimi ma che in realtà sono due concetti ben distinti. I due termini sono i seguenti [21]:

Metodologia. Una metodologia è la scienza del metodo che studia le tecniche della sistemazione e dello sviluppo delle conoscenze nell'ambito di una certa materia, nel nostro caso lo sviluppo di software. Essa si basa su certi principi che indicano lo scopo ma non le modalità con le quali raggiungerlo.

Modello. Un modello, invece, è un procedimento che segue delle regole rigide e ben definite in base alle quali si svolge un'attività teorica o pratica per raggiungere

uno scopo. Si può pensare ad un modello come al modo con cui si utilizza una metodologia di sviluppo.

1.2 La necessità di cambiare

Le cause che hanno portato alla nascita di nuove metodologie e, di conseguenza, anche di modelli che le possano applicare, sono proprio la consapevolezza di passare dal *caos* all'*ordine.*

L'evoluzione del processo di sviluppo del software è proporzionale all'aumentare della sua richiesta da parte dei clienti, che pretendono la realizzazione di un prodotto che sia conforme alle specifiche fissate, a prezzi contenuti, consegnato in breve tempo e che sia anche di alta qualità.

La diminuzione dei costi e dei tempi comportano un cambiamento/miglioramento nel processo di sviluppo e ciò è un compito che spetta alle aziende che devono trovare il modo più adatto per attuare una trasformazione nel processo di sviluppo.[4]

> *"Una persona di successo ha semplicemente trovato il modo di svolgere attività che le altre persone non sono in grado di fare."*
>
> *Dexter Yager*

Migliorare i processi di sviluppo dei sistemi software significa fare degli sforzi per far evolvere anche le metodologie che sostengono gli sviluppi. Questo, nel corso degli anni, ha portato alla nascita di diverse metodologie che hanno caratteristiche diverse e da utilizzare in base ai bisogni del progetto o del cliente.

Le categorie di metodologie presenti attualmente sono le seguenti:

- Metodologie pesanti (come il modello a cascata).
- Metodologie iterative (come il modello a spirale o il processo RUP).
- Metodologie agili.

Tutti i modelli di processo di sviluppo software possono comprendere le attività strutturali generiche che sono state descritte nel paragrafo 1.1.1, ma ognuno di essi applica un'enfasi differente a queste attività e definisce un flusso di lavoro che coinvolge ciascuna attività strutturale (e le azioni di ingegneria del software con i relativi task) in un modo differente.

Parte III

Modelli di Sviluppo del Software

2 Modello a cascata

Il modello a cascata (in inglese, waterfall model) è un *modello prescrittivo* secondo cui la realizzazione di un prodotto software consiste in una sequenza di fasi ben definita. Vengono chiamati *prescrittivi* poiché prescrivono un insieme di elementi del processo: attività strutturali, azioni di ingegneria del software, task, risultati, valutazione della qualità, meccanismi di controllo delle modifiche per ciascun progetto, prescrivendo, inoltre, il modo in cui essi vengono correlati fra loro. [4]

2.1 Caratteristiche

Il modello a cascata è stato adottato dagli informatici all'inizio degli anni '70 come primo metodo di lavoro con lo scopo di diminuire i costi relativi allo sviluppo, rispetto alle pratiche precedenti, e di iniziare ad adoperare un certo schema di lavoro.

Il modello a cascata è, probabilmente, il processo di sviluppo software più diffuso nel mondo perché ricorda la catena di montaggio tipica della produzione industriale, anche se non presenta una struttura a pipeline (sviluppo in parallelo). Esso suggerisce un approccio sistematico e strettamente sequenziale allo sviluppo di software nel quale ciascuna fase produce un ben preciso output (*deliverable*) che viene utilizzato come input per la fase successiva (da cui deriva la metafora della cascata). La grande quantità di documentazione è necessaria per istruire le persone che lavoreranno nella fase successiva e che non conoscono bene i requisiti[1]. [21]

La caratteristica principale di questo modello è la sequenza rigida delle fasi, che comporta la completa assenza di sovrapposizioni tra di loro, e l'assenza di ricicli, cioè l'impossibilità di ritornare ad una delle fasi precedenti per modificare qualcosa.

Un'ulteriore caratteristica è quella della convinzione che fosse possibile progettare correttamente l'applicazione sin da subito, grazie anche alla sostanziale stabilità dei requisiti per i software dell'epoca, fatto che, oggigiorno, è altamente improbabile.

2.2 Fasi del modello a cascata

Il modello a cascata mette in risalto principalmente quattro fasi (vedere la figura 2) che risultano fondamentali anche per tutte le altre metodologie conosciute.

[1]Per requisito si intende un bisogno/esigenza di un utente/cliente il quale necessita di una risposta da parte del software.

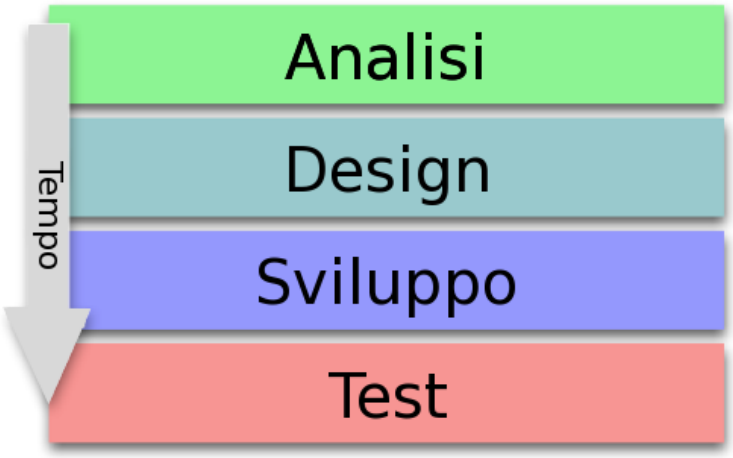

Figura 2: Rappresentazione delle fasi del processo di sviluppo del metodo a cascata che vengono eseguite in sequenza. [11]

Le fasi del modello sono le seguenti: [21]

Analisi dei requisiti

Identifica i requisiti generali dell'intero sistema (hardware, software, fonti di informazioni, persone, ...) ed il (sotto)insieme dei requisiti che dovranno essere realizzati nel prodotto software. I requisiti sono i bisogni che ha il cliente e che il software deve soddisfare. Esso produce un documento scritto che vincola il prodotto da sviluppare in base alle specifiche e che deve essere il più completo e preciso possibile perché su di esso si basa tutto il lavoro futuro che verrà fatto; un errore in questa fase potrebbe compromettere la riuscita del progetto.

Progettazione

Determina come verrà sviluppato il sistema secondo quanto stabilito nell'analisi dei requisiti, definendo l'architettura (hardware e software) del sistema ed identificando tutti i componenti da realizzare o modificare.

Sviluppo

È l'implementazione vera e propria del sistema con un linguaggio di programmazione per ogni funzionalità richiesta dal cliente.

Collaudo

Verifica la correttezza dell'implementazione del sistema e se le specifiche richieste vengono soddisfatte. Un ulteriore aspetto del collaudo è l'attività di *Integrazione* che serve per accertare la correttezza del funzionamento complessivo del sistema. Il test di sistema complessivo deve individuare gli errori presenti e assicurare che l'applicazione agisca come previsto. Si distingue a volte tra *alpha test* (cioè quando il sistema è rilasciato per l'uso interno dell'organizzazione del produttore) e *beta test* (cioè quando si ha un rilascio controllato a pochi e selezionati utenti del prodotto).

2.3 Analisi critica

Il modello a cascata è una metodologia molto corposa e offre il maggior successo quando i requisiti sono **stabili** e ben **chiari** già dall'inizio del progetto. Inoltre, è utilizzabile quando la tecnologia è ben nota, avendola magari già usata molte volte, e non si punta su qualcosa di sconosciuto. Infatti, quando viene utilizzata in contesti dinamici, destinati a cambiare rapidamente e di cui non si ha nemmeno una buona conoscenza, si ha un rischio molto alto dato soprattutto dalla rigidità delle sequenze delle fasi.

Esso ha contribuito, inoltre, a fissare molti concetti fondamentali (fasi, semilavorati, ecc) e ha rappresentato un punto di partenza per lo studio dei processi software: per la prima volta la pratica dello sviluppo del software viene concepita come *processo industriale* (con le relative necessità di documentazione e controllo) anziché come *attività artigianale* (il cosiddetto approccio code and fix).

Argomenti che tuttora sono largamente presi in considerazione sono, per esempio, la disciplina (il seguire un certo modo di lavorare secondo delle regole precise), la pianificazione (la necessità di dettare dei tempi e rispettarli), ma anche la conoscenza degli obiettivi prima dell'implementazione del prodotto e la suddivisione in fasi.

L'adozione di questi principi può sembrare estremamente produttiva ma la loro applicazione ha come effetto collaterale le difficoltà di coordinamento fra le diverse attività (fasi). Assumendo che i requisiti possano essere congelati alla fine della fase di specifica (quindi non modificabili), si deve produrre dopo ciascuna attività una grande quantità di documentazione per rendere il più chiaro possibile ai colleghi il lavoro da svolgere nella fase successiva. La difficoltà di *capire bene cosa fare* è una delle principali cause che fanno alzare la quantità di errori commessi.

Questo problema è messo in evidenza anche dall'assenza di qualsiasi forma di feedback, sia tra le attività e le persone che svolgono i lavori, ma anche con il cliente,

con il quale si ha dei contatti solamente all'inizio del progetto (fase di specifiche) e alla fine quando si consegna il prodotto.

Un'ulteriore difficoltà di questo modello è la rigidità della sua applicazione: è necessario infatti terminare completamente una fase prima di poter iniziare quella successiva. La suddivisione inflessibile del progetto in fasi distinte rende difficile rispondere alle eventuali richieste di cambiamento dei requisiti da parte del cliente. È proprio per questo motivo che questo modello è adatto solo se i requisiti sono ben chiari fin dall'inizio ed è difficile che cambino durante lo sviluppo.

Qualsiasi difficoltà o modifica dei requisiti porta a dei ritardi nelle fasi e di conseguenza all'intero progetto e ciò causa l'aumentare dei costi dello sviluppo e la posticipazione dell'uscita sul mercato (*time to market*).

Inoltre, il cosiddetto time-to-market è un ulteriore punto dolente di questa metodologia. Il tempo che può passare dalla commissione del progetto alla sua consegna al cliente può durare anche anni e questo può essere un problema perché un prodotto può essere già *vecchio* oppure obsoleto dopo così tanto tempo.

Un ulteriore effetto collaterale è l'incapacità di poter stimare le risorse e i costi in maniera accurata finché non sia stata svolta almeno la prima fase di analisi.

3 Processo RUP (Rational Unified Process)

Il Rational Unified Process (RUP) è un processo di sviluppo, o meglio un *framework di processo*[2], di tipo iterativo ed incrementale definito da Booch, Rumbaugh, Jacobson (autori dell'UML[3]) e da adattare alle diverse tipologie di progetto. L'articolazione di un progetto iterativo è guidata non da una rigida sequenza di fasi predefinite, ma da una gestione sistematica dei rischi di progetto (vedere figura 3), caratteristica anche del modello a spirale, per arrivare alla loro progressiva diminuzione.

Figura 3: Il modello a spirale.

3.1 Caratteristiche

Il RUP utilizza UML come un linguaggio per la specifica di molti artefatti del processo ed è basato su delle "Best Practices", cioè degli approcci allo sviluppo di sistemi software largamente utilizzati nell'industria. Incoraggia il controllo della qualità e la gestione del rischio, infatti è un processo controllato in tutte le sue fasi. [21]

Esso definisce, inoltre, un framework che può essere specializzato per una ampia classe di sistemi software, aree applicative, organizzazioni, livelli di competenza e dimensioni dei progetti ma è anche un prodotto commerciale sviluppato e mantenuto dalla Rational Software, la quale fornisce un insieme di tool per automatizzare il più possibile le attività di creazione e manutenzione dei diversi artefatti del processo.

Il RUP appartiene al ciclo di vita del software iterativo ed incrementale, in modo tale da assecondare la complessità dello sviluppo ed i cambiamenti nei requisiti e per essere più flessibile. Infatti, viene posta l'enfasi sui modelli più che sui documenti in linguaggio corrente e questo li rende più leggibili e più modificabili.

[2]Un framework è un quadro di riferimento generale.

[3]L'UML (Unified Modeling Language) è un linguaggio di modellazione definito nel 1996 che definisce una raccolta di *best practices* per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile ad un vasto pubblico.

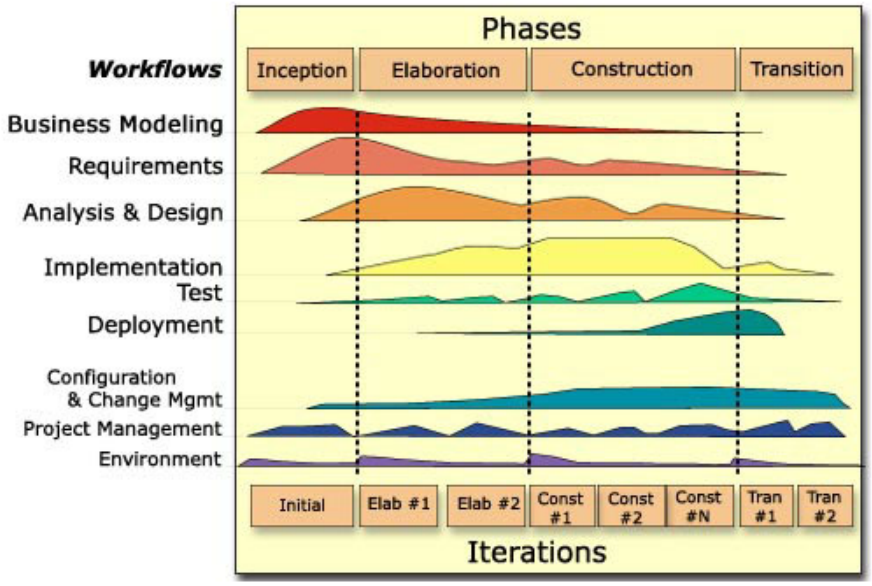

Figura 4: La struttura del processo RUP.

Grande importanza viene data, inoltre, all'architettura dell'applicazione e questo emerge sin dalle prime fasi progettuali, nelle quali si sviluppa in maniera parallela, puntando molto sul riutilizzo del codice e sull'elevata capacità di manutenzione, applicando dei raffinamenti successivi all'architettura tramite ricicli delle fasi di analisi/design/implementazione, ed in particolare ciascun ciclo porta ad affinare gradualmente i prodotti (semilavorati) realizzati, che non vengono scartati o riprogettati, ma piuttosto corretti o estesi. Fornisce, inoltre, una strategia per lo sviluppo di un prodotto software in passi "piccoli" e gestibili. [22]

3.2 Le fasi del RUP

Il ciclo di vita di RUP è suddiviso in una serie di iterazioni[4], ciascuna delle quali è composta da una serie di fasi:

Inception

Verifica la fattibilità tecnica e la giustificazione economica dell'intervento. Inoltre, vengono definiti: i business case (argomento trattato nel paragrafo 3.2.1) che devono includere i criteri di successo, la gestione dei rischi, la stima delle risorse necessarie, la pianificazione di massima e la schedulazione dei principali obiettivi del progetto stesso.

[4]Un'iterazione è una sequenza di attività con un piano prestabilito e dei criteri di valutazione, che termina con un rilascio eseguibile.

Elaboration

Definisce in dettaglio le caratteristiche funzionali, strutturali e tecniche del sistema. Le decisioni architetturali devono essere prese avendo una conoscenza dell'intero sistema e ciò implica la descrizione della maggior parte dei requisiti del sistema stesso.

Construction

Produce in modo incrementale una versione del sistema pronta per i test di accettazione[5]. Ciò implica la descrizione dei requisiti rimanenti e la definizione dei criteri di accettazione e di verifica del software.

Transition

Produce una versione del sistema rilasciabile in produzione. Parte con il rilascio di una versione *beta* sulla quale è necessario effettuare cicli di revisione per correggere eventuali bugs, poi viene data la possibilità di inserire requisiti addizionali (non di rilievo) o completare alcuni requisiti che erano stati posticipati.

Il processo RUP integra due diverse prospettive: una *prospettiva tecnica*, che tratta gli aspetti qualitativi, ingegneristici e di metodo di progettazione, ed una *prospettiva gestionale*, che si occupa degli aspetti finanziari, strategici, commerciali e umani. Le due prospettive sono rispettivamente articolate su sei e tre "core workflow" (vedere figura 4).

La dimensione temporale visibile in figura 4 rappresenta l'aspetto dinamico del processo (cicli, fasi, iterazioni e milestone) nel quale un prodotto software viene progettato e costruito. La dimensione strutturale, invece, rappresenta la struttura statica del processo, descritta mediante i suoi componenti: attività, workflow, artefatti e ruoli. [22]

3.2.1 Business case

Un business case, spesso presentato con un documento scritto ben strutturato, è uno strumento che supporta la pianificazione e il processo decisionale, catturando la motivazione per l'avvio o meno di un progetto o di un'attività.

Un business case convincente cattura adeguatamente le caratteristiche quantificabili così come quelle non quantificabili di un progetto proposto, mostrando le

[5] Un test di accettazione viene effettuato dal cliente e ha lo scopo di verificare che il sistema si comporti in maniera prevista.

conseguenze di ogni alternativa nel corso del tempo, calcolando inoltre i possibili benefici ed i costi previsti per ogni scenario.

Un buon business case identifica anche i fattori critici di successo e gli eventuali imprevisti che dovranno necessariamente essere gestiti, al fine di ottenere i risultati previsti, individuando i rischi o i fattori che difficilmente possono essere controllati e che possono causare risultati diversi da quelli auspicati.

3.3 Gli elementi principali

Il RUP è rappresentato da quattro elementi principali di modellazione:

I ruoli - il "chi". Il termine ruolo definisce i comportamenti e le responsabilità di un individuo o di un gruppo di individui che lavorano in squadra, ad ognuno dei quali viene associato un insieme di attività che logicamente vanno eseguite dalla stessa persona.

Le attività - il "come". Un'attività è un'unità di lavoro che un individuo in quel determinato ruolo deve eseguire e che produce un risultato significativo per il progetto, identificato da un artefatto.

Gli artefatti - il "cosa". Le responsabilità di ogni ruolo sono espresse in termini di elaborati che l'individuo con tale ruolo crea, modifica o controlla.

I Workflow - il "quando". I workflow si identificano con i ruoli che realizzano delle attività, ma anche le attività che producono gli artefatti.

3.4 Analisi critica

Un processo di sviluppo iterativo come il RUP fornisce la possibilità di reagire positivamente alle variazioni dei requisiti, infatti il management è in grado di prendere decisioni tattiche al momento giusto, secondo la strategia concordata col cliente.

Le parti sviluppate vengono integrate progressivamente ad ogni iterazione e questo viene facilitato dalla suddivisione in parti piccole, nelle quali i rischi o le criticità vengono rilevate velocemente.

Inoltre, può garantire il riutilizzo di software già sviluppato e questo è possibile perché piccole iterazioni sviluppano poche cose che, quindi, sono facilmente documentabili. Le iterazioni permettono revisioni di progettazione e di refactoring, individuando pattern e parti di codice da migliorare, ma permettono anche di perfezionare il processo produttivo.

Infine, è evidente che l'architettura ottenibile è più robusta: vengono individuati gli errori per poi essere rimossi nelle varie fasi di test, ad ogni iterazione.

4 Metodologia Agile

La metodologia Agile, ideata nel 2001 dalla Agile Alliance[6], è un metodo per lo sviluppo di software che coinvolge il più possibile il committente, in modo tale da avere un'elevata reattività alle sue richieste, cercando così di ridurre il rischio di fallimento dei progetti, ma aumentando invece la soddisfazione del cliente. [8]

4.1 Caratteristiche

L'idea di base delle metodologie agili è di non essere *predittive*, non cercano cioè di prevedere come evolverà il sistema, ma di essere *adattive*, propongono cioè valori e pratiche per meglio adattarsi all'evoluzione dei requisiti dell'utente, prima che del sistema software.

Nei progetti software, le esigenze del cliente sono in costante mutamento: sebbene l'avere in anticipo tutti i requisiti dell'utente sia un aspetto desiderabile, spesso non è ottenibile. [5]

> *"Tutto cambia nel software. I requisiti cambiano. La progettazione cambia. Gli aspetti commerciali cambiano. La tecnologia cambia. I componenti del team cambiano. Il problema non è il cambiamento, di per sé, perché i cambiamenti avverranno; il problema, piuttosto, è l'inabilità di far fronte ai cambiamenti quando essi avvengono.*
>
> *Kent Beck*

Questa metodologia porta profondi cambiamenti nel processo di sviluppo. Una delle modifiche più importanti è quella di ridurre il tempo di consegna al cliente: si suddivide il progetto totale in parti più piccole (decise dal cliente) e per ogni iterazione si consegnano le funzionalità pronte ad essere utilizzate dal cliente dopo poche settimane (funzionalità utilizzabili al 100%). Questo è dovuto dal fatto che è necessario arrivare sul mercato nel minore tempo possibile, velocizzando lo sviluppo delle funzionalità, e non come avviene con il modello a cascata (trattato nel capitolo 2) che impiega una quantità di tempo troppo elevata.

Ciascuna iterazione può essere immaginata come un piccolo progetto da svolgere, nel quale sono presenti le consuete fasi del processo di sviluppo del software (viste nel paragrafo 2.2), con la differenza, rispetto alle vecchie metodologie, che si ha un immediato riscontro da parte del cliente, dato soprattutto dalla stretta collaborazione tra il team di sviluppo e gli esperti di business.

[6] L'Agile Alliance è un'organizzazione no-profit formata da esperti informatici ed impegnata a promuovere i principi dello sviluppo Agile.

Per adottare pienamente questa metodologia è necessario essere coraggiosi ed avere una mente aperta per essere pronti all'inevitabile cambiamento. L'elevata disciplina e il continuo coinvolgimento del cliente devono essere le basi su cui implementare il proprio metodo agile, enfatizzando in particolare l'uso dei test per incrementare la qualità del codice.

Come evidenziato dalla figura 5, il metodo agile cambia il modo di concepire lo sviluppo di software: si passa da un modello (a cascata) basato su piani di lavoro fissi, ad uno che punta al valore del prodotto che verrà realizzato. Mentre nel metodo waterfall si fissa un requisito e si decide il numero di risorse da utilizzare e il tempo a disposizione per soddisfare tale requisito, nel metodo agile invece (non essendo un metodo prescrittivo) si capovolge questo concetto: si fissano le risorse e il tempo e si decide quante e quali funzionalità realizzare per il cliente, dividendoli eventualmente in moduli più piccoli.

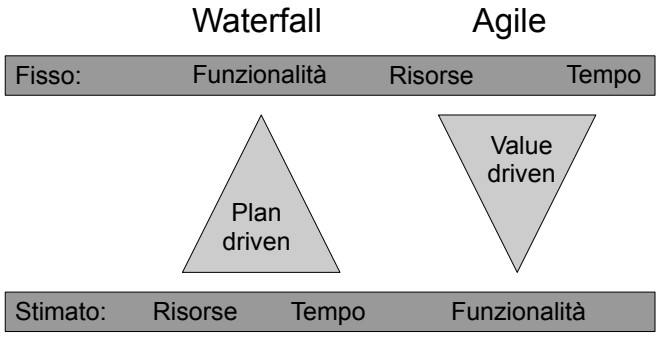

Figura 5: Due modelli messi a confronto.

L'uso di questa metodologia, inoltre, serve ad abbattere i costi di sviluppo del software e a ridurre al minimo la parte di progettazione e di conseguenza anche i tempi di consegna (spesso troppo dispendiosi), passando rapidamente dalle necessità del cliente allo sviluppo per soddisfare questo bisogno.

L'Agile Alliance creò un manifesto nel quale furono evidenziati i princìpi e i valori di questa metodologia allo scopo di diffonderla nel mondo per incentivare altri sviluppatori a creare software migliori.

4.2 Agile Manifesto[7]

Stiamo scoprendo modi migliori di creare software, sviluppandolo e aiutando gli altri a fare lo stesso.

[7]Agile Alliance, 2001, www.agilemanifesto.org

Grazie a questa attività siamo arrivati a considerare importanti:

- Gli individui e le interazioni più che i processi e gli strumenti.
- Il software funzionante più che la documentazione esaustiva.
- La collaborazione col cliente più che la negoziazione dei contratti.
- Rispondere al cambiamento più che seguire un piano.

Ovvero, fermo restando il valore delle voci a destra, consideriamo più importanti le voci a sinistra.

4.2.1 Princìpi

- La nostra massima priorità è soddisfare il cliente rilasciando software di valore, fin da subito e in maniera continua.
- Accogliamo i cambiamenti nei requisiti, anche a stadi avanzati dello sviluppo. I processi agili sfruttano il cambiamento a favore del vantaggio competitivo del cliente.
- Consegniamo frequentemente software funzionante, con cadenza variabile da un paio di settimane a un paio di mesi, preferendo i periodi brevi.
- Committenti e sviluppatori devono lavorare insieme quotidianamente per tutta la durata del progetto.
- Fondiamo i progetti su individui motivati. Diamo loro l'ambiente e il supporto di cui hanno bisogno e confidiamo nella loro capacità di portare il lavoro a termine.
- Una conversazione faccia a faccia è il modo più efficiente e più efficace per comunicare con il team ed all'interno del team.
- Il software funzionante è il principale metro di misura di progresso.
- I processi agili promuovono uno sviluppo sostenibile. Gli sponsor, gli sviluppatori e gli utenti dovrebbero essere in grado di mantenere indefinitamente un ritmo costante.
- La continua attenzione all'eccellenza tecnica e alla buona progettazione esaltano l'agilità.

- La semplicità, l'arte di massimizzare la quantità di lavoro non svolto, è essenziale.
- Le architetture, i requisiti e le progettazioni migliori emergono da team che si auto-organizzano.
- Ad intervalli regolari il team riflette su come diventare più efficace, dopodiché regola e adatta il proprio comportamento di conseguenza.

4.2.2 La carta dei diritti

Esiste inoltre una carta dei diritti del cliente e del team di sviluppo che vuole indicare delle linee guida per una pacifica collaborazione. [23]

I diritti di tutti:

- Essere trattati con rispetto.
- Produrre e ricevere un lavoro di qualità in un qualsiasi momento, concordando norme e princìpi riguardante il progetto.
- Stimare le attività in cui si è attivamente coinvolti e rispettare le stime degli altri.
- Devono essere fornite adeguate risorse (tempo, denaro, ...) per compiere il lavoro.
- Determinare come le risorse saranno introdotte. Per il personale di business sapere come i fondi saranno spesi e per il personale tecnico sapere quali sono i loro compiti.
- Dare l'opportunità di acquisire le conoscenze pertinenti per rendere il progetto un successo. Gli uomini di business vorranno probabilmente conoscere le tecnologie di base e le tecniche e il personale tecnico vorrà conoscere il business.
- Prendere decisioni e fornire informazioni in modo tempestivo.
- Conoscere i processi software della propria organizzazione, seguire e migliorare questi processi quando necessario.

Le responsabilità di tutti:

- Produrre un sistema che meglio soddisfi le proprie esigenze nell'ambito delle risorse che si è disposti ad investire in essa.

- Essere disposti a lavorare con gli altri, in particolare quelli che non hanno le stesse specializzazioni.
- Condividere tutte le informazioni, compreso il "work in progress".
- Espandere attivamente le conoscenze e qualifiche.

4.3 Fasi della Metodologia Agile

Nel modello a cascata si hanno quattro grandi fasi che vengono eseguite una dopo l'altra una volta terminata completamente quella precedente (vedere figura 2). Un problema di questa metodologia è quello dell'impossibilità di stabilire, in termini di percentuale, il livello di avanzamento del progetto. Per esempio, "Siamo arrivati al 50% del lavoro totale? Oppure siamo solo al 20%?". Non si può rispondere con certezza a questa domanda perché le fasi non hanno uguali contenuti e non le si può stimare in modo affidabile. [11]

Il modello Agile adotta, invece, un approccio di tipo *slicing* suddividendo il progetto complessivo in piccole parti di durata fissa per ogni intervallo e mantenendo le suddivisioni usuali al suo interno (vedere figura 6), in modo da conoscere esattamente quanto tempo viene impiegato e a che punto si trova l'avanzamento totale del progetto. Questo è dovuto grazie alla maggiore facilità di stimare la durata di una piccola operazione piuttosto che di una grande.

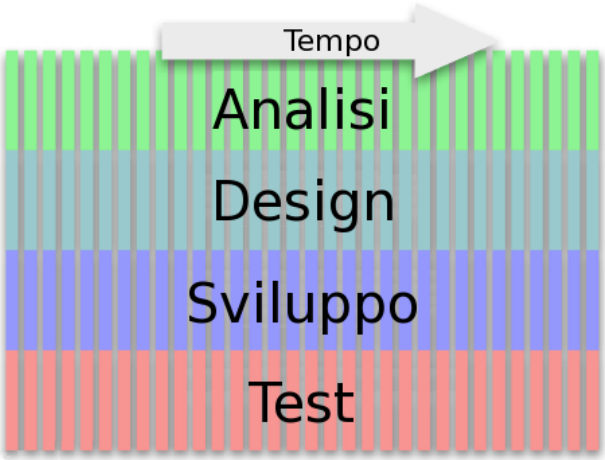

Figura 6: Rappresentazione delle fasi del processo di sviluppo del metodo Agile. [11]

Inoltre, risulta assai importante il fatto che l'output, che restituisce ogni iterazione (approccio *iterativo*), è un incremento di software (approccio *incrementale*) utiliz-

zabile al 100% e che ciascuno contribuisce alla riuscita del progetto intero. Questo assicura molti benefici: la soddisfazione del cliente, che può già usare una parte del software dopo la consegna, e si riceve anche un feedback immediato sul software della parte appena realizzata, il che aiuta a cambiare, correggere e migliorare più velocemente.

4.4 Benefici

Massimizza il ritorno sul capitale investito (ROI)

Con una metodologia Agile, grazie alla sua rapidità di consegna e di iterazioni brevi, è possibile anticipare la concorrenza ed uscire prima sul mercato (osservabile in figura 7), riducendo allo stesso tempo il rischio di fallimento del progetto e dando la possibilità al cliente di beneficiare del nuovo software in tempi molto più rapidi rispetto alle altre metodologie.

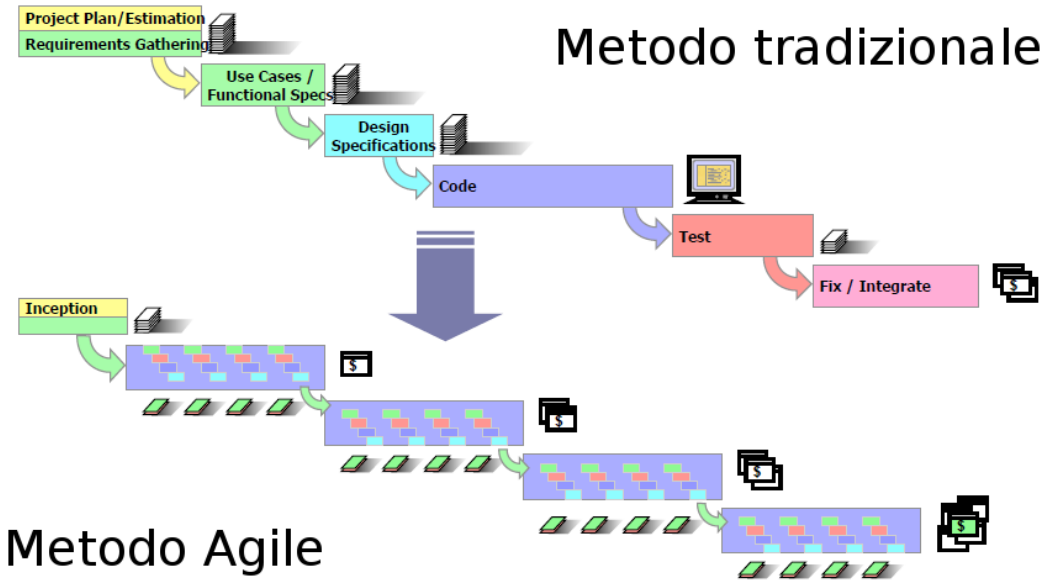

Figura 7: I due processi di sviluppo a confronto. [11]

Comunicazione e Trasparenza

I clienti perdonano anche gli errori, ma se non dovessero essere informati di questi, potrebbero crearsi dei problemi. La comunicazione tra team di sviluppo e cliente è di fondamentale importanza, sia nei casi positivi che in quelli negativi, perché aiuta a gestire le eventuali nuove esigenze e le priorità in maniera molto rapida. Forni-

sce, inoltre, una migliore visibilità (intesa come monitoraggio) dell'avanzamento del progetto.

Lavorando a stretto contatto con il cliente si possono comprendere meglio e più rapidamente le sue necessità, evitando di prendere percorsi sbagliati e di sprecare del tempo inutilmente.

L'efficacia della comunicazione con il cliente durante tutte le iterazioni porta alla completa soddisfazione del cliente stesso.

Miglioramento continuo dei processi

Attraverso piccole iterazioni e rapidi feedback, i problemi vengono identificati rapidamente e il codice può essere migliorato in qualsiasi momento, aumentandone così la qualità e riducendo i costi del cambiamento del software.

Aumento della produttività

Questo è un punto da non sottovalutare. Al team viene data la possibilità di lavorare insieme (pair programming) e questo porta ad una comprensione più rapida e migliore del software in via di sviluppo.

Minore costo del cambiamento

È stato dimostrato che il costo del cambiamento del software, cioè la manutenzione o la correzione di errori, è minore quando si utilizzano tecniche Agili rispetto a quelle tradizionali (vedere figura 8). Questo è dovuto alla maggiore attenzione alla qualità intrinseca (uno dei punti di forza di questa metodologia) nello sviluppo con metodi agili.

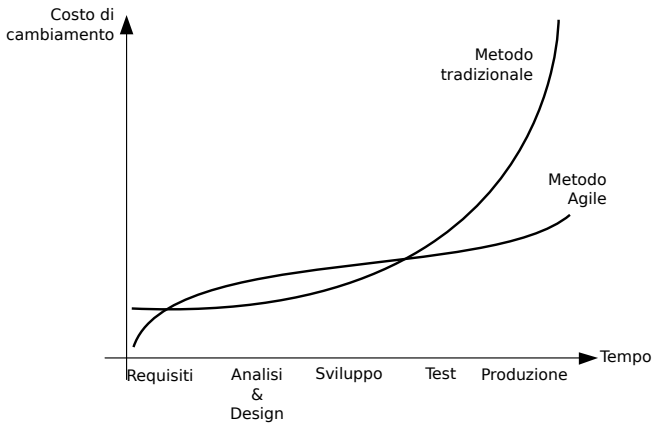

Figura 8: Gli andamenti dei costi di cambiamento dei due metodi a confronto.

Questa conclusione emerge osservando un processo implementativo con modello tradizionale il quale impiega molto tempo, al termine dello sviluppo, nel trovare e correggere i difetti del codice. Questo è dovuto al fatto che si effettuano i test soltanto alla fine dell'implementazione ed è proprio in questa fase che si perde la maggior parte del tempo.

Nelle metodologie agili, invece, vengono scoperti e corretti molti più difetti del codice durante lo sviluppo perché i test sono parte integrante dell'implementazione (Test-Driven Development, argomento che verrà trattato nel paragrafo 5.3) e questo causa un minore costo nella correzione successiva perché i problemi risultano di minore entità.

Nessuna superfluità di codice

La metodologia agile è nota per l'innovativo approccio alla progettazione e alla codifica, ponendosi come obiettivo il fatto di evitare di implementare più codice di quanto serva per la soddisfazione dei requisiti del cliente: scrivere un codice che non porta valore aggiunto, è superfluo se non addirittura inutile. Questo concetto è dimostrato anche da uno studio [14], il cui diagramma è mostrato in figura 9, che dovrebbe sensibilizzare gli sviluppatori a creare software realmente utili.

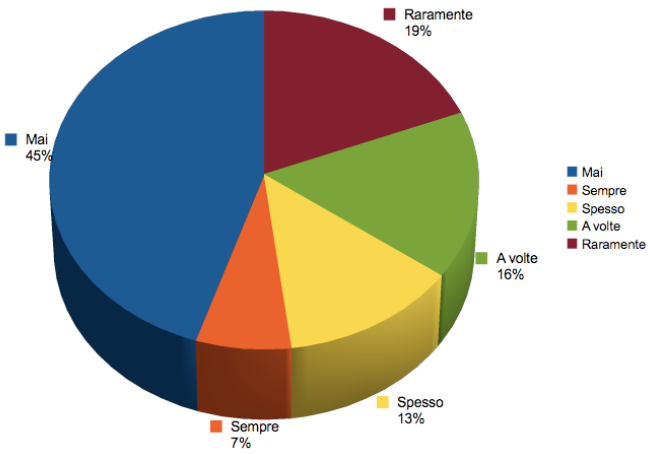

Figura 9: Grafico su uno studio dell'utilizzo delle funzionalità da parte degli utenti.

Sono stati individuati quattro elementi che conducono ad uno sviluppo software di successo [26]:

1. Rapido rilascio di una funzionalità per la valutazione del cliente.
2. Creare rilasci ed eseguire i test di integrazione quotidianamente.
3. Avere una squadra e/o un leader esperti per prendere adeguate decisioni.

4. Un'architettura modulare che possa sostenere la possibilità di aggiungere facilmente nuove caratteristiche.

4.5 Agile "sopra tutto"

In un certo senso, si può vedere la metodologia Agile come una raccolta di quelle tecniche che rivoluzionano i metodi tradizionali e che condividono i valori e i princìpi del Manifesto Agile. Sotto questo nome, infatti, si raggruppano metodologie innovative (vedere figura 10) come Extreme Programming (o XP, ideato da Kent Beck), Crystal (ideato da Alistair Cockburn), Scrum (ideato da Ken Schwaber) e Lean Software Development (ideato da Tom e Mary Poppendieck), che si definiscono appunto agili perché consentono di rivedere di continuo le specifiche e di cambiarle durante lo sviluppo mediante un forte scambio di informazioni e di pareri con il committente. [24]

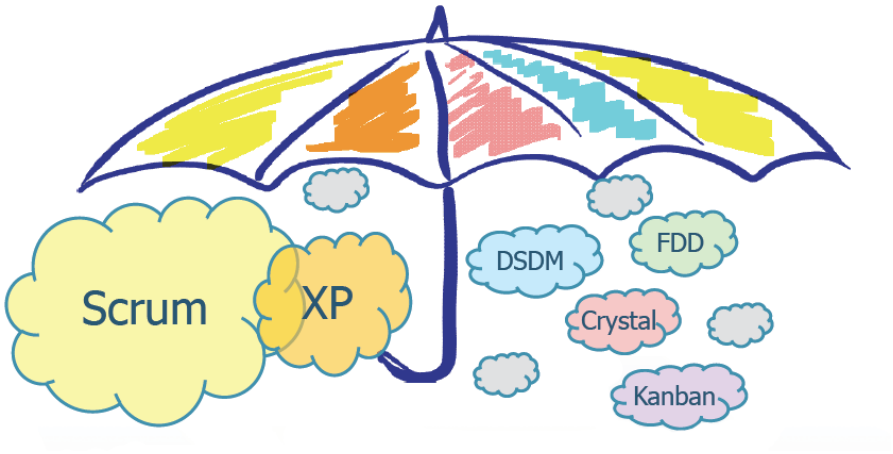

Figura 10: Grafico che mostra come il metodo Agile raccoglie le altre sotto-metodologie.

5 eXtreme Programming (XP)

L'Extreme programming, formulato da Kent Beck con l'aiuto di Ward Cunningham e Ron Jeffries, è un processo di sviluppo *adattivo e people oriented* che si implementa attraverso una serie di pratiche: comunicazione, semplicità, riscontri, rispetto e coraggio. [9]

Soprattutto la pratica del coraggio è quella che generalmente imprime più perplessità quando si parla di XP. In questo contesto si intende avere coraggio di incorrere nell'eventualità di cestinare il codice appena scritto e di iniziare di nuovo da capo per cercare di migliorare oppure ridurre la complessità (che è sempre possibile) del proprio codice.

XP introduce, inoltre, concetti utili alla gestione di un processo di sviluppo con la possibilità di monitorarlo, evidenziando i cambiamenti dei requisiti che possono variare rapidamente.

Per molte persone, XP non è nulla di più che una raccolta di "buon senso comune", come suggerito anche dallo stesso Kent Beck nel suo libro [7], dove consiglia l'impiego di modalità e pratiche molto leggere e l'utilizzo di un set di documentazione minimo ma funzionale e di facile manutenzione.

Proprio la quasi inesistenza di documentazione, rende la condivisione della conoscenza molto difficile perché è nota solo a poche persone (team agili composti da un ristretto numero). Questo emerge soprattutto quando cambiano le persone all'interno di un team; i nuovi arrivati avranno molta difficoltà a colmare le lacune.

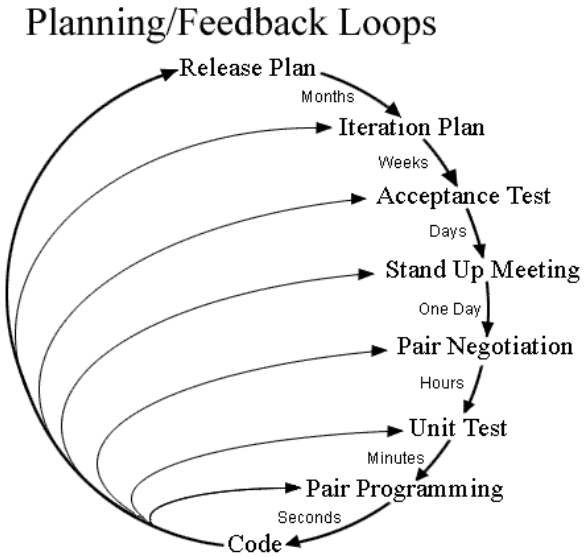

Figura 11: Grafico delle fasi dell'extreme programming.

5.1 Pratiche

Alla base dell'extreme programming ci sono alcune pratiche [6] che caratterizzano questa metodologia di processo:

Pianificazione. La pianificazione viene vista come un dialogo tra il cliente e il team di sviluppo e stabilisce un **piano di lavoro**, definito ad intervalli brevi e costantemente aggiornato. Il cliente descrive uno scenario che il sistema deve soddisfare e, dopo aver analizzato la stima fatta dal team di sviluppo, decide le priorità e le scadenze con le quale si andranno a sviluppare le funzionalità. La pianificazione crea un piano di lavoro con le funzionalità da realizzare riguardante quel determinato rilascio. La stretta collaborazione con il cliente, seguendo anche la carta dei diritti descritta nel paragrafo 4.2.2, gioca in questa fase un ruolo fondamentale.

Cliente sul posto. È fondamentale una stretta collaborazione con il cliente, che è la persona che realmente utilizza il sistema e che deve contribuire attivamente alla realizzazione del software ed essere disponibile a verificarne le funzionalità, oltre a rispondere a qualunque domanda posta dai programmatori. In questa fase per cliente si intendono le persone che sono specializzate nell'argomento che riguarda la specifica funzionalità che si andrà a realizzare, per esempio un responsabile di un reparto di un supermercato se si dovesse realizzare un software per l'assortimento automatico degli articoli in vendita.

Rilasci piccoli e frequenti. Le iterazioni di sviluppo sono molto brevi (quindi è piccolo anche il numero di funzionalità nei rilasci) e frequenti cosicché si può avere un rapido feedback da parte del cliente per riuscire a governare i possibili ed inevitabili cambiamenti.

Unit Testing e Continuous Integration. I *test unitari* (Unit Test) sono dei test che verificano la correttezza del codice in ogni sua piccola parte (da qui *unitario*) e vengono fatti costantemente utilizzando la tecnica del test-driven development (trattato nel paragrafo 5.3), che controlla e garantisce la qualità del software che si sta sviluppando e verifica se ci si sta attenendo ai requisiti funzionali del cliente.

I test di integrazione sono eseguiti sempre prima di aggiungere una qualsiasi nuova classe al progetto, dopo averne modificata una già esistente e aver verificato che i test unitari non diano problemi. Questo eviterà ritardi, verso la fine del ciclo del progetto, causati da problemi d'integrazione. In questo modo, inoltre, è sempre possibile (in qualsiasi momento) fare un rilascio del software in fase di sviluppo.

Design semplice. Il codice dovrà prevedere un design, inteso come il prodotto della fase della progettazione, il più semplice possibile per rendere più facile la manutenzione del prodotto una volta rilasciato al cliente.

Pair programming. Il pair programming consiste nella programmazione a coppie su una stessa macchina: due programmatori sono seduti l'uno accanto all'altro su una stessa macchina, con una tastiera ed un mouse. Ognuno dei programmatori ha un ruolo diverso, uno dei due scrive (test, codice, ecc.) mentre l'altro ne verifica la correttezza. I due programmatori possono/devono scambiarsi i ruoli dopo un po' di tempo. Questa pratica genera una maggiore attenzione al codice scritto aumentando la qualità e diminuendo gli errori commessi per distrazioni e il tempo per correggerli.

Refactoring. Il Refactoring consiste nel riprogettare di volta in volta il codice, cercando di migliorarlo e di renderlo il più semplice possibile in modo da favorire le modifiche e la manutenzione. Inoltre tutti i test devono continuare ad essere eseguiti correttamente.

Proprietà collettiva del codice e Standard di codifica. Chiunque può modificare o rifattorizzare qualunque parte del codice, in un momento qualsiasi, indipendentemente dal fatto di chi l'ha implementato. Questa facilità di modifica è data anche dall'utilizzo di codifiche standard che permettono a chiunque di capire il codice scritto da altre persone.

40 ore alla settimana. L'assicurazione della qualità intrinseca del codice può essere raggiunta anche con l'essere estremamente rispettosi delle persone (nel nostro caso: i programmatori) e dei loro bisogni personali, sociali e psicologici: *una persona non messa sotto stress e riposata, lavora meglio!*

5.2 Le quattro variabili

L'Extreme Programming distingue quattro diverse variabili (vedere figura 12) che entrano in gioco nei progetti software: costo, tempo, qualità e scope. [5]

L'elemento nuovo di XP è il fatto che solo tre di queste variabili sono definite a priori e restano fisse, mentre la quarta sarà quella che verrà stabilita dal gruppo di lavoro in base al valore delle altre.

Il management dei progetti solitamente tende invece a fissare a priori il valore di tutte e quattro le variabili, mettendo in difficoltà il gruppo di lavoro. Si può facilmente intuire che la variabile che normalmente tende ad essere la più trascurata,

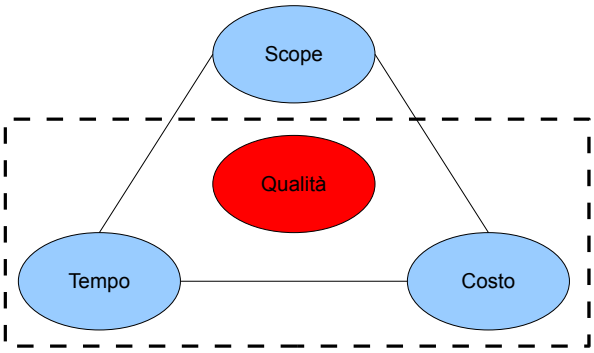

Figura 12: Rappresentazione che evidenzia il comportamento delle quattro variabili.

quando il tempo a disposizione è poco, è la qualità, cosa che si oppone alla filosofia agile. La diminuzione della qualità e lo stress del gruppo induce alla riduzione della velocità di progressione del progetto perché nessuno è in grado di lavorare bene quando è messo sotto pressione.

XP rende visibili a tutti (programmatori, clienti, project manager, ecc.) le quattro variabili. In questo modo i valori iniziali possono essere manipolati finché il quarto valore non soddisfi tutti (naturalmente, con la possibilità di scegliere differenti variabili da controllare). [6]

Per quanto riguarda lo scope del progetto, è una buona idea lasciarla come variabile libera in modo che, una volta fissate le altre tre variabili, il team di sviluppo potrà decidere la portata in termini di:

- Stima delle attività da svolgere per soddisfare i requisiti del cliente.
- Realizzazione anticipata dei requisiti più importanti in base alla regola 80/20, nella quale si sviluppa il 20% delle funzionalità totali che contribuiscono all'80% delle funzionalità utilizzate maggiormente.

5.3 Test-Driven Development (TDD)

Il Test-Driven Development (TDD) è una **tecnica di sviluppo (e non di test)** che combina il Test-first development (TFD) con il Refactoring per promuovere l'alta qualità nel codice.

Avere un approccio TDD significa scrivere i test prima ancora di avere scritto il codice da esaminare. Questo metodo di sviluppo, a prima vista, sembra non realizzabile (per via della "inversione" nella scrittura) invece costringe a pensare in primo luogo al design che avrà il codice ed è proprio questo che alza la qualità.

Il vantaggio nell'utilizzare la tecnica TDD è quello di sviluppare codice di alta qualità (uno dei fondamenti dell'agile), che sia anche già testato.

5.3.1 Utilizzo

La caratteristica del TFD, cioè scrivere dei test prima di creare il codice, è quella di aggiungere rapidamente un test (unit test[8]) formato da **asserzioni** riguardanti una piccola parte (da quì unitario). Ultimato il test, lo si esegue ed inizialmente deve fallire perché non è ancora stato creato il codice funzionale.

Successivamente si implementa il codice funzionale per farlo superare, ricordandosi però di concentrarsi solamente sulle parti interessate da quel test. Questo perché scrivere più codice di quanto effettivamente serva è un costo e si rischia di implementare cose che non danno valore aggiunto o che addirittura rischiano di essere sbagliate. Quindi lo sviluppatore deve scrivere codice funzionale il più semplice possibile affinché il test sia eseguito con successo.

Una volta terminata l'implementazione, si esegue un'altra volta il test/suite di test e se dà esito positivo si procede con il refactoring, sia del codice che del test, eliminando le eventuali duplicazioni oppure parti di nessuna utilità. In caso contrario, si corregge il codice finché non si superano i test. Per il procedimento osservare la figura 13.

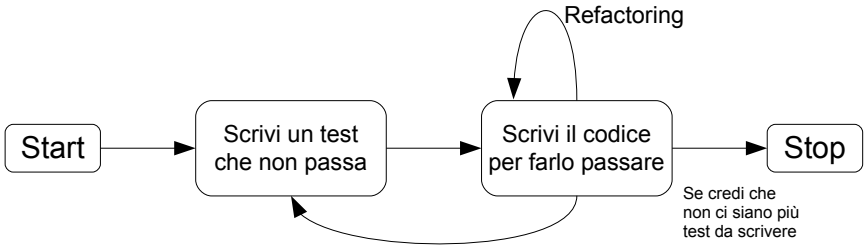

Figura 13: Il processo dello sviluppo Test-driven.

5.3.2 Considerazioni sul TDD

Il TDD deve fornire un approccio alla programmazione che favorisca la scrittura di un codice in moduli indipendenti ed interoperanti, che a sua volta contribuisce, insieme ai *Design pattern* e ad altre pratiche comuni alla realizzazione del miglior codice possibile. Esso, infatti, migliora il design del codice perché pensa prima alle interfacce e poi alla loro implementazione.

[8]Lo unit testing è una procedura usata per verificare singole parti di un codice sorgente. Per unit si intende genericamente la minima parte testabile di un codice sorgente: un'unità può rappresentare un singolo programma, funzione, procedura, ma può essere anche un metodo.

Il Refactoring che si esegue alla fine serve per migliorare il codice scritto, riducendo i conflitti e aumentando l'integrazione perché tutto viene monitorato dai test.

Questa metodologia sembra semplice, ma in realtà necessita di molta disciplina perché è facile non attenersi alle regole e scrivere del codice funzionale senza aver scritto precedentemente i test.

Un ulteriore vantaggio nell'utilizzare questa tecnica è l'immediato feedback che si riceve quando si aggiungono funzionalità o quando si modifica del codice, basta soltanto rieseguire i test e ci si accorge immediatamente se sono stati commessi degli errori.

5.4 Analisi critica

Alcuni aspetti delle tecniche agili di sviluppo come l'Extreme Programming vengono considerate poco applicabili per vari motivi [5]:

Non prescrittivo. Uno dei problemi dell'Extreme Programming è la poca chiarezza in merito alla concreta applicazione delle pratiche, dato che non è prescrittivo ma adattivo.

Funziona solo per buoni programmatori. Una delle più ricorrenti critiche emerse è che viene considerata valida solo per buoni programmatori perché necessita di un certo livello di stile nell'implementazione di codice con una predisposizione allo sviluppo Test-driven. Addetti meno esperti potrebbero avere più difficoltà a progettare in maniera semplice e manutenibile fin da subito.

Pair programming. Un altro problema riscontrato è quello riguardante il pair programming: non sempre c'è la disponibilità numerica per attuarla. Spesso c'è anche una "resistenza culturale" all'uso di questa pratica perché viene considerato come un eccessivo costo (doppio costo del personale) portando inoltre ad un lento avanzamento dei lavori, ma questo, in realtà, non è vero perché due programmatori (di pari livello di capacità) superano le difficoltà più facilmente rispetto ad un programmatore singolo, e con maggiore qualità. Secondo alcuni studi empirici [13] la programmazione a coppie può ridurre il tempo di uscita sul mercato anche del 29%.

Unit testing. Inoltre, sono sorte delle perplessità sull'utilizzo degli unit test in quanto comportano un doppio svantaggio: oltre ad avere infatti il doppio del numero di classi (in genere ogni classe di codice dovrà avere una classe di test), si deve

eseguire una modifica o un refactoring non soltanto sul codice ma anche sui test e quindi viene considerato come se fosse un doppio lavoro.

Orientamento alle persone. I metodi agili sono orientati alle persone e questo può portare ad una difficoltà per il management: bisogna sposare la metodologia e saper gestire il morale degli sviluppatori, applicando quindi la pratica in prima persona e non solo imponendola agli altri.

Costante disponibilità del cliente. Un'ulteriore critica viene fatta anche per quanto riguarda la presenza del cliente. Spesso il cliente non è disponibile per chiarire alcuni punti o rispondere alle domande di cui gli sviluppatori necessitano risposta oppure il cliente cambia talmente spesso i requisiti che i programmatori sono costretti a congelarli per proseguire con i lavori.

6 Scrum

"Scrum is not a methodology – it is a pathway"

Ken Schwaber

Scrum è un **framework agile**, all'interno del quale possiamo utilizzare vari processi e varie tecniche. È stato ideato da Ken Schwaber e da Jeff Sutherland tra il 1993 e il 1995 e si può definire come un approccio innovativo al project management, che trasforma in modo radicale due relazioni tradizionali nello sviluppo software: quella tra clienti e fornitori e quella tra capo progetto e sviluppatori. Scrum non ha delle regole rigide: ogni azienda, dunque, ha la piena libertà di adattare la metodologia come meglio crede. [3]

"Non attaccatevi troppo ad una qualsiasi arma o ad una singola scuola di combattimento."

Miyamoto Musashi

Il termine Scrum deriva dal rugby ed indica "il pacchetto di mischia", cioè tutti i protagonisti devono lavorare insieme per raggiungere un unico obiettivo: nel rugby, lo scopo è di spingere tutti nella stessa direzione per conquistare la palla; nell'informatica, fine ultimo è quello di sviluppare software più efficiente e di migliore qualità.

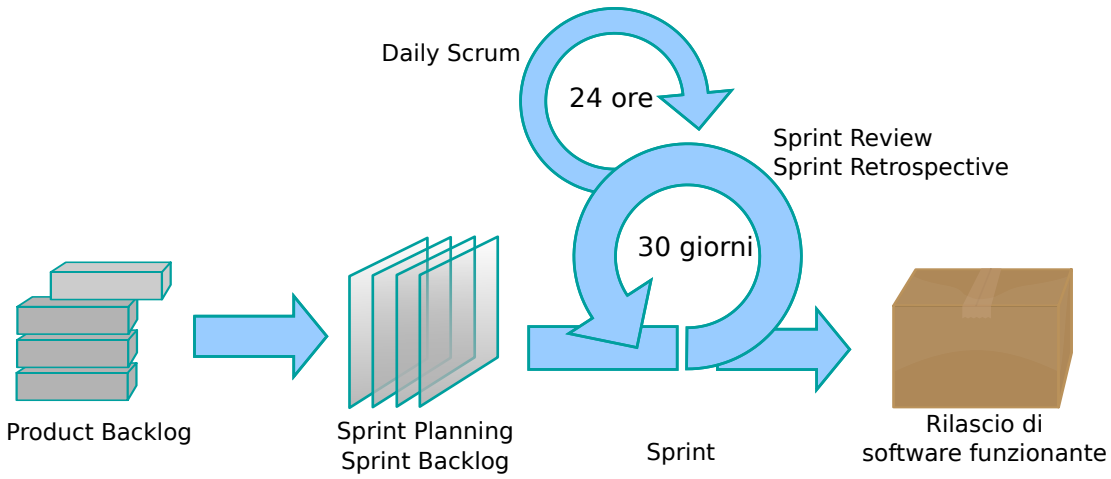

Figura 14: Il processo di sviluppo della metodologia.

6.1 Princìpi

Scrum è un processo (vedere figura 14) particolarmente indicato per situazioni progettuali caotiche, con obiettivi e requisiti in costante cambiamento, e quindi molto imprevedibili. Ciò non toglie che resta molto efficace ed efficiente anche per progetti meno problematici/confusionali.

6.1.1 Timeboxed

Una caratteristica di Scrum è quella di condurre i suoi progetti in modo iterativo ed incrementale (argomento trattato nel paragrafo 4.3), consegnando al cliente del software funzionante con sessioni di durata definita (quindi **timeboxed**).

Lavorare in questo modo è importante per quattro ragioni [19]:

- Permette di concentrarsi su ciò che è più determinante, evitando l'overengineering.
- Serve come indicatore di tempo effettivo da dedicare ad un lavoro, evitando le perdite di tempo ed aumentando l'efficienza.
- È uno strumento efficace contro la distrazione.
- Permette di anticipare del lavoro durante i momenti "liberi" tra un impegno e l'altro.

6.1.2 Sprint

Le sessioni di durata definita, di cui si è parlato nel paragrafo precedente, vengono chiamate **Sprint**, durante le quali si sviluppa il prodotto. Il cliente e il team di sviluppo identificano le caratteristiche di ogni versione da rilasciare in base alla priorità e alla complessità di ogni attività [19]. Tipicamente, hanno una durata che oscilla tra le tre e le quattro settimane, ma questo dipende dalle scelte progettuali.

Gli Sprint comprendono tutte le attività di sviluppo (analisi, progettazione, realizzazione, test, integrazione) dei modelli tradizionali e possono essere considerate come delle scatole nere, delle quali il cliente può soltanto controllare l'inizio o la fine del processo, ma non durante lo sviluppo che viene invece controllato solo dal team. [3]

All'interno di ciascuno sprint è estremamente importante che ogni giorno si tenga traccia del completamento dei task: questo serve per aiutare tutti ad essere a conoscenza, in qualsiasi momento, dello stato di avanzamento dello sviluppo. Inoltre spesso vengono utilizzati grafici e/o delle tabelle atte a monitorare l'intero sprint,

che vengono posti sulla cosiddetta **Task Board** (vedere figura 15), che può essere una parete oppure una lavagna. [2]

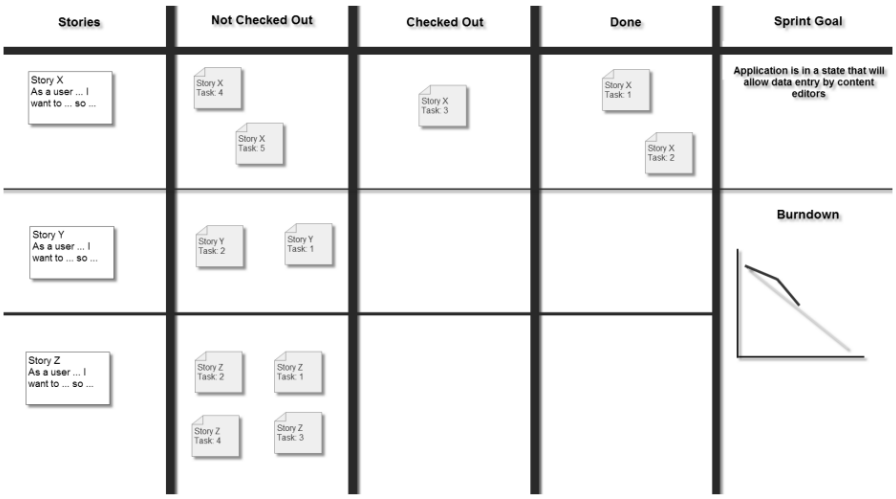

Figura 15: Un esempio di Task Board.

6.2 Ruoli

In Scrum si attribuiscono quattro ruoli diversi tra loro, ovvero: Product Owner, Scrum Master, Scrum Team e Stakeholders. [3]

6.2.1 Product Owner

Il Product Owner è il proprietario del prodotto ed è colui che dirige il progetto dal punto di vista del business, oltre a presentare la visione delle funzionalità al team degli sviluppatori.

Ha il compito di gestire il *Product Backlog* (argomento trattato nel paragrafo 6.3.1.1), aggiungendo e modificando le funzionalità, e di impartire le priorità, gestendo i conflitti che possono emergere. Per quanto riguarda le priorità, una volta decise e inserite nello sprint planning, nessuno ha la possibilità di cambiarle all'interno dello sprint stesso.

Un ulteriore compito del Product Owner è quello di quantificare l'impegno delle funzionalità (con l'aiuto del team) in modo approfondito e, se necessario, le segmenta in funzionalità più piccole per realizzarle in uno sprint successivo.

6.2.2 Scrum Master

Lo Scrum Master è il sovrintendente all'applicazione del processo di sviluppo del progetto e ha il compito di "proteggere" il team dalle distrazioni esterne al progetto (richieste estemporanee, ecc.) e di rimuovere qualsiasi blocco o impedimento per l'avanzamento delle attività durante uno sprint, restando al suo servizio per qualsiasi evenienza.

Inoltre, ha anche la funzione di incoraggiare e incentivare la comunicazione all'interno del team per semplificare la risoluzione dei problemi.

Un ulteriore compito dello Scrum Master è quello di cercare di aumentare la produttività dello Scrum Team e di controllare le fasi di ispezione e adattamento attraverso le riunioni e le retrospettive che verranno trattate nel capitolo 6.4.

6.2.3 Scrum Team

Lo Scrum Team consiste nell'insieme delle persone con competenze specialistiche diverse in grado di tradurre le richieste del Product Owner in un prodotto potenzialmente rilasciabile entro la fine dello Sprint. Le figure che si possono trovare all'interno dello Scrum Team sono programmatori, controllo di qualità, analisti, architetti, amministratori di database, ecc..

Tipicamente un team è composto da un gruppo che varia tra le quattro e le otto persone.

Durante le pianificazioni dello sprint valuta ciò che deve esser fatto e si impegna pubblicamente a realizzarne una parte, creando un elenco di tasks (lo *Sprint Backlog*) che sono tutte le micro-attività necessarie della durata tra le 4 e le 16 ore.

Per ottimizzare lo sviluppo, Scrum prevede che il team possa *auto-organizzarsi* il lavoro durante lo sprint nel senso che nessuno può decretare il come debba essere fatta una determinata attività, ma è il team stesso ad autogestirsi.

Inoltre, prevede che sia *cross-funzionale*, cioè i membri di un Team devono avere tutte le competenze necessarie per creare un incremento che conduca al prodotto finale utilizzabile. Un punto di forza di questa metodologia è il continuo aggiornamento del processo di sviluppo, grazie all'incontro quotidiano (Daily Scrum), al quale possono partecipare tutti i membri del team per esporre ciascuno il lavoro svolto il giorno precedente e le difficoltà riscontrate.

6.2.4 Stakeholders

Gli Stakeholders sono tutti coloro che possono inoltrare richieste relative all'evoluzione o alla modifica del prodotto (utenti o altri soggetti interessati).

Una loro caratteristica è quella di effettuare richieste di modifiche dei requisiti in qualsiasi momento, ma sono anche consapevoli che esse non verranno prese in carico in modo immediato ma solo al termine dello sprint, considerando anche la priorità che le viene associata.

6.3 Artefatti

6.3.1 Backlogs

I Backlogs sono dei documenti nei quali sono presenti gli elenchi delle funzionalità che sono emerse e che servono a costruire il prodotto. Le varie funzioni vengono, inoltre, ordinate per priorità (definite dal cliente), passando dalle più importanti a quelle meno importanti. [3]

I backlogs possono essere di due tipi:

- **Product Backlog**: è l'elenco di tutte le funzionalità del prodotto, ordinato per priorità. Viene iniziato nella fase di analisi e viene aggiornato costantemente con le nuove attività richieste.

- **Sprint Backlog**: è la lista delle funzionalità (prese dal Product Backlog) che devono essere implementate in uno sprint.

6.3.1.1 Product Backlog

Il Product Backlog è l'elenco principale di tutte le funzionalità desiderate nel prodotto (per un esempio, vedere figura 16) e gestito dal Product Owner. In Scrum non è necessario avere già all'inizio un elenco completo di requisiti (BRUF - Big Requirements Up Front), basta aggiornarlo di volta in volta aumentando così la conoscenza del prodotto. Le nuove specifiche verranno prese in considerazione nello sprint successivo. [2]

Di solito, le funzionalità del Product Backlog vengono espresse sotto forma di user stories[9], tecnica usata nell'extreme programming.

6.3.1.2 Sprint Backlog

Dopo aver stabilito la priorità per ogni elemento del Product backlog, il cliente espone durante lo sprint planning quali siano le componenti con priorità maggiore e il team determina quali di essi formeranno lo sprint backlog, cioè le funzionalità

[9]Una user story è un requisito di sistema in uno scenario specifico che il software dovrà soddisfare ed è composta da una o più frasi nella lingua di tutti i giorni o commerciale dell'utente.

che verranno sviluppate nello sprint in questione. Durante queste decisioni, gli elementi potranno essere suddivisi in elementi più piccoli per essere inseriti/sviluppati all'interno di uno sprint. [2]

	Item #	Description	Est	By
Very High				
	1	**Finish database versioning**	16	KH
	2	**Get rid of unneeded shared Java in database**	8	KH
	-	**Add licensing**	-	-
	3	Concurrent user licensing	16	TG
	4	Demo / Eval licensing	16	TG
		Analysis Manager		
	5	File formats we support are out of date	160	TG
	6	Round-trip Analyses	250	MC
High				
	-	**Enforce unique names**	-	-
	7	In main application	24	KH
	8	In import	24	AM
	-	**Admin Program**	-	-
	9	Delete users	4	JM
	-	**Analysis Manager**	-	-
	10	When items are removed from an analysis, they should show up again in the pick list in lower 1/2 of the analysis tab	8	TG
	-	**Query**	-	-
	11	Support for wildcards when searching	16	T&A
	12	Sorting of number attributes to handle negative numbers	16	T&A
	13	Horizontal scrolling	12	T&A
	-	**Population Genetics**	-	-
	14	Frequency Manager	400	T&M
	15	Query Tool	400	T&M
	16	Additional Editors (which ones)	240	T&M
	17	Study Variable Manager	240	T&M
	18	Haplotypes	320	T&M
	19	**Add icons for v1.1 or 2.0**	-	-
	-	**Pedigree Manager**	-	-
	20	Validate Derived kindred	4	KH
Medium				
	-	**Explorer**	-	-
	21	Launch tab synchronization (only show queries/analyses for logged in users)	8	T&A
	22	Delete settings (?)	4	T&A

Figura 16: Un esempio di un product backlog di un progetto reale.

6.3.2 Burndown Chart

Un ulteriore aiuto per tenere traccia dell'avanzamento dello sviluppo e del rispetto delle scadenze viene dato dal **Burndown chart** che può essere di due tipi:

- **Release Burndown**: misura la quantità di rilasci rimanenti di un Product Backlog rispetto al piano di rilascio totale (Release Plan).
- **Sprint Burndown**: misura la quantità di elementi rimanenti di uno Sprint Backlog nel corso di uno Sprint.

Il Burndown chart è un grafico (vedere figura 17) che misura la quantità di lavoro residuo di un Backlog.

Si contraddistingue per le seguenti caratteristiche:

- Mostra se gli elementi dello sprint backlog sono stati "fatti"[10].
- Sull'asse delle ordinate vengono messi il numero degli elementi da sviluppare e sulle ascisse i giorni nello sprint corrente.

[10]Per "fatti" si intende sviluppati e potenzialmente rilasciabili, per essere consegnati al cliente e utilizzabili come prodotto finito.

- Il team aggiorna il grafico quotidianamente.
- Deve essere aggiornabile e modificabile in modo semplice e non complicato.
- Può essere utile, tenere un elenco degli elementi dello sprint backlog da effettuare nello sprint attuale.

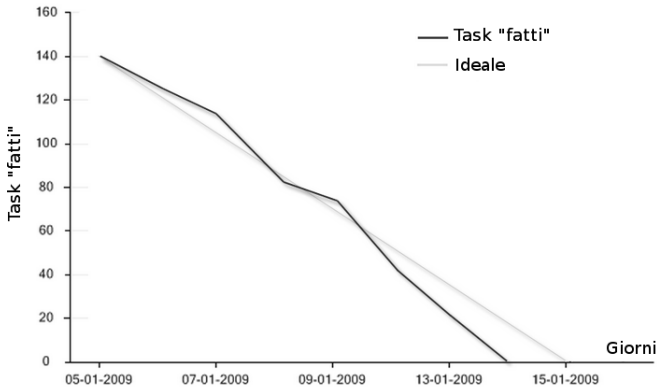

Figura 17: Un esempio di Burndown Chart.

6.4 Attività di Scrum

6.4.1 Daily Scrum

Il Daily Scrum è una riunione quotidiana di 15 minuti circa tra il capo progetto e tutti i partecipanti, da svolgere rigorosamente in piedi per evitare che duri di più. Viene effettuata sempre nello stesso luogo e alla stessa ora, ogni giorno. [3]

Ha lo scopo di identificare solamente i "colli di bottiglia", cioè le difficoltà riscontrate, e non la loro risoluzione. Ogni membro del team dice brevemente cosa ha concluso il giorno prima e quali ostacoli ha riscontrato nella sua attività e cosa farà quel giorno.

In questa riunione non viene fatta nessuna discussione (si possono fare eventualmente una volta terminato il breve incontro), proprio perché mira soltanto a mettere al corrente gli altri membri della progressione e di far risaltare gli eventuali problemi riscontrati.

6.4.2 Sprint Planning

Lo Sprint Planning è una riunione, fatta con il proprietario del prodotto e gli altri stakeholders, per la pianificazione dello sprint, nel quale si definiscono i contenuti del

prossimo rilascio, in base alle priorità evidenziate dal cliente e alle stime dell'impegno da parte del team.

Si distinguono due parti: nella prima parte viene deciso ciò che sarà fatto nello Sprint, mentre la seconda tratta il come il Team andrà ad implementare la funzionalità in modo da formare un incremento del prodotto durante lo Sprint. [2]

Durante la progettazione dello sprint, il team individua dei compiti che rappresentano le funzionalità necessarie per convertire il Product Backlog in software. I compiti dovrebbero essere suddivisi in modo tale da risultare fattibili in meno di un giorno.

6.4.3 Sprint Review e Sprint Retrospective

Lo Sprint Review e lo Sprint Retrospective sono delle riunioni, fatte con il proprietario del prodotto e gli altri stakeholders, per esporre gli elementi che sono stati sviluppati e per la revisione dello sprint appena concluso, verificando il lavoro (funzionalità rilasciate o modificate) e facendo un esame delle criticità riscontrate, in modo da aggiornare il processo lavorativo per aumentare l'efficienza e la produttività. [2]

6.5 Analisi critica

L'utilizzo di Scrum offre molti vantaggi. Ecco alcune considerazioni/particolarità che parlano a suo favore:

- Consente di lavorare in modo produttivo anche in situazioni particolarmente caotiche e confuse.
- È fortemente orientato verso risultati concreti e verso la gestione dei cambiamenti, piuttosto che alla pianificazione precisa.
- Porta ad un forte coinvolgimento del committente e ad una assunzione di responsabilità collettiva da parte del team.
- Favorisce la creazione di team coesi.

Allo stesso tempo presenta alcune caratteristiche intrinseche che aumentano i timori nell'adozione:

- È un processo che definisce esclusivamente pratiche di project management.
- Non fornisce indicazioni su come condurre altre discipline fondamentali (gestione requisiti, analisi e disegno, test, gestione configurazione, ecc.) ed è quindi opportuno integrarlo con altri approcci (ad esempio eXtreme Programming) per colmare le parti mancanti.

7 Lean Software Development

Il Lean Thinking[11] è una strategia operativa nata in Toyota negli anni '80, ma oggi viene universalmente applicata in settori e ambiti diversi per aumentare l'efficienza della produzione (nel nostro caso di software) ed eliminare gli sprechi, cioè tutto ciò che non porta valore aggiunto al cliente [15].

Il cosiddetto "Toyota Way", sinonimo di Lean, si basa fondamentalmente su due concetti: profondo rispetto per le persone (che possono essere i lavoratori, i partners oppure i clienti) ed il continuo miglioramento.

Figura 18: L'insegna all'entrata della sede della Toyota che evidenzia il loro motto.

> *"Entrare in ottica Lean non significa adottare un modello astratto, fatto semplicemente di principi e teoremi e lontano dalla realtà dell'azienda e dei lavoratori. Il Lean Thinking non è uno strumento promozionale, non è una collezione di slogan, né un giocattolo per imprenditori, investitori e management. Il Lean è un veicolo di trasformazione totale. Le aziende che lo metabolizzano imparano a ragionare in maniera diversa, innovativa, talvolta addirittura trasgressiva rispetto ai canoni del modello d'impresa tradizionale. E questa trasformazione è visibile."*
>
> *www.leanthinking.it*

Mary e Tom Poppendieck si possono considerare i pionieri dello sviluppo di software attraverso la tecnica industriale Lean, usata per lo più nel campo manifatturiero (come appunto il caso di Toyota). Dimostrarono come lo sviluppo di un software non sia poi così diverso dallo sviluppo di un prodotto. Evidenziarono anche quanto importante fosse l'evoluzione del processo di sviluppo stesso e come questo dipendesse dal prodotto che si stava sviluppando.

[11] Lean (o Lean thinking) è il termine inglese coniato da alcuni ricercatori del MIT (Massachusetts Institute of Technology) per contrastare il nominativo di produzione di massa e significa creare valore per il cliente con il minor numero di risorse.

7.1 Princìpi

"Toyota Way" è fondato su alcuni principi che stanno alla base del successo del Gruppo Toyota e che sono condivisi da tutta l'organizzazione, che a qualsiasi livello li applica nelle attività lavorative e nel rapporto con gli altri. [10]

7.1.1 Genchi Genbutsu ("Andare alla fonte")

Questo è un principio critico e fondamentale, molto spesso sottovalutato. In [17] è stato indicato come il primo fattore di successo per il continuo miglioramento. "Andare alla fonte", cioè recarsi lì dove il cliente utilizzerà il prodotto commissionato, è necessario per decidere quali siano le decisioni giuste da prendere, raccogliere consensi e raggiungere gli obiettivi prefissati. Il management non dovrebbe mai limitarsi ad analizzare i report ricevuti o fare riunioni con altri manager, ma dovrebbe, invece, andare sul posto del lavoro e guardare con i propri occhi per scoprire e capire i veri bisogni del cliente, in modo tale da rendere il processo migliore.

7.1.2 Kaizen

> *"ogni cosa merita di essere migliorata"*
>
> *detto giapponese*

Il Kaizen è una strategia di management giapponese che significa "miglioramento lento e continuo", che coinvolge l'intera struttura aziendale: si tratta di un credo che si basa sulla convinzione che tutti gli aspetti della vita possano essere costantemente migliorati. Deriva dalle parole giapponesi "Kai" che significa "continuo" e "Zen" che significa "miglioramento". [25]

Con la parola miglioramento si possono riconoscere due modi completamente diversi di progresso:

- Il Kaizen (un miglioramento lento ma costante e inarrestabile).
- L'innovazione (un miglioramento rapido, radicale che necessita di grandi risorse).

Nell'utilizzo pratico, il Kaizen descrive un ambiente in cui l'azienda e gli individui che vi lavorano si impegnano in maniera attiva per perfezionare i processi.

La forza di base che spinge le persone ad applicare il Kaizen è l'insoddisfazione per una certa situazione vigente in azienda, a prescindere da quanto questa sia stimata e quotata all'esterno. Restare fermi e non migliorarsi, infatti, significherebbe permettere alla concorrenza di crescere e dunque di sconfiggerci.

Lo sviluppo software è famoso per la sua tendenza ad ottimizzare tutto. Un'azienda deve cercare di migliorare sempre il proprio processo: dal momento in cui riceve un ordine fino a quando il software finito viene consegnato al cliente. Non si deve guardare soltanto una piccola fase ma il processo di sviluppo per intero per non creare disomogeneizzazione interna o dei colli di bottiglia (per esempio gli sviluppatori più veloci dei tester). Lo scopo principale di questo principio è quello di guardare sempre le parti specifiche in funzione del processo generale.

7.1.3 Rispetto per le persone

Un successivo principio del Lean Thinking, fondamentale nello sviluppo software e soprattutto per gli sviluppatori, è il rispetto per le persone, inteso non solo come dipendenti/colleghi, ma anche collaboratori e clienti.

> *L'essenza del sistema della Toyota è che viene data l'opportunità a ciascun impiegato di trovare i problemi nel suo modo di lavorare, di risolverli e di fare miglioramenti. [16]*
>
> *Satoshi Hino*

Guardando questi tre punti si ha una più ampia idea di che cosa possa significare il rispettare le persone:

1. Leader imprenditoriale: È una persona che ama lavorare su prodotti di successo. Una buona azienda dovrebbe formare buoni leader e fare in modo di favorire l'aumento dell'impegno per creare prodotti sempre migliori.

2. Organico tecnico esperto: Ogni azienda cerca di mantenere sempre un certo vantaggio rispetto alla concorrenza nell'area sviluppo. Deve però aspettarsi che, se si riesce ad acquisire nuove forze lavoro specializzate, lo possono fare anche i concorrenti. Aziende che provvedono ad acquisire appropriate competenze tecniche crescono e facilitano le proprie squadre a raggiungere i loro obiettivi.

3. Pianificazione e Controllo basato sulla responsabilità: Rispettare le persone significa dare ai team di sviluppo un piano generale con degli obiettivi e la possibilità di *auto-organizzarsi* per raggiungere quegli obiettivi, senza dire cosa e come devono fare, ma creare un'azienda in cui le persone pensano con la propria testa. È necessario poi controllare e migliorare il processo di auto-organizzazione dei team per evolvere e crescere ancora di più.

7.1.4 Eliminare gli sprechi

Nello sviluppo software Lean, l'obiettivo principale è quello di eliminare gli sprechi che non danno valore aggiunto al soddisfacimento del cliente nell'arco di tutto il processo.

Per eliminare gli sprechi bisogna riconoscere ciò che assicura realmente valore aggiunto (quello che vuole effettivamente il cliente) e quello che invece non lo fornisce. Tutto quello che non serve o non viene usato dal cliente è in pratica uno spreco: è statisticamente provato [14] che un utente utilizza per la maggior parte del tempo soltanto una piccola parte delle funzionalità totali a disposizione (vedere lo studio nel paragrafo 4.4).

> *"Un tipico sistema è costituito per il 64% da funzionalità usate raramente o del tutto inutilizzate, il che suggerisce che il terreno più fertile per il miglioramento della produttività dello sviluppo software stia nel non implementare funzionalità non necessarie." [12]*
>
> *Tom e Mary Poppendieck*

7.1.4.1 Value Stream Map

Sempre nell'ottica di individuare con precisione le attività che producono valore diretto per il cliente e distinguerle dalle attività accessorie, esiste la mappa del flusso del valore (**Value stream map**), che permette di individuare quali punti del processo ostruiscano la consegna del valore stesso al cliente.

Come attività complementare si stabilisce un ranking delle attività interne all'azienda secondo l'importanza percepita dal cliente rispetto al valore consegnato. Questo ranking è utile per capire in maniera più oggettiva quali siano le attività su cui cercare di tagliare i costi.

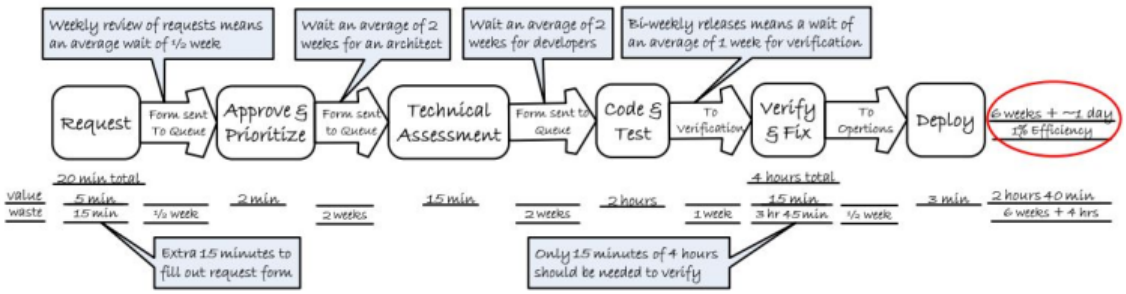

Figura 19: Un esempio di Value Stream Map. [28]

Prendendo come esempio il modello a cascata, è noto che il cosiddetto "magazzino" dei requisiti del cliente, cioè l'elenco completo delle specifiche con tutte le

esigenze da soddisfare (incluso quelle che non danno valore aggiunto), è uno spreco. Per ridurre i costi è necessario sviluppare le parti davvero importanti e cruciali in un progetto. In questo caso si parla di **flusso di valore**.

7.1.5 Creare conoscenza

Uno degli aspetti più importanti emerso già con il modello a cascata, è l'idea che la conoscenza di business (le esigenze del cliente) sia separata dalla codifica. Lo sviluppo del software è un processo di creazione della conoscenza e il modo migliore per crearla è quella di avere un continuo riscontro con la valutazione da parte dell'utilizzatore finale, che può significativamente aiutare a realizzare un prodotto migliore e di successo.

Le aziende dovrebbero dedicarsi di più alla ricerca e quando, durante lo sviluppo, riescono a scovare qualcosa di innovativo che fa progredire l'azienda stessa, devono renderlo parte integrante della conoscenza di base (*Knowledge base*) e condividerlo con i colleghi. Questo è importante soprattutto per facilitare il lavoro per prodotti futuri. [10]

7.1.6 Rinviare le decisioni in sistemi di schedulazione pull

"Pull" indica Just-In-Time cioè fare una determinata cosa nel momento in cui è necessaria. Ciò significa che il lavoro stesso e la scadenza della consegna non vengono imposti dall'esterno ma dal team che si impegna nella sua realizzazione.

Vista la rapidità con cui i requisiti e i bisogni del cliente variano, è preferibile posticipare il più possibile, all'interno delle iterazioni, le decisioni importanti e/o irreversibili per l'avanzamento dello sviluppo in modo da avere il maggior numero di informazioni possibili per evitare di intraprendere strade sbagliate.

7.1.7 Consegna rapida

> *"Tutto quello che facciamo è guardare il tempo, dal momento in cui riceviamo un ordine da un cliente al momento in cui riceviamo il denaro. Accorciamo il tempo di consegna riducendo gli sprechi che non danno valore aggiunto." [15]*
>
> *Taiichi Ohno*

Si deve riuscire a rilasciare il software così velocemente in modo da non consentire ai clienti di cambiare idea.

Le aziende che riescono a essere competitive sulla base delle tempistiche hanno un grande vantaggio: hanno eliminato una quantità enorme di sprechi ed è risaputo che gli sprechi costano. Inoltre, l'alta velocità di sviluppo dovrebbe portare con sè l'introduzione di un'elevata qualità intrinseca. L'obiettivo è quello di raggiungere una forma standardizzata del processo in modo da rendere più facile l'interscambio di persone all'interno del progetto. L'interscambio in sé però non rende un progetto veloce, la velocità viene dettata dalle persone che fanno il proprio lavoro e che riescono ad adattarsi ai problemi che incontrano.

7.2 Kanban

Il Kanban è sostanzialmente una lavagna/cartellino dove viene tracciato, secondo poche ma ben determinate regole, lo stato dei diversi task, abilitando il team ad una modalità di lavoro:

- *Concorrente*, dove sono ridotti al minimo gli stati di attesa per svolgere il proprio lavoro (Just-In-Time), infatti rispetta la logica di produrre quello che serve, quando serve e nella quantità che serve.

- *Pull*, dove si annulla la necessità di una figura responsabile dell'erogazione del lavoro da svolgere.

- *Trasparente*, dove ognuno sa e può sapere cosa sta facendo l'azienda nel suo insieme in un dato momento.

Si ottiene quindi un abbattimento della burocrazia, il trasferimento del controllo verso il basso e la tracciabilità di tutte le attività, riuscendo così a passare dai programmi settimanali ai programmi giornalieri e addirittura oltre. In questo modo, è molto più semplice variare i programmi di produzione in caso di necessità (errori nelle stime, condizioni di mercato mutate, ecc.).

L'applicazione del pensiero Lean porta ad eliminare una classe di problemi e/o di sprechi nell'attività produttiva attraverso un approccio sistemico ovvero creando un ambiente di lavoro che rende difficile commettere errori come, per esempio, avere molte user stories (argomento trattato nel paragrafo 11.2.6) in lavorazione (semilavorati) che non portano valore al sistema, evitando così il sovraccarico di lavoro del team, permettendo invece di individuare molto velocemente eventuali parti critiche dello sviluppo di una user story, ecc..

Questo è reso possibile grazie alla **Kanban Board** (vedere figura 20). Al contrario di quello che si possa pensare, Kanban non sono le carte delle user stories,

ma rappresentano i limiti della produttività del sistema, cioè il limite imposto sul numero delle user stories che possono essere in lavorazione.

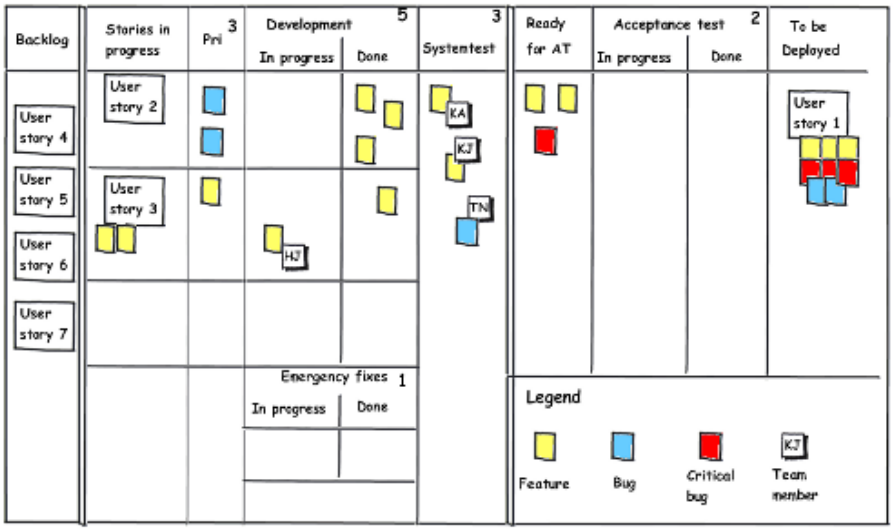

Figura 20: Un esempio di Kanban Board.

Per far funzionare Kanban in maniera efficace, è necessario seguire alcune regole fondamentali:

- Non consegnare materiale non funzionante al processo successivo.
- Produrre solo la quantità necessaria al processo successivo, bilanciando la produzione.
- Il kanban serve per l'ottimizzazione del processo di produzione.
- Stabilizzare e razionalizzare il processo.

7.2.1 Implementazione

L'implementazione di Kanban nello sviluppo di software riguarda tre punti fondmentali [27]:

- Visualizzare il flusso di lavoro: suddividere il lavoro in parti, scrivere ciascun elemento su un cartellino e attaccarlo al muro.
- Limitare il lavoro in produzione: assegnare dei limiti espliciti su quanti elementi possono essere in fase di produzione per ogni stato.
- Misurare il tempo medio per completare un elemento (lead-time), ottimizzare il processo per rendere il lead-time il più piccolo e prevedibile possibile.

7.2.2 Benefici

- I colli di bottiglia sono chiaramente visibili in tempo reale. Ciò conduce alla collaborazione tra persone proprio per ottimizzare l'intero processo e non solo una singola parte.
- Evoluzione più graduale per aiutare le aziende nel passaggio dal metodo waterfall al metodo agile.
- Fornisce un modo per *essere agili* senza essere vincolati dall'uso di iterazioni di lunghezza fissa come gli sprint Scrum (vedere paragrafo 6.1.2).
- Tende a diffondere informazioni all'intera organizzazione, il che aumenta la visibilità dei lavori svolti nell'azienda.

Parte IV

Caso di studio reale

8 Introduzione

Come anticipato nel sommario iniziale, il caso di studio esaminato riguarda l'adozione di metodologie *agili* nell'ambito di un **percorso di trasformazione** della divisione IT di una grande azienda.

Chiariamo innanzitutto un termine: *cosa si intende per caso di studio?* Un caso di studio reale è un'analisi effettuata su informazioni raccolte da fatti realmente accaduti, tracciando delle conclusioni.

8.1 Azienda Cliente

La divisione IT di cui si tratta è la Direzione Sistemi Informativi (Di.Si.) di un'importante realtà del panorama italiano della grande distribuzione organizzata (*retail*), con qualche centinaio di punti vendita di proprietà o in franchising.

Le dimensioni e la natura delle attività di un'azienda di questo tipo, richiedono il supporto di un sistema informativo complesso, in grado di rispondere alla rapida e continua evoluzione del business.

A partire dallo scorso anno, all'interno della Di.Si. è maturata l'esigenza di intraprendere un percorso di trasformazione della struttura (*transizione*) che portasse alla soluzione dei principali problemi che ne limitano l'efficacia dell'azione e ne ridimensionano il ruolo nell'ambito dell'azienda:

- Un elevato debito tecnico[12] - dovuto alla mancanza di un'architettura abilitante e un limitato skill tecnico del personale - che limita pesantemente la capacità di creazione di valore di business: ben il 60% delle risorse è impegnato dalla gestione operativa e solo 40% è investito in attività progettuali.

- Un basso livello qualitativo dello *stack*[13] applicativo.

- Un coinvolgimento degli interlocutori di business e una gestione della domanda in gran parte inefficaci.

[12]Per debito tecnico (termine ideato da Ward Cunningham) si intende una conoscenza non ottimale di tecnologie e linguaggi di programmazione necessari per la riuscita di un progetto. Il debito tecnico è la causa maggiore degli alti costi dello sviluppo di software (dimostrato dalla figura 8). Ci sono principalmente due modi di sviluppare: uno veloce ma disordinato, l'altro più lento ma con design più pulito. Il primo metodo crea più problemi rispetto al secondo perché le inefficienze emergono soltanto in un secondo momento, data dalla sua minore attenzione alla qualità del codice. Martin Fowler fa riferimento in [18] alla metafora degli interessi che si pagano con una metodologia di questo tipo.

[13]Per Stack applicativo si intende l'insieme degli applicativi necessario per svolgere dei compiti, nel nostro caso di studio l'insieme delle applicazioni disponibili in azienda.

L'analisi di tali problemi ha portato alla conclusione che la transizione, per potersi considerare di successo, dovesse da un lato portare a **ridurre il debito tecnico** al di sotto di una soglia fisiologica e, dall'altro, **migliorare il livello di collaborazione** con le unità organizzative di business (*business unit*).

Questi due obiettivi sono stati quindi declinati nel seguente modo:

- Costruzione di un'architettura abilitante di tipo SOA[14], in grado di meglio rispondere all'evoluzione del business.
- Aumento dello skill tecnico medio del personale che, fino ad ora, è limitato al linguaggio delle basi di dati (SQL) e a piccole modifiche ad hoc ai software attualmente a disposizione.
- Aumento della qualità intrinseca dello stack applicativo, in modo da ridurre i costi di gestione operativa e manutenzione.
- Adozione di un *approccio agile* alla gestione della domanda e alla raccolta dei requisiti.

> *"Non esistono progetti IT, ma progetti di business che hanno una componente IT abilitante."*
>
> *Citazione da una delle sessioni*

A questo punto, si è subito fatta chiara l'idea che tale trasformazione non potesse avvenire secondo un approccio *waterfall*, partendo da un'analisi completa e dettagliata di tutta l'azienda e dei suoi processi di business, una progettazione omnicomprensiva dell'architettura, ecc. ma dovesse realizzarsi mediante l'attuazione *agile* di un programma (insieme di progetti) che garantisca al contempo un'ininterrotta erogazione di valore verso il business.

La causa che ha scaturito la decisione di adottare delle metodologie diverse da quelle usate in passato è stata la sospensione di un progetto avvenuta lo scorso Ottobre 2010, determinata dalla constatazione di problematiche nate proprio dall'inefficacia e dall'inadeguatezza riscontrate durante la fase di valutazione da parte del cliente e, in particolare, dalla parte di business.

Il progetto consisteva nello sviluppo di uno strumento che potesse facilitare ed automatizzare la gestione dei prezzi di articoli presso i punti vendita dell'azienda.

[14]L'Architettura SOA (Service Oriented Architecture), o architettura orientata ai servizi, è realizzata dai Web Services (Servizi) che si presentano come moduli software distribuiti i quali collaborano tra loro e scambiano informazioni. Possono esporre sul Web, in modo sicuro e trasparente, la logica di business localizzata all'interno di un sistema aziendale e presentarla all'esterno.

Il progetto è stato sospeso, dopo un periodo di sviluppo iniziato nell'Aprile 2010 e finito nell'Ottobre 2010, per lo scarso coinvolgimento del Business e quindi, vista dal lato degli sviluppatori, per una scarsa conoscenza delle reali esigenze degli utenti. Inoltre, non sono state coinvolte in maniera efficace tutte le persone centrali del progetto (gli attori del sistema), come per esempio la figura dei Buyers.

Un ulteriore motivo della sospensione del progetto è stato la previsione di un cambio di modello volto alla riduzione del numero dei benchmark (spiegazione rimandata al paragrafo 11.1) sui quali i Buyers dovevano andare a lavorare. Questo per ridurre anche il carico di lavoro dei Buyers stessi, che si occupano anche di tutt'altro, non solo dei prezzi degli articoli.

È stato necessario quindi adottare dei cambiamenti profondi e creare una collaborazione più stretta tra la parte di business e la parte IT dell'azienda.

Per poter realizzare questo ambizioso disegno, si è scelto, come partner tecnico, un'azienda leader - a livello mondiale - nello sviluppo di progetti *agili* e nella consulenza sulle metodologie agili rivolta alle organizzazioni IT (*enablement*).

8.2 Partner tecnico

Una volta focalizzate le varie problematiche presenti nella parte IT, compreso il complesso contesto dell'azienda e la reale necessità di cambiare metodologia di lavoro, si è ipotizzato, appunto, di ingaggiare degli esperti come supporto nello sviluppo di un primo progetto e per l'assimilazione di tecniche più efficaci ed utili, affrontando così il problema del reparto IT.

L'ingaggio di ulteriori squadre di sviluppo però, anche se necessarie, non è sempre facilmente realizzabile perché chi finanzia i lavori deve essere convinto del fatto che, un'operazione di questo tipo, possa portare ad un reale beneficio all'azienda e non soltanto ad un dispendio finanziario. La riduzione dei costi di manutenzione, l'aumento della qualità del software, l'avvicinamento del business al team di sviluppo (quindi la realizzazione di strumenti realmente utili) sono fattori che influiscono sulla decisione di affidarsi ad ulteriori professionisti, esperti della nuova metodologia.

Una volta presa in considerazione l'idea di lavorare assieme a questo partner, è stato fatto un incontro esplorativo per capire le modalità di collaborazione tra i due "gruppi".

Il piano di lavoro condiviso durante tale incontro, nel quale sono state affrontate la fase organizzativa e di pianificazione del processo, si articolava quindi in due fasi (da completare entro dicembre 2010):

- Una **Discovery**, avente come obiettivo l'identificazione della soluzione architetturale e la selezione dei progetti per il programma.

- Un'**Inception** per un primo progetto (selezionato in base alla priorità posta durante la discovery), mediante la quale cominciare ad erogare valore ed attivare una nuova forma di collaborazione con le business unit.

Come anticipato nel capitolo riguardante le metodologie agili (capitolo 4), un grande coinvolgimento del cliente (nel nostro caso la parte di business) aiuta ad avere un'idea molto più chiara e precisa delle esigenze che il sistema deve soddisfare; ciò è dato anche alla sua partecipazione alla realizzazione del sistema stesso.

Per la seguente Discovery, la parte IT ha scelto di individuare al proprio interno i Business stakeholders perchè riteneva non ancora pronta la parte Business.

A valle dell'Inception sarebbe iniziata quindi la fase realizzativa (**Execution**) nell'ambito della quale sarebbe stato portato avanti l'enablement.

9 Discovery

9.1 Che cosa è una Discovery?

La fase di Discovery è una fase in cui si valutano le varie ipotesi per il proseguimento di un processo di sviluppo. In particolare, essa non riguarda soltanto un singolo progetto ma riguarda un sistema intero visto nel suo insieme, comprendente l'intera architettura.

Per quanto riguarda il seguente caso di studio reale, è stato effettuato ad inizio Dicembre 2010 e due sono i motivi che principalmente hanno spinto a favore della decisione di adottare questo metodo: identificare una soluzione architetturale dell'intero sistema, istruendo allo stesso tempo i nuovi collaboratori, e selezionare i progetti per il programma di trasformazione dell'IT.

In questa fase sono coinvolti tutti gli stakeholder legati al progetto: possono essere i manager di business, il cliente vero e proprio (inteso come proprietà), gli analisti, gli architetti, ecc.

9.2 Visione condivisa

La proposta di un piano da presentare alla proprietà ha implicato la necessità di avere una visione globale dello *stato dell'azienda* nel presente (come è ora, da chi è composta, ecc.) e un'attenzione particolare ai processi complessivi presenti in essa, indicandoli, in un primo momento, soltanto ad alto livello. Limitarsi ad individuare i punti fondamentali ad alto livello garantisce il fatto di strutturare più efficientemente e di individuare l'architettura dell'intero sistema informativo aziendale, quindi non solo di un singolo progetto.

La potenza dell'agile sta proprio nella capacità di modulare la descrizione di un determinato ambito in relazione alle esigenze del cliente, senza entrare particolarmente nel dettaglio, ottenendo così una visione globale dell'architettura e dello scope.

La necessità di trasformazione verso questo approccio deriva dall'esigenza di definire una architettura enterprise e riorganizzare e qualificare la struttura del sistema informativo aziendale. Questo implica, inoltre, la correzione delle inefficienze e dei problemi presenti ora nel sistema ed il miglioramento dei processi.

Secondo l'agile il livello di dettaglio deve essere correlato alle esigenze e al momento in cui serve, in base al progetto che si decide di sviluppare. Non è necessario, come invece per le metodologie tradizionali, analizzare tutta la struttura in maniera molto dettagliata già all'inizio di un progetto. Basta avere uno schema scheletro che

Figura 21: L'insieme degli elementi per avere una visione condivisa del sistema (1° parte).

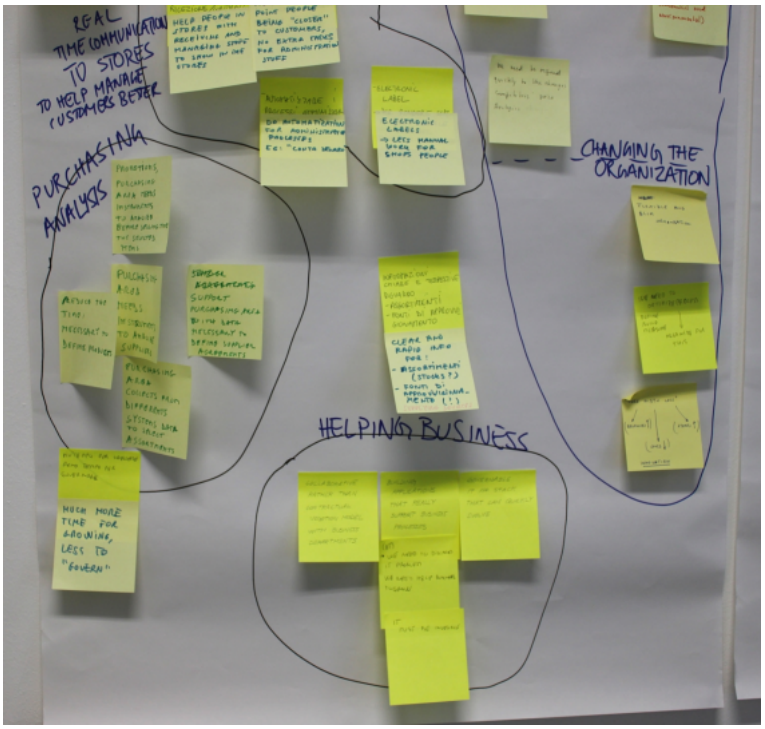

Figura 22: L'insieme degli elementi per avere una visione condivisa del sistema (2° parte).

dia le informazioni sufficienti per iniziare il progetto stesso e poi raffinare progressivamente la conoscenza del contesto quando si affrontano man mano le funzionalità da sviluppare. Analizzare approfonditamente sin da subito potrebbe portare ad un impiego di tempo troppo elevato per delle cose che magari, dopo poco tempo, risultano obsolete e quindi non più necessarie, il che risulta essere controproducente.

Il giusto livello di dettaglio consente di partire con un progetto con delle tecnologie e il design che fanno in modo di integrare facilmente del software nuovo o correggere qualcosa di esistente.

Il valore aggiunto che restituisce questa pratica è la visione complessiva e condivisa dell'architettura di sistema, arrivando ad un giusto dettaglio che consente di capire adeguatamente l'ambito.

9.2.1 Hopes & Concerns

Sono state, inoltre, individuate le **aspettative** (*hopes*) di tutti i partecipanti della riunione con le relative **preoccupazioni** (*concerns*), cioè le parti che potrebbero creare problemi in fase di sviluppo. Avere una visione globale e soprattutto condivisa dell'idea di business è un requisito fondamentale per progetti di grandi dimensioni e riguardanti clienti importanti perché garantisce chiarezza quando si discutono le soluzioni ai problemi.

Nelle figure 23 e 24 si vede come è stata realizzata la parte riguardante le aspettative e le preoccupazioni: utilizzando una parete libera e dividendola in due sezioni (Hopes e Concerns). Ciascun partecipante ha scritto successivamente le proprie opinioni su dei **post-it** e li ha poi appiccicati nella sezione appropriata, cercando di raggruppare per argomento i foglietti (procedimento conosciuto con il nome di *grouping*) con quelli dei colleghi che erano già presenti. Quando tutti avevano terminato, sono stati analizzati tutti i post-it e si sono messe in relazione le idee ed i punti di vista in modo da creare una visione accettata e compresa da tutti.

Con questa tecnica sono emerse molte aspettative che delineano le principali problematiche che la trasformazione dell'IT dovrà risolvere, per esempio la realizzazione degli obiettivi elencati in precedenza, la riduzione del tempo da quando il cliente presenta un bisogno da soddisfare a quando l'applicazione specifica viene rilasciata, ecc.. Gli *hopes* rappresentano quindi, dal punto di vista del partner tecnico e della divisione IT, i criteri di soddisfazione del cliente a cui fare costantemente riferimento.

Per quanto riguarda le preoccupazioni sono emersi altri fattori più tecnici come il problema della lingua (i collaboratori del partner tecnico erano principalmente indiani e quindi è stato necessario parlare in inglese), la paura di non riuscire a pianificare bene i tempi delle sessioni e non riuscire a definire correttamente l'architettura e la

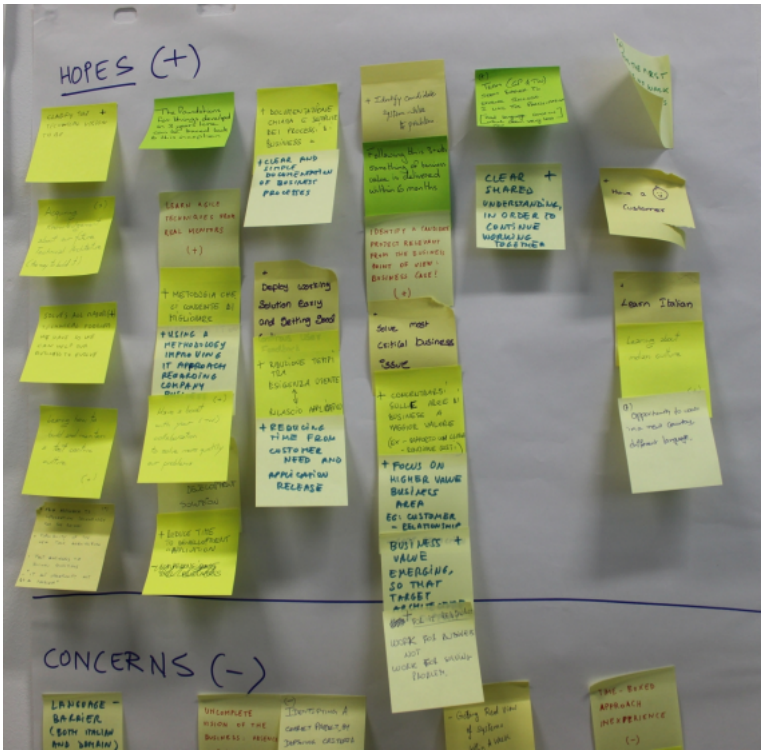

Figura 23: Le aspettative degli stakeholder.

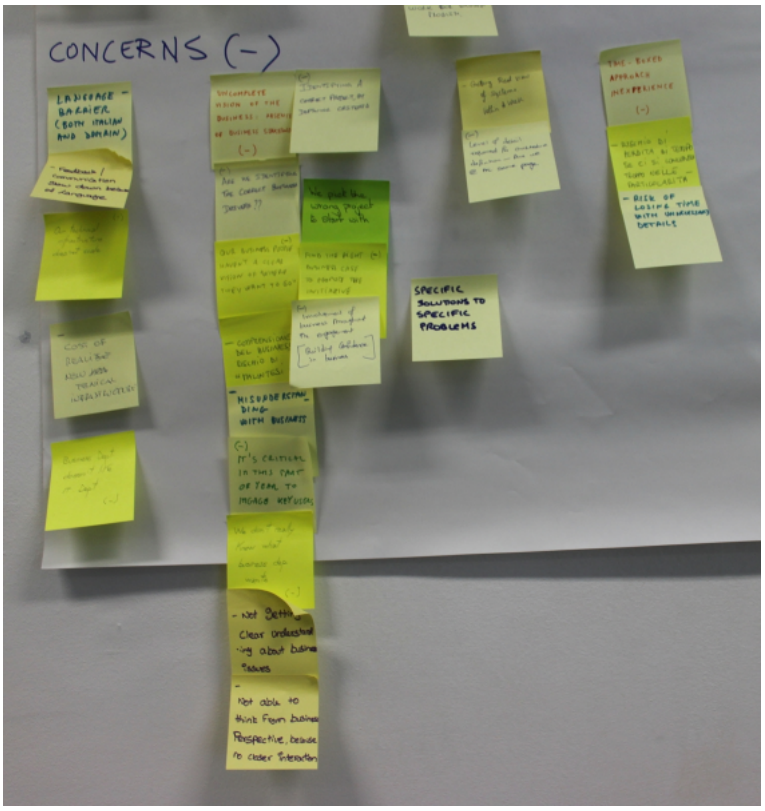

Figura 24: Le preoccupazioni degli stakeholder.

visione del business, rischiando così alzare ulteriormente i costi dell'infrastruttura, ecc. I *concerns* rappresentano quindi, dal punto di vista del partner tecnico e della divisione IT, i potenziali rischi da tenere sotto controllo.

Pit Falls Assieme alle preoccupazioni degli stakeholders è stato ritenuto utile soffermarsi maggiormente sulle criticità ravvisate nel modello architetturale delineato, dedicando 30 minuti circa per esaltare i problemi riscontrati. Alla stessa maniera sono state esaminate le aspettative e le preoccupazioni, dividendole per argomenti specifici e scrivendo su dei post-it i fattori emersi.

Come si può notare anche dalle immagini, nella metodologia agile i post-it occupano un ruolo centrale nella gestione dei progetti.

9.3 Scelta del progetto candidato

Dopo il consolidamento della visione globale e condivisa dei processi si è potuto affrontare la seconda parte della fase di Discovery che consiste nell'individuazione di un progetto candidato con cui iniziare l'applicazione di un approccio agile per la trasformazione dell'azienda.

9.3.1 Oppuntunity Landascape

È stata considerata fondamentale la scelta del progetto con il quale iniziare questo nuovo percorso proprio perché l'avvio di un processo è sempre una fase molto delicata. Inizialmente, il dubbio consisteva nello scegliere un progetto "minore" oppure sfruttare il progetto appena fallito, ma anche scegliere opportunamente su quale ambito di progetti concentrarsi maggiormente. Dopo alcuni dibattiti però è stata ritenuta più idonea una votazione per individuare una rosa di progetti candidati (**Opportunity Landscape**, vedere figura 25) su cui focalizzare la scelta. Per questo strumento è stato utilizzato come di consueto la tecnica dei post-it coinvolgendo tutti i partecipanti. Oltre alla categoria dei progetti, sono stati riportati anche i punti di forza e di debolezza propri di ciascun ambito per dare la possibilità di votare l'insieme dei progetti nel migliore dei modi.

9.3.2 Matrice di confronto

La scelta del progetto candidato è stata fatta applicando ai progetti della rosa scelta in precedenza un insieme di criteri, costruendo così una matrice (vedere figura 26). In particolare, sono stati evidenziati, per ciascun progetto, quali potessero essere i

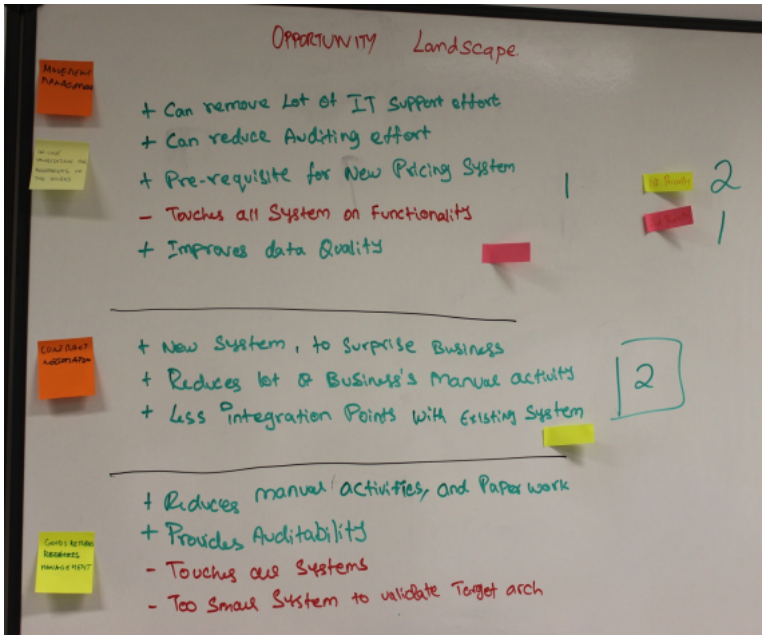

Figura 25: L'elenco di Opportunity Landscape emerso dalla riunione.

problemi che il progetto risolverebbe e quali invece trascurerebbe, i punti di integrazione con il sistema, la complessità e la probabilità di realizzare il progetto in 3 mesi, ecc..

Figura 26: La matrice di confronto con i progetti candidati.

Assieme alla fase iniziale di condivisone delle idee, la fase di Discovery riguarda il sistema informativo aziendale, dunque non soltanto un singolo progetto; per portare a termine quest'ultimo compito è stata effettuata un'Inception (trattata nel capitolo

10 e capitolo 11). Una delle problematiche maggiori che si possono riscontrare nelle grandi organizzazioni è la rapidità con la quale cambia il contesto di un'azienda. Per questo motivo è stato chiesto, durante gli incontri, come ciascun partecipante vedesse l'azienda nell'arco dei prossimi 5 anni. Ciò è stato di grande aiuto per l'individuazione del progetto candidato all'Inception perché ha permesso una maggiore sicurezza, diminuendo così la probabilità di scegliere progetti non utilizzabili nel futuro.

Durante la scelta del progetto ci sono state alcune difficoltà nel misurare in maniera adeguata il valore di business. Il team di sviluppo non era certo del fatto di scegliere, dal punto di vista del team di sviluppo, un progetto significativo per il business e quindi si è deciso di scegliere il progetto SIM (verrà affrontato nel paragrafo 10.2).

9.4 Parking Lot

Durante gli incontri sono emerse, inoltre, alcune funzionalità che esulavano dall'argomento trattato in quel momento oppure argomenti interessanti ma che implicavano una discussione più approfondita e quindi di durata maggiore di quella a disposizione nelle varie sessioni.

È stato introdotto perciò un metodo, chiamato **Parkling Lot** ed utilizzato principalmente dal FDD (Feature Driven Development), che non elimina direttamente gli argomenti non trattabili in quel dato momento ma vengono "parcheggiati" (da cui il nome) in una lista di funzionalità/argomenti che verranno discusse alla fine del progetto o quando il tempo durante le sessioni lo permette.

9.5 Retrospettiva e Showcase

Secondo le metodologie agili è fondamentale avere dei riscontri su ciò che è stato fatto, come descritto nel paragrafo 6.4.3. Sono stati utilizzati principalmente due modi diversi durante le fasi: la retrospettiva e lo showcase. La **retrospettiva** serve a catturare i feedback relativi al lavoro svolto e lo **showcase** ha lo scopo di condividere lo stato del progetto e, nel caso dello showcase finale, a trarne le conclusioni finali. Nel nostro caso sono stati usate dei diagrammi particolari, chiamati **Starfish diagrams** (vedere figura 27), che hanno lo scopo di evidenziare ciò che è stato realizzato in modo appropriato oppure ciò che occorre migliorare. In particolare è importante sottolineare come è stato diviso il grafico in *cinque* settori ("Iniziamo a fare...", "Continuiamo a fare...", "Facciamo meno...", "Facciamo meno...", "Smettiamo di fare..."), per ciascuno dei quali i partecipanti al progetto hanno inserito le

proprie idee. Questa è una pratica che aiuta a migliorare di sessione in sessione il lavoro svolto dalla squadra (*kaizen*, trattato nel paragrafo 7.1.2).

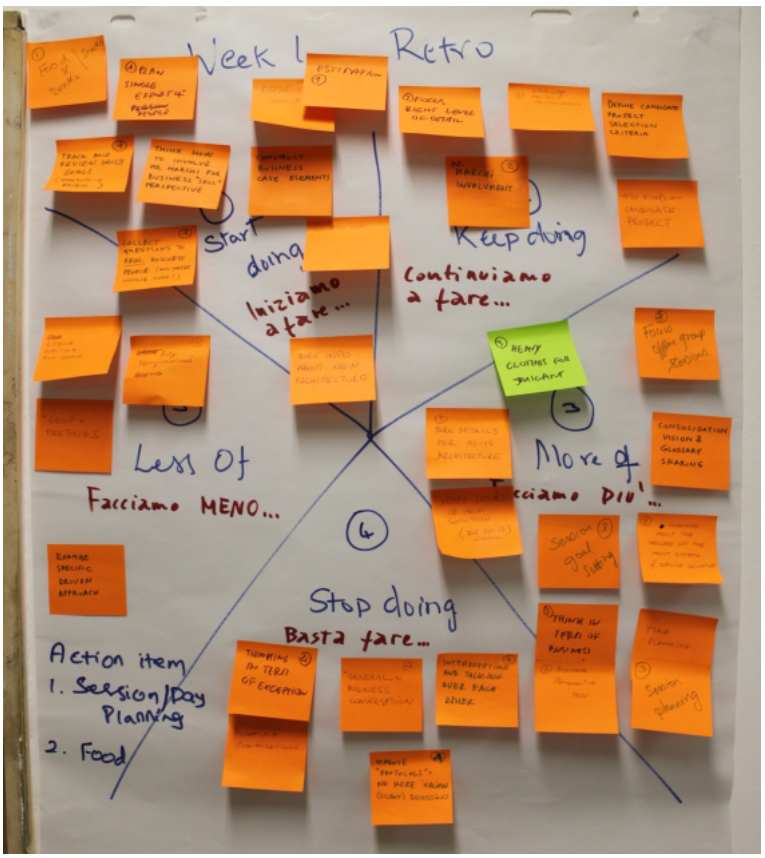

Figura 27: Un starfish diagram.

10 Inception 1

10.1 Che cosa è una Inception?

L'Inception è una fase del processo di sviluppo che ha lo scopo di studiare in modo approfondito i bisogni del cliente e di produrre le specifiche dei requisiti in modo tale che siano chiare, le più complete possibili e funzionali, utilizzando varie tecniche per far emergere tutti i bisogni e i rischi legati al progetto, così da consentire una *visione comune/condivisa di una soluzione*. È una fase molto importante e molto spesso viene sottovalutata: infatti, più un'Inception è di successo e migliore sarà il software che si andrà a realizzare.

Lo scopo dell'Inception è produrre una **visione condivisa degli obiettivi** e dei **contenuti di un progetto**.

Per fissare tutti i requisiti ed evidenziare i rischi riguardante il progetto sono state effettuate delle sessioni con il cliente e gli altri stakeholders, durante le quali si sono consolidate la conoscenza della realtà aziendale.

Nella fase di Inception, generalmente, si esegue lo studio di fattibilità del progetto e si cerca di comprendere il contesto organizzativo del cliente, cioè reperire tutte le informazioni necessarie per la realizzazione del progetto, come per esempio: il tipo di mercato del cliente, gli attori del sistema ed identificare gli elementi importanti e cruciali affinché esso conduca ad un successo commerciale. Vengono stabiliti così i parametri, i requisiti del sistema e la strategia che il cliente vuole adottare, tutto questo prima che i programmatori siano coinvolti.

10.1.1 Visione iniziale del progetto

All'inizio di ogni nuovo progetto, le persone di solito hanno idee molto diverse su ciò che porta un progetto ad essere un successo (vedere la prima parte della figura 28). Questo può portare al fallimento dei progetti, come è successo in precedenza, perché anche se tutti usano, per esempio, le stesse parole per descrivere un concetto, ci si accorge delle divergenze soltanto quando si entra più in dettaglio, come è evidenziato appunto in figura 28.

Ciò che serviva, era un modo di comunicare gli obiettivi, la visione e il contesto del progetto alla squadra di sviluppo e fornire tutte le informazioni di cui si ha bisogno per poter prendere decisioni per procedere o meno con il progetto. Per acquisire tutte queste informazioni dagli stakeholders, il modo più efficace è quello di utilizzare discussioni nelle quali è stato chiesto a ciascun interessato di esprimere la propria opinione al riguardo.

Molti progetti falliscono prima ancora di partire a causa della mancanza di dialogo e di chiarezza tra gli interessati. Per la buona riuscita di un progetto è necessario porsi molti quesiti per identificare i punti fondamentali del progetto. Non si può sviluppare un prodotto se non si conosce il perché ne è stata commissionata la realizzazione. È indispensabile conoscere profondamente l'obiettivo principale, affinché durante lo sviluppo si possano prendere le decisioni più intelligenti e corrette.

Figura 28: Il procedimento che mostra l'iniziale diversione delle idee che poi vengono convogliate in una unica e condivisa.

10.2 Progetto SIM

Il primo progetto selezionato per la fase di Inception è stato **SIM** (acronimo per Supplier Information Management), uno strumento destinato ai Buyers per l'acquisizione delle schede prodotte dai fornitori. L'idea era che i Buyers, alla ricezione di una proposta di vendita di un nuovo prodotto da parte di un fornitore, potessero richiedere a quest'ultimo la compilazione della scheda prodotto tramite un'applicazione web, SIM appunto.

Le esigenze recepite dai Buyers erano quelle di avere risposta ai seguenti problemi:

- L'onerosità del censimento di un nuovo articolo (un'operazione farraginosa, che richiede molto tempo ed è soggetta a molti errori).
- Lo scarso monitoraggio della qualità e della tempestività della risposta da parte dei fornitori.

Si era pensato di proporre quindi uno strumento che consentisse di implementare un modo standard, facile e veloce per censire un nuovo prodotto, nonché di fare sì che fossero i fornitori a mantenere aggiornato e fruibile il loro catalogo prodotti in modo tale i Buyers potessero di volta in volta scegliere gli articoli da comprare.

10.3 Sospensione

Dopo appena un paio di giorni l'Inception del progetto SIM è stata sospesa, per due ragioni:

- Il rischio intrinseco derivante dall'aver basato il funzionamento del sistema sulla disponibilità - non verificata - dei fornitori al suo utilizzo.
- La difficoltà di attribuire ad esso - in modo convincente - una priorità in termini di valore erogato, a causa dell'assenza degli interlocutori lato business.

La situazione che si è venuta così a determinare è stata gestita con un approccio adattativo e improntata alla collaborazione. Si sono registrati gli elementi utili a costruire una visione chiara e condivisa - presentata poi nello showcase finale - e definito un piano di azione per arrivare alla selezione di un nuovo progetto candidato e quindi sospeso.

L'individuazione del nuovo progetto è avvenuta nelle settimane seguenti al di fuori dell'agenda di lavoro inizialmente concordata con il partner tecnico, interagendo direttamente con le business unit, in particolare la Direzione Acquisti.

10.4 Scelta "offline"

Dopo aver analizzato nuovamente l'Opportunity Landscape (paragrafo 9.3.1 riguardante l'ambito dei progetti) e la Matrice di confronto (l'elenco di tutti i progetti dell'ambito specifico), è stato scelto il **progetto di Pricing** per rompere il ghiaccio ed iniziare con un progetto importante e critico, sfruttando, inoltre, l'occasione per introdurre nuove funzionalità ritenute dal Pricing Manager irrinunciabili.

La modalità di ingaggio dei business stakeholders, cioè Price Manager, Buyers, ecc., ed il perimetro entro cui devono essere individuati, è stata affidata alla parte IT, il che poteva risultare un rischio per l'esito dell'iniziativa di trasformazione e qualcuno sarebbe potuto non essere preso in considerazione.

Dopo aver preso questa decisione, è emerso un'ulteriore problematica: le parti di business hanno esternato delle perplessità per quanto riguarda la quantità di impegno (il coinvolgimento in termini di tempo) che dovevano dedicare al progetto e come riuscire a continuare a svolgere le proprie funzioni lavorative abituali contemporaneamente. Per aiutare a realizzare uno strumento che porti effettivamente valore aggiunto al proprio lavoro e che sia allo stesso tempo di qualità ed utile, hanno dovuto attuare un profondo cambiamento culturale e organizzativo per garantire un impegno corposo e costante per questa attività.

11 Inception 2

11.1 Progetto Pricing Tool

Il progetto Pricing Tool riguarda lo sviluppo di uno strumento a supporto del pricing basato sulla concorrenza, il processo di determinazione del prezzo di vendita dei prodotti in relazione al comportamento della concorrenza, secondo lo slogan "il prezzo lo fa il mercato".

Pricing Pricing è il processo che stabilisce il prezzo da applicare ai prodotti di una società secondo una strategia specifica, considerando il costo di produzione, il mercato (assieme alle sue condizioni), la concorrenza e la qualità del prodotto.

L'andamento del mercato è valutato localmente all'interno di un'area (bacino) - che può comprendere una o più città - nell'ambito della quale si identificano i punti vendita delle aziende concorrenti (competitors) presso cui vengono periodicamente rilevati (sia direttamente sia tramite aziende specializzate come Nielsen[15]) i prezzi degli articoli di riferimento (presenti anche nell'assortimento dei negozi della propria catena). A partire dai dati rilevati viene calcolato un benchmark di mercato (il prezzo più basso all'interno del bacino), rispetto al quale viene determinato il prezzo di vendita (il prezzo che compare sull'etichetta in scaffale), secondo una formula di calcolo basata su dati di mercato relativi al comportamento della clientela. L'obiettivo è quello di aumentare la percezione di convenienza.

Il processo vede coinvolte due unità organizzative dell'azienda (evidenziate anche in figura 29): la Direzione Marketing Operativo e la Direzione Acquisti.

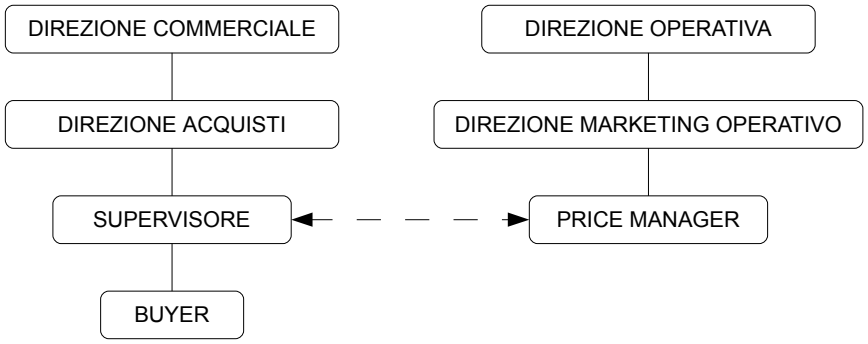

Figura 29: La divisione delle unità organizzative all'interno dell'azienda.

[15]http://it.nielsen.com/site/index.shtml

La prima - in particolare nella figura del Pricing Manager - ha la funzione di recepire e attuare le linee strategiche definite dal top management, individuando i concorrenti di riferimento, organizzando il calendario delle rilevazioni e la composizione dei panieri, declinando la politica dell'utilizzo della formula di calcolo.

Della seconda, invece, fanno parte:

- I Buyers, i quali si occupano di controllare i prezzi rilevati, vagliare il benchmark e - in casi eccezionali - di intervenire sul prezzo calcolato.
- Il Supervisore, che ha il compito di verificare l'operato dei Buyers e di valutare il raggiungimento degli obiettivi di posizionamento complessivo rispetto ai concorrenti assegnati dal top management, nonchè di modificare - d'accordo con il Pricing Manager - i valori della matrice, se necessario.

Come si può evincere dall'analisi del progetto di seguito esposta, i vari attori del processo sono portatori di interessi in parte contrapposti. L'interesse del top management e del Pricing Manager è quello che l'azienda acquisisca, agli occhi della propria clientela, un'immagine complessiva di convenienza e migliori il livello di competitività rispetto ai concorrenti che contendono la medesima quota di mercato. I Buyers, invece, sarebbero "naturalmente" interessati ad usare il pricing per perseguire i propri obiettivi economici di margine (rispetto a cui sono misurati), in aggiunta alle leve previste dal top management, quali la selezione dell'assortimento, le promozioni e la politica negoziale con i fornitori.

Uno degli obiettivi del progetto era quello di dare ai Buyers uno strumento veloce e facile da usare, che consentisse di ridurre il loro carico di lavoro manuale, gestendo al contempo le loro richieste in modo da non inficiare la strategia di pricing dell'azienda.

11.2 Come è stata fatta?

La seconda Inception ha avuto luogo dopo la discovery, dal 10 al 21 gennaio scorso.

11.2.1 Timeboxed

È stato deciso di adottare una gestione **timeboxed** del tempo di lavoro (come in Scrum, capitolo 6), articolando la giornata in *quattro sessioni della durata di 90 minuti*, nelle quali si sono trattati gli argomenti secondo un piano settimanale e cercando di non sforare con i tempi.

Questa divisione rigida del tempo aumenta sensibilmente la produttività perché costringe a raggiungere un obiettivo nel più breve tempo possibile. I succes-

sivi paragrafi evidenziano questi obiettivi ed indicano quali strumenti sono stati utilizzati.

11.2.2 Kick-off

Lo scopo del Kick-off è quello di sancire in modo formale l'avvio del progetto, dandone una visione del contesto che ne spieghi ragioni ed obiettivi, in altre parole rispondere alla domanda: *Perché siamo qui?*

Porsi questa domanda è importante perché è necessario spiegare in modo esaustivo i motivi per cui è stata presa le decisione di iniziare un progetto. Molti degli interessati potrebbero non sapere o non riconoscere l'esigenza di avere o di poter utilizzare uno strumento che li aiuti nel proprio lavoro. Durante questi colloqui vengono spiegati gli obiettivi ed i beneficiari dello strumento stesso.

La visione del contesto può comprendere anche una delle "storie" del progetto. Infatti, è stato utile fare una breve panoramica riguardante il progetto precedente del *Price*, grazie alla condivisione del glossario dei termini e all'esposizione delle difficoltà riscontrate.

A questo punto è stato chiesto, a ciascuno stakeholder, un'idea personale (quindi ad alto livello, senza entrare nei dettagli) della visione complessiva del progetto in modo da riconoscere il punto di vista di ognuno.

Il risultato del Kick-off è una visione del progetto da più punti di vista (che spesso sono estremamente differenti) e alla fine dell'Inception bisogna ottenere una visione complessiva che soddisfi le divergenze (una visione *condivisa*) e che venga accettata da tutti quanti.

11.2.3 Rompere il ghiaccio

Prima di questo però, è stato ritenuto più importante affrontare un altro problema: quello di costruire un gruppo. La squadra dell'Inception era composta da molte persone, la maggior parte delle quali non si era mai incontrata prima di queste riunioni, anche se colleghi. Questo è fisiologico all'interno di aziende di grandi dimensioni con una struttura organizzativa e logistica articolata (più sedi, più reparti, ecc.).

Uno dei princìpi più importanti della metodologia agile è quello di creare un gruppo coeso per rendere il lavoro più agevole e colloquiale ed evitare possibili ed inutili prese di posizione, per imporre la propria idea ai singoli individui, che bloccano il lavoro. Per raggiungere questo, è stato fatto una specie di *gioco* usando le cosiddette *Profiling cards*.

11.2.3.1 Profiling Cards

Le profiling cards non sono altro che delle schede che riportano delle informazioni e che raffigurano le persone che compongono la squadra (vedere figura 30). Il gioco, chiamato anche *Icebreaker* (in inglese rompi-ghiaccio), consiste nel formare delle coppie di persone che **non** si devono conoscere. Per ciascun componente della coppia, si effettuano delle interviste, si disegna poi (a mano libera) una caricatura dell'intervistato per avere una sorta di foto tessera. Successivamente, una volta terminate tutte le schede, a turno, vengono esposte le informazioni acquisite dall'intervista a tutti gli altri componenti del gruppo.

Queste informazioni condivise consentono di mettere a contatto e poi consolidare il rapporto fra i singoli individui in modo da creare un ambiente di lavoro amichevole e sereno. In alcuni fasi poi si può assistere anche a dei momenti di divertimento insieme (per esempio durante l'osservazione delle caricature disegnate).

Figura 30: Le profiling cards realizzate durante la fase di Team builiding.

11.2.4 Verso una visione condivisa del progetto

Al termine della prima fase di definizione degli obiettivi generali e della formazione della squadra, è stato affrontato il problema della rilevazione delle esigenze di ciascun attore che utilizzerà il sistema. Anche per questa pratica, dopo aver compreso gli obiettivi principali, è stato scelto di chiedere alle persone cosa si aspettassero dal sistema, in termini di funzionalità, e quali fossero le proprie preoccupazioni.

11.2.4.1 Hopes & Concerns

Un modo potente per iniziare a strutturare un progetto e scoprire le esigenze degli stakeholders è quello di utilizzare la tecnica degli *Hopes & Concerns* (come avvenuto nella Discovery, nel paragrafo 9.2.1) intervistando gli interessati. Lo scopo di questa tecnica è far emergere le condizioni di soddisfazione rispetto a cui valutare l'andamento del progetto (*hopes*) e i potenziali rischi, vincoli, ecc. (*concerns*) di cui tenere conto.

Figura 31: Una lavagna con l'insieme delle aspettative riscontrate.

Figura 32: Una lavagna con l'insieme delle preoccupazioni riscontrate.

Per aiutare l'esposizione delle aspettative è stato deciso di utilizzare un altro strumento molto utile per evidenziare le funzionalità principali che, secondo gli stakeholders, il sistema deve soddisfare: quello del *Product in a box*.

11.2.4.2 Product in a box

Per focalizzarsi sui veri punti fondamentali e sulle parti di valore che deve avere il sistema è stato deciso di realizzare un **Product in a Box** (figura 33). Si tratta di una sorta di "promozione del prodotto", nella quale si evidenziano le caratteristiche principali come se fosse una pubblicità. Spesso, infatti, viene fatto il paragone con una scatola di cereali sulla quale si trova l'immagine che caratterizza il prodotto ed i suoi punti di forza, per esempio la grande quantità di calcio o vitamine. Con poche semplici frasi bisogna far in modo di convincere un ipotetico cliente a comprare il prodotto, sottolineando tutti i benefici ma evitando inoltre di essere troppo complessi ed elaborati. Con il termine beneficio si intende una funzionalità che caratterizza il prodotto e che risponde ad un bisogno dell'utente.

Per una buona sintesi delle caratteristiche è sufficiente elencare i tre benefici più importanti che si traggono dall'acquisto di questo prodotto ed è utile inventare anche uno slogan, che serve appunto ad invogliare e convincere il cliente.

Figura 33: La forma di un Product in a Box per evidenziare le caratteristiche del prodotto.

Questa pratica mette in evidenza anche un ulteriore concetto, quello della prioritizzazione: infatti pensare ai benefici e all'ordine con il quale vengono elencati è un primo modo per evidenziare le funzionalità più importanti (sebbene ad alto livello) e per mettere delle priorità. Lo scopo di questa tecnica è quello di soffer-

marsi unicamente sulle esigenze che si hanno parlando dei benefici che si ottengono, tralasciando, per il momento, la ricerca della soluzione.

Elevator Pitch Elevator significa ascensore. L'Elevator pitch è infatti il discorso che un imprenditore farebbe ad un investitore se si trovasse per caso con lui in un ascensore. L'imprenditore, quindi, si troverebbe costretto a descrivere sé e la propria attività sinteticamente, chiaramente ed efficacemente per convincere l'investitore ad investire su di lui, ma nei limiti di tempo imposti dalla corsa dell'ascensore. Esso serve per cominciare a definire il progetto, il suo scope, il valore di business e i principali beneficiari.

È stato deciso di **non** eseguire un Elevator Pitch, a differenza dell'Inception riguardante il progetto SIM (paragrafo 10.2), perché lo si è considerato troppo rischioso, dato i punti di vista diversi dei Pricing Managers e dei Buyers.

Oltre al dialogo aperto con gli attori è stato ritenuto utile utilizzare anche un modello che mettesse in evidenza gli interessi degli stakeholder che fino a questo punto non erano ancora sufficientemente precisi.

11.2.4.3 Modello Stakeholder/Interessi

Il modello Stakeholder/Interessi (vedere figura 34) è un modello, ideato da Alistair Cockburn [29], che serve principalmente ad individuare tutti gli stakeholders e, per ciascuno di questi, cerca di evidenziare i relativi interessi che il sistema dovrebbe soddisfare o proteggere.

Stakeholder $\neq$ Attore In generale, si è stabilito che uno stakeholder è qualcuno che detiene degli interessi rispetto al sistema, che a sua volta implementa un *contratto* tra questi; un attore, invece, è qualcuno che utilizza il sistema in prima persona.

Durante le discussioni è emerso anche una divergenza per quanto riguarda la terminologia: uno stakeholder può **non** essere un attore del sistema SUD (acronimo di System Under Discussion, in italiano Sistema sotto analisi). Essi sono quindi concettualmente diversi. Per esempio: il cliente finale che compra prodotti in un punto vendita non è un attore del sistema perché non utilizzerà mai lo strumento, ma ha degli interessi rispetto al sistema perché il suo ruolo influisce certe decisioni da prendere.

Lo scopo essenzialmente di questo modello è quello di esplicitare un possibile insieme di interessi, per esempio gli interessi dei Buyers che potevano essere conflittuali rispetto agli interessi del Top Management. La potenza di questo strumento

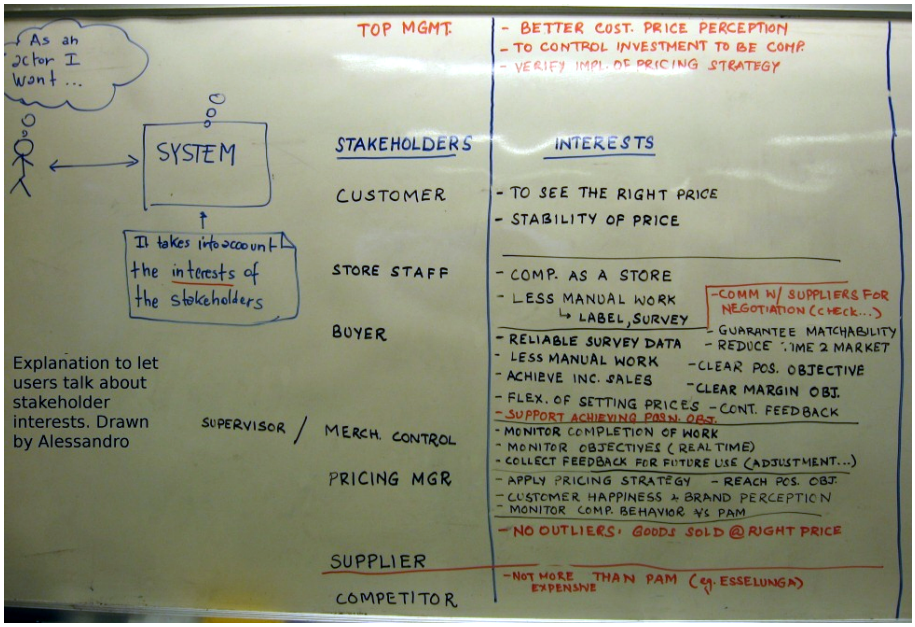

Figura 34: Il modello Stakeholder/Interessi.

sta proprio nel modo con cui mette in evidenza i fattori rendendo espliciti i contrasti tra gli interessi. Senza di esso non si riesce a cogliere appieno queste eventuali *situazioni di conflitto o di non compatibilità*.

Con il seguente modello è stata affrontata anche la parte riguardante i **Roles & Goles** (Ruoli ed Obiettivi), definendo nella tabella tutti gli stakeholders (ruoli) con i propri interessi (obiettivi). Tipicamente, viene utilizzato questo strumento (Roles & Goles) piuttosto che il modello stakeholders/interests ma è stato ritenuto più efficace quest'ultimo visto che, in un certo senso, lo racchiude al suo interno e le informazioni si possono comunque acquisire attraverso il modello.

La terza colonna, quella delle leve (*Levers*, in figura 35), è stata aggiunta solo in un secondo momento per evidenziare le azioni disponibili che possono intraprendere gli stakeholders per potere perseguire o proteggere i propri interessi e condividerle. È nata questa esigenza dopo che è emerso che i buyers non hanno la possibilità di decidere il prezzo degli articoli e quindi di modificare le matrici di competitività (che è compito dei supervisori o del pricing manager). È stato quindi deciso di aggiungere le leve per distinguere maggiormente le azioni che possono intraprendere gli stakeholders in modo da averle su un'unica tabella e da tenerne conto quando si affronterà la progettazione delle autorizzazioni per i cambi dei prezzi.

I buyers hanno la possibilità di raggiungere i propri margini grazie ad un miglior assortimento di prodotti oppure attraverso il co-marketing (collaborazione con il fornitore del prodotto ottenendo dei benefici, per esempio le promozioni). Questa attività però esula dal vero obiettivo del progetto, per esempio, quello di puntare

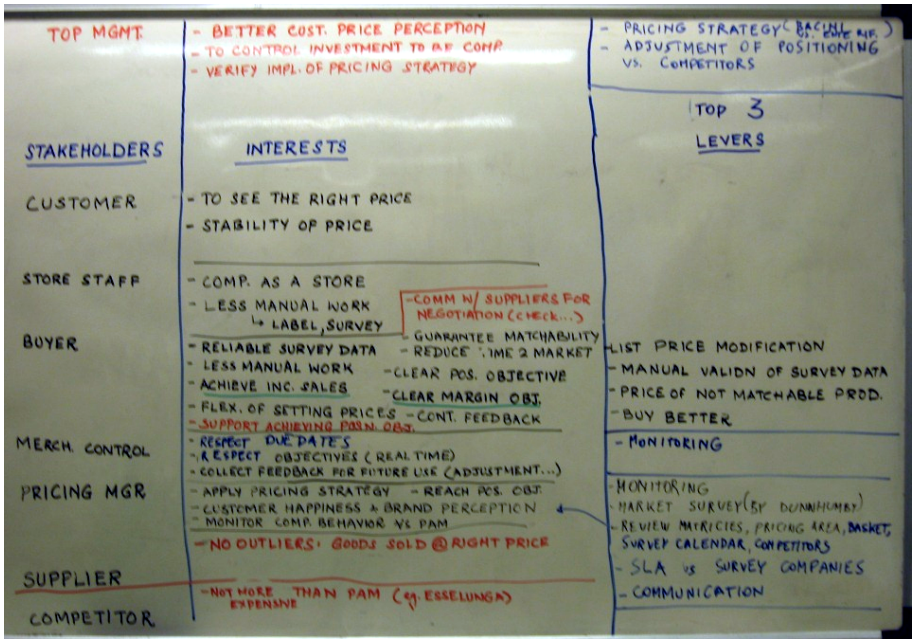

Figura 35: Il modello Stakeholder/Interessi con l'aggiunta della terza colonna.

sulla percezione (per il cliente finale) di una maggiore convenienza, e non, appunto, di monitorare le leve dei buyers.

Questo modello è molto utile per evidenziare (e anche in modo immediato) le differenze tra gli interessi degli stakeholders e rispondere quindi alla domanda: *come fanno gli stakeholders a raggiungere i propri interessi?*

11.2.5 Individuazione delle funzionalità

L'inception, come già detto, è utilizzato per definire i **contenuti del progetto** in termini di scope, cioè l'insieme delle funzionalità totali, e come questi possono essere riversate su un **piano di rilascio**. Ciascun rilascio contiene un certo numero di funzionalità decise nella compilazione del piano riguardante quello specifico rilascio in modo incrementale (argomento trattato nel paragrafo 11.2.8).

Le funzionalità sono le risposte alle esigenze che vengono evidenziate confrontando i diversi punti di vista per trovare le funzionalità che deve avere il sistema. Le funzionalità poi, partendo da quelle ad alto livello e definendo man mano il modello, sono state analizzate più nel dettaglio fino ad arrivare alle user stories che definiscono lo scope stesso.

L'individuazione del processo racchiude in sé la comprensione delle attività svolte nell'ambito del processo (pricing) e l'individuazione delle funzionalità a livelli progressivi di dettaglio che devono essere contenute nello scope. Si scende quindi progressivamente più nel dettaglio e ci si ferma quando una funzionalità è compren-

sibile e misurabile da parte di chi sviluppa. Le funzionalità compongono il piano di rilascio che viene deciso dal cliente in base alle stime fatte dagli sviluppatori (argomento che verrà trattato nel paragrafo 11.2.7).

11.2.5.1 Trade-off Sliders (Compromessi)

Analizzando gli interessi degli stakeholders è emerso un'ulteriore funzionalità: il supervisore voleva avere la possibilità di simulare un possibile scenario in base alle matrici di competitività, cambiando gli indici di posizionamento e facendo le proprie ipotesi per quanto riguarda la gestione dei prezzi. Una simulazione di questo tipo però richiede molto tempo per le operazioni ed è nata la necessità di stabilire dei *compromessi*.

Così è stato deciso di utilizzare dei *Trade-off Sliders*, una tecnica molto efficace ed immediata che implica la scelta tra due alternative: la perdita di valore (importanza) di una costituisce però un aumento di valore dell'altra. È una sorta di presa di posizione per decidere quale opzione privilegiare.

È molto utile quando si affrontano problematiche riguardo a funzionalità che sono "incompatibili" tra di loro, cioè non si può realizzare il 100% dell'una e, allo stesso tempo, anche il 100% dell'altra. Per esempio, attraverso la simulazione è emerso il problema della velocità: è più importante il time-to-shelf, cioè il tempo per arrivare sullo scaffale con il nuovo prezzo, oppure l'accuratezza della simulazione stessa? Per osservare un esempio reale, vedere figura 36.

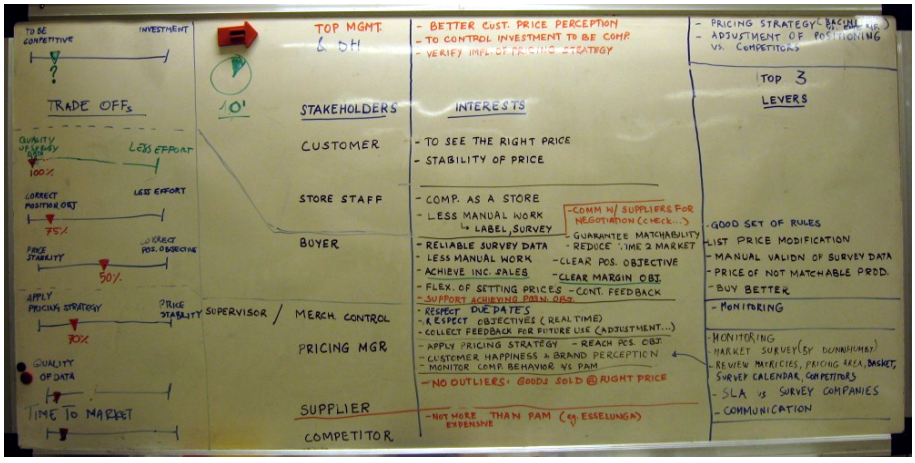

Figura 36: Un esempio di Trade-off slider nella parte sinistra della figura.

11.2.5.2 User Journey: percorso degli attori

Terminata la fase della visualizzazione condivisa del progetto e l'identificazione degli interessi dei vari stakeholders, è stato chiesto, a ciascun attore, come utilizzerebbe l'applicazione se dovesse sedersi su una scrivania davanti ad un computer, delineando come lo strumento dovrebbe supportarli durante lo svolgimento delle proprie attività.

Questa tecnica, effettuata ancora una volta attraverso interviste, produce le **user-journey** che riflettono i pensieri, le considerazioni e le esperienze che le persone attraversano nella loro *vita quotidiana* in un certo contesto. La loro creazione pone una forte enfasi sugli attori e si basa anche sulla creazione di scenari e dei flussi degli utenti. Servono per identificare effettivamente il contesto operativo nel quale agiscono gli attori, ossia le attività che svolgono, e di pensare anche alla realizzazione dello strumento, ragionando come deve avvenire questo.

Con le idee, cominciano ad emergere i tipi di contenuti e le funzionalità necessarie per soddisfare queste esigenze. Una user-journey, quindi, non è altro che il cammino passo-passo che un utente compie per raggiungere il proprio obiettivo, con un occhio alla progettazione e all'usabilità[16]. È una tecnica usata soprattutto nella progettazione di siti web.

Grazie a questo strumento, è stato possibile iniziare ad affrontare il design del software da sviluppare perché esso dà realmente un'idea delle necessità dell'utente ed in quale modo e in quale momento fornire le informazioni.

La decisione di utilizzare una user-journey è stata presa anche per dare delle priorità alle funzionalità messe a disposizione dell'utente. Questo sistema fornisce una visione chiara su quanto sia facile/difficile per l'utente tipico raggiungere il proprio obiettivo, permettendo di accelerare la progettazione del software perché si ha velocemente un'idea di come l'applicazione dovrebbe funzionare.

11.2.6 Individuazione delle User stories

Dopo aver distinto i diversi attori in gioco e aver fissato ed utilizzato le user-journey, si riescono a costruire, per ciascun attore, le **user stories**, strumenti che stanno alla base delle funzionalità e che formano un insieme che costituisce lo scope del rilascio del progetto.

Una user-story descrive delle funzionalità dalla prospettiva del cliente, mettendo l'accento su chi la desidera e sul come e perché quella funzionalità viene utilizzata

[16]L'Usabilità è definita come l'efficacia e la soddisfazione degli utenti con le quali raggiungono determinati obiettivi in determinati contesti.

[20], alzando sempre più il livello di dettaglio in modo che abbia senso per chi poi la andrà a realizzare (i programmatori).

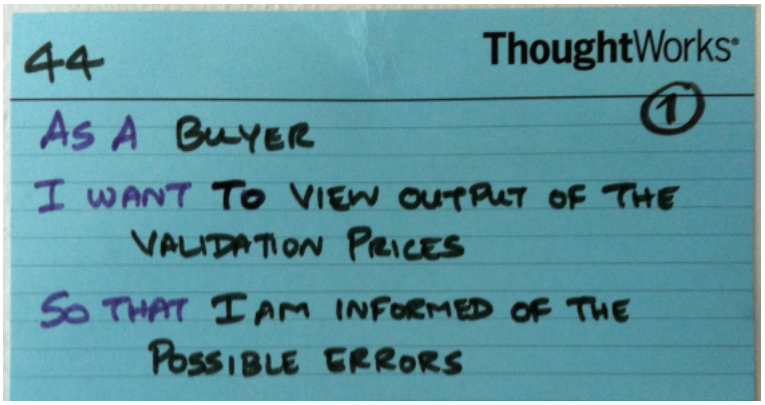

Figura 37: Un esempio di user story.

Le user stories (vedere figura 37) raffigurano delle azioni e, per mezzo di un linguaggio ad alto livello ed una struttura formale, aiutano la comunicazione tra cliente e programmatore. Esse sono scritte in inglese semplice, per essere più facilmente comprensibili, e sono necessariamente "piccole" perchè funzionalità piccole sono più facili da stimare e da sviluppare.

I componenti di base di una user-story sono definiti come le tre C:

- **Carta** - è la descrizione scritta della storia su una carta che serve come identificazione, promemoria ed aiuto nella pianificazione.
- **Conversazione** - è una *promessa di conversazione* (più approfondita) con il cliente riguardante la funzionalità scritta sulla "carta".
- **Conferma** - è il test d'accettazione che il cliente esegue per confermare che la funzionalità è stata completata (la condizione di soddisfacimento).

L'unione di tutte le storie forma i requisiti funzionali (scope) del progetto che sono legate necessariamente ad un beneficio che, a sua volta, non è altro che la risposta ad un bisogno.

Durante le varie iterazioni, i progetti agili sono guidati dalle storie, sulle quali viene posta continuamente l'enfasi, soprattutto su quelle espresse dal business. Esse possono essere modificate, aggiunte o rimosse proprio per soddisfare le esigenze degli attori principali ed è per questa possibilità di interscambio tra le schede che ad ognuna viene apposto un codice identificativo.

Per capire se le storie sono valide si può ricorrere all'acronimo "INVEST":

Independent (le storie devono essere indipendenti per essere lavorate facilmente e richiedono la comprensione).

Negotiable (le storie devono essere negoziabili, cioè non sono esplicitate rigidamente in un contratto ma possono essere decise e riposizionate nel piano di rilascio in base alla stima).

Valuable (le storie devono avere valore misurabile per il cliente).

Estimable (le storie devono essere stimabili, anche se non in maniera precisa, in modo da facilitare il cliente a porre le priorità).

Small (le storie devono essere necessariamente di piccole dimensioni in modo da poterle stimare e realizzare in poco tempo).

Testable (le storie devono essere verificabili per rafforzare la conoscenza di quella storia).

Un'ulteriore caratteristica delle user stories è quella della completezza. Una funzionalità descritta da una storia deve essere pienamente funzionante, attraversando tutti i layer dell'architettura[17] utilizzata dal progetto.

In generale, le storie riguardano delle vere e proprie funzionalità, però a volte, come è stato riscontrato durante le sessioni, ci sono dei requisiti non funzionali quali performance o design che non sono delle funzioni da sviluppare, piuttosto sono delle accortezze che i clienti desiderano. Le storie riguardanti queste attività sono state scritte su delle carte di colore diverso in modo da focalizzarle immediatamente sulla lavagna dove vengono attaccate.

Possiamo riassumere il termine scope come un insieme di user stories che facilita la visione gobale condivisa, cioè il consolidamento e la comprensione del sistema, e la sua realizzazione.

Non appena vengono definite le user stories, esse vanno stimate da chi andrà a realizzare il sistema.

[17] I Layer architetturali sono delle strutture logiche con dei compiti specifici e possono essere di molti tipi. Il Modello più utilizzato è quello a tre livelli: User Interface (interfaccia grafica che ha lo scopo di gestire l'interazione del sistema con il mondo esterno), Business Logic Layer (logica di business che indica l'insieme delle regole di business che regolano il funzionamento dell'applicazione) e Data Access Layer (accesso ai dati che si occupa di persistere le informazioni trattate dall'applicazione e conosce le modalità di lettura/scrittura nell'ambito di una sorgente dati).

11.2.6.1 Verso l'implementazione delle storie

I dettagli delle varie storie sono stati sviluppati utilizzando la tecnica denominata "*Given, When, Then*" che, usando un inglese molto semplice, esprime un possibile scenario in maniera chiara ed includendo le varie interazioni con il sistema.

Con questo sistema, nella fase di programmazione del progetto si entra nel dettaglio delle user-stories, arricchendole di dettagli (*narratives*) in modo da renderle comprensibili e quindi lavorabili da parte del programmatore. Per questo è stato utilizzato lo strumento Mingle®. Le componenti principali delle *narratives* sono:

- il *summary*, una descrizione sintetica del contesto in cui si inserisce la storia, utile a comprendere meglio il "perché" della storia stessa;
- i criteri di accettazione, che tipicamente secondo il formato:

```
GIVEN ...
WHEN ...
THEN ...
```

La clausola GIVEN descrive lo stato iniziale in cui si trova il sistema.

La clausola WHEN descrive come l'utente interagisce con il sistema.

La clausola THEN descrive l'esito atteso, ovvero la risposta che il sistema deve fornire nelle condizioni sopra descritte.

11.2.7 Stima delle storie

Successivamente, la parte IT si è riunita e ha iniziato a lavorare sulle user-stories raccolte. Lo scopo di questa fase di stima è quello di attribuire un'etichetta per differenziare le storie in base alla **dimensione**, eseguendo, inizialmente, una stima ad alto livello.

Le stime nelle metodologie agili non consentono di essere precisi ed accurati ma si focalizzano sulle cose davvero importanti cioè creare un piano di rilascio affidabile su cui il team ed il cliente possano lavorare.

Per definire questa etichetta sono state usate le misure classiche delle taglie delle t-shirt (S, M, L, XL) ed è stato utilizzato un metodo chiamato **poker estimation**: un "gioco" il cui obiettivo è quello che i partecipanti associno una taglia alla storia tutti allo stesso momento, in modo tale da evitare che si influenzino l'un l'altro. L'attribuzione della taglia è una valutazione di tipo comparativo: di volta in volta, si confrontano le storie prima della stessa taglia e poi di taglie diverse (triangolarizzazione) allo scopo di verificarne la consistenza.

Degli studi hanno dimostrato che, per gli umani, è più facile saper stimare qualcosa relativamente in base alla dimensione. Per esempio, possiamo facilmente confrontare due montagne tra di loro e dire, con accuratezza, quale delle due è più grande e quale più piccola. [1]

Durante questa pratica sono state eseguite le seguenti fasi:

- Lettura da parte del cliente della storia da stimare.
- Stima della dimensione da parte di ciascun membro del team di sviluppo[18].
- Discussione delle eventuali divergenze.
- Stima condivisa da tutti i membri del team della storia.

In questa fase la rapidità è fondamentale perché non è necessario entrare troppo in dettaglio, data anche la ancora scarsa conoscenza della funzionalità di quella specifica user-story. In queste condizioni una stima accurata da subito è quasi impossibile. Risulta molto più facile confrontare le storie tra di loro e dare una dimensione piuttosto che attribuirle un lasso di tempo per la realizzazione.

11.2.7.1 Story points

Al termine dell'attribuzione delle taglie sono stati fissati i **story-points** che sono dei valori associati alle taglie e che indicano la velocità con cui si può realizzare la funzionalità definita nella user-story. Quindi, su ogni storia sono stati riportati i valori ottenuti da questa considerazione per poi preparare il tavolo su cui si comporrà il piano di rilascio.

Per ottenere nel modo più semplice possibile la stima delle storie sono stati utilizzati dei numeri piccoli (nell'ordine delle poche unità). Dei numeri grandi consentirebbero una maggiore accuratezza, ma complicano inutilmente il processo di stima dato che comunque si tratta di stime non ancora affidabili.

Un sistema basato sugli story-points ricorda che le stime ad alto livello sono delle *supposizioni* (non possono quindi essere accurate), ci danno una misura semplice e veloce della dimensione delle storie (tralasciando, per ora, la durata in termini di tempo).

11.2.7.2 Velocità di sviluppo e calcolo del numero di Iterazioni

La parte IT si è poi riunita per quantificare il numero di story points che il team di sviluppo è in grado di realizzare in una iterazione (durata media due settimane), tenendo conto della velocità di sviluppo possibile e delle risorse dedicate al progetto.

[18] Ciascun componente del team (analisti, architetti, programmatori, ecc.) ha la possibilità di stimare le storie.

Per aumentare l'affidabilità della stima della velocità, sono stati decisi due valori: una grandezza minima, che indica una velocità più aggressiva/veloce, e una grandezza massima, che indica una velocità più realistica/lenta. Ci sono molte tecniche di calcolo per l'attribuzione della velocità di sviluppo, ma quella più affidabile resta comunque quella basata sull'esperienza.

Ottenuto il valore della velocità con la quale si è previsto di sviluppare il progetto e sommando i valori di tutti gli story points, è stato possibile calcolare il numero di iterazioni necessarie per completare le funzionalità incluse nello scope di questo rilascio facendo il rapporto tra il totale degli story-points e la velocità calcolata.

$$\#\text{ iterazioni} = \frac{\#\, totale\ story{-}points}{velocità\ team\ stimata}$$

11.2.8 Piano di rilascio

Terminata la stima, è stato chiesto alla parte business di prioritizzare le funzionalità per comporre il **piano di rilascio** nel quale si fissano le storie da realizzare nelle varie *iterazioni*, determinando così lo scope del progetto. La prioritizzazione impone, infatti, che vengano sviluppate prima le funzionalità più importanti (il 20% delle funzionalità che erogano l'80% del valore, secondo la legge di Pareto) ed, in seguito, quelle di valore minore. Ciascuna iterazione è così composta da un certo numero di storie (di grandezza non necessariamente uguale) che devono essere realizzate e che sono limitate dalla velocità fissata nella precedente stima. La rigidità sul totale degli story-points procede nel rispetto del limite fissato (punti per iterazione) sebbene sia anche necessario essere flessibili per quanto riguarda le funzionalità da scegliere.

Nella scelta delle storie la parte business è supportato da quella IT. Quest'ultima ha integrato il criterio del valore di business con altri due criteri:

- la sostenibilità: la somma dei punti delle storie selezionate non deve superare il limite determinato dalla velocità stimata (valutato attraverso l'utilizzo al grafico di Burn-up);
- la consistenza, ovvero il rispetto delle dipendeze "logiche" tra le storie stesse: ad esempio, la storia relativa alla modifica del prezzo di vendita non può essere implementata prima di quella inerente alla visualizzazione del prezzo stesso.

Il vantaggio di collaborare con il cliente in questa fase è di poter indirizzare la scelta iniziale verso quelle storie che effettivamente hanno un valore rilevante e che riducono i rischi che emergono dall'architettura del progetto.

La selezione delle storie da inserire nel piano di rilascio è stata eseguita sgomberando appositamente un tavolo (vedere figura 38), per poi posizionarci, in base

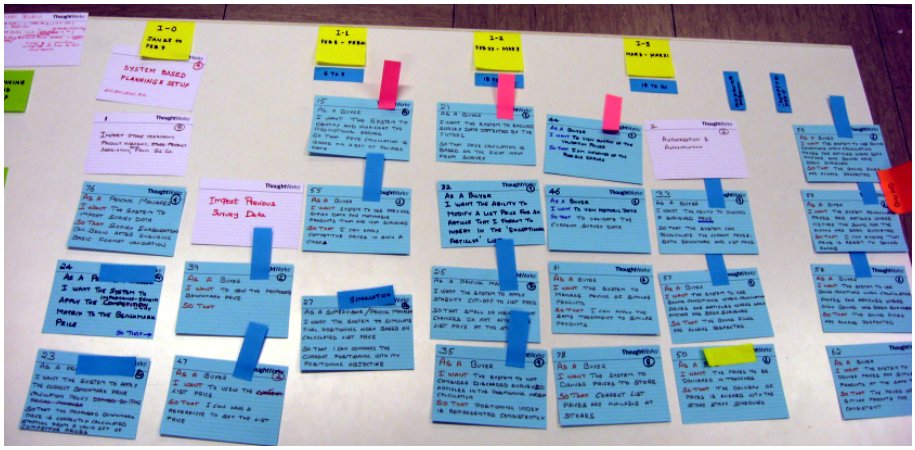

Figura 38: Scelta delle user-stories per formare il piano di rilascio.

alla suddivisione delle iterazioni, tutte le user stories raccolte nella fase precedente di individuazione. Storie stimate in modo opportuno e univocamente identificate.

Ultimato il piano di rilascio sono state fissate delle milestones, cioè delle date entro le quali ci si impegna a consegnare al Business una parte del software che possa erogare valore, pur non avendo un elenco prefissato di funzionalità da svolgere, in modo da rilasciare le prime funzionalità e ricevere un primo feedback.

Nel caso in cui il rilascio previsto per una delle date fissate dalle milestones non racchiude un insieme consistente di funzionalità è possibile rivedere l'elenco delle storie previste per l'iterazione di competenza e sostituirlo con delle storie dotate di più valore, prestando attenzione alla somma dei story-points (che devono comunque essere rispettati) quando si aggiungono o si sottraggono delle storie.

11.2.8.1 Burn Up

È stato realizzato un grafico di **Burn-Up** (e non di Burn Down, come suggerito invece da Scrum e trattato nel paragrafo 6.3.2) per calcolare la soglia di punti da raggiungere una volta fissata la milestone.

Questo grafico, in ogni caso, presenta le stesse caratteristiche del Burn Down però visto all'incontrario. Sono state poste sull'asse delle ascisse le varie iterazioni pianificate per il rilascio di competenza e sulle ordinate la somma degli story points delle storie da realizzare; l'inclinazione della retta è data dalla velocità di sviluppo del team. Si parte dal punto (0,0) e si arriva alla conclusione del rilascio di riferimento con l'ultima iterazione nel punto (N,N), cioè la somma degli story points totali delle varie funzionalità da realizzare.

Quando si osserva la corrispondenza sulla retta ideale (la velocità di sviluppo stimata) della milestone sull'asse delle ascisse, si determina il numero di punti da sviluppare per quella data che diventa il limite per le selezioni delle storie. Il nume-

ro ottenuto deve essere preso in considerazione quando viene eseguita la selezione delle storie perché l'insieme delle funzionalità sviluppate (racchiuso nel limite) deve comunque erogare valore.

Il grafico di Burn-up è, inoltre, un ottimo strumento che rende visibile in modo semplice ed immediato quanto lavoro è stato svolto, quanto ne manca, quale è la velocità del team di sviluppo avendo sempre ben presente la data di consegna (milestone) prevista per il rilascio delle funzionalità previste.

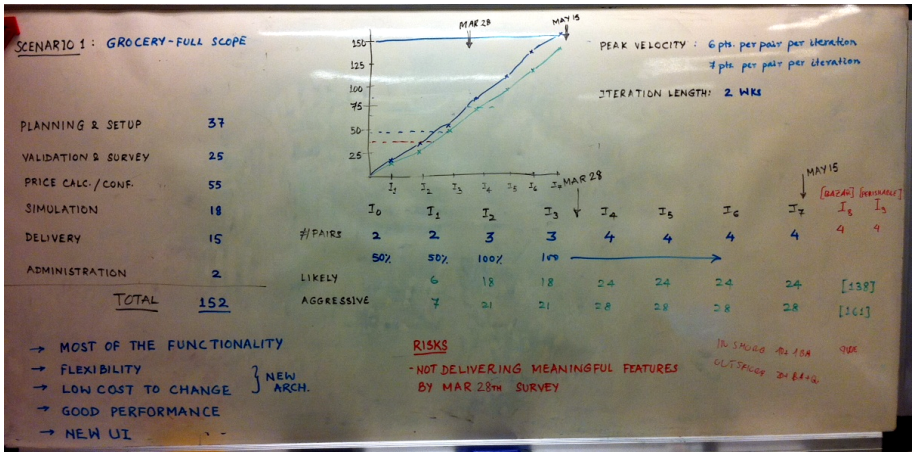

Figura 39: Un esempio di grafico Burn Up.

11.2.9 Showcase

Come ultima pratica dell'Inception è stato fatto, come di consueto, uno showcase tramite una presentazione su proiettore per riassumere tutte le attività svolte durante le sessioni dedicate ed evidenziare i risultati ottenuti. Si sono riprese le sotto-fasi riguardanti la comprensione del progetto (in termini di visione globale, ruoli ed obiettivi), l'analisi (con particolare attenzione ai rischi legati al progetto ed agli user-journey, ma anche ad una panoramica generale dello strumento di pricing) e al piano di rilascio (con riferimento alle storie, al modo di comunicazione tra business e parte IT e alle date di scadenza che sono state fissate).

Inoltre, sono state fissate le attività da affrontare nelle prossime fasi di sviluppo come, per esempio, l'inizio della prima iterazione di sviluppo e il coinvolgimento e collaborazione del business con il team di programmatori per l'approfondimento delle storie.

Parte V

Conclusioni

12 Conclusioni

Il caso di studio affrontato in questa relazione finale di tirocinio riguarda soltanto le due fasi iniziali del ciclo di vita di un progetto (Discovery e Inception): non è quindi possibile esprimere delle valutazioni definitive circa l'efficacia complessiva dell'applicazione delle metodologie agili. È possibile, però, dare riscontro di alcuni risultati significativi che la scelta agile ha prodotto, in particolare in termini di collaborazione tra parte IT e parte business, premessa fondamentale per il successo di ogni progetto.

Da un lato, la parte business ha molto gradito il cambio di approccio, ritenendolo un modo molto semplice ed efficace per mettere la parte IT in condizione di sviluppare degli strumenti software che realmente possano agevolare il proprio lavoro.

Dall'altro, la parte IT ha avuto la possibilità di comprendere più a fondo il dominio di business, di recepire più efficacemente i requisiti nonché di definire un piano di progetto sostenibile che al contempo preveda il rilascio - nel giro di poche settimane - di una prima versione dal contenuto significativo, in termini di valore erogato. Un risultato inedito prima d'ora: in precedenza, con il tradizionale approccio waterfall, i tempi medi per il primo rilascio - nel caso progetti simili per dimensione e complessità - erano dell'ordine di parecchi mesi.

Da questo punto di vista, si può possibile affermare che alcuni degli obiettivi del programma di trasformazione della struttura IT, sono stati quindi raggiunti.

Per quanto riguarda il giudizio finale sul tirocinio non posso che ritenermi più che soddisfatto per come è stata svolta l'attività di stage presso Larus Business Automation S.r.l.. Ho avuto modo di inserirmi in una realtà aziendale di successo ma comunque giovane e dinamica e di affrontare le problematiche che derivano dallo sviluppo di software, imparando a rapportarmi con i colleghi sentendomi parte integrante di una "squadra" di lavoro. Ho iniziato ad affrontare il lavoro anche sotto l'ottica del rispetto di scadenze che mi sono state fissate quindi con il vincolo di un certo lasso di tempo da rispettare assolutamente. Ho avuto quindi la possibilità di mettere in pratica molti dei concetti teorici che ho acquisito durante gli anni di studio.

Parte VI

Ringraziamenti

13 Ringraziamenti

Vorrei ringraziare, innanzitutto, il Prof. Michele Moro che, in qualità di relatore, mi ha seguito durante l'attività di tirocinio e durante la realizzazione di questa relazione.

Ringrazio l'azienda Larus Business Automation S.r.l. ed, in particolare, il Dott. Lorenzo Speranzoni per avermi dato la possibilità di svolgere lo stage presso le loro strutture.

Ringrazio il Dott. Alessandro Polo che, in qualità di tutor aziendale, ha diretto la mia attività di stage e mi ha supportato (con enorme pazienza) durante la realizzazione della relazione e ringrazio i colleghi che hanno contribuito alla mia formazione e con cui ho instaurato un ottimo rapporto personale.

Un ringraziamento speciale va ai miei genitori che mi hanno dato la possibilità di studiare e che mi sono stati sempre vicini nei momenti più difficili della mia vita.

Ringrazio la mia famiglia (in particolar modo mia sorella) e la mia ragazza Martina che mi hanno sempre sostenuto in ogni mia scelta.

Ringrazio i miei amici per i bellissimi momenti passati assieme e per avermi tirato su il morale quando ne avevo necessità.

Parte VII

Riferimenti bibliografici

Riferimenti bibliografici

[1] *The Agile Samurai: How Agile Masters Deliver Great Software*, Jonathan Rasmusson, The Pragmatic Bookshelf, 2010

[2] *Scrum e XP dalle Trincee - Come facciamo Scrum*, Henrik Kniberg, C4Media Inc., 2007

[3] *Scrum*, Ken Schwaber and Jeff Sutherland, 2010

[4] *Principi di Ingegneria del Software*, Roger S. Pressman, 2008

[5] *Extreme Programming e Metodologie Agili di Sviluppo Software: concetti, prodotti e risorse*, Massimiliano Bigatti, 2002

[6] *Un nuovo metodo di Sviluppo Software: eXtreme Programming (XP)*, César F. Acebal, Juan M. Cueva Lovelle, 2002

[7] *EXtreme Programming Explained: Embrace Change*, Kent Beck, Addison Wesley Longman, 2000

[8] www.agilealliance.org (sito ufficiale della Agile Alliance)

[9] www.extremeprogramming.org

[10] *Lean Principles and Practices*, Tom and Mary Poppendieck, 2006

[11] *Agile Overview*, Naresh Jain

[12] *Lean Software Development: An Agile Toolkit*, Mary & Tom Poppendieck, Addison-Wesley Professional, 2003

[13] *UML e ingegneria software: dalla teoria alla pratica*, Luca Vetti Tagliati, Hops Tecniche Nuove, 2003

[14] *Feature and Function Usage*, Jim Johnson, Standish Group International Inc., 2002

[15] *Toyota Production System: Beyond Large-Scale Production*, Taiichi Ohno, Productivity Press, 1988

[16] *Inside the Mind of Toyota: Management Principles for Enduring Growth*, Satoshi Hino, Productivity Press, 2006

[17] *Toyota Way 2001*, Toyota Motor Company, 2001

[18] *TechnicalDebt*, Martin Fowler, 2009

[19] *Time Boxing is an Effective Getting Things Done Strategy*, Dave Cheong, 2006

[20] *Essential XP: Card, Conversation, Confirmation*, Ron Jeffries, 2001

[21] *Principi di Ingegneria del Software - Cicli di vita e processi di sviluppo*, Cefriel, 2000

[22] *What Is the Rational Unified Process?*, Philippe Kruchten, 2001

[23] *Rights and Responsibilities*, Scott W. Ambler, 2010

[24] *The essence of Agile*, Henrik Kniberg, 2010

[25] *Lean Primer*, Craig Larman and Bas Vodde, 2009

[26] *Product-Development Practices That Work: How Internet Companies Build Software*, MIT Sloan Management Review, 2001

[27] *Kanban e Scrum – ottenere il massimo da entrambe*, Henrik Kniberg e Mattias Skarin, C4Media Inc., 2010

[28] *Value Stream Mapping - Finding the Constraint*, Mary Poppendieck, 2007

[29] *Writing Effective Use Cases*, Alistair Cockburn, Addison-Wesley Professional, 2000