ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

I MODELLI AGILI DI SVILUPPO: IL CASO SCRUM

Relatore

Prof. Paolo Ciancarini

Candidato

Giulia Baccolini

Anno Accademico 2013-2014

Sessione I

Alla mia famiglia.

INDICE

Introduzione **1**

1 Modelli di processo di sviluppo prima della nascita di agile **5**

1.1 Modello di processo 5

1.2 Introduzione al modello tradizionale 6

1.3 Il modello iterativo 8

1.3.1 Il modello a spirale 10

2 Modello iterativo Microsoft: Synch-and-Stabilize **15**

3 Nascita del modello agile e cambiamenti apportati da esso **21**

3.1 Introduzione al modello agile 21

3.2 Il Manifesto Agile: introduzione 22

3.2.1 Il Manifesto e i 12 principi 23

3.3 Il modello agile 25

3.4 Differenze tra modello iterativo e modello agile .. 31

4 La nascita di Scrum **35**

4.1 Com'è nato Scrum? 35

4.2 Le componenti 38

4.3 Il processo di sviluppo in Scrum 43

4.4 Difficoltà e punti di forza 46

5 Scrum alla Microsoft **49**

Conclusione **55**

Ringraziamenti **59**

Bibliografia 61

Sitografia 63

INTRODUZIONE

Il processo di sviluppo è un aspetto dell'ingegneria del software molto delicato, perché contiene diverse attività che devono coesistere ed essere tra loro compatibili per far si che il processo stesso termini con il risultato desiderato.

Il processo di sviluppo è quindi tanto proficuo quanto difficile da gestire, perciò necessita di una guida che possa illustrare come coordinarlo al meglio per raggiungere l'obiettivo fissato in tempi brevi e con budget minimo.

Questa guida è conosciuta come modello di processo ed è la base fondamentale su cui gli sviluppatori devono appoggiarsi per realizzare al meglio il loro progetto.

Il modello su cui si basa lo sviluppo deve riuscire a gestire al meglio il budget economico, il tempo a disposizione, la comunicazione e l'interazione tra gli sviluppatori, la piena comprensione dei requisiti e tutte le fasi che compongono il processo di sviluppo.

L'obiettivo del modello di processo è quindi quello di consentire agli sviluppatori di costruire il software richiesto incontrando meno ostacoli possibili.

Il primo modello nasce poco prima degli anni 70, insieme ai primi processi di sviluppo del software.

Nel corso degli anni, la tecnologia informatica è pian piano progredita, diventando sempre più ricca e corposa. Parallelamente si è sviluppato anche il modello di processo, che dovette iniziare a gestire processi via via sempre più complessi ed impegnativi.

Dagli anni 70 ad oggi sono intervenuti molti modelli di processo, ognuno migliore rispetto al precedente; il

cambiamento più significativo lo si identifica nel modello agile, nato nel 1999.

Il progresso del modello agile porta alla nascita di vari esempi di modelli agili, tra cui Scrum.

Scrum è uno dei migliori modelli agili esistenti al giorno d'oggi, ed è anche uno tra i più adottati.

Questo perché Scrum, rispetto ad altri modelli, gestisce al meglio gli aspetti e le componenti del processo di sviluppo, limitando notevolmente i problemi riscontrati dagli altri modelli agili e dai modelli precedenti.

In Scrum, il team di sviluppo opera come un'unica unità mitigando le problematiche di comunicazione e riuscendo a produrre software in tempi ridotti senza superare il budget previsto.

Grazie a studi statistici si può notare che Scrum è il modello più adottato sia in Italia che nel resto del mondo; uno studio effettuato alla Microsoft mostra come l'utilizzo di questo modello abbia condotto al miglioramento del processo di sviluppo, soprattutto in termini di qualità, produttività, stima e precisione.

Questo elaborato illustra come si sono sviluppati i modelli di processo, focalizzando l'attenzione sui modelli agili ed in particolare su Scrum, dimostrando come questo modello riesca ad eliminare il superfluo concentrandosi principalmente sulle fasi di sviluppo e sulla comunicazione, riuscendo a rendere il processo di sviluppo maggiormente proficuo.

Il primo capitolo spiega com'è nato il primo modello di processo e descrive i primi modelli usati, il modello iterativo e quello a spirale, con i relativi miglioramenti ed altrettanti difetti.

Il secondo capitolo si focalizza su un esempio di modello iterativo, il modello synchronize and stabilize, e su come esso venga seguito per un lungo periodo alla Microsoft.

Nel terzo capitolo viene introdotto e descritto il modello agile, sottolineando l'importanza del Manifesto Agile e dei 12 principi. Infine vengono esaminate le differenze sostanziali tra il modello iterativo e quello agile.

Il quarto capitolo parla di Scrum, descrivendone le componenti e le fasi del processo di sviluppo. Illustrando le difficoltà ma soprattutto i punti di forza viene mostrato che Scrum sia il migliore modello di processo attualmente usato.

Il capitolo finale è il quinto, che illustra sondaggi fatti sull'utilizzo di Scrum al giorno d'oggi e parla dell'esperienza positiva di tre team della Microsoft a stretto contatto con Scrum.

Capitolo 1

I MODELLI DI PROCESSO DI SVILUPPO PRIMA DELLA NASCITA DI AGILE

1.1 Modello di processo

La modellazione dei processi è uno dei capisaldi sui cui si basa il processo di sviluppo del software. Propriamente, un modello di processo software è una rappresentazione accurata di una famiglia di processi: ne presenta la descrizione di istanze viste da particolari prospettive, al fine di disporre di tutti gli strumenti per valutarle positivamente (o, talvolta, negativamente). [GJM04]

Un metodo (o modello) definisce in maniera precisa come le attività debbano essere svolte, quali metodi e strumenti sono impiegati e cosa deve essere prodotto. Durante il ciclo di vita, i vari team addetti alle fasi di realizzazione del software si basano su ciò che è descritto nel modello per riuscire ad ottimizzare soprattutto tempi e costi di rilascio del prodotto.

Gli obiettivi del modello di processo sono quindi determinare l'ordine degli stadi coinvolti nello sviluppo e nell'evoluzione del software e stabilire di criteri di transizione per progredire da uno stadio al successivo. [Boe88]

Metaforicamente, un modello è come un manuale di istruzioni: scandisce in modo preciso come affrontare ogni parte del ciclo di vita di un processo di sviluppo.

La modellazione porta benefici quali il miglioramento del processo (e quindi del prodotto finale) e del coordinamento del team di sviluppo, l'accumulo di esperienza ed infine la maggiore aderenza agli standard internazionali. Quando si definisce un modello però, è necessario tenere presente che il processo di sviluppo del software è caratterizzato da un

alto grado di instabilità: i requisiti cambiano in continuazione e, di conseguenza, i prodotti devono essere facilmente evolvibili. [GJM04]

In sintesi, il modello di processo software è una rappresentazione astratta di un processo da una particolare prospettiva, che stabilisce i principi di base su cui si fonda lo sviluppo del software. [Som07]

1.2 Introduzione al modello tradizionale

Originariamente, prima degli anni 70, i modelli di processo di sviluppo del software erano strumenti di natura esclusivamente concettuale: lo sviluppo del software era concepito banalmente come "programmazione per tentativi ed errori".

Il lavoro degli sviluppatori non aveva un punto di partenza: i team operavano senza avere un'idea dettagliata di cosa il software dovesse fare e di come dovesse essere implementato.

Questi erano modelli in cui il software si adattava progressivamente a ciò che il suo progettista desiderava, modelli senza schemi da seguire e regole su cui basarsi. L'unico pregio attribuibile a tali modelli era la capacità di raggiungere il risultato finale in tempi brevi (poiché mancava totalmente la parte di progettazione). In contrapposizione, però, c'era la necessità di un grande lavoro di manutenzione, spesso molto costoso. [GJM04]

Il modello usato in quei giorni era chiamato "code and fix" (produci e aggiusta): questo termine veniva usato per indicare un processo di sviluppo che non è né formulato precisamente, né controllato attentamente.

Non era un vero e proprio modello, tuttavia era molto usato per lo sviluppo dei software.

Questo modello descriveva il processo di sviluppo del software in due fasi:

1. Code: scrittura del codice
2. Fix: aggiustare il codice per rimuovere eventuali errori, migliorarne la funzionalità o aggiungere nuove caratteristiche

Gli sviluppatori iniziavano a lavorare basandosi su un insieme di requisiti molto ridotto: scrivevano il codice finchè c'erano tempo e denaro. Il rilascio del prodotto "dipendeva da un miracolo".

Il modello code and fix non prevedeva alcuna linea guida che indirizzasse il processo di sviluppo verso l'obiettivo fissato. Gli sviluppatori lavoravano senza avere un'idea ben precisa su cosa il prodotto finale dovesse fare e su come le funzionalità richieste dovessero essere implementate.

Pianificare il progetto e definire design e requisiti in anticipo sembrava, agli sviluppatori, una perdita di tempo. Per questo motivo iniziavano direttamente con la stesura del codice.

Con un po' di fortuna il prodotto riusciva ad essere rilasciato; altrimenti, il budget monetario veniva superato così come il tempo a disposizione, e il progetto entrava nell'elenco dei progetti cancellati.

Anche se questo modello prevedeva una stesura del codice molto veloce, era sicuramente impossibile che il risultato finale soddisfacesse i desideri e i bisogni dei clienti. [MoeLen04]

Molte furono le difficoltà riscontrate da questo modello. In particolare, dopo una serie di cambiamenti, la struttura del codice diventava disorganizzata, rendendo difficile apportare nuove modifiche.

Il fallimento del modello code and fix portò al riconoscimento della "software crisis".

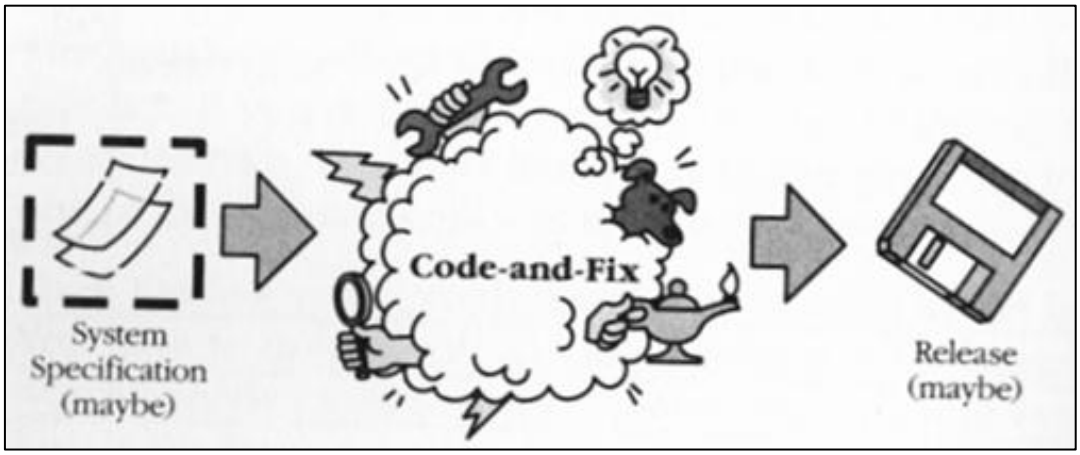

Figura 1.1 Modello Code and Fix [Web1]

Negli anni 70, però, a seguito della crisi, iniziarono a svilupparsi nuovi modelli più schematici. Il budget, gli standard di qualità e il tempo di sviluppo relativi ad un processo diventarono uno degli aspetti più importanti che i nuovi modelli dovevano trattare.

I clienti, come le aziende oppure i privati, iniziarono a commissionare lo sviluppo di software con precise funzionalità.

Come conseguenza di questa nuova esigenza iniziarono a svilupparsi nuovi modelli di processo software, tra i quali il modello iterativo (o incrementale).

1.3 Il modello iterativo

Il modello iterativo è il primo modello che permette la ripetizione delle fasi del ciclo di vita di un processo, ossia il ciclo può essere ripetuto più di una volta (il numero di iterazioni non è previsto a priori). Le

iterazioni terminano quando il prodotto diviene soddisfacente rispetto ai requisiti richiesti dal cliente. Ogni iterazione ha circa la stessa durata della precedente ed è volta all'incremento delle funzionalità del prodotto e all'eventuale miglioramento di qualsiasi funzionalità già esistente. Ad ogni istante dopo il primo rilascio (al termine della prima iterazione), esiste una versione k del prodotto in esercizio (chiamata prototipo) e una versione k+1 in fase di sviluppo. Il prototipo è un modello approssimato del prodotto finale che si vuole realizzare, le cui funzionalità sono definite in modo parziale. [Lar04]

Le varie iterazioni consentono anche di soddisfare l'importante e frequente necessità di adattarsi a cambiamenti come, ad esempio, la modifica o l' introduzione di nuovi requisiti da parte del cliente, l'evoluzione del problema di base, le diverse tecnologie da utilizzare, il cambiamento del mercato ecc. [Som07]

La capacità di adattamento al cambiamento è uno dei punti fondamentali che qualsiasi modello di processo di sviluppo software dovrebbe soddisfare: il mercato è in continuo mutamento e i bisogni dei clienti possono cambiare da un momento all'altro, quindi il team deve essere in grado di ovviare tutti i problemi relativi a questi cambiamenti.

Un altro punto importante di questo modello è la segmentazione dello sforzo nelle varie fasi, che permette la notevole riduzione dello sforzo stesso. Questa suddivisione dà luogo ad un continuo scambio di feedback tra le varie fasi durante le iterazioni: uno strumento di comunicazione molto importante che permette di ridurre gli errori (e lo sforzo richiesto per correggerli) dovuti alla scarsa o totale mancanza di comunicazione tra i vari team. [Lar04]

Grazie a queste caratteristiche, cresce la capacità di gestione dei rischi, poiché essi vengono valutati continuamente ed esplicitamente all'interno di ogni

iterazione. Questa capacità denota un grande miglioramento rispetto ai modelli tradizionali precedenti.

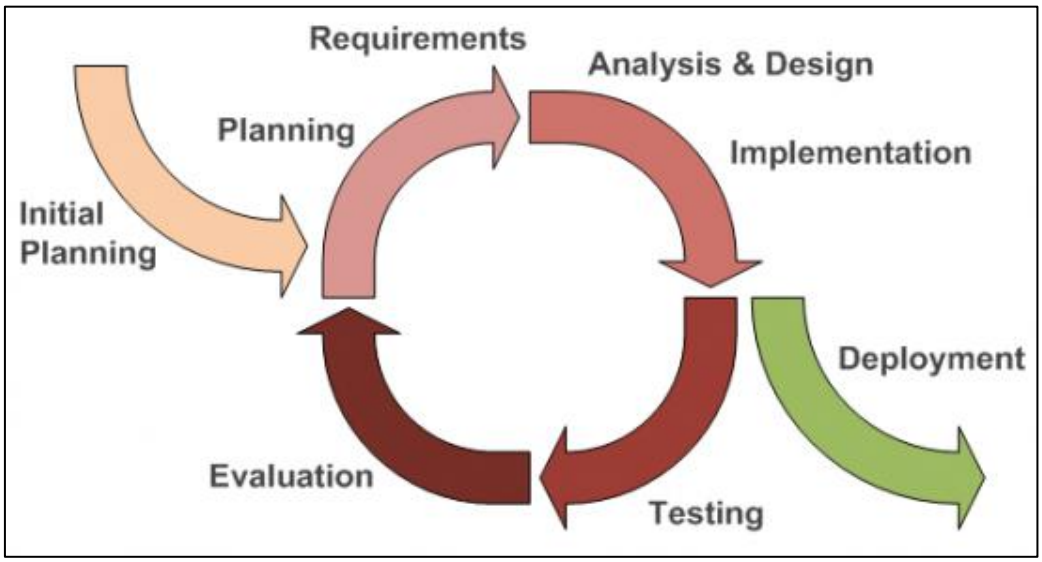

Figura 1.2 Ciclo del Modello Iterativo [Web2]

Il modello iterativo viene usato come base su cui appoggiarsi quando si vuole realizzare un progetto di grandi dimensioni, mediamente complesso e, soprattutto, con requisiti non del tutto stabili. Solitamente, nei progetti che durano circa 18 mesi servono in media da 3 a 6 iterazioni. [GJM04]

Il modello iterativo è rappresentato da un metamodello, chiamato modello a spirale. [Boe88]

1.3.1 Il modello a spirale

Il modello a spirale è stato proposto da Barry W. Boehm a maggio del 1988 ed è il modello che rappresenta l'intera classe dei modelli iterativi. [Lar04]

Questo modello si basa sul concetto del “rischio”, ovvero un insieme di circostanze avverse che possono pregiudicare il processo di sviluppo e la qualità del software.

Si concentra principalmente sull’identificazione e l’eliminazione dei problemi ad alto rischio tralasciando quelli banali, infatti nasce proprio a seguito dell’idea di aumentare e migliorare il controllo relativo a tutti i rischi possibili associabili al progetto.

Le caratteristiche principali del modello sono quelle di essere ciclico e non lineare (ma pianificato), flessibile, con alta valutazione del rischio (valutazione continua ed esplicita); può supportare diversi modelli iterativi e richiede il coinvolgimento del cliente. Il processo viene quindi modellato da una spirale (e non da una sequenza di attività): ogni spira della spirale è una fase del processo di sviluppo. [GJM04]

Ogni ciclo di spirale si compone di quattro fasi: pianificazione, analisi dei rischi, ingegnerizzazione e validazione (valutazione del cliente).

La prima fase, quella della pianificazione, identifica gli obiettivi e le alternative, analizza e registra tutti i dettagli relativi ai requisiti espressi dal cliente.

Alternative, obiettivi e requisiti vengono valutati nella seconda fase, quella dell’analisi dei rischi, in cui vengono evidenziate anche le potenziali aree di rischio (come, ad esempio, il budget non realistico, il personale inadeguato, lo scheduling sbagliato ecc).

La seconda fase e la terza fase sono separate da uno step molto importante, caratteristica unica del modello a spirale: la linea Go/No-go.

La linea Go/No-go è uno step che si incontra in ogni ciclo di spirale, nel quale si decide se proseguire nella realizzazione del progetto oppure fermarsi (ad esempio per rischi troppo elevati). In questo step viene effettuato un pass/fail test: il test è superato, quindi si può decidere

di proseguire con la costruzione del progetto, se la condizione di Go viene incontrata e la condizione di No-go fallisce. Questa linea è molto importante perché aiuta i team di sviluppo a valutare al meglio tutti i possibili rischi futuri. Superata lo step Go/No-go, si entra nella terza fase della spirale.

La terza fase, denominata ingegnerizzazione, consiste nello sviluppo e nella verifica del prodotto. Lo sviluppo del software avviene basandosi sugli obiettivi, sui requisiti e sulle eventuali alternative registrate nella prima fase (pianificazione) e valutati poi insieme ai rischi nella seconda fase (analisi dei rischi).

Terminata la fase di ingegnerizzazione, si conclude la spirale con la quarta ed ultima fase, quella della validazione (valutazione del cliente). Questa fase consiste nella revisione dei risultati ottenuti nelle tre fasi precedenti, soprattutto quelli della terza fase. La fase di validazione è importante perché permette al cliente di valutare in prima persona se i requisiti da lui espressi siano stati pienamente soddisfatti, soddisfatti in parte o non soddisfatti; inoltre, consente ai team di sviluppo di ricevere un prezioso feedback da parte del cliente, su cui baseranno gran parte della pianificazione (prima fase) del prossimo ciclo di spirale. [Boe88]

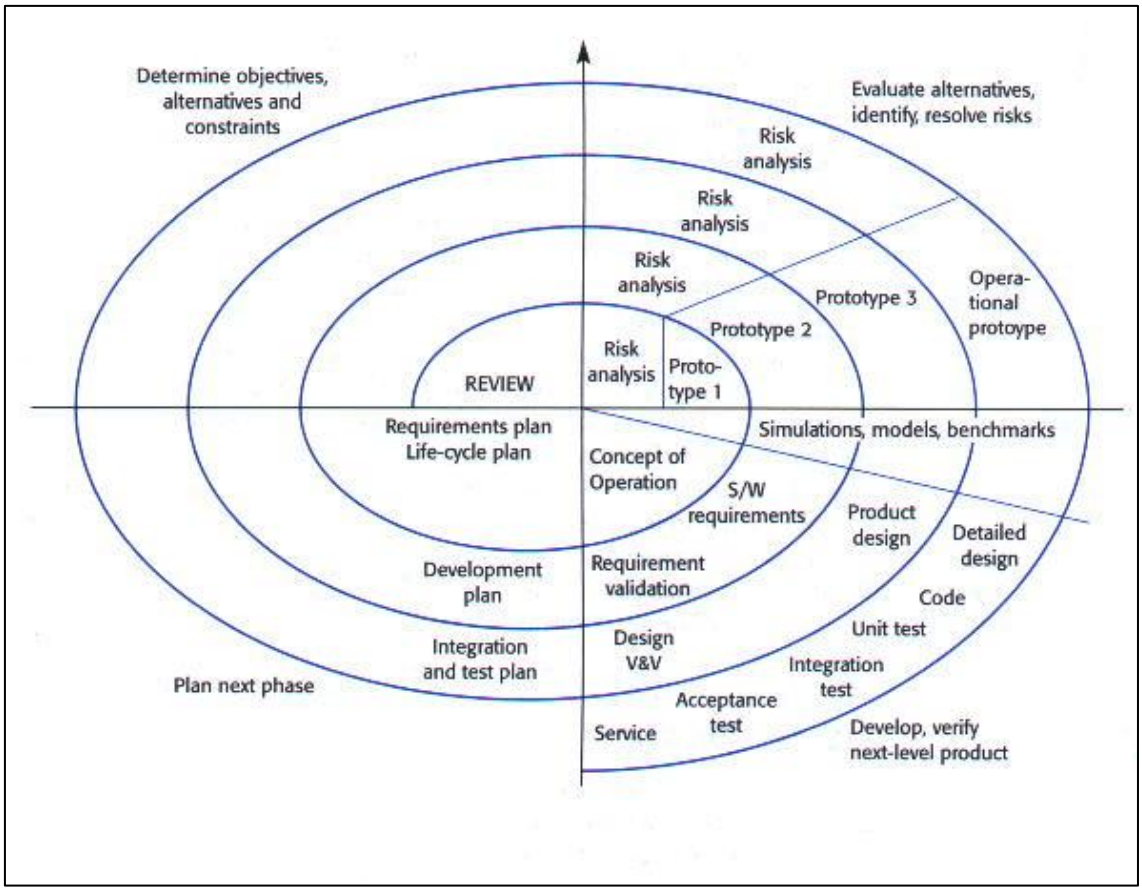

Figura 1.2 Modello a spirale [Web3]

Nel modello a spirale, il software viene sviluppato in una sequenza di versioni crescenti mediante la realizzazione di molteplici prototipi del prodotto, che caratterizzano ogni ciclo della spirale.

Il modello a spirale è consigliato per progetti di grandi dimensioni poiché permette di raffinare ad ogni iterazione il materiale precedentemente elaborato ed approvato dal cliente; all'inizio probabilmente sarà sola documentazione cartacea contenente gli intenti e lo scopo finale, poi si costruiranno, se ritenuti opportuni, dei prototipi da verificare e validare, e così via fino a giungere al prodotto conclusivo. [Boe88]

Esistono molti altri modelli iterativi di processo di sviluppo software, tra cui il modello Synchronize and Stabilize (Synch-and-Stabilize). [Cus97]

Capitolo 2

MODELLO ITERATIVO MICROSOFT: SYNCH-AND-STABILIZE

A partire dal 1980, la Microsoft ha riorganizzato il modo di progettare e realizzare software, in risposta alle problematiche riguardanti la qualità ma soprattutto a quelle legate alla tempistica.

I prodotti software richiesti sul mercato diventavano di realizzazione sempre più complessa ed impegnativa; quelli più ottimali registravano milioni di righe di codice: diventava quindi necessario assemblare team di produzione e testaggio di almeno 100 persone.

Nel 1996, la Microsoft registrava 20500 impiegati, 250 prodotti e 8,7$ milioni come reddito annuo.

L'obiettivo di questa riorganizzazione, nonché la futura filosofia di sviluppo adottata dalla Microsoft, era quello di far lavorare piccoli team o singoli programmatori in modo parallelo, in modo da creare un unico grande team. Questo al fine di produrre con maggiore velocità prodotti complessi, senza però impedire ai singoli individui ed ai piccoli team di sviluppare le proprie idee facendoli lavorare quasi autonomamente.

Ogni team ed ogni individuo dovrà sempre sincronizzare frequentemente tutte le modifiche che apporta al prodotto in costruzione, affinché ogni sua componente riesca a funzionare assieme a tutte le altre ed ogni team sia a conoscenza delle nuove modifiche apportate. [Cus97]

Sulla scia di questa riorganizzazione nasce, nel 1996, una nuova metodologia di sviluppo, ideata da David Yoffie e Michael Cusumano: il modello Synchronize and Stabilize (Synch-and-Stabilize). [GJM04]

Questo nuovo modello era talmente diverso dai precedenti che, non appena fu adottato, iniziò a creare subito diverse

difficoltà, nonostante fosse un modello funzionale e "vincente".

I team dovevano creare delle componenti che potessero essere sviluppate singolarmente, quindi delle componenti indipendenti dalle altre. Il problema risiedeva appunto nella difficoltà di creare fin dall'inizio del processo di sviluppo delle componenti che fossero, in futuro, sicuramente compatibili con tutte le altre. La Microsoft risolse il problema introducendo un meccanismo che permetteva agli sviluppatori di testare il prodotto insieme al cliente, in modo da creare le giuste componenti ed apportare le migliori modifiche in base all'esito del test, risparmiando tempo e denaro.

Un altro problema, forse il più frequente, era la difficoltà nel capire le necessità del cliente, che cambiavano di continuo. Era quindi azzardato definire totalmente un prodotto nelle fasi iniziali del processo di sviluppo. La Microsoft cercò di ovviare questo problema introducendo un approccio nel quale le specifiche iniziali venivano "congelate", veniva delineato il design, costruite le componenti e poi unite durante l'ultima fase, quella di integrazione e testing. Nel tempo, questo approccio non risultò molto funzionale, perciò venne pian piano abbandonato: la Microsoft capì che poter cambiare le specifiche "in corso d'opera", ottenere un feedback dal cliente e testare continuamente le componenti permetteva una produzione decisamente più ottimale. Le frequenti modifiche e integrazioni al prodotto in via di sviluppo permettevano anche di determinare quali parti funzionavano e quali no, senza dover aspettare la terminazione del prodotto. [Cus97]

L'approccio che venne successivamente adottato per definire un prodotto ed il suo processo di sviluppo si basava su cinque principi cardine:

1. Dividere grandi progetti in più sottoprogetti (3 o 4)
2. Usare la vision per instradare il progetto verso gli obiettivi prefissati
3. Basare la definizione delle caratteristiche del prodotto e delle relative priorità sui bisogni del cliente
4. Creare un design architetturale che rispecchi la struttura del prodotto e del progetto
5. Sfruttare al meglio i frequenti controlli dei committenti

La strategia di produzione della Microsoft si basava quindi su questi principi.

I team iniziano il processo di sviluppo creando una vision del prodotto, che definiva tutti gli obiettivi finali in base alle priorità del cliente.

Basandosi su questa vision, i manager definiscono le

caratteristiche del prodotto nel dettaglio, consentendo la suddivisione del lavoro tra i vari membri dei team, dando priorità allo sviluppo delle caratteristiche di base del prodotto.

Successivamente, il progetto viene sudddiviso in 3 o 4 grandi sottoprogetti (chiami anche "milestone"), assegnati di volta in volta ai team. Ogni team lavora parallelamente agli altri e sviluppa la parte del sottoprogetto che gli è stata assegnata: al completamento di un sottoprogetto, gli sviluppatori iniziano a definire tutti gli errori presenti in esso. Il momento della correzione dei bug è molto importante, perché stabilizza il progetto e permette ai team di capire quale parte del progetto si può considerare terminata e quale ancora no. [Cus97]

Terminata la fase di correzione ed eliminazione degli errori, i team iniziano a lavorare al sottoprogetto successivo.

Come la strategia di produzione, anche quella di gestione del prodotto si basa su alcuni principi cardine:

1. Lavorare parallelamente ma effettuare sincronizzazioni giornaliere
2. Avere sempre un prodotto pronto con versione adatta alle richieste di mercato
3. Usare lo stesso linguaggio in ogni settore di sviluppo
4. Effettuare continui testaggi del prodotto
5. Utilizzare anche dati statistici per determinare l'effettivo completamento del sottoprogetto

Il fulcro del metodo di sviluppo Synch-and-Stabilize era la concessione ai piccoli team e ai singoli individui della libertà di lavorare parallelamente tra loro dividendosi la mole di lavoro in modo da funzionare come un unico grande team (ma con maggiore efficienza) e, quindi, riuscire a costruire prodotti in larga scala molto più velocemente e ad un costo minore.

I team però devono rispettare tutti i principi cardine citati in precedenza, al fine di rafforzare comunicazione e coordinazione.

La sincronizzazione continua delle modifiche e dei miglioramenti al prodotto che ne deriva e la stabilizzazione periodica del valore del prodotto in termini di ricavo permettono di creare software che soddisfano a pieno e con tempestività le richieste di mercato. [GJM04]

La Microsoft ha tratto molti benefici dal modello Synch-and-Stabilize:

- continuo sviluppo del progetto anche nel caso in cui non si possa stabilirne il completo design e le precise caratteristiche fin dagli inizi
- semplificazione del lavoro dividendo il progetto in sottoparti minori; il lavoro viene svolto parallelamente con frequenti sincronizzazioni
- facilitazione della realizzazione del prodotto finale grazie all'introduzione di meccanismi di priorità: vengono completate prima le parti più importanti; le altre parti vengono sviluppate in seguito, oppure possono essere modificate o eliminate
- riuscire trarre il giusto beneficio dalle opinioni del cliente
- capacità di rispondere tempestivamente ai cambiamenti e alle

  richieste del mercato: c'è sempre un prodotto pronto per essere rilasciato in qualsiasi momento

La Microsoft ha prodotto diversi software basandosi sul modello Synch-and-Stabilize, tra cui Excel, Publisher, Word, Office, Windows 98, Windows NT.

La Microsoft, però, continuava a vedere come una grande sfida quella di evolversi migliorando il modello Synch-and-Stabilize per creare prodotti sempre più affidabili, sufficientemente potenti ma allo stesso tempo semplici da usare in modo da poter soddisfare anche i bisogni del cliente più inesperto.

Tuttavia, senza l'approccio Synch-and-Stabilize, la Microsoft non sarebbe stata in grado di disegnare, costruire e vendere i tanti prodotti che ha offerto sul mercato fino ad ora. [Cus97]

Capitolo 3

NASCITA DEL MODELLO AGILE E CAMBIAMENTI APPORTATI DA ESSO

3.1 Introduzione al modello agile

I modelli di sviluppo tradizionali, erano di natura lineare, metodica, strutturata e rigida: venivano infatti denominati modelli "peso massimo".

Nonostante questi modelli di processo di sviluppo software sembravano riuscire a soddisfare tutte le esigenze di chi li adottava, gli sviluppatori iniziarono a risentire di alcune problematiche: budget sempre più ridotto, pianificazioni spesso sbagliate, prodotti difettosi ecc.

A causa anche del continuo cambiamento della tecnologia e del business, creare prodotti sempre all'avanguardia diventava sempre più una grossa sfida.

Iniziò a diffondersi la necessità di evolversi e di costruire un nuovo modello maggiormente efficace. Il mercato iniziava a cambiare e i consumatori commissionavano prodotti sempre più complessi e di grandi dimensioni.

Nasceva quindi il bisogno di un modello capace di interpretare al meglio tutti i bisogni del cliente coinvolgendolo il più possibile all'interno dello sviluppo software in modo da ottenere un'immediata reazione alle sue richieste. Inoltre, cresceva la necessità di produrre in tempi sempre più ridotti risultati che riuscissero a migliorare la competitività sul mercato, anche e soprattutto in presenza di budget decrescenti. Era necessario quindi disporre di un nuovo modello capace di gestire al meglio le esigenze e i cambiamenti rapidi senza dover rinunciare alla qualità del risultato. [Som07]

Gli studi statistici sulla riuscita del progetti basati sui modelli tradizionali evidenziavano scarse percentuali:

- Il 16.2% dei progetti venivano completati correttamente, con le giuste caratteristiche, nei tempi prestabiliti e usando esattamente il budget a disposizione
- Il 52.7% dei progetti venivano completati senza aver soddisfatto tutti i requisiti, fuori budget consentito e oltre il tempo massimo
- Il restante 31.1% dei progetti veniva annullato

Per superare queste carenze, diversi sviluppatori professionisti introdussero un nuovo modello di processo di sviluppo software, il modello agile. [Cho08]

3.2 Il Manifesto Agile: introduzione

La formalizzazione e attuazione dei principi su cui si basa questo nuovo modello (che viene introdotto nel 1999) è stata oggetto del lavoro di un gruppo di progettisti software e guru dell'informatica che si sono spontaneamente riuniti nell'Agile Alliance. Il risultato di questo duro e lungo lavoro viene pubblicato nel febbraio del 2001, con il nome di Manifesto Agile, in cui, per la prima volta, appare ufficialmente il termine “agile”. [Mar09]

I firmatari del Manifesto Agile sono Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas. [BBBCCFGHHJKMMMSST01]

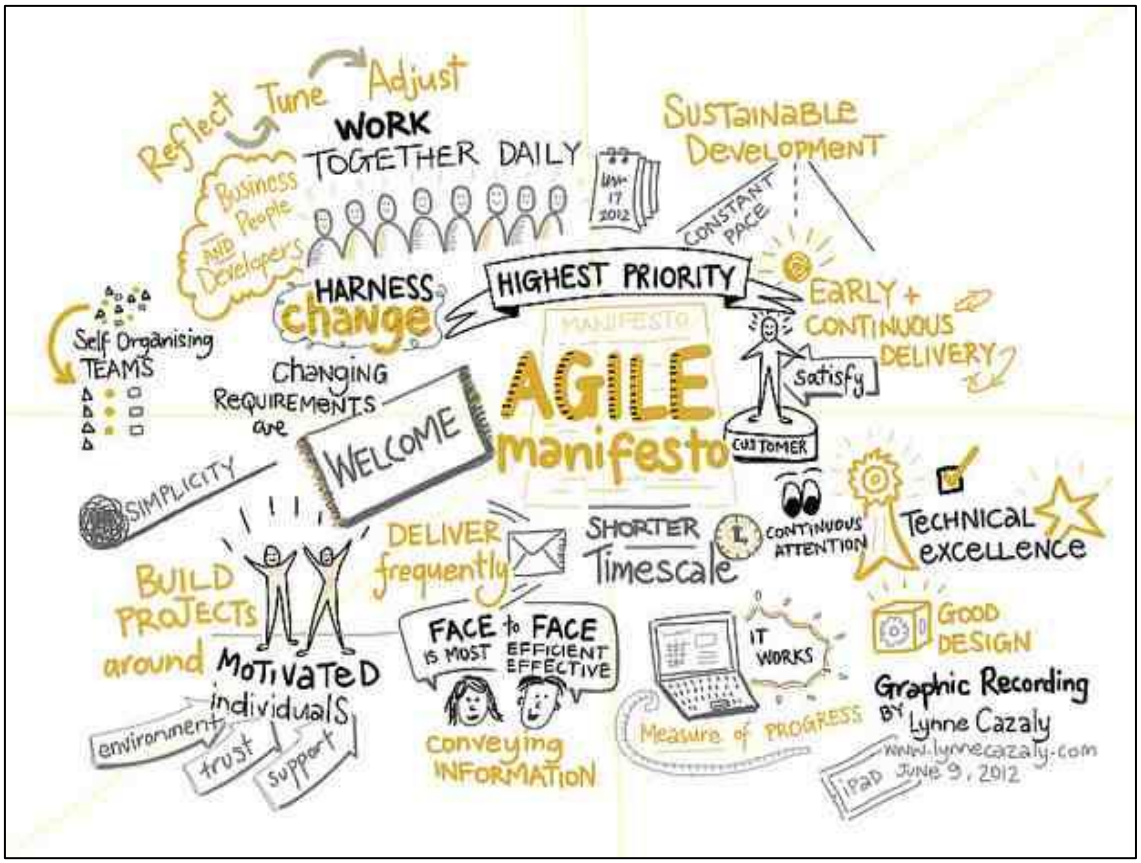

Figura 3.1 I principi del Manifesto Agile [Web4]

L'obiettivo del Manifesto Agile è fornire i concetti da seguire necessari ad una futura piena soddisfazione dei bisogni espressi dal cliente, a prescindere dalla terminazione del prodotto richiesto, cercando di ridurre il più possibile i costi di sviluppo e le tempistiche di realizzazione.

3.2.1 il Manifesto e i 12 principi

Il Manifesto Agile cita:

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. [BBBCCFGHHJKMMMSST01]

Sottostanti al Manifesto Agile, ci sono 12 importanti principi:

1. La nostra massima priorità è soddisfare il cliente rilasciando software di valore, fin da subito e in maniera continua.

2. Accogliamo i cambiamenti nei requisiti, anche a stadi avanzati dello sviluppo. I processi agili sfruttano il cambiamento
   a favore del vantaggio competitivo del cliente.

3. Consegnamo frequentemente software funzionante, con cadenza variabile da un paio di settimane a un paio di mesi,
   preferendo i periodi brevi.

4. Committenti e sviluppatori devono lavorare insieme quotidianamente per tutta la durata del progetto.

5. 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.

6. Una conversazione faccia a faccia è il modo più efficiente e più efficace per comunicare con il team ed all'interno del team.

7. Il software funzionante è il principale metro di misura di progresso.

8. I processi agili promuovono uno sviluppo sostenibile. Gli sponsor, gli sviluppatori e gli utenti dovrebbero

essere in grado di mantenere indefinitamente un ritmo costante.

9. La continua attenzione all'eccellenza tecnica e alla buona progettazione esaltano l'agilità.
10. La semplicità - l'arte di massimizzare la quantità di lavoro non svolto - è essenziale.
11. Le architetture, i requisiti e la progettazione migliori emergono da team che si auto-organizzano.
12. A intervalli regolari il team riflette su come diventare più efficace, dopodiché regola e adatta il proprio comportamento di conseguenza.

[BBBCCFGHHJKMMMSST01]

3.3 Il modello agile

Il metodo di sviluppo agile è profondamente radicato nel Manifesto Agile e nei 12 principi a suo seguito.

La data ufficiale di nascita della metodologia agile può essere identificata con quella della conferenza OOPSLA (Object-Oriented Programming, Systems, Languages and Applications), svoltasi a Denver dal 1 al 5 novembre 1999.

Inizialmente, i modelli agili avevano il nome di “lightweight” (leggeri) ma, con la pubblicazione del Manifesto Agile nel febbraio del 2001, viene introdotto per la prima volta il termine “agile”. [Mar09]

Il modello agile (in breve MA) evidenzia due importanti concetti: il primo è la minimizzazione del rischio definendo chiaramente i risultati in ogni breve iterazione; il secondo riguarda la frequente comunicazione diretta con il cliente, che permette di creare un’ampia documentazione. La ragione per la quale questi due concetti vengano enfatizzati è che entrambi aiutano i team ad adattarsi velocemente ai rapidi ed imprevedibili cambiamenti dei

requisiti e del mercato che vengono effettuati nella maggior parte dei progetti.

Le MA sono quindi un metodo per sviluppare software che cerca di gestire in modo esplicito gli inevitabili cambiamenti nei requisiti, anche a sviluppo inoltrato, senza impiegare molte energie. Cerca di massimizzare la comunicazione (preferendo un dialogo faccia a faccia piuttosto che una corposa documentazione), di valorizzare al massimo le risorse umane, di produrre software funzionante durante tutto il processo di sviluppo e di utilizzare di continuo il feedback del cliente sul progetto. [Boe02]

La maggior parte dei metodi agili tenta di ridurre il rischio di fallimento sviluppando il software in iterazioni di tempo limitato (solitamente di qualche settimana). Ogni iterazione è un mini progetto a sé stante e deve contenere tutto ciò che è necessario per rilasciare un piccolo incremento nelle funzionalità del software. Anche se il risultato di ogni singola iterazione non ha sufficienti funzionalità da essere considerato completo deve essere rilasciato e, nel susseguirsi delle iterazioni, deve avvicinarsi sempre di più alle richieste del cliente. [BoeTur03]

A livello di gestione del progetto, le MA seguono quindi questi principi:

- Rilasci frequenti del prodotto sviluppato
- Collaborazione continua del team di progetto con il cliente
- Documentazione di sviluppo precisa ma ridotta
- Valutazione sistematica e continua di valori e rischi dei cambiamenti

- Progettazione semplice: “You Aren’t Gonna Need It” (YAGNI, non ne hai bisogno)

[CocWil03]

Inizialmente vengono scomposti i requisiti definiti dal cliente in requisiti atomici (chiamati “user story”). La user story è una frase nel linguaggio comune, che indica una funzionalità che l’utente vuole sia sviluppata nel progetto da lui richiesto o, più semplicemente, è una breve descrizione di qualcosa di cui il cliente necessita (ad esempio: “il pass per il parcheggio può essere pagato online tramite carta di credito”). [Mar09]

Ognuna di esse ha una priorità definita dal cliente: quelle con priorità più alta devono essere ben definite, mentre quelle con priorità minore possono essere definite meglio in seguito. La descrizione può essere arricchita da altre user story durante lo sviluppo del software.

Ogni user story viene poi schematizzata e, in base alle priorità espresse e definite dal cliente e ai rischi e risorse necessarie valutate dagli sviluppatori, viene inserita in un elenco, pronta per essere analizzata. Le user story di più alto rischio e priorità vengono affrontate e sviluppate per prime e l’elenco delle restanti viene di conseguenza modificato.

In seguito alla definizione delle prime user story, si entra nella fase del Test-Driven Development (TDD).

In questa fase, gli sviluppatori scelgono una user story e scrivono i relativi test e, prima di iniziare a scrivere il codice, questi test devono girare correttamente. Inoltre, gli sviluppatori realizzano, insieme al cliente, un test chiamato “test di accettazione”, che consente di misurare in ogni momento il livello di progresso del progetto.

Il Test-Driven Development assicura quindi che il software sia dotato di un insieme di test automatici, in grado di proteggere il progetto stesso dagli effetti collaterali dei cambiamenti e di conoscere il livello di avanzamento in ogni momento. [Hig03]

Lo sviluppo del progetto procede per iterazioni brevi (solitamente della durata da 1 a 4 settimane) di lunghezza fissa. Durante l'iterazione i team si auto-organizzano; l'iterazione non può essere interrotta per delle variazioni dei requisiti. Al contrario, al termine di un'iterazione si può modificare l'insieme dei requisiti se la stima iniziale si rivela errata.

Si mostrano al cliente i risultati per ottenere un feedback (molto importante nei modelli agili), dando loro la possibilità di variare i requisiti stessi ripianificando così le iterazioni successive, anche in relazione alla stima della velocità di sviluppo.

Durante tutte le iterazioni vige il "principio di semplicità", secondo il quale non si devono implementare caratteristiche e funzionalità che non siano strettamente legate ai requisiti definiti inizialmente o in corso d'opera dal cliente. [Mar09]

Al termine di ogni iterazione, si effettua la ridefinizione delle priorità delle user story con l'eventuale aggiunta di nuovi requisiti (processo denominato "Planning Game") prima di entrare nell'iterazione successiva.

Una caratteristica fondamentale introdotta nel modello agile è il refactoring.

Il refactoring nasce per perfezionare il codice esistente senza cambiarne la funzionalità: il miglioramento della struttura interna del sistema durante lo sviluppo del progetto permette di poter effettuare modifiche più facilmente e impedisce il degrado dovuto ai continui cambiamenti. Il refactoring, quindi, semplifica il codice,

rimuove il codice ridondante e "butta via" (commentando o inserendo in un file di backup) alcune parti già scritte non pienamente soddisfacenti, riscrivendole da capo in una maniera diversa.

Il refactoring infine, usa i test creati nella fase di Test-Driven Development per assicurarsi che il nuovo codice appena scritto e le varie modifiche apportate non introducano errori.

Alcuni modelli agili, non tutti, utilizzano la pair programming (programmazione di coppia) per massimizzare la comunicazione tra gli sviluppatori dei team. È una tecnica che prevede due progettisti che lavorano insieme allo stesso compito, su un solo computer in un'unica postazione di lavoro.

Uno dei due progettisti, il "driver", controlla tastiera e mouse e scrive l'implementazione del codice. Il suo obiettivo principale è quello di portare a termine una soluzione funzionante del problema in considerazione.

L'altro, il "navigator", svolge il ruolo di supervisione e revisione simultanea del codice: osserva l'implementazione cercando difetti, segnalando gli eventuali errori al driver, proponendo strategie alternative di soluzione e, su richiesta, partecipa ai brainstorming.

Questi due ruoli vengono scambiati tra i due progettisti in maniera periodica (di solito lo scambio avviene giornalmente).

La coppia inizia scrivendo, ogni giorno, i test ed il codice relativi ad una parte di una user story. In seguito, effettua tutto il test di unità previsto, esegue l'integrazione di questi test e del codice con quelli creati i giorni precedenti e conclude con l'esecuzione di tutti i test di regressione.

Il giorno seguente, a mente sgombra, la coppia passa alla parte successiva della user story, oppure alla user story

seguente: l'obiettivo è prevenire il cosiddetto "Integration Hell", ossia le difficoltà relative all'integrazione di grosse porzioni di software sviluppati in modo indipendente su lunghi periodi di tempo e che, di conseguenza, potrebbero risultare significativamente divergenti al momento dell'integrazione. Lavorare fino a tarda notte danneggia le prestazioni: uno sviluppatore stanco commette più errori di uno fresco e riposato e, alla lunga, il gioco non vale la candela: per questo appena una parte della user story viene completata, è quasi obbligatorio iniziare la parte successiva il giorno dopo.

C'è una regola che dice che il codice che qualsiasi progettista scrive appartiene al progetto e non al progettista stesso ("collective ownership"). Perciò, durante lo sviluppo, chiunque può osservare, prendere spunto, copiare e modificare qualsiasi parte del codice già implementata.

A causa del refactoring, della pair programming e della collective ownership, occorre avere dei metodi brillanti, efficaci e veloci per poter addentrarsi con facilità in qualsiasi punto del codice altrui. Il fulcro di questo problema è commentare il codice: più alto è il numero dei commenti (solo quelli utili, senza esagerare), più facile sarà addentrarsi nel punto desiderato. Si cerca sempre di scrivere del codice che "svela" da solo il suo scopo e le sue funzionalità, quindi:

- Se il codice richiede un commento, riscrivilo
- Se non puoi spiegare il codice con un commento, riscrivilo

Infine, come accennato in precedenza, una caratteristica molto importante dei modelli agili è la presenza del cliente. Il cliente è sempre disponibile per chiarificare le user story e per prendere rapidamente decisioni critiche. In questo modello viene introdotta la

comunicazione “faccia a faccia”, che minimizza la possibilità di ambiguità ed equivoci. [Hig03]

Alla luce di ciò, gli sviluppatori non devono fare ipotesi, ma devono comunicare direttamente con il cliente e attenersi alle loro decisioni.

Il cliente deve essere coinvolto ad ogni step dello sviluppo perché, come dice la legge di Humphrey, “i clienti non sanno mai cosa desiderano fin quando non vedono il software funzionante”. Se i clienti non vedono il software funzionante fino alla fine del progetto, sarà troppo tardi incorporare le loro ulteriori richieste e suggerimenti.

Alcuni studi statistici hanno evidenziato che, quando non ci sono problemi di comunicazione tra sviluppatori e cliente, i team sono 50 volte più produttivi e hanno un indice di riuscita raddoppiato rispetto alla media del settore.

Gli autori del Manifesto Agile non sbagliavano quando sottolineavano che il coinvolgimento del cliente nel processo di sviluppo software è essenziale per la riuscita del progetto.

Attualmente, esistono molti modelli di sviluppo che seguono la metodologia agile, tra cui Extreme Programming (XP), Feature-Driven Development (FDD), Adaptive Software Process, Crystal Light Methodologies, Dynamic Systems Development Method (DSDM), Lean Development e Scrum. [Sut13]

3.4 Differenze tra modello iterativo e modello agile

Pur basandosi sulle iterazioni introdotte nel modello iterativo, il modello agile ha, rispetto ad esso, apportato delle migliorie alla sua strutturazione e al modo di gestire il processo di sviluppo del software. [Lar04]

Al contrario del modello iterativo, che è flessibile ma pianificato in quanto i requisiti vengono definiti fin dall'inizio, il modello agile è anch'esso flessibile ma non è pianificato: il processo di sviluppo è guidato dall'utente e dai test che vengono effettuati ad ogni iterazione e i requisiti possono essere modificati o introdotti in ogni momento.

Per quanto riguarda le dimensioni dell'organo lavorativo, il modello agile è adatto a piccoli prodotti e team ridotti (dato che il progetto viene suddiviso in sottoparti, ciascuna assegnata ad un team). La conoscenza tacita usata da ogni piccolo team crea, però, una limitazione della scalabilità verso prodotti più grandi. Contrariamente ad agile, il modello iterativo è consigliato per sviluppare grandi progetti ed è adatto a grandi team. Non è prevista conoscenza tacita, quindi il progetto può essere facilmente scalato verso prodotti più piccoli, ma ad un costo elevato. [Cho08]

Il fattore dinamismo rappresenta una differenza sostanziale tra i due modelli. Quello agile è presentato come un modello molto dinamico, usa il refactoring che permette di riscrivere, cambiare, introdurre, eliminare parti del codice implementato in qualsiasi momento. Il modello iterativo, invece, è un modello pianificato nel dettaglio fin dall'inizio del ciclo di vita del software ed è difficile apportare modifiche in corso d'opera (è un modello poco dinamico).

L'ultima differenza riguarda il personale e la documentazione. Nel modello agile, all'interno del personale servono sicuramente degli esperti capaci di gestire un processo di sviluppo che si basi appunto su agile. La documentazione è ridotta al minimo poiché si preferisce scrivere del codice chiaro, che spieghi "da solo" le sue funzionalità. Questo modello piace a chi preferisce avere libertà di lavorare in qualsiasi direzione. Il modello iterativo, al contrario, prevede una

documentazione corposa, che spieghi i motivi della prosecuzione del progetto dopo ogni ciclo in maniera dettagliata. All'interno del personale si richiede la presenza di esperti che coordinino la fase della pianificazione. Questo modello solitamente piace a chi preferisce lavorare con ruoli e procedure ben definiti. [Lar04]

Capitolo 4

4 LA NASCITA DI SCRUM

4.1 Com'è nato Scrum?

"*L'approccio a 'staffetta' nello sviluppo di un prodotto...potrebbe entrare in conflitto con l'obiettivo di massimizzare flessibilità e velocità. Al contrario, un approccio "olistico" (a mischia stile rugby), dove una squadra cerca di avanzare un passo alla volta, potrebbe essere più funzionale per centrare i requisiti di competitività.*" (Hirotaka Takeuchi, Ikujiro Nonaka). [NonTak86]

Nel 1986, Hirotaka Takeuchi e Ikujiro Nonaka descrissero un nuovo approccio allo sviluppo di prodotti software commerciali che avrebbe aumentato la velocità e la flessibilità, basato su casi di studio presi dall'industria automobilistica e quella relativa alla realizzazione di fotocopiatrici e stampanti.

Questo nuovo approccio venne inizialmente chiamato "approccio olistico" o "rugby", in quanto l'intero processo viene eseguito da un team interfunzionale su più fasi che si sovrappongono, dove il team "*cerca di raggiungere l'obiettivo come unità, passando la palla avanti e indietro*" (come, appunto, accade nel rugby). [NonTak86]

Il termine "Scrum" (che non è un acronimo) deriva quindi da quel termine del rugby che indica l'intera squadra in mischia che, passo dopo passo, procede per arrivare alla meta. Risulta evidente che questa sia una metafora, che ci aiuta a capire l'idea di base di Scrum: un team di sviluppo che deve lavorare insieme in modo che tutti gli sviluppatori del progetto spingano nella stessa direzione, agendo come un'unica entità coordinata che ha come obiettivo finale la realizzazione del prodotto. [Ver13]

Ken Schwaber e Jeff Sutherland hanno presentato per la prima volta Scrum alla conferenza OOPSLA del 1995. Questa presentazione ha essenzialmente documentato ciò che essi avevano appreso negli anni precedenti applicando Scrum. Nonostante Scrum fosse stato presentato da Ken Schwaber e Jeff Sutherland, i suoi "genitori" restano sicuramente Hirotaka Takeuchi e Ikujiro Nonaka, che lo crearono nel 1986. [Fra10]

Jeff Sutherland lo definisce come "*un framework di processo utilizzato dai primi anni 1990 per gestire lo sviluppo di prodotti complessi. Scrum non è un processo o una tecnica per costruire prodotti ma piuttosto è un framework all'interno del quale è possibile utilizzare vari processi e tecniche. Scrum rende chiara l'efficacia relativa del product management e delle pratiche di sviluppo usate in modo da poterle migliorare*." [SchSut11]

Scrum viene quindi identificato come un esempio di modello agile, attualmente il più usato tra le grandi società informatiche.

Come in parte accennato in precedenza, la metodologia Scrum è un esempio di modello agile che può essere usato per gestire e controllare lo sviluppo di prodotti software complessi. Scrum definisce un set di attività che consentono ai team di offrire più valore al cliente, in meno tempo. Queste attività consentono al cliente di esaminare, guidare ed influire sul lavoro dei team durante l'intera fase di sviluppo. Come vedremo in seguito, questo approccio non tenta di definire tutti i requisiti e le caratteristiche del prodotto all'inizio del progetto ma, dopo ogni iterazione (chiamata sprint), incrementa e perfeziona ciò che è stato creato in precedenza.

Essendo agile, Scrum si basa strettamente sui valori citati nel Manifesto Agile: è importante notare come, per ottenere i risultati sperati, i team di sviluppo debbano riconoscere ed apprezzare questi valori e principi, che costituiscono il fondamento di tutti i processi agili.

Il cofondatore di Scrum Ken Schwaber, introdusse nella metodologia un processo fondamentale, il processo di controllo, basato su tre rami fondamentali:

1. Trasparenza: ogni aspetto del processo che influenza il risultato deve essere visibile e conosciuto da ciascun membro coinvolto nel progetto. Gli aspetti significativi, soprattutto, devono essere visibili ai responsabili del lavoro. La trasparenza richiede che tutti quegli aspetti siano definiti da uno standard comune, in modo tale che i membri condividano una comune comprensione di ciò che deve essere fatto. Ad esempio: un linguaggio comune di riferimento al processo deve essere condiviso da tutti i partecipanti al progetto.

2. Ispezione: tutti gli aspetti del processo devono essere controllati con elevata frequenza affinché gli scostamenti indesiderati possano essere rilevati con facilità e tempestività. Inoltre, i clienti devono ispezionare frequentemente le funzionalità implementate e il progresso verso l'obiettivo fissato in modo da rilevare le variazioni non richieste. I controlli non devono però essere così frequenti da superare la soglia di tolleranza e rappresentare un'interruzione al lavoro: sono più utili quando eseguiti raramente ma in modo preciso ed approfondito.

3. Adattamento: se chi ispeziona verifica che uno o più aspetti del processo sono al di fuori dei limiti accettabili e che il prodotto finale non potrà sicuramente essere approvato, bisogna regolare il processo e riportarlo dentro al range stabilito. Questo adattamento deve essere portato a termine il più rapidamente possibile, per ridurre al minimo l'uscita dal range.

Solitamente, il processo di controllo è gestito dagli sviluppatori con molta esperienza e maggiormente informati.

Il modello Scrum prescrive tre occasioni formali (di cui parleremo in seguito) per l'ispezione e l'adattamento:

1. Sprint Planning Meeting
2. Daily Scrum Meeting
3. Sprint Review Meeting

[SchSut11]

4.2 Le componenti

Il modello Scrum è costruito su tre principali componenti: ruoli, meeting e oggetti.

Ci sono tre ruoli distinti riconoscibili all'interno del processo di sviluppo: il proprietario del prodotto, il team e lo Scrum Master.

Il proprietario del prodotto (Product Owner) è il responsabile dell'ottenimento del finanziamento iniziale e di quelli in corso d'opera, creando e definendo i requisiti globali, l'indice ROI (Return On Investment) che si vuole realizzare e rilasciando una prima stesura del piano del progetto. [Cho08]

Il compito più importante del Product Owner è di massimizzare il valore del prodotto e il lavoro svolto dal team. Ha la responsabilità esclusiva di gestire il Product Backlog (di cui parleremo in seguito).

Il proprietario del prodotto è solitamente un manager che conosce ciò che deve essere costruito per realizzare il progetto e come il processo di sviluppo dovrebbe progredire. Il Product Owner è quindi una persona, non un comitato. Può però rappresentare un desiderio di un comitato nel Product Backlog, ma chiunque voglia cambiare la priorità di un elemento deve convincere il Product Owner.

Affinché il ruolo Product Owner abbia successo, all'interno dell'organizzazione tutti devono rispettare le sue decisioni. A nessuno è permesso dire al Team di lavorare con un diverso ordine di priorità e in maniera differente da quanto disposto dal proprietario del prodotto, e i Team non sono autorizzati ad ascoltare chi sostiene il contrario. [SchSut11]

Il team è il responsabile dell'implementazione delle funzionalità descritte dai requisiti espressi.

I team sono autogestiti, auto-organizzati (il ruolo di leader non è fisso ma cambia a seconda delle necessità espresse in ogni iterazione) e interfunzionali al fine di massimizzare la performance loro e degli altri team.

Solitamente, il team è composto da 5-10 membri che lavorano sul progetto a tempo pieno: i singoli membri hanno diverse competenze specialistiche e aree di focus. Tutti i membri del team sono responsabili sia del successo che del fallimento della realizzazione dei sottoprogetti o, in casi estremi, dell'intero progetto. [Cho08]

I team non contengono sotto-team che si dedicano a particolari campi, come analisi o business.

Lo Scrum Master è il ruolo assunto tradizionalmente dal manager del progetto di un team leader. È responsabile di garantire che i valori, le prassi e le regole di Scrum vengano promulgate e, soprattutto, applicate. Lo Scrum Master è inoltre responsabile di eliminare qualsiasi impedimento posto agli sviluppatori durante il processo di sviluppo. [Cho08]

Lo Scrum Master lavora sia per il Product Owner, che per il team di sviluppo.

Opera per il Product Owner in diversi modi:

- Trova tecniche per la gestione efficace del progetto
- Comunica con chiarezza gli obiettivi e la visione al team di sviluppo
- Aiuta la pianificazione a lungo termine del prodotto
- Presiede i meeting se richiesto dal Product Owner

Lo Scrum Master lavora anche per il team:

- Insegna e guida il team per creare prodotti di alto valore
- Elimina qualsiasi ostacolo al progresso
- Aiuta a capire e mettere in atto Scrum nel processo di sviluppo
- Aiuta nell'aumento di produttività

[SchSut11]

Il secondo componente identificabile nel modello Scrum è il meeting.

Ci sono molti meeting in questo modello: il Daily Scrum Meeting (TDSM), il Daily Scrum of Scrums Meeting (TDSSM), lo Sprint Planning Meeting (TSPM) e lo Sprint Review Meeting (TSRM).

Il Daily Scrum Meeting è un meeting tenuto ogni giorno tra lo Scrum Master (che lo dirige come "chair") e i team. Ha una durata di 15 minuti, nella quale i team espongono cosa hanno terminato dal meeting precedente, quali parti saranno realizzate prima del prossimo meeting e quali ostacoli hanno riscontrato. Questo meeting facilita molto la comunicazione, identifica e rimuove gli impedimenti riscontrati, evidenzia e promuove le rapide prese di decisione e migliora la trasparenza (come ramo del processo di controllo descritto in precedenza). Bisogna dire, però, che il Daily Scrum Meeting non serve a risolvere i problemi

riscontrati dai team bensì invoglia i team a fare affidamento sugli altri e sullo Scrum Master affinchè il lavoro possa procedere nel modo più conveniente e senza intralci.

Il Daily Scrum of Scrums Meeting è un altro breve meeting giornaliero che segue la stessa forma del Daily Scrum Meeting. L'unica ragione per la quale si tiene questo meeting è riuscire a sincronizzare il lavoro tra i vari team che si occupano del progetto, favorendo l'integrazione, la sovrapposizione e la comunicazione. [Fra10]

Lo Sprint Planning Meeting è un meeting che si tiene una volta al mese all'inizio di ogni sprint (ossia l'iterazione); lo Scrum Master, il proprietario del prodotto e il team si riuniscono per discutere su ciò che dovrà essere completato nello sprint seguente, che dura solitamente 30 giorni. Questo meeting si suddivide in due momenti chiave: nel primo, si modifica e si stila il backlog (Product Backlog): si tratta fondamentalmente di una lista contenente tutti i requisiti del progetto. In seguito, il team definisce formalmente gli obiettivi che si prefissa di raggiungere al termine dello sprint che sta per iniziare. Nel secondo momento chiave del meeting, i membri del team si occupano di suddividere il progetto in tante sottoparti minori (lo Sprint Backlog), affinché tutte queste sottoparti più semplici possano essere iniziate e finite nel singolo sprint che sta per iniziare.

Lo Sprint Review Meeting è, come il precedente, un meeting che si tiene mensilmente, ma alla fine dello sprint. Partecipano il proprietario del prodotto, il team e i clienti (stakeholder). Durante il meeting, i membri del team mostrano ai clienti e al proprietario del prodotto tutto ciò che è stato sviluppato durante lo sprint appena concluso. La differenza tra lo Sprint Review Meeting e un meeting tradizionale è l'informalità con la quale il team dimostra le funzionalità sviluppate, informalità che non

crea al team alcun tipo di pressione e distrazione dal lavoro. [Fra10]

Il terzo ed ultimo componente del metodo Scrum è identificabile negli oggetti, che sono: il Product Backlog, lo Sprint Backlog e il Burndown Chart.

Il Product Backlog o, semplicemente, backlog, è una lista contenente i requisiti del progetto, ordinati in base alla loro priorità. La priorità viene definita in base alla user story relativa, alla complessità, all'importanza sul mercato, alla stima dell'ammontare di ore o giorni di lavoro che si pensa verranno impiegate per realizzare quel requisito (tutto ciò durante lo Sprint Planning Meeting). Sulla base di questa stima, viene definita la velocità del team e lo sforzo speso durante lo sprint seguente. Diversamente dai progetti tradizionali, il backlog è creato e gestito direttamente dal proprietario del prodotto. [Sch04]

Similarmente, lo Sprint Backlog è un sottoinsieme di requisiti del Product Backlog (di solito quelli con priorità più alta), che vengono scelti per essere sviluppati in un determinato sprint. Diversamente dal Product Backlog, questo backlog viene incrementato ogni giorno e può contenere fino ad un massimo di 300 requisiti da sviluppare. Inoltre, lo Sprint Backlog è interamente creato e gestito dai membri del team di sviluppo del progetto.

L'ultimo oggetto della metodologia Scrum è il burndown chart. Questo, è un grafico accessibile da tutti i membri che lavorano al progetto, e mostra lo sviluppo del lavoro nel tempo (nel grafico, il tempo è rappresentato sull'asse delle ascisse e il lavoro sull'asse delle ordinate). Ci sono tre tipi di burndown chart comunemente usati: lo sprint burndown chart (mostra il progresso dello sprint in corso), il release burndown chart (mostra il progresso della release) e il product burndown chart (mostra il

progresso globale dell'intero progetto). L'obiettivo del burndown chart è di fornire informazioni in maniera chiara, immediata e facilmente comprensibile. [Sch04]

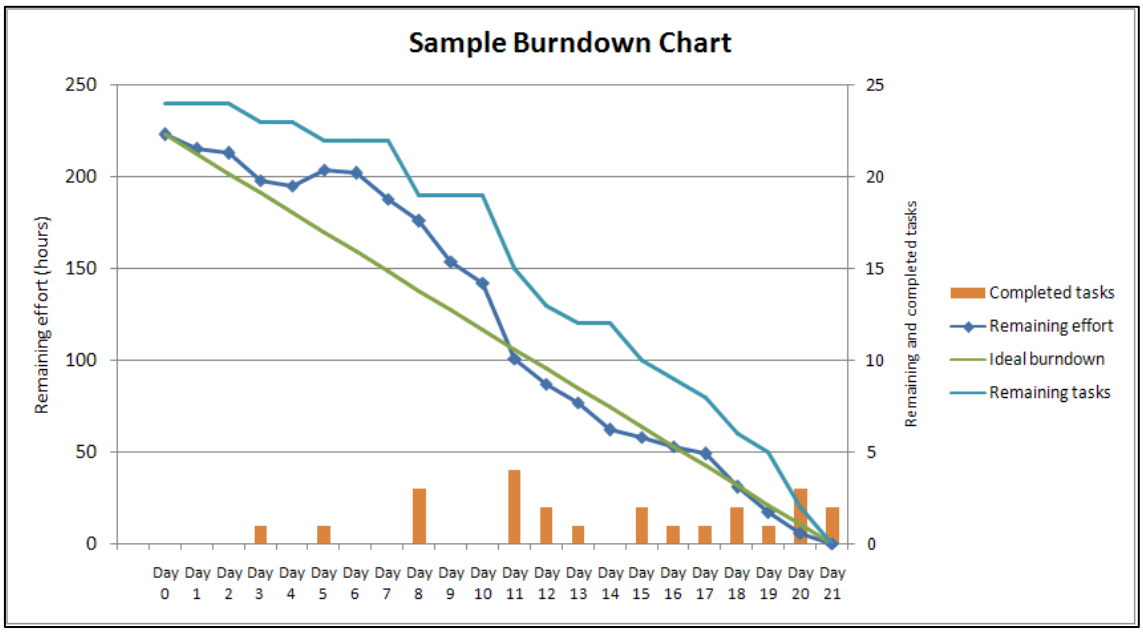

Figura 4.1 Esempio di burnchart down [Web5]

4.3 Il processo di sviluppo in Scrum

Le componenti descritte in precedenza interagiscono tra loro durante il processo di sviluppo del software.

Prima di avviare il processo di sviluppo, è necessario stabilire il business case.

Il business case è un insieme strutturato di informazioni utili per determinare se un progetto sia auspicabile, fattibile e realizzabile (quindi idoneo all'investimento). Il business case è statico e deve quindi essere continuamente aggiornato su costi, rischi e benefici relativi al progetto. Comprende anche le giustificazioni, le motivazioni e le esigenze aziendali per le quali si vuole realizzare il prodotto richiesto e che porteranno, se idonee, all'ottenimento di un finanziamento.

All’avvio del processo di sviluppo, si definisce subito una visione del sistema, definita in termini di business anziché tecnici. La vision inizialmente potrà non essere molto chiara: diventerà più precisa durante lo sviluppo del progetto. [Sch04]

In seguito, il Product Owner si occupa di trovare i finanziamenti necessari, presentare la vision a tutti i partecipanti al progetto, creare il Product Backlog e definire il ROI (Return On Investment: redditività ed efficienza economica del capitale che si sta per investire per il progetto); sulla base di questo indice e allo scopo di massimizzarlo, vengono pianificate tutte le fasi di sviluppo del progetto.

Definiti tutti questi particolari, inizia lo Scrum Flow (il ciclo di sviluppo).

Si parte con lo Sprint Planning Meeting: i requisiti con priorità più alta all’interno del Product Backlog vengono suddivisi in sottogruppi minori e vengono posti nello Sprint Backlog. Il Product Owner spiega ai membri del team il contenuto, il significato, lo scopo e le intenzioni di ogni oggetto presente nel Product Backlog. I membri, a loro volta, possono fare qualsiasi domanda al Product Owner in merito a chiarimenti riguardanti tali oggetti.

Al termine del meeting, inizia lo sprint. Lo sprint è un’iterazione della durata di 2-4 settimane (a discrezione del team), nel quale il team sviluppa tutti i requisiti presenti nello Sprint Backlog. Nel corso dello sprint, questi requisiti non possono essere modificati o rimossi dal backlog. [SchSut11]

Durante queste settimane si tengono, quotidianamente, il Daily Scrum Meeting, dove il team espone quali requisiti ha sviluppato e quali svilupperà nei giorni rimanenti dello sprint, e il Daily Scrum of Scrum Meeting, nel quale si prevede la comunicazione, la sincronizzazione e integrazione del lavoro svolto da tutti i team.

Dopo massimo un mese, al termine dello sprint, si tiene lo Sprint Review Meeting: i team mostrano materialmente al cliente i requisiti sviluppati nel corso dell'ultimo sprint e le migliorie apportate al progetto. Sebbene questi requisiti siano pronti e completi per poter essere rilasciati al termine dello sprint, è possibile che non apportino sufficiente valore funzionale da giustificare effettivamente il rilascio del prodotto. È necessario quindi ricominciare il ciclo di sviluppo per implementare altre funzionalità e completare il progetto. [SchSut11]

Al termine, il ciclo di sviluppo Scrum Flow ricomincia.

Man mano che i team procedono con il lavoro, il Product Backlog può essere incrementato con ulteriori requisiti, spesso aggiunti dal cliente, mentre alcuni (ritenuti inutili) possono essere rimossi. Inoltre, la priorità di alcuni requisiti può cambiare durante lo sviluppo del progetto.

Quando i requisiti sono stati tutti sviluppati, il prodotto è stato testato e non presenta alcun bug, il software è pronto per essere rilasciato definitivamente sul mercato. [SchSut11]

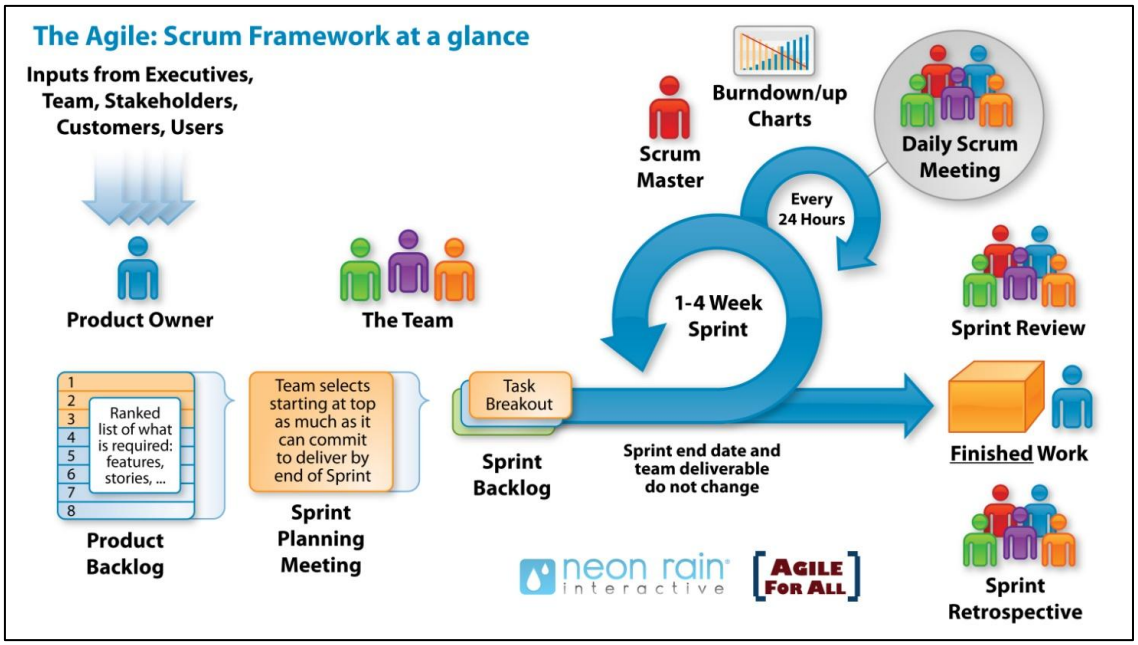

Figura 5.2 Processo di sviluppo in Scrum [Web6]

4.4 Difficoltà e punti di forza

È risaputo che una comunicazione inefficiente o addirittura inesistente è la causa principale della maggior parte del fallimenti riguardanti lo sviluppo di prodotti software.

Nonostante il modello Scrum riconosca l'importanza della comunicazione, essa continua ad essere una difficoltà: il Daily Scrum Meeting introduce un metodo comunicativo efficiente tra i membri di un singolo team; tuttavia, ogni team all'interno del progetto è generalmente separato dagli altri e non c'è mai abbastanza comunicazione tra tutti i team. Questa mancanza, in primo luogo, potrebbe causare problemi come la duplicazione del lavoro: due team non comunicano e svolgono quindi le stesse attività.

Questo problema potrebbe essere risolto o, al limite, mitigato, mediante un meeting tra gli Scrum Master di ogni team, in modo che ognuno di essi si possa assicurare che il suo team non stia duplicando il lavoro di un altro.

Se la comunicazione tra i team rappresenta un problema, quella con il cliente ne rappresenta uno ancora più grande: le maggiori difficoltà comunicative vengono riscontrate con il cliente, perché tende molto spesso a non fornire un feedback adeguato. [Sut12]

Purtroppo, il coinvolgimento del cliente nel processo di sviluppo del software è una parte cruciale, che garantisce la maggior parte delle volte la riuscita del progetto.

Nonostante il cliente ne sia a conoscenza, i team non ottengono mai il feedback desiderato: i clienti sono sempre impegnati e trovano sempre altre cose da fare piuttosto che parlare con gli sviluppatori quando essi hanno bisogno. Accettano spesso la maggior parte del prodotto creato dai team, a cui chiedono soltanto delle piccole modifiche.

Il più delle volte sembra che i clienti non sappiano cosa realmente vogliono sia implementato nel prodotto e questo rappresenta un ostacolo sia per i team di sviluppo, che non sanno bene cosa realizzare, che per i clienti stessi, che non ottengono il coinvolgimento necessario all'interno del progetto. [Fra10]

Il punto di forza principale nell'utilizzo della metodologia Scrum è la sua semplicità.

In Scrum, i ruoli sono ben definiti e non ci sono incomprensioni né intromissioni; i requisiti possono essere completamente testati e sviluppati in brevi cicli iterativi e i membri di ciascun team sostengono una responsabilità maggiore ma equamente divisa, che copre la parte del progetto che sviluppano.

Nonostante i problemi citati in precedenza, i metodi usati da Scrum rafforzano la già vasta comunicazione, che aiuta i team ad organizzarsi con maggiore efficienza. Ovviamente una frequente comunicazione non significa che la documentazione non sia richiesta e necessaria: senza gli adeguati controlli, il progetto potrebbe degenerare in un

disastro dovuto anche ad una documentazione scritta in maniera inadeguata. [Fra10]

Scrum è uno dei modelli più adatti al miglioramento delle prestazioni durante ciascuna fase del ciclo di sviluppo del software, perché si focalizza sull'eliminazione del superfluo, dei processi non necessari e delle pratiche gestionali non importanti.

Molto spesso, per cercare di comprendere al meglio i punti di forza di Scrum, lo si compara con i modelli precedenti.

I benefici riportati dai team derivano anche dai loro anni di esperienza, e riguardano:

• Produttività: il 68% dei team ritiene che Scrum abbia un alto grado di produttività. Il 27% lo ritiene un modello con una produttività mediocre. Il 5% lo ritiene il peggiore modello in questi termini.

• Team: il 52% dei team ha una buona cooperazione. Il 39% riporta una media collaborazione. Il 9% dei team ha una scarsa comunicazione.

• Adattabilità: il 63% dei progetti è facilmente adattabile ai cambiamenti dei requisiti e a quelli di mercato. Il 33% riporta una media capacità di adattamento. Il restante 4% non risulta essere molto adattabile.

L'85% dei team vorrebbe continuare ad usare Scrum (se la decisione di farlo spettasse a loro). [Sut12]

Nonostante il problema della comunicazione e alla luce dei punti di forza appena descritti, Scrum viene utilizzato e seguito da molte società (anche di alto livello) come linea guida per sviluppare i loro software:

- Oracle
- Adobe
- BBC

- Ibm
- Google
- Yahoo
- Philips
- Siemens
- Nokia
- Microsoft

Capitolo 5

SCRUM ALLA MICROSOFT

Come molte altre aziende, la Microsoft segue il modello Scrum per sviluppare i software richiesti dal mercato.

Secondo un sondaggio effettuato nel 2008 tra le industrie ingegneristiche di 80 nazioni diverse, il 49% di esse segue strettamente Scrum durante il processo di sviluppo software.

Lo stesso sondaggio è stato proposto agli ingegneri del software che lavorano alla Microsoft: più del 60% utilizza Scrum. [BMNW11]

Viene descritta in seguito l'esperienza che tre team della Microsoft hanno avuto lavorando con il metodo Scrum, insieme a 9 prassi costruttive ingegneristiche (la prima prassi è proprio il modello Scrum). Queste prassi sono attribuibili alla Microsoft Engineering Excellence, ossia un'organizzazione che supporta gli aspetti ingegneristici e propaga le prassi all'interno delle industrie e direttamente nell'ecosistema dell'information tecnology.

I risultati indicheranno come questi team siano diventati capaci di migliorare qualità, produttività, stima e precisione attraverso la combinazione di Scrum e delle prassi costruttive. Le informazioni relative a questa esperienza sono state ottenute intervistando alcuni membri di tutti e tre i team studiati e raccogliendo materiale sul lavoro svolto.

	Team A	Team B	Team C
tipo di progetto	Scrum	Scrum	Scrum
dimensione team	4	3	19
luogo	Redmond, Shanghai	Redmond	Redmond, Beijing
10anni esperienza	1	1	1
5anni esperienza	3	2	18
linguaggio	C#	C#	C++
padronanza linguaggio	alta	media	alta
linee di codice	24.952	8.826	31.399
linee di test	20.912	4.031	26.283
tempo impiegato	14 mesi	11 mesi	18 mesi
testing	ogni giorno	ogni giorno	ogni giorno
comunicazione team	email, chat, telefono	email	email, di persona

Tabella 3.1 Contestualizzazione team Microsoft [Cus97]

La tabella 3.1 fornisce una contestualizzazione dei tre team: dimensione, luogo, anni di esperienza ecc.

I team A è un team di 4 membri, suddivisi tra gli Stati Uniti e la Cina (precisamente a Redmond e Shangai). Come il team A, il team B è molto piccolo (solo 3 elementi), ma lavora interamente in un'unica città, Redmond (Stati Uniti). Infine, il team C è il più numeroso dei tre (vanta ben 19 membri) ed è anch'esso collocato in due città diverse, Redmond e Beijing (Stati Uniti, Cina). [BMNW11]

Prima di iniziare il processo di sviluppo, i team hanno dovuto comprendere un concetto fondamentale: in ambiente Scrum vige la regola "all or nothing" (tutto o niente); un

requisito realizzato al 99% non viene considerato completo. Alla fine di ogni sprint i team devono mostrare le nuove funzionalità implementate durante il meeting: le aspettative sono sempre alte e i team devono lavorare duramente per raggiungere gli obiettivi prefissati dopo un certo numero di iterazioni.

In seguito, per stimare le ore/persona richieste per completare lo sviluppo dei requisiti previsto in ciascuno sprint, i team utilizzano la tecnica del Planning Poker (la seconda prassi), utilizzata durante lo Spring Planning Meeting. La sessione del Planning Poker inizia quando i clienti o i rappresentanti del mercato illustrano i requisiti richiesti a tutti gli elementi coinvolti nel processo di sviluppo (team di sviluppatori, manager, tester, Scrum Master, proprietario del prodotto..). A turno, ogni team discute con il cliente sul lavoro da svolgere per soddisfare pienamente i requisiti, finché non ottiene tutte le informazioni necessarie per effettuare una stima dello sforzo (che farà privatamente in seguito). I team che hanno realizzato la stima più alta e quella più bassa la esporranno agli altri team: la discussione continua fino a quando tutti i team non saranno pronti per effettuare nuovamente una stima dello sforzo. I round del Planning Poker finiscono quando tutti i team esprimono la stessa stima dello sforzo. [GJM04]

In sintesi, il Planning Poker serve a:

- Ottenere una visione comune del progetto
- Esporre aspetti tecnici di implementazione e verifica
- Discutere sull'implementazione dei requisiti e su tutto ciò che comporta
- Appianare e risolvere ambiguità relative a diverse percezioni sui requisiti

- Esporre alternative facili e complesse per il raggiungimento degli obiettivi prefissati

Effettuato il Planning Poker, che ha determinato una bassa stima dello sforzo, e definiti i requisiti in maniera opportuna, i team B e C iniziano il processo di sviluppo con iterazioni della durata di 4 settimane, mentre il team A inizia con iterazioni di 2 settimane, ritenendo più semplice effettuare stime e pianificare per un periodo più breve, che consente anche di rispondere velocemente ai cambiamenti del mercato e di ridurre i rischi. [BMNW11]

Durante le iterazioni, soltanto tre volte a settimana, i team partecipano al Daily Scrum Meetin organizzato a Redmond.

Una delle prassi costruttive ingegneristiche più usata e ritenuta maggiormente importante è la continua integrazione (terza prassi), prassi nella quale i membri di un team integrano e sincronizzano il loro lavoro nel progetto principale con cadenza molto frequente (solitamente, ogni sviluppatore integra il proprio lavoro almeno quotidianamente). Ogni integrazione è controllata e verificata da un processo automatico che rileva gli errori il più velocemente possibile. In seguito, il team che ha sincronizzato il lavoro riceve una email di conferma che attesta la riuscita dell'integrazione (o il fallimento in caso di errore).

La quarta prassi, la meno usata, è lo sviluppo test-driven, che consiste nel realizzare un test di fallimento e il relativo codice per superarlo. Nessuno dei tre team in esame, però, ha scelto di seguire questa prassi.

Basandosi sul criterio "all or nothing" discusso in precedenza, ogni team della Microsoft ritiene importante la qualità di ogni parte del prodotto, evidenziando alcuni criteri di qualità descritti nella quinta prassi ingegneristica (Quality Gates):

- Tutti i test di qualità devono essere superati
- Il codice condiviso deve essere ben documentato
- Non devono essere presenti errori o warning
- Il prodotto deve rispecchiare ciascuna richiesta del cliente

Tutti i team considerano il prodotto completo solo se soddisfa tutti questi criteri di qualità.

La sesta prassi è il controllo delle sorgenti, ossia la gestione, mediante un sistema di controllo, dei cambiamenti relativi a documenti, programmi, documentazione e qualsiasi altra informazione archiviata nei computer della Microsoft. Soltanto gli amministratori del progetto possono aggiungere materiale all'interno del sistema: il codice inserito verrà estromesso soltanto alla fine di uno sprint. Questa prassi è molto utile quando si vuole archiviare qualcosa di completo ma momentaneamente non necessario, senza perderlo, per poi recuperarlo in futuro.

La settima, l'ottava e la nona prassi riguardano la gestione del codice. La settima, Code Coverage, monitora l'intero codice per trovare il "codice morto" e le aree che necessitano di essere testate al più presto. L'ottava, la Peer Review, è la prassi che consente agli sviluppatori con maggiore esperienza di condurre il controllo del codice, per migliorare la qualità e rimuovere gli errori che potrebbero sfuggire all'occhio di uno sviluppatore meno esperto. L'analisi statica è l'ultima prassi, la nona. Essa identifica gli errori più comuni che vengono commessi durante la fase di sviluppo. Queste ultime tre prassi non

sono molto seguite dai team in esame, poiché sono risultate essere molto costose e molto dannose se gestite in modo errato. [BMNW11]

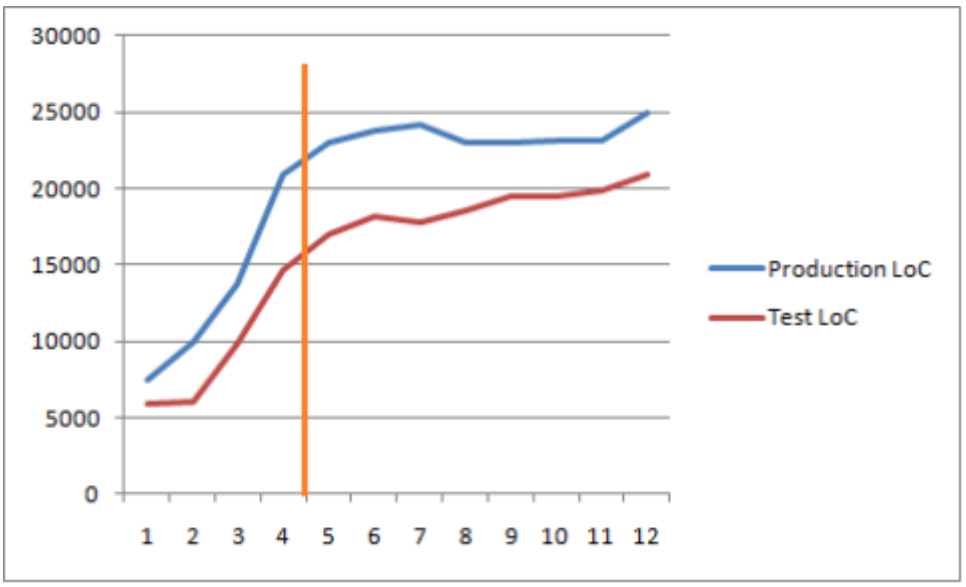

Tabella 3.1 Risultati ottenuti [Cus97]

In tutti e tre i casi, i team hanno raggiunto il primo rilascio del prodotto lavorando interamente con il linguaggio C++ o C#, producendo tra 9 e 31 milioni di righe di codice in un lasso di tempo dagli 11 ai 18 mesi.

I risultati ottenuti dai tre team della Microsoft al termine di questa esperienza sono stati pienamente soddisfacenti. Il livello di produttività è stato migliorato e incrementato del 90% e, l'utilizzo delle prassi costruttive ingegneristiche ha consentito la realizzazione di un notevole numero di storie utente in ogni sprint, sempre in accordo con i criteri di qualità. [BMNW11]

CONCLUSIONE

In questo elaborato viene presentato il graduale sviluppo dei modelli di processo; dal primo, code-and-fix, all'ultimo, Scrum.

I cambiamenti dagli anni 70 fino ad oggi sono stati notevoli e di grande importanza ed impatto sullo sviluppo del software. Cambiamenti e miglioramenti riguardanti l'analisi dei requisiti, il rapporto con il cliente, la pianificazione dello sviluppo, l'organo lavorativo, la comunicazione tra membri del team, l'interoperabilità e la capacità di gestione di tempi e budget assegnati.

I primi modelli di processo mostravano molti difetti e diventavano presto inadatti e obsoleti. Riscontravano difficoltà nell'analisi dei requisiti e nella comunicazione con il cliente, in quanto il modello era rigido e pianificato e non permetteva quindi la ridefinizione dei requisiti durante le fasi successive dello sviluppo. Tuttavia i clienti non sempre riuscivano a spiegare correttamente agli sviluppatori le caratteristiche desiderate nel prodotto finale, perciò il congelamento dei requisiti costituiva una grossa difficoltà.

Essendo un modello pianificato, tutti i dettagli dello sviluppo erano fissati fin dall'inizio ed era quindi molto difficile apportare modifiche in corso d'opera ed altrettanto complesso arginare problemi che avrebbero sicuramente portato al fallimento del progetto.

L'organo lavorativo era molto numeroso e ciò impediva una corretta trasparenza sul lavoro e causava una scarsa comunicazione tra i membri del team, che dovrebbe essere la base su cui si fonda l'interoperabilità degli sviluppatori. La mancanza comunicativa portava inevitabilmente ad una pessima gestione del processo di sviluppo in termini di tempo e di budget, con conseguente fallimento del progetto.

Nel corso degli anni tutte queste problematiche sono state mitigate o risolte e al giorno d'oggi si è sviluppato il migliore tra i modelli agili di processo, Scrum.

Scrum è un modello flessibile e non pianificato: i requisiti vengono definiti nella fase iniziale del processo di sviluppo ma possono essere modificati nelle fasi successive. Una migliore ma non ancora perfetta comunicazione con il cliente determina un calo della percentuale di fallimento dei progetti dovuto ad una pessima intesa con il cliente stesso.

L'organo lavorativo è ridotto, per facilitare la comunicazione tra membri del team, che possono contare su trasparenza (ogni aspetto del processo deve essere conosciuto da ciascun membro coinvolto nel progetto), controllo (tutti gli aspetti del processo devono essere controllati con elevata frequenza) e adattamento (gli aspetti del processo in ranghi inaccettabili devono essere aggiustati e riadattati).

Per sottolineare maggiormente l'importanza della comunicazione, Scrum introduce un meeting giornaliero nel quale ogni team espone cosa ha concluso dal meeting precedente, quali parti saranno sviluppate prima del meeting successivo e quali ostacoli hanno riscontrato.

Questa forte comunicazione porta quindi ad un notevole miglioramento nella gestione di tempo e budget assegnati al progetto.

Grazie a molti sondaggi effettuati in Italia e in 80 nazioni diverse, possiamo affermare che Scrum sia il migliore modello di processo al giorno d'oggi, e soprattutto il più adottato.

A sostegno di questa tesi è stato fatto uno studio sull'esperienza di tre team della Microsoft, che hanno lavorato allo sviluppo software basandosi strettamente su Scrum.

I risultati indicano che, rispetto a quando lavoravano seguendo altri modelli, i team sono riusciti a migliorare qualità, produttività e gestione del processo di sviluppo: il tempo impiegato è stato notevolmente ridotto, il budget assegnato non è stato superato e l'intero progetto è stato portato a termine con tutti i requisiti richiesti. I risultati ottenuti dai tre team sono stati pienamente soddisfacenti e mostrano un incremento del 90% della produttività, sempre in accordo con i criteri di qualità della Microsoft.

Questo studio mostra un esempio particolare di come la produttività sia incrementata dai primi modelli di processo fino a Scrum, sottolineando che Scrum sia attualmente il modello più adottato e quindi ritenuto il migliore.

RINGRAZIAMENTI

Ringraziare non è sempre facile, soprattutto quando un semplice grazie non può pagare tutto l'aiuto e il sostegno ricevuto.

Un grazie speciale al mio relatore, il prof Paolo Ciancarini, per avermi saputo ascoltare con pazienza ed attenzione e per avermi guidata in maniera impeccabile nella conclusione del mio percorso.

Grazie ai miei genitori, per tutto il sostegno che mi hanno dato, per la fiducia che hanno saputo trasmettermi, per il prezioso aiuto quando ero in difficoltà e per avermi insegnato a credere in me stessa e a non mollare mai.

Grazie a mio fratello, per avermi aiutata a seguire il perfetto esempio che è, per tutte le risate e i disastri combinati insieme che hanno trasformato tutti i momenti tristi in attimi di serenità.

Grazie alle mie amiche, per esserci sempre nel momento del bisogno, per le confidenze, per le risate fino ad avere le lacrime, per la nostra complicità, perché riescono sempre a strapparmi un sorriso cancellando tutte le mie preoccupazioni in un istante.

Grazie davvero di cuore a tutti voi, per essermi sempre stati accanto e soprattutto per avermi sopportata nei periodi negativi, e so che non è facile!

BIBLIOGRAFIA

Libri

[GJM04] Ghezzi C., Jazayeri M., Mandrioli D., *Ingegneria del software: fondamenti e principi*, 2, Italia, Pearson Education Italia, 2004.

[Hig03] Highsmith J.A., *Agile software development ecosystem,*Boston, Addison Wesley Professional, 2003.

[MoeLen04] Moeller T., Lenz G., *NET: a complete development cycle*, Boston, Addison Wesley Professional, 2004.

[Lar04] Larman C., *Agile and iterative development: a manager's guide,* Boston, Addison Wesley Professional, 2004.

[Sch04] Schwaber K., *Agile project management with Scrum,* Redmond, Microsoft Press, 2004.

[Som07] Sommerville I., *Ingegneria del software*, 8, Italia, Pearson Education Italia, 2007.

[Ver13] Verheyen G., *Scrum: a pocket guide*, , Van Haren Publishing, Zaltbommel, 2013.

Articoli

[Boe02] Boehm B., "Get ready for agile methods with care", *Computer*, 35(1), 2002, 64-69.

[Boe88] Boehm B., "A spiral model of software development and enhancement", *Computer*, 21(5), 1988, 61-72.

[BMNW11] Brown G., Meltzer A., Nagappan N., Williams L., "Scrum + Engineering Practices: Experiences of Three Microsoft Teams", *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*, 1, 2011, 463-471.

[BoeTur03] Boehm B., Turner R., "Using risk to balance agile and plan-driven methods", *Computer*, 36(6), 2003, 57-66.

[Cho08] Cho J., "Issue and challenges of agile software development with Scrum", *Issues in Information System*, 9(2), 2008, 188-195.

[Cus97] Cusumano M., "How Microsoft builds software", *Communications of the ACM*, 40(6), 1997, 53-61.

[Fra10] Frank Cervone H., "Understanding agile project management methods using Scrum", *OCLC Systems & Services*, 27(1), 2010, 18-22.

[NonTak86] Nonaka I., Takeuchi H., "The new new product development game", *Harvard Business Review*, 45(6), 1986, 23-33.

SITOGRAFIA

[BBBCCFGHHJKMMMSST01] Beck K., Beedle M., Bennekum A., Cockburn A., Cunningham W., Fowler M., Grenning J., Highsmith J., Hunt A., Jeffries R., Kern J., Marick B., Martin R., Mellor S., Schwaber K., Sutherland J., Thomas D., "Manifesto Agile", 2001, http://agilemanifesto.org/iso/it/, 24 settembre 2001.

[Mar09] Marchesi M., "Metodologie Agili per lo sviluppo: ormai inarrestabili e dappertutto!", 2009, http://win.itware.com/toolnews/toolnews.asp?id=ago-set2009StatoArte1, 18 luglio 2009.

[SchSut11] Schwaber K., Sutherland J., "The Scrum guide", 2011, https://www.scrum.org/Scrum-Guide, 11 luglio 2013.

[Sut12] Sutherland J., "The Scrum Papers: Nut, Bolts and Origins of an Agile Framework", 2012, http://jeffsutherland.com/ScrumPapers.pdf, 2 aprile 2012.

[Sut13] Sutherland J., "Principi e valori di Agile di Jeff Sutherland", 2013, http://msdn.microsoft.com/it-it/library/dd997578.aspx, 1 giugno 2013.

[Web1] http://ciamberlini.it/wp-content/uploads/2013/02/codefix.png

[Web2] http://www.atcetera.it/wp-content/uploads/2013/01/iterative-development1-e1357660310793.png

[Web3] http://users.dickinson.edu/~wahlst/491/lectures/lec02/process/spiral.jpg

[Web4] http://www.lynnecazaly.com.au/storage/agile%20manifesto%20screenshot.jpg?__SQUARESPACE_CACHEVERSION=1394163978554

[Web5]
http://upload.wikimedia.org/wikipedia/commons/0/05/SampleBurndownChart.png

[Web6] http://redletterday.ch/wp-content/uploads/2014/05/scrum_process_big.jpg