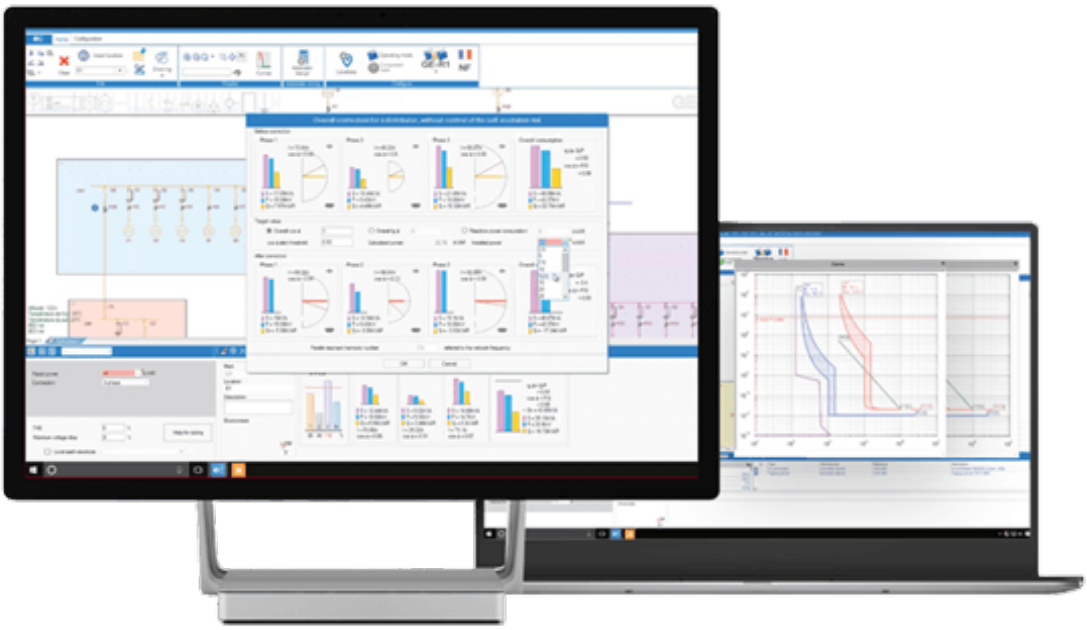
Dispense di Ingegneria del software

Emanuele Izzo, 0253052



Corso di laurea triennale Informatica
Università Tor Vergata, Facoltà di Scienze MM.FF.NN.
31/08/2020

Documento realizzato in LaTeX

Indice

Parte I

Introduzione

Capitolo 1

Il Software

1.1 Cos'è il software

La parola "software" racchiude un insieme di definizioni:

- Programmi;
- Documentazione relativa;
- Dati di configurazione per una corretta esecuzione dei programmi stessi.

Esistono due tipi fondamentali di software:

- *Generico*: sviluppato per essere utile ad un ampio insieme di persone;
- *Customizzato*: sviluppato su richiesta di uno specifico cliente con esigenze ben precise.

1.2 Cos'è un processo software

Un processo software è un concetto ingegneristico, che è stato tradotto in ciclo di vita del software. Anche se i prodotti software sono molto diversi tra loro, esistono delle fasi nel ciclo di vita del software che sono comuni a tutti:

- *Specifica*;
- *Sviluppo*;
- *Convalida*;
- *Evoluzione* (mantenimento del progetto).

1.3 Cos'è un modello di processo software

I modelli rappresentano i vari approcci al ciclo di vita del software (Alcuni esempi di modelli sono il *workflow*, che analizza le attività che vengono svolte, il *dataflow*, che analizza come vengono elaborati i dati, e il *ruolo=azione*, che stabilisce che deve svolgere i diversi compiti). La maggior parte dei modelli di processo si basa su uno dei seguenti modelli generici, o paradigmi, dello sviluppo del software:

- *Waterfall*: approccio a cascata che considera ogni fase separata da quella successiva;
- *Sviluppo ciclico*: unisce tutte le fasi, definendo inizialmente specifiche molto vaghe;
- *Ingegneria del software basata sui componenti*: che presuppone esaistano già alcune parti del sistema.

1.4 Quali sono i costi dell'ingegneria del software

Non c'è una risposta precisa, ma si possono comunque dare delle stime: il 60% sono costi di sviluppo e il 40% sono costi di testing e mantenimento. Quando il software è custom, cioè realizzato ad hoc per un cliente specifico, tipicamente l'adattamento nel tempo (evolution cost) è maggiore ai costi di sviluppo iniziali.

1.5 Cos'è un metodo di ingegneria del software

È un approccio strutturato allo sviluppo del software che facilita la produzione di alta qualità e a costi contenuti. Ne esistono vari:

- *Metodi orientati alle funzioni*, che identificano i componenti funzionali di base;
- *Metodi orientati agli oggetti* (tuttora usati).

Ora tutti i metodi sono stati integrati in un approccio unificato costruito attorno a UML (linguaggio di modellazione unificato)

1.6 Cos'è CASE

L'acronimo CASE significa Computer Aided Software Engineering. Sono gli strumenti informatici che forniscono supporto alle varie fasi del ciclo di vita del software. Per tutti i metodi è disponibile una tecnologia CASE. Solitamente gli strumenti CASE si dividono in due tipi:

- *Upper case*: per gestire le fasi iniziali, come la modellazione.
- *Lower case*: per gestire le fasi successive, come lo sviluppo e il testing.

1.7 Quali sono le caratteristiche di un buon software?

Le caratteristiche di un software valido come prodotto sul mercato sono principalmente:

- *Maintanability*;
- *Dependability*;
- *Efficiency*;
- *Acceptability*.

1.8 La gestione dei progetti software (Software Project Management)

La gestione di un progetto software implica la pianificazione, il monitoraggio ed il controllo di persone, processi ed eventi durante lo sviluppo del prodotto. Il Software Project Management Plan (SPMP) è il documneto che guida la gestione di un progetto software.

1.8.1 Struttura dell'SPMP

La struttura dell'SPMP prevede una Title Page di presentazione del progetto, una Lead Shett contenente le informazioni relative al progetto e una Table of Contents; la struttura interna del file è la seguente

1. Introduction
    - 1.1 Purpose
    - 1.2 Background
    - 1.3 Organization and Responsibilities
        - 1.3.1 Project Personnel
        - 1.3.2 Interfacing Groups
2. Statement of Problem
3. Technical Approach
    - 3.1 Reuse Strategy
    - 3.2 Assumptions and Constraints
    - 3.3 Anticipated and Unresolved Problems
    - 3.4 Development Environment
    - 3.5 Activities, Tools and Products
    - 3.6 Build Strategy
4. Management Approach
    - 4.1 Assumptions and Constraints
    - 4.2 Resource Requirements
    - 4.3 Milestones and Schedules
    - 4.4 Metrics
    - 4.5 Risk Management
5. Product Assurance
    - 5.1 Assumptions and Constraints
    - 5.2 Quality Assurance (QA)
    - 5.3 Configuration Management (CM)
6. References
7. Plan Update History

1.9 Le quattro P

La gestione efficace di un progetto software si fonda sulle quattro P:

- *Processo*;
- *Persone*;
- *Prodotto*;
- *Progetto*.

Capitolo 2

Sistemi socio-tecnici

> *Sistema: collezione significativa di componenti interrelati che lavorano assieme per realizzare un determinato obiettivo.*

I sistemi che includono software ricadono in due categorie:

- *Sistemi tecnico-informatici*: cioè sistemi automatizzati che includono hardware e software e dove gli operatori non sono considerati parte del sistema, e la conoscenza dello scopo per il quale le organizzazioni usano sistemi di questo tipo non fa parte del sistema stesso;
- *Sistemi socio-tecnici*: la conoscenza dello scopo del sistema fa parte del sistema stesso, e le persone fanno parte del sistema, ovvero prendono decisioni importanti e non solo all'interfaccia.

Le caratteristiche dei sistemi socio-tecnici sono tre:

- *Proprietà emergenti*: sono proprietà del sistema intero che dipendono sia dai componenti del sistema, sia dalle loro relazioni.
- *Non determinismo*;
- *Rapporti con gli obiettivi organizzativi*;

2.1 Proprietà emergenti di un sistema

Le proprietà emergenti di un sistema non possono essere attribuite a nessuna parte specifica del sistema, ma emergono solo quando i componenti del sistema sono stati integrati. Esempi di proprietà emergenti sono:

- *Reliability*;
- *Security*;
- *Repairability*;
- *Usability*.

Ci sono due tipi di proprietà emergenti:

- *Proprietà funzionali*: rispondono alla domanda "Cosa offre il sistema?";
- *Proprietà non funzionali*: rispondono alla domanda "Come funziona il sistema?".

2.2 Problematiche dell'ingegneria della reliability

L'ingegneria della reliability si occupa di definire tecniche per far fronte alle problematiche che si potrebbero avere ne sistema. In un sistema si hanno due possibili errori, fault e failure, così definiti e differenziati:

- *Fault*: malfunzionamento locale che può essere bloccato;
- *Failure*: fallimento nel non garantire una funzionalità del sistema.

Le influenze sulla reliability sono:

- *Affidabilità hardware*;
- *Affidabilità software*;
- *Affidabilità dell'operatore.*

Queste tre influenze sono strettamente correlate tra loro.

2.3 Processo di ingegneria dei sistemi

Contiene le metodologie di progettazione del rpocesso di produzione del sistema, ed è così composto:

- *Definizione dei requisiti di sistema*;
- *Progettazione del sistema*;
- *Sviluppo dei sottosistemi*;
- *Integrazione del sistema*;
- *Installazione del sistema*;
- *Evoluzione del sistema*;
- *Rimozione del sistema.*

2.3.1 Definizione dei requisiti di sistema

Significa cosa il sistema dovrebbe fare e le proprietà essenziali desiderate. Questa fase si concentra sulla creazione di tre tipi di requisiti:

- *Requisiti funzionali astratti*: funzionalità definite a livello astratto;
- *Proprietà del sistema*: requisiti non funzionali definiti in generale per il sistema;
- *Caratteeristiche che il sistema non deve avere.*

Una parte importante della fase di definizione dei requisiti è stabilire l'insieme degli obbiettivi generali che il sistema deve raggiungere. Gli obiettivi possono essere di due tipi:

- *Funzionali*;
- *Organizzativi.*

2.3.2 Progettazione del sistema

Definisce in quale modo i componenti devono fornire le funzionalità del sistema. Il processo di pregettazione del sistema è suddiviso in fasi:

- *Suddivisione dei requisiti*;
- *Identificazione dei sottosistemi*:
- *Assegnazione dei requisiti ai sottosistemi*;
- *Specifica delle funzionalità del sottosistema*;
- *Definizione delle interfacce dei sottosistemi.*

I requisiti sono di due tipi:

- *Immodificabili*, ovvero quelli che caratterizzano lo scopo del sistema, quelli che sono l'obiettivo del sistema;
- *Modificabili*, ovvero quelli che servono per raggiungere l'obiettivo, ma non sono quelli cruciali.

2.3.3 Modellazione del sistema

Un modello è la rappresentazione astratta della struttura del sistema. Esso rappresenta i componenti che ormano il sistema stesso. Il modello specifica anche il legame tra i vari sottosistemi.

2.3.4 Sviluppo dei sottosistemi

è la fase in cui si sviluppa quel che si è stabilito nella fase di system design.

2.3.5 Integrazione dei sistemi

Durante il rpocesso di integrazione dei sistemi si prendono i sottoprocessi sviluppati individualmente e si assemblano per formare il sistema completo.

2.3.6 Evoluzione del sistema

Alcuni sistemi grandi e complessi hanno un ciclo di vita molto lungo durante il quale vengono aggiornati per correggere errori ed implementare nuovi requisiti. L'evoluzione è intrinsecamente costosa per molte ragioni:

- Ogni modifica deve esseere valutata molto attentamente riguardo le prospettive aziendali e tecniche;
- Siccome i sottosistemi non sono indipendenti, cambiare un sottosistema potrebbe portare ad effetti negativi sugli altri che potrebbero dover essere cambiati a loro volta;
- È necessario capire nuovamente le motivazioni che hanno portato alle scelte di design fatte in precedenza;
- Con l'invecchiare del sistema la struttura originale si modfica a causa dei cambiamenti rendendo ulteriori cambiamenti sempre più costosi.

2.3.7 Rimozione del sistema

Quando un sistema diventa inutile viene disattivato.

2.4 Organizzazioni, persone e sistemi informatici

I sistemi socio-tecnici sono sistemi che devono perseguire qun qualche obiettivo aziendale, ed essendo integrati in un ambiente aziendale devono tener conto delle procedure e regole presenti. I fattori che influenzano il design del sistema sono:

- *Processo change*: se il sistema richiede cambiamenti nella lavorazione, sarà necessario una formazione degli utenti;
- *Job change*: se il sistema richiede agli utenti di cambiare modo di lavorare, questi potrebbero resistere attivamente all'introduzione di un nuovo sistema;
- *Organizational change*: il sistema potrebbe cambiare la struttura del potere politico di un'azienda.

2.5 Sistemi legacy

I sistemi ereditati (sistemi legacy) sono sistemi informatici socio-tecnici sviluppati in passato, spesso utilizzando tecnologie vecchie e obsolete. Questi sistemi includono software, hardware e anche procedure, lavorazioni vecchi modi di fare. I sistemi ereditati sono spesso dei sistemi critici, cioè mantenuti perché è troppo rischioso sostituirli. I componenti principali dei sistemi ereditati sono:

- *Hardware*;
- *Software d'appoggio*;
- *Software applicativo*;
- *dati delle applicazioni*;
- *Processi aziendali*;
- *politiche e regole aziendali.*

Capitolo 3

I Sistemi critici

Sono sistemi tecnici o socio-tecnici in cui il fallimento può portare a gravi conseguenze economiche e danni fisici a persone. Si parla dunque di criticità nei sistemi quando un aspetto, se trascurato, può avere conseguenze gravi. Ci sono tre tipi principali di sistemi critici:

- *Safety-critical*: i cui fallimenti possono provocare danni alle persone o all'ambiente;
- *Mission-critical*: i cui malfunzionamenti possono causare il fallimento di alcune attività a obiettivi diretti;
- *Business-critical*: i caui fallimenti possono portare a costi molto alti per le aziende che li usano.

La nozione che risponde al concetto di criticità nei sistemi è quella di *dependability*, e i sistemi *non-dependable* sono spesso rifiutati dagli utenti in quanto poco sicuri. Il problema della criticità si può presentare a livelli diversi in un sistema:

- *Livello hardware*;
- *Livello software*;
- *Livello umano.*

3.1 Dependability di un sistema

La dependability è una proprietà che si equipara alla fiducia che l'utente ripone nel sistema pensando cge questo operi secondo le aspettative e che non "fallisca" nel normale utilizzo. Ci sono quattro principlai dimensioni della fidatezza:

- *Availability*: disponibilità continua;
- *Reliability*: correttezza funzionale;
- *Safety*: sicurezza fisica;
- *Security*: sicurezza informatica.

Ci sono altre caratteristiche del sistema che devono essere considerate:

- *Repairability*: capacità del sistema di essere riparato;

- *Mantainability*: capacità di poter mantenere il sistema nel tempo, mantenerlo aggiornato rispetto alle specifiche;
- *Survivability*: capacità del sistema di offrire servizi ad un livello anche minimo in presenza di attacchi esterni;
- *Error tolerance*: capacità del sisstema di evitare e tollerare gli errori di immissione da parte dell'utente.

3.1.1 Dependability VS Performance

Come performance si intende il grado di efficienza del sistema nell'uso delle risorse disponibili. È la capacità del sistema di rispecchiare dei parametri di qualità di un determinato servizio. Le performance di un sistema sono strettamente legato al grado di dependability del sistema stesso.

3.1.2 Availability VS Reliability

Sono due concetti diversi ma strettamente collegati. Entrambi possono essere espressi come probabilità numeriche:

- *Availability*: probabilità che il sistema sia attivo e funzionale e in grado di fornire i servizi agli utenti quando li richiedono;
- *Reliability*: probabilità che i servizi siano correttamente forniti come specificato.

Terminologia della Reliability (correttezza funzionale)

Quando si discute di reliability è utile distinguere i termini difetto, errore e fallimento:

- *Fallimento del sistema (system failure)*: non viene fornito il servizio come gli utenti del sistema si aspettano;
- *Errore (system error)*: stato errone del sistema che può portare a comportamenti inaspettati;
- *Difetto (system fault)*: caratteristica del sistema software che può portare ad un errore, malfunzionamento dato da un guasto.

Come ottenere reliability

- *Evitare i difetti*;
- *Ricerca e rimozione dei difetti*;
- *Tolleranza dei difetti.*

Reliability modeling

La reliability modeling può essere intesa come la probabilità che un input non appartenga all'insieme di input che causano comportamenti

3.2 Safety

Intesa come sicurezza fisica, ovvero il sistema non deve mai danneggiare persone o ambiente, neanche in caso di fallimento. I sistemi odierni sono molto complessi e non possono essere controllati solo dall'hardware. Ecco perché si parla spesso di software a sicurezza critica. Tali software si dividono in due categorie:

- *Software primario a sicurezza critica*: software integrato il cui malfunzionamento può provocare malfunzionamenti hardware che a loro volta possono provocare danni umani o ambientali;
- *Software secondario a sicurezza critica*: software che può provocare indirettamente danni.

Da notare che safety e reliability sono strettamente collegate. Infatti, per esempio, un sistema con un'alta fault tolerance può portare il sistema a funzionale in modo anomalo e avere un comportamento che causa incidenti. Esistono molte altre ragioni per le quali i sistemi reliable non sono necessariamente e contemporaneamente safety:

- *Specifiche incomplete*;
- *Malfunzionamenti hardware*;
- *Generazione*, da parte degli utenti, di input errati (o non) in momenti sbagliati.

3.2.1 Terminologia della safety

- *Incidente (accident)*: evento o sequenza di eventi non pianificati che possono portare a danni fisici o ambientali;
- *Pericolo (hazard)*: condizione che, potenzialmente, contribuisce o causa un incidente;
- *Danno (damage)*: misura della perdita risultante da un incidente;
- *Gravità del pericolo (hazard sverity)*: stima del danno peggiore che può risultare dall'avvenimento di un pericolo;
- *Probabilità del pericolo (hazard probability)*: probabilità della comparsa di eventi che possono portare ad un pericolo;
- *Rischio (risk)*: probabilità che il sistema causi un incidente.

3.2.2 Come ottenere safety

- *Evitare i pericoli*;
- *Ricerca e rimozione dei pericoli*;
- *Limitare i danni*.

3.3 Security

Intesa come la capacità del sistema di proteggere se stesso da attacchi esterni, di natura accidentale o maliziosa. Ci sono tre tipi di danni che possono essere causati da attacchi esterni:

- *Negazione del servizio (denial of service)*;
- *Corruzione dei dati o dei programmi*;
- *Rivelazione di informazioni riservate.*

3.3.1 Terminologia della security

- *Exposure (esposizione)*: possibile perdita o danno in un sistema informatico;
- *Vulnerability (vulnerabilità)*: debolezza di un sistema informatico che può essere sfruttata per causare perdite o danni;
- *Attack (attacco)*: sfruttamento di una vulnerabilità del sistema;
- *Threads (minacce)*: circostanze che possono portare a danni o perdite (è la vulnerabilità di un sistema sotto attacco);
- *Control (controllo)*: misura di protezione che riduce le vulnerabilità di un sistema.

3.3.2 Come ottenere security

- *Evitare le vulnerabilità*;
- *Rilevamento e neutralizzazione degli attacchi*;
- *Limitazione delle esposizioni.*

Parte II

Processo

Capitolo 4

Processi Software

Il processo software (o ciclo di vita del software) è un insieme di attività che porta alla creazione di un prodotto software, ed è definibile come un intervallo di tempo che incorre tra l'istante in cui nasce l'esigenza di costruire un prodotto software e l'istante in cui il prodotto viene dismesso. Le attività comuni a tutti i processi software sono:

- *Specifiche del software*;
- *Progettazione ed implementazione del software*;
- *Convalida dal software*;
- *Evoluzione del software.*

4.1 Fasi del processo

Il processo software segue un ciclo di vita che si articola in tre stadi:

- *Sviluppo*;
- *Manutenzione*;
- *Dismissione.*

Nel primo stadio si possono riconoscere due tipi di fasi:

- La fase di tipo *definizione*, che si occupano di "cosa" il software deve fornire;
- la fase di tipo *produzione*, che definiscono "come" realizzare quanto ottenuto con le fasi di definizione.

Lo stadio di manutenzione è a supporto del software realizzato e prevede fasi di definizione e/o produzione al suo interno. I tipi di manutenzione che si possono effettuare sono:

- Manutenzione correttiva;
- Manutenzione adattiva;
- Manutenzione perfettiva;
- Manutenzione preventiva.

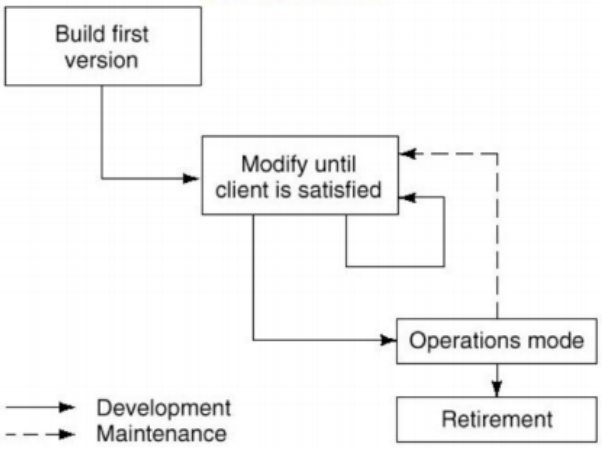
Durante ogni fase si procede ad effettuare il testing di quanto prodotto, mediante opportune tecniche di verifica e validazione (V&V).

4.2 Modelli del processo software

Un modello è la rappresentazione astratta del processo software, e specifica sia la serie di fasi attraverso cui il prodotto software progredisce, sia l'ordine con cui vanno eseguite, dalla definizione dei requisiti alla dismissione. Ogni modello si distingue dagli altri per la visione del ciclo di vita da un punto di vista diverso (diversa prospettiva). I principali modelli generici sono:

- *Modello a cascata*;
- *Modello evolutivo*;
- *Modello basato su componenti.*

L'assenza di un modello del ciclo di vita corrisponde ad una modalità di sviluppo detta "Build & Fix", in cui il prodotto software viene sviluppato e successivamente rilavorato fino a soddisfare la necessità del cliente.



4.2.1 Modello a cascata

Viene usato soprattutto nei sistemi critici, poiché una volta che si passa alla fase successiva, non si può tornare indietro. Tornare indietro può portare a gravi conseguenze: perdita eccessiva di tempo, perdita economica fallimento. La logica del progetto, quindi, dev'essere chiara fin dall'inizio. I principali stadi sono:

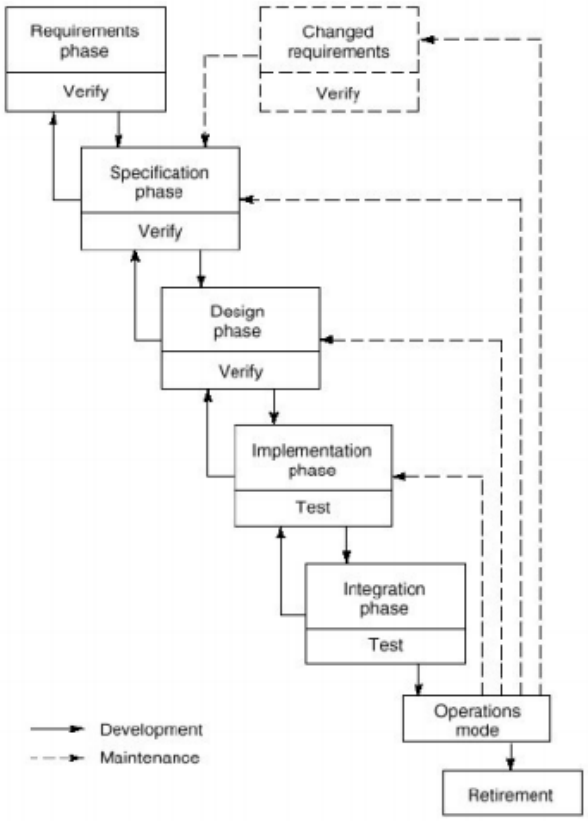
- *Analisi e definizione dei requisiti*;
- *Progettazione del sistema e del software*;
- *Implementazione e testing del sistema*;
- *Integrazione e testing del sistema*;
- *Operatività e manutenzione.*

Il risultato di ogni fase è costituito da uno o più documenti approvati: la fase successiva non parte finchè non è finita la precedente. I vantaggi sono:

- Ogni fase produce documentazione;

- Si allinea agli altri modelli di processi ingegneristici.

Lo svantaggio è che c'è una rigida suddivisione del progetto in stati distinti.



4.2.2 Modello evolutivo

Basato sull'idea di sviluppare un'implementazione iniziale, esporla agli utenti e perfezionarla attraverso molte versioni affinché non si ottiene il sistema voluto dal cliente. Viene usato principalmente per aiutare i clienti e gli sviluppatori a capire i requisiti del software. Ci sono due tipi di sviluppo evoluzionistico:

- *Lo sviluppo esplorativo*: il processo di "messa a fuoco" si fa con il cliente partendo da un modello di software molto semplificativo per far capire bene all'utente la fattibilità;
- *Prototipo usa e getta (throw-away)*: bisogna essere coscienti sin dall'inizio che i prototipi verranno buttati e non potranno mai costruire una base di costruzione del software.

I vantaggi sono:

- I servizi mancanti possono essere rilevati, e possono essere identificati servizi confusi;
- Il sistema prodotto soddisfa le immediate necessità del cliente, fornendo quindi un sistema funzionante all'inizio del processo;
- Le specifiche si possono sviluppare in modo incrementale;

- In questo modo gli utenti comprendono meglio le problematiche, permettendo anche la formazione degli utenti e i test sui prodotti, e le conseguenze si riflettono sul software.

Gli svantaggi sono:

- Il processo ha meno visibilità: in ogni momento posso mettere in discussione ogni fase del ciclo di vita del software;
- Dal momento che il sistema è sviluppato velocemente, non è economico produrre documentazione per ogni versione del sistema;
- Non avendo una chiara progettazione il rischio che si corre è quello di ottenere un sistema mal strutturato.

i campi di applicabilità sono:

- Sistemi di piccola e media grandezza;
- Piccole parti di grandi sistemi;
- Sistemi con vita breve.

4.2.3 Prototipazione da buttare via (throw-away)

Si basa sulla realizzazione di prototipo, ossia implementazioni pratiche del prodotto, realizzate per aiutare a scoprire i problemi dei requisiti, per poi essere scartati. Il prodotto è poi sviluppato utilizzando qualche altro processo di sviluppo. Questo metodo viene usato per ridurre i rischi dei requisiti, ed ogni prototipo viene sviluppato da un iniziale requisito. Il prototipo throw-away non può essere considerato come un prodotto finale, infatti:

- Alcune caratteristiche potrebbero essere state tralasciate;
- Non ci sono specifiche per la manutenzione a lungo termine;
- Il prodotto sarà scarsamente strutturato e difficile da mantenere.

I punti chiave della prototipazione sono:

- Un prototipo può essere utilizzato per dare agli utenti finali un'impressione concreta delle capacità del prodotto;
- La prototipazione sta diventando sempre più utilizzata per lo sviluppo di prodotti in cui il rapido sviluppo è essenziale.
- La prototipazione da buttare via viene utilizzata per comprendere i requisiti del prodotto.

4.2.4 Ingegneria del software basata sui componenti

È basato sul concetto del riuso sistematico di comportamenti nei sistemi integrati. I componenti spesso sono già esistenti e quindi riprogettarli dall'inizio risulta uno spreco di tempo e soldi. I principali stadi sono:

- *Analisi dei componenti*;
- *Modifica dei requisiti*;
- *Progettazione con riutilizzo*;
- *Sviluppo e integrazione*.

I vantaggi sono:

- Riduzione delle quantità di software da sviluppare, riducendo costi e rischi;
- Consegne più veloci

Spesso, però, per riutilizzare il maggior numero di componenti esistenti si rischia di dover scendere a compromessi nei requisiti.

4.3 Process iteration (cicli di processo)

Esistono due modelli di processo che sono stati progettati appositamente per supportare i cicli di processo:

- *Consegna incrementale*;
- *Sviluppo a spirale*.

4.3.1 La consegna incrementale

È applicabile ai sistemi che possono essere partizionati. Le specifiche, la progettazione e l'implementazione del software sono suddivisi in una serie di incrementi sviluppati uno alla volta; il prodotto viene sviluppato e consegnato in incrementi dopo aver stabilito un'architettura generale. I vantaggi sono:

- I clienti non devono aspettare finché il sistema completo sia consegnato prima di usufruirne;
- I clienti possono utilizzare i primi incrementi come prototipi e acquisire esperienza per migliorare i requisiti degli incrementi seguenti;
- Il rischio di fallimento è minore, perché il cliente testa man mano ogni incremento;
- I servizi con priorità più alta che vengono rilasciati con i primi incrementi verranno scansionati più volte e quindi i clienti hanno meno probabilità di incontrare fallimenti nel software nelle parti più importanti.

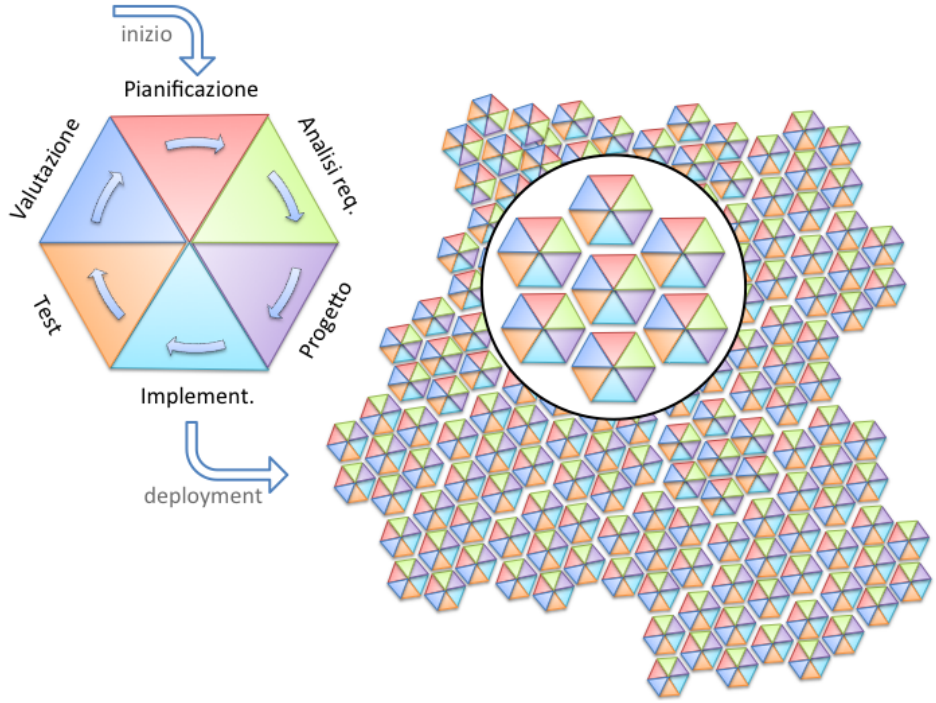
Gli svantaggi sono:

- Gli incrementi devono essere relativamente piccoli;

- Ogni incremento dovrebbe aggiungere funzionalità al sistema e può essere difficile predisporre i requisiti del cliente in incrementi della misura giusta;
- Alcune funzionalità base potrebbero essere utili in più parti del sistema, quindi finché queste parti non vengono sviluppate, è impossibile testare le funzionalità base.

Questo modello può essere realizzato in due versioni differenti:

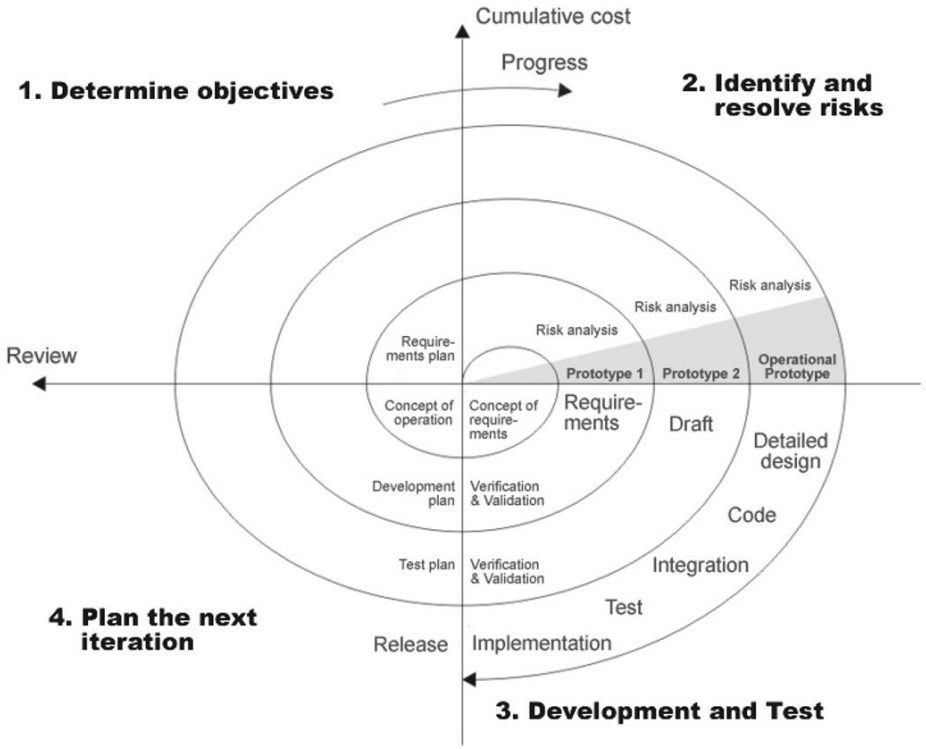
- Versione con overall architecture;
- Versione senza overall architecture (più rischiosa).



4.3.2 Sviluppo a spirale

I vari cicli della spirale sono un'approssimazione del sistema e si usa molto nella prototipizzazione throw-away. Il processo software viene rappresentato come una spirale in cui ogni giro corrisponde as una fase del processo. Ogni giro della spirale è diviso in quattro settori:

- *Determinazione degli obiettivi*;
- *Valutazione e riduzione del rischio*;
- *Sviluppo e convalida*;
- *Pianificazione.*



4.4 Attività del ciclo di vita del softawre

4.4.1 Specifiche del software (o ingegneria dei requisiti)

Processo per capire quali servizi sono richiesti dal sistema, e per identificare i vincoli all'operabilità e allo sviluppo. Le fasi principali sono:

- *Studio di fattibilità*;
- *Scoperta e analisi dei requisiti*;
- *Specifica dei requisiti*;
- *Convalida dei requisiti.*

4.4.2 Progettazione e implementazione del software

Processo di conversione delle specifiche in un sistema eseguibile. Il processo di progettazione è suddiviso nelle seguenti fasi:

- *Progettazione architetturale*;
- *Specifiche astratte*;
- *Progettazione dell'interfaccia*;
- *Progettazione dei componenti*;
- *Progettazione delle strutture dati*;

- *Progettazione degli algoritmi.*

Per quanto riguarda il lato dell'implementazione, tutto dipende dai programmatore che la attuano. Alcuni cominciano con l'implementare i componenti che conoscono meglio muovendosi verso quelle che comprendono meno; altri procedono in senso opposto. Tutti, comunque, alla fine eseguono il *processo di testing*, che verifica la presenza degli errori, e il *processo di debug*, che si occupa di localizzare gli errori e di correggerli.

4.4.3 Convalida del software

Serve a mostrare che il sistema è conforme alle sue specifiche e che soddisfa le aspettative del cliente. Il sistema, se è grande, viene testato in più fasi:

- *Test dei componenti*;
- *Test del sistema*;
- *Test di accettabilità.*

4.4.4 Evoluzione del software

È il processo di mantenimento del sistema nel tempo

4.5 RUP e XP

Nel tempo un nuovo tipo di metodologie di sviluppo software, le cosiddette metodologie "agili" (lightweight), in contrapposizione a quelle cosiddette "pesanti" (heavyweight):

- *Metodologie pesanti*, in cui le fasi del ciclo di vita del software sono eseguite sequenzialmente;
- *Metodologie agili*, le quali non cercano di prevedere come evolverà il sistema software, ma sono adattive, ovvero propongono valori e pratiche per meglio adattarsi all'evoluzione dei requisiti dell'utente, prima che del sistema software.

4.5.1 RUP

Il RUP è una metodologia di processo iterativo: ogni iterazione è un ciclo di sviluppo che porta al rilascio di una parte del prodotto finale. Le fasi che compongono un'iterazione sono:

- *Avvio (inception)*: studio di fattibilità;
- *Elaborazione (elaboration)*: viene sviluppata una struttura architetturale del sistema;
- *Costruzione (construction)*: vengono raffinati i requisiti tecnici e si procede con l'implementazione sulla base dell'architettura corrente.
- *Transizione (transition)*: ci si assicura che il sistema sia usabile.

La ciclicità e l'incremento sono supportati nel RUP in due modi:

- Ogni fase può essere eseguita in modo ciclico con risultati sviluppati in modo incrementale;
- L'intero insieme delle fasi può essere eseguito in modo incrementale.

Il RUP individua tre prospettive:

- *Prospettiva dinamica*, che mostra le fasi del modello nel tempo;
- *Prospettiva statica*, che mostra le attività di processo coinvolte, chiamate *workflow* nella descrizione RUP; i workflow statici nel RUP sono:
  - *Modellazione delle attività aziendali*;
  - *Requisiti*;
  - *Analisi e progettazione*;
  - *Implementazione*;
  - *Test*;
  - *Rilascio*;
  - *Gestione della configurazione e delle modifiche*;
  - *Gestione del progetto*;
  - *Ambiente*.
- *Prospettiva pratica*, che descrive la buona prassi di ingegneria del software che si raccomanda di usare nello sviluppo dei sistemi: le pratiche fondamentali sono sei:
  - *Sviluppo iterativo del software*;
  - *Gestione diretta dei requisiti*;
  - *Progettazione attraverso architetture component-based*;
  - *Modellazione visuale del software*;
  - *Verifica della qualità del software*;
  - *Gestione e controllo dei cambiamenti nel software*.

Il vantaggio è che le fasi del processo di sviluppo non sono associate a specifici workflow. Il RUP non è un processo adatto a tutti i tipi di sviluppo, ma rappresenta una nuova generazione di processi generici.

4.5.2 XP

Metodologia di sviluppo "leggero", basata soprattutto sull'interazione con il cliente. Lo sviluppo del progetto viene diviso in sottoparti con rilasci successivi . Il cliente, in questo modo, vede man mano i singoli gruppi di funzione che vengono rilasciati, anche pronti per essere messi in produzione. Anche l'eXtreme Programming si basa su valori fondamentali:

- *Comunicaione* tra tutti gli attori coinvolti;
- *Feedback* tra i team e il cliente per riuscire a governare i possibili cambiamenti;

- *Semplicità*;
- *Coraggio* nel modificare il sistema.

XP è un processo di sviluppo "adattivo" e "people oriented" che si implementa attraverso una serie di pratiche *(best practices)*:

- *Pratiche di codifica*:
  - *Coding standard*;
  - *Testing*;
  - *Refractoring*.
- *Pratiche di design*:
  - *Simple design*;
  - *Small release*;
  - *Integrazione continua*.
- *Pratiche sociali*:
  - *Pair programming*;
  - *Planning game*;
  - *on-site customer*;
  - *Proprietà collettive del codice*.

4.6 Computer-aided software engineering

Il termine CASE identifica i software usati per fornire supporto alle attività del ciclo di vita come l'analisi dei requisiti, la progettazione, l'implementazione e il testing. Alcuni esempi di attività con strumenti CASE sono:

- Sviluppo di un modello grafico come parte delle specifiche dei requisiti o del progetto software;
- Facilitare la comprensione del progetto attraverso dizionari di parole chiavi che conservano informazioni e relazioni del progetto;
- Creazione dell'interfaccia utente attraverso una GUI creata interattivamente.
- Debug del software;
- traduzione automatica da un linguaggio più vecchio a uno più recente.

4.6.1 Classificazione dei tool CASE

Ci sono differenti metodi di classificazione dei tool CASE, tra cui:

- *Prospettiva funzionare*: tool classificati in base alle loro funzioni specifiche;
- *Prospettiva di processo*: tool classificati in base all'attività di processo che integrano;
- *Prospettiva di integrazione*: tool classificati in base a come sono organizzati in unità integrati che forniscono supporto ad una o più attività del ciclo di vita.

Capitolo 5

Project Management

Questa fase è responsabile solo della pianificazione e dello scheduling del progetto. In tutti i sistemi software ci sono delle scadenze e dei budget da rispettare. Il lavoro del manager è quello di assicurasi che tali vincoli (scadenze e budget) siano rispettati e che venga prodotto software che soddisfi i requisiti. I manager del software svolgono lo stesso tipo di lavoro dei gestori di altri progetti ingegneristici, però l'ingegneria del software si differenzia per diversi motivi, che rendono la gestione più difficile:

- *Il prodotto è intangibile*;
- *Non esistono processi standard* per il software;
- I grandi processi software sono spesso *unici*.

5.1 Attività di gestione

La maggior parte dei gestori ha responsabilità in alcuni stadi delle seguenti attività:

- *Scrittura delle proposte*;
- *Pianificazione e tempistica del progetto*;
- *Costo del progetto*;
- *Monitoraggio e revisione del progetto*;
- *Selezione e valutazione del personale*;
- *Scrittura e presentazione dei report.*

5.2 Pianificare il progetto

Una gestione efficace di un progetto software dipende per intero dalla sua pianificazione aziendale. Il manager deve prevedere eventuali problemi che possono sorgere e deve preparare soluzioni alternative. All'inizio del processo di pianificazione bisogna:

- Stimare i vincoli che influenzano il progetto;
- Stimare i parametri del progetto (struttura, dimensione, ...);

- Definire le milestone e le consegne del progresso.

E infine il processo entra nel ciclo.

5.2.1 Piano del progetto

Stabilisce le risorse disponibili, la suddivisione del lavoro e la tabella di marcia da seguire. È un unico documento, e la struttura del piano del progetto dovrebbe includere le seguenti fasi:

- *Introduzione*;
- *Organizzazione del progetto*;
- *Analisi del rischio*;
- *Requisiti di risorse hardware e software*;
- *Dimensione del lavoro*;
- *Tempistica del progetto*;
- *Meccanismi di monitoraggio e supporto.*

5.2.2 Milestone e consegne

Quando si pianifica un progetto dovrebbero essere stabilite delle milestone, cioè dei punti cardine riconoscibili nell'attività di processo software. Ad ogni milestone dovrebbe esserci un output formale, un report per descrivere cosa è stato completato. Una consegna è un risultato del progetto associato alla fine di alcune fasi, come specifiche o progettazione. Quindi una consegna potrebbe corrispondere ad una milestone, ma il viceversa non è sempre vero.

5.3 Tempistica del progetto

La tempistica richiede la suddivisione del progetto in attività separate. Le attività del progetto dovrebbero durare almeno una settimana, ma vengono solitamente impostati dei range di tempo più ampi per evitare di trovarsi fuori tempo rispetto alle aspettative. Le tempistiche del progetto sono solitamente rappresentate con un insieme di grafici che mostrano la divisione del lavoro e le dipendenze tra le attività e l'assegnazione dello staff.

5.3.1 Grafici a barre e reti di attività

I *grafici a barre* mostrano chi è il responsabile e quando sono previsti la partenza e il completamento di ogni attività. Le *reti* mostrano le dipendenze tra le diverse attività del progetto. Nelle reti di attività i rettangoli rappresentano le attività, i rettangoli con gli angoli arrotondati rappresentano le milestone e le consegne. Tutte le attività devono finire in milestone e un'attività può partire solo quando la sua milestone precedente è stata completata.

5.4 Gestione del rischio

Il manager deve prevedere i rischi che potrebbero influenzare la tempistica del progetto o la qualità del software sviluppato, e dovrebbe prendere provvedimenti per evitarli. La gestione del rischio riguarda l'identificazione dei rischi e l'elaborazione di piani per minimizzare il loro effetto su un progetto. Ci sono tre tipi di rischio, tra loro collegati:

- *Rischi per il progetto*: influenzano la tempistica o le risorse del progetto;
- *Rischi per il prodotto*: influenzano la qualità o le prestazioni del software che si sta sviluppando;
- *Rischi economici*: influenzano l'organizzazione che sta sviluppando o acquistando il software.

Il processo di gestione dei rischi è suddiviso nelle seguenti fasi:

- *Identificazione*;
- *Analisi*;
- *Pianificazione*;
- *Monitoraggio*.

Questo processo è iterativo.

5.4.1 Identificazione del rischio

Può essere eseguita con un approccio *brain storming*, o solo basandosi sull'esperienza. Per facilitare il processo di identificazione può essere utilizzata una linea di diversi tipi di rischio:

- *Rischi tecnologici*;
- *Rischi riguardanti il personale*;
- *Rischi organizzativi*;
- *Rischi strumentali*;
- *Rischi dei requisiti*;
- *Rischi di stima*.

5.4.2 Analisi del rischio

Bisogna considerare ogni rischio identificato e valutarne probabilità e gravità. Queste sono le stime della probabilità:

- $< 10\% \rightarrow$ molto bassa;
- $10 - 25\% \rightarrow$ bassa;
- $25 - 50\% \rightarrow$ moderata;

- $50 - 75\% \rightarrow$ elevata;
- $> 75\% \rightarrow$ molto elevata.

Mentre questi sono gli effetti del rischio:

- *Insignificante*;
- *Tollerabile*;
- *Grave*;
- *Catastrofico.*

5.4.3 Pianificazione del rischio

Si cerca di trovare una strategia per riuscire a gestire ognuno dei rischi identificati. Le strategie possono essere di tre tipi:

- *Strategie per evitare*;
- *Strategie per minimizzare*;
- *Piani precauzionali.*

5.4.4 Monitoraggio del rischio

È la valutazione regolare di ogni rischio per decidere se sta diventando più o meno probabile o se stanno cambiando i suoi effetti. Per eseguire la valutazione, si esaminano i fattori di rischio e si valutano se gli effetti del rischio sono cambiati. Ogni rischio chiave dovrebbe essere discusso nelle riunioni sullo stato di avanzamento della gestione.

Capitolo 6

Requisiti

I requisiti di un sistema sono la descrizione dei servizi forniti e dei vincoli operativi da rispettare. Il proceosso di ricerca, analisi documentazione e verifica di questi servizi e vincoli è chiamato *ingegneria dei requisiti*. I requisiti sono divisibili in *requisiti utente* e *requisiti di sistema*.

6.1 Requisiti funzionali e non funzionali

I requisiti dei sistemi software sono divisi in:

- *Requisiti funzionali*: elenchi di servizi che il sistema deve fornire;
- *Requisiti non funzionali*: vincoli sui servizi o sulle funzioni offerti dal sistema;
- *Requisiti di dominio*: derivanti dal dominio di applicazione del sistema.

6.1.1 Requisiti funzionali

Descrivono quelle che il sistema dovrebbe fare. Le specifiche dei requisiti funzionali di un sistema dovrebbero essere:

- *Complete*, cioè tutti i servizi richiesti dal cliente devono essere definiti;
- *Coerenti*, cioè i requisiti non devono avere definizioni contraddittorie.

6.1.2 Requisiti non funzionali

Si riferiscono a proprietà del sistema, e specificano o limitano le proprietà emergenti del sistema: performance, safety, availability, etc. I requisiti non funzionali vengono determinati sulla base delle necessità degli utenti, del budget a disposizione e delle politiche organizzative. I principali tipi di requisiti non funzionali sono:

- *Requisiti del prodotto*: specificano il comportamento del prodotto;
- *Requisiti organizzativi*: derivanti dalle politiche organizzative del cliente e dello sviluppatore;
- *Requisiti esterni*: derivanti da fattori non provenienti dal sistema e dal suo processo di sviluppo.

6.1.3 Requisiti di dominio

Spesso riflettono i fondamenti del dominio di applicazione, e se non sono soddisfatti potrebbe essere impossibile utilizzare il sistema.

6.2 Requisiti utente

Dovrebbero descrivere i requisiti funzionali e non funzionali del sistema senza scendere troppo nel dettaglio, in modo da risultare comprensibili a coloro che non hanno le competenze tecniche. Dovrebbe essere usato il linguaggio naturale corredato con gli schermi intuitivi, anche se potrebbero sorgere problemi da ciò:

- *Mancanza di chiarezza*;
- *Confusione dei requisiti*;
- *Mescolanza dei requisiti.*

I consigli per minimizzare i fraintendimenti sono:

- Definire un formato standard e usare sempre questo;
- Usare il linguaggio in maniera coerente e dividere i requisiti obbligatori da quelli desiderati;
- Usare diversi stili di testo per indicare punti chiave nel requisito;
- Evitare, nei limiti del possibile, di usare il gergo informatico.

6.3 Requisiti di sistema

Aggiungono dettagli e spiegano come i requisiti utente dovrebbero essere forniti dal sistema. Non dovrebbero specificare troppi dettagli dell'implementazione, ma in pratica ciò è impossibile perché:

- Per la progettazione è necessario avere un'architettura iniziale del sistema per aiutare la specifica dei requisiti;
- É possibile che il sistema debba interagire con altri sistemi esistenti;
- Potrebbe essere necessario l'uso di una specifica architettura per soddisfare i requisiti non funzionali.

6.3.1 Specifiche in linguaggio strutturato

É un modo per scrivere i requisiti in maniera standardizzata che limita le libertà del semplice linguaggio naturale, mantenendo espressività e comprensibilità. Quando si usa una forma standard devono essere incluse le seguenti informazioni:

- Descrizione della funzione che si sta specificando;
- Descrizione degli input e da dove provengono;

- Indicazioni di quali altre entità saranno utilizzate;
- Descrizione dell'azione da eseguire;
- Precondizioni, postcondizioni;
- Possibili effetti collaterali.

6.4 Specifica delle interfacce

Se un nuovo sistema e uno esistente devono interfacciarsi, allora le interfacce del sistema esistente devono essere chiare e ben specificate. Le interfacce possono essere di tre tipi:

- Interfacce procedurali: dove il sistema esistente (o sottosistemi) offrono una gamma di servizi che sono accessibili usando delle procedure di interfaccia (API);
- Strutture dati: trasmesse da un sottosistema ad un altro;
- Rappresentazione dei dati.

6.5 Documento dei requisiti

Il documento dei requisiti (SRS: Software requirements specification) è una dichiarazione ufficiale di quello che gli sviluppatori del sistema dovrebbero implementare. Deve includere sia i requisiti utente, sia una specifica dettagliata dei requisiti di sistema. Il documento dei requisiti ha vari destinatari, che vanno dai manager di alto livello dell'azienda committente agli ingegneri del software che si occupano dello sviluppo. Il livello di dettagli del documento SRS dipende dal tipo di sistema che deve essere sviluppato e dal processo di sviluppo utilizzato. Sono stati sviluppati molti standard per il documento SRS, tra cui l'IEEE così suddiviso:

1. *Introduzione*
   (a) Scopo del documento dei requisiti
   (b) Scopo del prodotto
   (c) Definizioni, acronimi, abbreviazioni
   (d) Riferimenti
   (e) Descrizione del resto del documento
2. Descrizione generale
   (a) Prospettiva del prodotto
   (b) Funzioni del prodotto
   (c) Caratteristiche utente
   (d) Vincoli generali
   (e) Presupposti e dipendenze

3. Requisiti specifici: riguardano requisiti funzionali, non funzionali, di dominio e di interfaccia

4. Appendice

5. indice

Capitolo 7

Processi di ingegneria dei requisiti

Lo scopo del processo di ingengeria dei requisiti è quello di creare e mantenere un documento dei requisiti di sistema. Il processo generale include quattro sottoprocessi di alto livello:

- *Studio di fattibilità*;
- *Definizione ed analisi*;
- *Specifica*;
- *Convalida*.

Ci sono varie possibili rappresentazione del processo di ingegneria dei requisiti, tra cui:

- *Modello standard*: lo studio parte della scoperta, della documentazione e dal controllo dei requisiti;
- *Modello a spirale*: si svolge in tre fasi e tutte le attività sono organizzate in un processo iterativo intorno ad una spirale.

Il processo di gestione dei cambiamenti dei requisiti all'interno di questi modelli si chiama *"gestione dei requisiti"*.

7.1 Studi di fattibilità

Lo studio di fattibilità prende come input i requisiti aziendali preliminari, e restituisce in output un rapporto che indica se vale la pena o no di continuare con il processo di ingegneria dei requisiti e lo sviluppo del sistema. Lo studio di fattibilità è corto e mira a rispondere alle domande:

- *"Il sistema contribuirà agli obiettivi generali dell'azienda?"*;
- *"Può essere implementato usando la tecnologia attuale secondo i costi e i tempi prefissati?"*;
- *"Può essere integrato con i sistemi già installati?"*.

7.2 Deduzione e analisi dei requisiti

In questa fase gli ingegneri del software lavorano con i clienti e con gli utenti finali al fine di capire i servizi che il sistema dovrebbe offrire, le prestazioni che dovrebbe avere. Ricordando che gli *stakeholdel* sono qualsiasi persona o gruppo influenzato dal sistema, capire i loro requisiti è difficile per diversi motivi:

- Gli stakeholder spesso non sanno cosa vogliono dal sistema informatico;
- Gli ingegneri dei requisiti devono capire cosa hanno in mente gli stakeholders;
- Gli ingegneri dei requisiti devono considerare tutte le possibili sorgenti di requisiti provenienti dai diversi stakeholders;
- I fattori politici possono influenzare i requisiti di sistema;
- L'importanza dei requisiti varia nel tempo e nell'azienda.

Le fasi del progetto di deduzione e analisi dei requisiti sono:

- *Scoperta dei requisiti*;
- *Classificazione e organizzazione dei requisiti*;
- *Negoziazione e priorità dei requisiti*;
- *Documentazione dei requisiti.*

Questo processo è un processo iterativo.

7.2.1 Scoperta dei requisiti

É il processo di raccolta delle informazioni sui sistemi proposti ed esistenti, dalle quali ricavare i requisiti utente e di sistema.

Punti di vista

Uno dei punti di forza dell'analisi orientata ai punti di vista è la capacità di riconoscere molteplici prospettive e fornire una struttura per scoprire i conflitti nei requisiti proposti dai vari stakeholders.

Interviste

Le interviste con gli stakeholders fanno parte della maggioranza dei processi di ingegneria dei requisiti. In queste interviste gli ingegneri fanno domande agli stakeholders riguardo il sistema che usano e su quelle che verrà sviluppato.

Scenari

Le persone trovano più semplice riferirsi a situazioni di vita reale piuttosto che a situazioni astratte. Gli scenari possono risultare molto utili per aggiungere dettagli ai requisiti. Nell'eXtreme Programming viene fatto molto uso degli scenari. Ogni scenario descrive una o più possibili interazioni tra utenti e sistema e per ogni iterazione devono essere presenti più scenari, in modo che ciascuno fornisca informazioni a diversi livelli di dettaglio.

Casi d'uso

Sono una tecnica basata su scenari per la deduzione dei requisiti. Sono la funziona fondamentale in UML per descrivere modelli di sistema orientati agli oggetti.

Etnografia

I sistemi software non sono isolati, sono usati in un contesto sociale e organizzativo, e i requisiti di sistema possono derivare o essere vincolati da questo contesto. L'etnografia è una tecnica di osservazione che può essere utilizzata per capire i requisiti sociali e organizzativi .

7.3 Convalida dei requisiti

Consiste nel dimostrare che i requisiti definiscono veramente il sistema desiderato dal cliente. Esistono vari tipi di controllo dei requisiti:

- *Controllo di validità*;
- *Controlli di consistenza*;
- *COntrolli di completezza*;
- *Controlli di realismo*;
- *Verificabilità*.

Possono essere usate delle tecniche di validazione:

- *Revisione dei requisiti*: i requisiti vengono analizzati sistematicamente da un team di revisori che fanno parte sia dell'azienda cliente sia di quella appaltatrice; le revisioni dei requisiti possono essere *informali* o *formali*; inoltre dovrebbero controllare la loro *verificabilità*, *comprensibilità*, *tracciabilità* e *adattabilità*;
- *Prototyping*;
- *Generazione del test-case*.

7.4 Gestione dei requisiti

La gestione dei requisiti è il processo di gestione e controllo dei cambiamenti ai requisiti di sistema. Questo processo dovrebbe partire non appena è disponibile una bozza del documento SRS, ma pianificare la gestione delle modifiche dovrebbe iniziare durante il processo di scoperta.

7.4.1 Requisiti duraturi e volatili

Rispetto alla stabilità i requisiti possono essere divisi in due tipi:

- *Duraturi*: derivanti dall'attività basilare dell'azienda e che si riferiscono direttamente al dominio del sistema;
- *Volatili*: che probabilmente cambieranno nel tempo durante il processo software

7.4.2 Pianificare la gestione dei requisiti

La pianificazione è il primo passo nella gestione dei requisiti e solitamente è molto costosa. Per ogni progetto viene stabilito il livello di dettaglio richiesto attraverso le seguenti fasi:

- *Identificazione dei requisiti*;
- *Processo di gestione delle modifiche*;
- *politiche di tracciabilità*;
- *Supporto degli strumenti CASE.*

Oltre a queste relazioni ci sono anche le relazioni tra i requisiti e il perché sono stati indicati. Per questo è importante la tracciabilità di ogni requisito. Ci sono tre tipi di tracciabilità delle informazioni:

- *Tracciabilità della sorgente*;
- *Tracciabilità dei requisiti*;
- *Tracciabilità del progetto.*

Per la gestione dei numerosi requisiti può essere utile un supporto CASE:

- *Memorizzazione dei requisiti*;
- *Gestione delle modifiche*;
- *Gestione della tracciabilità.*

7.4.3 Gestione delle modifiche ai requisiti

La gestione delle modifiche ai requisiti dovrebbe essere applicata a tutti i cambiamenti proposti: usando un modello formale le modifiche saranno tracciabili. Il processo è diviso nelle seguenti fasi:

- *Analisi del problema e specifica delle modifiche*;
- *Analisi e stima dei costi*;
- *Implementazione della modifica.*

Capitolo 8

Modelli di sistema

Si possono usare diversi modelli per rappresentare il sistema secondo diverse prospettive:

- *Prospettiva esterna*;
- *Prospettiva comportamentale*;
- *Prospettiva strutturale*.

L'aspetto più importante di un modello è che esclude i dettagli, e i diversi tipi di modelli si basano sui diversi tipi di astrazione desiderati. Esistono cinque modelli principali che sono usati:

- *Modello data-flow*;
- *Modello aggregativo (composition)*;
- *Modello architetturale*;
- *Modello classificativo*;
- *Modello azione-reazione*.

8.1 Modelli contestuali

Un modello contestuale descrive l'ambiente in cui opera il sistema, ma non le sue relazioni con altri sistemi. Per questo spesso è necessario affiancarlo con altri modelli più specifici che mostrano le attività del sistema.

8.2 Modelli comportamentali

Utilizzati per descrivere il comportamento generale del sistema. Due tipi di modelli comportamentali sono data-flow e azione-reazione, e questi possono essere usati insieme o separatamente.

8.2.1 Modelli data-flow

Modello per mostrare il flusso dei dati nel sistema. le notazioni usate sono:

- Rettangoli arrotondati per le elaborazioni funzionali;
- Rettangoli per l'immagazzinamento dei dati;
- Frecce denominate per il passaggio dei dati tra le funzioni.

8.2.2 Modelli di macchina a stati

Descrive come un sistema risponde agli eventi interni od esterni. Le notazioni usate sono:

- Rettangoli arrotondati per gli stati del sistema che includono una breve descrizione delle azioni eseguite in tale stato;
- Frecce denominate per rappresentate gli stimoli che obbligano le transizioni da uno stato all'altro.

8.3 Modelli di informazione

Una tecnica di modellazione è la *modellazione Entità-Relazione-Attributo (E-R-A)*, ampiamente utilizzata nella progettazione di database (in UML si usa un modello E-R-A semplificato). In questo modelli mancano i dettagli, perciò questi ultimi si dovrebbero raccogliere in archivi (repository) o dizionario di dati. Un dizionario di dati è semplicemente una lista di dati ordinata alfabeticamente dei nomi inclusi nei modelli di sistema. I vantaggi nell'uso di un dizionario dei dati sono:

- *Meccanismo della gestione dei nomi*;
- *Salvataggio delle informazioni organizzative.*

8.4 Modelli a oggetti

L'approccio orientato agli oggetti è comunemente usato per l'intero processo di sviluppo software. Sviluppare modelli ad oggetti in fase i analisi semplifica le fasi di progettazione e di programmazione orientata agli oggetti. Durante l'analisi si dovrebbero modellare le entità del mondo reale usando le classi, senza includere dettagli dei singoli oggetti nel sistema (istanza della classe). Una classe in UML è rappresentata da un rettangolo verticale con tre sezioni:

- Nome della classe oggetto nella parte superiore;
- Attributi della classe nel mezzo;
- Operazioni associate alla classe nella parte inferiore.

8.4.1 Modelli ereditari

Nella modellazione ad oggetti è possibile individuare delle classi importanti nel dominio che si sta studiando, che sono organizzate in una tassonomia, cioè in uno schema di classificazione che mostra come una classe si colega con le altre attraverso attributi e servizi comuni.

8.4.2 Aggregazione di oggetti

Alcuni oggetti possono anche essere il risultato dell'aggregazione di altri oggetti.

8.4.3 Modellazione del comportamento di oggetti

Per modellare il comportamento degli oggetti si deve mostrare come sono utilizzare le operazioni fornite degli oggetti (in UML si usano gli use-case, i sequence diagram e i collaboration diagram).

8.5 Modelli strutturati

É un modo sistematico di produrre modelli di un sistema esistente o di un sistema che deve essere costruito. Essi forniscono una struttura per la modellazione dettagliata di un sistema come parte della scoperta e analisi dei requisiti. Il vantaggio nel loro utilizzo è che possono prtoare significativamente riduzioni dei costi perché usano notazioni standard, e si assicurano che sia prodotta della documentazione standard di progettazione. Tuttavia hanno una serie di punti deboli:

- Non forniscono efficace supporto per capire e modellare requisiti di sistema non funzionali;
- Non includono linee guida per capire e decidere se è un metodo appropriato per un particolare problema;
- Speso producono troppa documentazione e l'essenza dei requisiti può essere nascosta;
- I modelli prodotti sono troppo dettagliati e di difficile comprensione.

8.6 Altri modelli

- *Modello di ingegneria simultanea (o concorrente)*: ha come obiettivo la riduzione di tempi e costi di sviluppo, mediante un approccio sistematico al progetto integrato e concorrente di un prodotto software e del processo ad esso associato, e le fasi di sviluppo coesistono invece di essere eseguite in sequenza;
- *Modello basato su metodi formali*: comprende una serie di attività che conducono alla specifica formale matematica del software, al fine di eliminare ambiguità, incompletezze ed inconsistenze e facilitare la verifica dei programmi mediante l'applicazione di tecniche matematiche.

8.6.1 Il modello Microsoft

Un processo che è al tempo stesso iterativo, incrementale e concorrente e che permette di esaltare le doti di creatività delle persone coinvolte nello sviluppo di prodotti software.

Approccio synch-and-stabilize

L'approccio usato attualmente da Microsoft è noto come "synch-and-stabilize".

Parte III

Progetto

Capitolo 9

la fase di pianificazione

La pianificazione di un progetto software permette di definire un quadro di riferimento per controllare, determinare l'avanzamento ed osservare lo sviluppo di un progetto software in modo da sviluppare il prodotto software nei tempi e costi stabiliti, con le desiderate caratteristiche di qualità. I componenti fondamentali sono:

- *Dcoping (raggio d'azione)*, che comprende il problema ed i lavoro che deve essere svolto;
- *Stime*;
- *Rischi*;
- *Schedule*;
- *Strategia di controllo.*

9.1 Stime nei progetti software

Le attività di stima di tempi costi ed effort nei progetti software sono effettuate con gli obiettivi di ridurre al minimo il grado di incertezza e limitare i rischi. Le tecniche di stima possono basarsi su:

- Stime su progetti simili già completati;
- "Tecniche di scomposizione";
- Modelli algoritmici empirici.

Le tecniche di scomposizione utilizzano una strategia "divide et impera" e sono basate su:

- Stime dimensionali (ad esempio LOC (Lines Of Codes) o FP (Function Points));
- Suddivisione dei task e/o delle funzioni con relativa stima di allocazione dell'effort.

9.2 Lines Of Codes (LOC)

Il Lines Of Codes (LOC) è un'unità di misura che rappresenta le dimensioni basandosi sul numero di linee del codice sorgente.

9.3 Function Point (FP)

Il Function Point (FP) è un'unità di misura ponderata della funzionalità del software che misura la quantità di funzionalità in un sistema in base al sistema specifico (*stima prima dell'implementazione*). L'FP è calcolato in due fasi:

- Calcolo di un *Unadjusted Function point Count (UFC)*;
- Moltiplicare l'UFC per un *Technical Complexity Factor (TFC)*.

Da qui abbiamo che FP=UFC∗TFC.

9.3.1 Conteggio FP

Categorie di dati

Il conteggio viene effettuato considerando le categorie di dati:

- *Number of Internal Logical Files (ILF)*;
- *Number of External Interfaces Files (EIF)*.

Da nktare che il termine "file" non significa file nel senso tradizionale dell'elaborazione dei dati, se si riferisce ad un gruppo di dati logicamente correlati e non all'implementazione fisica di quei gruppi di dati.

Categorie di transizioni

Il conteggio viene effettuato considerando le categorie di transizioni:

- *Number of External Inputs (EI)*;
- *Number of External Outputs (EO)*;
- *Number of External Inquiries (EQ)*, ossia tutte le combinazioni di input/output univoche, in cui un input causa e genera un output immediato senza modificare lo stato dei file logici interni.

Fattori di influenza

Ad ogni fattore viene associato un valore intero compreso tra 0 (influenza irrilevante) e 5 (influenza essenziale).

Valore	Descrizione
0	Non presente, o non influente
1	Incidentalmente influente
2	Moderatamente influente
3	Mediamente influente
4	Significativamente influente
5	Forte influenza per tutto

i fattori da considerare sono:

- *Reliable back-up and recovery*;

- *Data communication*;
- *Distributed data processing*;
- *Performance*;
- *Heavily used configuration*;
- *Online data entry*;
- *Operational ease*;
- *Online update*;
- *Complex interface*;
- *Complex processing*;
- *Reusability*;
- *Installation ease*;
- *Multiple sites*;
- *Facilitate change.*

Calcolo TFC

Definiamo con F_j il valore assegnato al j-esimo fattore da considerare, il valore TCF viene calcolato nel seguente modo: $\text{TFC} = 0.65 + 0.01 \sum_{j=1}^{14} F_j$. Il TFC varia tra 0.65 (se tutte le F_j sono settate a 0) e 1.35 (se tutte le F_j sono settate a 5).

9.3.2 FP vs LOC

Attraverso vari studi è stato possibile mettere in relazione le metriche LOC e FP, definendo il numero medio di istruzioni codice sorgente per punto funzione (da notare che i linguaggi sono stati classificati in diversi livelli in base alla relazione tra LOC e FP)

9.3.3 COCOMO

Il COCOMO (COnstructive COst MOdel) è un modello algoritmico usato per determinare il valore dell'effort. Il valore ottenuto per l'effort viene successivamente utilizzato per determinare durata e costi di sviluppo. COCOMO comprende 3 modelli:

- Basic (per stime iniziali);
- Intermidiate (usato dopo aver suddiviso il sistema in sottosistemi);
- Advanced (usato dopo aver suddiviso in moduli ciascun sottosistema).

La stima dell'effort viene effettuata a partire da:

- La stima delle dimensioni del progetto KLOC;

- La stima del modo di sviluppo del prodotto, che misura il livello intrinseco di difficoltà nello sviluppo, tra:
  - Organic (per prodotti di piccole dimensioni);
  - Semidetached (per prodotti di dimensioni intermedie);
  - Embedded (per prodotti complessi).

Capitolo 10

la fase di progettazione

Nella fase di progettazione si decidono le modalità di passaggio da "che cosa" deve essere realizzato nel sistema software a "come" deve essere realizzato. Questa fase prende in input il documento di specifica (analisi dei requisiti) e produce un documento di progettazione che guida la successiva fase di codifica. La fase di progetto può essere suddivisa in due sottofasi:

- Progetto architetturale (o preliminare);
- Progetto dettagliato.

10.1 Principi di progettazione

- Stepwise refinement;
- Astrazione;
- Decomposizione modulare;
- Modularità;
- information hiding;
- Riusabilità.

10.1.1 Stepwise refinement

il procedere per raffinamenti successivi è una strategia di progettazione top-down proposta da Wirth nell'ambito della programmazione strutturata. Il raffinamento è un processo di elaborazione che parte dalla specifica di una funzione (o di dati) in cui non si descrive il funzionamento interno della funzione o la struttura interna dei dati. Il raffinamento elabora la specifica aggiungendo ad ogni passo un livello di dettaglio maggiore

10.1.2 Astrazione

L'astrazione consiste nel concentrarsi sugli aspetti essenziali di un'entità e nell'ignorare i dettagli secondari. Nell'ambito del processo software, ogni passo rappresenta un raffinamento del livello di astrazione della soluzione. i principali tipi di astrazione sono:

- Astrazione procedurale;
- Astrazione dei dati.

10.1.3 Modularità

I prodotti software fatti in un unico monolitico blocco di codice sono difficili da mantenere, correggere, capire ed usare. La soluzione consiste nel suddividere il prodotto software in segmenti più piccoli detti moduli.

10.1.4 Decomposizione modulare

Un modulo è un elemento software che:

- Contiene istruzioni, logica di elaborazione e strutture dati;
- Può essere compilato separatamente e memorizzato all'interno di una libreria software;
- Può essere incluso in un programma;
- Può essere usato invocando segmenti di modulo identificati da un nome e da una lista di parametri;
- Può usare altri moduli;

La suddivisione in moduli di un sistema software (decomposizione modulare) produce come risultato l'identificazione di un'architettura dei moduli (structure chart). La decomposizione modulare si basa sul principio del "divide et impera". Una buona divisione di un prodotto software in moduli è quella che permette di ottenere:

- Massima coesione (cohesion) interna ai moduli;
- Minimo grado di accoppiamento (coupling) tra i moduli.

Infatti, massima coesione e minimo coupling permettono di incrementare:

- Comprensibilità;
- Manutenbilità;
- Estensibilità;
- Riusabilità.

10.1.5 Coesione

La coesione di un modulo rappresenta la misura in cui il modulo espleta internamente tutte le azioni necessarie a espletare una data funzione. La coesione, quindi, misura il grado di interazione interna al modulo tra le azioni di una funzione. Esistono 7 livelli di coesione (il livello 1 è il peggiore, mentre il livello 7 è il migliore):

- *Casuale*: non esiste nessuna relazione tra gli elementi del modulo;

- *Logico*: gli elementi sono correlati, di cui uno viene selezionato dal modulo chiamante;
- *Temporale*: esiste una relazione di ordine temporale tra gli elementi;
- *Procedurale*: gli elementi sono correlati in base ad una sequenza predefinita di passi che vengono eseguiti sulla stessa struttura dati;
- *Comunicativo*: gli elementi sono correlati in base ad una sequenza predefinita di passi che vengono eseguiti sulla stessa struttura dati;
- *Funzionale*: tutti gli elementi sono correlati dal fatto di svolgere una singola funzione.

10.1.6 Accoppiamento

L'accoppiamento misura il grado di accoppiamento tra moduli. Esistono 5 livelli di accoppiamento (il livello 1 è il peggiore, il livello 5 è il migliore):

- *Contenuto*: un modulo da difetto riferimento al contenuto di un altro modulo;
- *Comune*: due moduli che accedono alla stessa struttura dati;
- *Controllo*: un modulo controlla esplicitamente l'esecuzione di un altro modulo;
- *Francobollo*: due moduli che si passano come argomento una struttura dati, della quale si usano solo alcuni elementi;
- *Dato*: due moduli che si passano argomenti omogenei, ovvero argomenti semplici o strutture dati delle quali si usano tutti gli elementi.

La forza dell'accoppiamento dipende da:

- Il numero di riferimenti di un modulo all'altro;
- La quantità di dati passati/condivisi tra i moduli;
- La complessità dell'interfaccia tra i moduli;
- La quantità di controllo esercitata da un modulo rispetto all'altro.

10.1.7 Information hiding

I concetti di astrazione procedurale e astrazione dei dati sono derivati da un concetto più generale detto information hiding. La tecnica