Analisi e Progettazione Software

DaveRhapsody

30 Marzo 2020 (Sì, in ritardissimo, lo so)

Indice

Capitolo 1

Modelli di processo

Per modello si intende una rappresentazione semplificata del processo, ed ognuno di questi si basa su un aspetto specifico:

- **Modello Workflow**
- **Modello Data-Flow**
- **Modello Role-Action**

Modelli generici:

- **Modello a cascata:** A cascata significa che l'inizio dell'attività successiva dipende dalla precedente, il che significa che se non è termminata la precedente, non parte la successiva.

  Feedback: Una volta arrivati al termine della cascata, si può risalire a ritroso, sempre con la stessa regola, l'utente può fornire feedback generalmente in seguito ai test del sistema (e vorrei vedere a consegnare qualcosa di non testato).

  Consegna incrementale: Invece di rilasciare tutto il sistema di botto alla fine, lo si rilascia in modo incrementale, poca roba per volta, come alpha e beta nei videogiochi, stesso discorso.

  Quali sono i vantaggi? Alcune funzioni importanti sono disponibili fin dal primo prototimo, hai meno rischi di fallimento, e i servizi a priorità più alta sono collaudati più a fondo

- **Modello ciclico:** Hai un certo numero di fasi, ciclicamente le ripeti una dopo l'altra all'infinito, come una spirale.
- **Ingegneria software basata sui componenti**

Il concetto è che in base al modello, cambia completamente il modo in cui si affronta lo sviluppo di un software, occorre (per trovare il migliore per le proprie esigenze) capire quale modello usare, e quando.

1.1 Modello UP (Unified Process) e RUP (Rational Unfied Process)

E' un processo iterativo e incrementale, che suddivide un processo gigante in iterazioni più controllate.

In sintesi: Poca roba per volta, feedback più rapidi, iterazioni di lunghezza definita e fissata, e modellazione con UML

Organizzazione del processo:

1. Avviamento
2. Elaborazione
3. Costruzione
4. Transizione

Ogni ciclo è un incremento, e questo porta vantaggi legati a minori fallimenti, riduzione precoce dei rischi, progrwssi più visibili, feedback precoci. Come caratteristiche principali ha che è iterativo e incrementale, ha maggiore enfasi sul modello invece che sul linguaggio naturale, e si centra sull'architettura.

1.2 Processo Scrum

E' un processo iterativo che si basa sul controllo dello stato di avanzamento. Come principi essenziali ha:

- **Visibilità**: Gli aspetti significativi devono essere visibili per tutti
- **Ispezione**: Letteralmente si ispeziona frequentemente il lavoro per capire se si sta andando nella direzione giusta
- **Adattamento**: Se si sta deviando dagli obbiettivi servirà un adattamento per minimizzare le deviazioni

1.2.1 Ruoli

1. **Product Owner**: Definisce le caratteristiche del progetto da sviluppare (1 persona sola), e gestisce il Product Backlog
2. **Team**: Dedicato allo sviluppo e rilascio del prodotto tramite incrementi successivi (da 3 fino a 8 membri)
3. **ScrumMaster**: Responsabile del fatto che lo SCRUM venga applicato nel modo corretto

NON E' UN PROJECT MANAGER: Il PM gestisce il lavoro, lo SM invece soltanto il processo scrum, son due cose diverse. Inoltre lo SM riduce il Gap tra il PM e lo sviluppo, supporta il PM, facilità creatività e affiatamento del team, è praticamente un motivatore.

1.2.2 Catatteristiche

1. **Release brevi**: I sottoinsiemi vengono cosegnati presto
2. **Progetto semplice**: In ogni istante tutti i test sono eseguibili, c'è un numero minimo di funzionalità e non hai logica duplicata
3. **Refactoring**: Banalmente, rolling release, miglioramento continuo
4. **Testing First**: Il testing è continuo
5. **Pair programming**: Più sviluppatori lavorano insieme
6. **Collective Ownership**: Il software non è di Pippo o Pluto, ma del team composto da essi
7. **Continuous integration**: Si hanno integrazioni continue e check-in frequenti
8. **Cliente sul campo**: Nel senso che il cliente è rompicazzo e lavora con il team.. Non so quanto positiva questa cosa, dipende da cliente a cliente.

Capitolo 2

Sistemi Socio-Tecnici

Definizione di sistema: E' un insieme di componenti che lavorano assieme per un obbiettivo comune

- Può essere sw + hw, questa cosa è già vista in Reti e Sistemi Distribuiti

2.1 Categorie di sistema

2.1.1 Tecnico-Informatici

Includono Hw + Sw e non includono operatori e processi operazionali. Inoltre il sistema non conosce lo scopo del suo utilizzo.

2.1.2 Socio-Tecnici

Includono uno o più sistemi tecnici, ma anche i processi operazionali e gli operatori, però di contro sono condizionati dalle politiche aziendali e dalle regole aziendali

Caratteristiche:

1. **Proprietà emergenti**: Le proprietà del sistema finale dipendono dalle sue componenti e dalle relazioni tra le compoenti, possono essere misurate solamente sul sistema finale
2. **Non-Determinismo**: Il sistema non risponde sempre con lo stesso output dato lo stesso input perchè è indipendente dagli operatoi umani
3. **Complesso legame tra i sistemi e gli obbiettivi aziendali**: L'efficacia nel supportare gli obiettivi aziendali non dipende dal sistema stesso
4. Volume di un sistema (spazio occupato)
5. Affidabilità
6. Protezione
7. Riparabilità
8. Usabilità

Proprietà del tipo "NON DEVE ACCADERE" Prestazioni o affidabilità possono essere misurate, ma altre proprietà consistono in cose che non devono succedere, dal punto di vista della **sicurezza** ma anche della **protezione**.

2.1.3 Organizzazione/Persone/Sistemi

I sistemi socio-tecnici sono pensati per obbiettivi aziendali, o organizzativi, nel senso che **se non si comprende l'ambiente organizzativo in cui un sistema è usato, allora le probabilità che so sviluppiun sistema soddisfacente son basse**

2.1.4 Sistemi critici

1. **Sistemi safety-critical**: Un fallimento porta anche fino alla morte di una vita umana
2. **Sistemi mission-critical**: Un fallimento ti manda all'aceto tutta l'attività a obbiettivi diretti (sistema di navigazione di un veicolo spaziale difettoso)
3. **Sistemi business-critical**: Perdite di soldi

2.1.5 Tipi di fallimenti

- **Fallimento Hardware**: Errori di produzione o consumo
- **Fallimento Software**: Errori del programma
- **Errori operativi**: I classicissimi errori umani

Riporto dalle slides l'esempio del caso Mizuho Securities:

2.1.6 Fidatezza dei sistemi

Nei sistemi critici, generalmente la fidatezza è la più importante proprietà del sistema, per via dell'alto costo di un fallimento.

Utilità e fidatezza non sono la stessa cosa: Un sistema può essere utile anche se gli utenti non hanno confidenza nel suo buon funzionamento.

2.1.7 Altre proprietà

1. Riparabilità: Riflette la facilità con cui un sistema può essere riparato in caso di errori
2. Mantenibilità: Come si adatta il sistema a requisiti nuovi
3. Sopravvivenza: Come fornisce servizi un sistema sotto attacco
4. Tolleranza degli errori: quanto si tollerano errori di immissione?

2.1.8 Metodi di sviluppo per sistemi critici

A causa del costo dei fallimenti, si hanno software di protezione che sono tutto meno che economici, e questo è dato dal fatto: pensa quanto risparmi avendo più sicurezza, confrontato a quanto perdi se un attacco va a buon fine, that's it.

Economie di fidatezza: A causa del costo di un sistema dotato di elevata fidatezza, potrebbe essere conveniente sviluppare un sistema non fidato e pagare per i costi di fallimento (Pagare con la vita dei propri operai, per esempio).

- La scelta dipende da fattori politici e sociali, qui la concorrenza se per ipotesi il prezzo fosse davvero la morte di qualche operaio, vince a mani bas
- La scelta dipenderà dal sistema, per un normale sistema di business, una fidatezza moderata può essere sufficiente

2.2 Analisi dei requisiti

E' un processo di ricerca, analisi, documentazione e verifica dei servizi richiesti dal cliente e i vincoli entro i quali i servizi devono operare

2.2.1 Tipi di requisito

1. Requisito utente: Affermazioni in **Linguaggio naturale**, corredate da tabelle e diagrammi che riguardano i servizi che il sistema offre, al cliente glieli si dà scritti
2. Requisito di Sistema: E' la base per il progetto della soluzione, può essere illustrato utilizzando i modelli di sistema
3. Requisito funzonale: E' l'insieme dei servizii che il sistema deve o non deve fornire
4. Requisito NON-funzionale: REquisito del prodotto, requisito esterno o anche organizzativo
    - Prodotto: come deve comportarsi il prodotto
    - Organizzativo: Politiche di organizzazione del cliente, se lavori per RadioMaria, il sistema non bestemmia, per intenderci

- Esterno: Il sistema deve rispettare le norme di dove verrù usato. Tipo un drone deve rispettare le leggi dei droni, questo discorso.

Problemi dei requisiti

1. Ambiguità: Non è chiaro chi fa cosa
2. Incompletezza: Non è possibile avere un documento dei requisiti completo
3. Inconsistenza: Una descrizione non deve contenere contraddizioni o conflitti.

Esempio: Avendo due requisiti di questo tipo:
> l'utente riceve tutte le news pubblicate dal suo ultimo accesso
> le news pubblicate rimangono disponibili per 7 giorni

Siccome specificare i requisiti in linguaggio naturale fa schifissimo forever, allora si usano alternative a quest'ultimo, come scrittura dei requisiti in un formato standard, oppure ci si basa sullo standard IEEE 830