Euristiche Ingegneria del software

Euristica per le relazioni extend e include.

- Utilizzare extend per le funzionalità eccezionali, opzionali o che si verificano raramente.
- Utilizzare include per funzionalità che sono condivise da due o più casi d'uso.

Euristica per l'identificazione degli oggetti di analisi iniziali.

- Termini che gli sviluppatori e gli utenti devono chiarire per comprendere il caso d'uso.
- Ricorrere a sostantivi nei casi d'uso. (es. Incidente).
- Le entità del mondo reale che il sistema deve tracciare. (es. Ufficiale su campo, Risorsa).
- I processi del mondo reale che il sistema deve tracciare. (es. Piano delle operazioni di emergenza).
- Casi d'uso.
- L'origine dei dati.
- Gli artefatti con la quale gli utenti interagiscono. (es. La stazione).
- Usare sempre termini del dominio applicativo.

Identificazione degli attori Domande da
porsi:

- Quale gruppo di utenti vengono supportati dal sistema per lo svolgimento del loro lavoro?
- Quale gruppo di utenti eseguono le principali funzioni del sistema?
- Quale gruppo di utenti svolge le funzioni secondarie, come manutenzione e amministrazione?
- Con quale sistema esterno, software o hardware, il sistema interagirà?

Indentificare gli scenari

- Non aspettarti che il cliente sia chiaro se il sistema non esiste.
- Non aspettare le informazioni se il sistema già esiste.
- Utilizza un approccio dialettico :
  - Aiuta il cliente a formulare i requisiti di cui ha bisogno.
  - Il cliente ti aiuterà a farti capire i requisiti.
  - I requisiti si evolvono mentre vengono sviluppati gli scenari.

Domande da porsi:

- Quali sono i principali compiti che gli attori vogliono che il sistema svolga?
- A quali informazioni gli attori accedono? Chi crea quei dati? Possono essere modificati o rimossi? Da chi?
- Quali sono le informazioni esterne che il sistema deve sapere? Quante volte? Quando?
- Di quali eventi il sistema deve informare l'attore? Con quale tempistica?

Identificare i casi d'uso (use cases).

Come determinare gli use case?

- Seleziona una stretta sezione verticale del sistema (uno scenario). Discutine in dettaglio con gli utenti per capirne lo stile preferito di iterazione.
- Seleziona una sezione orizzontale (diversi scenari) per definire lo scopo del sistema. Discuti dello scopo con gli utenti.
- Utilizza prototipi illustrativi (mock-up) e supporti visuali.
- Studia i comportamenti degli utenti. Osserva come fanno il proprio lavoro.

Semplice guida per scrivere un caso d'uso:

- I casi d'uso dovrebbero essere chiamati con delle **frasi verbo**. Il nome di uno caso d'uso dovrebbe indicare che cosa l'utente stia provando a fare.
- Gli attori dovrebbero essere chiamati con dei **sostantivi** (Vittima, poliziotto ecc.).

- I confini del sistema dovrebbero essere chiari. I passi compiuti dall'attore e i passi compiuti dal sistema dovrebbero essere distinti.
- I passi nel flusso degli eventi dovrebbero essere frasi in **FORMA ATTIVA,** in modo da rendere esplicito chi sta compiendo quell'azione.
- Le **relazioni** casuali tra le fasi successive dovrebbero essere chiare. Perché si verifica questo e cosa determina l'input di questa azione.
- Un caso d'uso dovrebbe descrivere un'operazione completa dell'utente. (deve portare ad una ben definita exit condition).
- Le **eccezioni** dovrebbero essere descritte separatamente.
- Un caso d'uso **NON** dovrebbe descrivere l'interfaccia utente del sistema (meglio attraverso prototipi o mock-up).
- Un caso d'uso **NON** dovrebbe superare le due o tre pagine di lunghezza. Altrimenti conterrebbe e estenderebbe relazioni da decomporre in casi d'uso più piccoli.

Euristica per lo sviluppo di scenari e casi d'uso.

- Utilizza gli scenari per comunicare con gli utenti e validarne le funzionalità.
- Prima raffina un singolo scenario per capire i presupposti dell'utente in merito al sistema. L'utente potrebbe avere familiarità con sistemi simili, in quel caso, adottare convenzioni specifiche per l'interfaccia utente renderebbe il sistema più utilizzabile.
- Definisci un po' di scenari non molto dettagliati per definire l'obiettivo del sistema. Validali con l'utente.
- Utilizza mock-up (modelli) solo come supporto visuale; Il disegno dell'interfaccia dovrebbe avvenire in un task separato dopo che la funzionalità è sufficientemente stabile.
- Presenta alternative multiple e molto diverse all'utente. Valutare diverse alternative allarga gli orizzonti dell'utente. Generando alternative differenti porta gli studenti a "think outside the box".
- Dettaglia una buona fetta verticale quando lo scopo del sistema e le preferenze degli utenti sono state comprese bene.

Euristica per il controllo dei casi d'uso e degli oggetti partecipanti.

- Quali casi d'uso creano questo oggetto? (es. in quale caso d'suo si trovano i valori degli attributi dell'oggetto inseriti nel sistema?).
- Quali attori possono accedere a tali informazioni?
- Quale caso d'uso modifica ed elimina questo oggetto? (es. quale caso d'uso modifica o rimuove questa informazione dal sistema?).
- Quale attore può attivare questo caso d'uso?È necessario questo oggetto? (es. c'è al massimo un caso d'uso che dipende da questa informazione?).

Euristiche requisiti funzionali

- Un requisito rispetta il seguente formato: [Condition][Subject][Action][Object][Constraint]
  - oppure: [Condiction] [Action or Constraint][Value]
  - oppure: [Subject][Action][Values]
- Un requisito descrive delle interazioni che avvengono tra il sistema e il suo ambiente, indipendentemente dal modo in cui è implementato
- Un requisito indica CHE COSA fa il sistema quando l'utente utilizza una sua funzionalità
- Un requisito deve essere scritto evitando l'uso di pronomi vagi(es. "esso", "questo", "quello" ecc.)
- Un requisito deve essere scritto evitando l'uso di termini non verificabili e senza limiti precisi(es. "ma no limitato a", "come minimo" ecc.)
- Un requisito deve essere univocamente identificato? (es. numero, tag)

Requisiti non funzionali

Categoria	**Cosa chiedersi?**

Usabilità	• Qual è il livello di abilità dell'utente? • Quali standard di interfaccia sono familiari all'utente? • Quale documentazione dovrebbe essere fornita all'utente?
Affidabilità	• Quanto affidabile, disponibile e robusto dovrebbe essere il sistema? • Il riavvio del sistema è ammissibile se si verifica un errore? • Quanti dati il sistema può perdere? • Come dovrebbe gestire le eccezioni il sistema?Ci sono requisiti safe nel sistema? (es. una centrale nucleare deve essere safe). • Ci sono requisiti secure nel sistema? (es. la lunghezza di una password è secure).
Prestazione	• Quanto reattivo dovrebbe essere il sistema? • Ci sono tempi critici per le attività degli utenti? • Quanti utenti concorrenti dovrebbe supportare? • Quanto è grande un tipico archivio di dati per dei sistemi comparabili? • Qual è il ritardo accettabile per gli utenti?
Supportabilità (include manutenzione e sopportabilità)	• Quali sono le estensioni previste nel sistema? • Chi si occupa della manutenzione del sistema? • Ci sono dei piani per trasferire il sistema in ambienti hardware o software differenti?
Implementazione	• Ci sono vincoli sulla piattaforma hardware? • Ci sono vincoli imposti dal team di manutenzione? • Ci sono vincoli imposti dal team di testing?
Interfaccia	• Il sistema dovrebbe interagire con sistemi già esistenti? • Come i dati vengono importati/esportati nel sistema? • Quali standard utilizzati dal cliente dovrebbero essere supportati dal sistema?
Operazione	• Chi gestisce il sistema attuale?

Impacchettamento	• Chi installa il sistema? • Quante istallazioni sono previste? • Ci sono limiti di tempo sull'istallazione?
Legale	Come dovrebbe essere autorizzato il sistema? • Ci sono problemi di responsabilità associati ai fallimenti del sistema? • Ci sono eventuali diritti d'autore o costi di licenza associati all'uso di algoritmi o componenti specifiche?

Euristiche requisiti non funzionali:

- Un requisito rispetta il seguente formato:
  - [Condition][Subject][Action][Object][Constraint]
  - oppure : [Condiction] [Action or Constraint][Value]
  - oppure [Subject][Action][Values]
- Il requisito indica COME il sistema dovrebbe essere?
- Un requisito deve essere scritto utilizzando espressioni positivo
- Un requisito deve essere scritto evitando l'uso di linguaggi soggetto (es: user friendly oppure easy to use)
- Un requisito non deve essere ambiguo
- Un requisito non deve essere scritto in forma passiva

Euristica use case diagram

- Il nome degli use case diagram deve rispettare questo formato: UCD_<acronimoGestione>: <nome use case diagram>.
- Nel diagramma gli attori sono rappresentati da omini stilizzati.
- Nel diagramma i casi d'uso sono rappresentati da ovali.
- Gli attori sono collegati con una linea continua ai casi d'uso a cui partecipano.
- Gli attori principali sono posizionati sul lato sinistro del rettangolo.
- Gli attori secondari sono posizionati sul lato destro del rettangolo.
- Prima del nome dell'attore che indica un sistema può essere aggiunto lo stereotipo "<<system>>".
- l'inclusione dei casi d'uso è indicata con un freccia accompagnata dallo stereotipo "<<include>>".
- L'estensione dei casi d'uso è indicata con una freccia accompagnata dallo stereotipo "<<extend>>".
- Non devono esserci catene di inclusioni.

Euristiche Statechart

- Il nome dello Statechart Diagram deve rispettare questo modello: SCD_<acronimoGestione>: <nome dell'entità coinvolta>.
- Nel diagramma deve essere presente lo stato iniziale
- Lo stato iniziale è rappresentato con un cerchio colorato di nero
- Gli stati generici sono rappresentati con un rettangolo i cui angoli sono stondati
- La sintassi relative alle transazioni segue questo modello: Evento [guardia]/azione 1; azione2;...;azione n
- Nel diagramma deve essere presente lo stato finale
- Lo stato finale è rappresentato dal simbolo dello stato iniziale inscritto in un cerchio più grande a sfondo bianco.

Euristica di Abbott

Ci si basa sull'analisi del linguaggio naturale per identificare oggetti, attributi, associazioni dalla specifica dei requisiti; mappano parti del parlato (nomi, verbo avere, verbo essere, aggettivi) per modellare componenti (oggetti, operazioni, relazioni di ereditarietà, classi). Lo schema è il seguente:

Parti del parlato	Componenti del modello
nome proprio	istanza
Nome comune	class
verbo fare/azione	operazione
verbo essere	gerarchia
verbo avere	aggregazione
aggettivo	attributo

Euristica oggetti Entity

- Termini che gli sviluppatori e gli utenti hanno bisogno di chiarire per comprendere gli use case (es: information submitted by FieldOfficer)
- Sostantivi ricorrenti negli use case (es. Incident)
- Entità del mondo reale che il sistema deve considerare (es. FieldOfficer, Dispatcher, Resource)
- Attività del mondo reale che il sistema deve considerare (es. EmergencyOperationPlan)
- Sorgenti o destinazioni di dati (es. Printer)

Per ogni oggetto identificato,

- si assegna un nome (univoco) e una breve descrizione, per gli oggetti Entity è opportuno utilizzare gli stessi nomi utilizzati dagli utenti e dagli specialisti del dominio applicativo
- Si individuano attributi e responsabilità (non tutti, soprattutto non quelli ovvii)
- Il processo è iterativo e varie revisioni saranno richieste: quando il modello di analisi sarà stabile sarà necessario fornire una descrizione dettagliata di ogni oggetto.

Euristica oggetti Boundary

- In ogni use case, ogni attore interagisce almeno con un oggetto Boundary.
- L'oggetto Boundary colleziona informazione dall'attore e la traduce in una forma che può essere usata sia dagli oggetti Control che Entity.
- Identifica i controlli della UI di cui l'utente ha bisogno per iniziare lo use case (ReportEmergencyButton)
- Identifica form di cui l'utente ha bisogno per inserire dati nel sistema (ReportEmergencyForm)
- Identifica avvisi e messaggi che il sistema usa per rispondere all'utente (AcknowlegmentNotice)
- Quando più attori sono coinvolti in uno use case, identifica gli attori che sono terminali (DispatcherStation) per riferirti alla UI in considerazione
- Non modellare aspetti visuali della UI con oggetti Boundary (meglio mock-up)
- Usa semprei termini dell'utente finale per descrivere l'interfaccia, non usare termini del dominio di implementazione

Euristica oggetti Control

- Gli oggetti Control sono responsabili del coordinamento degli oggetti Boundary e Entity.
- Identifica un oggetto Control per ogni use case
- Identifica un oggetto Control per ogni attore in uno use case
- La vita di un oggetto Control dovrebbe corrispondere alla durata di uno use case o di una sessione utente. Se è difficile identificare l'inizio e la fine dell'attivazione di un oggetto Control, il corrispondente use case probabilmente non ha delle entry e exit condition ben definite.

Euristiche per disegnare un Sequence Diagram

- La 1° colonna dovrebbe corrispondere all'attore che inizia lo use case
- La 2° colonna dovrebbe essere un oggetto Boundary(che l'attore usa per iniziare lo use case)
- La 3° colonna dovrebbe essere l'oggetto Controlche gestisce il resto dello use case

- Gli oggetti Control sono creati dagli oggetti Boundary che iniziano gli use case
- OggettiControl e Boundary accedono a oggetti Entity
- Gli oggetti Boundary sono creati da oggetti Control
- Gli oggetti Entity non accedono **mai** agli oggetti Control e Boundary: ciò rende più facile condividere oggetti Entitytra più use case.

Euristica System design Criteri di design

Possiamo selezionare gli obiettivi di design da una lunga lista di qualità desiderabili. I criteri sono organizzati in cinque gruppi:

- Performance
- Dependability
- Cost
- Maintenance
- End user criteria

Criteri di Performance

Includono i requisiti imposti sul sistema in termini di spazio e velocità

- Tempo di risposta: con quali tempi una richiesta di un utente deve essere soddisfatta dopo che la richiesta è stata immessa?
- Troughput: quanti task il sistema deve portare a compimento in un periodo di tempo prefissato?
- Memoria: quanto spazio è richiesto al sistema per funzionare?

Dependability criteria

Quanto sforzo deve essere speso per minimizzare i crash del sistema e le loro conseguenze?
Rispondono alle seguenti domande:

- Robustness (robustezza). Capacità di sopravvivere ad input non validi immessi dall'utente
- Reliability (affidabilità). differenza fra comportamento specificato e osservato
- Availability (disponibilità). Percentuale di tempo in cui il sistema può essere utilizzato per compiere normali attività
- Fault tolerance. Capacità di operare sotto condizioni di errore
- Security. Capacità di resistere ad attacchi di malintenzionati
- Safety. Capacità di evitare di danneggiare vite umane, anche in presenza di errori e di fallimenti

Cost criteria

Includono i costi per sviluppare il sistema, per metterlo in funzione e per amministrarlo.
Quando il sistema sostituisce un sistema vecchio, è necessario considerare il costo per assicurare la compatibilità con il considerare il costo per assicurare la compatibilità con il vecchio o per transitare verso il nuovo sistema I criteri di costo:
Development Cost: Costo di sviluppo del sistema iniziale
Deployment cost: costo relativo all'installazione del sistema e training degli utenti
Upgrade cost: costo di convertire i dati del sistema precedente. Questo criterio viene applicato quando nei requisiti è richiesta la compatibilità con il sistema precedente (backward compatibility)
Maintenance cost (costo di manutenzione). Costo richiesto per correggere errori sw o hw (bug)
Administration cost (costo di amministrazione). Costo richiesto per amministrare il sistema **Maintenance criteria**

Determinano quanto deve essere difficile modificare il sistema dopo il suo rilascio

- Estensibilità.Quanto è facile aggiungere funzionalità o nuove classi al sistema?
- Modificabilità. Quanto facilmente possono essere cambiate le funzionalità del sistema?
- Adattabilità. Quanto facilmente può essere portato il sistema su differenti domini di applicazione?
- Portabilità. Quanto è facile portare il sistema su differenti piattaforme?
- Leggibilità. Quanto è facile comprendere il sistema dalla lettura del codice?

- Tracciabilità dei requisiti. Quanto è facile mappare il codice nei requisiti specifici

End User Criteria

Includono qualità che sono desiderabili dal punto di vista dell'utente, ma che non sono state coperte dai criteri di performance e dependability. Criteri:

- Utilità: quanto bene dovrà il sistema supportare il lavoro dell'utente?
- Usabilità: quanto dovrà essere facile per l'utente l'utilizzo del sistema?

Scelta dei sottosistemi

Trovare I sottosistemi è simile ad individuare oggetti nell'analisi.

Criteri per la selezione dei sottosistemi: la maggior parte delle interazioni dovrebbe essere entro i sottosistemi, piuttosto che attraverso i limiti dei sottosistemi (alta coesione) sottosistemi (alta coesione):

- Quale sottosistema chiama qualche altro per ottenere servizi?

Primary Question:

- Quale tipo di servizio è fornito dal sottosistema?

Secondary Question:

- Possono essere i sottosistemi ordinati gerarchicamente (layers)?

Euristiche:

Tutti gli oggetti nello stesso sottosistema dovrebbero essere funzionalmente correlati.

- Assegnare gli oggetti identificati in un caso d'uso allo stesso sottosistema
- Creare un sottosistema dedicato per gli oggetti usati per muovere i dati fra i sottosistemi
- Minimizzare il numero di associazioni che attraversano i limiti dei sottosistemi