Stima dell'impegno basato sui casi d'uso

John Smith

White paper del software Rational

TP 171, 10/99

Rational®
the software development company

Indice

Il problema

Intuitivamente, sembrerebbe possibile eseguire una stima delle dimensioni e dello sforzo richiesti per lo sviluppo basato sulle caratteristiche del modello Caso d'uso. Dopo tutto, il modello Caso d'uso cattura i requisiti funzionali, dunque non dovrebbe esservi uno caso d'uso equivalente di punti di funzione. Si presentano varie difficoltà:

- Esistono molte variazioni dello stile e della formalità della specifica del caso d'uso che rende molto difficile definire la metrica, si dovrebbe essere disponibili, ad esempio, a misurare la lunghezza di un caso d'uso.
- I casi d'uso dovrebbero rappresentare la vista di un attore esterno di un sistema, quindi un caso d'uso per un sistema composto da un software di 500.000 righe di codice (sloc) è alquanto diverso da un livello di caso d'uso scritto per un sottosistema sloc di 5.000. (Cockburn97 descrive la nozione di livelli e obiettivi).
- I casi d'uso possono essere diversi per complessità, sia esplicitamente per scrittura, sia implicitamente nella realizzazione richiesta.
- Un caso d'uso dovrebbe descrivere il comportamento dal punto di vista dell'attore, ma ciò potrebbe risultare complesso, specialmente se il sistema ha degli stati (come nella maggior parte dei casi). Quindi, descrivere tale comportamento può richiedere un modello del sistema (prima che venga eseguita ogni realizzazione), che può avere come conseguenza un numero eccessivo di livelli della scomposizione e dei dettagli funzionali, nel tentativo di catturare l'essenza del comportamento.

È dunque necessario un tipo di caso d'uso per rendere possibile la stima? Forse le aspettative sulla stima direttamente con casi d'uso sono troppo alte e tracciare dei paralleli tra i punti di funzione e una nozione di punti di caso d'uso è incauto. Il calcolo dei punti di funzione richiede comunque un modello del sistema. La derivazione di punti di funzione dalle descrizioni di un caso d'uso richiederebbe una uniformità di livelli nell'espressione del caso d'uso ed è solo quando le realizzazioni iniziano ad emergere che si dovrebbe avere più fiducia nel conteggio del punto di funzione. Fetcke97 descrive una mappatura dal punto di caso d'uso a quello di funzione, ma il livello deve comunque essere appropriato per una mappatura valida. Altri metodi utilizzano metriche basate su classe/oggetto come origine, ad esempio PRICE Object Points (Minkiewicz96).

Altri lavori

Esiste un giusto carico di lavoro nella descrizione e formalizzazione dei casi d'uso, Hurlbut97 ha una buona valutazione. Si ottiene molto di meno derivando le metriche di stima dai casi d'uso. Graham95 e Graham98 contengono una severa critica dei casi d'uso (seppure non si capisce pienamente perchè) e propone l'idea di ‘script di compito’ come un modo per superare i problemi con i casi d'uso, comprese le variazioni di lunghezza e complessità. Lo ‘script di compito atomico’ di Graham è la base della collezione di una metrica del ‘punto di compito’. Il problema legato allo script di compito atomico è un livello molto basso: secondo Graham, dovrebbe rappresentare una singola sentenza e non è ulteriormente scomponibile utilizzando solo la terminologia di dominio. I ‘compiti basilari’ di Graham contengono uno o più script di compito atomico e ogni compito basilare corrisponde “esattamente ad una operazione di sistema: alla classe che avvia il piano”(Graham98). Tali compiti basilari sono molto simili a casi d'uso a basso livello mentre gli script di compito atomico sono simili alle operazioni in tali casi d'uso. Il problema del livello tuttavia resta.

A tale riguardo sono stati eseguiti altri lavori da Karner (Karner93), Major (Major98), Armour e Catherwood (Armour96) e Thomson (Thomson94). Il documento di Karner postula un metodo per calcolare i punti del caso d'uso, ma presuppone che questi siano espressi in un modo realizzabile da classi (ad esempio, ad un livello di dettagli migliore dei sottosistemi).

Si dovrebbe dunque evitare i casi d'uso per la stima e affidarsi alle realizzazioni di analisi e progettazione che emergono? Il problema che ne nasce è un rallentamento dell'abilità di fare stime e non sarà soddisfacente per il responsabile di progetto che ha scelto tale tecnologia, le prime stime saranno necessarie e dovranno essere utilizzati altri metodi. È preferibile che il responsabile di progetto sia capace di ottenere stime precedenti per la pianificazione, per poi perfezionarle iterazione dopo iterazione, piuttosto che posticipare le stime e procedere in modo non pianificato.

In questo documento è descritto un quadro in cui i casi d'uso di ogni livello possono essere utilizzati per fare una stima dell'impegno. Per presentare le idee, sono descritte strutture semplici e canoniche, con dimensioni e misure associate che hanno una base di esperienza. Il documento propone molte congetture ardite (oppure esplicite), non presenta altro, considerata la mancanza di lavoro e di dati in quest'area. Per la formulazione, si è ricorso all'idea di ‘sistemi di sistemi interconnessi’.

Di seguito si farà una breve digressione per definire alcuni pensieri fondamentali che hanno portato in questa direzione.

Evitare la scomposizione funzionale?

L'idea della scomposizione funzionale sembra essere un'anatema per molti nello sviluppo software. L'esperienza personale portata all'estremo (tremila transform primitivi in un diagramma del flusso di dati molto ampio, profondo cinque o sei livelli, eseguiti senza pensare all'architettura se non a livello di infrastruttura) non mi ha convinto. In questo caso, il problema non riguarda semplicemente la scomposizione funzionale, ma anche l'idea di non descrivere un processo finché il livello funzionale primitivo non sia stato raggiunto, punto nel quale la specifica dovrebbe consistere in meno di una pagina.

Il risultato è molto difficile da capire, ed è difficile discernere come il comportamento richiesto ad un livello più alto emerga da tali transform. Inoltre, non è ovvio il modo in cui la struttura funzionale dovrebbe associarsi ad una struttura fisica che soddisfi le prestazione ed altri requisiti di qualità. Il paradosso sta dunque nel fatto che abbiamo eseguito scomposizioni susseguenti fino a raggiungere il livello al quale si poteva ‘risolvere il problema’(livello primitivo), ma non era chiaro nè dimostrabile se i primitivi che lavorano insieme raggiungano realmente gli obiettivi a livelli più alti. Non c'è modo in questo metodo di prendere in considerazione requisiti non funzionali. L'architettura, nella sua totalità, non solo l'infrastruttura (comunicazioni, sistemi operativi ecc.), dovrebbe evolversi in parallelo alla scomposizione e dovrebbe influenzarsi reciprocamente con queste.

Per quanto riguarda l'editto di Bauhaus secondo cui la ‘forma segue la funzione’, vi sono molte cose positive che sono passate dall'approccio funzionale alla progettazione, ma anche altre negative, come l'utilizzo di tetti piatti ovunque. Se si è tenuto conto solo della funzione di un tetto e della progettazione subordinata del tetto come una riparo per gli abitanti, il risultato, perlomeno in alcune aree, sarà soddisfacente. Tali tetti sono difficili da impermeabilizzare, dunque raccoglieranno molta neve.

Ora tali problemi possono essere risolti, ma con una spesa maggiore rispetto all'utilizzo di una progettazione diversa . Sebbene sembri banale dirlo, la forma dovrebbe seguire i requisiti, funzionali e non, e questi in seguito potranno includere l'estetica. Il problema dell'architetto consisterà spesso nel fatto che i requisiti non funzionali sono esposti in modo scarso, per cui si pone molta fiducia sull'esperienza dell'architetto, rispetto al "modo in cui le cose dovrebbero essere! Dunque la scomposizione funzionale è negativa se guida unicamente l'architettura (se la scomposizione scende di molti livelli e i primitivi funzionali si associano uno ad uno con ‘moduli’) e se definiscono le proprie interfacce.

Considerazioni come questa mi hanno convinto che non avrebbe senso scomporre i casi d'uso in un livello normalizzato (che potrebbe essere realizzato dalla collaborazione di classi) in anticipo sul lavoro strutturale. La scomposizione avviene certamente quando il sistema è di una certa dimensione (vedere Jacobson97) ma i criteri e i processi di strutturazione per la scomposizione sono importanti, la scomposizione funzionale ad hoc non è sufficientemente utile.

Considerazioni di sistema

Gli ingegneri di sistema eseguono analisi funzionale, scomposizione e allocazione (nel sintetizzare una progettazione), anche se la funzione non è l'unica guida per l'architettura, i team di ingegneri specializzati contribuiranno alla valutazione di progettazioni alternative. IEEE Std 1220, the Standard for Application and Management of the Systems Engineering Process, descrive l'utilizzo della scomposizione funzionale nella sezione 6.3, Functional Analysisnella sottosezione 6.3.1 Functional Decomposition, e le soluzioni al prodotto del sistema nella sezione 6.5 Synthesis. Di particolare interesse sono le sottosezioni 6.5.1 Group and Allocate Functions e 6.5.2 Physical Solution Alternatives. Nella sezione 6.3.1, si afferma che la scomposizione è eseguita per capire chiaramente cosa deve compiere il sistema, in genere un livello di scomposizione dovrebbe essere sufficiente.

Notare che lo scopo della scomposizione funzionale non è formare il sistema (compito della sintesi), ma capire e comunicare ciò che il sistema deve fare (un modello funzionale è un modo valido per fare ciò). In sintesi, le sottofunzioni sono allocate a strutture di soluzione e la soluzione è poi valutata, considerando tutti gli altri requisiti. La differenza tra questo approccio e la scomposizione funzionale multi-livello è che a ciascun livello è possibile cercare di descrivere il comportamento richiesto e trovare una soluzione per implementarlo, prima di decidere se il comportamento nel livello successivo debba essere ulteriormente perfezionato e allocato in componenti a livelli inferiori.

Si può trarre la conclusione che non è necessario avere centinaia di casi d'uso per descrivere il comportamento ad ogni livello. Il numero di casi d'uso esterni (e di scenari associati) che coprirà adeguatamente il comportamento delle cose descritte (sistema, sottosistema, classe), può essere ristretto. È opportuno specificare cosa si intende per caso d'uso esterno. Prendiamo come esempio un sistema composto da sottosistemi, composti a loro volta da classi. I casi d'uso che descrivono il comportamenti del sistema e relativi attori sono definiti casi d'uso esterni. I sottosistemi possono avere i propri casi d'uso, che sono interni al sistema ma esterni al sottosistema. Il numero totale di casi d'uso, esterni e interni, utilizzato infine per costruire un sistema molto vasto (1.000.000 righe di codice) potrebbe aggirarsi sulla centinaia, poiché i sistemi di quella misura saranno costruiti come sistemi di sistemi, o almeno come sistemi di sottosistemi.

Presupposti sulla struttura e sulla dimensione

Numero di casi d'uso

La Rational« Software, insegna che il numero di casi d'uso dovrebbe essere ristretto (10-50) e ha osservato che un numero maggiore (oltre 100) può provocare un errore nella scomposizione funzionale, poiché un caso d'uso non consegna cose di valore all'attore. Tuttavia, utilizziamo un alto numero di casi d'uso in progetti concreti e non tutti sono 'negativi' (coprono un insieme di livelli), ad esempio in una e-mail interna di Rational, l'autore cita un esempio da Ericsson:

> Ericsson, nel modellare ampie porzioni di una nuova generazione di switch della telefonia, stima di utilizzare 600 collaboratori per anno (con un picco di 300-400 sviluppatori), 200 casi d'uso (utilizzando più di un livello di casi d'uso, fare riferimento a "Sistemi di sistemi interconnessi") (italico mio)

Per un sistema di 600+ collaboratori per anno (quanto sarà vasto? 1.500.000 righe di codice C++?), si sospetta che l'analisi del caso d'uso si sia interrotta ad un livello precedente il sottosistema (ovvero, se si definisce un sottosistema a 7000-10000 righe di codice), altrimenti il conteggio sarebbe stato ancora più alto.

Resto dunque dell'opinione che un numero ristretto di casi d'uso esterni sia l'ideale. Per far combaciare le strutture e le dimensioni che ho proposto, affermo che 10 casi d'uso esterni, ciascuno con 30 scenari associati [1] sono sufficienti per descrivere il comportamento [2]. Se in un esempio concreto, il numero di casi d'uso va oltre 10 e questi sono puramente esterni a quel livello, il sistema in descrizione è più vasto della forma canonica. Tenterò di fornire successivamente alcune motivazioni che sostengano la validità di tali numeri.

Gerarchia strutturale

La gerarchia strutturale proposta è:

4 — Sistema di sistemi

3 — Sistema

2 — Gruppo di sottosistema

1 — Sottosistema

0 — Classe

Le classi e i sottosistemi sono definiti in UML, gli aggregati maggiori sono sottosistemi (che contengono sottosistemi) in UML. Li ho chiamati in modo diverso per discuterne successivamente. Il gruppo di sottosistema aggregato è di misura simile a CSCI, per chi conosce la terminologia di standard militari come 2167 o 498 (che dovrebbero fare di un sottosistema un CSC e di una classe un CSU). Come ricordato, dopo le discussioni avvenute nei 2167 giorni occorsi a stabilire quali livelli associare ai diversi costrutti Ada, il pacchetto Ada infine risultava quasi sempre associato a CSU. Non sto suggerendo che il sistema deve essere rigidamente conforme a questa gerarchia, vi saranno commistioni tra livelli, ma la gerarchia consente di ragionare sugli effetti della misura sullo sforzo per caso d'uso.

Vi saranno casi d'uso ad ogni livello (anche se forse non per una classe individuale), ma non un'unico insieme di incredibili dettagli, piuttosto casi d'uso per ogni componente (ad esempio, sottosistema, gruppo di sottosistema ecc.) ad un livello. [3]. Ho asserito in precedenza che dovrebbero esserci 10 casi d'uso per ogni componente ad ogni livello. Se le descrizioni del caso d'uso hanno una media di 10 pagine, ciò dà vita ad un potenziale documento della specifica di 100 pagine (più un numero simile per i requisiti non funzionali). Si tratta di

[1] In UML1.3 uno scenario è descritto così: "scenario: sequenza specifica di azioni che illustrano comportamenti. Uno scenario può esser utilizzato per illustrare una interazione o l'esecuzione di un'istanza di caso d'uso". Qui viene utilizzato con il secondo significato di illustrare l'esecuzione di un'istanza di caso d'uso.

[2] Notare che questo numero (di scenari) riflette la complessità di un caso d'uso, non suggerisce che uno sviluppatore deve produrre e scrivere 30 scenari per ogni caso d'uso, piuttosto che 30 scenari catturino buona parte del comportamento di un caso d'uso sebbene vi possano essere molti più percorsi attraverso il caso d'uso.

[3] Alcuni revisori hanno espresso preoccupazione al prospetto di casi d'uso a quattro livelli, ma si ricordi che questo vale per un sistema di sistemi, che in genere è molto ampio. In tali casi, non sarei sorpreso di vedere casi d'uso a quattro livelli, in particolare se il lavoro è svolto dall' appaltatore principale (per il sistema di sistemi), dai subappaltatori (per i sistemi) e forse anche dai subappaltatori minori per i sottosistemi.

un numero preferito da Stevens98 e si avvicina a quello suggerito in Royce98. Ma perché 10 casi d'uso? Per giungere a tale conclusione, ho ragionato partendo dall'inizio, in base a quelle che ritenevo essere misure ragionevoli per varie classi per sottosistema, misura di classe, misura dell'operazione e così via. Tali misure sono raccolte nella seguente tabella, assieme ai riferimenti ad altre supposizioni.

Misura dell'operazione	70 sloc
Numero di operazioni per classe	12
Numero di classi per sottosistema	8
Numero di sottosistemi per gruppo di sottosistema	8
Numero di gruppi di sottosistema per sistema	8
Numero di per sistema di sistemi	8
Numero di casi d'uso esterni (per sistema, sottosistema ecc.)	10
Numero di scenari per caso d'uso	30
Pagine per descrizione di caso d'uso [4]	10

Non possiedo una considerevole quantità di dati empirici, ma ve ne sono di sparsi in tutti i testi. Lorentz94 e Henderson-Sellers96 hanno alcuni dati e io ne possiedo altri tratti da progetti in Australia, principalmente nel campo aerospaziale. Ad ogni caso, era importante a questo punto che i framework fossero posizionati correttamente.

Dimensione dei componenti nella gerarchia

Dovrei dire di aver utilizzato righe di codice sapendo che a qualcuno non piace la dimensione. Si tratta di righe di codice C++ (o linguaggio di livello equivalente), che dovrebbero rendere semplice un ritorno nei punti di funzione.

Dev'esserci una relazione tra i numeri di classi in un contenitore e la varietà del comportamento che può essere espressa. Ho scelto otto classi/sottosistemi[5], otto sottosistemi/gruppo di sottosistema, otto gruppi di sottosistema/sistema e così via. Perché otto?

- È entro 7, più o meno 2.
- Perché a 850 sloc di C++ per classe (12 operazioni ciascuna di 70 sloc), dà una dimensione di sottosistema di 7000 sloc, una buona parte di funzionalità/codice eseguibile da un team ristretto (3-7 persone) in 4-9 mesi, che dovrebbe allinearsi con la lunghezza dei sistemi nella misura di 300.000-1.000.000 sloc (RUP99).[6]

Qual è dunque il numero di casi d'uso che esprimono il comportamento (esternamente) di otto classi, le quali sono coesive e sono state collocate in un sottosistema? Non è solo il numero di casi d'uso, ma anche il numero degli scenari per ogni caso d'uso a determinare la varietà. C'è molto da dire sulle linee guida per l'espansione scenario/caso d'uso, Grady Booch indica in Booch98 che: “Esiste un fattore di espansione dai casi d'uso agli scenari. Un sistema di moderata complessità può avere una dozzina di casi d'uso che catturano il proprio comportamento e ciascun caso d'uso potrebbe espandersi a varie dozzine di scenari…”, e Bruce

Successivamente, tale discorso verrà perfezionato per diverse classi di sistemi.

[5] Ritengo che questa specie di conteggio sia rappresentativo dell'analisi, avverrà un'espansione e un refactoring mediante la progettazione e l'implementazione e il numero delle classi aumenta per un fattore di tre o più, mentre la dimensione dell'operazione e della classe diminuisce in modo equivalente.

[6] Per i sistemi più piccoli (con tempi di iterazione più brevi), i sottosistemi possono essere pianificati per essere più piccoli, oppure è sempre possibile pianificare una produzione parziale per ogni iterazione, sebbene ciò richieda un controllo attento e eventualmente la produzione di ‘stub’.

Powel Douglass dice in Douglass99, "…. diverse dozzine". Ho scelto 30 scenari/caso d'uso, che sono inferiori a 'diverse dozzine", ma Rechtin (in Rechtin91) dice che gli ingegneri possono gestire 5-10 variabili interagenti (che rapportandomi allo scopo di questo argomento interpreto come 5-10 classi in una collaborazione) e 10-50 interazioni (che ho interpretato come scenari). Interpretati in tale modo, casi d'uso multipli consistono in istanze multiple di questo spazio variabile.

Per cui, 10 casi d'uso, ciascuno con 30 scenari, ci dicono che 300 scenari totali (che successivamente porteranno a ~300 casi di test) sono sufficienti per coprire il comportamento interessante di otto classi. Abbiamo altre indicazioni sul fatto che questo sia un numero ragionevole? Se viene applicata la regola di Pareto 80-20, il 20% delle classi produrranno l'80% della funzionalità e, ugualmente, l'80% della funzionalità verrà prodotta dal 20% delle operazioni in ciascuna classe. Per essere prudenti possiamo ipotizzare di aver bisogno del 20% delle classi ecc. per raggiungere il 75% della capacità e costruire una distribuzione di Pareto mediante questo punto (Figura 1).

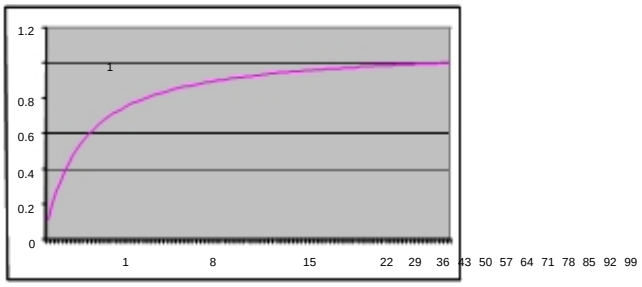


Figura 1: Un esempio di distribuzione di Pareto

Se volessimo l'80% di copertura del comportamento generale e la regola di Pareto venisse applicata a varie classi, operazioni e scenari, avremmo bisogno del 93% (0.93^3 è 0.8) di copertura del comportamento da ognuno, che corrisponde al 50% di ciascuno, ad esempio 4 classi e 5 operazioni (= (12 meno 2 costruttori/distruttore)/2). Il numero dei diversi collegamenti interni dell'albero nodi costruiti per rappresentare i modelli di esecuzione di quattro classi con cinque operazioni ciascuna potrebbe arrivare a diverse migliaia. Ne ho costruita una con massimo tre link da ogni nodo, supponendo una gerarchia, con 10 operazioni (operazioni di interfaccia) in alto che formi un albero di tre livelli,con un risultato di quasi 1000 percorsi o scenari. 500 scenari dovrebbero dare il 93% della copertura. Con 300 scenari, (utilizzando la stessa supposizione) dovremmo ottenere circa il 73% della copertura. Esaminando il modo in cui l'albero dovrebbe essere sfrondato per eliminare le specifiche comportamentali eccessive permette di capire che sono sufficienti anche numeri inferiori, a seconda dell'algoritmo scelto.

Un altro modo di affrontare la questione è chiedere quanti casi di test (derivati da scenari) ci si aspetterebbe per 7000 sloc di C++. Tale test non sarebbe niente rispetto al livello di test di unità e vi sono delle prove nei progetti Jones91 e Boeing 777 (Pehrson96) della sicurezza di questo numero, quantomeno nel fatto che rappresenti una consuetudine. Le fonti citate indicano come numeri sostanzialmente corretti 250 280 [7]. Ad un livello completamente differente, il progetto CAATS (Canadian Automated Air Traffic System) utilizza 200 test di sistema (comunicazioni private).

Dimensione dei casi d'uso

Quanto dovrebbe essere 'grande' un caso d'uso? Abbastanza per presentare dettagli sufficienti perché il comportamento desiderato venga realizzato, ciò dipenderà dalla complessità, interna ed esterna, collegata al tipo di sistema. Qui si incorre nel problema di quanto dell'azione interna di un sistema debba essere descritto. Costruire un sistema da una descrizione del proprio comportamento interno richiede ovviamente che gli output siano collegati agli input. Se ad esempio, il comportamento è sensibile al tempo e complesso, sarà molto difficile descrivere qualche modello concettuale dell'interno del sistema e le azioni che richiede. Notare che non viene descritto necessariamente il modo in cui il sistema deve essere costruito interamente, questo sarà compito di una progettazione che soddisfi i requisiti non funzionali e che corrisponda al comportamento del modello.

[7] Nei feedback ricevuti dai revisori della Rational si riscontra che è più che sufficiente per molti sistemi non fondamentali, avere meno di 30 scenari per caso d'uso. Sarebbe interessante avere maggiori dati relativi a ciò e alla relazione tra i numeri dei casi di test e il numero degli errori scoperti con l'uso.

La definizione offerta in UML1.3 è la seguente: “Caso d'uso [classe]: la specifica di una sequenza di azioni, comprese le varianti, che un sistema (o un'altra entità) può eseguire, interagendo con attori del sistema”. Per i comportamenti complessi, tale definizione può essere considerata per includere ragionevolmente azioni interne, a meno che non si decida di posticipare l'operazione fino alla realizzazione, un ulteriore passaggio che allontana dall'utente. I ruoli di business dovrebbero anche essere incorporati nei casi d'uso per vincolare il comportamento degli attori, ad esempio, in un sistema ATM una banca può stabilire una regola secondo cui non è possibile ritirare più di $500 in una singola transazione, qualunque sia il bilancio del conto.

Con questo tipo di interpretazione, la descrizione del flusso di eventi del caso d'uso può variare dalle 2 alle 20 [8]pagine. Sistemi algoritmicamente semplici con un comportamento semplice non avranno ovviamente bisogno di descrizioni di lunghezza. Forse si potrebbe dire che i sistemi di business semplici sono caratterizzati dalle 2 alle 10 pagine per una media di 5. I sistemi di business e scientifici più complessi variano dalle 6 alle 15 pagine per una media di 9, e i comandi e controlli complessi variano dalle 8 alle 20 pagine per una media di 12 (tali cifre riflettono la relazione non lineare dell'impegno di un sistema per sistemi della stessa misura) sebbene non abbia dati per confermare. Forme descrittive più espressive, macchine di stato o diagrammi dell'attività ad esempio, richiedono meno spazio. Esiste ancora la tendenza ad enfatizzare il testo, per cui per il momento ignorerò gli altri, seppure vi siano pochi o nessun dato.

Gli sviluppi che differiscono sistematicamente da tali dimensioni dovrebbero applicare un moltiplicatore alle ore per casi d'uso derivati da questa euristica (suggerisco di aggiungere un determinatore dei costi di stile COCOMO che è la dimensione media suggerita/osservata per il sistema classificazione di business semplice, più complesso, comando e controllo ecc.).

Un altro aspetto della dimensione del caso d'uso e il livello dello scenario, ad esempio un caso d'uso di sole 5 pagine può avere una struttura complessa che consente molti percorsi. Ancora, il numero di scenari deve essere stimato e il rapporto di questi a trenta (una mia supposizione iniziale di scenari per casi d'uso) utilizzato come determinatore dei costi.

La conseguenza è che si sta asserendo che una specifica basata sul caso d'uso di ~100 pagine dovrebbe essere sufficiente per una specifica esterna ad ogni dato livello, in aggiunta alla specifica supplementare. La quantità varia da 20 a 200 pagine (tali limiti sono approssimativi). Notare che il totale per un sistema (di gruppi di sottosistemi) a livello più basso è di 3-15 pagine/ksloc (sistema di business semplice), 12-30 pagine/ksloc (comandi e controlli complessi). Ciò sembra spiegare l'apparente contraddizione tra Royce98 Tabella 14-9, dove i conteggi delle pagine per artefatti sono alquanto brevi, e l'osservazione dei progetti reali, che in difesa ha prodotto un alto numero di documenti. Tale documento è tratto da un livello di specifica che non deve essere impegnato al documento, Royce ha ragione, le cose importanti, come la specifica della visione, dovrebbero essere nell'ordine indicato nella tabella, 200 pagine per i sistemi vasti e complessi.

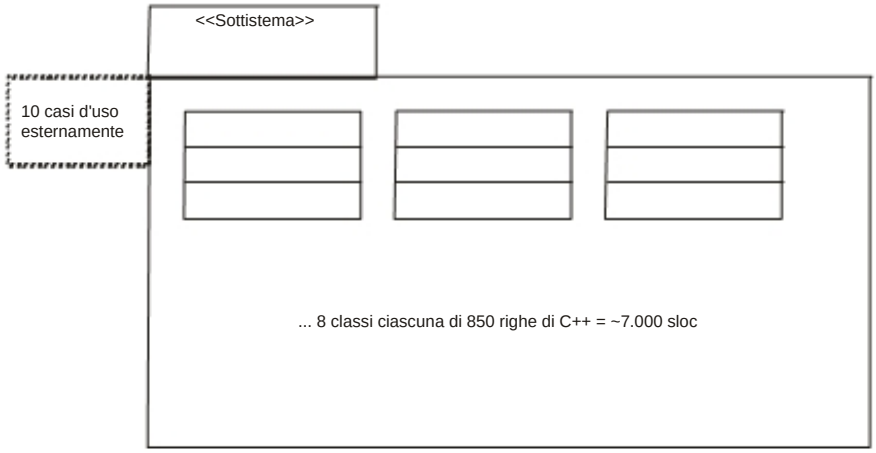
La gerarchia del sottosistema

Come viene rappresentata questa gerarchia di sottosistemi? Ecco le forme di ‘standard’ semplici che ho utilizzato. Notare che si tratta delle forme concettuali per realizzare un sistema. Il confine reale del sistema supera una collezione di queste forme e la somma dei casi d'uso esterni per ciascuna è il totale dei casi d'uso esterni per il sistema, un sistema reale dunque può avere più di dieci casi d'uso esterni, ma il confine superiore non è illimitato, come vedremo dopo. Notare che non si sta suggerendo che tutti gli sviluppi devono utilizzare quattro livelli di caso d'uso nella descrizione. I sistemi più piccoli (<50.000 sloc) ne utilizzeranno probabilmente solo uno o due.

[8] Notare che non deve essere un confine superiore stabile, la lunghezza della descrizione di un caso d'uso seguirà una specie di distribuzione statistica, dove gli estremi hanno minore probabilità di occorrenza.

Livello 1

A Livello 1, abbiamo utilizzato casi d'uso realizzati da classi in nessuno o più sottosistemi:



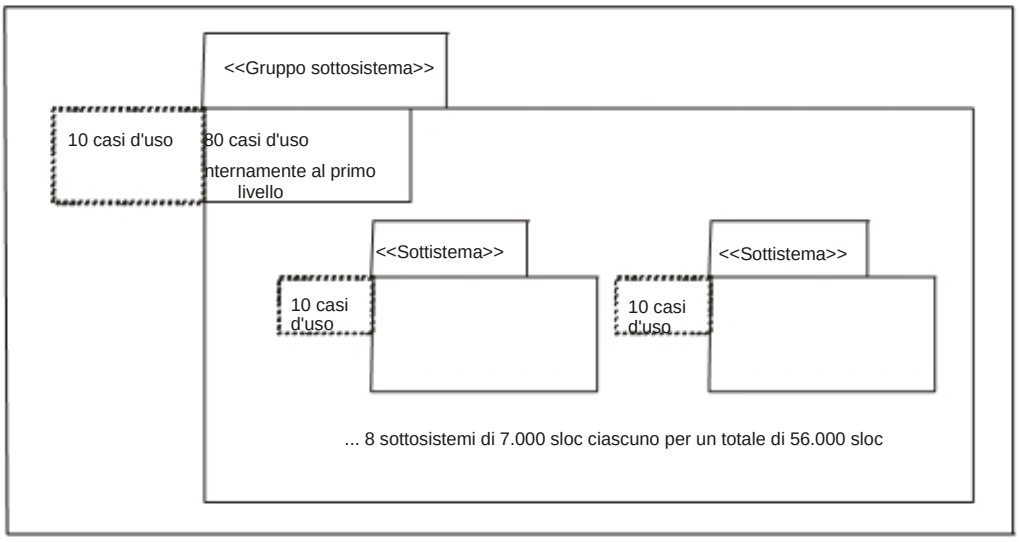
Fare una stima della quantità di dimensioni per sistemi a questo livello (utilizzando la nozione di 7 più o meno 2):

- da 2 a 9 classi (non formati in sottosistemi)— da 1700 sloc a 8000 sloc, o
- 1 sottosistema di 5 classi per un totale di 4000 sloc fino a
- 9 sottosistemi di 7 classi per un totale di 53.550 sloc,

con i casi d'uso espressi perché siano realizzabili da istanze di classe. È un range di 2-76 casi d'uso. Esistono dei confini approssimativi, almeno il confine superiore è che la probabilità di costruire un sistema in questo modo (di tale dimensione), senza mai esprimere il comportamento desiderato in una forma ad alto livello, dovrebbe scendere a zero in questo limite. Un conteggio di caso d'uso più vasto potrebbe indicare qualche patologia.

Livello 2

Al livello successivo, abbiamo un gruppo di sottosistema di otto sottosistemi. Ritengo che sia l'equivalente di CSCI nella terminologia di difesa. A questo livello, i casi d'uso sono realizzati da collaborazioni di sottosistemi:



Fare una stima della quantità di dimensioni per sistemi a questo livello (utilizzando la nozione di 7 o più o meno 2):

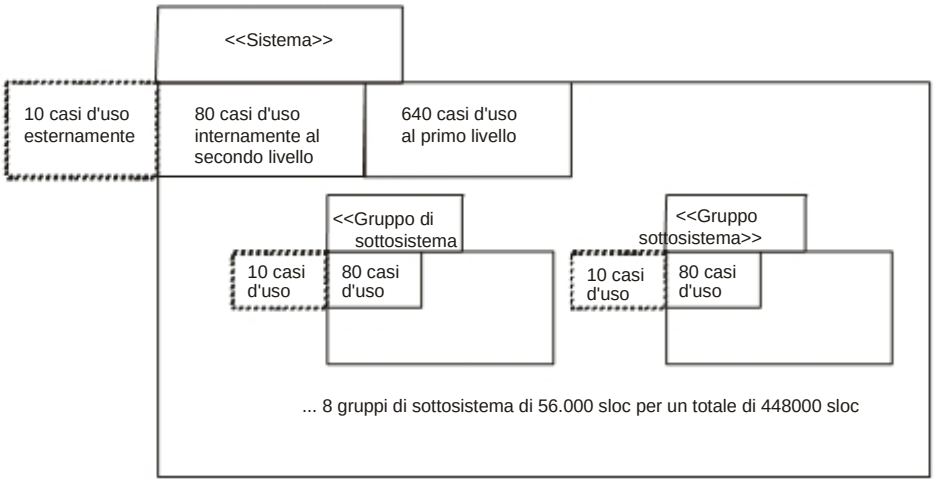
 da 1 gruppi di sottosistema di 5 sottosistemi di 5 classi per un totale di 22.000 sloc, a

 9 gruppi di sottosistema di 7 sottosistemi ciascuno di 7 classi, per un totale di 370.000 sloc

È un range di 4-66 casi d'uso esterni. Ancora una volta, si tratta di confini approssimativi.

Livello 3

A questo livello successivo, abbiamo un sistema (di gruppi di sottosistema). A Livello 3, i casi d'uso sono realizzati da collaborazioni di gruppi di sottosistemi:



Fare una stima della quantità di dimensioni per sistemi a questo livello (utilizzando la nozione di 7 più o meno 2):

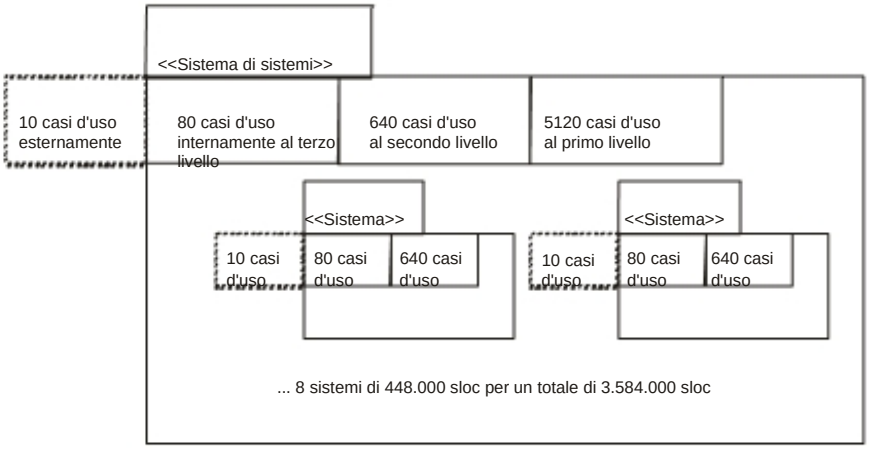
 da 1 gruppo di sottosistema di 5 sottosistemi di 5 classi per un totale di 110.000 sloc, a

 9 sistemi di 7 gruppi di sottosistemi ciascuno di sette sottosistemi ciascuno di 7 classi, per un totale di 2.600.000 sloc.

È un range di 3-58 casi d'uso esterni. Ancora una volta, si tratta di confini approssimativi.

Livello 4

Al livello successivo, abbiamo un sistema di sistemi. Al Livello 4, i casi d'uso sono realizzati da collaborazioni di sistemi:



Fare una stima della quantità di dimensioni per sistemi a questo livello (utilizzando la nozione di 7 più o meno 2):

- da 1 sistema di sistemi di 5 sistemi di 5 gruppi di sottosistema di 5 sottosistemi di 5 classi per un totale di 540.000 sloc, a
- 9 sistemi di sistemi di 7 sistemi ciascuno di 7 gruppi di sottosistema ciascuno di 7 sottosistemi ciascuno di 7 classi, per un totale di 18.000,000 sloc.

È un range di 2-51 casi d'uso esterni. Ancora una volta, si tratta di confini approssimativi. Ritengo che siano possibili degli aggregati più vasti, ma è preferibile non pensarci!

Impegno per caso d'uso

È possibile avere un'idea dell'impegno per caso d'uso facendo una stima dell'impegno per queste misure simboliche ad ogni livello. Utilizzando il tool Estimate Professional™ [9] (basato sui modelli COCOMO 2[10] e SLIM di Putnam[11]), impostando il linguaggio su C++ (altri determinatori dei costi impostati come simbolo) e calcolando l'impegno per ciascun esempio di tipi di sistema ad ogni punto di dimensione simbolico (supponendo 10 casi d'uso esterni), si ottengono i risultati riportati nella Tabella 1.

Tabella 1: Impegno per Caso d'uso per vari esempi di tipo

Dimensione (sloc) d'uso complesso	sistema di business	Impegno ore/caso d'uso semplice sistema	Impegno ore/caso d'uso scientifico comando e controllo sistema
7000 (L1)	55 (range di 40-75)	120 (range di 90-160)	260 (range di 190-350)
56000 (L2)	820 (range di 710-950)	1700 (range di 1500-2000)	3300 (range di 2900-3900)
448000 (L3)	12000	21000	38000
3584000 (L4)	148000	252000	432000

[9] Software Productivity Center Inc, http://www.spc.ca/ fornisce il tool Estimate Professional.

[10] Vedere Boehm81 e l'indirizzo internet http://sunset.usc.edu/COCOMOII/cocomo.html.

[11] Vedere Putnam92.

I range mostrati nella Tabella 1 per il Livello 1 (L1) e il Livello 2 (L2) tengono conto della complessità di un caso d'uso individuale, stimato per analogia con la matrice di complessità del codice COCOMO. In L2, credo che la variazione con la complessità inizierà ad essere sussunta nella caratterizzazione per tipo di sistema, così che un caso d'uso di un sistema di comando e controllo complesso e ad alto livello conterrà un mix di complessità ad un livello più basso. Ipotizzando ciò su scala log-log si ottiene la Figura 2.

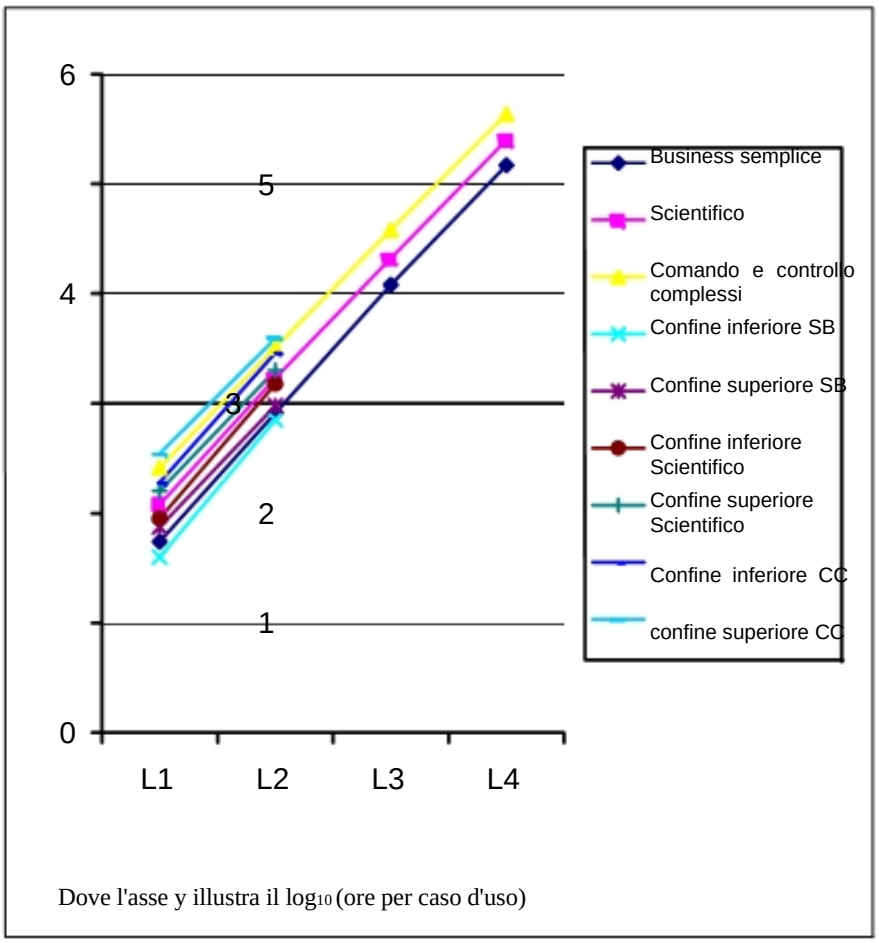


Figura 2: Impegno del caso d'uso per dimensione

Si può dedurre che il numero stabilito da Objectory di 150-350 ore/caso d'uso ($10^{2.17}$-$10^{2.54}$) si adatta a L1, ad esempio questi sono casi d'uso che possono essere realizzati con le collaborazioni delle classi, questo numero ha dunque una giustificazione. Tuttavia, non è adatto a caratterizzare tutti i progetti durante l'analisi, come un collega ha scritto in una comunicazione via e-mail, "è troppo ‘piatto’".

Stima dell'impegno

I sistemi reali non si adattano a tali slot convenzionali, per dare delle indicazioni su come un sistema debba essere caratterizzato, è possibile utilizzare dei limiti approssimativi dedotti strada facendo e impostarli come illustrato nella Figura 3.

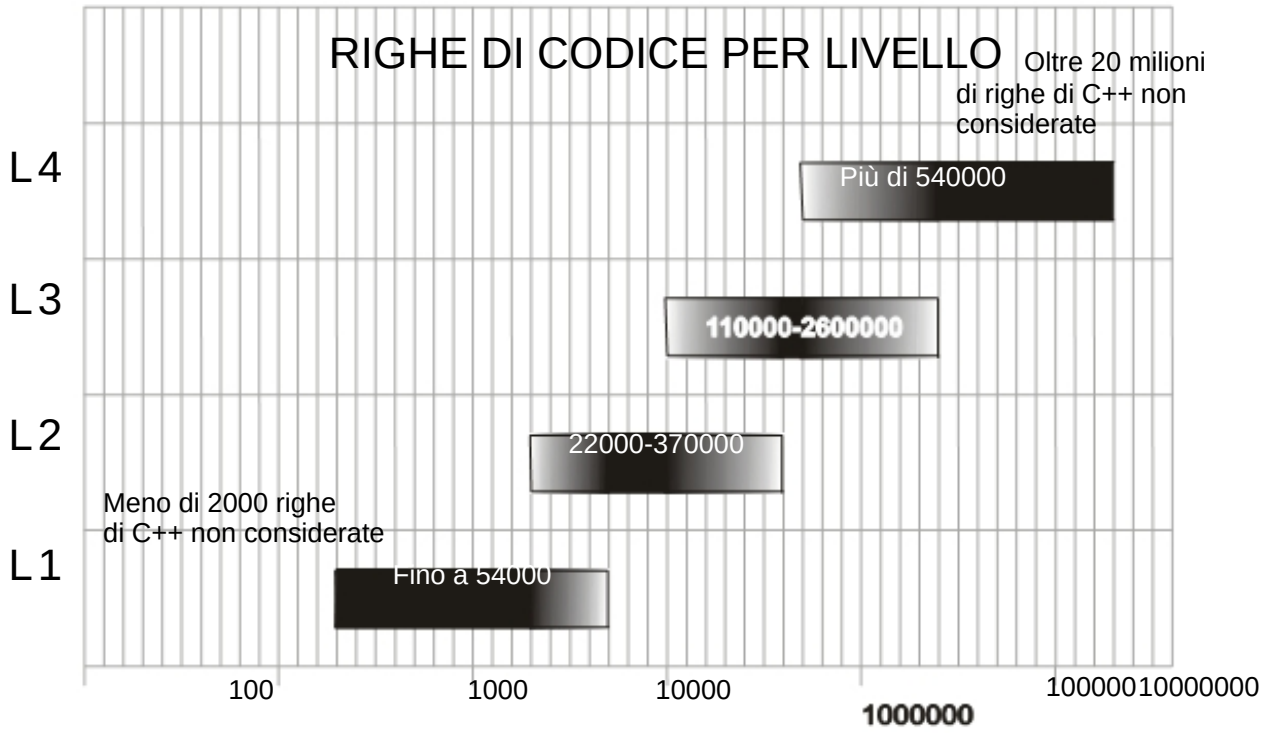


Figura 3: Bande di dimensione per ogni livello

Dalla Figura 3 si può vedere che i sistemi fino a 22000 sloc sono probabilmente da descrivere a Livello 1, con un conteggio di casi d'uso tra 2 e 30. I conteggi di caso d'uso superiori di questa dimensione possono indicare che la granularità di questi casi d'uso è troppo precisa.

Tra i 22000 e 54000 sloc, potrebbe esservi un mix di casi d'uso nei Livelli 1 e 2, per un conteggio di caso d'uso tra 4 (tutto il Livello 2) e 76 (tutto il Livello 1). Come si vuole mostrare nel grafico, tali valori estremi hanno bassa probabilità.

Tra 54000 e 110000 sloc, è possibile che un sistema ben strutturato possa essere descritto interamente al Livello 2, con un conteggio di caso d'uso tra 10 e 20, il mix potrebbe essere L1/L2/L3 (1-160 casi d'uso, con tali estremi aventi probabilità molto bassa).

Tra 110000 e 370000 sloc, potrebbe esservi un mix dei Livelli 2 e 3, per un conteggio di casi d'uso tra 3 (tutto il Livello 3) e 66 (tutto il Livello 2).

Tra 370000 e 540000 sloc, se descritti interamente a Livello 3, potrebbe esserci un conteggio di caso d'uso tra 9 e 12, il mix potrebbe essere L2/L3/L4 (1-100 casi d'uso, con tali estremi aventi probabilitàmolto bassa).

Tra 540000 e 2600000 sloc, potrebbe esservi un mix dei Livelli 3 e 4, per un conteggio di casi d'uso tra 2 (tutto il Livello Level) e 60 (tutto il Livello 3).

Oltre i 2600000 sloc, il conteggio di casi d'uso a Livello 4 si alza a ~8.

Quanti casi d'uso sono sufficienti?

Ne derivano alcune osservazioni interessanti che supportano alcune regole fondamentali. Ricorre spesso la seguente domanda: "Quando diventa eccessivo il numero di casi d'uso?" La domanda è riferita alla cattura dei requisiti. La risposta sembra essere che più di ~70 casi d'uso , anche per quanto riguarda i sistemi più ampi, implicano una granularità eccessiva prima della progettazione. Tra 5-40 è accettabile, ma il numero in sé, senza considerare il livello, non può essere utilizzato per fare una stima della dimensione e dell'impegno. Si tratta del numero iniziale, adatto ad un livello specifico. Le centinaia di casi d'uso occorrono se un supersistema vasto viene scomposto in sistemi, più sottosistemi e così via. Se sono stati sviluppati casi d'uso finché il livello di classe non è stato raggiunto, il conteggio finale potrebbe ammontare a centinaia o addirittura migliaia (~600 per un progetto di 140 collaboratori-anno, o qualcosa come 15 punti di funzione per caso d'uso). Tuttavia ciò non avverrà come pura scomposizione del caso d'uso, indipendente dalla progettazione. Tali casi d'uso derivano dal processo descritto in Jacobson97, dove i casi d'uso a livello di sistema sono suddivisi in un comportamento allocato nei sottosistemi, per cui i casi d'uso a livello più basso possono essere scritti (con altri sottosistemi come attori).

Procedura della stima dell'impegno

Come si procede dunque nel fare una stima? Esistono dei prerequisiti: una stima basata su casi d'uso non può essere fatta senza una conoscenza del dominio del problema, senza avere già un'idea della dimensione del sistema proposta e senza un'idea dell'architettura, appropriate al punto in cui si sta facendo la stima.

La prima bozza di una stima può essere fatta utilizzando l'opinione di un esperto o in modo più formale con la tecnica Wideband Delphi (inventata dall'organizzazione Rand nel 1948, per una descrizione vedere Boehm81). Ciò consentirà all'estimatore di posizionare il sistema in una delle bande di dimensione nella Figura 3. Tale posizione suggerirà un range per il conteggio del caso d'uso e indica il livello di espressione (L1, L1/L2 e così via). L'estimatore deve poi decidere, sulla base dell'attuale conoscenza dell'architettura e del vocabolario del dominio, se i casi d'uso si adattino ad un livello, siano suddivisi discretamente o siano un insieme di livelli (nel modo in cui è espresso il flusso di eventi).

Da tali considerazioni dovrebbe trasparire anche se i dati sono patologici, ad esempio se la stima di Delphi ammonta a 600.000 righe di codice (o punto di funzione equivalente) ed è stata eseguita una breve progettazione, così che non si conosce ancora molto sulla struttura del sistema. La Figura 3 suggerisce che il conteggio del caso d'uso dovrebbe essere tra 2 (tutto il Livello 4) e 14 (tutto il Livello 3). Se il conteggio del caso d'uso ammonta a 100, i casi d'uso potrebbero essere stati scomposti prematuramente, oppure la stima di Delphi è molto lontana.

Continuando l'esempio: se il conteggio del caso d'uso reale è di 20 e l'estimatore decide che siano tutti a L3, che la lunghezza del caso d'uso è in media di 7 pagine e che il sistema sia di tipo di business complesso, allora le ore per caso d'uso (nella Figura 2) sono 20.000. Ciò dev'essere moltiplicato per 7/9 per rendere conto dell'apparente bassa complessità (basata sulla lunghezza del caso d'uso). L'impegno totale sarà dunque 20*20000*(7/9) = ~310.000 collaboratori-ore, o 2050 collaboratori-mesi. Secondo Estimate Professional, 600.000 righe di codice C++, per un sistema di business complesso, richiedono 1928 collaboratori-mesi. In questo esempio c'è dunque una sufficiente concordanza.

Se il conteggio del caso d'uso reale fosse 5 e l'estimatore decidesse che sono divisi 1 a L4 e 4 a Livello 3 e ancora che il caso d'uso a L4 risulta di 12 pagine mentre il caso d'uso a L3 ha una media di 10 pagine, l'impegno sarà dunque 1*250,000*12/9+4*21000*(10/9) = ~2800 collaboratori-mesi. Questo sembra suggerire che la stima di Delphi necessita di una revisione, pur considerando che una buona parte del sistema è compreso ancora ad un livello molto alto gli errori di confini restano troppi.

Se la stima di Delphi originale era stata di 100.000 righe di C++, l'indicazione dalla Figura 3 dice che i casi d'uso dovrebbero essere a L2 e dovrebbero essere circa 18. Se in realtà sono 20, come nel primo esempio, l'applicazione del metodo senza considerare il livello reale di caso d'uso darà un risultato difettoso, nel caso in cui la stima di Delphi sia in larga misura errata.

L'estimatore deve controllare che i casi d'uso siano realmente al livello di astrazione suggerito (L2) e che possano essere realizzati tramite una collaborazione di sottosistemi, e che i casi d'uso non siano realmente a L3, per quanto il metodo Wideband Delphi non sia così male (ad esempio, predire 100.000 mentre l'effettivo numero si avvicina a 600.000). Il punto è che tale metodo di valutazione non può procedere in sicurezza senza la costruzione di un'architettura concettuale o nozionale, che si allinei con il livello caso d'uso. Per un estimatore con molta esperienza nel dominio, il modello può essere mentale, per consentire un giudizio del livello da eseguire, per un estimatore e un team con minore esperienza, è consigliabile eseguire una modellazione strutturale per vedere come i casi d'uso possono essere realizzati ad un livello specifico.

Il conteggio di un'espressione varia del caso d'uso (cioè un mix di Livello N e Livello N+1) dovrebbe essere conteggiato come $n=8_{(\text{distanza frazionale tra i due livelli})}$ del tipo di caso d'uso al confine inferiore. Per cui un caso d'uso stimato al 50% L1 e 50% L2 dovrebbe essere conteggiato come $8_{0.5}$ = 3 casi d'uso a L1 per ottenere un conteggio complessivo. Un caso d'uso stimato al 30% tra L2 e L3 dovrebbe essere conteggiato come $8_{0.3}$ casi d'uso a L2 = 2 casi d'uso a L2. Un caso d'uso stimato al 90% tra L2 e L3 dovrebbe essere conteggiato come $8_{0.9}$= 7 casi d'uso a L2.

Modifica delle dimensioni della tabella

Vi sono comunque ulteriori modifiche da effettuare nelle figure individuali ore/uso per tenere in considerazione la dimensione generale, le figure di impegno sono adatte ad ogni livello nel contesto dei sistemi della stessa dimensione. Dunque a L1, nella Tabella 1, 55 ore per caso d'uso verranno applicate nel costruire un sistema di 7000 sloc. Il numero effettivo dipenderà dalla dimensione totale del sistema, quindi se il sistema da costruire è di 40.000 sloc e vi sono 57 casi d'uso a Livello 1 che lo descrivono, l'impegno non sarà di 55*57 ore per un sistema di business semplice, ma di $(40/7)^{0.11}$* 55 = 66 ore/caso d'uso. Ciò si basa sulla relazione COCOMO 2 della misura dell'impegno. secondo il modello COCOMO, Impegno = A * $(\text{dimensione})^{1.11}$, laddove:

- La dimensione è ksloc
- A avrà determinatori di costo fattorizzati
- I fattori di scala del progetto sono simbolici (con esponente 1.11)

Notare che questi calcoli potrebbero essere inseriti in un tool come Estimate Professional per eliminare il peso del calcolo, li mostrerò per precisione.

L'impegno per ksloc, o per unità se si vuole, equivale a A* $(\text{Dimensione})^{1.11}$/Dimensione, che dà A* $(\text{Dimensione})^{0.11}$, e un rapporto di impegno/unità della dimensione S1 all'impegno/unità della dimensione S2 è $(S1/S2)^{0.11}$.

Oltre alla stima di Delphi, la dimensione del sistema può essere calcolata in modo approssimativo dal conteggio del caso d'uso a vari livelli: se ci sono casi d'uso N1 a Livello 1, N2 a Livello 2, N3 a Livello 3 e N4 a Livello 4, la dimensione totale sarà [(N1/10)*7 + (N2/10)*56 + (N3/10)*448 + (N4/10)*3584] ksloc. Possiamo dunque calcolare i moltiplicatori dell'impegno per ogni sforzo per figure di caso d'uso, nella Tabella 1, dividendo tale dimensione totale per la dimensione ad ogni livello (in ksloc) mostrato nella colonna 1 della Tabella 1.

- Di conseguenza, a Livello 1 $(0.1*N1 + 0.8*N2 + 6.4*N3 + 51.2*N4)^{0.11}$
- A Livello 2 $(0.0125*N1 + 0.1*N2 + 0.8*N3 + 6.4*N4)^{0.11}$
- A Livello 3 $(0.00156*N1 + 0.0125*N2 + 0.1*N3 + 0.8*N4)^{0.11}$
- A Livello 4 $(0.00002*N1 + 0.00156*N2 + 0.0125* N3 + 0.1*N4)^{0.11}$

Chiaramente, a Livello 4 ad esempio, il numero dei casi d'uso a Livello 1 ha un debole effetto, se paragonato al numero del Livello 3 o 4.

Riepilogo

È stato presentato un framework per la stima, basato su casi d'uso. Per rendere più concreta la presentazione, sono stati scelti alcuni valori per i parametri di framework, che non contengono molti errori. Come sempre, tali congetture dovrebbero essere verificate nella realtà e i parametri rivalutati mentre si raccolgono i dati. Il framework prende in considerazione l'idea del livello, della dimensione e della complessità del caso d'uso per diverse categorie di sistema e non ricorre ad una scomposizione funzionale precisa. Per alleviare il peso del calcolo, è possibile costruire un front end in un tool come Estimate Professional che fornisca un metodo alternativo per immettere la dimensione, basato sul caso d'uso.

Per commenti e feedback sul presente white paper, contattare John Smith, jsmith@rational.com.

Riferimenti

1. Armour96: Experiences Measuring Object Oriented System Size with Use Cases, F. Armour, B. Catherwood, et al., Proc. ESCOM, Wilmslow, UK, 1996
2. Boehm81: Software Engineering Economics, Barry W. Boehm, Prentice-Hall, 1981
3. Booch98: The Unified Modeling Language User Guide, Grady Booch, James Rumbaugh, Ivar Jacobson, Addison Wesley, 1998
4. Cockburn97: Structuring Use Cases with Goals, Alistair Cockburn, Journal of Object-Oriented Programming, Sett-Ott 1997 e Nov-Dic 1997
5. Douglass99: Doing Hard Time, Bruce Powel Douglass, Addison Wesley, 1999
6. Fetcke97: Mapping the OO-Jacobson Approach into Function Point Analysis, T. Fetcke, A. Abran, et al., Proc. TOOLS USA 97, Santa Barbara, California, 1997
7. Graham95: Migrating to Object Technology, Ian Graham, Addison-Wesley, 1995
8. Graham98: Requirements Engineering and Rapid Development, Ian Graham, Addison-Wesley, 1998
9. Henderson-Sellers96: Object-Oriented Metrics, Brian Henderson-Sellers, Prentice Hall, 1996
10. Hurlbut97: A Survey of Approaches For Describing and Formalizing Use Cases, Russell R. Hurlbut, Technical Report: XPT-TR-97-03, http://www.iit.edu/~rhurlbut/xpt-tr-97-03.pdf
11. Jacobson97: Software Reuse - Architecture, Process and Organization for Business Success, Ivar Jacobson, Martin Griss, Patrik Jonsson, Addison-Wesley/ACM Press, 1997
12. Jones91: Applied Software Measurement, Capers Jones, McGraw-Hill, 1991
13. Karner93: Use Case Points - Resource Estimation for Objectory Projects, Gustav Karner, Objective Systems SF AB (copyright di proprietà di Rational Software), 1993
14. Lorentz94: Object-Oriented Software Metrics, Mark Lorentz, Jeff Kidd, Prentice Hall, 1994
15. Major98: A Qualitative Analysis of Two Requirements Capturing Techniques for Estimating the Size of Object Oriented Software Projects, Melissa Major and John D. McGregor, Dept. of Computer Science Technical Report 98 002, Clemson University, 1998
16. Minkiewicz96: Estimating Size for Object-Oriented Software, Arlene F. Minkiewicz, http://www.pricesystems.com/foresight/arlepops.htm, 1996
17. Pehrson96: Software Development for the Boeing 777, Ron J. Pehrson, CrossTalk, Gennaio 1996
18. Putnam92: Measures for Excellence, Lawrence H. Putnam, Ware Myers, Yourdon Press, 1992
19. Rechtin91: Systems Architecting, Creating & Building Complex Systems, E. Rechtin, Prentice-Hall, 1991
20. Royce98: Software Project Management, Walker Royce, Addison Wesley, 1998
21. RUP99: Rational Unified Process, Rational Software, 1999
22. Stevens98: Systems Engineering - Coping with Complexity, R. Stevens, P. Brook, et al., Prentice Hall, 1998
23. Thomson94: Project Estimation Using an Adaptation of Function Points and Use Cases for OO Projects, N. Thomson, R. Johnson, et al., Proc. Workshop on Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, OOPSLA '94, 1994

Rational®
the software development company

Sedi principali:

Rational Software
18880 Homestead Road
Cupertino, CA 95014
Tel: (408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
Tel: (781) 676-2400

Numero verde: (800) 728-1212
E-mail: info@rational.com
Sito Web: www.rational.com
Sito internazionale: www.rational.com/worldwide