Metriche Software

Riferimenti

- I. Sommerville – Ingegneria del Software – 8a edizione – Cap. 27
- R. Pressman – Principi di Ingegneria del Software – 5a edizione italiana - Cap. 17
- Ghezzi, Jazayeri, Mandrioli, Ingegneria del Software, 2a edizione, Capitolo 2

- N. Fenton, Software Metrics - A Rigorous and Practical Approach, 1997

Misurare il software: una premessa

- L'Ingegneria del Software non studia le leggi quantitative della fisica, ma processi e prodotti legati ad attività umane.
- Non è possibile effettuare misure assolute, ma è necessario ricorrere a misure indirette.

  – *Negli ultimi trent'anni molti studiosi hanno tentato di sviluppare una singola metrica che offrisse una misura globale della complessità del software*

    – Fenton paragona questi tentativi alla ricerca del "sacro Graal"

    …

Misurare il software

- La misurazione del software ha lo scopo di assegnare un valore ad un attributo caratterizzante un processo o prodotto software
  - consente una comparazione obiettiva tra prodotti/processi
  - rende misurabili processi, risorse, prodotti rilevanti della Ingegneria del Sw

- Sebbene molte aziende produttrici di software utilizzino procedure di misurazione, l'uso sistematico della misurazione del software non è ancora una pratica comune
  - Ancora pochi standard in questa area

Metrica

Metrica: misura quantitativa del grado di possesso di uno specifico attributo da parte di un sistema, un componente, o un processo [IEEE Std 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology]

- Es: numero di linee di codice di un modulo, numero di casi d'uso di un'applicazione, ...

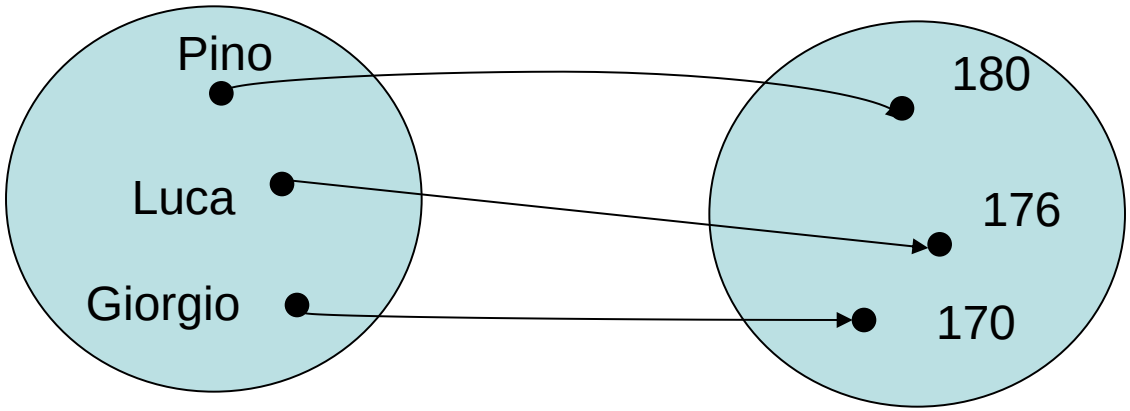
- Molto spesso, metrica è usato come sinonimo di misura o misurazione
- la definizione di metrica include procedure e modalità di misurazione.

Misurazione e Misura

- La **Misurazione** è il processo mediante il quale si assegnano numeri o simboli ad attributi di entità del mondo reale in modo tale da descriverle secondo regole chiaramente definite [Fenton]
  - **L'entità** rappresenta l'oggetto che si vuole sottoporre a misurazione.
  - **L'attributo** dell'entità, invece, è l'aspetto di tale oggetto che interessa descrivere o rappresentare.
  - L'insieme dei simboli o valori che si possono assegnare all'attributo costituisce la "forma" di rappresentazione dell'attributo, o "**scala dei valori**"
- Una **misura** è una funzione che mappa un insieme di oggetti in un altro insieme di oggetti (tipicamente numeri o insiemi di numeri, ma anche simboli)
  - La Misurazione è l'attività generale, una Misura è l'effettiva assegnazione di valori.

Un esempio di Misura di attributi

Entità= persone; Attributo= Altezza;



Misura dell'altezza di persone come funzione di mapping fra due insiemi

Misura e Metrica

- Spesso i termini metrica e misura (e anche misurazione) sono usati come sinonimi
  - Dai testi classici della misura
    - *Una misura è una empirica oggettiva assegnazione di un numero (o simbolo) ad una entità al fine di caratterizzarne uno specifico attributo*
  - Un tentativo di distinzione:
    - *Una metrica caratterizza con valori (numericamente o con simboli) attributi semplici*
    - *Una misura è una funzione di metriche che può essere usata per valutare o predire attributi più complessi*

Attributi

Attributi esterni:

- attributi di un'entità che sono visibili e di interesse per *l'utente* del prodotto software; descrivono l'aspetto esterno di un'entità e il loro rapporto con l'ambiente in cui vengono usate, indipendentemente dalla implementazione;
  - *ad esempio: facilità d'uso; portabilità, efficienza, affidabilità*

Attributi interni:

- attributi di un'entità che sono visibili e di interesse del *produttore*, i cui valori dipendono dalla implementazione;
  - *ad esempio: modularità, strutturazione, tracciabilità, testabilità, dimensione, complessità*

Misure e metriche

- Misura diretta: **la misura di un attributo che non dipende da quella di altri attributi**
- Misura indiretta: **la misura di un attributo che dipende da quella di almeno un altro attributo**

- Metriche: **tutto ciò per cui possiamo fare misure dirette**
  - Misure dirette ≡ Metriche

- Misure: **tutto ciò per cui dobbiamo fare misure indirette**
  - Misure indirette ≡ f(metriche)

- ad attributi semplici sono associate le metriche
- ad attributi complessi sono associate le misure

Entità

Entità di interesse per la misurazione nella IS sono:

- **Prodotti:**
  - tutto ciò che viene prodotto nel CVS (documentazione, software, test data, …);
- **Processi (attività)**
  - produzione specifiche, progettazione, codifica, testing, manutenzione, quality assurance, riuso, reengineering, …;
- **Risorse:**
  - hardware, software, documentazione (della risorsa), risorse umane, banche dati, conoscenza, ….

Esempi di Entità e attributi

	ENTITA	ATTRIBUTI INTERNI	ATTRIBUTI ESTERNI
Prodotti	Specifica	Dimensione, riuso, modularizzazione, funzionalità, correttezza sintattica	Comprensibilità, manutenibilità
	Progetto	Dimensione, riuso, accoppiamento, coesione, modularizzazione, funzionalità	Qualità, comprensibilità, manutenibilità
	Codice	Dimensione, riuso, accoppiamento, modulariz-zazione, funzionalità, complessità algoritmica	Affidabilità, manutenibilità, usabilità
	Dati test	Dimensione, copertura	Qualità, affidabilità
Processi	Specifica	Durata, sforzo, n. cambiamenti nei requisiti	Qualità, costo, stabilità
	Progettazione	Durata, sforzo, numero di errori individuati	Costo, stabilità
	Test	Durata, sforzo, numero di errori trovati	Costo, controllabilità
Risorse	Personale	Età,costo	Produttività, esperienza
	Team	Dimensione, livello di comunicazione, strutturazione	Produttività
	Software	Costo, dimensione	Affidabilità
	Hardware	Costo, prestazioni	Affidabilità

Scale di valori

Una Scala di valori è l'insieme:

- dei numeri / simboli da assegnare ad un attributo di un'entità
- delle relazioni tra tali numeri / simboli
- Esistono 5 tipologie di scala:

1. Nominale

- La relazione tra i valori consente una semplice *classificazione* degli oggetti della misurazione, ma non ci indica nulla sulla relazione fra di essi
- Esempio:
  - *la classificazione degli errori di programmazione in lessicali, sintattici e semantici ci consente di suddividerli ma non di sapere se un errore sia più o meno grave di un altro;*
- in termini più analitici questo tipo di scala coincide con l'introduzione nell'insieme di partenza di classi di equivalenza
- Operazioni applicabili: Moda

Scale di valori

2. Ordinale

- si introduce una relazione d'ordine fra gli oggetti, per cui si è in grado di determinare le posizioni relative degli oggetti (cioè dire se uno venga prima di un altro);
- non si è però in grado di quantificare la distanza fra gli oggetti
- Operazioni applicabili: Mediana
- Esempio:
  - *in una scala di valutazione (sufficiente, buono, ottimo) possiamo dire che buono è peggiore di ottimo ma non di sufficiente*

Scale di valori

3. **Intervalli**:
   - Si è in grado di misurare la distanza fra gli oggetti del dominio, e non solo posizionarli tra di loro come con le scale ordinali
   - Operazioni applicabili: Media Aritmetica
   - Esempio:
     - *la classificazione del rendimento scolastico con "6", "8" e "10" in sostituzione di "sufficiente", "buono" e "ottimo", rispettivamente, consente non solo di dire che "6" precede "8", ma anche di quanto*

4. **Ratio**
   - introduce l'elemento zero che rappresenta l'assenza totale dell'attributo che si sta misurando nell'entità sottoposta a misurazione. L'introduzione dello zero, inoltre, conferisce al valore di un attributo il senso di positività, negatività o nullità.
   - Esempio:
     - *la misura di una temperatura, quantificata con un numero maggiore, minore, o uguale a zero.*

Scale di valori

5. Assoluta

- esiste una strategia di conteggio per la quale possiamo assegnare ad ogni oggetto un numero univocamente determinato.
- La scala assoluta è usata per quegli attributi di un oggetto che richiedono un semplice conteggio di elementi.
- Esempio:
  - *si consideri l'entità "studente" e si voglia quantificare il suo attributo "numero di esami superati". La misura in questo caso consiste nel semplice conteggio degli esami superati da uno studente; il totale conteggiato rappresenta lo studente in esame, visto attraverso l'attributo "numero di esami superati".*

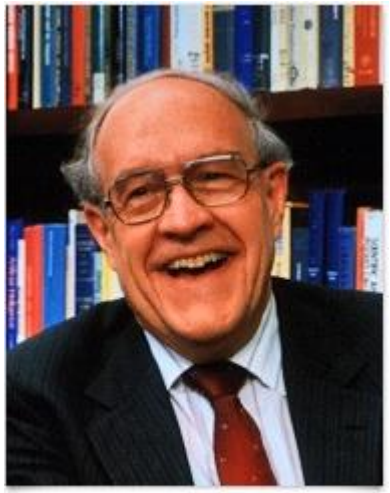
Caratteristiche di metriche software efficaci

- Semplici e Calcolabili
- Convincenti a livello empirico ed intuitivo.
- Coerenti e obiettive.
- Coerenti nell'uso di unità e dimensioni.
- Indipendenti dal linguaggio di programmazione.
- Devono essere un meccanismo efficace per un feedback sulla qualità.
  - "L'esperienza insegna che la metrica di un prodotto viene utilizzata solo se è intuitiva e facile da calcolare. Se occorre svolgere decine di calcoli complessi, è improbabile che tale metrica venga largamente adottata" [Pressman]
  - Non tutte le metriche soddisfano tutte le caratteristiche di qualità indicate.

Metriche di sforzo

- Per sforzo (effort) si intende la misura del quantitativo di lavoro umano necessario per svolgere un lavoro relativo ad un processo di sviluppo software
- L'unità di misura più comunemente usata è il mese/uomo (man-month) e tutti i suoi multipli e sottomultipli
  - Un mese/uomo si definisce, teoricamente, come il quantitativo di lavoro che un singolo dipendente può svolgere in un mese lavorativo

The Mythical Man-Month


Frederick Brooks
*1931

- Problemi della definizione del mese-uomo
  - Ogni persona ha una sua diversa produttività, che può variare nel tempo (con l'età e la conoscenza, ad esempio)
  - La produttività di una persona dipende dalla complessità del problema da risolvere
  - Aumentando la dimensione del gruppo di lavoro aumenta proporzionalmente la produttività?
    - *Se una persona può svolgere 1 man-month di lavoro in un mese, quanto lavoro potranno svolgere 10 persone nello stesso mese?*
- Tutti questi problemi furono espressi compiutamente per la prima volta da Brooks nel famoso libro The Mythical Man-Month: Essays on Software Engineering, nel 1975
- In conclusione: si tratta di una metrica di facile misura (ma solo a posteriori) ma di difficilissima stima

Brooks (1975). *The Mythical Man-Month*. Addison-Wesley.

Una Classificazione di Metriche software relative al prodotto

- **Metriche per il Modello di Analisi** :
  - Funzionalità fornita
  - Dimensioni del sistema (in termini di informazioni presenti sul modello di analisi)
  - Qualità delle specifiche
- **Metriche per il Modello di Design**:
  - Metriche dell'architettura
  - Metriche a livello dei componenti
  - Metriche della progettazione dell'interfaccia
  - Metriche specializzate relative al design object-oriented
- **Metriche relative al codice sorgente**
  - Metriche di Halstead
  - Metriche di complessità
  - Metriche di lunghezza
- **Metriche di testing**
  - Metriche sulle istruzioni e ramificazioni
  - Metriche relative ai difetti
  - Test dell'efficacia
  - Metriche sul processo

Metriche per la fase di Analisi

- Che tipo di metriche si possono usare in fase di analisi?
  - Metriche che consentono al management di monitorare e controllare Costi, Scheduling, Qualità
  - Ad esempio: Metriche aventi lo scopo di poter "prevedere" quale sarà la "taglia" del processo software all'inizio del suo ciclo di vita, in modo da poter dimensionare le risorse umane e temporali da dedicare ad esso.

- Metriche basate sulle funzionalità
  - *Function Points (FP)*

- Metriche per la qualità delle specifiche

Function Point Analysis

- Tecnica proposta da **Albrecht** [*Measuring Application Development Productivity, 1979*]
- Approccio indipendente dal linguaggio di programmazione per misurare le funzionalità del sistema.
- Può essere usata per:
  - Stimare il costo richiesto per programmare, testare il software
  - Prevedere il numero di errori che si rileveranno durante il testing
  - Prevedere il numero di componenti o di LOC del sistema

Cosa sono i Function Point (FP)

- I FP sono una misura di funzionalità basata su entità logico-funzionali che l'utente facilmente comprende (es. Input, Output,etc.)
- I FP sono pertanto indipendenti dal linguaggio di programmazione, quindi la produttività può essere confrontata tra diversi linguaggi
- Un FP non è una singola caratteristica ma una combinazione di caratteristiche del sistema
  - *Destinati a supportare stime, pianificazioni relative a sistemi Sw*
  - *Nati per essere applicati a Sistemi Sw di tipo Business*
  - *Misura che può essere effettuata 'presto' nel CVS*
  - *Misura solo i requisiti funzionali*

Formulazione iniziale del metodo dei Function Point

- I FP vengono derivati usando una relazione empirica che si basa su:
  - Misure dirette (espresse in valori interi) del Dominio delle informazioni del software
  - Valutazione della complessità del software

- Le Misure del Dominio delle Informazioni sono catturate con riferimento al Confine (Boundary) del sistema software.

Boundary

- Secondo il concetto di Boundary, esiste una linea chiusa che definisce il confine del sistema software cui è rivolta la misura.
- Con riferimento al confine, sono misurate o valutate le seguenti caratteristiche del programma:
  - External input (EI)
  - External output (EO)
  - External inquiry (EQ)
  - Internal Logical File (ILF)
  - External Interface File (EIF)

External Input (EI)

- Un EI rappresenta un tipo di input unico (proveniente dall'esterno dell'applicazione) fornito o da un utente o da un altro sistema, che verrà elaborato dall'applicazione.
  - Un input è unico se il formato è unico, o la logica con cui viene elaborato è unica.
- Questi input tipicamente devono aggiungere, cambiare o cancellare dati in un File Logico Interno (ILF).
- I dati possono essere informazioni di controllo o informazioni del dominio applicativo (business information). Se i dati sono informazioni di controllo non devono aggiornare nessun ILF.

External Output (EO)

- Un EO è un tipo di dati unico che viene creato nell'applicazione ed è destinato all'utente, o ad altre applicazioni, attraversando il confine dell'applicazione.
  - Un output è unico se il formato o la logica elaborativa è unica.
- Un EO consiste in report, schermate, file, messaggi d'errore…
- Un EO può addizionalmente aggiornare un ILF.
- Questi reports o files sono creati a partire da uno o più ILF e EIF.

External Inquiry (EQ)

- Una interrogazione esterna EQ è definita come una coppia input/output unica, dove un input online produce la generazione di una risposta immediata del software, sotto forma di output online.
  - Un'interazione è unica se il formato o la logica di elaborazione è unica.
- Una EQ ricerca dati o informazioni di controllo (da uno o più ILF e/o EIF) per una risposta immediata.
- Il processo di input non deve aggiornare nessun ILF, mentre il processo di output non contiene dati "derivati"
- Se c'è un coinvolgimento di updating del file logico interno o una produzione di dati derivati o calcolati, deve essere considerato un input seguito da un output.

Internal Logical File

- Un ILF è un raggruppamento unico di dati logicamente correlati, o di informazioni di controllo, identificabile dall'utente, usato ed aggiornato dall'applicazione.

- I dati relativi non attraversano il confine, ma risiedono internamente al confine dell'applicazione e sono gestiti attraverso gli external input (EI).

External Interface File

- Una EIF è un unico gruppo di dati logicamente correlati, o di informazioni di controllo, identificabile dall'utente che viene usato dall'applicazione.

- I dati relativi attraversano il confine e risiedono interamente all'esterno del confine dell'applicazione e sono mantenuti da un'altra applicazione.

- Un EIF è un ILF di un'altra applicazione.

Calcolo dei Function Point

- Esistono diverse teorie, più o meno complesse per calcolare i FP in funzione del conteggio degli elementi prima descritti
  - Il livello di complessità di ciascun elemento individuato per ognuna delle 5 precedenti categorie viene classificato in:
    - *Basso (semplice)*
    - *Medio*
    - *Alto (complesso)*
  - Ad ognuno dei livelli è associato un peso che varia da 3 (più semplice) a 15 (più complesso).

Matrice dei pesi: un esempio

	Componenti	Livello di complessità		
		Basso	Medio	Alto
Transactional function types	Ext. Input (EI)	3	4	6
	Ext.Output (EO)	4	5	7
	Ext. Inquiry (EQ)	3	4	6
Data function types	Internal Logical files (ILF)	7	10	15
	External Interface files (EIF)	5	7	10

UFC e FP

Una prima stima dei FP è detta **UFC** (Unadjusted Function Point Count):

$$UFC = \sum(\text{number of elements of given type}) \times (\text{weight})$$

Lo UFC è poi corretto da un fattore moltiplicativo TFC compreso tra 0.65 e 1.35

DFP = UFC * TCF

(Delivered Function Point)

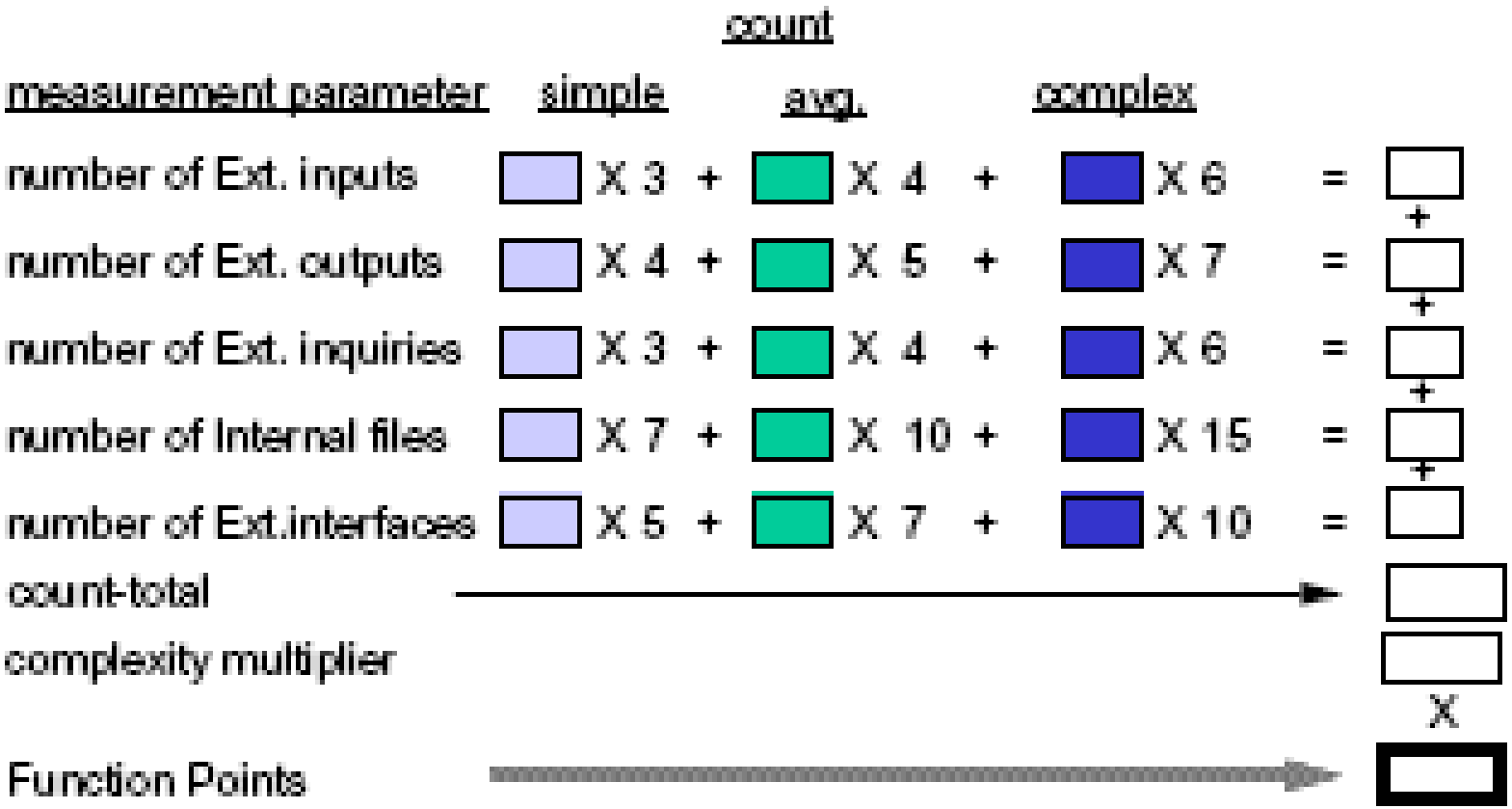
TCF- Fattore di Complessità Totale

- Il TCF è calcolato sulla base di 14 Fattori di complessità (CF), valutati su una scala da 0 (ininfluente) a 5 (essenziale):

1. Il sistema richiede backup e recovery affidabili?
2. Sono richieste comunicazioni specializzate per trasferire dati da/verso l'applicazione?
3. Vi sono funzionalità richiedenti elaborazioni distribuite?
4. Le prestazioni sono un elemento critico?
5. Il software funzionerà in un ambiente operativo esistente già pesantemente utilizzato?
6. Il sistema richiede inserimenti online di dati?
7. L'input online di dati richiede che la transazione relativa sia progettata su più schermate o più operazioni?
8. Il file principali IFL sono aggiornati online?
9. I dati di I/O, i file, le interrogazioni sono complesse?
10. L'elaborazione interna è complessa?
11. Il codice è progettato/scritto per essere riusabile?
12. Il progetto comprende anche l'installazione e la conversione?
13. Il software è stato progettato per essere installato presso diversi utenti?
14. Il software è stato progettato per facilitare le modifiche e per un semplice uso da parte dell'utente finale?

$$TCF = 0.65 + 0.01 * \Sigma_{i=1..14} CF_i$$

Schema di calcolo FP



Applicazione dei FP

- Dalla loro definizione, molti gruppi si sono internazionalmente occupati di poter fornire definizioni e metodologie di stima dei valori dei FP che fossero il più soddisfacenti possibili. Ad esempio:
  - International Function Point User Group, http://www.ifpug.org/
  - Gruppo Utenti Function Point Italia, http://www.gufpi-isma.org/
- Ciò nonostante, le metodologie che portano alla migliore stima sono quelle che sono state pensate, provate e migliorate con l'utilizzo nell'ambito di un gruppo di lavoro stabile e nell'utilizzo di cicli di vita e tecnologie ben stabilizzati

Utilizzi dei FP

- Per stimare la dimensione finale del codice:
  - Studi hanno cercato di rilevare una correlazione tra FP e numero di LOC, variabile a seconda del linguaggio di programmazione.
  - Es. COBOL: 1 FP -> 100 LOC
  - PL1 : 1FP -> 80 LOC
  - OO lang : 1 FP ->60 LOC
- Per stimare lo sforzo (in ore/uomo) di sviluppo:
  - Es. se 1 mese/uomo ->12 FP , allora …
- Per valutare la completezza del testing
  - studi hanno misurato la correlazione fra FP e numero di difetti scoperti durante il testing

Use Case Point

- Una proposta più recente sposta l'attenzione verso la misura degli use case (Karner, 1993)
- Nella tecnica degli use case point vengono tenuti in conto nella misura:
  - *Casi d'uso*
  - *Attori*
  - *Scenari*
- La tecnica di stima è simile a quella dei FP

Gautam Banerjee,Use case points, http://www.bfpug.com.br/Artigos/UCP/Banerjee-UCP_An_Estimation_Approach.pdf

Use Case Point

- Gli attori sono pesati in base alla loro complessità (altri programmi, utente esperto, utente inesperto)
- I casi d'uso sono pesati in base alla complessità delle interazioni (in funzione del numero di interazioni dei suoi scenari, ad esempio)
  - Nella complessità del caso d'uso si tiene conto anche della possibilità di riutilizzare software esistente
- Si aggiusta il conteggio con un fattore tecnico dipendente dai requisiti non funzionali richiesti e da condizioni ambientali legate al processo e agli strumenti di sviluppo
- Si calcola una stima generale dello sforzo in giorni/uomo

Vantaggi e svantaggi dei FP

Vantaggi:

- Indipendenti dal linguaggio
- Ottime indicazioni per applicazioni di elaborazione dati, che usano linguaggi convenzionali o non procedurali
- Basati su quei dati che hanno la maggior probabilità di essere noti all'inizio di un progetto

Svantaggi

- Soggettività nell'assegnazione dei pesi
- Dati sul dominio delle informazioni possono essere difficili da reperire
- Nessuna valutazione della complessità dell'algoritmo (a parità di pochi input e output potrebbe essere banale ed estremamente complicato)
- I FP non hanno un diretto significato fisico

Feedback e Tuning

- La stima dello sforzo con FP e UCP può diventare più affidabile con l'esperienza che si acquisisce nel loro utilizzo, specie in un ambiente di sviluppo invariato

- Al termine di ogni progetto è possibile cercare di ricavare dei feedback per il tuning dei parametri del metodo di calcolo dei FP

Metriche per la qualità delle specifiche

- Sono stati proposti alcuni **attributi di qualità** dello SRS (non ambiguità, completezza, correttezza, comprensibilità,verificabilità, coerenza interna ed esterna, realizzabilità, concisione, tracciabilità, modificabilità, precisione e riusabilità)
- Un metodo per valutare tali attributi:
  - *n_R requisiti, di cui n_f funzionali e n_{nf} non funzionali : $n_R=n_f+n_{nf}$*
  - *Una metrica proposta per ognuno degli attributi di qualità:*
    - Ad esempio, per l'assenza di ambiguità:

      $Q_1 = n_{ui}/n_R$

      Rapporto tra il numero di requisiti che non risultano ambigui (secondo i revisori dell'SRS) e il numero di requisiti funzionali
- Metriche analoghe sono proposte per tutti gli altri attributi di qualità dell'SRS

Metriche di progetto architetturale

- Si concentrano sulle caratteristiche dell'architettura del software, non tenendo conto della struttura interna dei singoli moduli.
- Si basano sull'analisi di modelli di progetto nei quali si evidenziano i moduli del sistema (qualora esso sia scomponibile) e i dati che vengono scambiati tra essi.
- Molto semplici da misurare, ma poco affidabili in quanto a legame con lo sforzo effettivo legato allo sviluppo del sistema.

Esempi di Metriche di Progetto Architetturale

- Architectural design metrics (*applicabili a moduli di architetture gerarchiche)*
  - **Structural complexity = f(fan-out)**
  - **Data complexity(mi) = f(input & output variables, fan-out)**
  - **System complexity = f(structural & data complexity)**

- Henry e Kafura (HK) metric**:** complessità architetturale vista come funzione del fan-in e fan-out dei moduli

- Morphology metrics (Fenton): una funzione del numero dei moduli e del numero delle interfacce tra i moduli

Metriche per il Design Object Oriented

- Peculiarità Sistemi O-O
  - Localizzazione: il modo con cui le informazioni e loro elaborazioni sono 'confinate' in un sistema
    - Nei sistemi Function oriented le informazioni sono localizzate intorno alle funzioni (moduli procedurali)
    - Nei sistemi Object Oriented sono concentrate incapsulando dati e procedure che li elaborano
  - Incapsulamento:
    - *una delle conseguenze sulle metriche è che l'unità da considerare è l'oggetto piuttosto che il sotto-programma*
  - Information hiding:
    - *influisce sull'accoppiamento fra oggetti*
  - Inheritance: *varie forme di influenza sul progetto*
  - Abstraction:
    - *possibilità di considerare gli oggetti da vari livelli di astrazione*

Metriche orientate alle classi

- Metriche di Chidamber and Kemerer **[A Metrics Suite for Object-Oriented Design, 1994]:**
  - weighted methods per class (WMC)
  - depth of the inheritance tree (DIT)
  - number of children (NOC)
  - coupling between object classes (CBO)
  - response for a class (RFC)
  - lack of cohesion in methods (LCOM)

Weighted Method per Class (WMC)

- **WMC è la somma pesata dei metodi di una classe**
  - Il peso di un metodo è dato da un fattore di complessità a scelta.
    - Il fattore di complessità dei metodi può essere una metrica di complessità proposta per le funzioni (ad esempio la complessità di Mc Cabe)
    - Al crescere di WMC aumenta la complessità della classe e diminuisce la speranza di poterla riusare.
    - Problema: come tener conto della complessità dei metodi ereditati?

Profondità dell'albero di ereditarietà di una classe- DIT

- **Distanza massima di un nodo (una classe) dalla radice dell'albero rappresentante la struttura ereditaria**
  - Maggiore è la profondità della classe nella gerarchia, maggiore è il numero di metodi che essa può ereditare, rendendo più complesso predire il suo comportamento.
  - Alberi di ereditarietà con elevata profondità possono aumentare la complessità del progetto (più classi e metodi sono coinvolti)
  - Maggiore è la profondità di una classe in una gerarchia, maggiore è il riuso potenziale dei metodi ereditati.
  - Ma i metodi della superclasse devono essere nuovamente testati per ciascuna sottoclasse.

Numero di figli - NOC

- **Numero di sottoclassi, figlie di una super-classe**
  - Al crescere del NOC, cresce il livello di riuso (NOC è dunque un indicatore di Riuso) …
  - ma tende ad 'evaporare' l'astrazione della Classe madre.
  - aumenta la possibilità che una Classe figlia non sia membro appropriato della madre
  - Al crescere del NOC, cresce la quantità di test case necessari a testare ogni figlia.

Risposte per classe RFC

- **Insieme dei metodi che possono essere potenzialmente eseguiti in risposta ad un messaggio ricevuto da un oggetto della Classe**
  - Include l'insieme dei metodi della classe e di tutte le classi a cui essa invia messaggi.
  - RFC è un indicatore del 'volume' di interazione fra classi
  - A valori elevati di RFC:
    - *aumenta la complessità progettuale della classe*
    - *cresce lo sforzo per il testing*

Accoppiamento tra le classi - CBO

- **Numero di collaborazioni di una classe, ovvero numero di altre classi cui essa è accoppiata**
  - L'accoppiamento può avvenire a seguito di lettura/modifica attributi, chiamata metodi, instanziazione oggetti
  - Eccessivo accoppiamento è negativo per la modularità ed il riuso;
  - Maggiormente indipendente è una classe più facilmente è riusabile;
  - Per migliorare la modularità, l'accoppiamento va tenuto al minimo; esso influisce sull'impatto di modifiche in altri moduli
  - Valori elevati del CBO complicano testing e modifiche.

Mancanza di coesione nei metodi - LCOM

- La coesione di un metodo esprime la proprietà di un metodo di accedere in maniera esclusiva ad attributi della classe.
- La Mancanza di Coesione deriva dalla presenza di più metodi che accedono ad attributi comuni.

- Se ogni attributo è acceduto da un solo metodo, allora LCOM=0 (massima coesione dei metodi).
- Altrimenti, LCOM >0 ed aumenta quanto più I metodi vanno ad operare sugli stessi attributi, rendendo quindi difficile il controllo di tutte le interazioni tra i metodi dovuti all'uso degli attributi comuni.

- Quindi se LCOM è elevato, I metodi potrebbero essere molto accoppiati tra loro attraverso gli attributi, ed aumenta la complessità del design della classe.

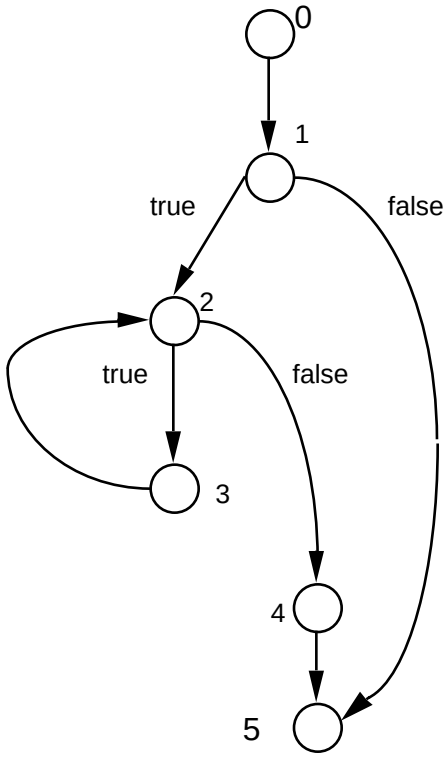
Metriche di design a livello dei componenti

- Metriche che valutano caratteristiche interne dei moduli, quali **coesione**, **coupling** e **complessità**. Ad esempio:
  - Le metriche di **coesione** di Bieman [Measuring Functional Cohesion, 1994] si basano sull'analisi delle funzioni e dei dati elaborati da un modulo, cercando di stabilire se il modulo svolge funzioni che operano sugli stessi dati (buona coesione), oppure su dati disgiunti (scarsa coesione)
  - La Metrica di **accoppiamento** proposta da Dhama [Quantitative Models of Cohesion and Coupling in Software, 1995] tiene conto del flusso di dati e di controllo fra i moduli, dell'accoppiamento a variabili globali, e dell'accoppiamento ambientale (mediante fan-in e fan-out) fra moduli.
  - Le metriche di **complessità** misurano in vario modo la complessità del CFG.

Metrica di McCabe

- Metrica strutturale relativa al Control Flow di un programma G
- E' detta anche numero ciclomatico V(G)
- Rappresenta la complessità logica del programma e quindi lo sforzo per realizzarlo (o per comprenderlo)
- per programmi strutturati V(G) è uguale al n.ro di predicati aumentato di uno:
  - V(G) = P + 1
    - *P: n.ro di predicati (semplici ed a due vie) in G*
    - *Predicati: decisioni - semplici, ovvero non composte con connettivi logici - in strutture di controllo (if-then-else, repeat-until, whiledo,…);*

Complessità ciclomatica di un programma



Complessità ciclomatica del programma è 3

Dato un grafo G di *n nodi* ed *e archi* il numero **ciclomatico** è dato da:

V(G) = *e*- *n*+ *2*
oppure:
V(G)= *d* + *1*
d= # nodi branch

V(G)= 3 =>3 cammini indipendenti

c1= 0-1-2-4-5
c2= 0-1-**2-3-2**-4-5
c3= 0-**1-5**

Cammini linearmente indipendenti

- V(G) rappresenta il n° di cammini linearmente indipendenti del Control Flow Graph nella teoria dei grafi V(G) rappresenta il numero di circuiti linearmente indipendenti che congiungono il nodo iniziale con quello finale
- Un cammino si dice indipendente se introduce almeno un nuovo insieme di istruzioni o una nuova decisione, rispetto ad un altro;
- In un CfG un cammino è indipendente se attraversa almeno un arco non ancora percorso dagli altri già considerati.
- L'insieme di tutti i cammini linearmente indipendenti di un programma formano i cammini di base;
- Tutti gli altri possibili cammini nel CfG sono generati da una combinazione lineare di quelli di base.
- Dato un programma, l'insieme dei cammini di base non è unico.

Metriche per il Codice Sorgente

- Linee di Codice
- Metriche di Halstead

Lines of Code (LOC)

- E' una metrica dimensionale
- Misura la lunghezza di un programma con il numero di linee di codice
- Esistono diverse possibili definizioni
  - qualsiasi linea di codice, inclusi i commenti;
  - tutte le linee di codice, esclusi i commenti (in questo caso si distinguono in Non Comment LOC (NCLOC) e Effective LOC (ELOC)
  - solo le istruzioni eseguibili (EXLOC), con anche le Data Declaration LOC (DDLOC), cioè le istruzioni dichiarative dei dati.

- La definizione più accettata è la seguente:
  - Una linea di codice è ogni linea di testo di un programma che non sia bianca o un commento, indipendentemente dal numero di istruzioni in essa inclusa. In particolare, tali linee di codice possono contenere intestazioni di unità di programma, dichiarazioni ed istruzioni esecutive

Osservazioni sulle LOC

- I commenti sono una parte importante del processo di sviluppo di un programma, per cui ignorarle, e quindi non invogliare il programmatore ad inserirle, può essere deleterio; ovviamente vanno prodotti in modo consistente (e non inseriti all'ultimo solo per motivi di tariffazione).
  - Le Comment LOC (CLOC) possono essere valutate in base alla loro importanza
    - *differenza fra commenti inseriti utilizzando metodi formali (per cui, utilizzando appositi tool, si possa effettuare un testing automatico del programma) e commenti inseriti all'ultimo momento prima della consegna.*
- Alcune misure derivate dalle LOC sono:
  - Densità di commento = CLOC/LOC
  - Effort = $5.2*KLOC^{0.91}$
    - *con Effort misurato in mesi/uomo*

Vantaggi e svantaggi

Vantaggi:

- Molto semplici da calcolare automaticamente
- Molto intuitive

Svantaggi

- Dipendenti dal linguaggio di programmazione
  - *Studi empirici sono stati svolti per valutare la dipendenza del numero di LOC dal linguaggio, a parità di algoritmo*
- Dipendenti dallo stile di programmazione
  - *Bisogna definire uno standard di formattazione del testo sorgente cui conformarsi, oppure uno strumento di formattazione automatica*
  - *Facilmente "falsificabili"!*
- Diventano una misura inconsistente in presenza di generatori automatici di codice

Altri usi delle LOC

- Le LOC sono utilizzate per misurare indirettamente altri attributi
  - Esempi:
    - *la probabilità di presenza di errori nel programma (cresce con la lunghezza)*
    - *il tempo per produrlo*
    - *la produttività (o il compenso) di un programmatore.*
    - *stima dei costi di un progetto SW*
- Nelle LOC bisognerebbe contare anche tutte le linee di codice non direttamente legate ai prodotti consegnati, ad esempio:
  - Codice per il testing
  - Codice per I prototipi

Strumenti a supporto

Alcune estensioni di Eclipse in grado di calcolare la maggior parte di queste metriche strutturali:

- Eclipse Metrics
- Stan4J (già visto per l'analisi delle dipendenze)
- Google CodePro Analytix

CodePro Analytix

- Plug-in multifunzionale per Eclipse offerto da Google
  - https://developers.google.com/java-dev-tools/codepro/doc/
  - Scaricabile direttamente da:
    - http://dl.google.com/eclipse/inst/codepro/latest/3.7
    - Aggiornato ad Eclipse Indigo; dovrebbe funzionare anche per le versioni successive
- Tutorial e documentazione accessibili da:
  - https://developers.google.com/java-dev-tools/codepro/doc/

Metrics

- CodePro è in grado di calcolare un notevole quantitativo di metriche di prodotto
  - Metriche dimensionali (classiche e Object Oriented)
  - Metriche di complessità ciclomatica
  - Metriche di coesione e accoppiamento
  - Metriche di Halstead
  - Metriche di ereditarietà
  - Metriche della documentazione

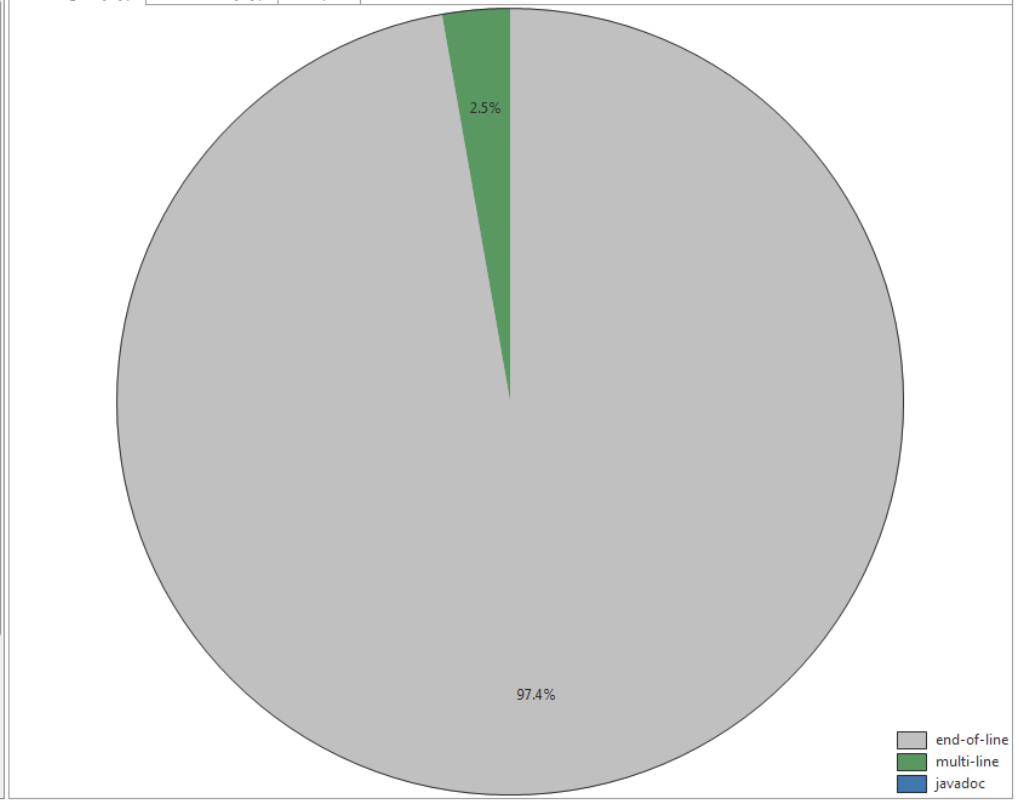
Metrics

- Il calcolo è completamente automatico e può essere eseguito con CodePro Tools/Compute Metrics sul menu di contesto
- Per ogni metrica sono riportati:
  - *I valori misurati*
  - *Il significato della metrica*
  - *E' possibile porre delle soglie in modo da evidenziare rispetto a quali metriche viene rilevato un cattivo comportamento*
  - *Genera automaticamente un report HTML*

Metrics

Calendario_Integration at 25/10/14 10.25

Metric	Value
Abstractness	0%
Afferent Couplings	0
Average Block Depth	1.39
Average Cyclomatic Complexity	9.00
calendario	9.00
Calendario.java	9.00
Average Depth of Inheritance Hierarchy	2.00
Average Lines Of Code Per Method	18.00
calendario	18.00
Calendario.java	18.00
Average Number of Constructors Per Type	0.00
Average Number of Fields Per Type	0.00
Average Number of Methods Per Type	5.00
Average Number of Parameters	2.20
Average Number of Subtypes	0.00
Comments Ratio	41.9%
Difficulty	32.19
Distance	0.00
Efferent Couplings	1
Effort	82,645.54
Instability	1.00
Lines of Code	93
Number of Characters	3,360
Number of Comments	39
Number of Constructors	0
Number of Fields	0
Number of Lines	147
Number of Methods	5
Number of Operands	233
Number of Operators	156
Number of Packages	2
Number of Semicolons	49
Number of Types	1



Caratteristiche di qualità del testing

- Deve aiutare a localizzare i difetti
  - Per facilitare il debugging
    - *Tanti test semplici consentono una più semplice localizzazione rispetto a pochi test complessi*
- Dovrebbe essere ripetibile
  - Potrebbe non essere possibile se l'esecuzione del caso di test influenza l'ambiente di esecuzione senza la possibilità di ripristinarlo
  - Potrebbe non essere possibile se nel software ci sono degli elementi indeterministici
    - *Ovvero dipendenti da input non controllabili*
- Dovrebbe essere preciso
  - La valutazione del risultato deve essere univoca

Efficacia ed Efficienza

- Il Testing deve essere:
  - **Efficace**
    - *Condotto attraverso strategie che consentano la scoperta di quanti più difetti possibili;*
  - **Efficiente**
    - *In grado di trovare difetti provando il minor numero possibile di casi di test*

- Necessità di uso di metodi e tecniche per ridurre lo sforzo (inteso come impiego di risorse umane, tecnologiche e di tempo) per poter individuare il massimo numero possibile di malfunzionamenti (con il minimo numero possibile di test case) prima che il prodottto sia rilasciato
    - *circa il 40% dei costi di produzione del software per il raggiungimento di ragionevoli livelli di qualità sono richiesti dal testing*

Misura dell'efficacia e dell'efficienza

- L'**Efficacia** di un'attività di testing può essere definita come:
  Numero di difetti scoperti / numero di difetti esistenti

- L'**Efficienza** di un'attività di testing può essere definita come:
  Numero di casi di test rilevanti difetti / numero di casi di test eseguiti

- Per aumentare l'efficacia spesso si aumenta il numero di casi di test eseguiti (diminuendo l'efficienza)
- Per aumentare l'efficienza si eseguono soltanto i casi di test corrispondenti ad esecuzioni "critiche" del software (diminuendo spesso l'efficacia)

Misura dell'efficacia e dell'efficienza: problemi

- L'efficienza del test può essere misurata in termini di numero di casi di test eseguiti, tempo o sforzo necessario per eseguirli
  - Il numero di test è la grandezza più semplice da misurare. Se il testing è automatico, la misura più utile è il tempo, altrimenti la più utile è lo sforzo
- L'efficacia dei casi di test in generale non può essere misurata
  - Non sappiamo quanti difetti sono presenti nel software!
    - *Ciò è possibile solo in ambiente sperimentale, nel caso in cui iniettiamo appositamente difetti nell'applicazione allo scopo di valutare l'efficacia della test suite nel rivelare malfunzionamenti*
  - Possiamo, però, sempre misurare l'efficacia relativa di una test suite rispetto ad un'altra in base al rapporto tra i difetti scoperti

Misura indiretta dell'efficacia

- In mancanza di misure dirette, l'efficacia si può misurare in termini di copertura. Ad esempio
  - CovLOC=LOC coperte da almeno un caso di test / Totale LOC
  - CovMetodi=Metodi coperti da almeno un caso di test / Totale Metodi
  - CovClassi=Classi coperte da almeno un caso di test / Totale Classi

- La copertura può essere interpretata come una «speranza» di verifica
  - se non si esegue la riga difettosa, sicuramente non si riscontrerà il fallimento; in caso contrario c'è la possibilità (non certezza) di riscontrarlo

Valutazione della copertura

- Per valutare la copertura è necessario inserire (possibilmente in maniera automatica) sonde nel codice dell'applicazione, che ci permettano di monitorare i percorsi di esecuzione

- La copertura si riferisce tecnicamente alle righe di codice eseguibile: esistono delle difficoltà nel riportarlo al codice sorgente o al modello

Efficacia ed efficienza

- Le varie tecniche abbinano livelli crescenti di efficacia a livelli decrescenti di efficienza
  - Non è facile trovare il giusto compromesso

- L'efficacia va privilegiata quando si vuole un software affidabile (ad esempio per applicazioni critiche)

- L'efficienza va privilegiata se si vuole ridurre lo sforzo allocato alle attività di testing
  - in particolare se non può essere eseguito automaticamente

Efficacia ed efficienza al variare delle tecniche

- Tecniche di automazione del testing consentono di:
  - Ridurre lo sforzo a parità di efficacia → aumentare l'efficienza

- Tecniche di riduzione dei casi di test (o di loro prioritizzazione) consentono di:
  - Ridurre il numero di casi di test a parità di efficacia → aumentare l'efficienza

- Tecniche combinatoriali, tecniche di generazione automatica di casi di test particolari, tecniche di mutazione di casi di test consentono di:
  - Aumentare l'efficacia a scapito dell'efficienza

Metriche per la manutenzione

- Le metriche per la manutenzione si occupano di prevedere la complessità del processo di manutenzione
  - Dipendono, quindi, dal processo di manutenzione che si va a istanziare

Ad esempio (Indice di Maturità del Software):

$SMI = [M_T - (F_a + F_C + F_d)] / M_T$

- M_T: numero di moduli nella versione corrente
- F_C: numero di moduli modificati
- F_a: numero di moduli aggiunti
- F_d: numero di moduli eliminati

- **Al tendere di SMI a 1 il prodotto tende a stabilizzarsi**
- **Il tempo medio per produrre una nuova versione del prodotto è correlato a SMI.**

Appendice:

Proporre delle metriche con il paradigma GQM

Proporre delle metriche

- Per proporre delle metriche valide, bisogna trovare le relazioni tra entità e metriche
- Tali relazioni possono essere ottenute tramite studi empirici…
  - *si cerca di trovare una legge secondo la quale la misura di un'entità possa essere valutata come funzione delle metriche di attributi direttamente misurabili.*
- Oppure tramite la costruzione di modelli metrici arbitrari, nei quali le relazioni tra attributi interni ed esterni sono ipotizzati da esperti e successivamente validati.

- In [Kan, Metrics and Models in Software Quality Engineering, Second Edition, 2002] si possono trovare molte indicazioni su come effettuare una campagna di misure e su come validare empiricamente l'efficacia di metriche proposte nella descrizione di dati fattori di qualità
- Molto utile anche il libro di Fenton e Pfleeger [Software Metrics, Second Edition, 1997]

Goal-Question-Metric Paradigm

- Introdotto da Basili e Rombach a partire dal 1988
  - Basili, V. R., Caldiera, G. and Rombach, H. D., "The Goal Question Metric Approach", Encyclopedia of Software Engineering, Wiley&Sons Inc., 1994

- L'applicazione di un processo GQM permette di effettuare un programma di raccolta dati
  - Per effettuare l'assessment (valutazione) dell'efficacia dei processi aziendali
  - Per valutare la qualità di prodotti esistenti

GQM

- Basato sull'idea che per poter misurare con successo occorre :
  - Stabilire i Goals (gli Obiettivi) che si intendono perseguire con il programma di raccolta dati;
  - A partire dagli obiettivi, derivare i dati che permettono di valutare il raggiungimento degli obiettivi;
  - Definire uno schema di riferimento per interpretare i dati rispetto agli obiettivi fissati.
- Il risultato sarà un modello di misurazione a tre livelli (Concettuale, Operazionale, Quantitativo) corrispondenti a Goal/ Questions/ Metrics, ottenuto in tre macro-fasi.

Goal-Question-Metric Paradigm

Goals - Obiettivi

- Quali sono gli obiettivi (attributi esterni) che si vogliono perseguire?
  - *Un goal si definisce individuando **l'oggetto** dello studio, le **motivazioni** per la misurazione, un **modello di qualità** di riferimento, un **punto di vista** per l'osservazione, **l'ambiente** a cui l'oggetto appartiene.*
  - *Possibili oggetti: Prodotti/ Processi/ Risorse.*
- Esempio di Goal:
  - *Analizzare le attività di progettazione e verifica del processo di sviluppo, al fine di valutare l'efficacia del processo di rilevazione dei difetti presenti nel software, dal punto di vista del management e del team di sviluppo. Si vuole verificare se una tipologia di test ha un rapporto costo/benefici soddisfacente.*

Questions

Questions - Domande

- Quali domande ci permettono di valutare se e fino a che punto gli obiettivi vengono raggiunti?

- Esempio di domande:
  - *Q1) Quale è il costo del test?*
  - *Q2) Quali sono I vantaggi?*

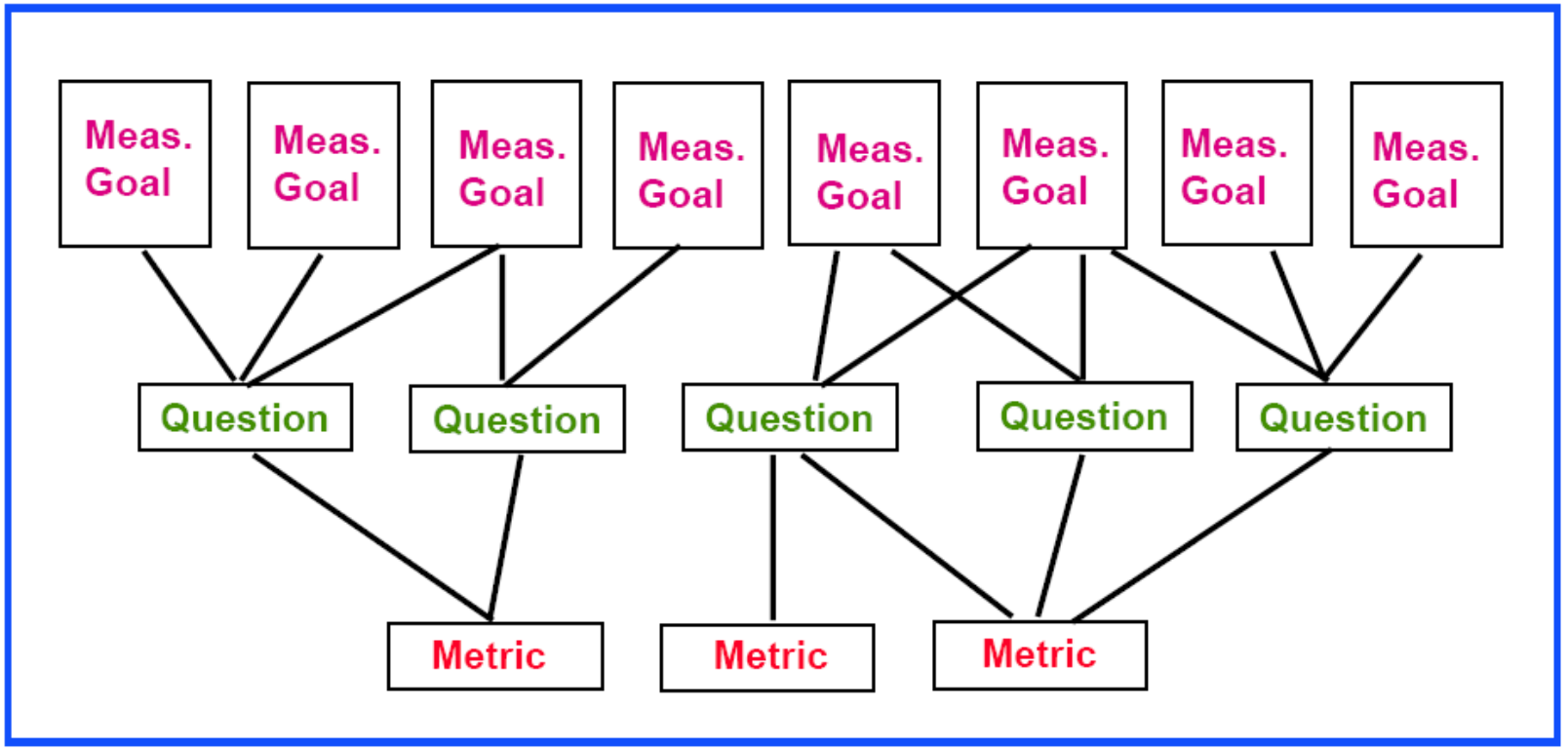
Metrics

Metrics - Metriche

- Quali metriche consentono di dare risposta alle domande?

- Q1)-> M11: Giornate per singolo test
  - -> M12: Costo per figura professionale
  - -> M13: Numero di Persone coinvolte
- -Q2) -> M21: Numero di errori gravi rilevati
  - -> M22: Numero di errori lievi rilevati

Misurazione Goal-Based



Osservazioni

- I valori per le matrici QM e GQ sono spesso ottenuti da questionari compilati dagli esperti dei processi aziendali e dai responsabili della qualità che utilizzano un insieme discreto di valori possibili (poi normalizzati tra 0 e 1)
- La difficoltà nell'esecuzione di un GQM sta nella ricerca di matrici che portino a risultati soddisfacenti
- I valori possono essere aggiustati a seguito di osservazioni fatte a posteriori che vadano a correggere delle dipendenze tra question e goal che non siano soddisfacenti
  - Dei feedback possono essere ottenuti dall'applicazione dello stesso modello GQM a diversi progetti nell'ambito della stessa azienda e dal confronto dei risultati ottenuti

Appendice:

Metodi decisionali: Tecnica AHP

Metodi decisionali

- La realizzazione di misure molto spesso supporta processi decisionali
  - Processi nei quali si vuole scegliere tra diverse alternative in maniera oggettiva e replicabile, sulla base di misurazioni quantitative

- Un metodo decisionale aspira a poter realizzare una tecnica (al limite un'espressione logica) che possa consentire di prendere una decisione

Metodi decisionali

- Per poter scegliere tra diverse alternative, avremmo bisogno di una metrica assoluta (quanto meno di una metrica che ammetta una relazione d'ordine) in base alla quale «ordinare» tutte le alternative, in moda da scegliere quella che ha il valore «migliore»

- Nella maggior parte dei casi, non esiste una metrica di tale tipo, quindi bisogna «combinare» tra loro diverse metriche
  - Spesso tali metriche non variano in maniera monotona tra le alternative: a valori «migliori» di una metrica corrispondono valori «peggiori» di altre metriche
  - La combinazione delle metriche dovrebbe esprimere una relazione d'ordine totale tra tutte le alternative

Tecnica AHP

- Vogliamo scegliere tra diversi oggetti quello che soddisfa al meglio le nostre richieste di qualità
  - Utilizzando appieno GQM potremmo arrivare a definire un modello di qualità complessivo e infine calcolare una metrica complessiva
  - Tramite la tecnica AHP (Analytic Hierarchical Process) possiamo arrivare allo stesso risultato basandoci soltanto su confronti a due a due di fattori di qualità e valori offerti
    - Studi preliminari svolti hanno dimostrato come sia più efficace comparare due alternative piuttosto che trovare una relazione d'ordine totale

- Thomas L. Saaty, *Multicriteria decision making - the analytic hierarchy process. Planning, priority setting, resource allocation* , RWS Publishing, Pittsburgh, 1988.

Passi della tecnica AHP

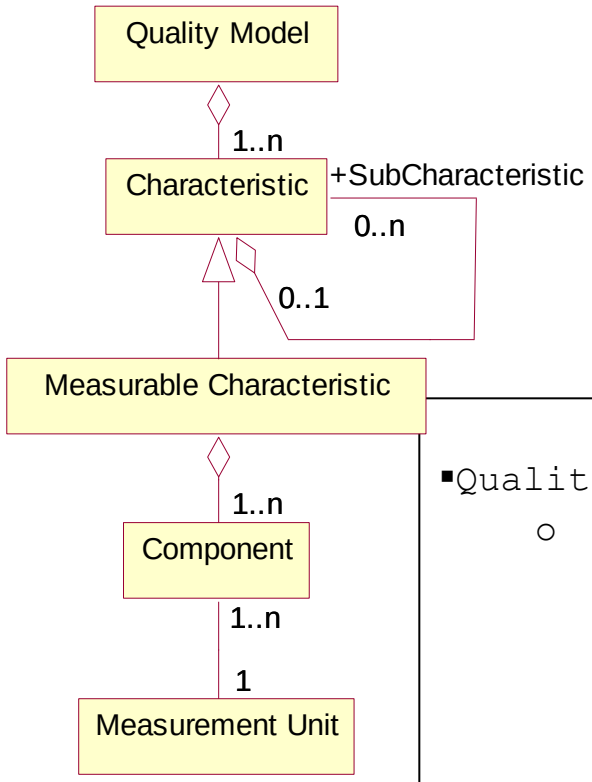
1. Modellazione della qualità richiesta e della qualità offerta
   - Dobbiamo organizzare in un meta-modello e instanziare in un modello quelli che sono I requisiti di qualità richiesti e offerti

2. Formulazione di un modello per la misura della qualità con la tecnica AHP

3. Formulazione di un modello decisionale con la tecnica AHP

4. Misura della qualità delle alternative e scelta del processo che meglio bilancia le nostre richieste con l'offerta

To define a Quality Meta-Model



• Quality Characteristic: any quality requirements, such as Performance, Security, Cost, Maintainability

•Characteristics may be arranged in a hierarchy (Measurable Characteristics are the leaves)

• Measurable Characteristic: a Quality Characteristic that can directly be measured

- Quality Characteristic: Efficiency
  - Quality Characteristic: Time Behaviour
    - Quality Characteristic: Response Time
      - Measurable Quality Characteristic: Average Response Time
        - Component: Average Response Time Value
          - Measurement Unit: Seconds
        - Component: Samples Number
          - Measurement Unit: Natural Number

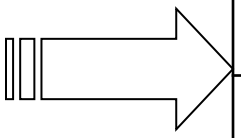
The Analytical Hierarchy Process

1. **The decision framework design activity:**
   1. **Weight Assignment step**: the relative importance of the characteristics is rated;
   2. **Clustering step:** for each measurable characteristic, the sets of values that will be considered equivalent for the aims of the evaluation are defined;
   3. **Rating Step:** each set is associated to a rating value;

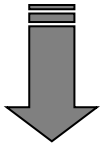
2. **The decision making activity:** to compare the quality of an offered service (formalised in a Quality Offer Model) against requestor needs (formalised in a Quality Request Model)

Casola, V.; Fasolino, A.R.; Mazzocca, N.; Tramontana, P., "An AHP-Based Framework for Quality and Security Evaluation," *Computational Science and Engineering, 2009. CSE '09. International Conference on* , vol.3, no., pp.405,411,

Intensity of Importance and its interpretation	
Intensity of Importance	**Interpretation**
1	Equal Importance
3	Moderate Importance
5	Strong Importance
7	Very strong Importance
9	Extreme Importance



	Average Response Time	Standard Deviation Response Time	Maximum of Response Time
Average Response Time	1	3	7
Standard Deviation Response Time	1/3	1	5
Maximum Response Time	1/7	1/5	1



Characteristic Weights are assigned by comparing their relative importance:

$$w(i) = \sum_{k=1}^{n} \frac{m'(i,k)}{n} \quad \forall \mathrm{i}$$

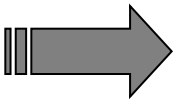
	Average Response Time	Standard Deviation Response Time	Maximum Response Time	Weights
Average Response Time	21/31	15/21	7/13	0.64
Standard Deviation Response Time	7/31	5/21	5/13	0.28
Maximum Response Time	3/31	1/21	1/13	0.07

Step 2: Clustering

Let's consider the Average Response Time characteristic

R = Offered_value / Requested_value

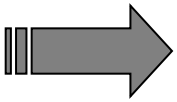
Possible Solutions are clustered in three levels:



R < 0.5 (very fast response);
0.5≤ *R* <1 (sufficiently fast response);
1≤ *R* <2 (quite slow response).

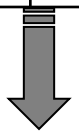
Step 3: Rating

Intensity of Goodness and its interpretation	
Intensity of Goodness	**Interpretation**
1	Equivalent
3	Moderately better
5	Strongly better
7	Very strongly better
9	Extremely better



Ratings are assigned to clusters by comparing their relative Goodness

	R<0.5	0.5≤R<1	1≤R<2	Rating
R < 0.5	1	3	5	0.63
0.5≤R<1	1/3	1	3	0.26
1≤ R<2	1/5	1/3	1	0.11



Satisfaction Function S(R)

$$S(R) = \begin{cases} 0.63 & if \quad R < 0.5 \\ 0.26 & if \quad 0.5 \le R < 1 \\ 0.11 & if \quad 1 \le R < 2 \end{cases}$$

The Decision Making Activity

The Quality of different Web Services is compared by evaluating:

1. a Satisfaction Function for each Measurable Characteristic.
2. a Satisfaction Function for each non-Measurable Characteristic

$$S_c(request, offer) = \sum_{sc \in C(c)} w_{sc} S_{sc}(request, offer)$$

3. the Overall Satisfaction Function:

$$S(request, offer) = \sum_{c \in Characteristic} w_c S_c(request, offer)$$

The Web Service with the greater Satisfaction Function value is chosen

Appendice:

Generazione di strumenti di misura: ANTLR

Bibliografia

Terence Parr, The Definitive ANTLR Reference: Building Domain-Specific Languages, The Pragmatic Programmers, http://www.pragprog.com/titles/tpantlr

http://www.antlr.org/

Giulio Iannello, dispense di Algoritmi e Strutture Dati, http://www.docenti.unina.it/docenti/web/download.php?id_prof=56

ANTLR

ANTLR (Another Tool for Language Recognition) è un sofisticato generatore di parser che può essere utilizzato per implementare riconoscitori, compilatori, traduttori, strumenti di misura di metriche di prodotto per qualsiasi linguaggio, in particolare per I cosiddetti DSL (domain-specific languages)

Esempi classici di DSL sono file dati, I file di configurazione, I protocolli di rete e molti altri linguaggi specifici di un dominio.

L'utilizzo di Antlr non è consigliabile per quei linguaggi che sono dei dialetti di XML, in quanto, per essi, è più conveniente riutilizzare le librerie per il parsing XML

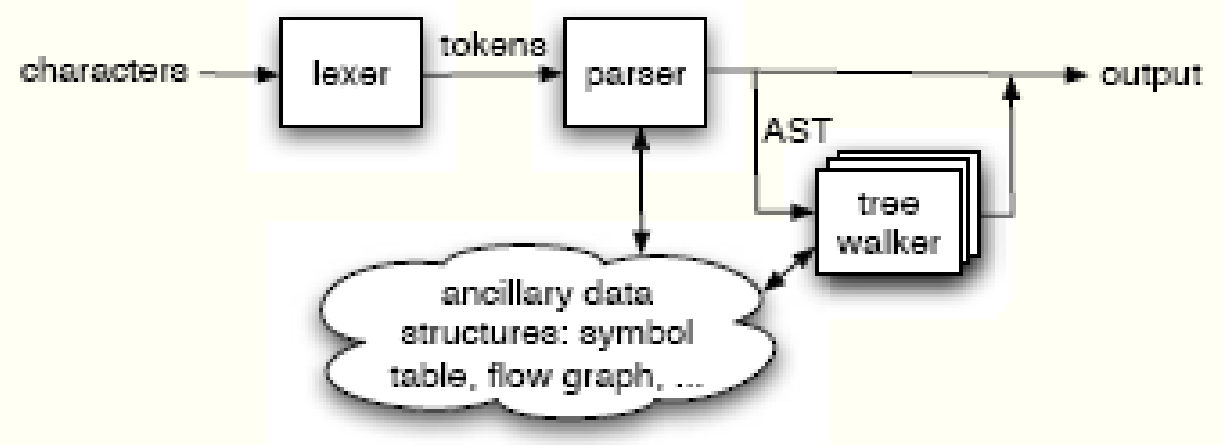
Traduttori

Un traduttore lavora su di un documento in input, scritto in un certo linguaggio, e traduce ogni frase in un'altra nel linguaggio di output.

Per realizzare questa trasformazione, esso esegue del codice sorgente realizzato ad hoc per il inguaggio da analizzare

Un traduttore dovrà quindi esibire comportamenti diversi per ognino delle possibili tipologie di frasi (sentences) trovate nel documento in input.

Il processo di traduzione

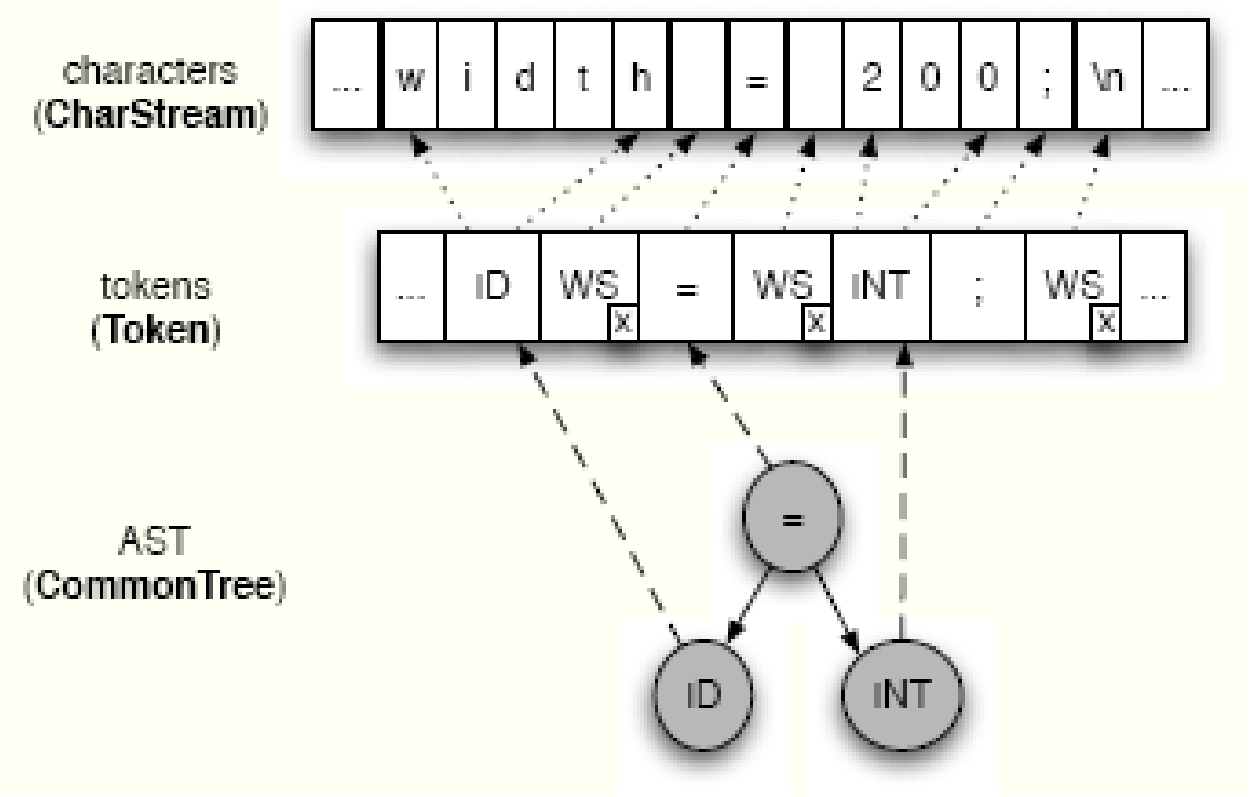


Il Lexer riconosce tokens (ad esempio keywords) dall'analisi del testo (analisi lessicale);

Il parser riconosce strutture formate da token (analisi sintattica)

L'uscita più efficiente da gestire è quella in forma di AST (Abstract Syntax Tree)

Esempio


characters
(CharStream)
... w i d t h = 2 0 0 ; \n ...
tokens
(Token)
... ID WS = WS INT ; WS ...
AST
(CommonTree)
=
ID
INT
Lexer
Parser

In dettaglio

La prima fase della traduzione è l'analisi lessicale, che opera direttamente sullo stream in ingresso, riconoscendo *tokens* a partire dall'analisi dei caratteri dello stream di input

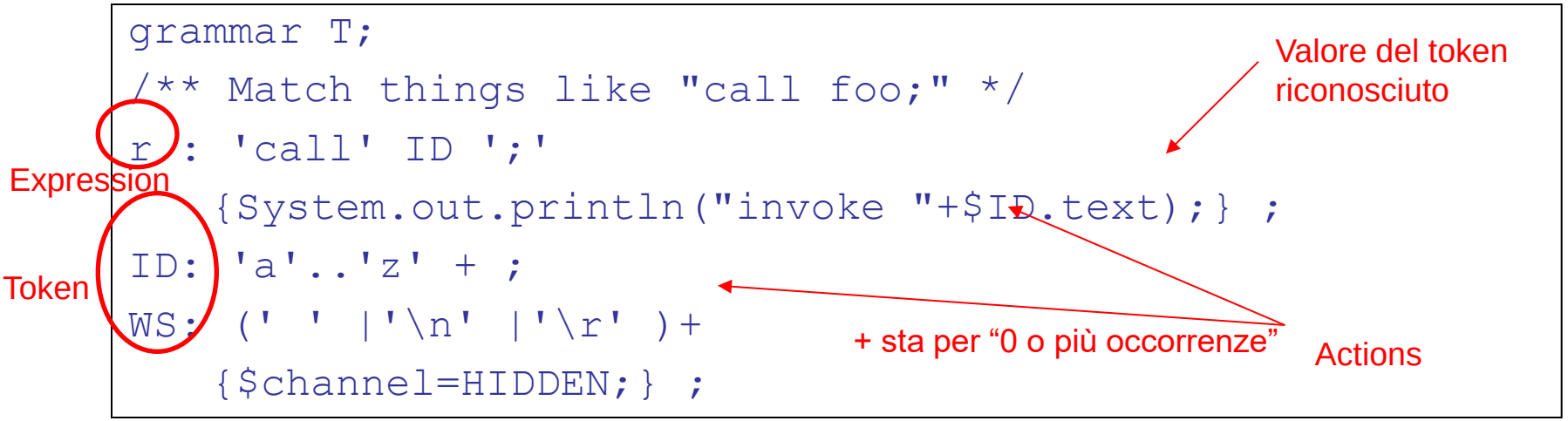
La seconda fase è il parsing propriamente detto, che elabora i tokens restituiti dal lexer, riconosce la loro organizzazione sintattica ed esegue delle elaborazioni conseguenti al loro riconoscimento

ANTLR è in grado di generare automaticamente il codice sorgente dell'analizzatore lessicale e del parser sulla base della descrizione della grammatica e delle elaborazioni conseguenti ai riconoscimenti dei token e delle strutture (che devono essere forniti dall'utente)

Riassumendo ...

- L'analizzatore lessicale organizza in token lo stream di input
- Il parser legge questa sequenza di token e cerca di riconoscere le frasi e la loro struttura
- Il più semplice traduttore si limita a proporre in output le frase riconosciture, senza ulteriori fasi
- Traduttori più complessi costruiscono strutture organizzative come gli AST, per consentire un'agevole interpretazione ad altri software da posizionare più a valle

Esempio (banale) di grammatica



$ java org.antlr.Tool T.g
ANTLR Parser Generator Version 3.0 1989-2007
$ ls
T.g TLexer.java T__.g T.tokens TParser.java

Main di test

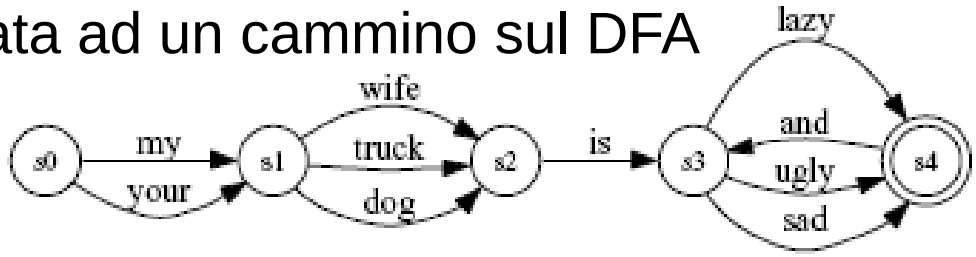
```
import org.antlr.runtime.*;
public class Test {
   public static void main(String[] args) throws Exception {
   // create a CharStream that reads from standard input
    ANTLRInputStream input = new ANTLRInputStream(System.in);
    // create a lexer that feeds off of input CharStream
    TLexer lexer = new TLexer(input);
    // create a buffer of tokens pulled from the lexer
    CommonTokenStream tokens = new CommonTokenStream(lexer);
    // create a parser that feeds off the tokens buffer
    TParser parser = new TParser(tokens);
    // begin parsing at rule r
    parser.r();
    }
}
```

```
$ javac TLexer.java TParser.java Test.java
$ java Test
call foo; EOF
invoke foo
```

State Machines

- Gli automi deterministici a stati finiti (DFA) possono essere utilizzati come generatori di espressioni regolari

- Nell'esempio, ogni percorso dallo stato iniziale a quello finale corrisponde ad una delle possibili espressioni regolari
  - Quindi, un'espressione è regolare, se può essere associata ad un cammino sul DFA

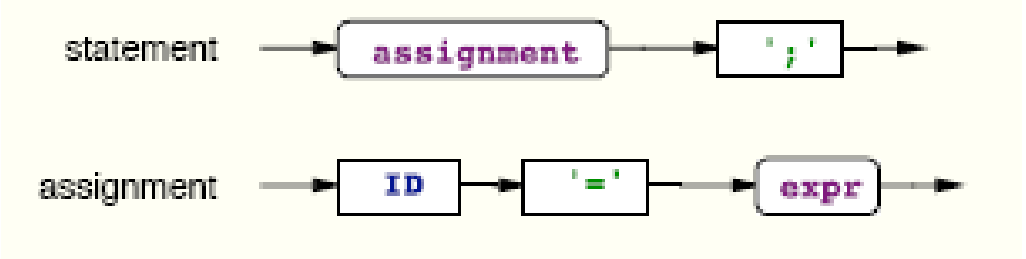


PushDown Automata

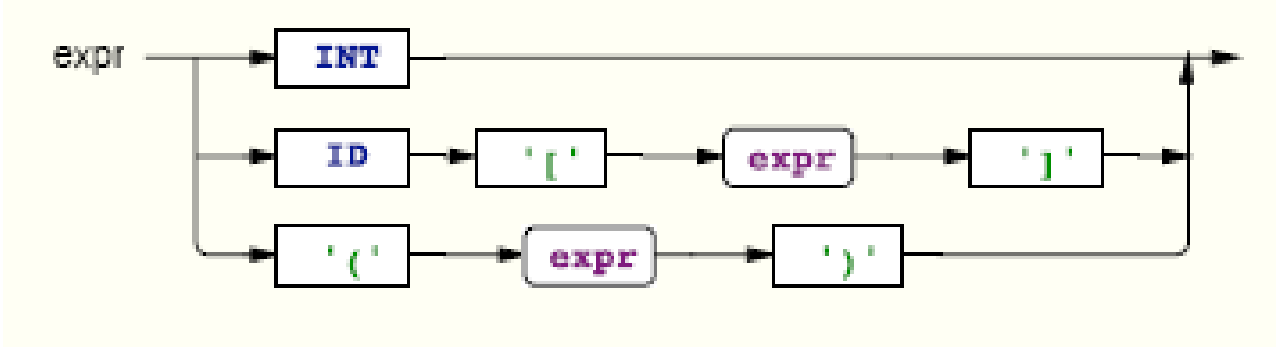
- Le macchine a stati hanno un comportamento che non dipende dalla "memoria" della macchina
  - In pratica bisogna considerare stati diversi per ogni possibile valore della memoria
    - *Il numero di stati andrebbe ad esplodere*
- Per questo motivo si preferiscono i Pushdown Automata
  - Si tratta di Automi a Stati Finiti dotati di memoria a stack
  - Un Pushdown Automata, essendo dotato di memoria, può essere visto come un insieme di submachine
    - *Le transizioni tra uno stato e l'altro dipendono, oltre che dagli eventi che si verificano, anche dal valore di variabili memorizzate nello stack*
  - Graficamente si indicano tramite Syntax Diagrams

Syntax Diagram

Syntax Diagram



- Syntax Diagram ricorsivo



Un esempio completo

Si vogliono riconoscere e calcolare espressioni di assegnazione tali che:

- sul lato destro ci sono espressioni aritmetiche con gli operatori di somma e prodotto e l'uso delle parentesi per esprimere precedenza
- sul lato sinistro (opzionale) ci sono variabili cui assegnare il valore calcolato sul lato destro

Esempi:

- a=3 (memorizza 3 in a)
- b=4 (memorizza 4 in b)
- 2+a*b (restituisce 14)

Obiettivi

Gli obiettivi da perseguire sono due:

1. Costruire una grammatica che descriva la struttura sintattica delle espressioni e delle assegnazioni

2. Scrivere del codice, contestualmente alla grammatica, che esegue le corrette azioni in conseguenza del riconoscimento delle espressioni della grammatica stessa (ad esempio effettuare una somma quando si riconosce un'espressione con +, e così via)

Grammatica 1/2

```
grammar Expr;

prog:   stat+ ;

stat:   expr NEWLINE
    |   ID '=' expr NEWLINE
    |   NEWLINE
    ;

expr:   multExpr (('+'|'-') multExpr)*
    ;

multExpr
    :   atom ('*' atom)*
    ;
```

"un prog è un insieme di (0 o più) stat (statements)"

"una stat è una expr oppure un ID seguito da '=' e da una expr oppure una riga vuota"

"una expr è data da uno o più coppie di multExpr con in mezzo '+' o '-'"

"una multExpr è data da uno o più coppie di atom con in mezzo '*'"

Grammatica 2/2

```
atom:    INT
     |   ID
     |   '(' expr ')'
     ;
```

"una atom è data da un INT oppure un ID oppure una expr tra parentesi"

```
// TOKEN
ID  :   ('a'..'z'|'A'..'Z')+ ;
INT :   '0'..'9'+ ;
NEWLINE:'\r'? '\n' ;
WS  :   (' '|'\t'|'\n'|'\r')+ {skip();} ;
// END:tokens
```

Se si incontra un WS (Whitespace) lo si può saltare senza fare nulla (funzione skip())

"ID è dato da zero o più caratteri alfabetici;
INT è dato da zero o più cifre;
NEWLINE è il codice ASCII \n, eventualmente preceduto da \r
WS è un carattere di spazio, tabulazione o da capo, isolato"

Testing della grammatica

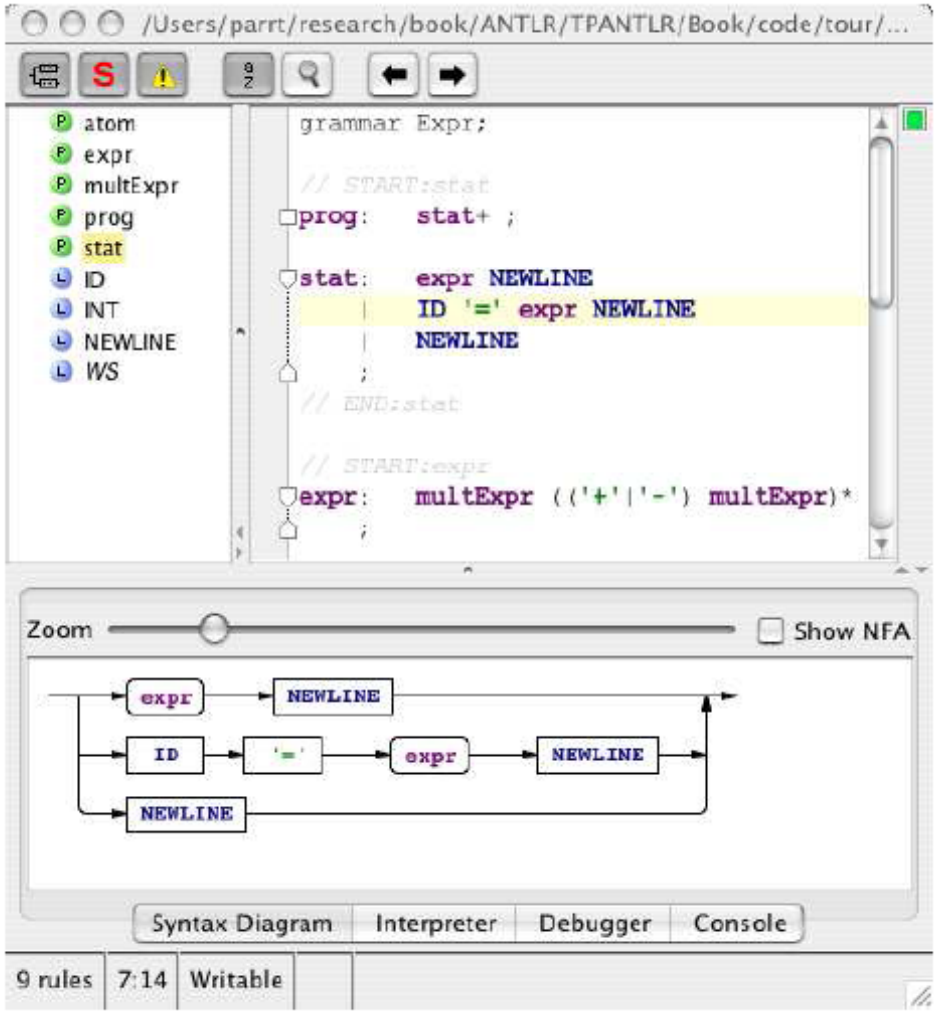
```
import org.antlr.runtime.*;

public class Test {
    public static void main(String[] args) throws Exception {
        // Create an input character stream from standard in
        ANTLRInputStream input = new ANTLRInputStream(System.in);
        // Create an ExprLexer that feeds from that stream
        ExprLexer lexer = new ExprLexer(input);
        // Create a stream of tokens fed by the lexer
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        // Create a parser that feeds off the token stream
        ExprParser parser = new ExprParser(tokens);
        // Begin parsing at rule prog
        parser.prog();
    }
}
```

Questa classe di test consente di elaborare lo stream di input proveniente da System.in

L'unico output possibile, per ora è costituito dalle eccezioni che possono rilevare parser o lexer

Antlr Works

- Dal sito web di ANTLR, è possibile scaricare anche Antlr Works, un tool visuale che supporta la scrittura di grammatiche per antlr
- http://tunnelvisionlabs.com/products/demo/antlrworks
- AntlrWorks è un ambiente GUI che integra al suo interno anche Antlr



AntlrWorks

AntlrWorks ha numerose funzioni che semplificano la realizzazione e il testing di grammatiche:

- Visualizzazione/modifica Syntax Diagram
- Generazione ed editing di Parser e Lexer
- Debugging delle grammatiche
  - *Con visualizzazione grafica dell'AST trovato*

Tutte le prove successive verranno effettuate tramite AntlrWorks

Prove della grammatica

L'input:

a=3

b=4

3+(a*b)

Genera un output vuoto (abbiamo scritto solo il riconoscitore, ma nessuna azione è stata associata)

L'input:

3+@

Genera il seguente output:

line 1:2 no viable alternative at character '@'

Grammatica con azioni 1/2

```
grammar Expr;

@header { import java.util.HashMap; }

@members { HashMap memory = new HashMap();}

prog:   stat+ ;

stat:  expr NEWLINE {System.out.println($expr.value);}
    |   ID '=' expr NEWLINE
    {memory.put($ID.text, new Integer($expr.value));}
    |   NEWLINE
    ;

expr returns [int value]
    :   e=multExpr {$value = $e.value;}
        (   '+' e=multExpr {$value += $e.value;}
        |   '-' e=multExpr {$value -= $e.value;}
        )*
    ;
```

Inclusioni necessarie

Istanzia una variabile memory

Gli elementi riconosciuti sono indicati con $elemento

$ID.text è il nome dell'elemento ID

Value è un attributo definito per questo ed altri elementi

Grammatica con azioni 2/2

```
multExpr returns [int value]
    :   e=atom {$value = $e.value;} ('*' e=atom {$value *= $e.value;})*
    ;

atom returns [int value]:
   INT {$value = Integer.parseInt($INT.text);}
    |   ID
        {
        Integer v = (Integer)memory.get($ID.text);
        if ( v!=null ) $value = v.intValue();
        else System.err.println("undefined variable "+$ID.text);
        }
   |   '(' expr ')' {$value = $expr.value;}
    ;
ID  :   ('a'..'z'|'A'..'Z')+ ;
INT :   '0'..'9'+ ;
NEWLINE:'\r'? '\n' ;
WS  :   (' '|'\t'|'\n'|'\r')+ {skip();} ;
```

Converte l'INT trovato in un intero

Sostituisce la variabile col suo valore