Università degli Studi di Pisa
Laurea Magistrale in Informatica Umanistica

Seminario di Cultura Digitale

Prof.ssa Enrica Salvatori

Literate Programming

Studente: Giorgio Spugnesi

Anno Accademico 2011 - 2012

Introduzione

Il titolo della presente relazione riprende quello di un importante articolo sulla programmazione pubblicato nel 1984 da Donald Knuth sulla rivista "The Computer Journal"[1]. Scopo della relazione è presentare sia la programmazione come forma di letteratura sia il contributo che Knuth ha dato all'affermazione di questo concetto. Prendendo avvio dal seminario di Vincenzo Gervasi e Marco Grandi "Programmare in formule: tipografia matematica di alta qualità in un ambiente di programmazione"[2] se ne sottolineeranno gli aspetti più inerenti ai paradigmi di programmazione, tralasciando l'approfondimento di quelli legati ai linguaggi tipografici (TEX e METAFONT), settore nel quale il contributo di Knuth è stato più che mai fondamentale.

Gli altri seminari potranno apparire sconnessi dall'argomento; tuttavia, ogni qualvolta si fa uso di un sistema informatico, questo prevede una programmazione ossia la codifica di istruzioni che l'uomo fornisce alla macchina. Sarà proprio sull'aspetto letterario che questa serie di istruzioni può assumere che si incentrerà la relazione. In quest'ottica, allora, la riflessione sottende a, più o meno, tutti gli altri interventi permettendo di scoprire l'aspetto, potremmo dire, "artistico – creativo" che sta dietro un'apparenza meramente tecnologica.

Donald Ervin Knuth

Appare opportuno presentare la figura del singolare studioso americano che per primo ha coniato il termine di *Literate Programming*, ripercorrendone brevemente la biografia e prestando particolare attenzione al significativo metodo di lavoro che prevede la costruzione di strumenti concettuali anche complessi per la soluzione dei problemi che vengono via via configurandosi nella ricerca.

Donald Knuth nasce a Milwakee nel 1938 e, dopo la laurea e il dottorato in matematica, entra a far parte della Stanford University di cui è tutt'ora membro con il titolo di *Professore emerito nell'arte della programmazione del computer.*

"L'arte della programmazione del computer" (*The Art Of Computer Programming* – TAOCP)[3] è anche il titolo della sua opera principale, un imponente lavoro sull'algoritmica del quale sono usciti 4 dei 7 volumi previsti.

Il piano dell'opera prevedeva infatti un primo volume, iniziato a scrivere nel 1962, con l'intenzione di raccogliere tutto quanto all'epoca esisteva in merito agli algoritmi e alla programmazione del computer. Fu stampato, nel 1968, usando una macchina linotype con caratteri a piombo fornendo un ottimo risultato tipografico. Nel 1976, per la pubblicazione della seconda edizione del secondo volume, fu invece utilizzata una delle prime macchine per fotocomposizione. I risultati tipografici

1 Knuth 1984.
2 Il seminario si è svolto mercoledì 22 febbraio 2012 presso l'Aula Seminari Est del Dipartimento di Informatica.
3 Knuth 2011.

furono ritenuti da Knuth inaccettabili tanto che decise di sviluppare egli stesso un linguaggio che consentisse la stampa tipografica di testi scientifici. Nel 1978 quindi dette vita a TEX, sicuramente il più completo e longevo sistema di composizione tipografica basato su istruzioni dichiarative. Insieme a TEX creò, l'anno successivo, METAFONT, il primo sistema di caratteri vettoriali.

A questo punto, Knuth disponeva degli strumenti per pubblicare in modo corretto il secondo volume del suo lavoro. Ma prima di fare questo decise di documentare i due strumenti che era andato creando, scrivendo, nel 1984, *The TeXbook*[4] (manuale di TEX) e, nel 1986, il *The METAFONTbook*[5] (manuale di METAFONT).

In pratica, al fine di pubblicare in modo corretto uno dei volumi sull'algoritmica, Knuth si era trovato a dare vita, con i suoi software, ad una sorta di rivoluzione tipografica che ha consentito, e consente tutt'ora, una corretta divulgazione scientifica. Gli strumenti creati per risolvere un problema concreto si trasformano in complessi ed universali linguaggi, destinati ad essere uno standard nel mondo scientifico.

Dopo questa parentesi, durata oltre dieci anni, Knuth ha ripreso la compilazione della sua *opus magna* sull'algoritmica, disciplina che, nel frattempo, si era notevolmente evoluta ad ampliata per il contributo di centinaia di informatici, matematici, ecc. Il numero dei volumi è quindi andato aumentando; lo stesso volume 4 è composto da diversi fascicoli, l'uscita del volume 5 è prevista per il 2020 mentre i volumi 6 e 7 devono essere ancora progettati.

Da questa vicenda, però, prende avvio anche il tema che andremo ad affrontare ovvero la capacità di presentare in modo descrittivo, immediatamente comprensibile all'uomo, il linguaggio delle macchine. La teoria di un linguaggio di programmazione che fosse in grado di dialogare contemporaneamente con altri uomini, rispetto al programmatore, oltre che con le macchine, assieme alla relativa pratica, frutto di dieci anni di sperimentazione, fu presentata nel già citato saggio *Literate Programming* che andremo ad analizzare nel dettaglio.

Non si può tralasciare, nell'inquadramento biografico di Knuth, di fare cenno ad alcune delle sue stranezze che caratterizzano il personaggio come la passione per i giochi e gli scherzi matematici. Ne citeremo solo un paio. I numeri di versione dei suoi software TEX e METAFONT tendono asintoticamente a due numeri particolari: *pi greco* ed *e* (costante di Nepero) ai quali aggiunge un decimale ad ogni rilascio. Per ogni errore individuato nei suoi testi, Knuth deposita, a vantaggio dello scopritore, la cifra di $2.56 (256 pennies ovvero un dollaro esadecimale 0x$1.00) nella fittizia banca di San Seriffe; la cifra è tuttavia realmente esigibile ma fino ad oggi solo 9 creditori hanno deciso di riscuoterla.

4 Knuth, 1984(2).
5 Knuth, 1986.

Programmazione per istruzioni e programmazione letteraria

Tradizionalmente, dall'invenzione del computer, l'attività di programmazione è sempre consistita nella definizione di una serie di istruzioni esposte in sequenza in modo comprensibile dalla macchina.

Tuttavia nessuno scrive programmi in linguaggio macchina, ossia nella sequenza di 0 e 1 che la macchina è in grado di elaborare.

Linguaggi come l'Assembly si collocano ad un livello molto basso e richiedono una traduzione particolarmente semplice che trasforma ogni parola dell'Assembly, in modo univoco, in una parola del linguaggio macchina. Questo compito è delegato ad un apposito programma detto compilatore.

Per rendere più agevole la programmazione, sono stati inventati linguaggi di alto livello nei quali la sintassi è più simile a quella del linguaggio naturale, espresso nella lingua inglese. In questo modo il programmatore può scrivere le istruzioni in modo più agevole, lasciando al compilatore il compito di tradurle in linguaggio macchina.

Tutti i linguaggi, comunque, prevedono una comunicazione uomo-macchina; anche se possono essere letti da un altro essere umano, non sono tuttavia rivolti ad esso e non rispettano la forma e la sintassi del linguaggio che si adopererebbe per comunicare con un umano.

La programmazione è indubbiamente anche una forma d'arte: ha la sua eleganza, forse classica, il suo linguaggio, le sue forme sintattiche, i suoi canoni ed anche una certa storia, sia pur breve. In senso astratto, la programmazione è l'arte di risolvere i problemi mediante algoritmi. Ma ancora di più, facendo uso di espressioni testuali, la programmazione è una forma di letteratura, assimilabile forse alla letteratura scientifica ancor più che a quella tecnica.

Se si dispone dei giusti strumenti interpretativi, di fronte ad un programma ben scritto si prova la stessa sensazione che un matematico prova di fronte ad una dimostrazione corretta. Sensazione che non è molto diversa da quella che proviamo leggendo un libro scritto veramente bene o guardando un'opera d'arte figurativa.

La programmazione quindi è una forma d'arte e come tale deve essere comunicativa. Per questo motivo, Donald Knuth capovolge il paradigma ed afferma che, in primo luogo, i programmi devono essere scritti per essere letti da un altro essere umano che deve venir convinto della validità di quanto affermiamo. Nel suo articolo afferma infatti:

> Instead of imaging that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do.[6]

Partendo da questo assunto e dalla considerazione che il programma è un *works of literature*, Knuth dà vita a quello che definisce *Literate Programming*.

6 Knuth 1984, p.97.

Il programmatore quindi agirà come un saggista, prestando attenzione all'esposizione e all'eccellenza dello stile, scegliendo opportunamente i nomi delle variabili e spiegandone il significato ed introducendo i concetti nell'ordine più comprensibile all'intelligenza umana piuttosto che nell'ordine di elaborazione effettivo.

Questo tipo di approccio, non solo porta ad una migliore comprensione del programma, ma genera anche programmi migliori perché incoraggia a lavorare con maggiore metodo, con una comprensione più profonda del problema, con una strutturazione delle argomentazioni più vicina alla logica del pensiero. Per questo motivo, i programmi realizzati con il paradigma del *Literate Programming*, sebbene siano molto pochi, presentano molti meno errori e si dimostrano più stabili.

WEB

Ovviamente a Knuth non bastava aver rivoluzionato l'approccio alla programmazione, doveva anche fornire gli strumenti pratici perché questa rivoluzione potesse prendere atto.

Così, assieme ai suoi associati, crea quello che viene definito il sistema WEB. Occorre precisare che il nome del sistema, coniato negli anni '70, non ha niente a che vedere con il World Wide Web che nasce nel 1991[7]. Fa però riferimento al concetto di ragnatela come complessa e delicata struttura composta di semplici pezzi:

> We understand a complicated system by understanding its simple parts, and by understanding the simple relations between those parts and their immediate neighbors. If we express a program as a web of ideas, we can emphasize its structural properties in a natural and satisfying way.[8]

Il sistema WEB è dato dalla combinazione di altri due linguaggi, un linguaggio di formattazione documentale e un linguaggio di programmazione (di tipo tradizionale). La scelta del linguaggio documentale fu semplice: Knuth aveva appena dato vita a TEX che si prestava perfettamente allo scopo. Come linguaggio di programmazione fu scelto PASCAL a motivo della sua diffusione, all'epoca, negli ambienti accademici. Tuttavia, ricorda Knuth, il sistema funziona anche con altri linguaggi sia documentali, sia di programmazione; anzi ha il vantaggio di non rendere obsoleti i linguaggi ma di donare loro una nuovo campo applicativo.

Nella pratica, il programmatore scrive un unico programma che servirà da sorgente per la documentazione e per il codice macchina. Questo fa sì che il codice sia non solo manutenibile ma costantemente consistente in quanto entrambi i prodotti finali sono generati dallo stesso sorgente.

Il codice in formato WEB conterrà, contrassegnati da appositi marcatori preceduti dal carattere @ che ne determinano, come in un libro, le sezioni e i capitoli, sia le parti descrittive, sia le istruzioni.

7 Convenzionalmente si fa iniziare il Web dal 6 agosto 1991, giorno in cui Tim Berners-Lee mise on-line su Internet il primo sito Web.

8 Knuth 1984, p.97.

L'ordine di presentazione dei concetti può essere quello più adatto alla comprensione umana; sarà l'elaborazione successiva a riordinare il codice affinché la macchina possa eseguirlo correttamente. In questo modo, il lavoro del programmatore somiglia alla scrittura di un saggio con il quale espone la propria soluzione al problema dato, in testo e in formule matematiche, lasciando che sia il sistema WEB a generare gli output opportuni.

Questo processo è compiuto da due programmi di sistema chiamati WEAVE (tessere) e TANGLE (intrecciare).

Gervasi, nella sua relazione, ricorda che i nomi sono stati ripresi da una citazione da *Marmion* di Walter Scott

> Oh, what a tangled web we weave when first we practice to deceive.

ed, in effetti, i due termini riprendono la metafora della tela di ragno che contraddistingue il sistema. Inoltre WEAVE rende l'idea di qualcosa di ordinato, di tessuto secondo un disegno prestabilito mentre TANGLE si riferisce ad un intreccio di parti diverse che vengono annodate tra loro. Questo rispecchia i risultati dei due programmi: il WEAVE genera infatti un documento stampabile, perfettamente comprensibile da un essere umano e tipograficamente ben formattato mentre il TANGLE mette insieme i pezzi di codice, concatenando una serie di istruzioni, sequenziali e non formattate, comprensibili da una macchina ma un po' meno da un essere umano.

Il processo avviene sottoponendo il file in formato WEB come input al WEAVE ottenendo in output un file .TEX che a sua volta può essere processato dal processore TEX ottenendo un file .DVI ossia una descrizione binaria “device-indipendent” di come la documentazione debba essere stampata. Poi lo stesso file WEB viene passato in input al TANGLE che produrrà un file .PAS ovvero un sorgente PASCAL che potrà essere compilato con un comune compilatore PASCAL.

In questo modo, da un unico file sorgente, si ottiene sia una versione della documentazione perfettamente stampabile, sia un eseguibile adatto alla macchina di cui dispongo, entrambi perfettamente coerenti ed allineati. Ad ogni modifica del sorgente, si dovrà solamente lanciare di nuovo i due programmi di sistema ed ottenere i due file destinazione aggiornati.

L'articolo di Knuth prosegue con un esempio pratico di un semplice programma per il calcolo e la visualizzazione tabellare dei primi n numeri primi, ripreso da un testo fondamentale sulla programmazione di Edgser Dijkstra, *Notes on Structured Programming*[9]. L'autore ha modo così di dimostrare non solo come si procede nella scrittura di codice in formato WEB ma anche come l'output possa essere formattato, ad esempio, per un articolo scientifico come è quello in oggetto, con tanto di sommario ed indice analitico generati direttamente dal software.

WEB deve essere visto prevalentemente come un paradigma di programmazione, più che un

9 Dahl, Dijkstra, Hoare. 1972, pp.26-39.

linguaggio. Uno dei suoi punti di forza è la portabilità sia su hardware che su linguaggi differenti.
Superando le molte limitazioni che il linguaggio PASCAL presenta, Knuth, assieme al collega di Berkley, Silvio Levy ha realizzato CWEB che consente la generazione di codice nei linguaggi C, C++ e Java. Analogamente potrebbe essere possibile replicare l'approccio di WEB per tutti i linguaggi di programmazione disponibili, riscrivendo opportunamente il TANGLE.
Per quanto riguarda la compatibilità tra hardware o sistemi operativi differenti, Knuth risolve il problema in modo sorprendentemente semplice: sia WEAVE che TANGLE accettano in ingresso non uno ma due file. In aggiunta al file sorgente è possibile specificare un "change file" (.CH) che contiene quanto deve essere modificato per la generazione del file di output sulla specifica macchina.
In questo modo si ottengono due vantaggi. Primo, si separa la logica del codice dalle specifiche implementazioni legate alla macchina e al sistema operativo. Secondo, in caso di aggiornamento del TANGLE o del WEAVE, è possibile usare la nuova versione senza dover riscrivere le personalizzazioni legate all'ambiente. In modo molto chiaro, Knuth afferma

> In other words, this dual-input-file scheme works when the WEB file is constant and the CH file is modified, and it also works when the CH file is constant but the WEB file is modified.[10]

Knuth conclude l'articolo con una serie di considerazioni tra cui quelle che vanno sotto il titolo di Economic Issues[11] e che si aprono con la domanda "What does it cost to use WEB?". Avendoci avvertiti di considerare quanto scritto come "the ravings of a fanatic who thinks he has just seen a great light"[12], è abbastanza scontato aspettarsi una stima molto ottimistica dei costi di applicazione del sistema. L'autore afferma che, in termini di risorse macchina, il sistema WEB non è più esigente di altri compilatori. Dove però si riscontra il guadagno maggiore è in termini di tempo speso per il debug. Il tempo speso a commentare in modo esteso e preciso il codice che si sta scrivendo, riduce infatti il tempo necessario ad individuare eventuali errori proprio in virtù della chiarezza espositiva del codice. Inoltre, scrivere codice secondo il paradigma WEB, quasi come fosse un flusso di coscienza, spinge il programmatore a chiarirsi maggiormente le idee, ad organizzare la struttura del software in modo più efficiente e questo porta a commettere molti meno errori già in fase di stesura del codice. L'idea di dover comunicare ad altri esseri umani la soluzione di un problema piuttosto che dover indicare ad una macchina quali istruzioni eseguire, sembra quindi dare vita a lavori formalmente più corretti e funzionanti oltre che più belli a vedersi.
Knuth chiude l'articolo con una serie di speranze e di propositi per il futuro, rivolgendosi in particolare ai *computer scientists* che potranno apprezzare il *Literate Programming* con la sua

10 Knuth 1984, p.107.
11 Knuth 1984, p.109.
12 Knuth 1984, p.97.

commistione di competenze verbali e matematiche. Teorizza inoltre lo sviluppo di un compilatore unico che faccia sia il TANGLE che la compilazione, magari permettendo il debug “step by step” del codice sorgente, e, soprattutto, auspica l'affermazione del sistema WEB in ambienti di sviluppo realmente effettivi, cosa che, salvo rare eccezioni, fino ad adesso non sembra essere avvenuta.

Vantaggi, svantaggi, limiti e prospettive

Un paradigma di programmazione come quello del *Literate Programming* apporta all'informatica indubbi vantaggi sia teorici che pratici. Il primo di essi è la possibilità di superare il dualismo tra l'approccio top-down e quello bottom-up. Da sempre infatti, nell'affrontare un problema informatico, è possibile prevedere un approccio che parte da una descrizione di alto livello che viene successivamente rifinita e scomposta in processi più semplici. Ma esiste anche un approccio che parte dai processi e dagli oggetti di base per comporre poi la struttura principale.

L'approccio *Literate Programming,* procedendo secondo il pensiero e visualizzando il software più come una ragnatela che come un albero, fa sì che non sia necessario compiere una scelta tra le due metodologie.

Come sottolineato più volte da Knuth, scrivere pensando ad un lettore aiuta ad essere più chiari, più espliciti, più organizzati nella stesura del codice e porta a fare molti meno errori. Questo, unito alla semplicità di debug, che può essere compiuto con facilità anche da chi non è il programmatore originario, porta innumerevoli vantaggi nell'adottare il paradigma.

Esistono però, è innegabile, anche alcuni svantaggi. Lasciamo per un attimo il contesto accademico e proviamo a calare la riflessione nell'informatica applicata all'azienda e ai settori produttivi.

Tralasciando i necessari tempi di apprendimento che ogni nuovo linguaggio o paradigma di programmazione comporta, la scrittura di *Literate Programms* richiede sicuramente più tempo della semplice scrittura di istruzioni di codice. E richiede anche una certa capacità “letteraria” che non sempre gli informatici, specie se non umanisti, hanno. In considerazione delle tempistiche con cui i software vengono realizzati, appare poco verosimile che una azienda possa investire su questo sistema di programmazione.

E' vero che la documentazione del software è una componente fondamentale, che dovrebbe essere sempre generata assieme al codice ma è anche vero che di questo aspetto, molto spesso, non se ne comprende l'importanza e si considera più una misura cautelativa in vista di eventuali problemi (bugs, cambio di personale in azienda, ecc.) che non una parte integrante del prodotto del programmatore.

A mio modesto avviso, il mondo delle aziende presenterebbe molta resistenza nei confronti dell'adozione di un approccio letterario alla programmazione e forse questo è il motivo per cui non

vi è stata una sua grande diffusione.

Il *Literate Programming*, inoltre, si presta facilmente ad un equivoco: quello di essere visto come un metodo per generare la documentazione direttamente dal codice. Esistono infatti sistemi, come ad esempio Javadoc o Doxygen che, partendo dai commenti, opportunamente formattati, che il programmatore scrive nel codice, sono in grado di generare ed impaginare documentazione delle classi e delle librerie di codice. In questo modo, programmando in maniera opportuna, è possibile ottenere la documentazione in modo contestuale e sempre allineata, semplicemente lanciando l'apposito software.

Ma il *Literate Programming* è cosa ben diversa; non si tratta di documentare il codice ma di scriverlo in un linguaggio più vicino a quello naturale con l'ausilio del linguaggio matematico-scientifico per la scrittura di formule. La documentazione, o meglio la versione stampabile, non è generata *dal* codice ma è generata *con il* codice, in modo contestuale. E' solo una diversa visualizzazione dello stesso concetto espresso, un output da leggere (per l'uomo) accanto ad un output da eseguire (per la macchina).

Chiarito questo dubbio, appare evidente che non si tratta di sfruttare uno strumento per ottenere la documentazione del codice ma capovolgere l'approccio alla scrittura del codice. E, ancora una volta, non so quanto le aziende (ed anche i programmatori) siano pronti.

Riscontro inoltre un grosso limite al *Literate Programming* ed in particolare al sistema WEB, limite che deriva, probabilmente, dall'ambiente in cui è nato: non viene mai preso in considerazione l'aspetto dell'interfaccia utente.

Tutti gli esempi di programmi scritti in WEB hanno interfaccia testuale per l'input dei parametri e l'output dei risultati. Ma come potrebbe comportarsi il sistema in applicazioni che richiedono una interazione uomo-macchina più complessa? In quale strato potrebbe andarsi ad inserire la definizione e l'implementazione dell'interfaccia?

Nei moderni programmi, l'interfaccia sta assumendo un ruolo sempre più rilevante; è finita l'epoca delle interfacce testuali o a bottoni ed esistono esperti di *User Interface* in grado di analizzare il progetto e fornire la migliore disposizione grafica degli elementi.

L'interfaccia è un elemento imprescindibile nella generazione di un prodotto software commerciale.

La sfida che si pone al *Literate Programming* è riuscire ad integrare nella descrizione di come risolvere il problema anche quella di come interagire con un terzo soggetto umano, oltre al programmatore e al lettore: l'utente.

Senza la pretesa di trovare una soluzione vorrei prospettare due possibili strade.

La prima rientra più nell'ottica di Knuth ossia quella di avere un unico file in grado di descrivere il problema ma anche la disposizione che gli elementi del problema devono avere nel corso della

soluzione e lasciare al TANGLE, o ad un analogo compilatore, il compito di generare l'interfaccia. In sostanza il TEX fa un lavoro simile, disponendo parti di informazione in modo graficamente comprensibile. Sulla carta quindi, il sistema potrebbe generare una sorta di mockup che fornisca al lettore un'idea dell'interfaccia. Per quanto riguarda l'eseguibile, le informazioni contenute nel codice di programmazione potrebbero mettere in grado il linguaggio o eventuali librerie grafiche di interpretare le istruzioni e disegnare a video l'interfaccia.

La possibile alternativa invece prevede una separazione tra logica del programma e interfaccia.

Da tempo, i moderni paradigmi di programmazione stanno spingendo su questa separazione ed anche il Web (World Wide Web, questa volta), soprattutto il 2.0, separa i contenuti (HTML) dal layout (CSS).

Esiste un paradigma che ho sempre trovato molto interessante, denominato MVC (Model-View-Controller), presente ormai da alcuni anni ed applicato a tutti i principali linguaggi di programmazione. E' basato sulla separazione dei diversi compiti tra tre moduli software: il *model* che contiene la logica dell'applicazione e l'eventuale accesso ai dati, la *view* che si occupa dell'interfaccia utente e il *controller* che consente il dialogo tra le altre due parti.

Sistemi di *Literate Programming,* come WEB, potrebbero trovare il modo di integrare l'interfaccia sovrapponendo al codice uno strato ulteriore dialogante con esso.

Supponiamo di poter identificare con appositi marcatori la disposizione degli elementi di input ed output direttamente nel codice sorgente. E supponiamo che, sempre con i soliti marcatori, un esperto di User Interface o un grafico sia in grado di dare un layout a questi elementi.

Potremmo mantenere separate logica e visualizzazione e lasciare che il compilatore, facendo corrispondere i marcatori, generi l'output corretto.

In questo modo avremmo inoltre il vantaggio di poter cambiare l'interfaccia senza modificare il programma o viceversa (se gli elementi di input e output non cambiano).

Tornando in ambito umanistico, sarà come avere uno scrittore ed un illustratore, cosa non molto dissimile dalla effettiva produzione letteraria. Entrambi manterrebbero la propria autonomia e la propria creatività potendo lavorare, separatamente, ad un unico prodotto finito. Se poi lo scrittore ha anche capacità grafiche, niente gli impedisce di svolgere entrambi i compiti.

Con un pizzico di umorismo, alla maniera di Donald Knuth, potrei trovare un nome al nuovo sistema: secondo Wikipedia[13], il termine inglese *cobweb* definirebbe una ragnatela ricoperta di polvere, una ragnatela che non è semplice struttura ma possiede una sua consistenza data da uno strato (di polvere) che si aggiunge su di essa. COBWEB potrebbe essere una futura versione di WEB che oltre a produrre un output tipograficamente gradevole sarà in grado, attraverso l'aggiunta

13 Wikipedia voce Spider web.

di uno “strato”, di mostrare a video (o su qualsiasi dispositivo di interazione uomo-macchina) un risultato gradevole graficamente e soprattutto usabile.

Bibliografia

Dahl O.J., Dijkstra E.W., Hoare C.A.R. 1972. *Structured Programming*, Academic Press, London and New York

Knuth, Donald. 1984. *Litarate Programming* in “The Computer Journal” (1984) 27 (2) pp. 97-111.

Knuth, Donald. 1984(2). *The TeXbook.* Reading, Massachusetts. Addison-Wesley

Knuth, Donald. 1986. *The METAFONTbook.* Reading, Massachusetts. Addison-Wesley

Knuth, Donald 2011. *The Art of Computer Programming: Volumes 1-4A Boxed Set 3rd Edition* Reading, Massachusetts. Addison-Wesley.

Donald Knuth's Home Page http://www-cs-faculty.stanford.edu/~uno/index.html (visitato il 15/07/2012)

Wikipedia, voce Donald Knuth http://en.wikipedia.org/wiki/Donald_Knuth (visitato il 15/07/2012)

Wikipedia, voce Spider web http://en.wikipedia.org/wiki/Spider_web (visitato il 15/07/2012)