Rappresentazione di algoritmi mediante le carte di Nassi-Shneiderman

Crescenzio Gallo, Michelangelo De Bonis, Pasquale Cariello
IEEE Members
[*c.gallo, m.debonis*]*@ieee.org*, *pasquale.cariello@istruzione.it*
Dipartimento di Scienze Economiche, Matematiche e Statistiche
Dipartimento di Scienze Biomediche
Università di Foggia
Largo Papa Giovanni Paolo II n.1, 71121 Foggia, Italy
Phone +39 0881-753708 Fax +39 0881-753709

Sommario

Esistono forme "alternative" di strumenti di lavoro per la stesura degli algoritmi. Uno di questi strumenti è noto come le "carte di Nassi-Shneiderman". Queste potenti tecniche sono a tutt'oggi sottovalutate rispetto ad altre metodologie che nel corso degli anni hanno trovato una loro fisionomia all'interno degli interventi didattici come ad esempio la pseucodifica o i "diagrammi di flusso" tradizionali. Scopo di questo paper è quello di mostrare la valenza didattica delle carte di Nassi-Shneiderman, confrontare questa metodologia con tecniche più consolidate, come i diagrammi di flusso, ed infine esporre dei brevi esempi di applicazione dello strumento.

Indice

Elenco delle figure

1 Introduzione

Un efficace metodo di progettazione e un buon metodo per effettuare la documentazione del progetto sono elementi da usare in modo sinergico per realizzare una efficiente programmazione strutturata. La ricerca di queste metodologie ha portato allo sviluppo di diverse strategie e di molte tecniche [5]. Una di queste tecniche sono senza dubbio i "diagrammi a blocco strutturati di Nassi e Shneiderman".

I diagrammi di Nassi-Shneiderman, o meglio noti come le *Carte di Nassi-Shneiderman* hanno molte caratteristiche che ne fanno lo strumento ideale per l'uso nella programmazione strutturata di tipo top-down. Infatti gli autori hanno introdotto questa metodologia di progettazione nel 1973 ma sfortunatamente questo strumento di descrizione degli algoritmi non ha avuto fortuna, soprattutto nell'ambiente didattico. Altri strumenti vengono usati per poter descrivere gli algoritmi: la pseudocodifica o i diagrammi di flusso (*flow-chart*) di tipo tradizionale. L'uso delle carte di Nassi-Shneiderman però è particolarmente diffuso in quasi tutti coloro che sono, per necessità, nella condizione di progettare e documentare programmi sempre più complessi ed in modo semplice e preciso.

2 La programmazione strutturata

È noto come il lavoro di organizzare un programma consiste, in prima approssimazione, nella stesura della sequenza di operazioni che devono essere eseguite per ottenere uno scopo prefissato.

Dal punto di vista informatico la stesura di un programma è uno dei passi necessari all'interno di un lavoro più complesso che costituisce la realizzazione di un applicativo software.

La programmazione è un'attività complessa che può essere suddivisa in almeno quattro fasi:

- definizione del problema, e dei realtivi dati di input e di output;
- organizzazione dell'*algoritmo* risolutivo;
- stesura del programma, cioè la traduzione dell'algoritmo nel linguaggio di programmazione;
- prove di esecuzione del programma realizzato.

Poiché la stesura dell'algoritmo risolutivo è una delle fasi fondamentali del lavoro di programmazione, si tratta di definire un insieme di regole e di

linee direttrici, che devono essere eseguite per una corretta organizzazione del lavoro. Tali regole hanno lo scopo di trasformare la programmazione, da attività laboriosa e disordinata, in attività sistematica e orientata al raggiungimento di un buon livello di qualità, e costituiscono il metodo di lavoro che va sotto il nome di programmazione strutturata.

Obiettivo didattico è presentare ai "giovani" programmatori metodi di strutturazione degli algoritmi conforme ad un insieme di regole che rendono più facile la costruzione, la lettura e la manutenzione di un programma.

Le idee chiave della programmazione strutturata si possono ricondurre alla critica della struttura di controllo del salto incondizionato (il fantomatico *goto*), che rappresentava, negli anni sessanta, lo strumento fondamentale per la definizione di algoritmi complessi nel software. Dijkstra [1] discusse approfonditamente gli effetti deleteri dell'istruzione *goto* sulla qualità del software, e in particolare sulla sua leggibilità e modificabilità (il cosiddetto problema dello *spaghetti-code*).

Un'altra celebre pubblicazione che giocò un ruolo fondamentale per l'affermazione definitiva della programmazione strutturata fu quella in cui Corrado Böhm e Giuseppe Jacopini [2] dimostrarono il loro celebre teorema, secondo il quale:

> *qualsiasi programma scritto usando il* goto *può essere riscritto senza, a patto di avere a disposizione altri tre tipi di strutture di controllo: sequenza, ripetizione e alternativa.*

Il risultato del teorema di Böhm-Jacopini può anche essere espresso dicendo che, dato un qualunque diagramma di flusso, ne esiste almeno un altro che a parità di input produce sempre gli stessi output del primo e in cui i nodi del diagramma di flusso rispettano un certo insieme di vincoli.

Ovviamente con il termine *programmazione strutturata* non si intende solamente, come molti programmatori credono, un insieme rigido di regole di codifica e restrizioni [4]. La programmazione strutturata è uno stile, un atteggiamento nei confronti della programmazione che inizia con la fondamentale consapevolezza di quali sono gli obiettivi del processo di programmazione. Nella letteratura classica, gli obiettivi della programmazione sono sempre stati *correttezza*, *efficienza* e per ultima, ma non meno importante, la *creatività*. Di questi, la correttezza è l'unico obiettivo di programmazione valido ancora oggi. Infatti l'efficienza è diventata di minore importanza con l'avvento di dispositivi hardware di velocità elevata e sistemi operativi che implementano in modo efficace tecniche di memorie virtuali. La creatività, quell'atto di fantasia, di inventiva e di immaginazione del programmatore tale da trovare soluzioni originali e alternative, è stata sempre orientata verso

un atteggiamento più pragmatico e banale, qualcosa che ricadesse nel conosciuto e meno "incognito" estro con risultati non sempre positivi per la programmazione.

Per quanto riguarda gli obiettivi della programmazione "moderna" la correttezza rimane di primaria importanza; tuttavia, la *manutenzione*, cioè la facilità di correzione degli errori, la *modificabilità*, la facilità nell'apportare le modifiche come richieste da Dijkstra, e la *leggibilità*, la chiarezza nella lettura del codice di un programma, hanno sostituito l'efficienza e la creatività come caratteristiche auspicabili nella realizzazione di programmi.

Questi nuovi obiettivi sono stati a lungo realizzati ed insegnati ai giovani programmatori con le strutture di controllo che il Teorema di Böhm-Jacopini citava per la realizzazione di un programma strutturato.

3 Le strutture di controllo

Le strutture di controllo ammesse dai linguaggi strutturati, e le regole sintattiche e semantiche del loro uso, possono variare nei dettagli specifici; tuttavia, devono essere rispettati un insieme di requisiti fondamentali.

Completezza. Un linguaggio strutturato deve fornire la sequenza, almeno una struttura di tipo alternativa, e almeno una struttura di tipo iterativa.

Singolo punto di ingresso e di uscita. Idealmente, ogni struttura di controllo, completa delle istruzioni controllate, deve poter essere considerata come una singola macro-istruzione dal punto di vista del controllo complessivo, con un ben identificato punto di ingresso e un ben identificato punto di uscita.

Componibilità. Ogni struttura di controllo può avere fra le sue istruzioni controllate, incapsulate, altre strutture di controllo (senza limiti e senza ordini di precedenza).

Gli ultimi due vincoli fanno sì che, per quanto concerne il flusso del controllo, ciascuna struttura di controllo definisca un ambito completamente isolato dalle altre, e incapace di interferire o subire interferenze. Questo rappresenta un requisito indispensabile per l'applicazione di metodologie di progetto e sviluppo tradizionalmente legate alla programmazione strutturata, come la progettazione top-down e lo sviluppo per raffinamenti successivi.

Ma proprio sugli ultimi due requisiti fondamentali l'uso di certi tipi di metodologie di progetto può indurre in confusione o non essere progettazione

coerente nell'ambito di una ingegnerizzazione del software. Le tre strutture principali utilizzate con il metodo dei *diagrammi di flusso* di tipo classico sono esposte nelle sezioni successive.

3.1 Sequenza

La sequenza è la struttura di esecuzione delle istruzioni dell'algoritmo ed è schematizzata dalla Figura 1. Si noti come il relativo punto di ingresso e il punto di uscita sono resi visibili da dei puntini rossi, in cui quello superiore identifica l'ingresso e quello inferiore l'uscita.

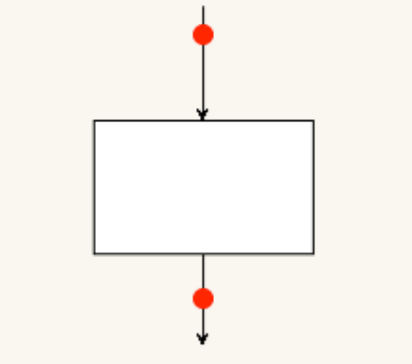


Figura 1: Blocco della sequenza nel *Diagramma di Flusso*

3.2 Alternativa

L' alternativa (Figura 2) è la struttura che implementa una scelta nel controllo dell'esecuzione di alcune parti dell'algoritmo. Anche in questa figura i simboli rossi identificano l'unico punto di ingresso (quello in alto) e l'unico punto di uscita (quello in basso).

3.3 Iterazione

L'alternativa (Figura 3) è la struttura che implementa nell'algoritmo la possibilità di ritornare, in un modo *corretto*, indietro nel fluire del "flusso" dell'algoritmo. L'iterazione può essere espressa in alcune varianti:

iterazione precondizionale: in cui la condizione indicata dal rombo nei blocchi dei diagrammi di flusso è posta all'inizio del nucleo dell'iterazione stessa;

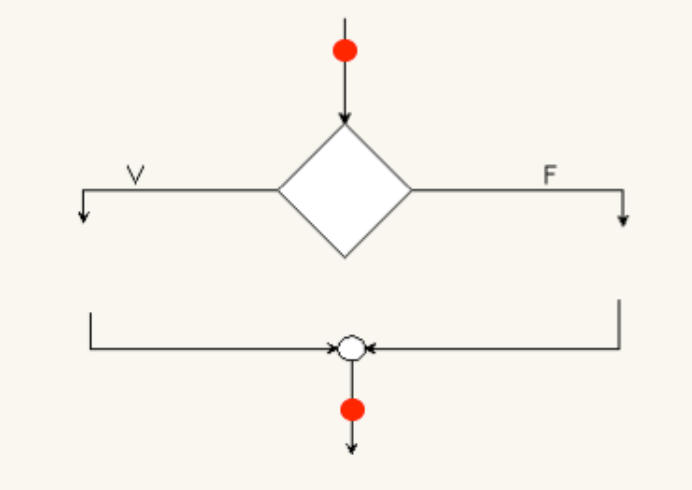


Figura 2: Blocco dell'alternativa nel *Diagramma di Flusso*

iterazione postcondizionale: in cui la condizione è posta alla fine del nucleo;

iterazione con contatore: in cui il nucleo è ripetuto un numero precisato di volte senza verificare il valore di condizione (questo tipo di iterazione è anche chiamata enumerativa).

Comunque in qualunque variante si consideri l'iterazione, anche con questa struttura di controllo si rispetta il punto singolo di ingresso e di uscita.

3.4 Prime considerazioni

La domanda che chiunque abbia un ruolo nella didattica di queste metodologie si pone è: "Perché chi si avvicina alla programmazione con queste tecniche incontra problemi e, si spera solo nei primi momenti, fa confusione con le diverse strutture di controllo?".

La risposta è semplice: tutta colpa del *rombo*. Infatti in tutte le strutture di controllo, tranne la sequenza, fa capolino il rombo che identifica in modo grafico il verificarsi della condizione. Il fatto che non sia "univoca" la rappresentazione di ogni singola unità, o struttura di controllo, porta alla logica confusione. Questo senso di smarrimento di solito passa con la comprensione delle strutture e l'astrazione dei macro-blocchi rispetto ai singoli elementi grafici che lo compongono. Di solito, però.

Il fatto stesso che sussista una possibilità di confusione e di incertezza pone gli esperti del settore a porsi il quesito successivo: "È realmente il

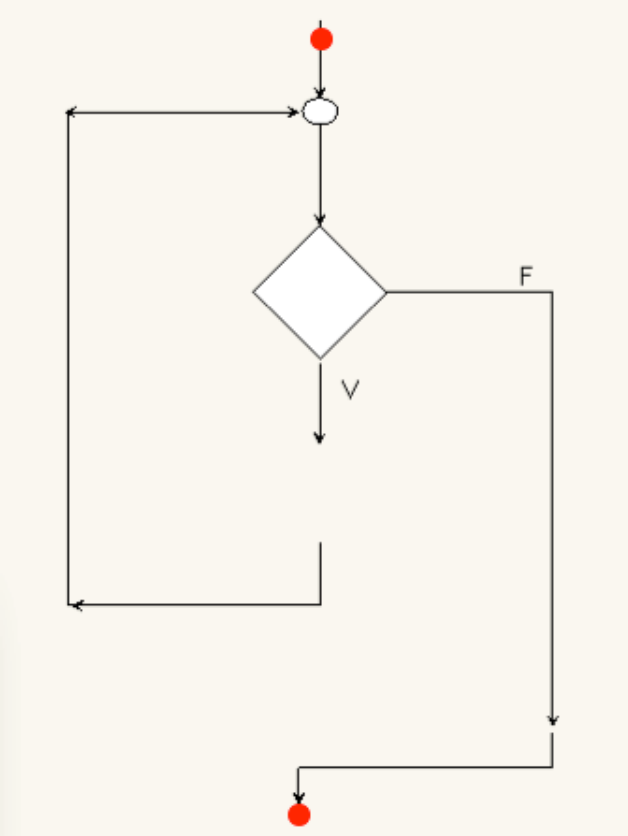


Figura 3: Blocco dell'iterazione precondizionale nel *Diagramma di Flusso*

metodo migliore per rappresentare graficamente l'algoritmo? o ne esistono di alternativi e di migliori?". La risposta a questa domanda potrebbero essere i diagrammi a blocco di Nassi-Shneiderman.

4 Carte di Nassi-Shneiderman

Gli autori Nassi e Shneiderman pubblicarono [3] una nuova metodologia di realizzare i diagrammi di flusso con una struttura strettamente affine a quello del codice strutturato. I vantaggi sostenuti dall'utilizzo di questa metodologia per la progettazione sono i seguenti:

1. L'ambito di iterazione è ben definito e visibile.
2. L'ambito di clausole IF THEN ELSE è ben definita e visibile.
3. Lo scopo delle variabili locali e globali è immediatamente "visibile".
4. Trasferimenti *arbitrari* di controllo sono impossibili.
5. Strutture di pensiero completo possono e devono essere illustrate in una stessa pagina (per esempio non sono ammessi connettori di tipo *off-page*).
6. La ricorsione è una rappresentazione banale.
7. Questi diagrammi sono adattabili alle peculiarità del sistema o del linguaggio di programmazione che vengono utilizzati.

Combinando e nidificando le strutture di base delle carte di Nassi-Shneiderman, che sono tutti rettangolari, un programmatore può progettare in modo strutturato, senza incorrere in errori di progettazione non strutturata.

4.1 Processo

Il simbolo base è denominato "processo" (Figura 4). Graficamente è un rettangolo che rappresenta assegnazioni, chiamate, operazioni di input/output, o qualsiasi altra operazione di tipo sequenziale. Inoltre, un simbolo di processo può contenere al suo interno altri simboli annidati.

Il simbolo di processo può essere di qualsiasi dimensioni scelte a condizione che il simbolo si inserisca in una sola pagina.

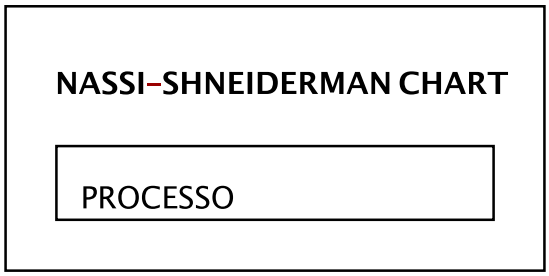


Figura 4: Blocco del processo nel *Diagramma di Nassi-Shneiderman*

4.2 Alternativa

Il simbolo utilizzato per rappresentare una decisione è mostrato in Figura 5. Questo simbolo corrispondente alla Figura 2 per i diagrammi di flusso o alla clausola IF THEN ELSE e contiene il test, o la decisione, nel triangolo superiore e gli esiti possibili del test nei triangoli inferiori. Le risposte possibili "SI" e "NO" possono essere sostituite dalla sintassi booleana "True" (Vero) e "False" (Falso), e non c'è nessuna predilizione per un particolare passaggio a destra o a sinistra, anche se per consistenza con la codifica il ramo *Vero* dovrebbe essere sempre presente. I rettangoli contengono le funzioni da eseguire per ognuno dei risultati. Si noti sia il segmento del THEN (Vero) che quello dell'ELSE (Falso) sono in realtà simboli di processo e quindi esso può contenere qualsiasi valido elemento di PROCESSO o strutture nidificate.

Struttura più articolata dell'alternativa è quella definita dalla clausola CASE. In cui si testa una variabile e, in funzione del suo contenuto, si esegue il segmento del blocco corrispondente. La Figura 6 mostra la simbologia di questa alternativa estesa e, per certi versi, più potente, in quanto si richiede l'uso oculato da parte del progettista per essere sicuri che tutte le condizioni scelte siano mutuamente esclusive e si possa così coprire tutte le condizioni necessarie.

4.3 Iterazione

I processi di ripetizione sono rappresentati da un simbolo di iterazione. Uno dei tre simboli possono essere utilizzati a seconda se l'iterazione è di tipo precondizionale, Figura 7 (controllo del ciclo all'inizio del nucleo), oppure iterazione postcondizionale, Figura 8 (controllo del ciclo alla fine del nucleo), o infine iterazione di tipo enumerativa, Figura 9 (nessun controllo *classico* ma il conteggio dell'esecuzione del nucleo).

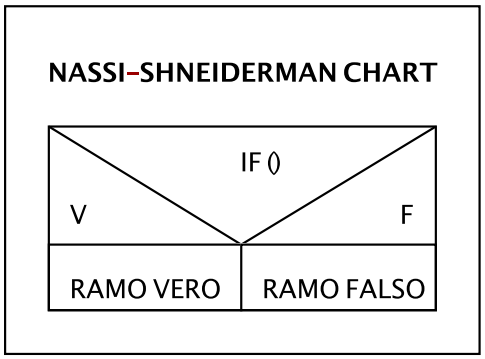


Figura 5: Blocco dell'alternativa nel *Diagramma di Nassi-Shneiderman*

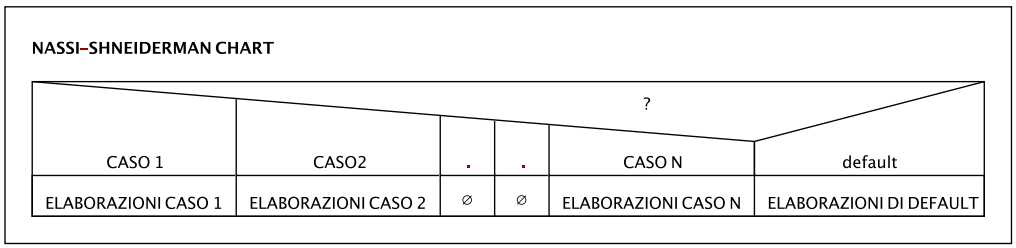


Figura 6: Selezione multipla nel *Diagramma di Nassi-Shneiderman*

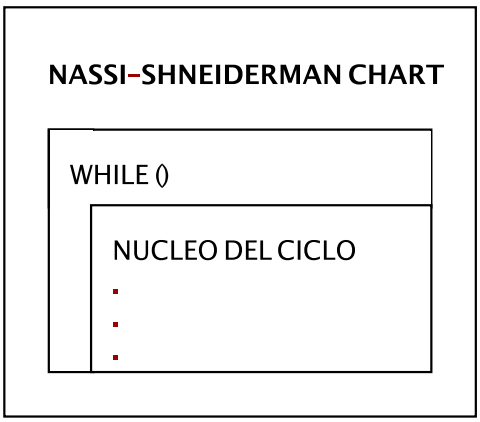


Figura 7: Blocco dell'iterazione precondizionale nel *Diagramma di Nassi-Shneiderman*

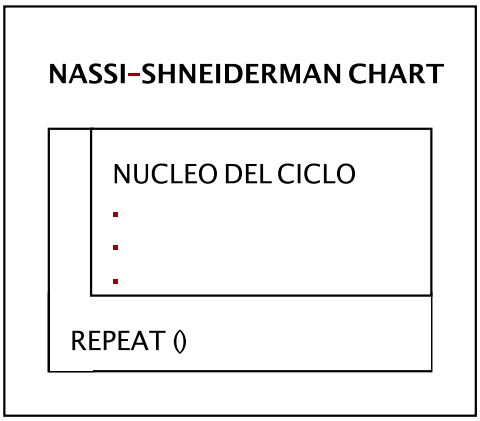


Figura 8: Blocco dell'iterazione postcondizionale nel *Diagramma di Nassi-Shneiderman*

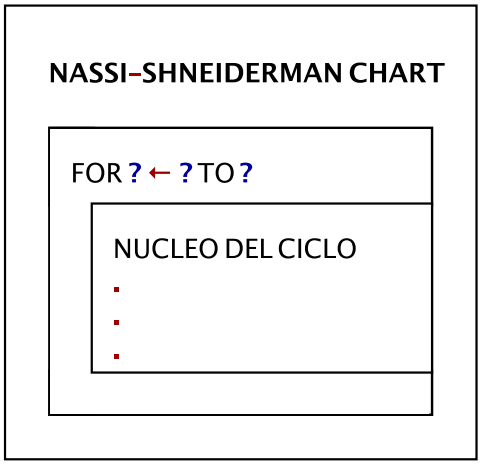


Figura 9: Blocco dell'iterazione enumerativa nel *Diagramma di Nassi-Shneiderman*

4.4 Procedure e Programmazione parallela

I blocchi della Figura 10 e Figura 11 descrivono rispettivamente la chiamata ad una procedura, o ad una funzione, e l'esecuzione di procedure parallele ove si ha possibilità di programmazione *multi-thread*.

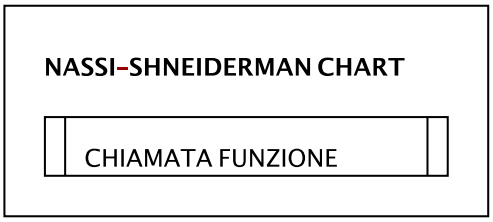


Figura 10: Blocco della chiamata ad una procedura (o funzione) nel *Diagramma di Nassi-Shneiderman*

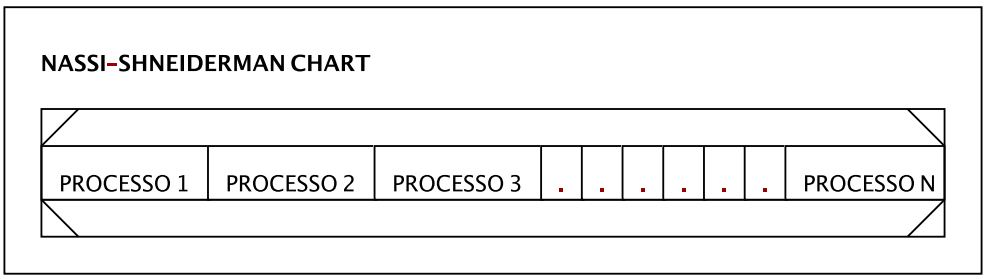


Figura 11: Blocco dell'esecuzione contemporanea di procedure (o funzioni) nel *Diagramma di Nassi-Shneiderman*

4.5 Considerazioni sui simboli

I Simboli per la progettazione di algoritmi con l'uso delle carte di Nassi-Shneiderman sono tutti quelli espressi nelle Sezioni precedenti. Ovviamente la caratteristica del *nesting* delle strutture, cioè dell'incapsulamento, per la realizzazione di algoritmi sempre più complessi e articolati è un'estensione evidente dell'uso dei simboli base.

5 Realizzazione di un progetto con l'uso delle carte di Nassi-Shneiderman

Le carte di Nassi-Shneiderman sono stati sviluppati, meglio dei diagrammi di flusso, per descrivere la logica di un programma strutturato. Disegnare il diagramma e sviluppare la logica di definizione dell'algoritmo sono due fasi che viaggiano di pari passo, con i vincoli delle carte di Nassi-Shneiderman (pagina singola, senza simboli ramo ma solo blocchi consecutivi o annidati) costringono lo sviluppo di un progetto strutturato, che a sua volta porta a codice strutturato.

5.1 Come iniziare

Supponiamo che un design funzionale per un progetto è stato completato, e che una tecnica di costruzione modulare è stata utilizzata per determinare la funzione di ingresso e uscita per ogni modulo da programmare. Il programmatore è ora pronto per la progettazione logica per la codifica strutturata.

La carta di Nassi-Shneiderman inizia sempre con un rettangolo disegnato nella parte superiore della pagina.

Questo blocco potrebbe essere uno qualsiasi dei simboli di Nassi-Shneiderman, a seconda della funzione del modulo. Se il modulo richiede l'inizializzazione di alcune variabili, il primo blocco è probabilmente un simbolo processo di elaborazione. Se la funzione del modulo è eseguire ripetutamente un blocco, molto probabilmente occorrerà iniziare con un simbolo iterativo. Se, invece, la funzione da eseguire è subordinata al verificarsi di una condizione, il simbolo della decisione è quello da utilizzare.

5.2 Organizzare la Struttura

Quando un blocco viene disegnato simboleggia una decisione, cioè il programmatore deve prendere una decisione circa la reale assegnazione di percorsi di elaborazione nelle carte di Nassi-Shneiderman. Una tecnica efficace è quella di individuare sulla destra il sentiero che in codifica sarebbe equivalente alla clausola THEN ("allora") di un affermazione IF ("se"), e per localizzare sulla sinistra il percorso equivalente alla clausola "else". Un'altra tecnica consistente per l'assegnazione del cammino quello in cui c'è più facilità per la leggibilità.

5.3 Organizzare la Modularità

Il programmatore che si accinge a sviluppare una progettazione con i diagrammi a blocchi classici può rendersi conto di aver consumato lo spazio tra blocchi di iterazione e blocchi di alternative annidati tra di loro. Non di poco conto è il rendersi conto di non capire immediatamente lo scopo di un modulo perché ci si perde nella miriade di controlli che il modulo effettua a scapito della facilità della lettura.

Queste problematiche sono risolte alla base con l'uso delle carte di Nassi-Shneiderman. Infatti qualsiasi parte del diagramma di Nassi-Shneiderman può essere rimosso dalla routine principale, sostituito da un blocco, processo, di elaborazione in un cui si effettua la chiamata ad una *subroutine* che è visualizzata a parte. Ovviamente la scelta degli blocchi da sostituire variano in funzione della capacità del progettista di riconoscere gruppi di istruzioni e parti di processi che possono essere considerati funzioni del programma.

5.4 Esempi

Nella seguente sezione si analizzano una serie di esempi di algoritmi realizzate con le carte di Nassi-Shneiderman e con la tecnica più tradizionale del diagramma di flusso (Figure 12–14). Pe realizzare i diagrammi di Nassi-Shneiderman si è usato un software freeware "Structorizer" [6] che oltre alla realizzazione delle carte effettua anche l'esportazione del codice in un linguaggio a scelta (Cfr. Sez. 6.1 Figure 15 e 16).

Non serve una grande riflessione per notare come tutti i vantaggi della tecnica di realizzazione degli algoritmi con le carte di Nass-Shneiderman sono resi noti da questi semplici esempi.

6 Programmazione dalle Carte di Nassi-Shneiderman

Una volta che la progettazione logica per il modulo è completato, si può effettuare la fase di codifica e di test del modulo. Entrambe queste fasi possono usare le carte di Nassi-Shneiderman come linee guida di realizzazione strutturata.

6.1 Codifica

Tradurre il codice, in particolare di un linguaggio ad alto livello, dalle carte di Nassi-Shneiderman è molto semplice; questa facilità è uno dei motivi

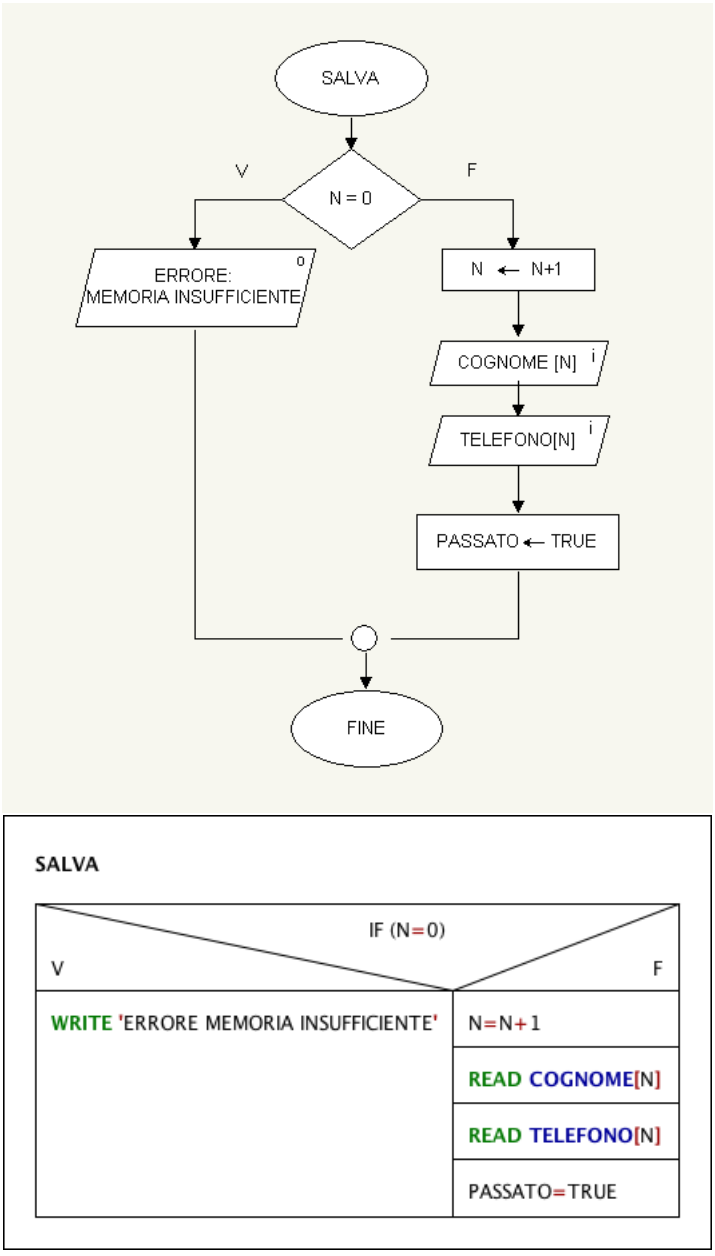


Figura 12: Funzione di caricamento (Flow-Chart – Nassi-Shneiderman Chart)

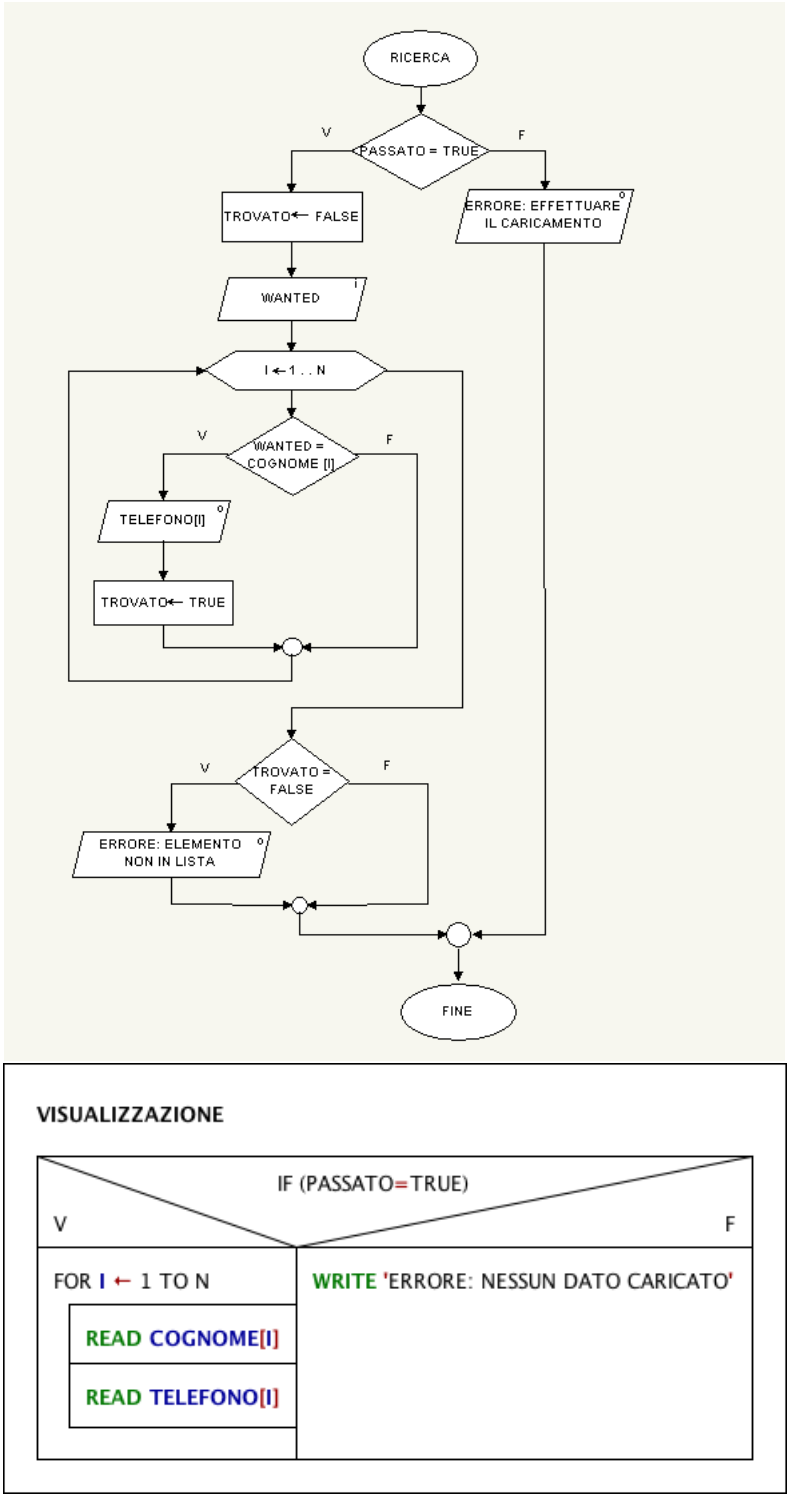
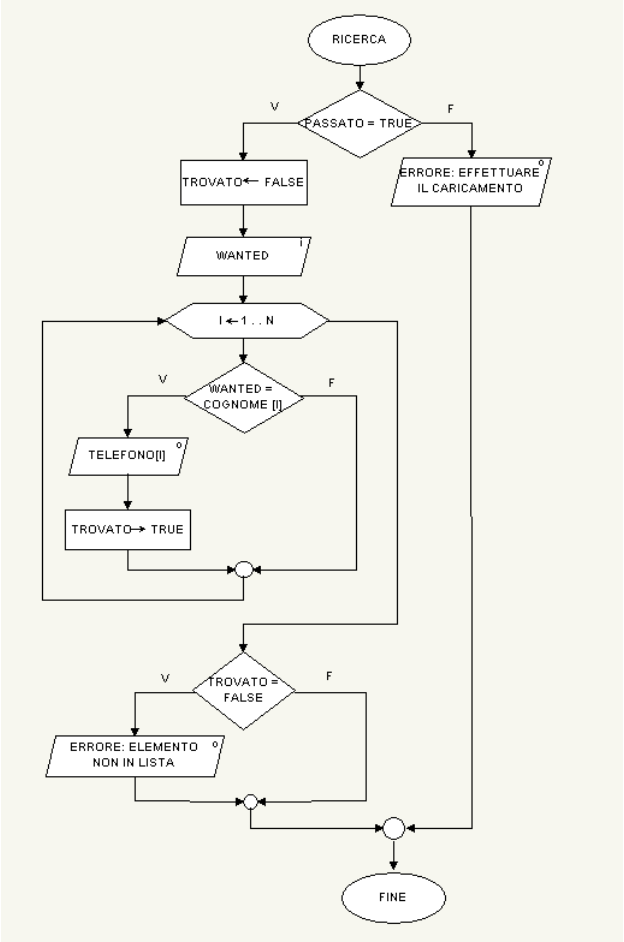


Figura 13: Funzione di visualizzazione (Flow-Chart – Nassi-Shneiderman Chart)



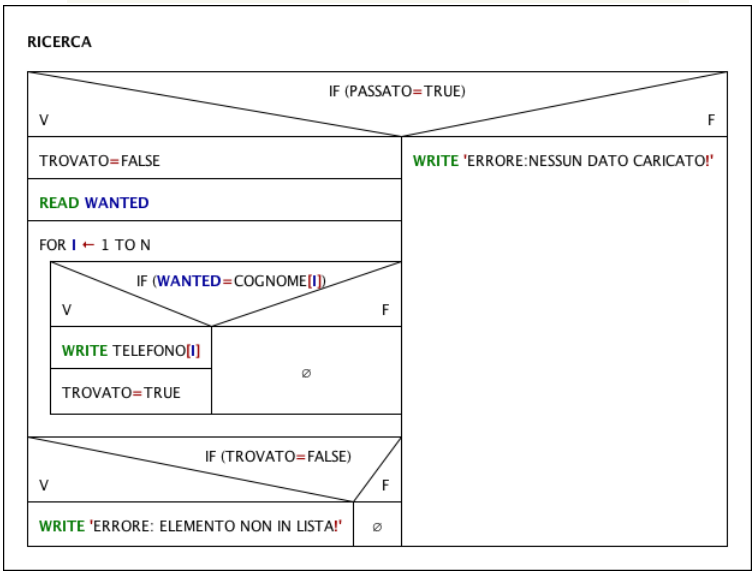


Figura 14: Funzione di ricerca (Flow-Chart – Nassi-Shneiderman Chart)

per cui le carte di Nassi-Shneiderman sono state accolte con entusiasmo dai programmatori che le hanno provate.

Il codice sarà strutturato, non ci sarà alcuna possibilità di un ramo non chiuso, che il codice segmenti in modo errato. Dichiarazioni IF THEN ELSE sono ben definite dai diagrammi come anche i limiti delle strutture iterative. La Figura 15 rappresenta un esempio di come il software *Structorizer* possa facilmente esportare i diagrammi di Nassi-Shneiderman in un linguaggio ad alto livello come il Pascal/Delphi (Figura 16).

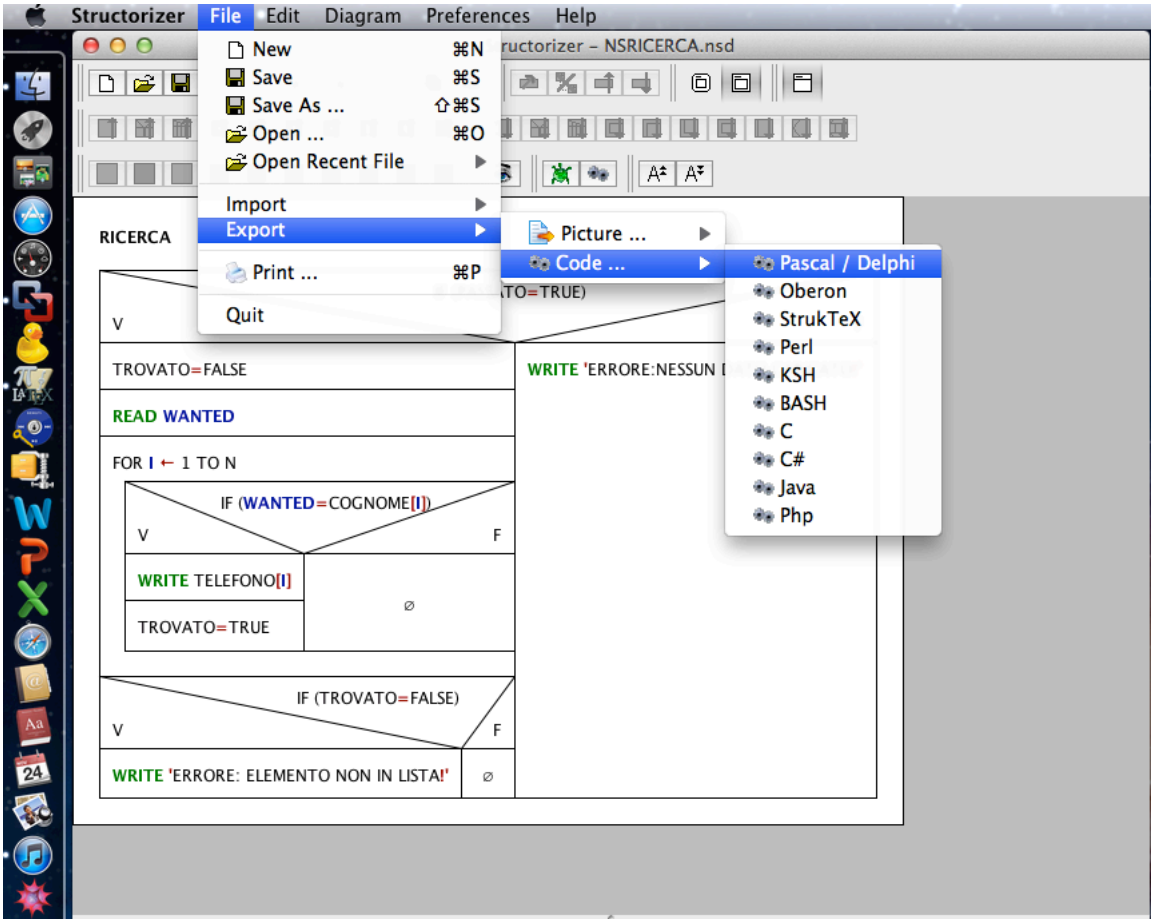


Figura 15: Esportazione automatica nei codici sorgenti strutturati

6.2 Test

Le carte di Nassi-Shneiderman possono essere usate anche come linee guida durante la fase di testing del modulo. Il test può essere effettuato direttamen-

```
 1 program RICERCA;
 2
 3 // declare your variables here
 4
 5 begin
 6   if (PASSATO=TRUE) then
 7   begin
 8     TROVATO=FALSE;
 9     readln(WANTED);
10     for I:=1 to N do
11     begin
12       if (WANTED=COGNOME[I]) then
13       begin
14         writeln(TELEFONO[I]);
15         TROVATO=TRUE;
16       end;
17     end;
18     if (TROVATO=FALSE) then
19     begin
20       writeln('ERRORE: ELEMENTO NON IN LISTA!');
21     end;
22   end
23   else
24   begin
25     writeln('ERRORE:NESSUN DATO CARICATO!');
26   end;
27 end.
```

Figura 16: Esportazione codice Pascal/Delphi

te sui blocchi delle carte di Nassi-Shneiderman. I casi di utilizzo del modulo possono essere valutati dall'analisi delle carte e, cosa importantissima, la figura professionale che effettua il test può essere una figura con competenze non specifiche nel linguaggio sorgente in cui il modulo sarà implementato. Questo perché il test è effettuato sulla logica di funzionamento delle carte a livello grafico.

7 Le carte di Nassi-Shneiderman come documentazione del programma

Le carte di Nassi-Shneiderman sono una rappresentazione grafica della progettazione "logica" di un modulo, di un progetto e, in fondo, del relativo codice. Questo lo rende un ottimo strumento didattico da utilizzare per educare altri programmatori sulla funzione del modulo. Quindi le carte di Nassi-Shneiderman forniscono al programmatore un sistema di manutenzione del codice stesso con un riferimento rapido per trovare nel codice qualsiasi funzione logica.

8 Conclusioni

Le carte di Nassi-Shneiderman hanno dimostrato di essere utili in quasi tutte le fasi dello sviluppo di un programma: dalla progettazione iniziale attraverso il *walk-through*, la codifica, la fase di test e la validazione degli utenti. Una tecnica grafica eccellente, le carte di Nassi-Shneiderman forniscono un semplice, ma elegante linguaggio che, volutamente, è compatibile con lo stile della programmazione strutturata.

Infatti come hanno scritto Nassi e Shneiderman:

> "I programmatori che imparano a progettare algoritmi con questi simboli non svilupperanno mai le cattive abitudini che altri sistemi di notazione di diagrammi a blocchi permettono. Dato che non più di quindici o venti simboli possono essere disegnati su di un singolo foglio di carta, il programmatore deve rendere necessariamente modulare il suo programma, suddividendolo in sezioni significative. La tentazione di utilizzare connettori *off-page*, che portano solo alla confusione, è stata eliminata. Infine, la facilità con cui un diagramma può essere tradotto in un programma strutturato è sorprendente."

Le carte di Nassi-Shneiderman non sono note e, quindi, il metodo non è stato pienamente sfruttato. C'è un grande potenziale nelle applicazioni in molte aree per l'utilizzo di questi diagrammi strutturati. Se l'impegno, anche didattico, è di scommettere sul pensiero logico e su come la programmazione può essere connessa ad altre discipline, non possiamo perdere l'opportunità di far diventare questa metodologia il buon *metodo* della programmazione strutturata.

Riferimenti bibliografici

[1] Dijkstra E. (1968) *Goto statement considered harmful.*

[2] Böhm C., Jacopini G. (1966) *Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules.*

[3] Nassi I., Shneiderman B. (1974) *Flow Chart Techniques for Structured Programming.*

[4] Yolder C. M., Schrag M. (1978) *Nassi-Shneiderman charts an alternative to flowcharts for design.*

[5] Capozza F., Gallo C., Esposito F. (1985) *Strumenti Grafici di Software Design.* Sistemi e Automazione.

[6] *Structorizer*, Software Open Source per piattaforme Mac, Windows, Linux. http://structorizer.fisch.lu (ultimo accesso 31/10/2011).