La Natura dei Programmi

Un quadro di riferimento per insegnanti

ITiCSE 2022 - Working Group 5

Violetta Lonati, Andrej Brodnik, Tim Bell, Andrew Paul Csizmadia,
Liesbeth De Mol, Henry Hickman, Therese Keane, Claudio Mirolo, Mattia Monga

Cos'è un programma informatico? Questa potrebbe sembrare una domanda banale, invece i programmi sono strane creature, con una natura sfaccettata che sfugge definizioni semplici. L'obiettivo di questo documento è fornire a insegnanti e altri operatori culturali una prospettiva ampia e integrata sulla Natura dei Programmi, allo scopo di informare la progettazione e l'implementazione di attività didattiche orientate a presentare agli studenti i principi base dell'informatica e della programmazione.
Nella prima parte del documento descriviamo la natura dei programmi mediante sei diverse sfaccettature, ognuna delle quali mostra un modo diverso di intendere cosa sia un programma. Nella seconda parte discutiamo cosa comporti il processo di creazione e sviluppo dei programmi, al di là della semplice "scrittura di codice" (*coding)*, e mettiamo in relazione il concetto di programma con altri concetti centrali nella programmazione e nell'informatica.

Traduzione in italiano del testo originale disponibile online all'indirizzo
https://drive.google.com/file/d/1lhVPDhSHu3ivRXvcK7BVEhSmpqz5izow
Per una introduzione in italiano all'iniziativa si veda:
https://mondodigitale.aicanet.net/2022-4/Articoli/01%20Di%20cosa%20parliamo%20quando%20parliamo%20di%20programmi.pdf

Sommario

Introduzione

I programmi informatici permeano le nostre vite più di quanto molte persone possano riconoscere. Ogni "*app*" e dispositivo digitale che usiamo coinvolge l'esecuzione di programmi che sono stati sviluppati per uno scopo determinato. Per esempio, c'è un programma dietro a:

- la sveglia che suona al mattino;
- l'elettrodomestico che usiamo per preparare la colazione;
- la condivisione di eventi tramite i social network;
- le previsioni meteo che condizionano le nostre decisioni su cosa indossare;
- il trasporto (pubblico o privato) che prendiamo e che è molto probabilmente controllato da un computer;
- alcuni compiti che svolgiamo durante il giorno, che potrebbero includere l'uso della posta elettronica, l'elaborazione di testi, strumenti controllati da un computer, documenti consegnati digitalmente e telefonia mobile;
- le casse di un negozio;
- l'applicazione dello *smartphone* che lo fa funzionare come un telefono o una radio;
- e molto altro ancora!

In altre parole, i programmi sono diventati ubiqui e, in molti casi, in maniera invisibile: sono lì, ma non si vedono facilmente.

Dunque, perché i programmi informatici sono così diffusi, e cosa li rende così versatili e potenti? C'è un valore nel fatto che gli studenti capiscano cosa c'è dietro queste *app* che regolano e controllano le nostre vite quotidiane? E che ruolo ha la programmazione?

Rispondere a queste domande non è immediato, poiché la natura sfaccettata dei programmi rende difficile riassumere in poche parole le loro caratteristiche fondamentali. Invece, ci sono così tanti aspetti diversi della programmazione che è interessante esplorare in dettaglio, come faremo in queste pagine. Siamo particolarmente interessati a farlo affinché gli educatori possano sviluppare una visione ampia di cosa sono i programmi e come vengono creati.

Nella Parte I, descriviamo cos'è un programma informatico usando sei diverse sfaccettature, che mostrano modi complementari di intenderne il significato. Nella Parte II analizziamo, senza scegliere nessuna particolare metodologia di sviluppo del *software*, seppure ve ne siano diverse, cosa comporta la creazione dei programmi, in modo da estendere l'idea di programmazione al di là della sola "scrittura di codice" (*coding*).

Il termine "programma" è usato qui in un senso molto ampio, che include sia i semplici programmi che si studiano nei corsi introduttivi di programmazione, sia sistemi *software* o servizi complessi come sistemi operativi o *social network*. Siamo infatti convinti che questi diversi "programmi" - nonostante si differenzino notevolmente in dimensioni, complessità, tipologia di utilizzatori - condividano in realtà la stessa natura, come descritto e argomentato nella descrizione delle sei sfaccettature. Questo non toglie che le suddette differenze abbiano un ruolo molto rilevante, soprattutto nel processo di sviluppo e nell'impatto che hanno sui loro utilizzatori.

Ci limitiamo, però, a programmi che si potrebbero definire "tradizionali", ci concentriamo cioè su programmi che sono un'implementazione di algoritmi progettati da programmatori umani, e per i quali è quindi possibile spiegare in modo diretto come ottengano il risultato desiderato. Tale decisione lascia fuori dal nostro ambito programmi la cui logica implementativa non è prodotta direttamente da

programmatori umani ma da una macchina (ad esempio, attraverso programmazione evolutiva o tecniche di apprendimento automatico).

Parte I: la natura sfaccettata dei programmi

I programmi sono molte cose diverse per persone diverse: per alcuni sono strumenti importanti per il lavoro quotidiano, e hanno addirittura creato lavori che non esistevano fino a qualche anno fa (come, ad esempio, *YouTuber*, *Influencer* o *Data Scientist*); per altri sono oggetti di per sé degni di studio; e per altri ancora rappresentano una minaccia per l'ambiente e la società. In ambito scolastico, la programmazione può essere un'attività di *problem-solving* che può aprire nuove possibilità, oppure potrebbe essere semplicemente un interessante esercizio intellettuale.

Le motivazioni e i metodi per insegnare agli studenti a programmare variano anche a seconda di quale delle visioni sopra menzionate viene enfatizzata. Il nostro obiettivo è esplorare a fondo cosa sia un programma affinché coloro che sono coinvolti nell'istruzione (sia insegnanti che studenti) possano sviluppare una visione più ricca del ruolo di questo componente fondamentale dell'informatica.

Per riflettere una visione così ampia, abbiamo identificato sei sfaccettature, che insieme ci danno una comprensione diversificata dei programmi informatici, come segue:

- I programmi sono usati come strumenti
- I programmi sono tecnologia creata dall'uomo
- I programmi sono oggetti fisici
- I programmi sono entità astratte
- I programmi sono entità eseguibili
- I programmi sono entità linguistico-notazionali

È proprio perché i programmi coinvolgono tutte queste diverse sfaccettature che possono essere differenziati da altre tecnologie e idee. La Figura 1 mostra visivamente le sei diverse sfaccettature dei programmi di cui discuteremo di seguito.

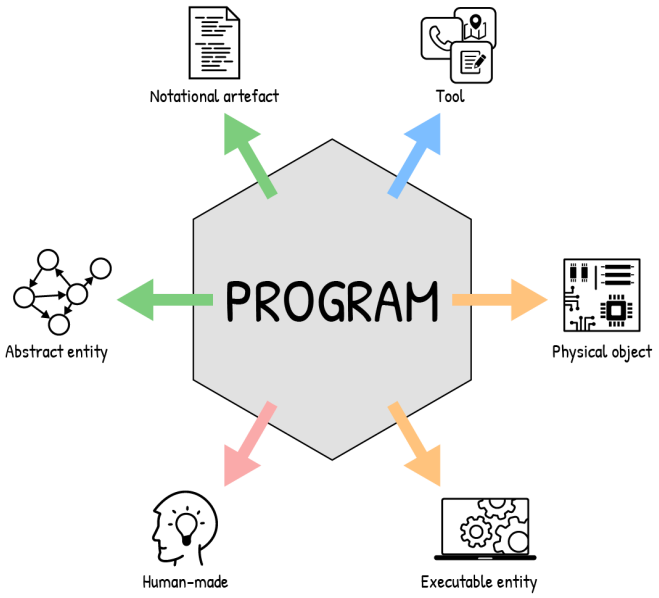


Fig. 1: Le sei sfaccettature dei programmi e le loro interrelazioni
(i colori delle frecce riflettono le mappe concettuali della Parte II)

Le sfaccettature sono complementari e, a volte, apparentemente contraddittorie. Oltre ad esplorare ciascuna delle sei sfaccettature, discuteremo tre dicotomie tra coppie di sfaccettature che si trovano ai lati opposti nel diagramma: notazionale vs eseguibile, astratto vs fisico, e creato dall'uomo vs essere uno strumento che gli umani usano.

Il resto di questa sezione esplora le sfaccettature in dettaglio. Ognuna di queste sarà presentata tramite una descrizione, una discussione del suo impatto, alcuni esempi di come la sfaccettatura potrebbe essere osservata in azione e, per ogni coppia opposta, una breve esplorazione della dicotomia che comporta. Per risparmiare tempo, vi incoraggiamo a leggere la descrizione, e poi a sfogliare le sezioni extra per idee di supporto utili.

Un programma è uno *Strumento*

Per molte persone che utilizzano dispositivi digitali, un programma è uno strumento che le supporta nel loro lavoro e nel tempo libero. Le persone dipendono dagli strumenti per le loro attività quotidiane. I programmi (come un software di fotoritocco, una sveglia, un foglio di calcolo, un videogioco, un editor video o un sistema di previsione del tempo) vengono utilizzati per consentire alle persone di lavorare più efficacemente, di essere più creative e di comunicare con un pubblico più ampio di quanto potrebbero senza lo strumento.

Un programma può trasformare lo stesso dispositivo fisico (come un telefono cellulare) in un sistema di messaggistica, un blocco per appunti, un sistema di previsione del tempo, un gioco o addirittura una fotocamera se sono disponibili i componenti hardware adeguati (come lenti ottiche e sensori di luce). Questo è reso possibile dal fatto che i dispositivi sono programmabili, e questa flessibilità ha portato all'onnipresenza dei programmi, poiché è possibile creare un nuovo software per aggiungere nuove funzioni all'hardware esistente. Pertanto, non è insolito che gli utenti di telefoni cellulari ospitino decine di programmi, rendendo la funzione di telefono solo una delle molte funzionalità del dispositivo.

Spesso questi strumenti sono dati per scontati e sono così onnipresenti che si fondono con lo sfondo. A volte diventano visibili solo quando qualcosa va storto. Inoltre, gli strumenti non sono neutri, ma incorporano valori e costringono le persone ad agire secondo visioni ed aspettative predefinite.

Impatto dei programmi in quanto strumento

Poiché i programmi sono anche strumenti, quando se ne propone l’uso agli studenti, è importante considerare il loro impatto su chi li usa (i cosiddetti “utenti”) e il modo in cui l'utente viene in essi modellato. Le aziende informatiche non hanno solo creato strumenti per un'ampia gamma di usi (dalle applicazioni militari alla regolazione delle nostre vite sociali) ma hanno anche sviluppato interpretazioni implicite su come uno strumento potrebbe essere utilizzato, così questi programmi ridefiniscono anche il dominio di utilizzo, risultando in nuove problematiche e sfide. Ad esempio, i social media sono nati come un modo per accedere a notizie su persone ed eventi, ma i modi particolari con cui i social media ci forniscono queste informazioni implicano la ricezione di notizie mirate e scelte da un algoritmo basato sui dati che ha raccolto sull'utente. Questo rende le notizie rilevanti per l'utente e ne aumenta l'impatto, poiché mirate in modo specifico all'utente ma, di contro, può anche avere l'effetto indesiderato di amplificare informazioni fuorvianti. Inoltre, mentre i programmi nascono con un preciso scopo, nelle mani degli utenti possono evolvere in uno strumento a sé con potenziali usi al di là di quello che i progettisti avevano previsto.

Esempi di programmi in quanto strumento

- Trasporti, come l'automazione del funzionamento di un'auto, l'assistenza alla navigazione, l'organizzazione di viaggi.
- Supportare le persone nella comunicazione, inclusi i social media, le videoconferenze, le e-mail, la messaggistica mobile.
- Automatizzazione della spesa al supermercato (che include l‘uso dei lettori dei codici a barre alle casse, o le casse self-service), lo shopping online e altri modi per dare al cliente il controllo sul processo.
- Scopi ricreativi, inclusi giochi, intrattenimento, creazione e visione di film e creazione e ascolto di musica.

- Gestione dell’ ambiente, inclusi l'accesso e l'uso dell'energia, il riscaldamento, l'illuminazione e i sistemi di sicurezza.
- App create per scopi di nicchia, come i radar per la pioggia, o conoscere le restrizioni di parcheggio nelle strade locali.

Un programma è un'*Opera Umana*

Le persone creano programmi per soddisfare i bisogni umani, inclusi quelli indotti o il bisogno di esprimere sé stessi. Si potrebbe considerare i fenomeni della natura stessa come sistemi che elaborano informazioni (pensiamo alla trascrizione del DNA o alle piante che usano sostanze chimiche per trasmettere informazioni). Ma, a differenza di questi sistemi, i programmi informatici sono costruiti intenzionalmente dagli esseri umani per sfruttare i dispositivi di elaborazione delle informazioni.

I programmi sono creati come strumenti con uno scopo non solo per l'utente (ad esempio fornire un buon elaboratore di testo) ma anche per il creatore (per guadagnarsi da vivere o acquisire informazioni). Comunemente un programma è realizzato da team di persone, e i sistemi sviluppati possono consistere di molti programmi in interazione, quindi per essere bravi a creare questa tecnologia, le abilità sociali entrano in gioco molto rapidamente (come il lavoro di squadra, la comunicazione, la collaborazione, la creatività, la capacità di problem-solving e il pensiero critico). Uno sviluppatore deve essere in grado di discutere le opzioni, argomentare a favore di diverse soluzioni per un problema e lavorare con altri per creare una soluzione adeguata. Molti passaggi di questo processo utilizzano essi stessi strumenti software per rendere la programmazione più efficiente, ma il motore principale è una (o più) persona con un'idea.

Poiché i programmi sono fatti dall'uomo, non sono privi di valori. I valori, le idee e i modi di pensare degli sviluppatori e delle aziende che creano programmi si riflettono, esplicitamente o implicitamente, in un programma e nei modi in cui può essere utilizzato. Inoltre, poiché i programmi sono realizzati da esseri umani, e gli esseri umani non sono infallibili, è necessario tenere presente che anche i programmi possono contenere errori.

Impatto dei programmi in quanto opera umana

I programmi sono creati dall'uomo, il che significa che incorporano le ideologie, le intenzioni e i modi di pensare degli sviluppatori e delle organizzazioni che li rendono disponibili. È importante che i bambini (e non solo) siano consapevoli di questa sfaccettatura. Ad esempio, è importante rendersi conto che quando si utilizza un'app gratuita sul proprio telefono, il gruppo di persone che ha creato quel programma potrebbe guadagnare in un altro modo (raccogliendo i dati dell'utente e vendendoli ad altre aziende, o utilizzando quei dati per alimentare i loro programmi di apprendimento automatico). D'altra parte, il "*software libero*" (usando "libero" nel senso di "libertà di parola") mette tutti sullo stesso piano, poiché è un modo efficiente di condividere idee ed evitare che il potere sia detenuto da organizzazioni particolari.

Inoltre, il fatto stesso che per creare programmi si abbia bisogno di persone esperte sul tema (progettazione, codifica, mantenimento, test e così via) ha tormentato il campo della programmazione fin dagli anni '50: la carenza di programmatori qualificati ha limitato il ritmo di produzione del software e fatto aumentare i costi legati all'assunzione di programmatori, (rendendo, d'altro canto, più attrattive le carriere in questo ambito). Questo è stato anche un fattore trainante per sviluppare strumenti di programmazione che automatizzano la creazione di programmi, che vanno da ambienti di sviluppo migliori all'idea di sistemi low-code o no-code che supportano lo sviluppo rapido (ma che sono a loro volta strumenti che i programmatori devono sviluppare!).

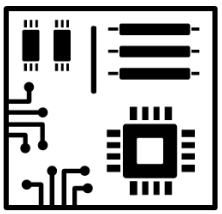
Esempi di programmi in quanto opera umana

- Migliaia di nuove "app" vengono rilasciate per i telefoni cellulari ogni mese. Per giustificare il lavoro necessario per crearle, devono riflettere un'innovazione dove qualcuno ha identificato un pubblico che ne beneficerebbe.
- L'Iniziativa di Difesa Strategica (Star Wars) che fu lanciata negli Stati Uniti da Ronald Reagan negli anni '80 ha portato a uno scandalo pubblico quando David Parnas, un noto ingegnere del software, si dimise dal progetto perché i requisiti sul software erano irrealistici. Questo era dovuto, tra le altre cose, al fatto che sarebbe stato impossibile essere sicuri che non contenesse errori e, in questo tipo di contesto, gli errori del software possono portare a catastrofi.
- Ci sono state alcune conseguenze molto costose a seguito di un errore umano nella produzione di software, tra cui il fallito primo volo dell'Ariane 501, il millennium bug e la costosa gestione errata dei bagagli da parte del nuovo sistema software al terminal 5 di Heathrow.
- La visione del mondo e l'esperienza limitata dei programmatori dietro a un particolare programma potrebbero portare a condizionamenti insiti nel prodotto finale, ad esempio non considerando che l'utente possa avere difficoltà visive, o assumendo che il particolare tipo di interfaccia usata sia di immediata comprensione.
- L'ambiente di programmazione Scratch è un esempio di come i sistemi software incorporano una certa idea o concezione di "bisogno". Scratch è oggi uno degli ambienti principali in cui i bambini imparano a scrivere "codice", ma data la sua particolare visione sull'apprendimento dell'informatica, offre ai bambini anche un'esperienza molto particolare di "programmazione" che potrebbe trarre in inganno.

Dicotomia del programma come strumento vs opera dell'uomo

Una gran parte dell'appello affinché gli studenti imparino la programmazione è per consentire loro di essere "creatori" piuttosto che solo "utenti" delle tecnologie digitali. I programmi per computer sono strumenti preziosi e educare gli studenti sull'uso di questi strumenti (come motori di ricerca, fogli di calcolo ed editor video) è importante, poiché questi consentono loro di essere produttivi e creativi. Ma se sono limitati a essere solo utenti distrumenti senza apprezzare il fatto che siano creati da persone che sono abili nella programmazione, perdono l'opportunità di vedere la programmazione come un percorso di carriera per loro stessi, o almeno di realizzare da dove hanno origine i programmi e cosa comporta crearli. Questo ha portato all'insegnamento di programmi basati su un approccio "usa-modifica-crea", dove gli studenti esplorano programmi esistenti, ma crescono fino a essere in grado di creare i propri.

Un programma è un *Oggetto Fisico*



Anche se il software è un concetto astratto, i programmi sono memorizzati su un supporto fisico e possono essere eseguiti, letti, copiati, modificati o anche soggetti a corruzione. Inoltre, la loro esecuzione richiede tempo e utilizza energia anche se, a differenza della maggior parte degli altri oggetti fisici familiari, il tempo, l'energia e lo spazio consumati sono solitamente su una scala estremamente piccola.

La sfaccettatura “fisica” potrebbe sembrare secondaria ma, come un romanzo non esiste senza un supporto fisico che trasporti il suo messaggio, un programma non “vive” solo in una nuvola immaginaria - infatti, anche se viene eseguito su una macchina virtuale nel “cloud", sta utilizzando energia e spazio (fisico e di memoria) in una determinata posizione.

È possibile che i programmi possano anche essere eseguiti dagli umani (ad esempio, tracciando l'esecuzione di un programma su un pezzo di carta) e possono anche essere memorizzati in un supporto fisico che non è una macchina di per sé (ad esempio, diagrammi di flusso degli anni '50; o istruzioni su carta). Tuttavia, di solito, quando parliamo di programmi ci aspettiamo che siano eseguiti su una macchina, che può eseguire milioni di istruzioni in modo affidabile in un brevissimo lasso di tempo.

Impatto dei programmi in quanto oggetto fisico

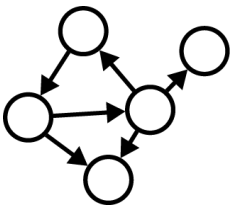
I programmi che vengono eseguiti su un dispositivo informatico consumano spazio, energia e tempo, e la costruzione del dispositivo utilizza risorse come lavoro e metalli preziosi. Queste osservazionipossono di per sé sollevare preoccupazioni. È importante che gli studenti si rendano conto che per eseguire un programma, c'è un impatto nel "mondo reale", che include anche implicazioni sociali e psicologiche.

Oltre alle questioni legate all'uso di dispositivi personali per eseguire un programma, è importante anche che gli studenti capiscano che qualcosa "in cloud" o "sul web" ha sempre una posizione fisica alla quale si applicano determinate leggi e che potrebbero essere molto diverse dalle leggi che si presume si applichino. Avere un servizio cloud basato in Europa, per esempio, offre una protezione dei dati molto migliore rispetto ad averlo localizzato negli Stati Uniti, grazie al GDPR dell'UE. Inoltre, se da un lato l'osservazione che mentre sono in funzione i programmi consumano energia e risorse potrebbe essere stata meno preoccupante 20 anni fa, al giorno d’oggi l'impatto sull'ambiente sta diventando sempre più una preoccupazione crescente.

Esempi di programmi in quanto oggetto fisico

- Le app sono comunemente memorizzate su memoria flash nei dispositivi mobili e un software è tipicamente memorizzato su dischi rigidi nei computer; in entrambi i casi, si tratta di programmi, ma il nome usato per riferirsi ad essi è differente a seconda del contesto in cui vengono utilizzati.
- I programmi vengono spostati nella memoria principale (e in altri tipi di memoria come le *cache*) per consentirne l'esecuzione.
- Ci vuole tempo per scaricare i dati digitali necessari per installare o aggiornare il software.
- I programmi utilizzano energia per funzionare e l'importanza del processo di raffreddamento per la maggior parte dei computer evidenzia l'impatto ambientale che potrebbero avere.
- I dispositivi digitali sono costruiti utilizzando risorse scarse, e la costruzione di dispositivi per eseguire i nostri programmi necessità di considerazioni di sostenibilità (ambientale e non).
- I programmi percepiscono e influenzano l'ambiente fisico attraverso input e output, che potrebbero essere minimi come posizionare un'immagine su uno schermo, o sostanziali come controllare l'acqua che scorre attraverso una diga.
- Le criptovalute come il bitcoin sono note per avere un enorme impatto ambientale a causa dell'intenso dispendio energetico delle modalità con cui le transazioni devono essere verificate, così come l'uso di energia per il "mining" della valuta. L'impatto è così alto che enti di beneficenza come Greenpeace e WWF hanno abbandonato l'uso delle criptovalute.

Un programma è un'*Entità Astratta*



L'astrazione è fondamentale per i programmi e la programmazione. Allo stesso modo in cui una storia, una barzelletta, un'idea o un concetto sono astratti, un programma è qualcosa che si riferisce (e manipola) a nozioni ed entità astratte. I dati che i programmi manipolano sono solo cifre, che sono un concetto astratto anche se sono utilizzati per rappresentare qualcosa di fisico nel mondo reale, che sia la quantità di denaro che qualcuno ha in banca, il colore di un pixel, la registrazione di musica, o anche qualcosa che di per sé è astratto, come una storia. Inoltre, i modi particolari in cui i numeri sono rappresentati nella macchina sono sempre un'astrazione del meccanismo fisico e continuo sottostante. Il programma determina infine una serie di azioni che il sistema esegue, e di conseguenza il programma determina effetti fisici nel sistema stesso (come la memoria del dispositivo). Tuttavia, ciò che è rilevante non sono questi effetti fisici in sé, ma il fatto che possono essere interpretati secondo le astrazioni che il programma manipola, come un numero, una parte di un'immagine o un suono.

I programmi implementano algoritmi, e un algoritmo non è fisico - è un'idea su come fare qualcosa. Un algoritmo cattura una visione astratta di una classe di programmi e, come altri concetti astratti, è una costruzione della mente di qualcuno (e non necessariamente la stessa costruzione per tutte le menti). L'unico modo per descrivere un algoritmo ad altri, tuttavia, è redigere un programma più o meno formale/dettagliato in qualche linguaggio (scritto o orale), che è strettamente connesso alla sfaccettatura dell'"entità linguistico-notazionale" descritta in seguito. In questo senso, possiamo vedere un programma come una particolare incarnazione di un'idea astratta.

Impatto dei programmi in quanto entità astratta

Poiché l'astrazione è una parte importante della programmazione, gli studenti devono imparare a decidere come modellare i dettagli su cui desiderano scrivere programmi. Questo significa, inevitabilmente, che ci sono molti modi per ottenere lo stesso risultato generale; per esempio, una data come l'11 luglio 2022 potrebbe essere astratta in tre numeri (11, 7, 2022) o il numero di giorni dal 1° gennaio 1900 (44.753), o in qualsiasi altro dei diversi formati.

L'astrazione che circonda la programmazione significa anche che un algoritmo non ha esattamente un programma a cui corrisponde; infatti, è improbabile che più programmatori che implementano lo stesso algoritmo finiscano con il produrre programmi identici. Più in generale, non esiste un unico programma "corretto" che raggiunge un particolare risultato, mentre gli studenti possono essere abituati all'idea che normalmente nella risoluzione dei problemi esista solo una risposta corretta. In realtà, esiste un numero infinito di programmi che raggiungono un particolare risultato (ne esiste anche un numero infinito che sono incorretti!).

Di conseguenza, se uno sviluppatore decide di rappresentare un dato oggetto nel mondo in un modo particolare, allora chi studia il software dovrebbe essere consapevole che tale rappresentazione non è assoluta ma relativa agli obiettivi e alle visioni del mondo dello sviluppatore, del team di sviluppatori o del cliente. Questo spiega in parte i problemi legati ai condizionamenti nei programmi informatici. In generale, poiché non esiste una regola definita per decidere sulle “migliori astrazioni”, si potrebbe anche finire con “cattive astrazioni” che risultano in programmi malfunzionanti o complessi.

Esempi di programmi in quanto entità astratta

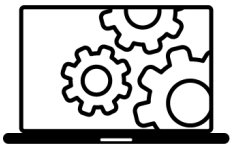
- I linguaggi di programmazione semplici per principianti a volte hanno set di istruzioni come "Avanti", "Destra" e "Sinistra" per programmare un folletto o un robot per muoversi. Queste istruzioni possono astrarsi dal fatto che i movimenti fisici non sono mai perfettamente orientati; anche la direzione di un robot è soggetta ad alcuni errori (forse impercettibili). Tuttavia, queste astrazioni sono utili per approssimare sufficientemente bene la descrizione dei movimenti da eseguire.
- Quando si pianifica come implementare un programma, degli approcci intermedi (come descrizioni informali, diagrammi di flusso o pseudo-codice) sono utili per aiutare il programmatore a lavorare a un livello di astrazione appropriato e non essere sopraffatto dal dettaglio dell'implementazione.
- Quando nominiamo le variabili in un ambiente di programmazione, esse suggeriscono spesso una visione dell'oggetto del programma come metafora di qualcosa del mondo reale. Potremmo ad esempio definire una classe di oggetti nota come "persona" e dare a questi un certo numero di proprietà (ad esempio "nome" e "codice fiscale"). Questa classe di oggetti astrae ciò che il programmatore ha bisogno di sapere su una persona per il loro contesto, e una "persona reale" è molto più di un nome e un numero.
- Il modo in cui i numeri sono rappresentati in un computer è un concetto di base nell'analisi numerica. Poiché i "numeri reali" in un ambiente di programmazione possono essere solo approssimazioni finite di "numeri reali" effettivi, i programmatori che lavorano in un contesto di modellizzazione devono essere molto attenti a potenziali problemi come gli errori di arrotondamento.

Dicotomia del programma come oggetto fisico vs entità astratta

Potrebbe sembrare strano che abbiamo affermato che un programma sia allo stesso tempo sia astratto che fisico. Sebbene apparentemente antitetici tra loro, quando pensiamo ai programmi i tratti concreti e astratti solitamente coesistono. Da un lato, di solito diamo senso al comportamento di un programma immaginando un mondo di astrazioni che in qualche modo prendono vita nella nostra mente, avendo quindi effettivamente a che fare con un'entità astratta. D'altra parte, è anche un oggetto fisico concreto non appena lo codifichiamo, manipoliamo ed eseguiamo mediante un dispositivo di calcolo. A causa di questo tipo di ambivalenza, l'informatica è stata addirittura definita come "la disciplina delle astrazioni concrete" (“the discipline of concrete abstractions”[1]).

[1] Max Hailperin, Barbara Kaiser, and Karl Knight. 1999. *Concrete Abstractions*. Brooks/Cole, Pacific Grove, CA

Un programma è *eseguibile automaticamente*



Un programma può essere eseguito su un dispositivo fisico, in momenti diversi da utenti diversi. Un programma può essere caratterizzato come qualcosa che viene automaticamente eseguito da un agente di elaborazione delle informazioni, dove l'agente segue senza domande le istruzioni senza alcuna comprensione dello scopo dell'insieme di istruzioni che gli è stato dato. Qualsiasi decisione presa da un programma è già stata programmata in esso dal programmatore. Una volta che un utente scarica e inizia a utilizzare un programma, il programmatore non ha più un controllo; ogni risposta all'interazione dell'utente e ai dati in input è determinata da ciò che è già nel programma.

L'automazione si basa su un insieme finito e molto preciso di istruzioni che un dispositivo digitale è in grado di eseguire. Al livello più basso, il programma viene eseguito utilizzando istruzioni macchina per il particolare processore in uso, sebbene di solito queste istruzioni siano generate da strumenti che consentono ai programmatori di lavorare in un linguaggio di alto livello che ha a sua volta un set di istruzioni ben definito.

Impatto dei programmi in quanto eseguibili automaticamente

Poiché l'esecuzione di un programma è automatica, in situazioni commerciali tipiche i programmatori rilasciano il programma agli utenti, momento in cui il programmatore non ha più il controllo su ciò che accadrà e, in molti casi, potrebbe anche nemmeno sapere quando il programma viene utilizzato. Questo sottolinea l'importanza dei test: prima che il software venga rilasciato, deve essere testato per garantire che funzioni bene per la grande varietà di modi in cui può essere utilizzato. Gli studenti possono perdere di vista questo aspetto perché, essendo principianti, sono spesso sia il programmatore, sia il tester, sia l'utente dei propri programmi, il che purtroppo significa che come utenti comprendono già bene il programma e potrebbero non vederlo dal punto di vista di un utente tipico. Avere qualcuno diverso dal programmatore che testa un programma (compresa la sua interfaccia utente) può essere prezioso per i principianti per dare loro una visione di come sarà visto dagli altri e programmarlo in modo che una versione automatizzata sia utilizzabile e utile.

Il fatto che un programma eseguito su una macchina informatica debba essere automatico pur essendo in grado di gestire il comportamento in gran parte imprevedibile delle interazioni degli utenti o altre interazioni con l'ambiente esterno, porta al problema di sviluppare programmi che controllano comportamenti futuri che non possono essere dettagliati completamente. È qui, quindi, che diventa critico il problema di trovare le opportune rappresentazioni del mondo all'interno del programma affinché il programma funzioni in modo soddisfacente.

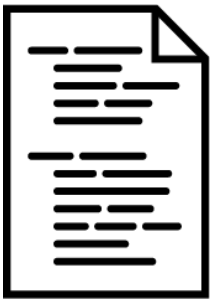
Nella pratica, questo è particolarmente importante se il software controlla processi delicati come aerei, navicelle spaziali, sistemi finanziari e centrali nucleari, dove le conseguenze di non anticipare cosa accadrà nell'automazione potrebbero essere catastrofiche.

Al contrario, ciò significa anche che un programmatore può scrivere un programma che può essere utilizzato da milioni di persone e per lunghi periodi di tempo, il che crea un modello economico abbastanza diverso dalla produzione e distribuzione convenzionale.

Esempi di programmi in quanto eseguibili automaticamente

- Le persone possono usare un programma senza capire come funziona: il programmatore ha creato uno strumento sotto forma di programma che l'utente può adoperare se possiede un dispositivo che eseguirà il programma per loro, ma l'utente può solo trattare il programma come una scatola nera. Infatti, molti software commerciali rendono intenzionalmente il programma stesso opaco per l'utente, poiché il codice è un segreto commerciale.
- Il computer che esegue il programma è inconsapevole dello scopo del programma; segue istruzioni che provengono da un insieme di istruzioni ben definito facendo esattamente ciò che ogni istruzione richiede.
- Un'interazione con il programma che non era stata prevista dal programmatore, può risultare in programmi malfunzionanti. Questo è di particolare importanza nei sistemi critici per la sicurezza come le auto a guida autonoma.
- L’esecuzione di un programma è, a volte, poco esplicita: ad esempio, quando facciamo una telefonata, il nostro smartphone sta eseguendo un programma.
- Esempi più vecchi di macchine automatiche (programmate) sono il telaio di Jacquard (le cui versioni digitali sono ancora in uso) e le scatole musicali (ancora in uso, seppur in misura limitata).

Un programma è un'*Entità Linguistico-Notazionale*



Allo stesso modo di un manoscritto, un libro o uno spartito musicale, un programma è un'entità linguistico-notazionale: si basa su una notazione con una particolare sintassi, secondo alcune regole formali, notazioni linguistiche e convenzioni. Proprio come la scrittura di un libro da parte degli umani solitamente coinvolge una certa creatività e conoscenza tacite, lo stesso vale per i programmi realizzati dagli umani. I programmi possono esistere di per sé come oggetto di interesse, ma sono principalmente utilizzati per comunicare (dai programmatori alle macchine, o tra programmatori) dei processi da svolgere per raggiungere un certo obiettivo.

Quando sono destinati a essere comunicati alle macchine, i programmi si affidano a linguaggi formali che possono essere automaticamente analizzati ed eseguiti. Tuttavia, poiché sono anche destinati ai programmatori umani, tipicamente includono molti elementi linguistici non formali, come nomi di variabili significativi, commenti chiari e precisi, traduzioni dell'interfaccia e documentazione di supporto. Tutti questi elementi lavorano insieme per rendere il programma utile ed efficiente per le persone che ci lavorano.

Un'altra caratteristica delle notazioni di programmazione intese per la macchina è che possono sempre essere convertite in un'altra notazione utilizzando un "traduttore" automatico (ad esempio, un compilatore o un interprete). L'idea che si possa sempre convertire una notazione in un'altra è stata un'intuizione fondamentale nella progettazione dei linguaggi di programmazione dagli anni '50. Infatti, è proprio grazie a questa possibilità che possiamo astrarci dall'hardware del computer per presentare programmi in una notazione che è anche comprensibile e utilizzabile dagli umani.

Impatto dei programmi in quanto entità linguistico-notazionale

Gli script dei programmi sono scritti per le persone. Il programma stesso è solitamente destinato a un utente finale, che potrebbe aver ricevuto una versione dello script che è stata convertita in un'app eseguibile e che non ha interesse nella notazione che rappresenta. Ma i programmi sono scritti anche per il prossimo programmatore che potrebbe doverli modificare o eseguirne il debug. Il "prossimo programmatore" può essere l'autore originale che ha bisogno di ricordare come il programma è stato progettato, dato che i programmi sono spesso di complessità tale che è difficile comprenderli completamente. Quindi, scrivere buoni programmi richiede abilità di comunicazione scritta che rendono il programma facile da capire per un essere umano. Anche scegliere un buon nome per una funzione o una variabile è una sfida; se il nome di una variabile è troppo corto, il suo significato è poco chiaro, e se è troppo lungo porta a codice difficile da leggere; scegliere una dicitura breve, significativa e non ambigua per i nomi delle variabili e i commenti è un'abilità importante.

I programmi esprimono idee che ci permettono di articolare i nostri pensieri e condividerli con altri. Un autore è arrivato a dire che "un linguaggio informatico non è solo un modo per far eseguire operazioni a un computer, ma piuttosto è un nuovo mezzo formale per esprimere idee sulla metodologia. Quindi i programmi devono essere scritti per essere letti dalle persone e solo incidentalmente per essere eseguiti dalle macchine" ("a computer language is not just a way of getting a computer to perform operations but rather it is a novel formal medium for expressing ideas about methodology. Thus programs must be written for people to read, and only incidentally for machines to execute"[2]).

[2] Gerald Jay Sussman. 2004. *The Legacy of Computer Science*. In Computer Science: Reflections on the Field, Reflections from the Field, Committee on the

Esempi di programmi in quanto entità linguistico-notazionale

- È importante utilizzare efficacemente la notazione di programmazione. La struttura di un programma può influenzare la sua leggibilità: il codice mal strutturato o con numerosi rimandi interni (incluso il cosiddetto “spaghetti code") è difficile da seguire e riorganizzare il codice può servire a preservare la comprensibilità dei programmi.
- Scrivere un commento conciso che spiega come funziona un programma o scegliere un nome di variabile che spieghi precisamente il suo scopo è un'**abilità di alfabetizzazione** chiave per una programmazione di successo.
- La notazione usata per i programmi si conforma a linguaggi formali ben definiti ed evita l'ambiguità dei linguaggi naturali.
- Ci sono premi per gli autori di programmi deliberatamente offuscati. Sono una sorta di forma d'arte, ma non sono utili per ottenere risultati nella programmazione convenzionale e generalmente evidenziano l'ostacolo che un programma scritto male dà a coloro che cercano di comprenderlo.

Dicotomia del programma come entità eseguibile automaticamente vs entità linguistico-notazionale

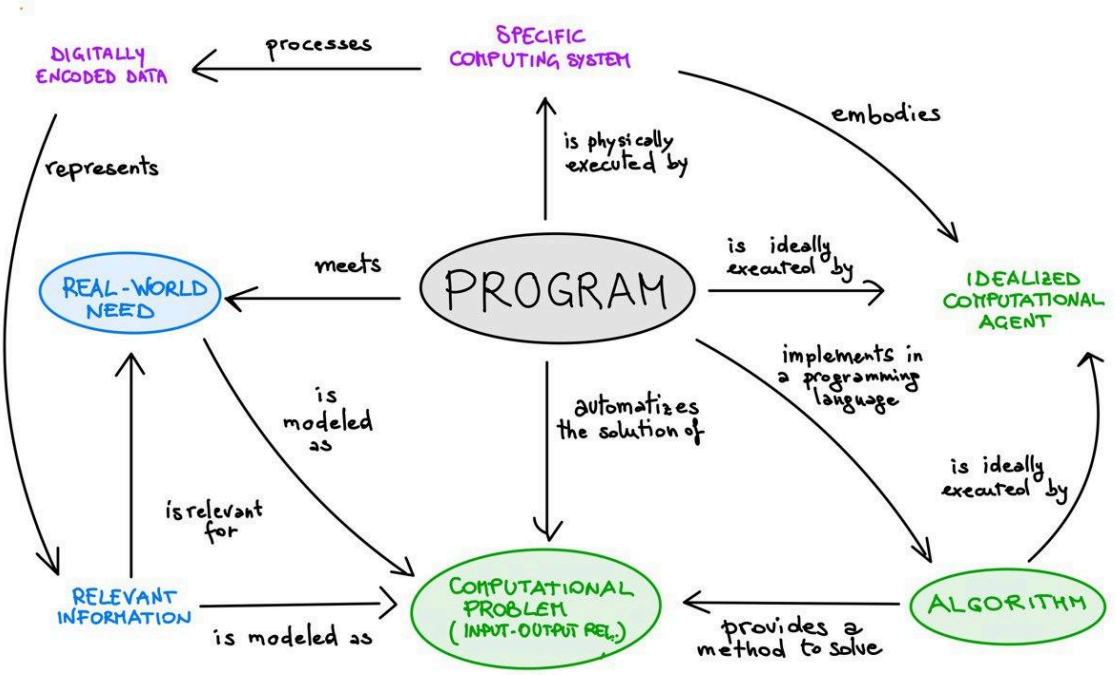
L'idea della programmazione è nata al fine di automatizzare i compiti e, quindi, è inevitabilmente importante che un programma possa essere eseguito. Tuttavia, ora disponiamo di una moltitudine di linguaggi di programmazione che offrono una varietà di notazioni per la creazione di programmi, e la notazione è diventata un importante aspetto di studio a sé stante. Poiché un programma viene eseguito automaticamente, non c'è bisogno che l'utente finale comprenda la notazione. Al contrario, la notazione può essere utilizzata semplicemente come un modo per esprimere idee, e elementi di essa (come i nomi delle variabili e i commenti) possono essere più importanti per comunicare le idee ad altri programmatori che per influenzare l'esecuzione del programma. In casi estremi, un programma può essere scritto principalmente come un'opera d'arte (come nella "programmazione letteraria" o nel "live coding") o persino per umorismo (come nei concorsi di codice offuscato).

Fundamentals of Computer Science: Challenges, Computer Science Opportunities, and National Research Council Telecommunications Board (Eds.). The National Academies Press, 180–183. This can also be found in the co-authored book: Harold Abelson, Gerald J. Sussman, and Julie Sussman. 1984. *Structure and Interpretation of Computer Programs*. The MIT Press, Cambridge, MA.

Parte II: come vengono creati i programmi

In questa sezione verrà utilizzata la seguente mappa concettuale per esplorare come nascono i programmi e la loro relazione con gli algoritmi, i sistemi informatici e il mondo reale.

Mappa dei concetti



I **programmi** sono scritti per descrivere *l'automazione delle soluzioni* dei **problemi computazionali**. I problemi computazionali sono il corrispettivo delle necessità o problematiche del mondo reale nell'elaborazione delle informazioni, e modellano i bisogni del mondo reale r*appresentando* le **informazioni rilevanti** con dati adatti.

Un programma può essere eseguito automaticamente, ovvero *fisicamente eseguito* da un **sistema di calcolo specifico** (ad esempio, un computer o un telefono cellulare). Tali sistemi normalmente elaborano solo codifiche digitali (ovvero simboliche) dei dati, quindi i dati devono essere **codificati digitalmente** - ovvero rappresentati come sequenze di simboli.

Un **algoritmo** *fornisce un metodo* (cioè, una procedura precisa che può essere eseguita in modo prescrittivo) per risolvere un problema computazionale. L'algoritmo è progettato per essere *eseguito* da un **agente computazionale idealizzato**, capace di eseguire un insieme predefinito di azioni. Un sistema di calcolo effettivo istanzia tale agente computazionale idealizzato.

Per essere automaticamente eseguito da un sistema di calcolo effettivo, un algoritmo deve essere *implementato*, ovvero scritto in un *linguaggio di programmazione* secondo regole rigorose e conformemente ai vincoli specifici del sistema di calcolo, risultando così in un programma.

Esempio: la cassa del supermercato

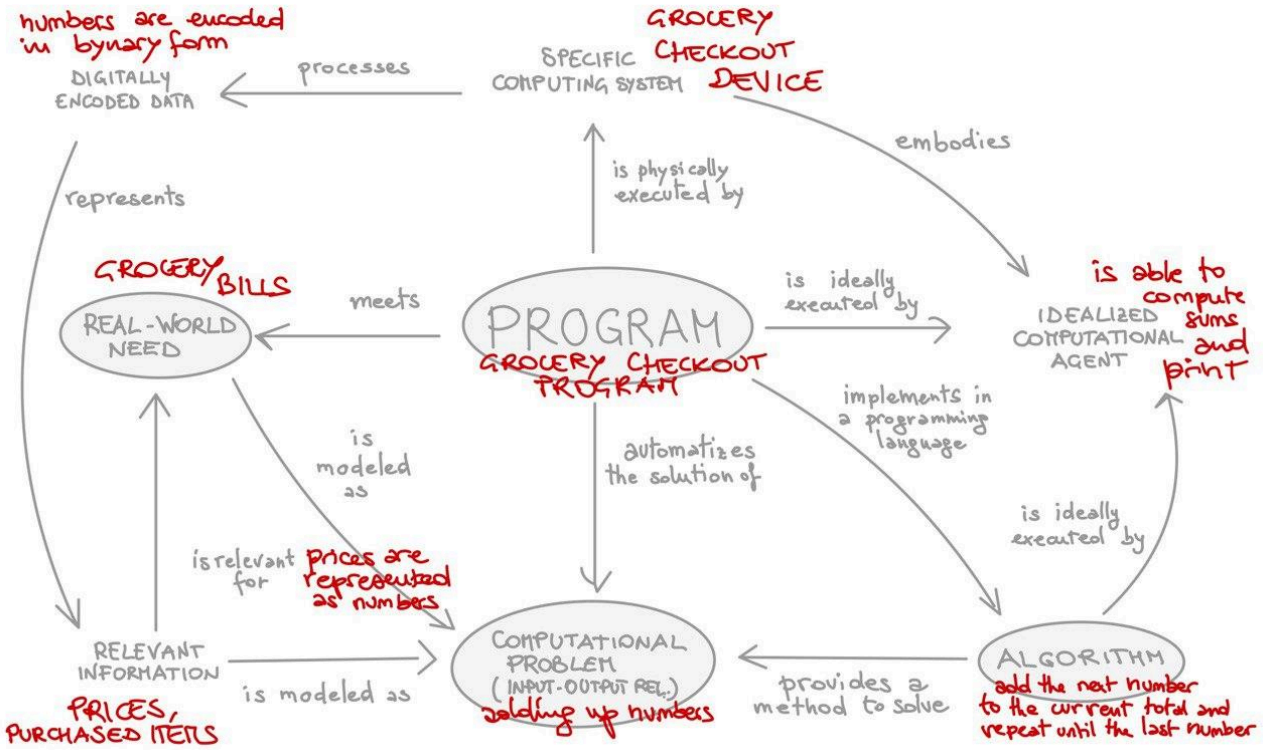
Questo esempio viene utilizzato per illustrare come il diagramma sopra si applicherebbe a una situazione specifica (di seguito ne è fornita una versione annotata sulla base dell'esempio).

Un programma per la cassa del supermercato è destinato a supportare i negozianti che hanno bisogno di calcolare i conti della spesa dei (e per i) loro clienti. I prezzi degli articoli acquistati possono essere modellati come numeri. Pertanto, calcolare il totale della spesa può essere descritto come il problema computazionale generale di "sommare una serie di numeri".

Il programma per la cassa del supermercato viene eseguito da un dispositivo di cassa, che può leggere i numeri scansionando i codici a barre e stampare il risultato su scontrini di carta. Il dispositivo manipola effettivamente numeri codificati in forma binaria.

L'algoritmo tipico per sommare una serie di numeri è il seguente: 1) utilizzare una variabile per tenere traccia del totale corrente e impostarla inizialmente a zero; 2) fino a quando non hai finito i numeri ripeti la seguente istruzione: aggiungi (somma) il numero successivo alla variabile che tiene traccia del totale corrente; 3) alla fine stampa la variabile che tiene traccia del totale corrente. Quando progettiamo questo algoritmo, assumiamo che il suo esecutore possa idealmente inserire qualsiasi numero, sommare qualsiasi coppia di numeri e stampare qualsiasi numero. Il dispositivo di cassa del supermercato esegue tali istruzioni fisicamente. Questo implica alcuni vincoli, per esempio, che il dispositivo può elaborare solo un intervallo limitato di numeri.

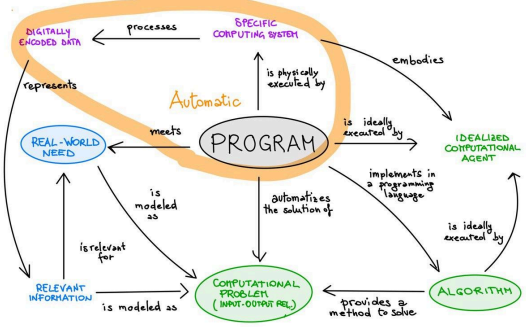
L'algoritmo è probabilmente implementato in un linguaggio di programmazione specializzato per dispositivi di cassa. Tuttavia, sia il problema computazionale sia l'algoritmo possono adattarsi ad altre esigenze reali simili, come calcolare la popolazione totale di un paese basandosi sulla conoscenza della popolazione di ciascuna delle sue regioni. Pertanto, se implementato in un linguaggio di programmazione generale, il programma potrebbe essere utilizzato per diversi contesti.



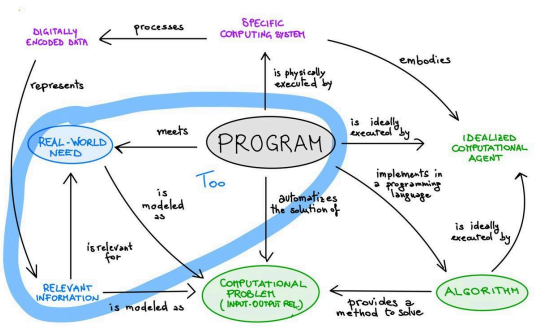
Connessione tra mappa dei concetti e sfaccettature di un programma

Il seguente diagramma mostra gli aspetti più rilevanti rispetto alle sfaccettature descritte nella parte I di questo documento. L'ultimo di questi viene descritto più dettagliatamente di seguito.

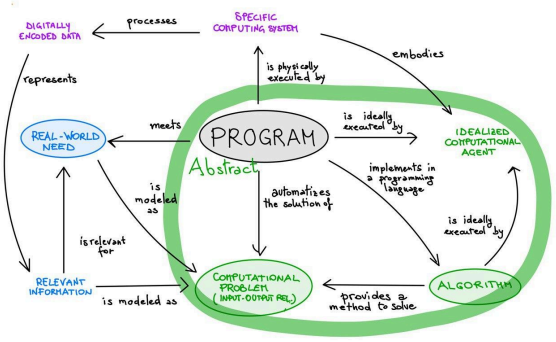
Automatic / Physical



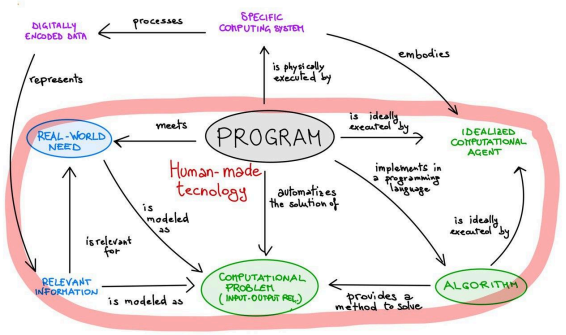
Tool



Abstract / Notational

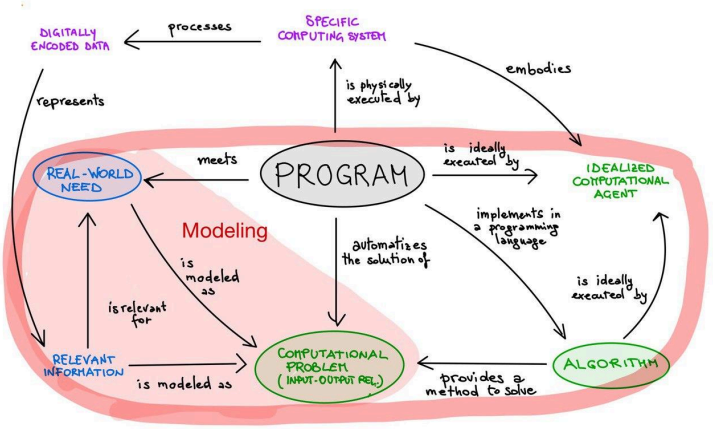


Human-made technology



Programmi e programmazione

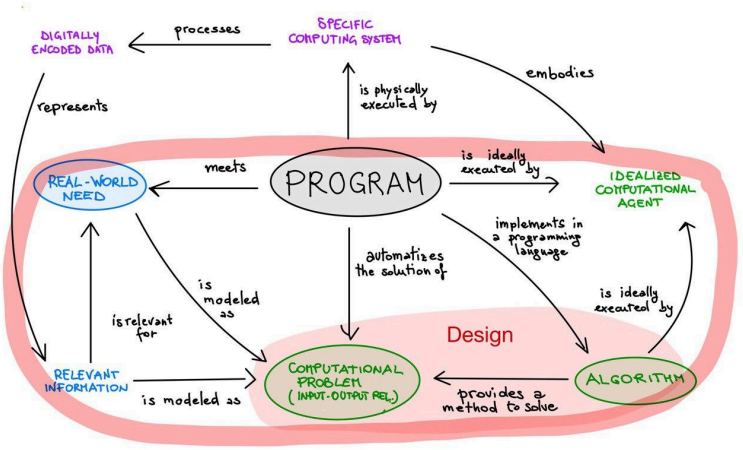
I programmi sono una tecnologia creata dall'uomo. *Programmare* è il processo di creazione e sviluppo di programmi. Non esiste un unico modo per sviluppare programmi; invece, vengono utilizzate pratiche diverse quando si programma, in accordo con il tipo di compito che il programma sta automatizzando. Tuttavia, nella realizzazione di qualsiasi programma si verificano alcuni processi fondamentali, che sono tipicamente eseguiti da esseri umani (solitamente con il supporto di altri programmi). Sebbene concettualmente valga la pena identificare separatamente questi processi, nella pratica la distinzione tra di essi è molto più sfumata; l'ordine lineare in cui li presentiamo non dovrebbe quindi suggerire che debbano essere eseguiti sequenzialmente: potrebbero essere necessarie anche reiterazioni o inversioni di marcia durante l'intero processo di programmazione.



Modellazione - I bisogni e i problemi del mondo reale vengono modellati come problemi computazionali. I problemi della vita reale si presentano con istanze specifiche, ma i problemi computazionali generalmente coprono un intero insieme di istanze specifiche. Modellare un problema del mondo reale significa comprendere le principali entità coinvolte e le loro relazioni, e decidere come esprimere le informazioni rilevanti per il problema utilizzando dati appropriati. I dati di input rappresentano informazioni note, e i dati di output, una volta interpretati nel contesto appropriato, forniscono la risposta al problema. È comune distinguere tra dati e informazioni, dove i dati sono solo modelli grezzi di segni, in grado di fornire informazioni solo quando interpretati dagli esseri umani nel contesto del problema da risolvere.

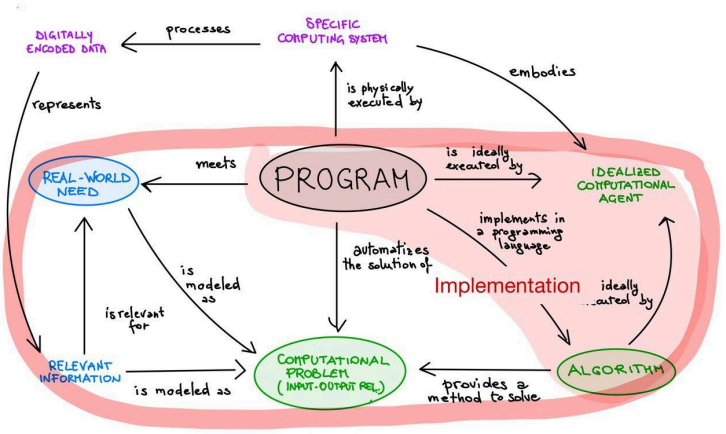
Un problema computazionale è specificato definendo la relazione tra i dati di input e i dati di output (ovvero come i dati di output dipendono dai dati di input). Solitamente dati di input diversi producono dati di output diversi. Inoltre, i dati di input e output non sono necessariamente disponibili all'inizio dell'esecuzione o prodotti alla fine dell'esecuzione, ma possono anche far parte di un'interazione continua tra l'utente, specifiche unità sensoriali e il sistema di calcolo.

Progettazione - Un algoritmo per un problema computazionale è una descrizione di come ottenere automaticamente i dati di output associati - in quel problema computazionale - a qualsiasi dato di input fornito (gli informatici dicono che l'algoritmo per un problema computazionale è una *soluzione* per quel problema computazionale). Una difficoltà rilevante nella progettazione degli algoritmi è il grande numero di casi diversi che possono verificarsi durante la sua esecuzione e che devono essere considerati in anticipo. Diversi algoritmi possono essere progettati per risolvere lo

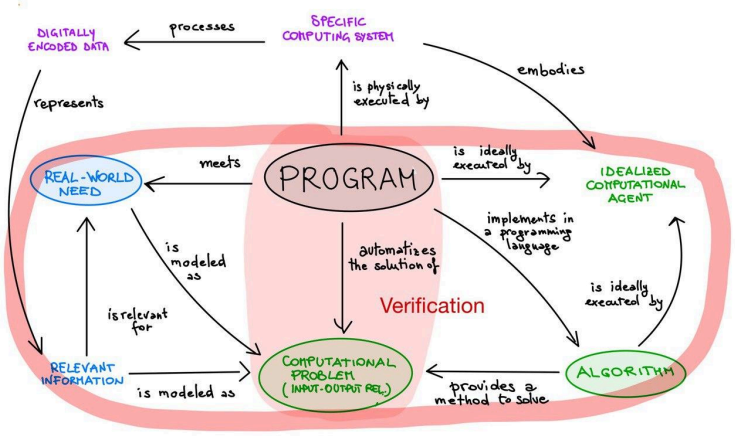


stesso problema computazionale. Varie tecniche di problem-solving possono essere utilizzate per ideare e progettare un algoritmo.

Implementazione - Gli algoritmi sono implementati come programmi, cioè sono scritti in un *linguaggio di programmazione* secondo regole rigorose e conformi ai vincoli del sistema di calcolo. Questa attività è talvolta chiamata *coding*. Il termine *programmazione* è anche spesso usato per riferirsi specificamente a questa attività di implementazione. Tuttavia, preferiamo usarlo in senso più ampio per includere l'intero processo, che comprende anche la modellazione, la progettazione, la verifica e la validazione.



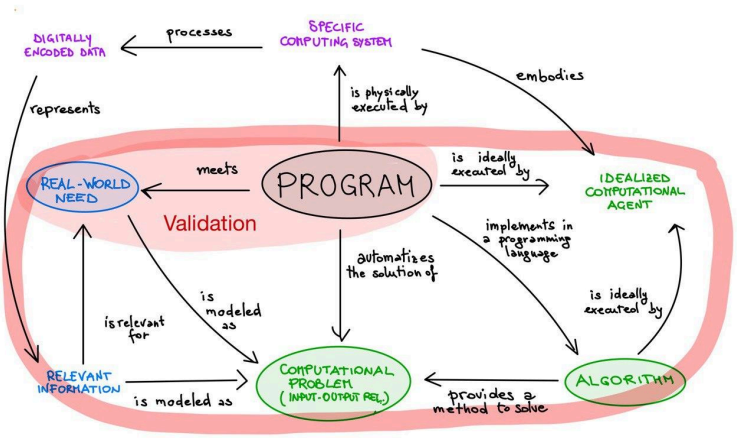
Perché un programma possa essere eseguito da un sistema di calcolo potrebbe essere necessario tradurlo dal linguaggio di programmazione in cui è scritto, in un linguaggio specifico compreso dal sistema di calcolo, e ciò viene fatto automaticamente da altri programmi specializzati (ad es., compilatori). Il termine "programma" può riferirsi sia al programma scritto nel linguaggio di programmazione sia alla sua versione eseguibile. Per distinguere tra queste due entità, il primo è anche chiamato *codice sorgente* del programma, e il secondo *programma eseguibile*. Il codice sorgente del programma è infatti non solo una descrizione indirizzata alle macchine, ma anche un mezzo di comunicazione tra i programmatori e tutti gli stakeholder interessati a comprendere i dettagli del calcolo (vedi la sfaccettatura notazionale).



Verifica - Un programma dovrebbe automatizzare la soluzione del problema computazionale per il quale è stato scritto. Tuttavia, i processi di progettazione e implementazione sono di per sé imprese complesse e non prive di errori; il programma risultante potrebbe contenere difetti (a volte chiamati *bug*). Il processo di verifica mira a testare se il programma risolve davvero il problema computazionale, cioè se è in grado di fornire l'output atteso per qualsiasi dato di input. Il processo di verifica si basa sulla possibilità di eseguire il programma su sistemi di calcolo reali, su una gamma di dati di input diversi. Pianificare un'ampia gamma di test è in generale un compito difficile, poiché i programmi di solito devono affrontare un enorme numero di casi diversi. Un fallimento durante la verifica del programma può portare a ulteriori revisioni del programma stesso o persino del suo progetto complessivo.

Convalida - Una verifica di successo di un programma non è sufficiente a garantirne l'adeguatezza dal punto di vista dell'utente, poiché il processo di verifica si riferisce solo al problema computazionale (e non al bisogno del mondo reale modellato dal problema computazionale).

In particolare, le decisioni riguardanti la modellizzazione delle informazioni rilevanti possono avere un impatto significativo sulla qualità dell'intero processo, e ciò potrebbe diventare evidente solo una volta che il programma è pronto e può essere eseguito e utilizzato. Questa aderenza al bisogno del mondo reale deve essere convalidata coinvolgendo gli utenti target del programma.



Esempio: la cassa del supermercato

Modellazione: i prezzi sono modellati come numeri; il bisogno del mondo reale è descritto computazionalmente come il problema di sommare una serie di numeri.

Progettazione: progettiamo un algoritmo che utilizza una variabile per tenere traccia del totale e lo aggiorna ripetutamente.

Implementazione: l'algoritmo è scritto in un linguaggio di programmazione specializzato per dispositivi di cassa.

Verifica: il programma è testato utilizzando diverse serie di codici a barre dei prodotti.

Convalida: anche se il programma è corretto nel sommare una serie di prodotti, i negozianti potrebbero trovarlo non adeguato perché necessitano che il programma sia anche in grado di includere sconti o di calcolare il resto.