

Marco Casu

∞ Ingegneria del Software ∞



SAPIENZA  
UNIVERSITÀ DI ROMA

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica  
Dipartimento di Informatica

Questo documento è distribuito sotto la licenza [GNU](#), è un resoconto degli appunti (eventualmente integrati con libri di testo) tratti dalle lezioni del corso di Ingegneria del Software per la laurea triennale in Informatica. Se dovessi notare errori, ti prego di segnalarmeli.



# INDICE

<b>1</b>	<b>Introduzione</b>	<b>3</b>
1.1	Modellazione . . . . .	3
1.2	Catene di Markov per la Progettazione . . . . .	4
1.3	Implementazione delle DTMC . . . . .	7
1.4	Raccolta dei File . . . . .	13
1.4.1	<code>main.h</code> . . . . .	13
1.4.2	<code>dtmc.h</code> . . . . .	14
1.4.3	<code>dtmc.cpp</code> . . . . .	15
1.4.4	<code>randgen.h</code> . . . . .	17
1.4.5	<code>randgen.cpp</code> . . . . .	17
<b>2</b>	<b>Planning del Progetto</b>	<b>18</b>
2.1	Sviluppo Plan-Driven . . . . .	18
2.2	Scheduling del Progetto . . . . .	20
2.3	Sviluppo Agile . . . . .	22
<b>3</b>	<b>Processi Software</b>	<b>24</b>
3.1	Definizione e Modelli . . . . .	24

## CAPITOLO

# 1

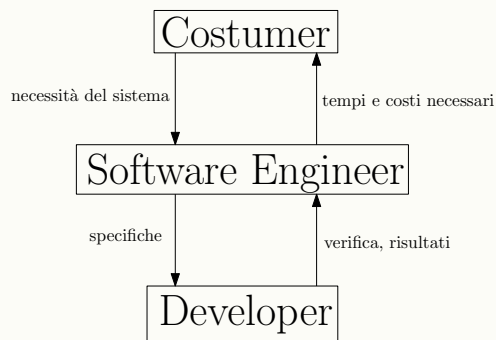
## INTRODUZIONE

Quando si vuole produrre software complesso, non terminante e di grandi dimensioni, è necessario ingegnerizzare il metodo di sviluppo ed astrarre il modello del software. L'obiettivo dell'ingegneria del software non riguarda il codice stesso, ma la definizione dell'architettura e delle specifiche del sistema. Vanno definiti i processi di sviluppo.

È cruciale l'analisi dei requisiti, e la modellizzazione delle specifiche di sistema, UML è un linguaggio che permette di descrivere la *dinamica* ed il comportamento del modello. Una volta definiti i requisiti, si affronta la fase di *planning*.

Un modello computazionale che risulterà utile prende il nome di *digital twin*, ossia una copia del software sulla quale è possibile testare i vari *scenari operativi*, tramite la generazione di test automatici. Essendo il sistema discreto, a stati non necessariamente finiti, è possibile descriverlo tramite una catena di Markov.

Nella produzione software prendono parte 3 principali attori



### 1.1 Modellazione

Il linguaggio UML è utilizzato per modellare sia i requisiti che il sistema stesso, esso è composto da differenti diagrammi

- activity
- use case
- sequence

- class
- state
- context

Si consideri il seguente esempio di un sistema di gestione di una clinica psichiatrica, lo schema in figura 1.1, è il context diagram, e rappresenta l'insieme dei servizi che il sistema offre. Un activity diagram describe

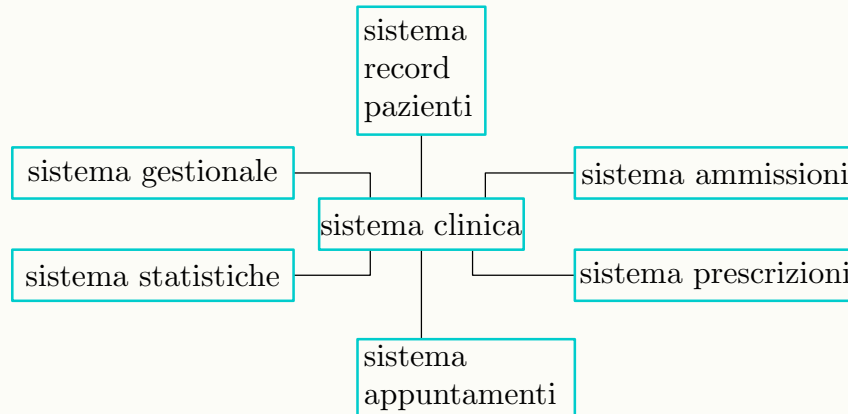


Figura 1.1: context diagram

l'evoluzione di una certa attività/task che il sistema deve poter implementare, si considere l'esempio in figura 1.2 riguardante l'inserimento di un paziente nella clinica.

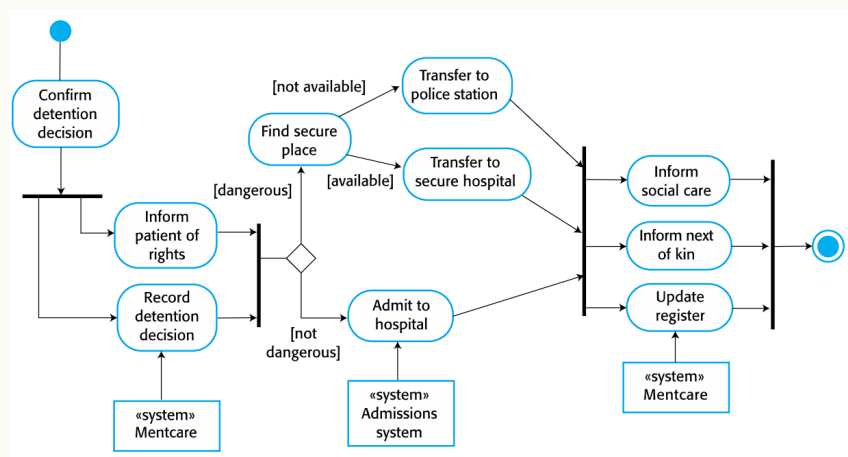


Figura 1.2: activity diagram



## 1.2 Catene di Markov per la Progettazione

La moderna progettazione di sistemi complessi si basa sul concetto di digital twin, in modo da poter definire un modello che simula il sistema su cui verranno eseguiti determinati esperimenti al fine di collaudo prima della progettazione vera e propria. Questo capitolo si concentra sull'aspetto *predittivo* dei digital twin : Si vuole utilizzare il modello per fare predizioni sull'andamento della progettazione.

**Definizione (DTMC) :** Una **Catena di Markov Discreta** è una tupla  $(U, X, Y, p, g)$  tale che

- $U$  è un insieme (possibilmente vuoto, finito o non) denotato *input/ingresso* del sistema.
- $X$  è un insieme non vuoto (finito o non) denotato *stato* del sistema.

- $Y$  è un insieme non vuoto (finito o non) denotato *output/uscita* del sistema.
- $p$  è la *probabilità di transizione*, è una funzione

$$p : X \times X \times U \rightarrow [0, 1]$$

tale che  $p(x'|x, u)$  rappresenta la probabilità che la DTMC si sposti dallo stato  $x$  allo stato  $x'$  quando l'ingresso è  $u$ . Inoltre, per ogni  $\tilde{x} \in X$  e per ogni  $\tilde{u} \in U$ , si ha che

$$\int_{x' \in X} p(x'|\tilde{x}, \tilde{u}) = 1$$

informalmente, per ogni possibile coppia stato-ingresso, esiste uno stato successivo.

- $g$  è una funzione

$$g : X \rightarrow Y$$

detta *funzione di output*.

Data una catena di markov  $M$ , denotiamo  $M(U), M(X), M(Y)$  rispettivamente : gli ingressi, gli stati e le uscite.

**Esempio tempi di completamento** : Si consideri il processo di sviluppo mostrato in figura 1.3, si vuole calcolare la probabilità che il processo termini in esattamente 10 mesi.

Si identificano i possibili cammini la cui somma dei sotto processi impiega 10 mesi

$$\begin{aligned} 0 &\rightarrow 1 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \\ 0 &\rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 3 \rightarrow 4 \end{aligned}$$

La probabilità che ciò accada è la somma delle probabilità sui due percorsi

$$\begin{aligned} 1 \cdot 0.3 \cdot 0.7 \cdot 0.6 \cdot 0.6 &= 0.0756 \\ 1 \cdot 0.7 \cdot 0.6 \cdot 0.1 \cdot 0.6 &= 0.02520 \end{aligned} \implies p(\text{finire in 10 mesi}) = 0.1008$$

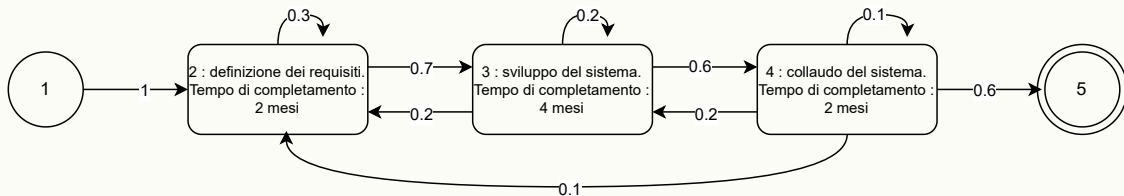


Figura 1.3: Esempio 1

Qual'è invece la probabilità di finire il processo in esattamente 12 mesi? Si identificano i cammini

$$\begin{aligned} 0 &\rightarrow 1 \rightarrow 1 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \\ 0 &\rightarrow 1 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 3 \rightarrow 4 \\ 0 &\rightarrow 1 \rightarrow 2 \rightarrow 2 \rightarrow 3 \rightarrow 4 \\ 0 &\rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 3 \rightarrow 3 \rightarrow 4 \end{aligned}$$

la probabilità è

$$\begin{aligned} &1 \cdot 0.3 \cdot 0.3 \cdot 0.7 \cdot 0.6 \cdot 0.6 + \\ &1 \cdot 0.3 \cdot 0.7 \cdot 0.6 \cdot 0.1 \cdot 0.6 + \\ &1 \cdot 0.7 \cdot 0.2 \cdot 0.6 \cdot 0.6 + \\ &1 \cdot 0.7 \cdot 0.6 \cdot 0.1 \cdot 0.1 \cdot 0.6 = \\ &0.022680 + 0.007560 \cdot 0.05040 \cdot 0.002520 = 0.08316 \end{aligned}$$

**Esempio costi di completamento** : Si consideri il processo di sviluppo mostrato in figura 1.4, si vuole calcolare la probabilità che il processo abbia un costo totale di esattamente 80.000 euro.

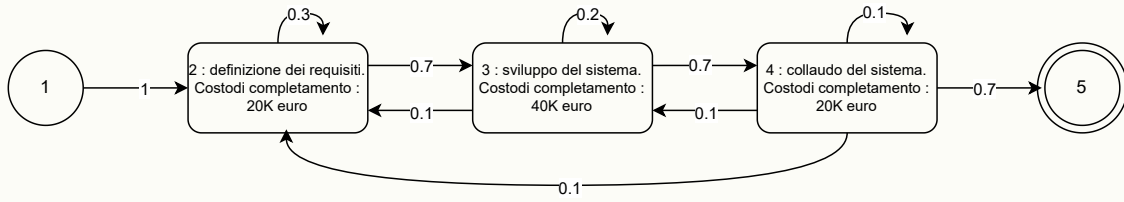


Figura 1.4: Esempio 2

L'unico cammino possibile è

$$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4$$

La probabilità è

$$1 \cdot 0.7 \cdot 0.7 \cdot 0.7 = 0.343$$

**Esempio costi di design** : Si consideri il processo di sviluppo mostrato in figura 1.5, quale deve essere il minimo valore di  $p$  tale che la probabilità che il processo termini in esattamente 8 mesi sia almeno 0.5?

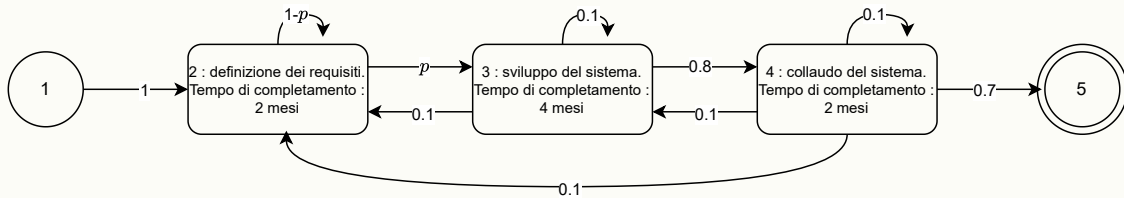


Figura 1.5: Esempio 3

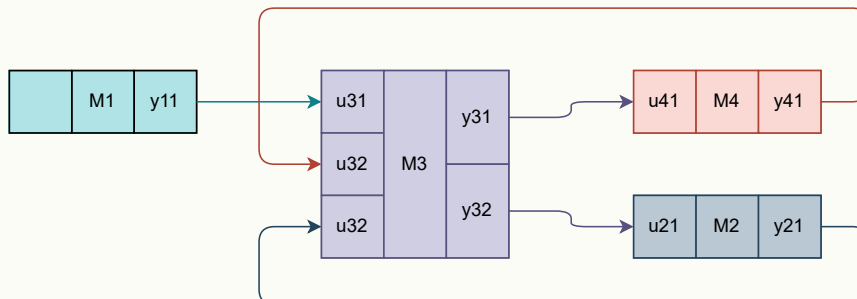
L'unico cammino possibile è

$$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4$$

la cui probabilità deve essere

$$1 \cdot p \cdot 0.8 \cdot 0.7 \geq 0.5 \implies p \geq \frac{0.5}{0.8 \cdot 0.7} = 0.8928$$

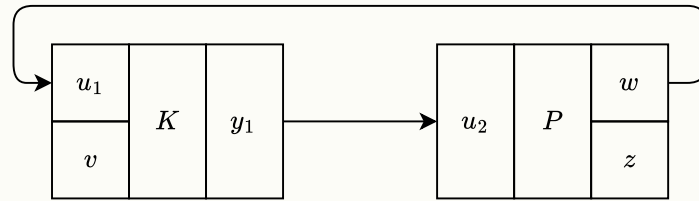
Una **rete di catene di Markov** è un insieme di catene di Markov interconnesse tramite gli ingressi e le uscite.



Inoltre, una rete di catene di Markov è rappresentabile come una catena di Markov singola, si considerino le catene di Markov  $K, P$  in figura 1.6, formalmente

$$K = (U_1 \times V, X_1, Y_1, p_1, g_1)$$

$$P = (U_2, X_2, W \times Z, p_2, (g_{2w}, g_{2z}))$$

Figura 1.6:  $K$  e  $P$  interconnesse

Nota :  $u_1$  rappresenta un elemento dell'insieme  $U_1$  (ingresso di  $K$ ),  $v \in V$ ,  $y_1 \in Y_1 \dots$  analogo per gli altri elementi.

Un ingresso di  $K$  è un uscita di  $P$

$$M_1(U_1) = M_2(W)$$

e l'ingresso di  $P$  è l'uscita di  $K$

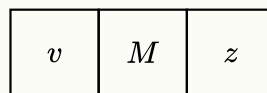
$$M_2(U_2) = M_1(Y_1)$$

Si può definire il sistema globale equivalente

$$M = (V, X_1 \times X_2, Z, p, g)$$

Dove

- $p((x'_1, x'_2)|(x_1, x_2), v) = p_1(x'_1|x_1, (g_{2_w}(x_2), v)) \cdot p_2(x'_2|x_2, g_1(x_1))$
- $g(x_1, x_2) = g_{2_z}(x_2)$



## 1.3 Implementazione delle DTMC

In questa sezione verrà trattata una possibile implementazione delle DTMC in *C++*. Si assume che il lettore sia a conoscenza delle regole di scrittura e strutturazione degli header e file C.

Una DTMC sarà implementata in due file

`dtmc.h`  
`dtmc.cpp`

Nel file `dtmc.h` come prima cosa, si definisce il *tipo* delle variabili di ingresso, stato, e uscita, tramite il costrutto `typedef`.

```
1 typedef double state_type;
2 typedef double input_type;
3 typedef double output_type;
```

Va poi definita la classe `DTMC` le cui istanze rappresenteranno proprio le catene di Markov, gli elementi privati di tale classe saranno 2

- Un puntatore ad una classe `RandGen` definita in un'apposito file `randgen.cpp`
- Una funzione `dtmc_body` che verrà definita nel file `dtmc.cpp`



```

1 class DTMC {
2
3 private:
4     RandGen *p; // pointer to random generator class
5     void dtmc_body(RandGen *p);

```

Fra gli elementi pubblici della classe ci saranno poi le dimensioni dei vettori di ingresso, stato ed uscita [di default, assumeremo che tali vettori siano in  $\mathbb{R}^2$ ], insieme a delle variabili che rappresentano lo stato corrente, l'ingresso e l'uscita. Ci saranno poi due costruttori, uno più esplicito dell'altro

- Il primo costruttore riceve in ingresso esclusivamente il puntatore all'oggetto di tipo `RandGen`, lasciando invariate le dimensioni dei vettori di stato, ingresso ed uscita a 2.
- Il secondo costruttore è più esplicito e permette la modifica delle dimensioni dei vettori di stato, ingresso ed uscita.

```

1 public:
2     int STATE_SIZE = 2; // state size
3     int INPUT_SIZE = 2; // input size
4     int OUTPUT_SIZE = 2; // output size
5
6     state_type *x; // present state
7     input_type *u; // input
8     output_type *y; // output
9
10    DTMC(RandGen *p);
11    DTMC(int xsize, int usize, int ysize, RandGen *p);

```

Ci sono poi le varie funzioni della classe, precisamente per

- inizializzare la catena con lo stato iniziale
- calcolare il prossimo stato dato lo stato attuale e l'ingresso
- dato uno stato restituisce l'uscita del sistema

Ci sono poi due funzioni di servizio per la visualizzazione dello stato in console (oppure su un file).

```

1 // define initial state
2 void init();
3
4 // update state
5 void next();
6
7 // update output
8 void output();
9
10 // print log on stdout
11 void printlog(int t);
12
13 // print log on file
14 void fprintflog(FILE *fp, int t);

```

Verrà considerata adesso la definizione delle funzioni dichiarate, precisamente, nel file `dtmc.cpp`. Ovviamente vanno incluse le dichiarazioni.

```

1 #include "dtmc.h"
2 using namespace std;

```

La funzione `dtmc_body` verrà richiamata dai costruttori per definire ed inizializzare la catena di Markov. Deve ricevere in input il puntatore al generatore di numeri casuali ed assegnarlo alla variabile interna (privata). Dopo aver fatto ciò viene fatto un controllo sulle dimensioni del vettore di stato che devono essere strettamente maggiori di zero.

```

1 void dtmc_body(RandGen *myp) {
2     p = myp;
3     if (STATE_SIZE <= 0)
4     {
5         fprintf(stderr, "DTMC(): Error: state size cannot be 0\n");
6         exit(1);
7     }

```

A questo punto, bisogna inizializzare il vettore di stato, dato che le dimensioni non sono note a priori, sarà necessario allocare dinamicamente lo spazio.

```

1     x = (state_type *) malloc(sizeof(state_type)*STATE_SIZE);
2
3     if (x == NULL)
4     {
5         fprintf(stderr, "DTMC(): memory allocation for state failed\n");
6         exit(1);
7     }

```

Si prosegue analogamente con l'ingresso e l'uscita, si ricordi che una DTMC può avere ingresso nullo, ma deve necessariamente esserci un'uscita.

```

1     if (INPUT_SIZE <= 0)
2     {
3         u = NULL;
4     }
5     else
6     {
7         u = (input_type *) malloc(sizeof(input_type)*INPUT_SIZE);
8         if (u == NULL)
9         {
10            fprintf(stderr, "DTMC(): memory allocation
11                        for input failed\n");
12            exit(1);
13        }
14    }
15
16    if (OUTPUT_SIZE <= 0)
17    {
18        fprintf(stderr, "DTMC(): Error: output size cannot be 0\n");
19        exit(1);
20    }
21
22    y = (output_type *) malloc(sizeof(output_type)*OUTPUT_SIZE);
23    if (y == NULL)
24    {
25        fprintf(stderr, "DTMC(): memory allocation
26                        for output failed\n");
27        exit(1);
28    }
29 }

```

A tal punto i due costruttori possono richiamare tale funzione per inizializzare la catena.

```

1 DTMC(RandGen *myp) {
2     dtmc_body(myp);
3 }
4
5 DTMC(int xsize, int usize, int ysize, RandGen *myp) {
6     STATE_SIZE = xsize;
7     INPUT_SIZE = usize;
8     OUTPUT_SIZE = ysize;

```

```

9   dtmc_body(myp);
10 }

```

La funzione di inizializzazione inizializza a zero il vettore di stato.

```

1 void init() {
2     /* initial state */
3     for (int i = 0; i < STATE_SIZE; i++)
4     {
5         x[i] = 0;
6     }
7 }

```

La funzione `next` dovrà calcolare lo stato successivo

```

1 void next() {
2     /* initial state */
3     for (int i = 0; i < STATE_SIZE; i++)
4     {
5         x[i] = p -> get_random_double();
6     }
7 }

```

La funzione `get_random_double` è definita in `RandGen` e restituisce un double casuale compreso fra 0 e 1. In questo caso specifico, lo stato è un vettore di numeri reali, quindi è una DTMC a stati infiniti. Si definisce poi la funzione in output, in questo specifico caso, l'output sarà identico allo stato.

```

1 void output() {
2     /* initial state */
3     for (int i = 0; i < OUTPUT_SIZE; i++)
4     {
5         y[i] = x[i];
6     }
7 }

```

Ovviamente le dimensioni devono combaciare. Si definiscono poi le funzioni di servizio che si occuperanno di eseguire un print dello stato della catena di Markov in formato CSV, così poi da rendere facile la generazione di un grafico.

```

1 void printlog(int t) {
2     /* t e' il tempo corrente */
3     int i;
4     printf("%d ", t);
5     for (i = 0; i < STATE_SIZE; i++)
6     {
7         printf("%lf ", x[i]);
8     }
9     for (i = 0; i < OUTPUT_SIZE; i++)
10    {
11        printf("%lf ", y[i]);
12    }
13    for (i = 0; i < INPUT_SIZE; i++)
14    {
15        printf("%lf ", u[i]);
16    }
17    printf("\n");
18 }

```

La funzione `fprintflog` è analoga ma scrive su un file.

Si da ora una descrizione della classe `RandGen`, definita nei file

`randgen.h`

`randgen.cpp`

Il file header è molto semplice, contiene la dichiarazione del costruttore, una variabile (seed) e due funzioni per la generazione di numeri double e interi.

```
1 class RandGen{
2 public:
3     RandGen();
4     unsigned int seed;
5     double get_random_double();
6     int get_random_int();
7 }
```

Il seed è un intero che condiziona la generazione dei valori. Il file `randgen.cpp` contiene ovviamente le definizioni ed utilizza una serie di funzioni già preesistenti del C++ definite nella libreria matematica. Oltre ad includere il file header, si utilizza un oggetto già definito per la generazione del seed. Dopo di ciò, si inizializzano arbitrariamente delle distribuzioni. Nell'esempio seguente, per i double i valori che possono essere estratti sono compresi fra 0 ed 1, per gli interi da -10 a 10.

```
1 #include "randgen.h"
2 random_device myRandomDevice;
3 unsigned int myseed = myRandomDevice();
4
5 default_random_engine myRandomEngine(myseed);
6 uniform_int_distribution<int> myUnifIntDist(-10,10);
7 uniform_real_distribution<double> myUnifRealDist(0.0,1.0);
8 RandGen(){
9     seed=myseed;
10 }
```

Si definiscono poi le funzioni che generano il numero casuale.

```
1 double get_random_double() {
2     return(myUnifRealDist(myRandomEngine));
3 }
4
5 int RandGen::get_random_int() {
6     return(myUnifIntDist(myRandomEngine));
7 }
```

A questo punto, si è definita in tutto e per tutto una catena di markov a tempo discreto, si vuole scrivere un *simulatore* che si occupi di istanziare la classe e di eseguire dei test. Si definiscono quindi due file

`simulator.h`  
`simulator.cpp`

Come prima cosa nel file header si definisce la classe `Simulator` i cui oggetti faranno "muovere" la/le catena/e di Markov. È una classe con una sola funzione pubblica.

```
1 #include "main.h"
2 class Simulator {
3 public:
4     // run simulation with horizon T
5     void run(int T);
6 };
```

La funzione `run` prende come argomento "l'orizzonte", ossia il numero di step finiti che la catena di Markov deve compiere.

Il simulatore è in tutto e per tutto la funzione `run` definita nel file `simulator.cpp`, che definisce come prima cosa gli elementi necessari alla DTMC.

```
1 void Simulator(int T){
2     RandGen p;
3     DTMC mc1(&p);
4     int t=0;
5     FILE *fp;
```

```

6  printf("seed : %u\n",p.seed); //servizio
7  printf("inizialization : t= %d\n",t); //servizio
8  mc1.init();
9  fp=fopen("logfile.csv","w");

```

Il tempo `t` si inizializza a zero. Il file `fp` sarà quello in cui verrà scritto il log della catena. A tal punto va eseguita la simulazione vera e propria eseguendo precisamente `T` step sulla catena di Markov, scrivendo lo stato sul log ad ogni passo.

```

1  for (t = 0; t <= T; t++)
2  {
3      mc1.output();
4      mc1.u[0] = t;  mc1.u[1] = mc1.y[1] + 1;
5      mc1.printlog(t);
6      mc1.fprintflog(fp, t);
7      mc1.next();
8      #if 0
9          printf("get state of system at time %d\n", t);
10         printf("get output of system at time %d\n", t);
11         printf("update inputs with outputs at time %d\n", t);
12         printf("update state of system with
13             inputs at time %d\n\n", t);
14     #endif
15 }
16 fclose(fp);
17 }

```

Ad ogni passo, il vettore in input assume il valore (tempo,uscita + 1). L'output si ricordi essere funzione esclusivamente dello stato e non dell'input, è quindi perfettamente legittimo utilizzare come input l'uscita del sistema, in tal modo detto "a retroazione".

$$u(t) = \begin{bmatrix} t \\ y(t-1) + 1 \end{bmatrix}$$

Il progetto può essere poi compilato con un makefile, un possibile makefile generico è il seguente.

```

CC=g++
#CFLAGS=-std=c++11 -lm -I. -I/usr/include/postgresql -lpq
#CFLAGS=-std=c++11 -I. -I/usr/include/postgresql -lpq -lm
CFLAGS=-std=c++11 -I. -lm
DEPS = $(wildcard *.h)
objects := $(patsubst %.cpp,%.o,$(wildcard *.cpp))
%.o:    %.cpp $(DEPS)
        $(CC) -c -o $@ $< $(CFLAGS)

main:   $(objects)
        $(CC) -o main $(objects) $(CFLAGS)

clear:
    rm *.o

clearall:
    rm main ; rm *.o

```

## 1.4 Raccolta dei File

### 1.4.1 main.h

```
1 #ifndef main_h
2 #define main_h
3
4 using namespace std;
5
6 #include <iostream>
7 #include <random>
8
9 #include "randgen.h"
10 #include "dtmc.h"
11 #include "simulator.h"
12
13 #if 0
14 #include "clock.h"
15 #include "logger.h"
16 #include "gtime.h"
17 #include "timer.h"
18 #include "basic.h"
19 #include "system.h"
20
21 #include "ctr.h"
22 #include "plant.h"
23 #include "root.h"
24 #endif
25
26 #define DEBUG 1000
27
28 #define HORIZON 10
29
30 #if 0
31 extern Logger val2log;
32 extern GlobalTime gtime;
33 extern Clock ck;
34 extern random_device myRandomDevice;
35 extern unsigned int seed;
36 extern default_random_engine myRandomEngine;
37 extern uniform_int_distribution<int> myUnifIntDist;
38 extern uniform_real_distribution<double> myUnifRealDist;
39 #endif
40
41 #if 0
42     random_device myRandomDevice;
43     unsigned int seed = myRandomDevice();
44     // Initialize a default_random_engine with the seed
45     default_random_engine myRandomEngine(seed);
46     // Initialize a uniform_int_distribution to produce values between -10
47     // and 10
48     uniform_int_distribution<int> myUnifIntDist(-10, 10);
49     // Initialize a uniform_real_distribution to produce values between 0
50     // and 1
51     uniform_real_distribution<double> myUnifRealDist(0.0, 1.0);
52 #endif
53 #endif
```

### 1.4.2 dtmc.h

```
1 #ifndef dtmc_h
2 #define dtmc_h
3
4 #include "main.h" /*include il file dove e'
5                     definita la classe RandGen*/
6
7 typedef double state_type;
8 typedef double input_type;
9 typedef double output_type;
10
11 class DTMC {
12
13 private:
14
15     RandGen *p; // pointer to random generator class
16
17     void dtmc_body(RandGen *p);
18
19 public:
20
21     int STATE_SIZE = 2; // state size
22     int INPUT_SIZE = 2; // input size
23     int OUTPUT_SIZE = 2; // output size
24
25     state_type *x; // present state
26     input_type *u; // input
27     output_type *y; // output
28
29     DTMC(RandGen *p);
30     DTMC(int xsize, int usize, int ysize, RandGen *p);
31
32     // define initial state
33     void init();
34
35     // update state
36     void next();
37
38     // update output
39     void output();
40
41     // print log on stdout
42     void printlog(int t);
43
44     // print log on file
45     void fprintflog(FILE *fp, int t);
46 };
47
48 #endif
```

### 1.4.3 dtmc.cpp

```
1 #include "main.h"
2 #include "dtmc.h"
3
4 void dtmc_body(RandGen *myp) {
5     p = myp;
6     if (STATE_SIZE <= 0)
7     {
8         fprintf(stderr, "DTMC(): Error: state size cannot be 0\n");
9         exit(1);
10    }
11    x = (state_type *) malloc(sizeof(state_type)*STATE_SIZE);
12    if (x == NULL)
13    {
14        fprintf(stderr, "DTMC(): memory allocation for state failed\n");
15        exit(1);
16    }
17    if (INPUT_SIZE <= 0)
18    {
19        u = NULL;
20    }
21    else
22    {
23        u = (input_type *) malloc(sizeof(input_type)*INPUT_SIZE);
24        if (u == NULL)
25        {
26            fprintf(stderr, "DTMC(): memory allocation
27                        for input failed\n");
28            exit(1);
29        }
30    }
31    if (OUTPUT_SIZE <= 0)
32    {
33        fprintf(stderr, "DTMC(): Error: output size cannot be 0\n");
34        exit(1);
35    }
36    y = (output_type *) malloc(sizeof(output_type)*OUTPUT_SIZE);
37    if (y == NULL)
38    {
39        fprintf(stderr, "DTMC(): memory allocation
40                        for output failed\n");
41        exit(1);
42    }
43 }
44
45
46 DTMC(RandGen *myp) {
47     dtmc_body(myp);
48 }
49
50 DTMC(int xsize, int usize, int ysize, RandGen *myp) {
51     STATE_SIZE = xsize;
52     INPUT_SIZE = usize;
53     OUTPUT_SIZE = ysize;
54     dtmc_body(myp);
55 }
56
```





```
57 void init() {
58     int i;
59     for (i = 0; i < STATE_SIZE; i++)
60         x[i] = 0;
61 }
62 void next() {
63     int i;
64     for (i = 0; i < STATE_SIZE; i++)
65         x[i] = p -> get_random_double();
66 }
67 void output() {
68     int i;
69     for (i = 0; i < OUTPUT_SIZE; i++)
70         y[i] = x[i];
71 }
72
73 void printlog(int t) {
74     int i;
75     printf("%d ", t);
76     for (i = 0; i < STATE_SIZE; i++)
77     {
78         printf("%lf ", x[i]);
79     }
80     for (i = 0; i < OUTPUT_SIZE; i++)
81     {
82         printf("%lf ", y[i]);
83     }
84     for (i = 0; i < INPUT_SIZE; i++)
85     {
86         printf("%lf ", u[i]);
87     }
88     printf("\n");
89 }
90
91 void fprintflog(FILE *fp, int t) {
92     FILE *fp;
93     int i;
94     fp = fopen(filename, "w");
95     fprintf(fp, "%d ", t);
96     for (i = 0; i < STATE_SIZE; i++)
97     {
98         fprintf(fp, "%lf ", x[i]);
99     }
100     for (i = 0; i < OUTPUT_SIZE; i++)
101     {
102         fprintf(fp, "%lf ", y[i]);
103     }
104     for (i = 0; i < INPUT_SIZE; i++)
105     {
106         fprintf(fp, "%lf ", u[i]);
107     }
108     fprintf(fp, "\n");
109 }
```



#### 1.4.4 randgen.h

```
1 #ifndef randgen_h
2 #define randgen_h
3 #include "main.h"
4
5 class RandGen {
6 public:
7     RandGen();
8     unsigned int seed;
9     double get_random_double();
10    int get_random_int();
11 };
12 #endif
```

#### 1.4.5 randgen.cpp

```
1 #include "main.h"
2 #include "randgen.h"
3
4 random_device myRandomDevice;
5 unsigned int myseed = myRandomDevice();
6
7 default_random_engine myRandomEngine(myseed);
8 uniform_int_distribution<int> myUnifIntDist(-10, 10);
9 uniform_real_distribution<double> myUnifRealDist(0.0, 1.0);
10 RandGen() { seed = myseed; }
11
12 double get_random_double() {
13     return(myUnifRealDist(myRandomEngine));
14 }
15
16 int get_random_int() {
17     return(myUnifIntDist(myRandomEngine));
18 }
```

## CAPITOLO

# 2

## PLANNING DEL PROGETTO

Il planning di un progetto, consiste nel suddividere il carico di lavoro in diverse parti, da assegnare ai vari membri del team, cercando di prevedere possibili problemi che potrebbero insorgere durante lo sviluppo, pensando e preparando eventuali modi per risolverli.

Il *piano del progetto* viene preparato all'inizio dei lavori, viene utilizzato per comunicare ai vari membri del team ed al cliente come esso è stato suddiviso. Le fasi in cui il planning viene definito sono

- Durante la proposta, quando avviene la contrattazione con il cliente riguardo lo sviluppo del software
- Durante la fase di avvio dei lavori, quando si decide come e a chi assegnare il lavoro, e quali risorse dovranno essere allocate
- Periodicamente durante lo sviluppo, monitorando ed appositamente modificando i piani in base all'andamento del progetto e alle esperienze pregresse

Lo scopo del planning è quello di avere un'idea chiara sul progetto in modo che si possa decidere un prezzo d'accordo con il cliente, valutando e stimando quanto denaro sarà necessario per lo sviluppo considerando variabili del tipo

- costo dei dipendenti
- costo dell'hardware necessario
- costo del software

Durante la fase di planning, sono noti i requisiti del sistema, ma non è ancora chiara la struttura del software, tanto meno la sua implementazione, il planning deve essere preciso a sufficienza per far sì che sia possibile definire il budget e lo staff necessario. Inoltre bisogna definire i meccanismi con la quale verrà monitorato lo sviluppo.



### 2.1 Sviluppo Plan-Driven

Con *plan driven development*, si intende un approccio all'ingegneria del software in cui i processi di sviluppo sono programmati in principio, in maniera dettagliata. Si basa sulle tecniche di gestione, tipiche dei progetti ingegneristici, e sulle tecniche "classiche" di gestione di grandi progetti software.

Un project plan ha l'obiettivo di definire e monitorare il lavoro da svolgere, in che modo deve essere

svolto, da chi, e quali prodotti sono necessari. È scopo del manager, servirsi di un project plan (che da ora chiameremo semplicemente "piano") per supportare le decisioni da prendere durante lo sviluppo, e per misurarne il progresso.

- Un lato favorevole di tale approccio, è che una pianificazione a monte permette di aggirare problemi organizzativi in principio, determinando potenziali problemi e dipendenze da soddisfare prima che il progetto sia avviato, piuttosto che durante la lavorazione.

*meglio prevenire che curare*

- Un lato sfavorevole, è che molte decisioni prese in principio vanno riviste dati possibili cambiamenti dell'ambiente in cui il software deve essere adoperato.

Un piano deve definire, le risorse disponibili per il progetto, la suddivisione del carico di lavoro, una schedule per portare a termine il lavoro. Precisamente, un piano consiste nelle seguenti sezioni

1. introduzione
2. organizzazione del progetto
3. analisi dei rischi
4. risorse hardware e software necessarie
5. struttura di scomposizione del lavoro
6. schedule del progetto
7. meccanismi di monitoraggio e report

Ci sono inoltre altri tipi di "piano" che possono essere aggiunti a quello principale come supporto:

Piano	Descrizione
<b>Configuration management plan</b>	descrive la configurazione delle procedure di gestione, la loro struttura ed il loro utilizzo
<b>Deployment plan</b>	descrive come il software deve essere integrato con l'apposito hardware di riferimento del cliente, con eventuali piani di migrazione dei dati in uso su sistemi precedentemente adoperati
<b>Maintenance plan</b>	predizione dei requisiti, costi e lavoro per la manutenzione del software
<b>Quality plan</b>	descrizione delle procedure e standard di qualità utilizzati nel progetto
<b>Validation plan</b>	descrizione degli approcci, risorse e schedule utilizzati per la convalida del sistema

Il planning del progetto è un processo *iterativo* che viene sottoposto ad inevitabili cambiamenti durante lo sviluppo progressivo. Con l'aumentare delle informazioni relative al sistema durante lo sviluppo, è doveroso revisionare i requisiti iniziali, dei cambiamenti nel business possono portare a grandi cambiamenti nei requisiti del progetto, portando ad un eventuale ri-pianificazione totale.

### Assunzioni del planning

- Le assunzioni da fare durante la definizione del project plan non devono essere ottimistiche, bensì realistiche.
- Durante lo sviluppo insorgeranno inevitabilmente dei problemi che causeranno dei ritardi nella consegna.
- Le assunzioni iniziali e lo scheduling saranno inevitabilmente soggetti a problemi inaspettati.

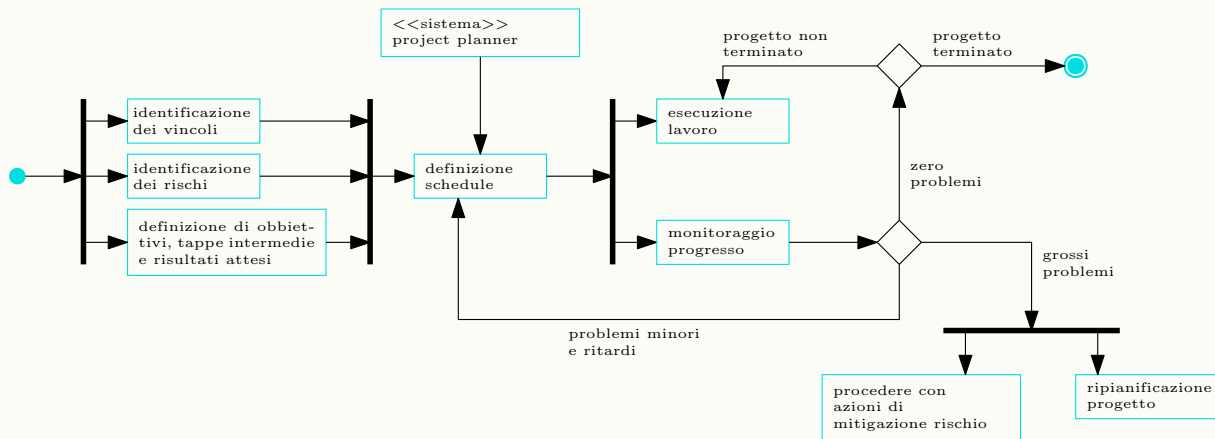


Figura 2.1: Processi del project planning

- Bisogna sempre considerare ogni imprevisto, in modo che eventuali problemi non siano troppo gravanti sulla schedule.

In caso di seri problemi durante lo sviluppo, è necessario applicare delle procedure di *mitigazione del rischio*, onde evitare il fallimento del progetto. In congiunzione a ciò, può essere necessaria la ripianificazione del progetto.

Ciò può comportare la rinegoziazione con il cliente dei risultati attesi e dei vincoli da rispettare. Una nuova schedule, e data di consegna dovrà essere definita in modo da stabilire un accordo con il cliente.



## 2.2 Scheduling del Progetto

**Definizione :** Con *scheduling del progetto* si intende il processo di decisione riguardante il come il carico di lavoro di un progetto deve essere organizzato in differenti attività, ed in che ordine queste attività devono essere eseguite.

Viene stimato un calendario con i tempi necessari al completamento delle attività, lo sforzo necessario ed il personale al quale delegarlo. È anche necessaria una stima delle risorse necessarie, come lo spazio su disco necessario per il software, ed il budget da muovere.

1. Suddivisione del progetto in varie attività e stima delle risorse per ognuna di esse
2. Organizzazione delle attività da eseguire in concorrenza per ottimizzare i tempi
3. Minimizzazione delle dipendenze fra le varie attività, in modo da ridurre i ritardi dovuti ad attese

Questi ultimi fattori dipendono anche dall'intuizione e dalla esperienza del project manager.

### Problemi nello scheduling

- Stimare la difficoltà del lavoro ed i costi di sviluppo è molto difficile
- La produttività non è proporzionale al numero di persone coinvolte nel lavoro
- L'aggiunta di personale a progetto già avviato può causare ritardi
- Accade sempre l'inaspettato, bisogna considerare ogni evenienza nel planning

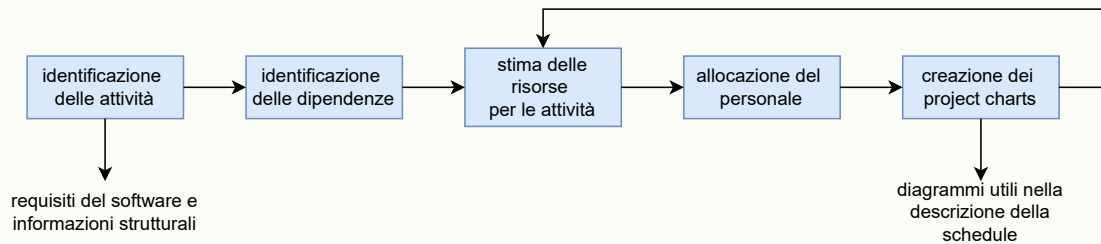


Figura 2.2: Scheduling

Esiste un'annotazione grafica utile nella rappresentazione della schedule, mostra le diverse attività, il periodo in cui vanno terminate e le varie dipendenze fra esse, il diagramma a barre mostra le attività come risorse da disporre sull'asse dei tempi. Le "project activities" sono gli elementi di base del grafico, comprendono

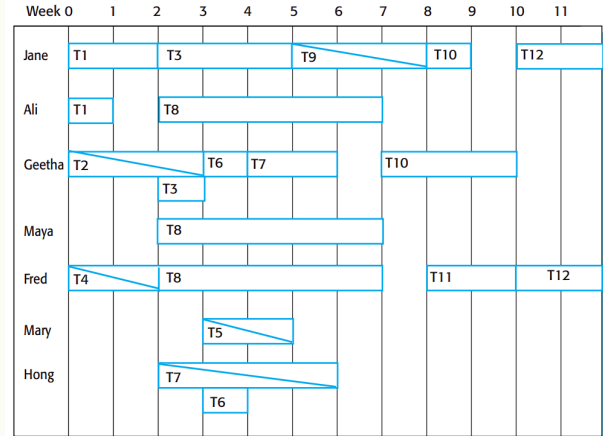
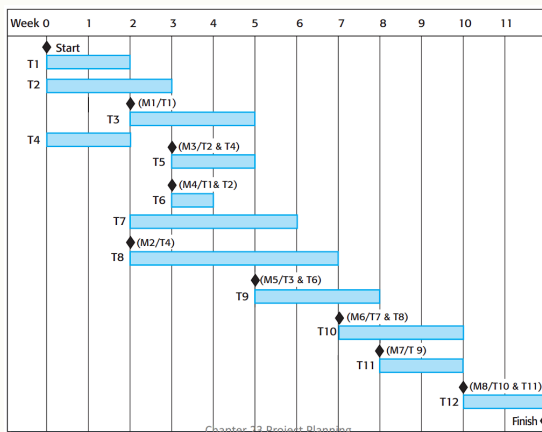
- una durata sul calendario, di giorni o mesi
- un carico di lavoro stimato, misurato in numero di impiegati al giorno necessari
- una deadline per ogni attività che ne vincola il completamento entro una certa data
- un punto specifico che descrive il terminamento di un'attività, può essere un documento, una riunione o il completamento di tutti i test

Una **milestone** non è altro che una "tappa fondamentale" durante lo svolgimento delle varie attività, può rappresentare un momento in cui si valuta la progressione del progetto.

Con **deliverables** si definiscono dei risultati ottenuti durante la lavorazione da presentare al cliente.

task	personale necessario (persone/giorno)	durata in giorni	dipendenze
T1	15	10	
T2	8	15	
T3	20	15	T1 (M1)
T4	5	10	
T5	5	10	T2, T4 (M3)
T6	10	5	T1, T2 (M4)
T7	25	20	T1 (M1)
T8	75	25	T4 (M2)
T9	10		T3, T6 (M5)
T10	20	15	T7, T8 (M6)
T11	10	10	T9 (M7)
T12	20	10	T10, T11 (M8)

Nella tabella sopramostrata, sono riportate diverse attività di un processo di sviluppo, ogni attività  $T_i$  (o task) ha associato un certo numero di persone necessarie ed una durata in giorni. Inoltre, un task può dipendere da uno o più task. Con  $M_i$  si identificano le milestones.



Un *activity bar chart* è un diagramma in cui sull'asse temporale vengono rappresentate le varie attività ed il loro scheduling, in modo che le varie dipendenze siano rispettate.

Analogamente, uno *staff allocation chart* mostra la distribuzione del personale suddiviso nelle varie settimane, indicando il progetto alla quale lavorano.



## 2.3 Sviluppo Agile

Con sviluppo agile, si denota una metodologia di approccio iterativa in cui il processo di sviluppo ed i requisiti vengono rimodellati dinamicamente, in quanto il progetto in fase di sviluppo viene considerato dal cliente prima del suo completamento. Diversamente dallo sviluppo plan driven, questo approccio non è predeterminato e le decisioni vengono prese durante lo sviluppo, possibile aggiunte o rimozioni dipendono da come il progetto procede e dalle nuove priorità del cliente.

Le priorità e le esigenze del cliente cambiano, quindi ha senso avere un piano flessibile che possa accogliere questi cambiamenti.

Lo sviluppo agile prevede due stadi

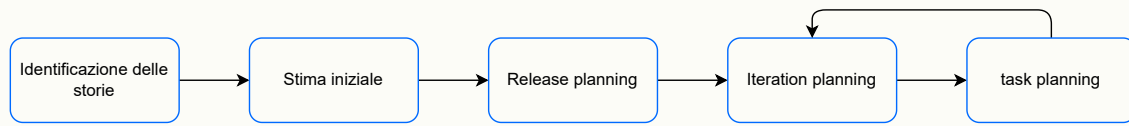
- *Release planning* : prevede la pianificazione a lungo termine delle funzionalità che dovranno essere implementate nei mesi a venire.
- *Iteration planning* : ci si concentra sugli incrementi a breve termine del sistema, tipicamente si tratta di aggiunte che richiedono dalle 2 alle 4 settimane.

Una delle metodologie dello sviluppo agile è la pianificazione basata su storie (**story-based planning**). Consiste nel suddividere le funzionalità di un sistema in piccole unità chiamate "storie utente" (*user stories*). In particolare, sono previsti i seguenti step

1. Creazione delle storie: Le storie utente sono brevi descrizioni di una funzionalità del sistema dal punto di vista dell'utente finale. Solitamente seguono un formato semplice: "Come utente, voglio [azione] in modo da [obiettivo]".
2. Stima dello sforzo: Il team di sviluppo stima il tempo necessario per implementare ciascuna storia, assegnandole un valore numerico (effort points) che riflette la complessità e la dimensione della storia.
3. Prioritizzazione: Le storie vengono ordinate in base alla priorità, tenendo conto di fattori come il valore commerciale, l'urgenza e le dipendenze.
4. Pianificazione delle iterazioni: Le storie prioritarie vengono raggruppate in iterazioni (sprint) di durata definita.
5. Misurazione della velocità: Viene calcolata la "velocità" del team, ovvero la quantità di lavoro (misurata in effort points) che il team riesce a completare in un'iterazione.



6. Previsione: Sulla base della velocità, il team può stimare il tempo totale necessario per completare il progetto.



Il Release Planning si occupa di selezionare e raffinare le storie utente (features) che verranno implementate in una specifica versione (release) del prodotto. Questo processo definisce quali funzionalità saranno incluse nel rilascio e l'ordine in cui verranno sviluppate.

L'Iteration planning si concentra sulla scelta delle storie utente che verranno implementate in ciascuna iterazione (spesso chiamate sprint), che sono periodi di tempo fissi (solitamente 2 o 3 settimane) durante i quali il team si impegna a completare un insieme di lavoro. Il numero di storie scelte per ogni iterazione dipende dalla capacità del team (velocità) di completare il lavoro.

Durante la fase di pianificazione dei task, gli sviluppatori suddividono le storie utente in compiti di sviluppo più piccoli.

- Un task di sviluppo dovrebbe richiedere tra le 4 e le 16 ore.
- Tutti i task necessari per completare tutte le storie di quell'iterazione vengono elencati.
- I singoli sviluppatori si offrono volontari per i task specifici che desiderano implementare.

Benefici di questo approccio:

- L'intero team ha una visione d'insieme dei task da completare in un'iterazione.
- Gli sviluppatori si sentono responsabili dei loro task e questo li motiva a completarli.

Un incremento del software<sup>1</sup> viene sempre consegnato alla fine di ogni iterazione del progetto. Se le funzionalità da includere nell'incremento non possono essere completate nel tempo previsto, si riducono gli obiettivi prefissati del lavoro, ma la tempistica di consegna non viene mai prolungata.

Svantaggi dell'agile planning

- La pianificazione Agile dipende fortemente dal coinvolgimento e dalla disponibilità del cliente.
- può essere difficile da organizzare, poiché i clienti devono spesso dare priorità ad altro lavoro e non sono sempre disponibili per le sessioni di pianificazione.
- alcuni clienti potrebbero essere più abituati ai piani di progetto tradizionali e trovare difficile partecipare a un processo di pianificazione agile.

La pianificazione agile funziona bene con team di sviluppo piccoli e stabili che possono riunirsi e discutere le storie da implementare. Tuttavia, quando i team sono grandi e/o geograficamente distribuiti, o quando la composizione del team cambia frequentemente, è praticamente impossibile coinvolgere tutti nella pianificazione collaborativa che è essenziale per la gestione agile di un progetto.

Lo sviluppo agile è un approccio molto efficace per team piccoli e coesi, richiede adattamenti e strumenti specifici per funzionare bene in contesti più grandi e complessi.

---

<sup>1</sup>un incremento del software rappresenta una porzione funzionante e valutabile del prodotto software che viene consegnata alla fine di ogni iterazione



## PROCESSI SOFTWARE

### 3.1 Definizione e Modelli

**Definizione :** Il processo software è un insieme strutturato di attività necessarie per sviluppare un sistema software. Esistono diversi processi software, ma tutti coinvolgono specifiche, progettazione e implementazione, validazione ed evoluzione del sistema per rispondere ai bisogni dei clienti.

Quando descriviamo e discutiamo di processi, di solito parliamo delle attività che li compongono, ad esempio, la specifica di un modello di dati, la progettazione dell'interfaccia utente, ecc., e dell'ordine in cui queste attività vengono eseguite. Le descrizioni dei processi possono includere anche:

- Prodotti: ovvero i risultati di un'attività del processo;
- Ruoli: che riflettono le responsabilità delle persone coinvolte nel processo;
- Pre-condizioni e post-condizioni: ossia affermazioni che sono vere prima e dopo che un'attività del processo è stata eseguita o un prodotto è stato creato.

Come già accennato, i processi software possono essere di due tipi

- **processi plan-driven** : Sono processi in cui tutte le attività vengono pianificate in anticipo e l'avanzamento del progetto viene misurato in base a questo piano iniziale. È come seguire una ricetta precisa, dove ogni passo è definito fin dall'inizio
- **processi agili** : In questi processi, la pianificazione è più flessibile e si adatta ai cambiamenti. Si lavora a iterazioni brevi (sprint) e si può modificare il piano di volta in volta in base alle nuove esigenze del cliente. È come cucinare senza una ricetta precisa, ma adattando i piatti ai gusti degli ospiti man mano che si procede.

Nella pratica La maggior parte dei progetti software combina elementi di entrambi gli approcci. A volte è necessario un piano dettagliato per le fasi iniziali, mentre in altre si può essere più flessibili.

Il processo software vive attraverso differenti formulazioni, o *modelli*, precisamente verranno trattati i seguenti

- *waterfall model*
- *incremental development*
- *integration and configuration*

Nella pratica, i sistemi di grandi dimensioni incorporano elementi da più modelli.

## Waterfall Model

È uno dei modelli di sviluppo software più antichi e tradizionali è di tipo plan driven e consiste nell'individuare e separare le distinte fasi di specifica e sviluppo. Tali fasi sono

- analisi e definizione dei requisiti
- progettazione del sistema e del software
- implementazione e collaudo
- integrazione e deployment del sistema
- mantenimento

Gli svantaggi di tale modello risiedono nella poca flessibilità e robustezza nell'accomodare richieste di cambiamento una volta che il progetto è indirizzato su una certa via. Generalmente, è necessario terminare una fase prima di passare alla successiva. Il modello a cascata divide il progetto in fasi distinte e rigide,

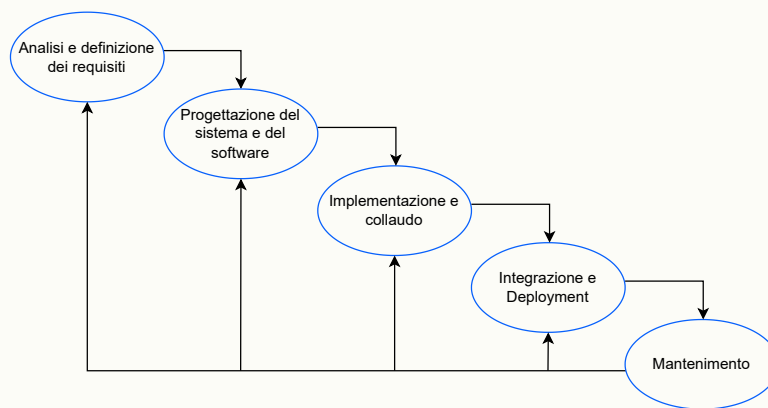


Figura 3.1: Il modello waterfall

rendendo difficile adattarsi ai cambiamenti nei requisiti del cliente.

- Limitazioni nei requisiti: Questo modello è adatto solo quando i requisiti sono ben definiti all'inizio del progetto e non si prevedono grandi modifiche durante lo sviluppo.
- Instabilità dei requisiti: Nella maggior parte dei casi, i requisiti dei sistemi informatici non sono statici, ma cambiano nel tempo.

Il modello a cascata viene spesso utilizzato in progetti di ingegneria software su larga scala, dove il sistema viene sviluppato in più sedi. In questi casi, la natura pianificata del modello a cascata aiuta a coordinare le attività tra i diversi team di sviluppo.

## Incremental Development

Tale modello di *sviluppo incrementale* punta alla riduzione dei costi necessari per accomodare le nuove richieste del cliente che insorgono a progetto già avviato. La quantità di analisi e documentazione da rifare è molto inferiore rispetto al modello a cascata : si riduce la necessità di ricominciare da capo se i requisiti cambiano, risparmiando tempo e risorse.

- *È più facile ottenere feedback dal cliente sul lavoro di sviluppo svolto* : Il cliente può vedere e provare il software in fase di sviluppo e fornire un feedback diretto, contribuendo a migliorare il prodotto finale.
- *È possibile una consegna e un deployment più rapidi del software utile per il cliente* : Il cliente può iniziare a utilizzare il software prima rispetto al modello a cascata, ottenendo così un beneficio più rapido dal prodotto.

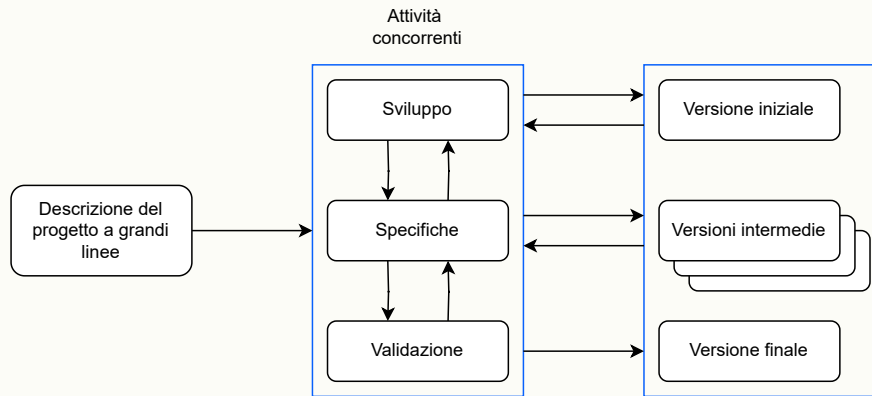


Figura 3.2: Incremental Development

Il modello comporta alcuni svantaggi

- **Mancanza di visibilità per i manager:** I manager spesso necessitano di deliverable regolari per misurare l'avanzamento del progetto. In uno sviluppo incrementale rapido, produrre documentazione dettagliata per ogni versione può risultare costoso e inefficiente. La natura iterativa dello sviluppo incrementale può rendere complesso tracciare tutti i cambiamenti apportati al sistema nel tempo.
- **Degradazione della struttura del sistema:** L'aggiunta continua di nuove funzionalità senza una adeguata manutenzione può portare al degrado della struttura del codice, rendendolo più complesso e meno manutenibile. Man mano che il sistema cresce, diventa sempre più difficile apportare modifiche senza introdurre nuovi bug o impatti negativi su altre parti del sistema.

### Integration and configuration

È un modello basato sul riutilizzo del software: I sistemi vengono costruiti combinando componenti software preesistenti o sistemi applicativi già pronti all'uso (spesso chiamati COTS - Commercial-off-the-shelf). Spesso avviene una riconfigurazione degli elementi riutilizzati, tali componenti possono essere personalizzati per adattarsi alle specifiche esigenze dell'utente.

Il riutilizzo del software è diventato il metodo più comune per sviluppare molti tipi di sistemi aziendali, verrà trattato approfonditamente in seguito. Esistono diversi tipi di **software riutilizzabile**

- Sistemi applicativi autonomi (a volte chiamati COTS) che vengono configurati per l'uso in un ambiente specifico.
- Collezioni di oggetti sviluppati a mò di pacchetto da integrare in un framework di componenti come .NET o J2EE.
- Servizi web sviluppati secondo standard e disponibili per essere integrati remotamente.

Le fasi chiave del processo sono

1. Specifica dei requisiti
2. Definizione e valutazione del software
3. Raffinamento dei requisiti
4. Configurazione del sistema
5. Adattamento e integrazione dei componenti

Vantaggi

- Costi e rischi ridotti, poiché si sviluppa meno software da zero

- Consegna e implementazione più rapida del sistema

#### Svantaggi

- sono inevitabili dei compromessi sui requisiti, quindi il sistema potrebbe non soddisfare pienamente le esigenze degli utenti
- Perdita di controllo sull'evoluzione degli elementi del sistema riutilizzati, Non avendo il controllo completo sul codice sorgente dei componenti riutilizzati

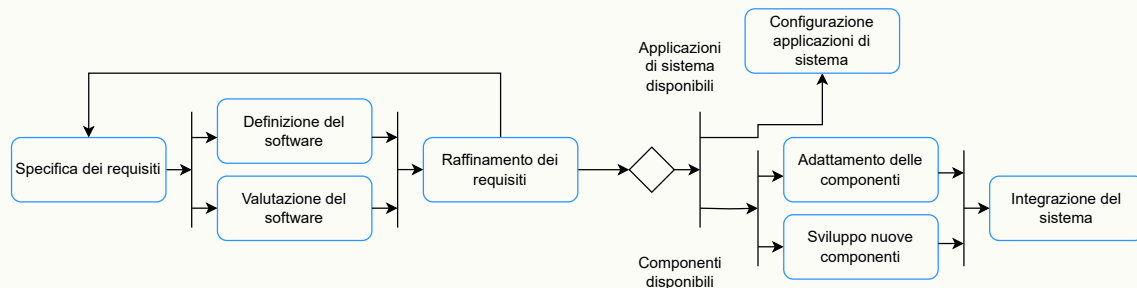


Figura 3.3: Ingegneria del software orientata al riciclo

#### Finire capitolo 2

##### The requirements

- engineering process
- design process
- stage testing generalizza, in un caso di un Planet drive e in caso agile
- software prototype pro e contro

#### iniziare il capitolo 3 agile software development

- plan base vs agile
- principles of agile methods
- the extreme programming release cycle ( un tipo di agile)