

## UNIT 9-10-11-12

### 1.1 Gestione mail con PHP

Funzione che permette l'invio delle mail tramite php.

**mail(\$mail, \$subject, \$message, \$headers);**

\$mail indirizzo a cui viene inviata la mail. per inviare la mail a più destinatari basta separare ogni indirizzo con una virgola. ATTENZIONE: usate con moderazione, evitate di fare spam

\$subject titolo della mail inviata

\$message corpo della mail

\$headers sono informazioni aggiuntive alla mail (come ad esempio l'indirizzo di chi la invia, delle copie carbone o delle copie nascoste).

La funzione mail funziona anche se non viene specificato alcun header. Se non si precisa l'indirizzo di chi la invia (il From negli headers), come indirizzo di provenienza viene visualizzato NOME\_SERVER@DOMINIO.

Ecco un banalissimo esempio:

```
mail("php@openskills.info", "Ciao", "Questo è una prova", "From: Max_Rispetto  
<max\_rispetto@openskills.info>");
```

In questo caso viene spedita una mail all'indirizzo "php@openskills.info" con titolo "Ciao" e testo "Questo è una prova" inviata da "Max\_Rispetto" con indirizzo "max\_rispetto@openskills.info"

Vediamo ora un esempio un po' più complesso, ossia come utilizzare la funzione mail in combinazione con un form di richiesta dati.

Ipotizziamo di dover gestire una newsletter. In una pagina vengono richiesti alcuni dati anagrafici di chi si vuole iscrivere e, una volta completata l'operazione, verrà inviata una mail di conferma.

Bisogna creare due pagine php, una chiamata form.php (in cui verranno immessi i dati) e una chiamata send.php (in cui si gestirà la spedizione vera e propria della mail).

Cominciamo con la pagina form.php:

```
<html>  
<body>  
<form name="form" method="post" action="send.php">  
Inserisci il nome <input type="text" name="nome">  
Inserisci la mail <input type="text" name="mail">  
</form>  
</body>  
</html>
```

Ricordarsi di far puntare il form alla pagina send.php!

Creiamo ora la pagina send.php:

```
<html>
<body>
<?php
$from="newsletter@openskills.info";
$titolo="Benvenuto!";
$testo="Ciao $nome. Ti sei iscritto alla newsletter.";
mail($mail, $titolo, $testo, "From: $from");
?>
</body>
</html>
```

Da notare: ho preparato in anticipo i campi \$titolo e \$testo, in modo da snellire la funzione mail.

Questo è tutto: con queste due paginette abbiamo un semplice sistema di invio mail. Naturalmente i più esperti possono utilizzare una sola pagina sia per fare il form che per inviare la mail.

Per maggiori informazioni:

<http://it.php.net/manual/it/function.mail.php>

## 1.1 Interazione tra database e PHP

PHP supporta da svariati anni un gran numero di database presenti sul mercato, nonostante questo però normalmente PHP si sposa con il DB relazionale MySQL.

Vedremo di seguito alcuni semplici comandi su come gestire l'interazione fra PHP e MySQL.

## 1.2 MySQL Connect, apertura di una connessione ad un DB MySQL

Apri una connessione verso un DB MySQL.

```
$db = mysql_connect([$db_host[, $db_username[, $db_password[, $newlink[, $clientflags]]]]]);
```

`$db` Restituisce un identificatore della connessione col DB (da utilizzare nelle chiamate alle successive funzioni di interrogazione) e FALSE in caso di fallimento

`$db_host` Nome host del database server e relativa porta. Default: localhost:3306

`$db_username` Nome dell'utente per il login al database. Di default corrisponde all'utente che possiede il processo corrente.

`$db_password` Password per il login al database. Di default è una stringa vuota

`$newlink` Di default, se viene effettuata una seconda chiamata a `mysql_connect` con gli stessi parametri, non viene aperta una nuova connessione ma la funzione restituisce l'ID della connessione esistente. Se questo parametro è impostato a TRUE viene aperta una nuova connessione ad ogni chiamata.

`$clientflags` Combinazione dei flag `MYSQL_CLIENT_COMPRESS` (usa la compressione dei dati),

`MYSQL_CLIENT_IGNORE_SPACE` (permette gli spazi dopo i nomi delle funzioni) o

`MYSQL_CLIENT_INTERACTIVE` (lascia trascorrere `interactive_timeout` secondi - anziché `wait_timeout` - di inattività prima di chiudere la connessione).

Compatibilità: il parametro `$newlink` è stato introdotto dal PHP 4.2.0; mentre `$clientflags` dal PHP 4.3.0

La connessione verrà chiusa al termine dell'esecuzione dello script, a meno che non venga chiusa esplicitamente da una chiamata a `mysql_close()`.

## 1.3 Chiusura connessione a MySQL, mysql\_close()

```
$success = mysql_close ($db)
```

`$success` La funzione restituisce TRUE in caso di successo e FALSE in caso di fallimento.

`$db` Identificatore della connessione da chiudere (restituito da `mysql_connect()`). Se non specificato, chiude l'ultima connessione aperta.

#### 1.4 Inviare una query da eseguire a MySQL, `mysql_query()`

Indica al database server MySQL di eseguire la query passata come argomento.

*`mysql_query($query, $link)`*

il parametro opzionale `$link` serve ad indicare quale connessione attiva utilizzare per eseguire il codice SQL presente nella stringa `$query`. se non viene indicato il parametro `$link` allora PHP tenta di utilizzare l'ultima connessione aperta, se non ce ne è nessuna verrà tentata una connessione senza nessun parametro:

`mysql_connect()`.

In caso di errore, sia legato all'impossibilità di connessione ad un db sia a errori nel codice SQL, `mysql_query()` ritornerà FALSE.

Nel caso di risultato positivo il valore ritornato sarà un identificatore di risorsa nel caso di query "descrittive" come ad esempio SELECT, SHOW etc. sarà invece un qualsiasi valore diverso da FALSE per query del tipo UPDATE, INSERT e altro. Attenzione che il valore ritornato non è necessariamente legato al numero di righe risultanti dalla query.

#### 1.5 Gestione dei dati ottenuti da una query MySQL

Una volta eseguita una query di SELECT è possibile eseguire delle istruzioni di codice per ognuno dei record risultanti.

PHP offre diverse funzioni che permettono di estrarre i valori dei campi dei record risultanti da una query di selezione. La differenza principale tra le varie possibilità offerte è data dalla tipologia di dato che viene associata al record.

`mysql_fetch_row($result)` estrae il record successivo nel record-set rappresentato da `$result` in forma di array. E' possibile cioè accedere ai valori dei singoli campi estratti utilizzando codice del tipo

```
$sql = "SELECT codice, descrizione FROM prodotti";
```

```
$result = mysql_query($sql);
```

```
$array = mysql_fetch_row($result);
```

```
print "CODICE: ".$array[0]."<br>";
```

```
print "DESCRIZIONE: ".$array[1]."<br>";
```

Verranno stampati il codice e la descrizione del primo prodotto trovato dalla query `$sql`.

Se non esiste nessun record successivo il valore ritornato da `mysql_fetch_row()` sarà FALSE.

Una modo più comodo per accedere ai campi, soprattutto in presenza di query con un gran numero di dati estratti è `mysql_fetch_array($result)`

il comportamento di fetch dei dati è simile a quello di `mysql_fetch_row()`, il valore aggiunto è dato dal fatto che sarà possibile accedere al particolare campo indicando esattamente il suo nome, vediamo lo stesso esempio:

```
$sql = "SELECT codice, descrizione FROM prodotti";
```

```
$result = mysql_query($sql);
```

```
$array = mysql_fetch_array($result);
```

```
print "CODICE: ".$array['codice']."<br>";
```

```
print "DESCRIZIONE: ".$array['descrizione']."<br>";
```

come si può notare `mysql_fetch_array()` rende il codice più leggibile senza per questo influire sulle performance del recupero di dati.

Una possibile fonte di errore può essere invece il fatto che può capitare, lavorando con query su diverse

tabelle, di dovere estrarre campi con lo stesso nome, in questo caso sarà necessario utilizzare degli alias all'interno del codice SQL perchè altrimenti il valore di `$array[$nome_campo]` sarà relativo solamente all'ultimo campo di nome `$nome_campo` presente nella query come nel caso di

```
$sql = "
SELECT prodotti.codice, prodotti.descrizione, categoria.descrizione
FROM prodotti, descrizione
WHERE descrizione.id = prodotti.id_descrizione";
$result = mysql_query($sql);
$array = mysql_fetch_array($result);
print "CODICE Prod: ".$array['codice']."<br>";
print "DESCRIZIONE Prod: ".$array['descrizione']."<br>";
print "DESCRIZIONE Cat: ".$array['descrizione']."<br>";
mentre
$sql = "
SELECT prodotti.codice, prodotti.descrizione, categoria.descrizione AS desc_cat
FROM prodotti, descrizione
WHERE descrizione.id = prodotti.id_descrizione";
$result = mysql_query($sql);
$array = mysql_fetch_array($result);
print "CODICE Prod: ".$array['codice']."<br>";
print "DESCRIZIONE Prod: ".$array['descrizione']."<br>";
print "DESCRIZIONE Cat: ".$array['desc_cat']."<br>";
ci mostra un output più consono alle nostre aspettative.
```

E' possibile utilizzare le funzioni di fetch ricorsivamente per estrarre tutti i record risultati da una query SQL:

```
$sql = "
SELECT prodotti.codice, prodotti.descrizione, categoria.descrizione AS desc_cat
FROM prodotti, descrizione
WHERE descrizione.id = prodotti.id_descrizione";
$result = mysql_query($sql);
while ($array = mysql_fetch_array($result))
{
print "CODICE Prod: ".$array['codice']." - ";
print "DESCRIZIONE Prod: ".$array['descrizione']." - ";
print "DESCRIZIONE Cat: ".$array['desc_cat']."<br><br>";
}
}
```

la condizione del ciclo while sarà TRUE fino a che sarà possibile estrarre record dalla risorsa ottenuta dall'esecuzione della query, dopo l'ultimo record la condizione diventa FALSE e il ciclo termina.

## 1.6 Verifica del numero di record riscontrati in modifica, inserimento, visualizzazione

Possiamo, senza necessariamente scorrere tutto il record-set, avere indicazioni sul numero di record che la nostra query ha estratto o modificato utilizzando la funzione `mysql_num_rows($result)`.

Essa ritorna, solamente nel caso di query di tipo SELECT, il numero di record che sono stati estratti:

```
$result = mysql_query("SELECT * FROM prodotti");
```

```
$num_prodotto = mysql_num_rows($result);
```

`$num_prodotto` conterrà il numero totale di prodotti presenti nel database.

Si noti come lo stesso risultato si potrebbe ottenere abbassandosi di livello e facendo effettuare il conteggio direttamente al database:

```
$result = mysql_query("SELECT COUNT(*) AS tot, * FROM prodotti");
```

```
$array = mysql_fetch_array($result);
```

```
$num_prodotto = $array['tot'];
```

`$num_prodotto` conterrà nuovamente il numero totale di prodotti presenti nel database.

Nel caso invece di statement SQL che modificano in qualche modo i record presenti nella nostra tabella, ad esempio con

istruzioni di INSERT, UPDATE e di DELETE, per conoscere il numero di record "toccati" si utilizza la funzione

```
mysql_affected_rows()
```

Si ponga la dovuta attenzione però ad alcune eccezioni che potrebbe indurre ad errori difficilmente debuggabili:

- nel caso in cui la query sia del tipo DELETE FROM tabella, quindi una cancellazione di tutti i record presenti in tabella senza nessun vincolo di WHERE, la funzione ritornerà sempre e comunque 0.

- quando si effettua uno statement SQL di update, il motore MySQL non "tocca" i record che già contengono lo stesso valore che si vorrebbe inserire, quindi potrebbe esserci il caso in cui `mysql_affected_rows()` non ritorni esattamente il numero di record riscontrati dalla query ma solo quelli effettivamente modificati.

## 1.7 Ottenimento ultimo id inserito a DB

PHP interfaccia l'API di MySQL `last_insert_id()` per ritornare l'indice associato ad un campo `AUTO_INCREMENT` del record che abbiamo appena inserito, la funzione che lo permette è `mysql_inserted_id($link)`.

In pratica capita spesso di dover inserire in una tabella un record di definizione di una insieme di oggetti che verranno inseriti in un'altra tabella, ad esempio una nuova categoria di prodotti ed un elenco di prodotti legati alla nuova categoria.

Per fare questo in un'unica pagina diventa necessario conoscere l'ultimo valore che è stato generato da MySQL nel campo `AUTO_INCREMENT` della tabella, consideriamo questo semplice esempio in cui supponiamo di ricevere in input la descrizione di una nuova categoria e di un prodotto ad essa associato, in pratica il risultato di una `print_r` dei valori che ci arrivano in POST sia qualcosa del tipo

Array

```
(  
  [desc_cat] => 'cibi'  
  [desc_prod] => 'pasta'  
)
```

Possiamo inserire la nuova categoria e associarvi i nuovi prodotti direttamente in un'unica pagina, tramite codice del tipo

```
$sql = "INSERT INTO categoria (id, descrizione) VALUES (NULL, '$_POST[descrizione]')"; // sia id di  
tipo AUTO_INCREMENT
```

```
$result = mysql_query($sql);
```

```
$new_id_cat = mysql_inserted_id();
```

```
$sql = "INSERT INTO prodotto (id, id_categoria, descrizione) VALUES (NULL, '$new_id_cat',  
'$_POST[descrizione]')"; // sia id di tipo AUTO_INCREMENT
```

```
$result = mysql_query($sql);
```

## 1.8 Gestione degli errori

Buona norma di programmazione è verificare e gestire sempre gli eventuali errori che possono nascere durante l'esecuzione, soprattutto quando i comandi contengono parti variabili come nel caso di query da fare eseguire al database.

In questo caso, o meglio ancora dopo ogni chiamata a funzione che interagisce con MySQL è bene mostrare, nel caso la funzione ritorni un valore `FALSE` spesso sintomo di un errore, le cause dell'errore.

Per fare ciò si utilizza la funzione `mysql_error($link)` al solito non specificando l'identificativo della connessione `$link` verrà considerata l'ultima connessione aperta come attiva.

In questo modo possiamo delineare una forma standard per l'esecuzione di query in questo modo:

```
$result = mysql_query($sql)
```

```
or die("<br>Invalid query: $query\n<br>\n" . mysql_error());
```

In questo modo nel caso ci siano errori ritornati dal motore MySQL lo script si interromperà e ci verrà mostrato l'errore stesso e la query responsabile.

## PHP Object Oriented

La programmazione ad oggetti in PHP4 può risultare limitata, soprattutto per chi ha avuto modo di sperimentare questo paradigma di programmazione in altri ambiti come ad esempio Java. Naturalmente la teoria di questo paradigma rimane la stessa, ci sono però numerosi aspetti da valutare con attenzione nell'implementazione della teoria da parte di PHP 4.

### 8.1 Classe: proprietà e metodi

L'elemento fondamentale della OOP è la **classe**, una classe contiene la definizione di dati (in terminologia Object Oriented: **proprietà**) e di funzioni (**metodi**) che condividono alcune caratteristiche tali da suggerirne l'**incapsulamento** in una singola struttura, la classe appunto.

La definizione di una classe utilizza l'istruzione **class**:

```
class la_mia_prima_classe
{
    var $variabile;

    function la_mia_prima_classe ($in)
    {
        $this->variabile = $in;
    }
}
```

Le proprietà vengono definite tramite l'istruzione **var** seguite dal nome della variabile, successivamente viene definito il metodo *la\_mia\_prima\_classe*, si noti come questo metodo abbia lo stesso nome della classe, questo di fatto lo identifica come **costruttore** della classe stessa.

Un costruttore di una classe viene invocato automaticamente dall'interprete PHP quando la classe viene **istanziata**, quando cioè viene creato un **oggetto** di quella classe.

All'interno del costruttore abbiamo assegnato il valore della variabile *\$in* alla proprietà *\$variabile* tramite il costrutto

```
$this->variabile = $in;
```

la variabile **\$this** si riferisce all'oggetto corrente e può essere utilizzato solo all'interno di un metodo di una classe.

Sempre all'interno della definizione di una classe è possibile fare riferimento ai metodi utilizzando la sintassi

```
$this->metodo();
```

Una classe non può essere utilizzata direttamente dato che in effetti non è niente altro che una **dichiarazione** di proprietà e metodi che definiscono quindi un particolare tipo di dato. Quello che ci permette di utilizzare queste definizioni è la creazione di un **oggetto** come **istanza** della particolare classe:

```
$oggetto = new la_mia_prima_classe("qualcosa in input");
print $obj->variabile;
```

L'operatore **new** crea una nuova istanza di *la\_mia\_prima\_classe* e la assegna a *\$oggetto*.

Il codice quindi produrrà l'output:

qualcosa in input

I metodi della classe definita possono essere utilizzati sia **dinamicamente** cioè istanziando un oggetto (è il caso dell'esempio precedente) sia in modo **statico** cioè utilizzando la classe come un **namespace**. Vediamo un esempio:

```
class seconda_classe
{
    var $variabile;

    function seconda_classe ($in)
    {
        $this->$variabile = $in;
    }

    function moltiplica ($num1, $num2)
    {
        return $num1*$num2;
    }
}

print seconda_classe::moltiplica(1234,4321);
```

utilizzando l'operatore :: possiamo utilizzare staticamente il metodo (la funzione) *moltiplica* che abbiamo definito all'interno della classe.

Uno dei problemi che si riscontra più spesso utilizzando gli oggetti in PHP4 è dovuto al fatto che quando si passa un oggetto come argomento di una funzione, esso viene trattato esattamente come ogni altro tipo di dato, viene cioè passato **per valore**: l'interprete crea una copia dell'oggetto e la passa alla funzione, tale copia verrà distrutta al termine dell'esecuzione della funzione stessa.

## 8.2 Ereditarietà

Una delle prerogative della OOP è l' **ereditarietà**: la possibilità cioè di definire una **classe figlio** che acquisisca dalla **classe padre** le proprietà e i metodi avendo la possibilità di aggiungerne di nuovi o di ridefinire quelli esistenti. In PHP4 l'ereditarietà viene attuata tramite l'**estensione** di una classe:

```
class padre
{
    var $var_padre;

    function padre ($in)
    {
        $this->$variabile = $in;
    }

    function moltiplica ($num)
    {
```

```
        return $this->var_padre * $num;
    }
}

class figlio extends padre
{
    var $var_figlio;

    function figlio ($in)
    {
        $this->var_figlio = $in;
        $this->var_padre = $in + 100;
    }
}

$oggetto = new figlio (2);
print $oggetto->moltiplica(4);
```

come si può notare anche istanziando un oggetto di classe *figlio* possiamo utilizzare direttamente il metodo *moltiplica* che avevamo definito nella classe *padre*.

Quando si renda necessario *passare* degli oggetti tra diverse pagine PHP è possibile **serializzare** gli oggetti cioè indicare all'interprete PHP di esportare le proprietà attuali dell'oggetto, così da poterle passare come parametri GET o POST alla pagina successiva che potrà così "ricostruire" l'oggetto. Le funzioni che possiamo utilizzare sono

```
serialize($oggetto)
```

e

```
unserialize($oggetto)
```

### 8.3 PHP 5, alcune novità

PHP5 introduce una vasta gamma di novità, sia nella logica della programmazione che nelle estensioni utilizzate:

-**Nuovo modello ad oggetti:** grazie alla nuova infrastruttura il PHP5 fornisce tutti gli strumenti per la programmazione ad oggetti. Le keyword **public**, **private**, **protected** per definire la visibilità delle proprietà e dei metodi, un costruttore standard **\_\_construct()**, un distruttore standard **\_\_destruct()**, introduzione delle interfacce, ecc.

-**Metodi e proprietà statici:** Il modello ad oggetti utilizza anche la proprietà **static** per definire sia metodi che proprietà di un oggetto come statici e quindi invocabili senza un'istanza della classe in cui sono definiti.

-Metodo **\_\_autoload()** invocato ogni qual volta si tenta d'istanziare una classe non ancora caricata nel flusso corrente del programma.

-Paradigma **try-throw-catch**: in tutti i linguaggi evoluti esiste questo altrettanto evoluto paradigma di gestione specializzata di errori ed eccezioni (essendo anchesse oggetti diventa possibile effettuare una buona gestione degli errori).

-**SPL**: Standardizzazione delle librerie tramite l'utilizzo della libreria *Standard PHP Library* che implementa anche una nuova gestione degli iteratori (utilizzabili anche tramite riferimenti) per eliminare alcune limitazioni del PHP4.

-**Argomenti delle funzioni**: a differenza del PHP4, in PHP5 e' possibile definire un valore di default anche per gli argomenti delle funzioni passati come riferimenti.

-**Standardizzazione XML**: Migrazione verso le librerie libxml utilizzate dal parser DOM viste le notevoli prestazioni. Introdotto anche il parser con organizzazione ad albero SimpleXML.

-**Nuovissima estensione MySQLi** per sfruttare appieno le funzionalità di MySQL >= 4.1 (nuovo protocollo, prepared statement, ecc).

-**SQLite embedded** in PHP5, senza la necessita' di un DBMS ma con ottime prestazioni per piccoli-medi progetti.

-**Tidy**, potente estensione per moltissime operazioni su file HTML (parsing, pulizia, riparazione, ecc.).

-**Introspezione**, ovvero la possibilità di ottenere informazioni riguardanti lo script in esecuzione.

-Nuovo supporto per gli **ambienti multi-threaded**.

## Sicurezza

Nella seguente unit verranno esposti brevemente alcuni punti salienti riguardanti la sicurezza delle applicazioni WEB sviluppate con il linguaggio PHP, dando spazio sia alle problematiche di configurazione dell'interprete che a quelle in carico al programmatore. Questa unit offre una serie di strumenti per aumentare il livello di sicurezza di un'applicazione WEB tenendo però sempre presente che tutte le problematiche hanno come unica porta d'ingresso i parametri in ingresso. nonostante inizialmente possa sembrare superfluo o argomento di poca importanza, per un'applicazione WEB la sicurezza è cruciale visto il grande quantitativo di utenti che spesso queste applicazioni servono.

### 12.1 Configurazione

Il file di configurazione "php.ini" prevede diversi accorgimenti per semplificare, migliorare e mantenere un livello minimo di sicurezza per l'intero sistema, indipendentemente dalle applicazioni presenti:

- register\_globals: permette di abilitare o meno l'automatizzazione di generazioni di variabili sulla base dei parametri presenti in \$\_GET o \$\_POST
- magic\_quotes\_\*: serie di variabili per abilitare o meno il filtraggio automatico dei parametri in ingresso
- safe\_mode: permette di abilitare o meno una particolare configurazione di PHP restringendone molto le funzionalità, solitamente usato da società che offrono hosting in grande quantità (non trattato in questa unit).

Semplice programma di esempio da valutare da parte degli alunni con register\_globals attivato e disattivato:

```
<?php
```

```
$utente[1] = "Admin";
```

```
$utente[2] = "Mario rossi";
```

```
$utente[3] = "Fabio Verdi";
```

```
$permessi['admin'] = "Amministratore";
```

```
$permessi['user'] = "Accesso parziale ai dati con permessi di modifica";
```

```
$permessi['user'] = "Accesso parziale ai dati con permessi di sola visualizzazione";
```

```
    echo "Ciao".$utente[$utente_attuale].". Hai permessi di: ".$permessi[$permessi_attuali];
```

```
?>
```

Valutare quali problemi insorgono al modificarsi del parametro do configurazione.

Effettuare lo stesso tipo di lavoro per il parametro magic\_quotes.

### 12.2 Programmazione

Per quanto riguarda la programmazione esistono diversi strumenti e librerie per aumentare il livello di sicurezza. Mentre questi strumenti possono cambiare nel tempo esistono una serie di regole che ogni programmatore dovrebbe sempre tener presente durante lo sviluppo di un'applicazione (non necessariamente WEB o scritta in linguaggio PHP).

- 1) Non avere mai la certezza che i parametri in ingresso siano sicuri (dimensioni, contenuto, ecc), per ciò abilitare SEMPRE un filtro prima che questi dati vengano effettivamente utilizzati o elaborati.
- 2) Non avere mai la certezza che i parametri in ingresso, una volta filtrati per l'elaborazione, siano adatti ad essere utilizzati come parametri per l'accesso a basi di dati. Per ciò utilizzare un filtro appositamente studiato.
- 3) In caso di applicazioni con accesso utente, permessi, ecc, verificare periodicamente la personalità di ogni utilizzatore. Evitare comunque che un'utente possa assegnarsi autonomamente una personalità diversa dalla propria.
- 4) Evitare di mostrare qualunque informazione di sistema non strettamente necessaria (parametri di accesso, nomi campi database, nomi parametri in ingresso, ecc...).
- 5) Studiare una metodologia per riconoscere utenti che possono accedere al sistema da utenti indesiderati o pericolosi.
- 6) Avere un'efficace gestione degli errori che segnali ad un responsabile eventuali tentativi d'utilizzo indesiderato del sistema.
- 7) Conoscere i limiti degli strumenti utilizzati e, per quanto possibile, evitarli alla fonte senza però dimenticare di studiare metodologie per superare tali problematiche.