

Costruire Software Sicuro dalle fondamenta

Net System Security 2007

Antonio Parata

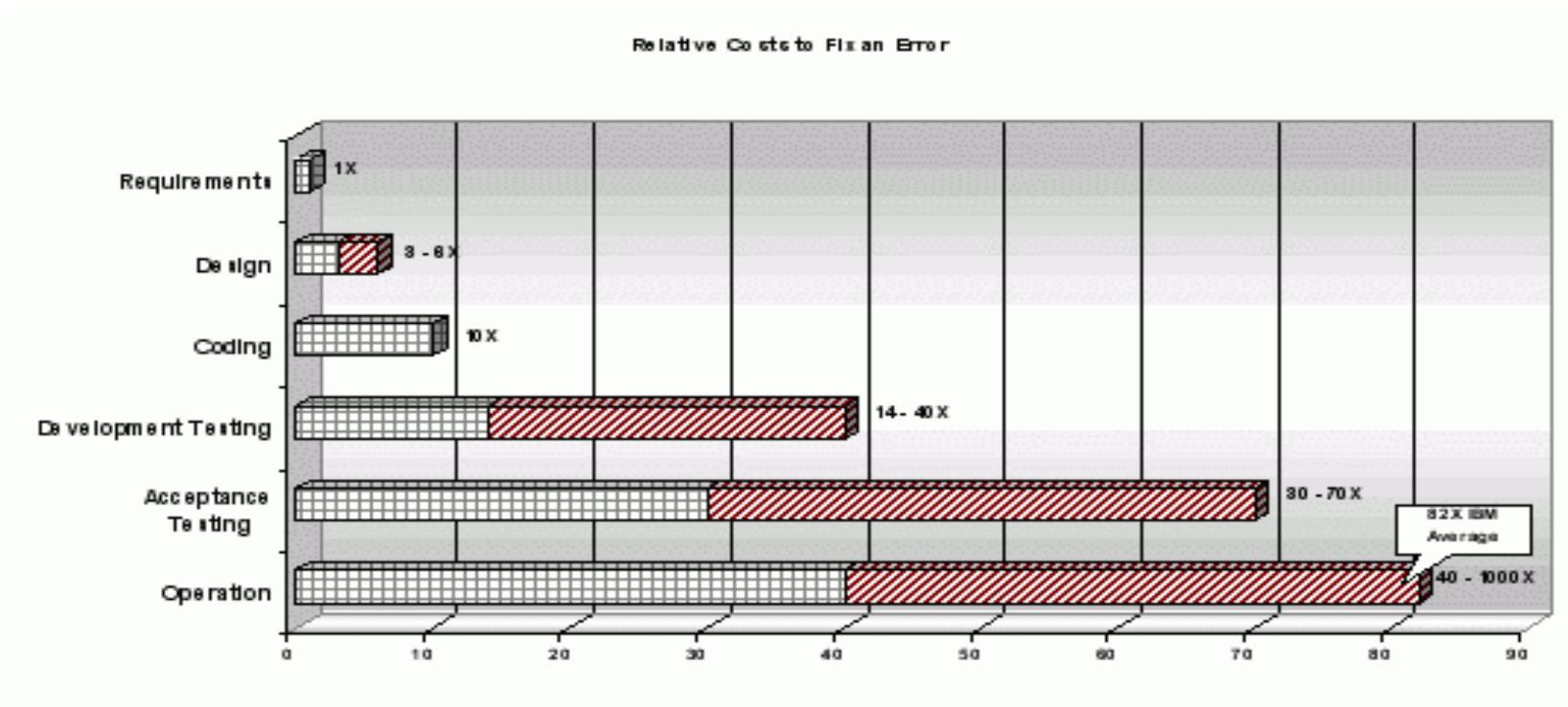
antonio.parata@emaze.net

Indice

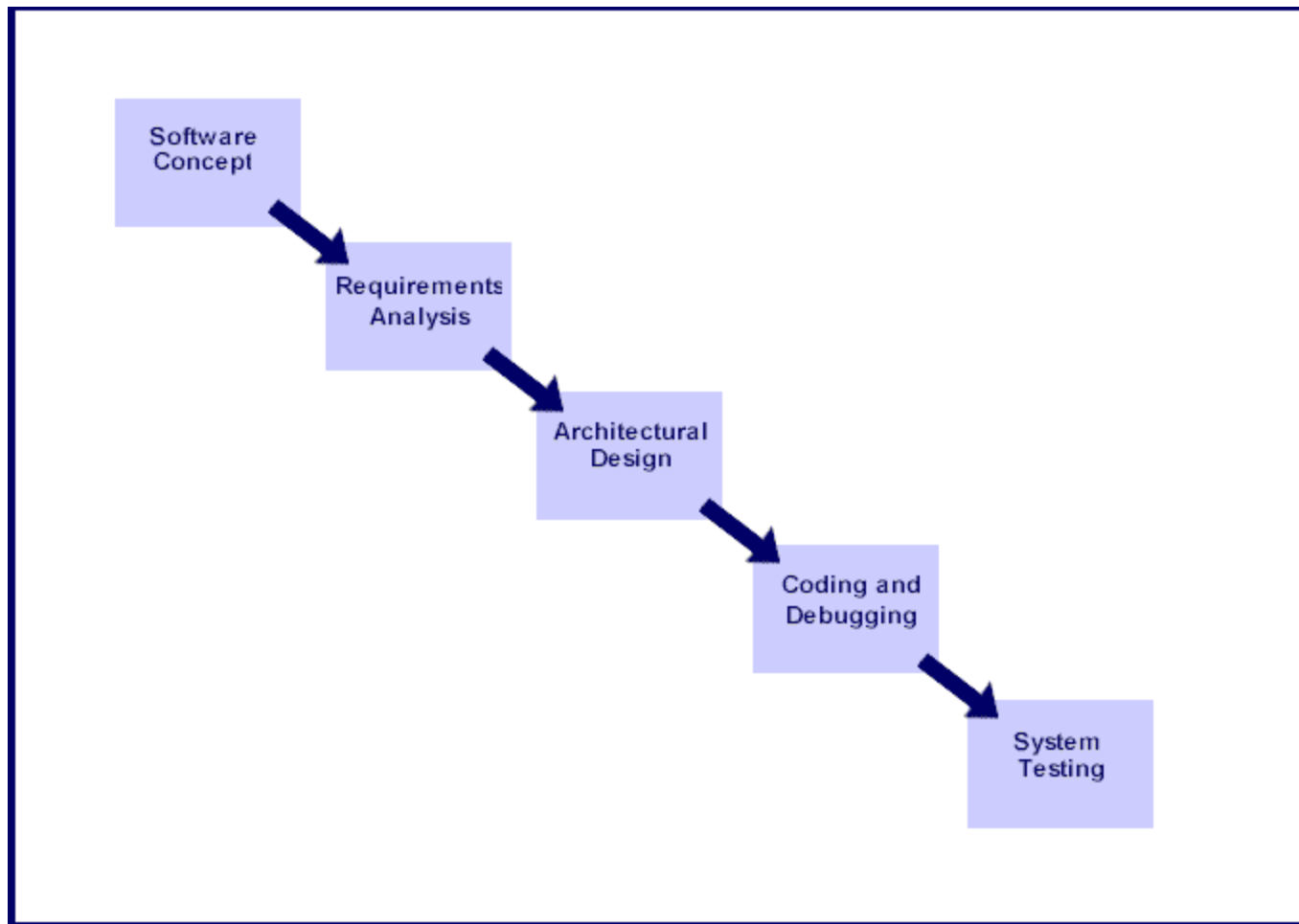
- Introduzione
- Ciclo di sviluppo software
- Attività Security Related
 - Attack Patterns
 - Abuse Cases
 - Security Requirements
 - Security Patterns
 - Threat Modeling
 - Coding Security Principles
 - Security Code Review
 - Security Testing
 - Penetration Testing
- Visione Globale
- Conclusioni e Sviluppi Futuri

Introduzione

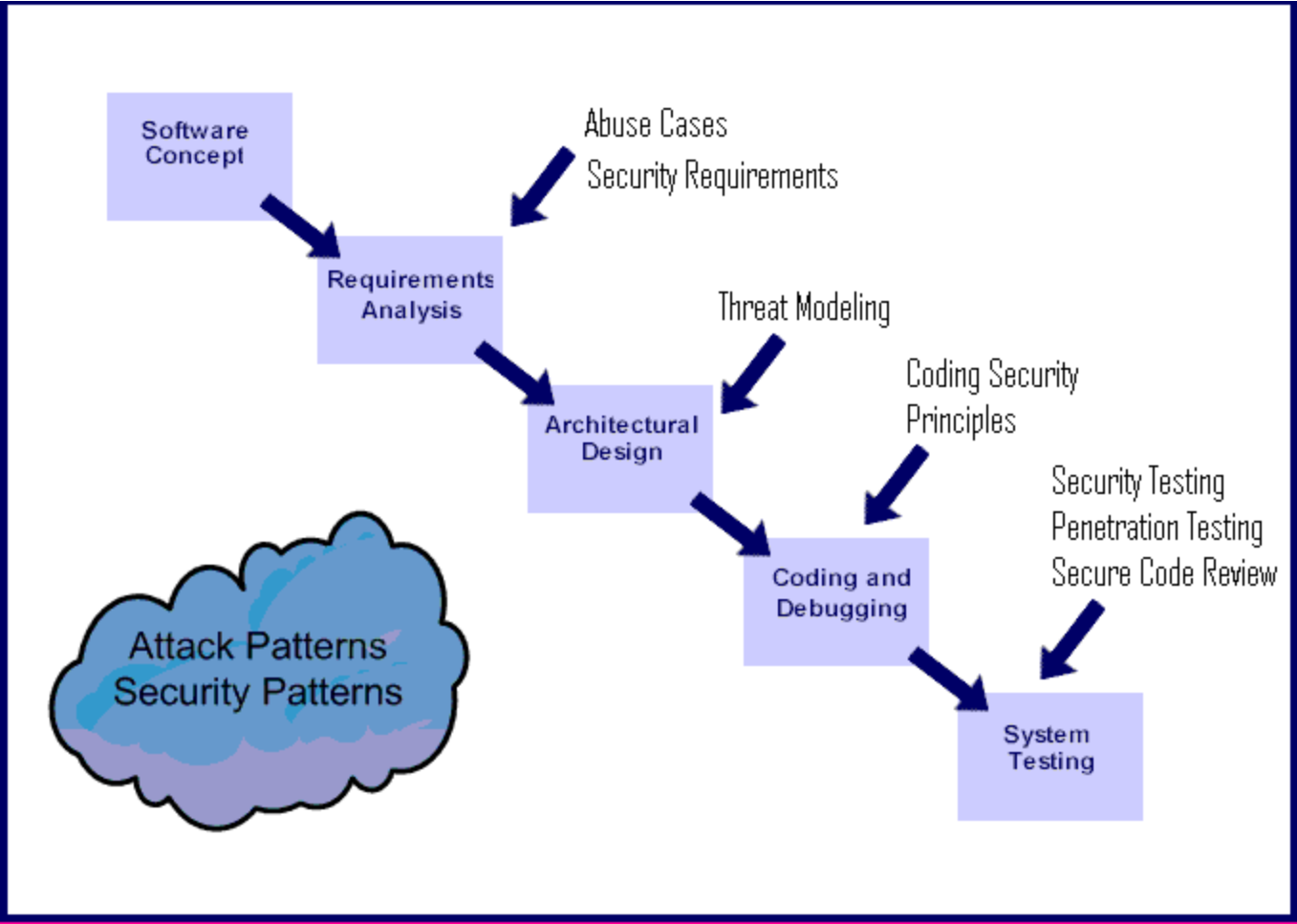
Il costo per rimediare ad una vulnerabilità aumenta con il progredire delle fase.



Ciclo di sviluppo software



Ciclo di sviluppo software (Security Oriented)



Attack Patterns

- Concetto derivante dai Design Patterns.
- Raggruppano le caratteristiche che hanno in comune alcune vulnerabilità.
- Utilizzati per **individuare** quali sono le **maggiori aree a rischio** nella propria applicazione.
 - Vengono utilizzati non solo nella stesura dei requisiti, ma anche nelle successive fasi del SDLC
- Non è necessario modificare il SDLC.
 - Vengono utilizzati come Knowledge Base aziendale
 - Necessitano di uno sforzo iniziale di definizione e catalogazione

Attack Patterns

Buffer Overflow Attack Pattern

Goal: Sfruttare vulnerabilità da buffer overflow per eseguire un codice malevolo sul sistema target.

Precondizioni: Un attaccante deve essere in grado di poter eseguire programmi sul sistema target.

Attacco:

AND

1. Identificare programmi eseguibili privilegiati sul sistema target che siano suscettibili a buffer overflow.
2. Creare un vettore d'attacco che conterrà il codice malevolo che si vuole far eseguire.
3. Creare il codice malevolo che si vuole far eseguire (anche detto payload).
4. Eseguire il programma in modo che prenda in input il vettore d'attacco creato al punto 2.

Postcondizioni: il payload viene eseguito sul sistema target.



Abuse Cases

- Anche detti *Misuse Cases*, vengono utilizzati per aiutare il progettista a **pensare come un attaccante**.
- Generalmente creati tramite sessioni di Brainstorming.
- A differenza dei requisiti, gli Abuse Cases dovranno avere un corrispettivo **piano di mitigazione all'interno dell'applicazione**.
- L'integrazione all'interno del SDLC avviene nella fase dei requisiti.
 - Viene aggiunta un'ulteriore fase in parallelo a quelle esistenti.
- Esempio:
 - *“Un utente malintenzionato può aggirare le restrizioni sull'input che vengono applicate sul client, inviando una richiesta appositamente forgiata utilizzando tool esterni.”*



Security Requirements

- Utilizzo ortogonale a quello degli abuse cases.
 - Il loro compito è specificare in che modo si intende porre rimedio in fase di design/sviluppo ai possibili attacchi identificati tramite gli abuse cases.
- La loro integrazione all'interno del SDLC è nella fase di definizione dei requisiti.
 - È una fase parallela a quella di definizione degli abus cases, ma è fondamentale tenere le due fasi separate.
- Esempio:
 - *“Tutte le richieste ricevute dal server vengono convogliate verso un unico punto di validazione dell'input, in cui vengono applicati una serie di filtri con tipologia white list.”*

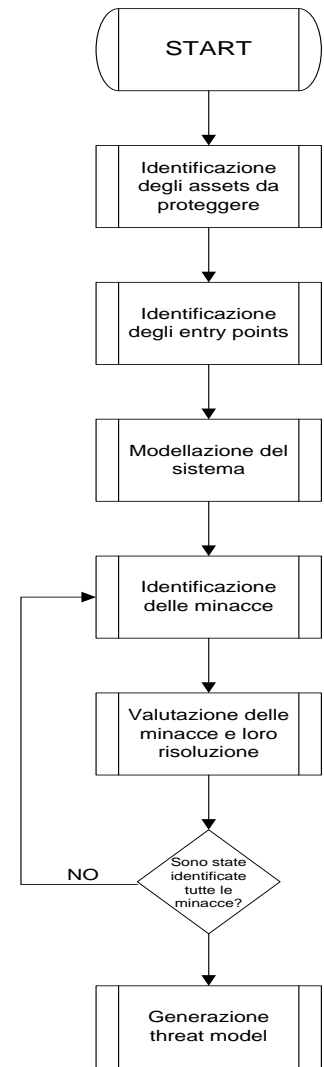
Security Patterns

- Utilizzati nella fase di progettazione, per costruire un design “sicuro by default”.
- Patterns-Driven Security Design:
 - Creare un’architettura (design di alto livello) candidata dell’applicazione
 - Eseguire un’analisi del rischio sull’architettura prescelta
 - Progettare l’applicazione utilizzando i security design conosciuti, in modo da soddisfare i requisiti di sicurezza dell’applicazione.
 - Se non esiste un security pattern adatto, modificare il design dell’applicazione in modo che rispetti il requisito. In seguito aggiungere il nuovo pattern di design all’elenco posseduto.
- Come gli attack pattern non vi è una vera e propria integrazione all’interno del SDLC.



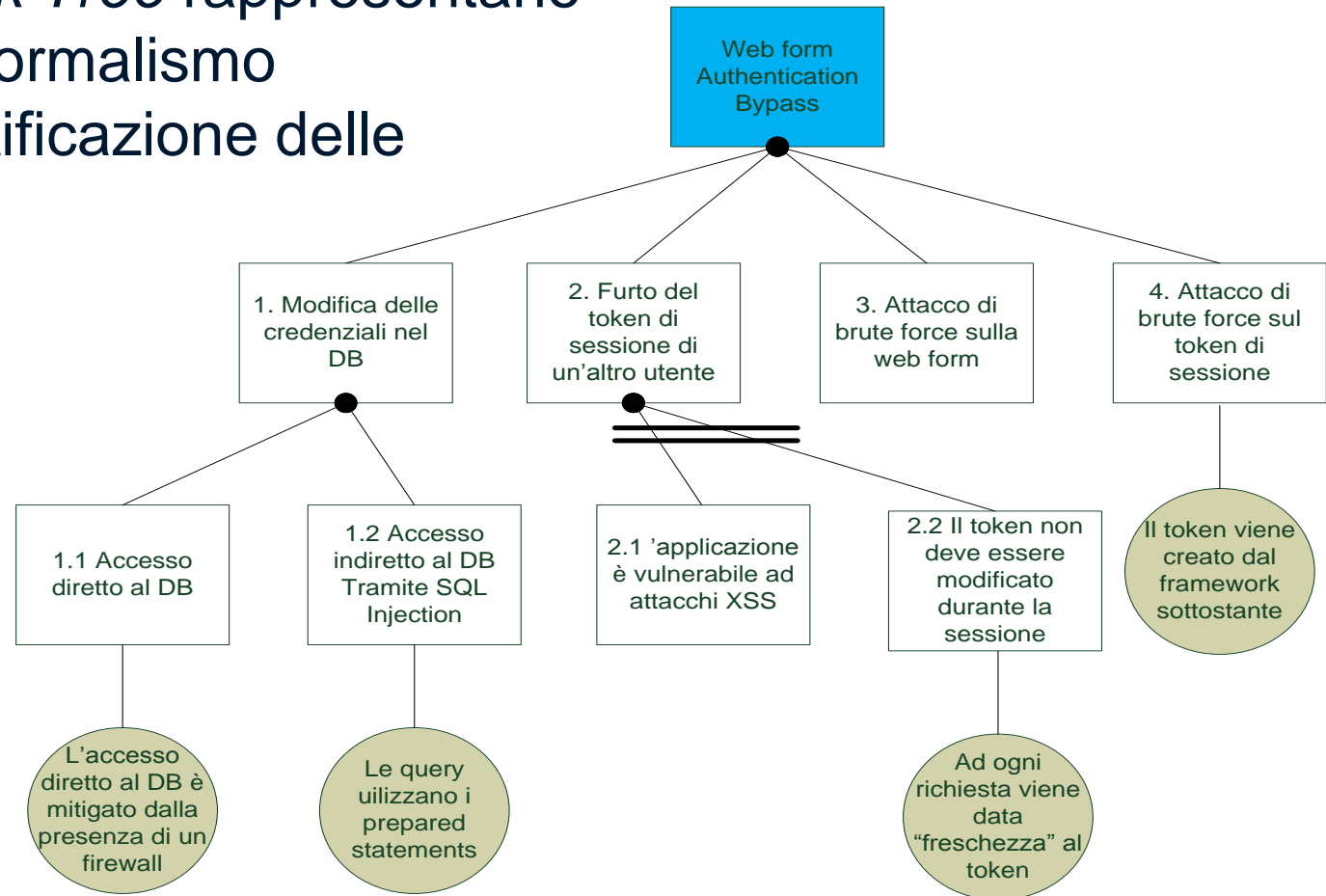
Threat Modeling

- Tra le fasi più importanti per la produzione di software sicuro.
- Lo scopo è quello di **identificare, valutare e porre rimedio alle minacce identificate**.
 - Esistono molteplici definizioni
- Utilizza
 - Attack tree
 - Data flow diagram
 - Risk Evaluation model
- Attività di tipo iterativo da affiancare in parallelo alla fase di design.
- Produce un documento, il Threat Model, contenente le minacce e le vulnerabilità identificate (tra le altre cose).



Threat Modeling

- Gli *Attack Tree* rappresentano un utile formalismo nell'identificazione delle minacce.



Coding Security Principles

- Utilizzo di librerie e funzioni “safe”
 - Safe String handling
 - Secure random number generator
 - Secure unique file creation
- Utilizzo di Secure Coding Convention
 - Controllare sempre il valore di ritorno
 - Utilizzo di header file che sollevano un errore quando si utilizza una funzione considerata “unsafe”
- Eliminare il codice obsoleto o mai raggiungibile (dead code)
- Riutilizzo del codice ove possibile

Security Code Review

- Tra le fasi più efficaci per l'identificazione di errori nel codice.
 - Sessioni da massimo quattro ore con un intervallo a metà sessione
- Varie strategie di esecuzione
 - Bottom Up
 - Top Down
 - Ibrida
- Attività “*Brain Intensive*”
 - Un buon tool di analisi statica del codice sorgente può far diminuire drasticamente i tempi di analisi.
- Attività da includere nella fase di testing e analisi.

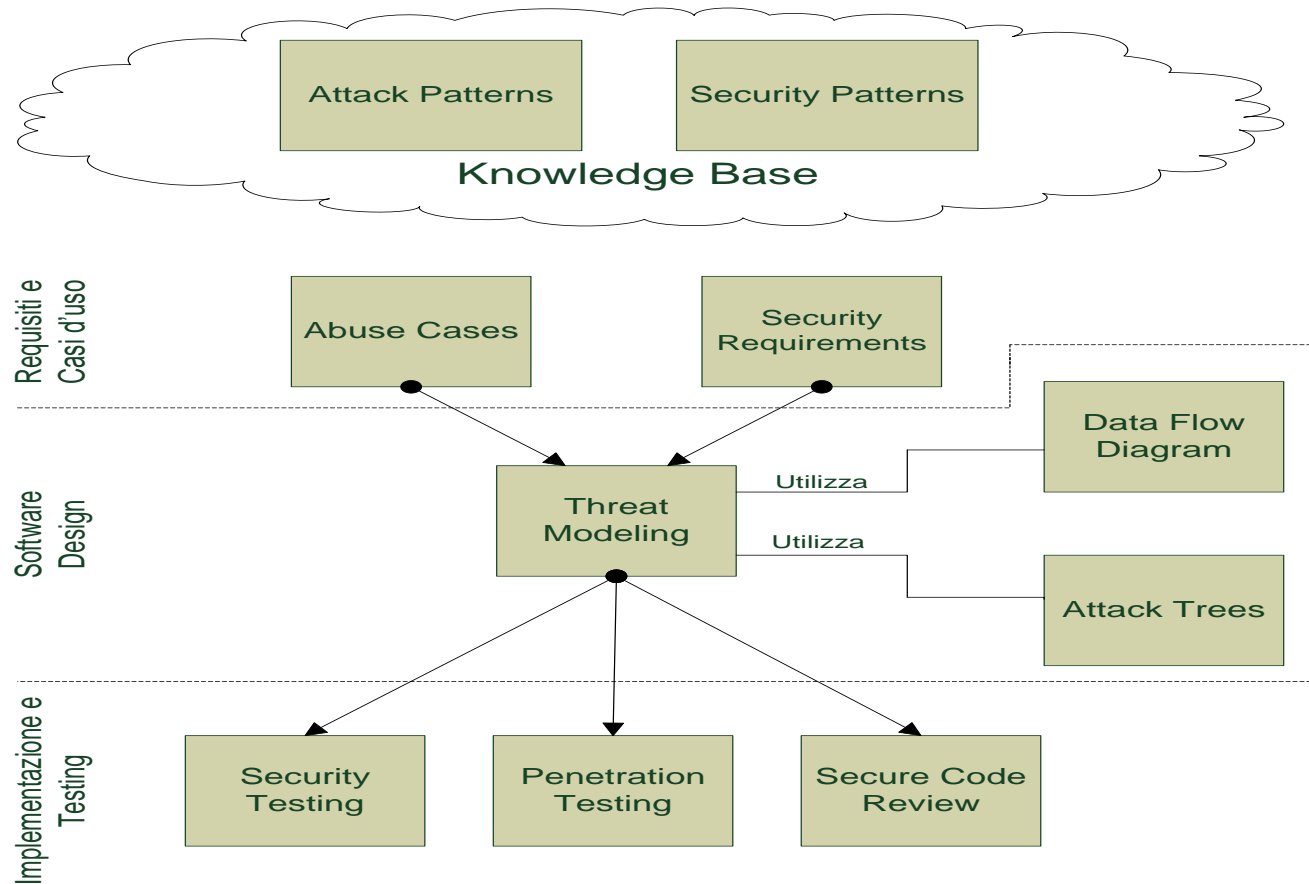
Security Testing

- Il testing è l'attività **ortogonale alla fase di analisi**.
 - A volte è più facile identificare degli errori con dei semplici test invece di analizzare il codice
- Viene effettuato utilizzando tutta la conoscenza posseduta (no Black Box)
 - Non dare nulla per scontato
- Necessità di una fase iniziale di configurazione dell'ambiente (Scaffolding)
 - Creazione degli eventuali Driver
 - Creazione degli eventuali Stub
 - Creazione degli oracoli
- La sua integrazione all'interno del'SDLC si basa sulla modifica della normale attività di testing.

Penetration Testing

- Tra le attività più in voga per mettere alla prova la sicurezza dei propri applicativi.
- Tipicamente eseguita in fasi inoltrate del SDLC
- Esistono varie tipologie di testing che si differenziano sul livello di conoscenza posseduto
 - Black Box
 - White Box
 - Gray Box (anche detto Glass Box)
- In un ciclo di sviluppo security oriented, la finalità principale dovrebbe essere di **individuare vulnerabilità nel Deployment dell'applicazione.**

Visione globale



Conclusioni e Sviluppi Futuri

- Correzione delle vulnerabilità già dalle prime fasi del SDLC.
- Esistono svariate attività che possono essere incluse nel SDLC per aumentare la sicurezza dell'applicazione già dalle prime fasi.
 - Includere tali attività in modo incrementale all'interno del proprio SDLC.
- In futuro si prevede:
 - Un'ulteriore evoluzione per le attività presenti nelle prime fasi del SDLC.
 - Comparsa di tool per automatizzare o comunque facilitare buona parte delle attività presentate.
 - Comparsa di (ulteriori) metodologie per quantificare (qualitativamente o quantitativamente) la sicurezza del proprio software.

