Facultad de Ingeniería
Universidad Nacional de Mar del Plata

Curso de PHP Avanzado

Programación de sitios web dinámicos
PHP - JavaScript - MySQL

Primera Parte

Coordinación: **Carlos Rico**

Autor: **Leonardo Tadei**

Marzo 2013

Software:

Existen varias definiciones similares aceptadas para software, pero probablemente la más formal sea la siguiente:

Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación.

Extraído del estándar 729 del IEEE

Considerando esta definición, el concepto de software va más allá de los programas de computación en sus distintos estados: código fuente, binario o ejecutable; también su documentación, los datos a procesar e incluso la información de usuario forman parte del software: es decir, abarca todo lo intangible, todo lo "no físico" relacionado.

Ahora bien, para crear un software, existen varias técnicas y procedimientos que son exclusivos de esta actividad, y que nos ayudan a llegar el resultado final, es decir, nuestro producto de software, independientemente de que sea un software para la web, para PCs de escritorio, para teléfonos móviles, para servidores, centrales telefónicas o automóviles.

Siguiendo estas pautas, no se garantiza que el software funcione bien o que resuelva el problema para el cual había sido creado, pero no seguir estas pautas, reduce mucho las posibilidades de terminar bien, es decir, con un software correcto, completo, sin fallos, entregado a tiempo y dentro del presupuesto estimado.

Proceso de creación del software

Se define como Proceso al conjunto ordenado de pasos a seguir para llegar a la solución de un problema u obtención de un producto, en este caso particular, para lograr la obtención de un producto software que resuelva un problema.

El proceso de creación de software puede llegar a ser muy complejo, dependiendo de su porte, características y criticidad del mismo. Por ejemplo la creación de un sistema operativo es una tarea que requiere proyecto, gestión, numerosos recursos y todo un equipo disciplinado de trabajo. En el otro extremo, si se trata de un sencillo programa (por ejemplo, la resolución de una ecuación de segundo orden), éste puede ser realizado por un solo programador (incluso aficionado) fácilmente. Es así que normalmente se dividen en tres categorías según su tamaño (líneas de código) o costo: de Pequeño, Mediano y Gran porte.

Se estima que, del total de proyectos software emprendidos, un 28% fracasan, un 46% caen en severas modificaciones que lo retrasan y un 26% son totalmente exitosos.

Cuando un proyecto fracasa, rara vez es debido a fallas técnicas, la principal causa de fallos y fracasos es la falta de aplicación de una buena metodología o proceso de desarrollo. Entre otras, una fuerte tendencia, desde hace pocas décadas, es mejorar las metodologías o procesos de desarrollo, o crear nuevas y concientizar a los profesionales en su utilización adecuada.

Es común para el desarrollo de software de mediano porte que los equipos humanos involucrados apliquen sus propias metodologías, normalmente un híbrido de los procesos habituales y a veces con criterios propios.

El proceso de desarrollo puede involucrar numerosas y variadas tareas, desde lo administrativo, pasando por lo técnico y hasta la gestión y el gerenciamiento. Pero casi rigurosamente siempre se cumplen ciertas etapas mínimas; las que se pueden resumir como sigue:

* Captura, Elicitación , Especificación y Análisis de Requerimientos (SRS)
* Diseño
* Codificación
* Pruebas (unitarias y de integración)
* Instalación y paso a Producción
* Mantenimiento

En las anteriores etapas pueden variar ligeramente sus nombres, o ser más globales, o contrariamente, ser

más refinadas.

Pero independientemente del tamaño del software, la primer etapa de Elicitación de Requerimientos es necesaria, ya que no podremos escribir un software son saber antes qué software debemos escribir, y daremos aquí unas breves pautas aplicadas a proyectos web típicos que usaremos durante todo este curso.

Gestión de Proyectos de Software

Una de las principales incertidumbres a la hora de querer trabajar construyendo aplicaciones web y software en general la produce el no saber por dónde empezar. En esta sección vamos a tratar de esclarecer aunque superficialmente cómo abordar un proyecto de software.

Para esto, vamos a comenzar con la definición de Proyecto: "*Se puede definir PROYECTO como un conjunto de actividades interdependientes orientadas a un fin específico, con una duración predeterminada*".

Esto nos divide el problema en 3 partes que podremos abordar por separado:

determinar la finalidad del proyecto: parece obvio, pero la mayoría de los proyectos fracasan porque no está claramente determinado cuál es el resultado final al que se debe llegar. Esto convierte a la etapa de especificación de requerimientos en un paso muy importante, ya que el éxito o el fracaso de un proyecto se determina viendo si cumplió o no con su objetivo, para lo cual necesitamos primero conocer nosotros y el interesado en el software perfectamente ese objetivo. El requerimiento debe volcarse por escrito, y hasta se suele firmar por las partes a modo de contrato.

determinar cuales serán las actividades independientes: estas actividades pueden variar de proyecto en proyecto, pero las que siempre estarán presentes serán a) saber qué hay que hacer (requerimiento), b) armar un modelo de datos válido, c) diseñar los procesos que manipularán los datos, d) diseñar las interfaces con que el usuario usará los procesos y visualizará los datos (en esta etapa pueden surgir nuevos procesos "de interfaz"), e) probar que el sistema cumpla con el requerimiento determinado al principio.

calcular la duración: una vez determinadas las actividades independientes recién podrá estimarse la duración del proyecto. En este cálculo los informáticos tendemos a ser muy optimistas y consideramos menos tiempo del que realmente necesitamos, no por no saber calcular el tiempo en hacer una actividad, sino por no considerar factores ajenos que nos impiden dedicar todo nuestro tiempo al desarrollo. Es una buena política sumar a los proyectos entre un 10 y un 25% del tiempo calculado optimistamente hasta ser capaces de incluir estas demoras en el propio cálculo. Naturalmente que el cálculo del tiempo es una parte importante del cálculo de costos de un sistema, junto con su complejidad, arquitectura, restricciones de hardware, etc.

Especificación de Requerimientos de Software (SRS)

Una especificación de requerimientos del software es una descripción completa del comportamiento del sistema a desarrollar. Incluye un conjunto de casos de uso que describen todas las interacciones que se prevén que los usuarios tendrán con el software. También contiene requerimientos no funcionales (o suplementarios). Los requerimientos no funcionales son aquellos que imponen restricciones al diseño o funcionamiento del sistema (tal como requerimientos de funcionamiento, estándares de calidad, o requerimientos del diseño).

Las estrategias recomendadas para la especificación de los requerimientos del software están descritas por IEEE 830-1998. Este estándar describe la estructuras posibles, contenido deseable, y calidades de una especificación de requerimientos del software.

Los requerimientos pueden ser de tres tipos:

* Requerimientos Funcionales: son los que el usuario necesita que efectúe el software. Ej: el sistema debe emitir un comprobante al registrar la entrega de mercadería.
* Requerimientos No funcionales: son los "recursos" para que trabaje el sistema de información (redes, tecnología).Ej: el soporte de almacenamiento a usar debe ser MySQL
* Requerimientos Empresariales u Organizacionales: son el marco contextual en el cual se implantará el sistema para conseguir un objetivo macro. Ej: abaratar costos de expedición.

Una correcta Especificación de Requerimientos de Software produce requerimientos:

- organizados
- medibles
- comprobables
- expresados con lenguaje mínimo
- sin ambigüedades o contradicciones
- ranqueables
- homogéneos

La especificación de requerimientos es un "documento vivo" que se usa durante todo el ciclo de desarrollo del sistema y muchas veces cambia y se actualiza, por tanto para su mejor uso, debe tener una portada con el nombre del proyecto, los autores y la versión de la especificación.

Definición de requerimientos funcionales:

La definición de requerimientos funcionales consiste en la caracterización de lo que el sistema (artefacto) debe hacer. Debe hacerse referencia exclusivamente a lo que el sistema debe hacer, y nunca a lo que el usuario hará con el sistema: estamos caracterizando al sistema, no a quienes lo usarán.

Cada requerimiento debe estar numerado, de forma tal que pueda hacerse referencia a él desde otro requerimiento, y ordenado de forma tal que cuando un requerimiento mencione a otro, este otro ya haya sido definido.

Cada requerimiento debe definir una funcionalidad y solo una funcionalidad del sistema, ya que de otra manera se pierde la homogeneidad de la SRS. En caso de que un requerimiento sea demasiado extenso, lo que significa que su definición puede ser demasiado general, se subdivide en items, en los cuales se redactarán los subrequerimientos que lo componen.

Dado que lo que debemos hacer es definir qué hace el sistema, y no por ejemplo qué es lo que los usuarios harán con el sistema, cada requerimiento funcional comienza con "El sistema debe..." y luego lo que hará el sistema. Ej: El sistema debe mostrar el saldo de cuenta corriente de los clientes.

Es por esto que, y aunque parezca extraño a primera vista, en la Especificación de Requerimientos Funcionales, el usuario jamás es nombrado, ya que la lista de funcionalidades que el sistema debe cumplir no depende del usuario que la usa (si bien podrá darse el caso de que un usuario no pueda acceder a ella, de todos formas el sistema sí debe implementarla).

Por una cuestión de simplicidad, convenimos en usar la palabra "gestión" como un sinónimo de Presentación, Altas, Bajas y Modificaciones de los datos. Por Ej: "El sistema debe gestionar clientes" es una abreviatura de cuatro requerimientos funcionales, a saber "El sistema debe mostrar clientes", "El sistema debe dar de alta clientes", "El sistema debe dar de baja clientes" y "El sistema debe modificar clientes".

La Especificación de Requerimientos Funcionales tiene generalmente dos apéndices. Uno es el Diccionario, en dónde se definen los términos ambiguos o desconocidos usados en la redacción de los requerimientos y se los acota al contexto del sistema. Por ejemplo un "vendedor" no es lo mismo para un sistema de ventas en un comercio, que para uno de una compañía de seguros, que para una empresa que envía vendedores a recorrer ciudades y tomar pedidos. En el Diccionario es el lugar ideal para, al definir el término, enumerar los atributos que los caracterizan, como por ejemplo:

Vendedor: persona que recorre en un auto propio pueblos y ciudades y toma pedidos a los Clientes de la Empresa, a la vez que intenta sumar nuevos. Atributos:
Nombre
Apellido
Tipo y Nro de Documento.
Teléfono
Sueldo base
Comisión por venta (%)
Cuota de ventas ($)
Premio por cuota cumplida ($)

El otro apéndice usado es el que tiene un modelo de los listados a emitir por el sistema, detallando el título, las columnas del listado y los totalizadores. Por cada Requerimiento Funcional que implique emitir un

listado, debería haber un modelo de la salida impresa a generar. Es habitual hacerlos en hojas de cálculo, para documentar también las fórmulas que se usan para los totalizadores.

Una Especificación de Requerimientos Funcionales detallada, puede ser usada como un contrato informal entre el cliente que pide el desarrollo del sistema y los que lo construyen, ya que ahí estás enumeradas cada una de las funcionalidad a cumplir, con lo que conseguimos determinar claramente se un pedido del cliente es una omisión nuestra, de la cuál deberemos hacernos cargo, o un olvido del cliente, en cuyo caso deberemos cotizar la ampliación del sistema para que sea aceptada. Dado que el producto de nuestro trabajo es intangible, ser muy claros siempre es una cuestión importante.

Cómo funciona la World Wide Web:

Funcionamiento de un Web Site:

El funcionamiento de un Web-Site es un ejemplo típico de arquitectura de servicio, en donde múltiples clientes se conectan a un servidor (en algunos casos varios) en forma simultanea. En general el servidor depende de la instalación del sitio mientras que el cliente suele ser un navegador web (browser), por ejemplo Mozilla Firefox, Iceweasel, Opera, Safari, Internet Explorer, Galeon, Lynx, Chrome, etc. Como en todo esquema de servicio debe existir un protocolo que especifique de que forma se comunican e intercambian datos el cliente y el servidor, el protocolo utilizado en un web site es el protocolo HTTP que funciona “encapsulado” sobre el protocolo TCP/IP.

Introducción al Protocolo HTTP:

Sus siglas significan Hyper Text Transfer Protocol (Protocolo de Transferencia de HiperTexto). Básicamente el protocolo es iniciado por el cliente con un “request”, es decir un pedido de un recurso determinado, que es casi siempre contestado por el server con el envío de una respuesta (“response”) que incluye un código indicando si el pedido pudo ser resuelto por el server o no.
Un request genérico tiene la forma:

METODO URI PROTOCOLO CrLf
HEADERS*
CrLf
Datos

El MÉTODO en general puede ser GET o POST
URI es el identificador del recurso que se desea pedir, el formato es: http://host:port/path?query_string
PROTOCOLO debe ser HTTP / 1.1
CrLf es un Carriage Return seguido de un New Line (0x13,0x10)
Headers son de tipo: Header-Name: Value CrLf, y pueden indicar varias cosas.

Dado que el protocolo es "de texto", todo lo que se transmite entre el cliente y el servidor es simplemente texto, con comandos específicos que indican qué es lo que el cliente pide. La respuesta también es texto, que es analizados por el navegador para visualizarlo decorado como los tags lo indican.

Un ejemplo de pedido usando un cliente TELNET es:

```
$ telnet intranet.com 80
Trying 192.168.0.99...
Connected to intranet.com.
Escape character is '^]'.
GET / HTTP/1.1 <Enter>
Host: intranet.com <Enter>
<Enter>
```

Nótese que la página solicitada es "/", es decir, se pide por la raíz del sitio, y que se especifica la versión del protocolo, en este caso la 1,1 y que se indica el "Host" o nombre de dominio requerido, ya que este es un requisito de la versión 1,1 del protocolo que de esta manera permite que existan varios dominios en una misma dirección IP de servidor.

El servidor distingue que la petición ha terminado cuando detecta una línea en blanco, representada

en el ejemplo con el <Enter> vacío.

Un ejemplo de respuesta de un servidor podría ser:

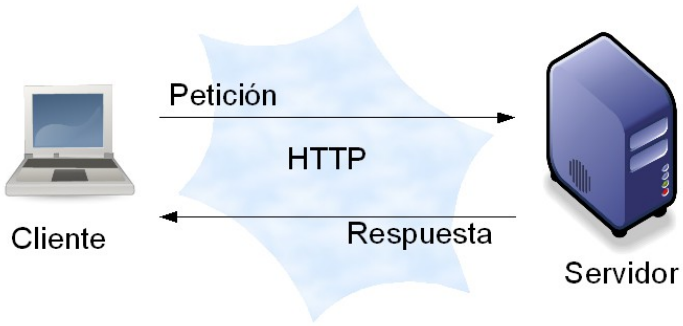
```
HTTP/1.1 200 OK
Date: Thu, 25 Mar 2010 04:25:01 GMT
Server: Apache/2.2.4 (Debian) PHP/5.2.6-1+lenny3 with Suhosin-Patch
Last-Modified: Wed, 22 May 2002 21:23:59 GMT
ETag: "e823c-24f-d0b8b9c0"
Accept-Ranges: bytes
Content-Length: 591
Content-Type: text/html

<html>
Datos…..
```

Nótese como otra vez, una línea en blanco distingue la cabecera de la respuesta del principio del contenido a mostrar al usuario.

Es importante destacar que en ambos casos es necesario enviar dos <enter> para indicar que el pedido termina. De la misma manera, el servidor separará con dos <enter> la cabecera de la respuesta del contenido, que en este caso es un archivo HTML.

Los datos que el servidor envía al browser (navegador) dependen del “Content-Type” declarado, básicamente los tipos más usados son texto plano (text/plain), código html (text/html), o imágenes (image/gif u otros).
De esta forma el cliente y el servidor se comunican por medio de tantos ciclos REQUEST-RESPONSE como sean necesarios, es de destacar que por cada REQUEST se debe iniciar una conexión nueva entre el cliente y el servidor ya que el protocolo HTTP en su forma básica no admite que en una misma conexión se haga más de un pedido al servidor.
En una página html simple con 3 imágenes por ejemplo es normal que se efectúen 4 conexiones al servidor: una para la página y luego una por cada imagen.



Tecnologías disponibles para el desarrollo de aplicaciones:
Para desarrollar aplicaciones y dotar a las páginas web de funcionalidad se puede trabajar tanto en el lado del cliente como en el lado del servidor, las variantes son:

Programación en el cliente:
· El browser envía un request.
· El server envía un response que contiene código que el browser entiende.
· El browser interpreta el código enviado por el server y realiza una determinada acción.

Programación en el servidor:
· El browser envía un request.
· El server ejecuta una aplicación que realiza una determinada acción.
· El server envía el resultado de dicha aplicación al cliente.
· El browser muestra el resultado recibido del server.

Esquema mixto: (programación en el cliente y en el servidor)
· El browser envía un request.
· El server ejecuta una aplicación que realiza una determinada acción.
· El server envía el resultado de dicha aplicación al cliente conteniendo código a interpretar por el browser.
· El browser interpreta el código enviado por el server y realiza una determinada acción.

La programación del lado del cliente tiene como principal ventaja que la ejecución de la aplicación se delega al cliente, con lo cual se evita recargar al servidor de trabajo. El servidor sólo envía el código, y es tarea del browser interpretarlo. La gran desventaja de esta metodología es que el código que el server envía es "sensible" a que cosas puede o no hacer el browser. El usuario puede, por ejemplo, decidir deshabilitar una funcionalidad del browser que es necesaria para que se ejecute un determinado servicio o peor aún, browsers distintos pueden interpretar el mismo código de distintas formas. Típicamente Netscape y Microsoft, que producían los dos browser más usados del mercado, no se ponen de acuerdo sobre como se implementan diversas tecnologías en el cliente.
Programar del lado del servidor tiene como gran ventaja que cualquier cosa puede hacerse sin tener en cuenta el tipo de cliente, ya que la aplicación se ejecuta en el servidor que es un ambiente controlado. Una vez ejecutada la aplicación, el resultado que se envía al cliente puede estar en un formato "normalizado" que cualquier cliente puede mostrar. La desventaja reside en que el server se sobrecarga de trabajo ya que además de servir páginas es responsable de ejecutar aplicaciones. A menudo esto redunda en requisitos de hardware mayores a medida que el server ejecuta más y más servicios.

Programación en el cliente	**Programación en el servidor**
HTML CSS DHTML JavaScript Java (applets) VBScript	CGI (Cualquier Lenguaje) ASP PHP mod_perl Java (servlets) mod_python

Debido a las incompatibilidades existentes y a la posibilidad de que el usuario controle que cosas se ejecutan y cuales no la programación del lado del cliente no es muy recomendable y debe limitarse a código altamente standard que pueda interpretarse de cualquier forma en cualquier browser, lo cual obliga a ejecutar la gran mayoría de las aplicaciones y servicios de un web site del lado del servidor.

Server Side Programming.

Para el desarrollo de aplicaciones del lado del servidor existen 3 grandes metodologías, utilizar el protocolo CGI, utilizar una API provista por el web-server o bien utilizar un "módulo" del web server.

El protocolo CGI:
El protocolo CGI (Common Gateway Interface) fue creado para establecer un protocolo standard de comunicación entre el web-server y cualquier lenguaje de programación de forma tal que desde el lenguaje "x" puedan recibirse datos que el usuario envía usando el método "POST" o "GET" y además el resultado de la aplicación sea derivado por el web-server al browser. Típicamente para recibir datos se usa alguna biblioteca o módulo del lenguaje elegido que implementa el protocolo CGI y para enviar datos simplemente se envían al standard-output desde el lenguaje elegido y el web-server se encarga de redireccionar esto al browser.
Para ejecutar una aplicación CGI el web-server en general procede de la siguiente manera:
· Se toma el "request del browser" y los datos que se envían al server por método "GET" o "POST" se pasan a variables de ambiente.
· El server redirecciona su salida standard al browser.
· El server crea un proceso (Fork) (que tiene la salida standard redireccionada)
· El server ejecuta en el proceso creado la aplicación deseada.
· Se ejecuta la aplicación

Cuando la aplicación termina de ejecutarse el proceso muere. Dentro de la aplicación se usa algún mecanismo para recuperar los datos enviados por el browser desde las variables de ambiente (todos los lenguajes manipulan variables de ambiente). El protocolo CGI justamente consiste en especificar de que forma los datos enviados por el browser se convierten en variables de ambiente, esto en general es transparente al usuario.
De esta forma pueden realizarse aplicaciones para un web-site en casi cualquier lenguaje, los lenguajes interpretados rápidamente ganaron terreno ya que tienen un ciclo de desarrollo en tiempo inferior a los lenguajes compilados y son más fáciles de debuggear dentro del ambiente CGI.
Los lenguajes no interpretados (C, C++) tienen como ventaja que requieren menos recursos del server al generarse el proceso CGI (no hace falta un interprete) y además suelen ser mucho más veloces en su

ejecución (no se necesita interpretar nada), sin embargo el desarrollar y debuggear suelen ser tareas muy complejas y no siempre se justifica el esfuerzo si la aplicación es pequeña. En los comienzos de la web la gran mayoría de las aplicaciones se encontraban en la categoría chica / muy chica por lo que la eficiencia no era un factor importante y por eso los lenguajes compilados no se utilizaron demasiado.
La desventaja de las aplicaciones CGI consiste en que el server debe realizar un fork, y ejecutar la aplicación o bien el interprete de la aplicación, y este ciclo que se cumple cada vez que se ejecuta la aplicación CGI insume muchos recursos y en general es costoso en tiempo para el server. Durante muchos años este esquema no muy eficiente dominó ampliamente el mundo de las aplicaciones Web.

Uso de una API del servidor:
Otra técnica factible consiste en utilizar una API (application programming interface) provista por el webserver para desarrollar aplicaciones, es decir que el web-server provee un lenguaje en el cual se pueden desarrollar aplicaciones. Este esquema, como podemos apreciar, es mucho más eficiente que el anterior ya que el web-server es el encargado de ejecutar las aplicaciones en forma directa sin necesidad de crear un proceso. Las desventajas son sin embargo importantes: en primer lugar las aplicaciones creadas en este marco no son portables ya que sólo pueden ejecutarse en un web-server determinado, esto es una gran desventaja frente a las aplicaciones CGI que podían una vez desarrolladas ejecutarse en cualquier servidor. La segunda gran desventaja es que frecuentemente un error de programación de una aplicación podría ocasionar que el web-server deje de funcionar, genere un error, se cuelgue, pierda memoria u otros problemas. Esto ocasiona que este tipo de aplicación no sea confiable.

Uso de un "módulo del web-server"
La tecnología más reciente para la ejecución de aplicaciones consiste en anexar a un web-server "módulos" que permiten al web-server interpretar un determinado lenguaje. De esta forma se logra eficiencia ya que el server no necesita crear un nuevo proceso por cada aplicación que ejecuta. Las aplicaciones son portables ya que son desarrolladas en un lenguaje standard que no depende del web-server, las aplicaciones son confiables ya que si bien pueden producir un error en el lenguaje en que están diseñadas si el módulo es sólido dichos errores no pueden comprometer al web-server. Al combinar las ventajas más importantes del ambiente CGI y un desarrollo basado en APIs evitando los inconvenientes de los mismos este esquema suele ser el más adecuado para ejecución de aplicaciones.

Cuadro Resumen:

	CGI (interpretado)	**CGI (compilado)**	**API del server**	**Modulo del server**
Ejemplos	Perl, Python	C, C++	Netscape Enterprise	PHP, ASP, Mod_perl, mod_python, FastCGI
Tiempo de desarrollo	Corto	Largo	Medio	Corto
Debugging	Sencillo	Complejo	Complejo	Sencillo
Confiabilidad	Alta	Alta	Baja	Alta
Eficiencia	Baja	Media	Alta	Alta

Benchmarks:

Test 1: 1000 ejecuciones de un programa de 1 línea.

Tecnología	**Tiempo (segundos)**
CGI : C	20,6
CGI : Perl	23,8
CGI : Python	45,2

PHP	16,2
Mod_python	30
Mod_perl	16,6
FastCGI	16,4

Test 2: 100000 ejecuciones de un programa de 1 línea.

Tecnología	Tiempo (segundos)
CGI : C	2522
CGI : Perl	2886
CGI : Python	5486
PHP	2420
Mod_python	3519
Mod_perl	2117
FastCGI	2120

Test 3: 10000 ejecuciones de un programa de 7000 líneas.

Tecnología	Tiempo (segundos)
CGI : C	258
CGI : Perl	963
CGI : Python	978
PHP	304
Mod_python	347
Mod_perl	476
FastCGI	280

En los tests puede verse como las variantes CGI insumen mucho tiempo de “lanzamiento” de la aplicación. Esto se prueba ejecutando muchas veces aplicaciones muy chicas (solo 1 línea de código). De esta forma podemos ver como en este aspecto las tecnologías que no requieren que el server genere un nuevo proceso (php, fastcgi, mod_perl, mod_python) son mucho más eficientes que aquellas que sí lo necesitan.
En el tercer test donde se ejecuta una aplicación de gran tamaño se puede apreciar que la variante “compilada” en CGI es muy eficiente ya que no requiere tiempo alguno de interpretación y el tiempo necesario para generar el proceso es mínimo en comparación con el tamaño de la aplicación. Lo importante de este tercer test es que tanto php, como mod_perl o fastcgi pese a requerir de un interprete son también veloces en este tercer test.

Evolución de los Web-Sites

Desde la aparición de la web distintos “modelos” o prototipos de web-sites fueron siendo predominantes en distintos periodos de tiempo, de acuerdo a las características “comunes” de los sitios predominantes en cada momento podemos diferenciar 3 generaciones de web-sites distintas:

Web-Sites de primera generación:
En un web-site de primera generación las páginas se desarrollaban, se subían al servidor y el servidor se encargaba de enviar las páginas al browser, es un modelo basado en páginas estáticas, en donde predominaba el uso de texto, links a otros sites o a otras páginas del mismo site y listas con “bullets” para enumerar cosas. Frecuentemente se usaban líneas horizontales “rules” para separar contenidos y las páginas de gran extensión vertical con gran cantidad de texto, listas y links eran comunes. Si bien no eran visualmente atractivos estos sites estaban enfocados a funcionar en forma veloz y entregar al usuario gran cantidad de información interrelacionada.

Web-Sites de segunda generación:
La segunda generación de web-sites implicó una revolución en lo visual, a medida que los web-sites se volvían emprendimientos más comerciales que científicos el hecho de “capturar” usuarios se torno una premisa y por ello se le dio gran importancia al aspecto visual. Las páginas con “layouts” visualmente atractivos fueron más y más populares, el uso abundante de imágenes, imágenes animadas y elementos multimedia se volvió común, aparecieron páginas que controlaban los fonts, el estilo y la posición en la que los mismos se ubicaban. La gran mayoría de los sites usaban tablas HTML para controlar la posición exacta en que los elementos aparecerían en el browser y tags nuevos en html como “font” u otros para controla la forma en la cual los mismos se verían. El concepto fue dominar la presentación de la información. Otra característica importante de esta segunda generación de Web-sites fue la aparición explosiva de más y más aplicaciones a medida. Los servicios que ofrecía un site se volvían factores importantes en la atracción de usuarios, chats, foros de discusión, banners, contadores y muchas aplicaciones más empezaron a aparecer y las falencias del protocolo CGI, a medida que las aplicaciones eran mas grandes y la cantidad de usuarios crecía exponencialmente, comenzaron a hacerse notar.

Web-Sites de tercera generación:
La tercera generación de web-sites siguió basada en lo visual, el gran cambio vino en la forma cómo se generaba la información. Las páginas estáticas que dominaban el 100% de los sitios de primera y segunda generación fueron reemplazadas por páginas dinámicas que el web-server generaba en el momento, a partir de información que en general se guardan en una base de datos. Estos sitios “dinámicos” permiten actualizar la información e incluso cambiar completamente la forma en que se muestran dichos datos en velocidades asombrosas. Los sitios de tercera generación facilitaron las aplicaciones interactivas, la información en tiempo real y que día a día se ofrecieran nuevos servicios. Las aplicaciones empezaron a desarrollarse también usando otras tecnologías dejando de lado el protocolo CGI. Aplicaciones en ASP, mod_perl o PHP, mucho más poderosas y eficientes que sus pares CGI, son el standard de este tipo de sitios.