Recibido: 24 de setiembre del 2016
Aprobado: 11 de octubre del 2016

# Modelo de procesos para el desarrollo del *front-end* de aplicaciones *web*

**José Jesús Valdivia Caballero**
Jovaldiv@ulima.edu.pe
Universidad de Lima. Lima, Perú

## Resumen

La frase 'divide y vencerás', propone una estrategia de dividir una estructura de poder centralizada en grupos de poder más pequeños. Con base en este principio se propone el proyecto para desarrollar una aplicación web, donde pueda dividirse el proceso de implementación del aplicativo en procesos con sus herramientas, recursos propios y ciclo de vida. Tanto O'Reilly (2007) y Ginige (1998) hacen mención que las tecnologías a usar en front-end de una aplicación web son distintas a las del back-end, abriendo así la posibilidad de desacoplar la implantación en componentes propios del lado del cliente o browser web y del servidor de aplicaciones, bases de datos y archivos estáticos.

*Palabras clave: ingeniería de software / ingeniería web / migración*

## Abstract

The phrase 'divide and rule' and which proposes a strategy of dividing a centralized power structure into smaller power groups. Based on this principle, this study proposes that in the project management for the development of a web application, where the application's implementation can be divided in processes with their own tools, resources and life cycle. Both O'Reilly (2007) and Ginige (1998) mention that the technologies to be used in the front-end of a web application are different from those of the back-end, making it possible to decouple the implementation into its own components from the side of the customer or web browser and from the applications servers, databases and static files.

*Keywords: software engineering / web engineering / migration*

## 1. Introducción

La ingeniería de *software* es una rama de la ingeniería relativamente nueva (Naur y Randell, 1969), y la ingeniería *web* de *software* más nueva aún. De esta última se ha discutido desde finales de los años noventa (Pressman, 1998) y principios del 2000 (Ginige y Murugesan, 2001) respecto a los retos que se presentan con el desarrollo de este tipo de software, así como también sobre la necesidad de adaptar procesos convencionales del desarrollo de software a este ámbito.

Desde estas primeras disertaciones e inquietudes a la actualidad han surgido propuestas de metodologías para el desarrollo de *software* basadas en el modelo incremental en espiral (Mathai, Venugopal y Abraham, 2015) y Ginige (1998), como también alternativas o adaptaciones del estándar UML (Unified Modeling Language) (Booch, Rumbaugh y Jacobson, 1998) para el diseño de *software web*, tales como UWE (UML-based Web Engineering) (Koch y Kraus, 2002), IFML (Brambilla, 2015) y SysML (OMG SysML, 2015) para el desarrollo de aplicaciones *web,* porque las metodologías tradicionales no separan la metodología de modelamiento del *front-end* a la del *back-end.*

Además, con las tecnologías actuales usadas para el desarrollo de aplicaciones *web*, se ha desacoplado los lenguajes implementados en el *back-end*, como PHP, Java, Ruby, Python, Lua, entre otros; y del *front-end,* tales como HTML5, CSS3, Javascript y AJAX, siendo estas tecnologías con las que interactúa el usuario (O'Reilly, 2007), diferenciándose así de las aplicaciones de escritorio que usan las mismas tecnologías tanto en el cliente como en el servidor.

Por último, tenemos estándares de calidad dentro de la industria del *software* (ISO/IEC, 2006) (Perú. Comité Técnico de Normalización en Ingeniería de *Software* y Sistemas de Información [CTN-ISSI], 2006) (Oktaba, Alquicira, Su, Martínez, Quintanilla, Ruvalcaba, …, Flores, 2005), los cuales brindan una referencia como marco de procesos, mas no explican cómo implementar estos procesos en un ambiente de desarrollo de aplicaciones *web*.

**Tabla 1.** ***Comparación entre la ingeniería de* software *y la ingeniería* web *de* software**

| Software engineering | Web engineering |
|---|---|
| Software system has small user range. | WebApps has large user range. |
| User requirements are specific. | User requirements are changes with time. |
| Growth and change is small. | Rapidly changing. |
| Development budgets varies in a wide range according to the size of the company. | Development budgets are small. |
| Development time is longer. | Development time is small. |
| Hardware and software environments constraints are specific. | Hardware and software environments constraints are not specific. |
| Design and development expertise is few. | Design and development expertise is available in wide rang. |
| Security and legal issues are not much important. | Security and legal issues are not much important. |
| Less emphasis on user interfaces. | More emphasis on user interfaces. |

Traducción:

| Software engineering | Web engineering |
|---|---|
| El sistema de software tiene un pequeño rango de usuarios. | WebApps tiene un amplio rango de usuarios. |
| Los requerimientos del usuario son específicos. | Los requerimientos del usuario cambian con el tiempo. |
| El crecimiento y el cambio son reducidos. | Cambia rápidamente. |
| Los presupuestos de desarrollo varían en un amplio rango según el tamaño de la empresa. | Los presupuestos de desarrollo son reducidos. |
| El tiempo de desarrollo es más largo. | El tiempo de desarrollo es más corto. |
| Las restricciones de los entornos de hardware y software son específicas. | Las restricciones de los entornos de hardware y software no son específicas. |
| La experiencia en diseño y desarrollo es escasa. | La experiencia en diseño y desarrollo está disponible en una amplia gama. |
| La seguridad y las cuestiones legales no son muy importantes. | La seguridad y las cuestiones legales no son muy importantes. |
| Menor énfasis en las interfaces de usuario. | Mayor énfasis en las interfaces de usuario. |

Fuente: Mathai, Venugopal y Abraham (2015)

## 2. Marco teórico

### 2.1. *Frameworks*

Los *frameworks* proveen una implementación del andamiaje para el desarrollo completo de una aplicación, facilitando la reutilización de componentes presentes en la estructura. Proporcionan una serie de puntos donde se pueden acoplar funcionalidades adicionales. No son un patrón arquitectural pero sí una colección de patrones de diseño y clases trabajando en conjunto, que tienen como fin resolver un problema específico. Por esta razón, se cuenta con una gran variedad de *frameworks* en distintos lenguajes de programación ya sea en PHP, Java, Ruby, Pyton, Javascript, entre otros (Pressman, 2010).

### 2.2. Librería

El concepto de "librería" es más antiguo que el de *framework*. Las librerías consisten en una colección de clases o métodos que proveen comportamiento a otra aplicación. También se diferencian de los *frameworks* en que no se especializan en un control de flujo de datos interno, uso de herencia y patrones de diseño. Dentro de estas librerías tenemos por ejemplo Jquery y Mootools como librerías de Javacript y Twitter Bootstrap como librería de CSS (Rotem-Gal-Oz, 2007).

### 2.3. Editor de código fuente

Permite ver y editar archivos propios de una aplicación. Muchos de ellos ofrecen al programador, según el lenguaje de programación que se esté editando, ayuda para la edición del código, ayuda visual en el uso de palabras reservadas del lenguaje, asistencia automática para la identación, navegación dentro de los archivos y carpetas, entre otros. Esta funcionalidad está integrada dentro de los IDEs (como Netbeans, Eclipse, Visual Studio, etc.), pero, a diferencia de un IDE, los editores de código fuente como Sublime Text, Brackets, Pluma no tienen fácil instalación de *plugins*, opciones de compilación, *debugers*, detección de errores de sintaxis, entre otros (Haughee, 2013).

### 2.4. Microservicios

Los microservicios son pequeños, autónomos e independientes servicios que trabajan juntos para reemplazar el uso de una aplicación desarrollada bajo una arquitectura monolítica. La forma de despliegue de estos microservicios es a manera de plataforma como servicio (PAAS), pues cada uno puede estar en un sistema operativo diferente, como también cada uno puede ser implementado en lenguajes de programación distintos (Rotem-Gal-Oz, 2007).

## 2.5. *Front-end*

Dentro del contexto del desarrollo de aplicaciones *web*, implica el uso de las tecnologías con las que interactúa directamente el usuario. Normalmente estas tecnologías son desarrolladas en los lenguajes de HTML, CSS y Javacript; también se usan las herramientas de diseño gráfico como Photoshop o Fireworks. El objetivo es desarrollar la interfaz gráfica de usuario (GUI), buscando una experiencia de uso bien valorada por el usuario final, siendo en algunos casos necesario hacer investigación, estudios y pruebas para llegar a este fin. Además, dentro del desarrollo de las aplicaciones *web* es posible desarrollar el *front-end* de la aplicación sin contar con una aplicación *back-end* que interactúe con la base de datos (Kavourgias, 2015).

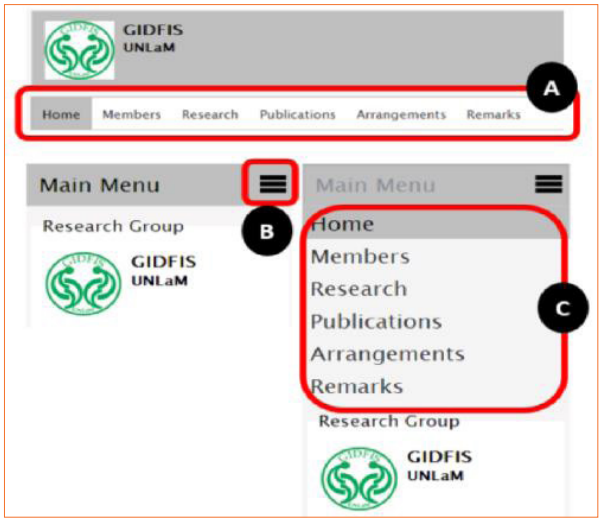
## 2.6. *Back-end*

En el contexto del desarrollo de aplicaciones están implicadas las actividades realizadas del lado del servidor; es decir, las tareas de base de datos y los servidores de aplicaciones que el usuario no puede visualizar en el explorador de Internet. Los lenguajes usados comúnmente son PHP, Java, Ruby, .NET, Python, entre otros, los cuales son los encargados de interactuar con la base de datos (Kavourgias, 2015; Alvarado, 2012).

## 2.7. *Responsive web design*

Este término es usado en referencia a la manera en que cada explorador de Internet responde a su ambiente, aportando así a la mejora de la experiencia del usuario en el uso del sitio, independientemente del tamaño de resolución y pantalla del dispositivo. Para lograr esto es imprescindible el uso de CSS3 acompañado del uso *media queries* dentro del mismo CSS (Fielding, 2014).

Figura 1. **Ejemplo de un página web usando el comando @media screen.**




Fuente: Gavino, Fuertes, Loprosti, Defranco y Lara (2014)

### 2.8. *Task runner tool*

Son herramientas de *software* que permiten automatizar, mediante tareas o pasos, el andamiaje, la creación, las pruebas y la compilación de una aplicación de *software.* Dentro de los beneficios de usar estas herramientas se tiene la reducción del tiempo y esfuerzo para la realización de las tareas (Pillora, 2014).

### 2.9. Herramienta de gestión de proyectos

Son herramientas de *software* que automatizan la gestión de la información generada en los proyectos por los *stakeholders*. Estas herramientas deben facilitar lo siguiente:

- Gestión de usuarios, roles y privilegios
- Elaboración de diagramas de Gantt
- Gestión de cronogramas y calendario
- Gestión de las actividades
- Gestión y versionamiento de documentos por medio de archivos y directorios
- Integración con sistemas de control de versiones de código fuente (Lesyuk, 2013)

### 2.10. Base de datos orientada a documentos

Una base de datos orientada a documentos es aquella donde el concepto de fila es reemplazado por uno más flexible llamado "documento", el cual permite la representación jerárquica en lugar de relacionar un registro de una fila con otro; aunque sí está abierta la posibilidad (no recomendada) de relacionar documentos. El formato de documentos más utilizado es el formato JSON (Javascript Object Notation). Dentro de los principales beneficios de este tipo de base de datos está la fácil escalabilidad y rendimiento (Chodorow, 2013).

### 2.11. Modelado de *software*

El modelado de *software* es el análogo de los planos de arquitectura para la construcción de una casa. Los diagramas se modelan bajo el principio *top-down*. El modelado puede tener varias orientaciones; dirigido a la información, patrones, requerimientos, objetos, entre otros. Se deben considerar una serie de principios para el modelado de *software* (Pressman, 2010):

- El diseño debe ser trazable hacia el modelo de requerimientos.
- Considerar siempre la arquitectura del sistema que se está modelando.
- El diseño de los datos es tan importante como el diseño de los flujos de los procesos.
- Las interfaces deben ser diseñadas con cuidado.
- El diseño de interfaces de usuario deben ser potenciadas con las necesidades de los usuarios.
- El diseño de componentes debe ser funcionalmente independiente.
- Los componentes diseñados deben estar desacoplados uno del otro en su ambiente externo.
- Los modelos diseñados deben ser fáciles de entender.
- El diseño debe ser desarrollado iterativamente.

## 3. Estado del arte

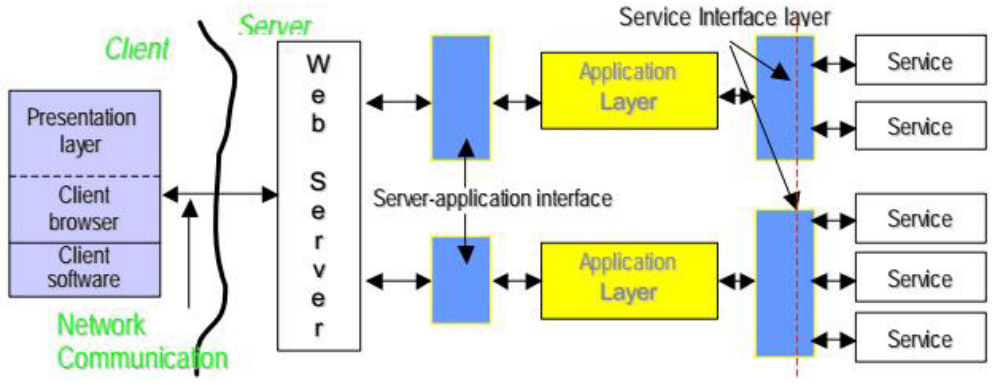
### 3.1. Ingeniería de *software web*

El autor Ginige (1998) define la ingeniería *web* como el desarrollo y mantenimiento sistemático de sistemas de información accesibles por la *web.* Considera también a esta

ingeniería como una rama de la evolución de la ingeniería de *software*. En su artículo describe una aproximación de las metodologías usadas para el desarrollo de aplicaciones *web* de gran tamaño y menciona previamente a estas metodologías, algunos de los principales factores que incurren en inconvenientes en este tipo de proyectos, como por ejemplo la falta de adecuación de los modelos de procesos y la alta frecuencia de cambios a las que se ven sujetas los entregables.

El modelo de procesos propuesto es un modelo de desarrollo en espiral que contemple los grupos de planificación; gerencia del proyecto y producto, análisis, diseño, producción, entrega y mantenimiento. Cabe resaltar que el autor propone además una arquitectura flexible que soporte soluciones basadas en aplicaciones *web.*

**Figura 2. Arquitectura flexible para el desarrollo de aplicaciones web**

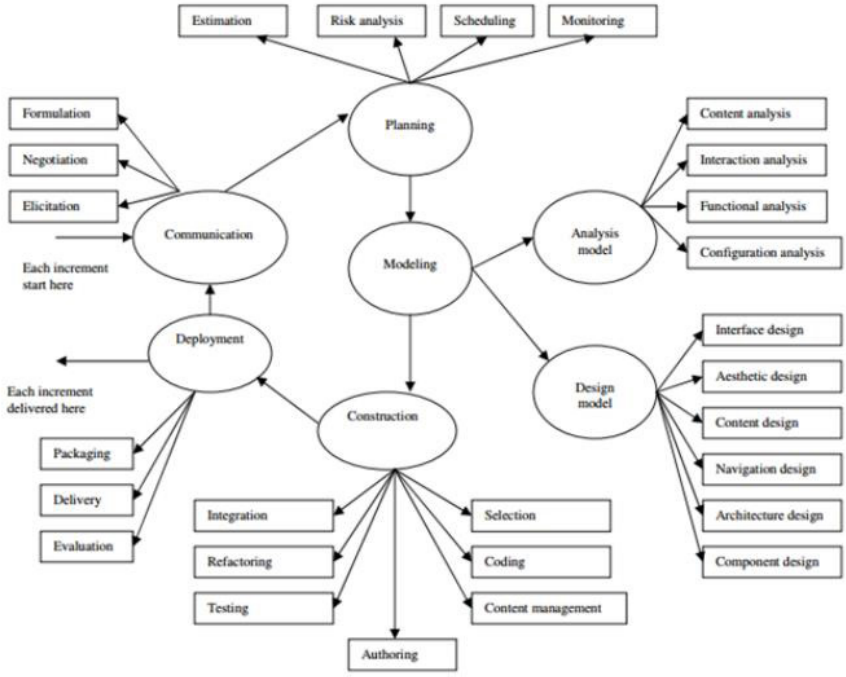



Fuente: Ginige (1998)

Los autores Mathai, Venugopal y Abraham (2015) exponen que existen diferencias entre los procesos involucrados en el desarrollo de aplicaciones *web* contra los procesos de desarrollo convencionales de *software*. Asimismo, explican las razones que producen esta diferencia, tales como el acceso a las soluciones *web* por parte de gran cantidad de usuarios, los cuales no se pueden considerar como facilitadores de información para los requerimientos en el inicio del proyecto, así como los cambios frecuentes por la evolución de los requerimientos.

A pesar de las pequeñas diferencias que puedan existir entre la ingeniería de *software* y la ingeniería *web* de *software*, no se descarta que en ambos casos se deban considerar grupos de procesos de análisis, diseño, implementación y mantenimiento.

**Figura 3. Modelo de procesos para el desarrollo de aplicaciones *web***




Fuente: Mathai, Venugopal y Abraham (2015)

Cabe resaltar que estos autores proponen un modelo de procesos de desarrollo en espiral incremental para el desarrollo de este tipo de *software.*

De igual manera, los autores Kumar y Sangwan (2011) expresan que la evolución de las aplicaciones basadas en la *web* genera el inconveniente de que los modelos de ingeniería de *software* convencional no se pueden usar directamente y, así también, que el ciclo de vida tradicional del *software* debe ser adaptado a los requerimientos propios de las aplicaciones *web*. Lo antes referido da la pauta de entrada para determinar que hay una necesidad de adaptar los modelos de ingeniería de *software* a los requerimientos de desarrollo y mantenimiento de estas nuevas soluciones. Adicionalmente, los autores señalan que no es conveniente usar el modelo de cascada, pues los requerimientos no son estables durante el proyecto y proponen como solución el modelo incremental.

Es importante resaltar la mención a los atributos de calidad propios de una aplicación *web* y las comparaciones que hacen sobre la ingeniería de *software web* contra la tradicional, siendo en esta última destacable que en un proyecto *web* hay una mayor cantidad de usuarios, menor presupuesto, limitaciones por el ambiente del *hardware*, seguridad y mayor énfasis en las interfaces de usuario (GUIs).

**Tabla 2.** ***Atributos de calidad de una aplicación*** **web**

| | |
|---|---|
| 1. Network intensiveness. | 2. Concurrency. |
| 3. Unpredictable load. | 4. Performance. |
| 5. Availability. | 6. Data driven. |
| 7. Content sensitive. | 8. Continuous evolution. |
| 9. Immediacy. | 10. Security. |
| 11. Aesthetics. | 12. Usability. |
| 13. Functionality. | 14. Efficient and reliable. |
| 15. Compatibility. | 16. Interoperability. |

Traducción:

| | |
|---|---|
| 1. Intensidad de la red. | 2. Concurrencia. |
| 3. Carga impredecible. | 4. Rendimiento. |
| 5. Disponibilidad. | 6. Dirigido por los Datos. |
| 7. Contenido sensible. | 8. Evolución continua. |
| 9. Inmediatez. | 10. Seguridad. |
| 11. Estética. | 12. Usabilidad. |
| 13. Funcionalidad. | 14. Eficiente y confiable. |
| 15. Compatibilidad. | 16. Interoperabilidad. |

Fuente: Kumar y Sangwan (2011)

Adicionalmente, los autores Rodríguez, Vera, Vallés, Martínez y Giulianelli (2014) comentan que los nuevos estándares *web* están en constante mejora con el pasar de los días y esto conlleva a la interrogante de si será necesario construir una aplicación móvil nativa o una aplicación *web* elaborada en HTML5 y CSS3, que se adapte a los dispositivos móviles por medio del uso de media *query*, aun cuando una aplicación nativa tiene mayor capacidad de procesamiento. Como parte de su investigación, ellos indican y describen estrategias que ayudarán a crear aplicaciones *web* adaptadas a los dispositivos móviles:

- Conectividad: Dado que el contenido a procesar es descargado de Internet usando principalmente HTML5, es necesario manejar la caché de la aplicación *web* en el servidor de archivos estáticos, con el fin de reducir la dependencia de la descarga continua de archivos estáticos.

- Almacenamiento: Aprovechando el almacenamiento local del dispositivo se puede reducir los volúmenes de tráfico presentes en el servidor. Este almacenamiento en el dispositivo también incluye mantener los datos de la sesión.

- Acceso al *hardware:* Si bien las aplicaciones *web* no pueden acceder a todas las funcionalidades del *hardware* de un dispositivo móvil, están emergiendo herramientas que facilitan este trabajo, como los sensores que identifican la orientación de la pantalla, el estado de la batería, la temperatura, el acceso a la cámara, entre otras.

- Gráficos: Con el uso de HTML5 se puede realizar gráficos dentro de un mismo sitio *web,* o también es posible escalar una misma imagen mediante el uso del formato SVG.

- *Look and feel:* Una de las características sobresalientes de HTML es que permite su adaptación al tamaño de la pantalla del dispositivo con el que se está accediendo al sitio *web*. Este concepto es llamado *responsive web design* y se logra por medio del uso de la tecnología de CSS con los atributos Media Query.

- Portabilidad: Una gran ventaja es que una aplicación *web* es portable, porque corre sobre un *web browser* y estos a su vez pueden estar soportados en distintas plataformas de sistemas operativos.

En pocas palabras, concluyen que la diferencia de la capacidad de una aplicación *web* contra una aplicación nativa se irá reduciendo a medida que los estándares *web* faciliten el acceso al *hardware* de los dispositivos móviles.

### 3.2. Lenguajes de modelado de sistemas de *software web*

Los autores Koch y Kraus (2002) exponen que existen limitaciones con la notación UML al granular los componentes usados en las aplicaciones *web.* Proponen una extensión metodológica del UML llamada UWE (UML Web) la cual provee una guía para la construcción de modelos sistemáticos enfocados en la personalización de cada producto de un proyecto.

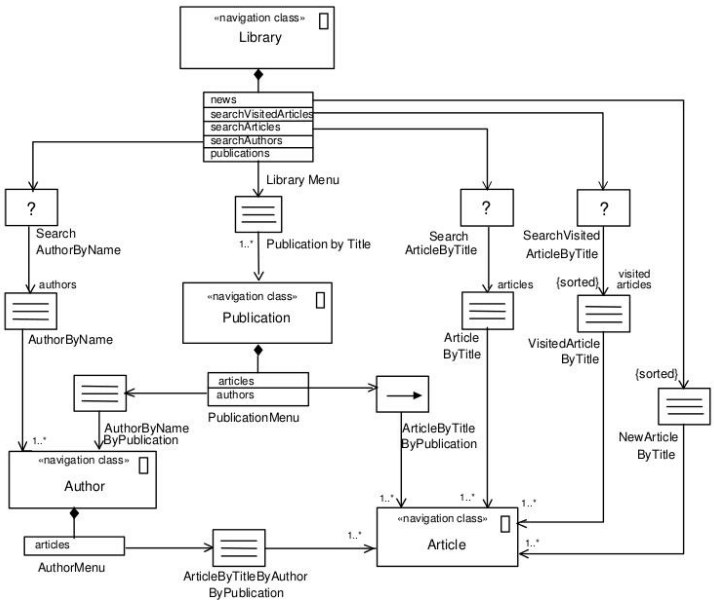
A un nivel de análisis y diseño proponen que a partir de un modelo de casos de uso y un diagrama conceptual de las clases de diseño (ambas de UML), se deben desarrollar a manera de flujo de eventos, los modelos de navegación, la presentación de la información y las tareas a realizar.

Es importante resaltar que los autores concluyen que ven conveniente refinar los diagramas presupuestos, como también incluir diagramas usados en UML (adaptándolos), como por ejemplo los diagramas de secuencia.

Por otro lado, tenemos también la especificación IFML, la cual es una notación de modelado que describe las principales partes de una aplicación *web* del lado *front-end*. El nombre de esta notación es IFML (Interaction Flow Modeling Language) y surge como

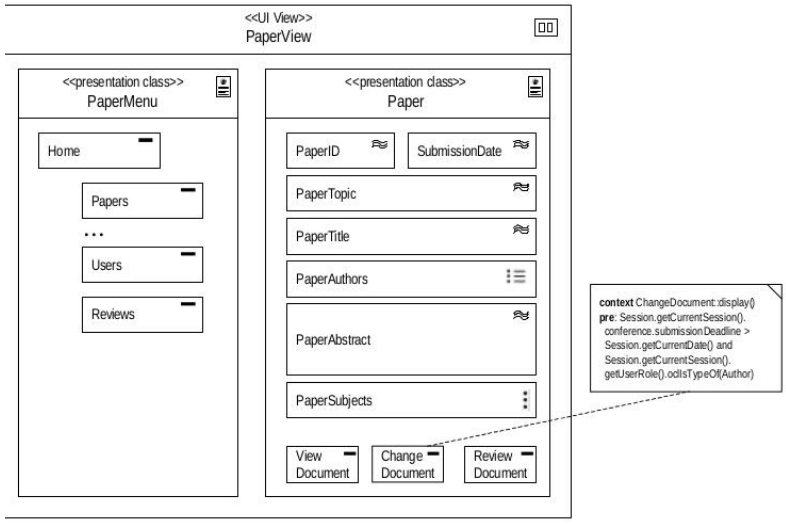
respuesta a la convergencia que están teniendo los sistemas de información al estándar HTML5, independientemente de los dispositivos en los que se esté usando la aplicación.

**Figura 4.** Modelo de navegación usando UWE




Fuente: Koch y Kraus (2002)

**Figura 5.** Ejemplo de modelado de una GUI del sistema usando UWE

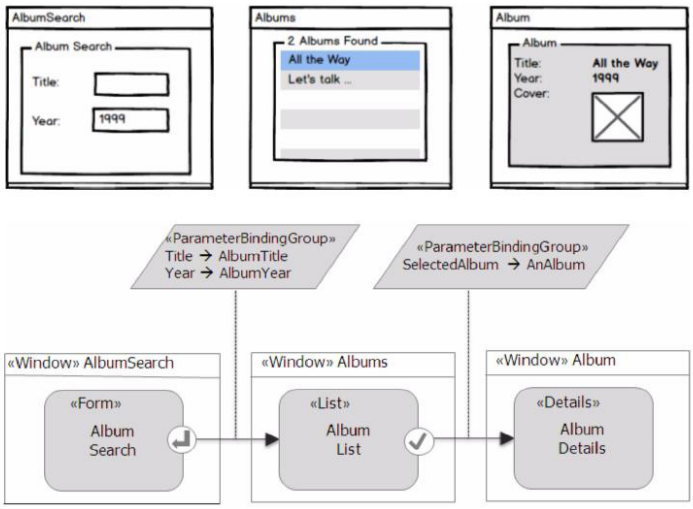



Fuente: Koch, Kraus y Hennicker (2001)

Entre los beneficios de esta metodología tenemos, por ejemplo, la especificación formal de las perspectivas del *front-end*, como el contenido, la composición de las interfaces, la interacción con la navegación y la conexión entre la capa de negocio y presentación. De esta manera se permite separar las tareas en función de los roles y capacidades de un equipo de desarrollo, como también tener tanto información técnica y no técnica para ser validada por los *stakeholders*.

- Dentro de las perspectivas de diseño que nos ofrece IFML se cuenta con las siguientes:
- Vista de especificación estructural: Consiste en la definición de *containers*, relaciones anidadas con su respectiva visibilidad.
- Vista de especificación del contenido: Referida a la definición de los componentes visuales.
- Especificación de eventos: Radica en definir los eventos que se darán con la GUI.
- Especificación de los parámetros de unión: Consiste en la definición de las dependencias de entrada y salida de los componentes de la vista.

**Figura 6. Ejemplo del uso de un modelo en IFML (abajo) y su contraparte con una GUI (arriba)**




Fuente: Brambilla (2015)

## 4. Aporte teórico

Dentro del aporte teórico a presentar relacionado a los modelos de procesos exclusivos para el desarrollo del ciclo de vida del *front-end* de aplicaciones *web* podemos decir que no hay evidencia de trabajos similares realizados por otros pares, ni tampoco estándares internacionales; pero sí encontramos evidencia que no se encontraría en dicho campo la ingeniería de *software*, sino de la ingeniería (de *software*) *web* (Pressman, 2010).

Este nueva ingeniería, según la información encontrada, ha sido mencionada en discusiones que tuvieron lugar a finales de la década del noventa (Pressman, 1998) y principios del 2000 (Ginige y Murugesan, 2001), acerca de los nuevos retos presentes con el desarrollo de este tipo de *software*, como también sobre la necesidad de adaptar procesos convencionales del desarrollo a este ámbito.

Entre los aspectos claves que mencionaron en esos artículos resaltan los siguientes:

- Comprender el sistema en su totalidad de funciones y ambiente operativo.
- Tender identificado claramente a los *stakeholders*.
- Especificar claramente los requerimientos técnicos y no técnicos de la solución.
- Desarrollar una arquitectura que soporte los requerimientos técnicos y no técnicos.
- Desglosar la arquitectura en subproyectos o subprocesos para su posterior implementación.
- Incorporar mecanismos de control efectivos para la ejecución de los procesos a llevar a cabo.
- Aprender no solo de los requerimientos de negocio, sino de aspectos sociales y personales del ambiente donde se desplegará la solución.
- Medir el rendimiento del sistema.
- Poder refinar y actualizar el sistema una vez terminado.
- Adaptar los procesos del desarrollo de las aplicaciones *web* a procesos de ingeniería.
- Desarrollar aplicaciones *web* dinámicas escalables.

Dado que el objetivo general del presente artículo es desarrollar y validar el modelo de procesos en una pequeña empresa dedicada al desarrollo de *software*, será idóneo usar la

ISO 29110 (Perú. Comité Técnico de Normalización en Ingeniería de *Software* y Sistemas de Información, 2012) y Moprosoft 1.3 (Olktab, et al., 2005) porque ambas normas tienen como público objetivo las pequeñas organizaciones desarrolladores de *software*. Se usará la ISO 29110 para determinar y adaptar las tareas a realizar en esta etapa y Moprosoft para determinar y adaptar los indicadores y roles utilizados.

En ambos estándares se menciona que es indispensable tener una gestión de la configuración de la documentación del proyecto, como también poder atender las incidencias presentadas durante el proyecto. Por ello sería necesario usar un *software* de gestión de proyectos, encontramos los siguientes de código abierto: Redmine, Alfreso y Tuleap.

**Tabla 3. *Herramientas de código abierto a usar dentro del grupo de procesos de elaboración del* front-end *de una aplicación* web**

| Etapa | Herramienta | Tipo |
|---|---|---|
| Gestión del proyecto | Redmine, Alfreso, Tuleap | Herramienta de gestión de proyectos |
| Todas | Elementary OS, Ubuntu Mate, Linux Mint | Sistema operativo. |
| Etapa de análisis | LibreOffice 5 | Suite de herramientas de ofimática. |
| Etapa de diseño | Inkscape, GIMP, StarUML | Herramientas para modelar requerimientos. |
| Etapa de construcción | Git y Bazaar | Versionamiento de código fuente |
| Etapa de construcción | Ruby, PHP, Python, Java, Go. | Lenguajes para desarrollar el ruteador |
| Etapa de construcción | Sublime Text, Pluma, Brackets, Atom | Herramientas para escribir código fuente (no IDEs) |
| Etapa de pruebas. | Jasmine, Grunt, Open HMI Tester | Herramientas para elaborar pruebas |

Elaboración propia

El principio de "divide y vencerás" será usado en este grupo de procesos, con miras a que la finalidad de los mismos sea el desarrollo de un *front-end* de bajo acoplamiento con el *back-end* que tendrá la lógica de negocio y persistencia de datos, postulando que el *front-end* tendrá su proceso de ciclo de vida independiente al del *back-end*, siendo este primero el tema a tratar, pues está dentro del alcance del presente estudio. Es importante recalcar que el grupo de procesos a proponer en la etapa de construcción e integración deberá contemplar su integración con el *back-end*, el cual deberá tener su propio grupo de procesos, con roles y herramientas. El concepto de que se maneje el ciclo de vida del *front-end* se creó como una conclusión propia luego de analizar el estado del arte.

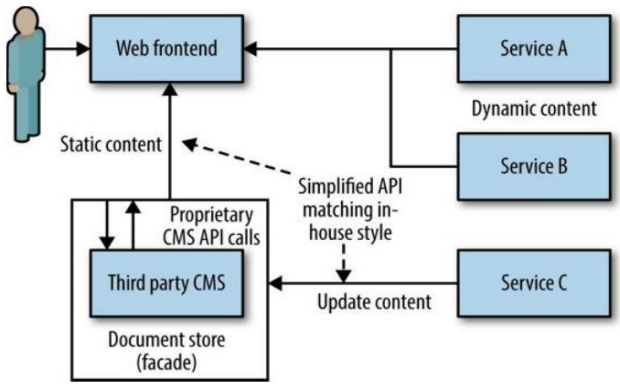
Otro de los aspectos a considerar dentro de esta etapa es que, de acuerdo a la literatura revisada, el modelo en cascada no es el idóneo para este tipo de trabajos sino el modelo

iterativo incremental, sin obviar los grupos de procesos de análisis, diseño, implementación y mantenimiento; esto es según Mathai, Venugopal y Abraham (2015). Se deberá considerar también a Kuhl (2014), que en su experiencia de veinte años en la docencia, señala que el uso de metodologías ágiles permite también una rápida reacción ante los cambios presentados.

Dentro de lo investigado en el estado del arte, con respecto a la arquitectura de *software* a usar solo se proponía una donde haya desacoplamiento en los componentes de la aplicación *web* (O'Reilly, 2007), para lo cual también habría que desacoplar inclusive los componentes de la solución de *back-end*, y según Newman (2015) la mejor opción sería usar la arquitectura de microservicios, donde el componente a desarrollar sería un microservicio encargado del "ruteo" a los microservicios que tendrían acceso a las bases de datos a usar.

El beneficio de escoger esta arquitectura y desarrollar este microservicio "ruteador" sería que además permitiría manejar los archivos estáticos de Javascript, CSS y Fonts en un servidor *web* independiente (ver figura 7) donde pueda manejar la caché del mismo para mejorar los tiempos de respuesta, tal como lo sugieren Rodríguez et al. (2014).

**Figura 7. Arquitectura de microservicios dividiendo el front-end, back-end y archivos estáticos.**




Fuente: Newman (2015)

Para el "ruteador", se desarrolló uno basado en el *framework* Sinatra del lenguaje de programación Ruby, lenguaje que según Benedetto, Carabio, Ramona, Alvez, Fernández, Etchart, … Martínez (2015) es uno de los de mayor dominio dentro de las aplicaciones web, con buena legibilidad y cantidad de código presente en Github (ver tabla 4.5 y tabla 4.6). El *framework* que usaremos para desarrollar el "ruteador" será Sinatra, como se indicó anteriormente, debido a que después del *framework* MVC Ruby On Rails (ROR) de Ruby, es el que mejor documentación brinda en su página web (tutoriales, códigos, contribuciones

de usuarios, entre otros), y de la que se puede encontrar en la Internet. Pero a diferencia de este último ofrece mayor flexibilidad para ser adaptado a las necesidades del desarrollador y proyecto, aparte de ser más ligero en tiempos de respuesta y uso de recursos (comparando con ROR).

**Tabla 4.** ***Tabla comparativa de lenguajes orientados a objetos***

| Orden | Legibilidad | Buena documentación | Eficiencia | Reusabilidad | Dominios de aplicación web | Desarrollo de aplicaciones con interfaces amigables |
|---|---|---|---|---|---|---|
| 1 | Python | Java | Assembler | Ada | Javascript | C# |
| 2 | Eiffel | Python | C | Haskell | Ruby | Haxe |
| 3 | Haxe | Mathematica | Forth | Eiffel | Scala | Delphi |
| 4 | Go | C# | Fortran | O'Caml | Python | Objective C |
| 5 | Lua | Ada | C++ | D | Haxe | Scala |
| 6 | F# | C | Eiffel | Common Lisp | Clojure | Java |
| 7 | Smalltalk | Common Lisp | D | Clojure | Groovy | Visual Basic |
| 8 | Ruby | Perl | Ada | Scala | PHP | F# |
| 9 | Groovy | Factor | O'Caml | F# | Erlang | Python |
| 10 | Haskell | Objective C | Go | Smalltalk | Java | Clojure |

Fuente: Benedetto et al. (2015)

**Tabla 5.** ***Tabla de rankings TioBe, Github y PYPL. Febrero 2015***

| Orden | TioBe | GitHut | PYPL |
|---|---|---|---|
| 1 | C | JavaScript | Java |
| 2 | Java | Java | PHP |
| 3 | C++ (con tendencia ascendente) | Python | Python |
| 4 | Objective-C (con tendencia descendente) | CSS | C# |
| 5 | C# | PHP | C++ |
| 6 | JavaScript (con tendencia ascendente) | Ruby | C |
| 7 | PHP (con tendencia descendente) | C++ | Javascript |
| 8 | Python | C | Objective-C |
| 9 | Visual Basic .NET (con tendencia ascendente) | Shell | Matlab |
| 10 | Visual Basic | C# | R (con tendencia ascendente) |

Fuente: Benedetto et al. (2015)

Otro beneficio de desarrollar un "ruteador" basado en este *framework* del lenguaje de Ruby es que fue de inspiración para la elaboración de *frameworks* como Flask de Python, Lumen de Php, Spark de Java, Martini de Go, Scalatra de Scala. Por tal motivo, este desarrollo podría ser el punto de partida para futuro desarrollo de "ruteadores" en los lenguajes ya citados.

Este componente a desarrollar, documentar y publicar en Github en la presente investigación será el encargado, dentro de la arquitectura de microservicios, de ser el intermediario de comunicar al usuario de la aplicación *web* con los microservicios que tendrán acceso a datos; protegiendo así las direcciones IPs de estos últimos. Otra función de este "ruteador" será la de renderizar las vistas HTML y decorarlas con los archivos JS y CSS, los cuales estarán en un servidor Gnix o Lighttpd. Por último, este componente igualmente deberá manejar las sesiones de usuario y restringir el acceso a funciones de llamada a microservicios o renderizado de vistas con la ayuda de un microservicio que tenga la lista de control de accesos de los usuarios.

También, dentro de la etapa de pruebas y mantenimiento, según lo investigado y para comprobar el grado de satisfacción de los usuarios, emplearemos las herramientas *open source* sugeridas por Mateo (2014) como Open HMI Tester, *software* que permite registrar las acciones y navegación del usuario en una GUI. De esta manera nos ayudará a determinar patrones de comportamiento de los usuarios con la GUI elaborada. Para elaborar las GUIs de prueba se utilizó el "ruteador" propuesto a desarrollar, pues este nos debería permitir

elaborar las vistas HTML desacoplada de los componentes del *back-end* a desarrollar; que se podrían reemplazar por microservicios de prueba que simulen la interacción con la base de datos real a desarrollar, siendo reemplazada por una base de datos no relacional como MongoDB (Chodorow, 2013), donde estarían contenidos los documentos JSON para realizar las pruebas con el usuario.

Para realizar pruebas con el otro grupo de *stakeholders* mencionado (el equipo de desarrollo), se utilizó la herramienta de *software* libre llamada Grunt (Pillora, 2014) la cual ayudará a automatizar mediante tareas o pasos el andamiaje, la creación, las pruebas, la reducción de los archivos Javacript y CSS de la aplicación a desarrollar. Esta herramienta funciona bajo la tecnología de Javascript de NodeJS y además Grunt utiliza la librería de Jasmine JS para la ejecución de las pruebas unitarias. Otro beneficio que tendremos por utilizar Grunt es que se automatizaría la generación de los archivos reducidos de Javascript y CSS, los cuales, al usarse a petición del "ruteador" a desarrollar para decorar las vistas HTML, reducirán el tráfico en la red y acelerarán el tiempo de carga de las vistas, porque los archivos reducidos tienen un menor peso en *kilobytes* (Kb).

Para la etapa de mantenimiento sería indispensable usar las herramientas de gestión de proyectos, siendo Redmine la mejor alternativa a usar pues se le puede instalar módulos para la gestión y seguimiento de incidencias de *software* e integración con repositorios de versionamiento de código como Git y Bazaar.

## 5. Conclusiones

Si bien la ingeniería de *software* web es de finales de los años noventa (Pressman, 1998) y principios del 2000 (Ginige y Murugesan, 2001), en la actualidad se cuenta con una diversidad de herramientas de *software* libre que pueden ser usadas durante todo el ciclo de vida de una aplicación *web* (ver tabla 4.7). En adición, ya se producen disertaciones sobre el grupo de procesos de desarrollo de *software* que se deben seguir para el desarrollo de aplicaciones *web*, siendo el modelo iterativo incremental el modelo a emplear, sin obviar los grupos de procesos de análisis, diseño, implementación y mantenimiento; esto es según Mathai, Venugopal y Abraham (2015).

Otro aspecto importante a considerar es que si se desarrolla una aplicación *web* bajo la arquitectura de microservicios, se podría aplicar el principio "divide y vencerás" porque sería posible que el *front-end* tenga un grupo de procesos de ciclo de vida independientes al *back-end*, donde se lograría un desacoplamiento de componentes de *software* y también los procesos, recursos y artefactos a usar en las etapas de análisis y diseño. El IFML (Brambilla, 2015) sería una alternativa para desarrollar documentación y artefactos propios dentro de una organización para soportar los procesos de desarrollo y mantenimiento del *front-end.*

Fundamentado en la arquitectura de microservicios, desarrollar un "ruteador" basado en el *framework* Sinatra de Ruby, que a diferencia de Ruby on Rails, este ofrece mayor flexibilidad para ser adaptado a las necesidades del desarrollador y del proyecto, además de ser más ligero en tiempos de respuesta y uso de recursos.

Otro beneficio de este ruteador sería también que nos permitiría desacoplar los componentes a desarrollar propios del *front-end* tales como los archivos HTML, Javascript y CSS en un servidor de archivos externos de alta concurrencia como son Gnix o Lighttpd, los cuales nos permitirán gestionar los tiempos de respuesta ante las peticiones HTTP del usuario.

## Referencias


Alvarado, I. (2014, 12 de abril) ¿Qué es *front-end* y *back-end* en la programación *web*? [Mensaje en un blog]. Recuperado de http://serprogramador.es/que-es-frontend-y-backend-en-la-programacion-web/

Benedetto, M., Carabio, A., Ramona, A., Alvez, C., Fernandez, M., Etchart, G., Martinez, D. (2015). Selección de lenguajes orientados a objetos para un estudio comparativo y análisis de rendimiento. *XVII Workshop de Investigadores en Ciencias de la Computación.* Recuperado de http://sedici.unlp.edu.ar/bitstream/handle/ 10915/45742/Documento_completo.pdf?sequence=1

Booch, G., Rumbaugh, J., Jacobson, I. (1998). *The Unified Modeling Language User Guide.* California: Addison Wesley

Brambilla, M. (2015). IFML: Building the Front End of Web and Mobile Applications with OMG's Interaction Flow Modeling Language. Recuperado de http://www.slideshare.net/mbrambil/ifml-interaction-flow-modeling-language-tutorial-ui-ux-modeling-design-icwe-2014-object-management-group-by-marco-brambilla.

Chodorow, K. (2013). *MongoDB: The Definitive Guide* (2ª ed.). California: O'Reilly Media.

Fielding, J. (2014) Introduction to Responsive Design. En Apress, US (Ed.), *Beginning Responsive Web Design with HTML5 and CSS3* (1ª ed.), (p.2).

Gavino, S., Fuertes, L., Loprosti, L., Defranco, G., Lara, M. (2014). Aplicaciones para dispositivos móviles: Una aproximación en las prácticas de enseñanza de los sistemas de representación**.** *III Jornada de Investigación, Transferencia y Extensión de la Facultad de Ingeniería*, (pp.591-598). Recuperado de http://sedici.unlp.edu.ar/ bitstream/handle/10915/47900/Documento_completo.pdf?sequence=1

Ginige, A. (1998) Web engineering: Methodologies for Developing Large and Maintainable Web Based Information Systems, *IEEE International Conference on Networking,* (pp. 89-92).

Ginige, A. y Murugesan, S. (2001). The Essence of Web Engineering. Managing the Diversity and Complexity of Web application Development, *IEEE MultiMedia Magazine, 8*(2), 22-25.

Haughee, E. (2013). *Instant Sublime Text Starter.* Birmingham: Packt Publishing.

International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) (2006). *ISO/IEC 14764:2006: Software Engineering, Software Life Cycle Processes Maintenance.* Geneva, Switzerland: Autor

Kavourgias, C. (2015) What's the Difference Between the Front-End and Back-End? [Mensaje en un blog]. Recuperado de http://blog.digitaltutors.com/whats-difference-front-end-back-end/.

Koch, N. y Kraus, A. (2002). The Expressive Power of UML-Based Web Engineering, *2nd International Workshop on Web-oriented Software Technology*.

Koch, N., Kraus, A. y Hennicker, R. (2001). The Authoring Process of the UML-Based Web Engineering Approach**.** *International Workshop on Web-Oriented Software Technology*.

Kuhl, J. (2014). Incorporation of Agile Development Methodology into a Capstone *Software* Engineering Project Course. *2014 ASEE North Midwest Section Conference.*

Kumar, S. y Sangwan, S. (2011). Adapting the *Software* Engineering Process to Web Engineering Process, *International Journal of Computing and Business Research, 2(*1), 42-63.

Lesyuk, A. (2013). *Mastering Redmine.* Birmingham: Packt Publishing.

Mateo, P. (2014). *Mejora de la calidad del software y de la experiencia del usuario a través de las interfaces humano-máquina* (tesis doctoral). Universidad de Murcia, España.

Mathai M., Venugopal, R., y Abraham, J. (2015) Software Engineering Process in Web Application Development. *IOSR Journal of Computer Engineering, 17*(1), 28-32.

Naur P., y Randell, B. (eds) (1969). Software Engineering. *Nato Software Engineering Conference.*

Newman, S. (2015). *Building Microservices. Designing Fine-Grained Systems.* California: O'Reilly Media.

Oktaba, H., Alquicira, C., Su, A. Martínez, A., Quintanilla, G., Ruvalcaba, M.,Flores, M. (Eds.) (2005). Modelo de procesos para la industria de software Moprosoft. Por niveles de capacidad de procesos. Recuperado de http://lsc.mxl.uabc.mx/~angelica/V_1.3_MoProSoft_por_niveles_de_capacidad_de_procesos.pdf

OMG SysML (2015). *OMG Systems Modeling Language (OMG SysMLTM). Version 1.4.* Recuperado de http://sysml.org/docs/specs/OMGSysML-v1.4-15-06-03.pdf

O'Reilly, T. (2007) What is web 2.0: Design Patterns and Business Models for the Next Generation of Software. *Communications & Strategies, 65*(1), 17-37.

Perú. Comité Técnico de Normalización en Ingeniería de Software y Sistemas de Información. – INDECOPI [CTN-ISSI] (2006). Tecnología de la información, procesos del ciclo de vida del software, NTP-ISO/IEC 12207.

Pillora, J. (2014). *Getting Started with Grunt: The JavaScript Task Runner.* Birmingham: Packt Publishing.

Pressman, R. (1998). Can Internet-Based Applications be Engineered?. *IEEE Software, 15*(5), 104-110. DOI: 10.1109/MS.1998.714869

Pressman, R. (2010) *Software Engineering a Practitioner's Approach* (7ª ed.). Nueva York: McGraw-Hill.

Rodríguez, R., Vera, P., Vallés, F., Martínez, M., Giulianelli, D. (2014) Analysis of Current and Future Web Standars for Reducing the Gap between Native and Web Applications. *XX Congreso Argentino de Ciencias de la Computación*.

Rotem-Gal-Oz, A. (2007). *Frameworks vs. Libraries,* Recuperado de https://web.archive.org/web/20070504053354/http://www.ddj.com/blog/architectblog/archives/2006/07/frameworks_vs_l.html