Desarrollo de software de mediana y gran envergadura

Alejandro Teruel

23 de abril de 2013

Objetivo del curso

Introducir al estudiante a la problemática y a aquellas técnicas de desarrollo de software de mediana envergadura representativas del enfoque disciplinado asociado con la Ingeniería de Software. Se enfatizarán las técnicas de diseño y verificación de sistemas de programas que:

1. Incrementen la productividad del desarrollador de software;
2. Permiten controlar la complejidad inherente a sistemas de mediana envergadura;
3. Introducen al estudiante al desarrollo en equipos (*teams*).

¿Qué es software de mediana envergadura?

Distinguiremos entre software de pequeña, mediana y gran envergadura:

- Un software es de pequeña envergadura si lo puede desarrollar un programador;
- Un software puede considerarse de mediana envergadura si su desarrollo requiere de un equipo de entre dos y diez personas.
- Un software puede considerarse de gran envergadura si su desarrollo requiere de la labor de múltiples equipos, totalizando más de veinte personas.

¿Cuál es el tamaño de un software de mediana envergadura?

Una forma más convencional, aunque polémica, de definir la envergadura de un software es por el número de líneas de código fuente (SLOC, una abreviación de *Source Lines Of Code*). Por ejemplo:

- El software desarrollado como parte de una asignación en un curso introductorio de programación no debería sobrepasar los 200 SLOC y su envergadura es claramente muy pequeña. Un programador profesional puede desarrollar software como éste por su cuenta en menos de dos días de trabajo.
- La envergadura típica de un software desarrollado como proyecto de laboratorio en un trimestre o semestre de la carrera puede llegar a incluir de 2 a 3 KSLOC. Este tipo de software todavía está dentro del alcance de un solo programador profesional y se considera de envergadura pequeña.
- Para mediados de la década de 1990 el software controlador de una planta generadora de electricidad o de un marcapasos podía incluir hasta 50 KSLOC. Si bien se estima que un programador profesional puede desarrollar hasta 180 KSLOC por año, la complejidad, las exigencias de calidad y el caracter delicado de estos controladores hace más sensato que su desarrollo se realice por un equipo de tres a cuatro desarrolladores, con al menos uno de ellos dedicado a las actividades de validación y verificación del software. Si bien hace tiempo se distinguía la envergadura de un software netamente por el tamaño de su código fuente, más recientemente se considera que la envergadura del software debe estar calificado por factores como su complejidad y calidad. Por su tamaño, los dos software mencionados en este aparte aún son de envergadura pequeña, pero por su complejidad y exigencias de calidad podrían considerarse como de envergadura mediana.
- Un compilador optimizante contiene del orden de 100 KSLOC. Este es el tamaño de un compilador optimizante y se podría considerar de envergadura mediana. Tengo entendido (pero no confirmado) que haciia elaño 2000, el software de control para una red de cajeros automáticos en Venezuela era de este mismo orden.

- Para 1995, Microsoft Word incluía más de 2 millones de SLOC[1] y era claramente de gran envergadura. El tamaño de los sistemas de operación ha tendido a crecer, como podemos ver para el caso de dos familias de sistemas de operación:

Año	Windows	SLOC
1993	Windows NT 3.1	6 millones
1994	Windows NT 3.5	10 millones
1996	Windows NT 4.0	16 millones
2000	Windows 2000	29 millones
2002	Windows XP	40 millones
2007	Windows Vista	~50 millones

Fuente: http://www.codinghorror.com/blog/2006/07/diseconomies-of-scale-and-lines-of-code.html

Año	Linux	SLOC
2000	Debian 2.2	55-59 millones
2002	Debian 3.0	104 millones
2005	Debian 3.1	215 millones
2007	Debian 4.0	283 millones
2009	Debian 5.0	324 millones

Fuente: http://en.wikipedia.org/wiki/Source_lines_of_code
Algunas fechas de liberaciones agregadas por el autor

Un ejercicio de visualización

Supongamos que lo contratan para desarrollar un programa de 100 KLOC y supongamos que, para poder competir exitosamente, ese programa tiene que estar listo en menos de un año. ¿Desarrollaría Ud. sólo el programa?

Hagamos una cuenta optimista. Supongamos que Ud. es un programador fuera de serie y que logra programar en promedio cien líneas de código al día. Sin tomar vacaciones, ni fines de semana, sin enfermarse sino dedicando todos los días del año a pensar, escribir, compilar y depurar cien líneas diarias, terminaría el proyecto en 2.73 años...

Además, si usted quiere disfrutar de una vida saludable, tampoco debe programar los 365 días del año -debería dejar de trabajar al menos los fines de semana (104 días) y tomarse 20 días hábiles de vacaciones al año. Lo cuál significa que los 2.73 años de desarrollo, se convertirían en 4,14 años. Por ende para cumplir sólo con el año de plazo dado por su cliente, debe escribir y depurar 400 líneas de código diariamente.

Pero, ¿es realista pensar en programar cien o cuatrocientas líneas de código al día en promedio? Los estudios muestran que el programador profesional produce, en promedio, ¡menos de diez líneas de código al día durante el desarrollo de un proyecto de mediana o gran envergadura! Más adelante veremos el por qué de esta cifra que luce, a primera vista, escandalosamente baja.

Para tener una mejor idea de lo que son cien mil líneas de código, consideremos qué espacio ocupa el imprimirlas. Una página de un libro de texto tiene alrededor de 40 líneas por página. Por ende, si se hacen los cálculos, imprimir 100.000 SLOC produce 5 tomos de 500 páginas cada uno. ¡Ni George R.

1 *Fuente:* http://www.wired.com/wired/archive/3.09/myhrvold.html

R. Martin, el autor de la serie "Game of Thrones" es así de productivo!

Y eso no es todo. Ningún cliente sensato va a aceptar que usted produzca un código que no pueda mantener, es decir, que lo obligue a él a depender de usted para actualizarlo, ajustarlo, extenderlo o incluso corregir estos pequeños defectos que quedaron después de su asombroso esfuerzo. Para ello, el código debe estar documentado apropiadamente, para que otros desarrolladores puedan entenderlo. Capers Jones reportó, hace ya unos años, que, en promedio, en la industria americana de software se elaboran cuatro páginas de documentos adicionales por cada 53 líneas de código Java[2]. O sea que a los cinco tomos de listado, hay que agregar unos 18 tomos adicionales de documentación. En el 2001, en los cursos de Ingeniería de Software en la USB, encontramos que ciertas metodologías de desarrollo obligan a producir entre 300 y 500 páginas de documentación para un software de 9.5 KLOC, lo que extrapolado linealmente a 100 KLOC nos lleva a estimar que en tal proyecto pudieran tener que elaborarse el equivalente a entre 6 y 10 tomos adicionales de documentos adicionales al listado de código fuente.

¿Qué contienen estas páginas? Entre otros aspectos: la arquitectura del software, la filosofía de diseño, justificaciones de diseño, el diseño de la base de datos, las normas o estándares que aplican, las pruebas que se diseñaron y el resultado de aplicarlas y el manual del usuario. Para un software de cierta envergadura, sólo el manual del usuario puede llevarse varios tomos. Por ejemplo, la documentación asociada a los manuales de un sistema de operación UNIX normalmente sobrepasa las mil páginas.

La única manera que puedas desarrollar ese software de 100 KSLOC en un año por tu cuenta es basándote en algún código similar desarrollado previamente. Sin embargo, entender un código ajeno no es sencillo. En 1989, Buckley sugirió que el tamaño máximo de un programa que se puede entender contiene entre 7 KLOC y 15 KLOC de código fuente y estimó que entender ese programa lo suficientemente bien para modificarlo se llevaría entre tres y seis meses de trabajo.

La conclusión es inescapable: para desarrollar un software de 100 KSLOC en un plazo competitivo, aprovecha código ya hecho y trabaja en equipo.

Un ejemplo de software exitoso y actual de mediana envergadura

Según una noticia del 2010, un equipo pequeño (iniciado por tres ingenieros de software) en *Facebook* desarrollaron, en apenas 18 meses, HipHop, un traductor de PHP a C++. Esto permite un mayor rendimiento de los servidores de web de Facebook, un elemento crítico para obtener una velocidad de respuesta satisfactoria[3].

¿Por qué distinguimos entre software de envergadura pequeña, mediana y grande?

El desarrollo de un software de un tipo de envergadura u otra presenta diferencias *cuantitativas* y, más

2 Capers Jones: *Assessment and Control of Software Risks,* Prentice-Hall, 1994

3 *Fuente*: http://royal.pingdom.com/2010/06/18/the-software-behind-facebook/

críticas aún, diferencias *cualitativas* características. Supongamos que nuestro hipótetico cliente acepta que el software de 100.000 SLOC lo desarrolle un equipo de hasta cuatro profesionales, liderizados por usted, siempre y cuando la entrega del producto se haga en un año. ¿En qué se diferencia tal desarrollo en equipo de un desarrollo en solitario?

Para comenzar, tiene que organizar a su equipo: ¿como jefe, usted se reserva la responsabilidad de tomar todas las decisiones importantes[4] en forma ejecutiva, o las decisiones se toman en consenso? ¿Todos hacen de todo, o cada quien toma un rol especializado (por ejemplo (1) arquitecto-diseñador (2) programador (3) responsable de pruebas (*tester*) y (4) responsable de herramientas de apoyo (*toolsmith*)? En todo caso usted percibirá inmediatamente que ahora tiene que invertir más tiempo en comunicaciones. Si optó por centralizar las decisiones importantes, ahora lo interrumpen más para consultarle decisiones, o para que las aclare o para que las reconsidere; si por el contrario optó por una organización más consensual, usted se encontrará dedicando más tiempo a reuniones con sus colegas. Alguien tiene que descomponer el trabajar en tareas, asignar las tareas y cerciorarse que las tareas no se retrasen. También tendrá que ver qué hace si alguno de sus colegas se desmotiva, se enferma o renuncia al trabajo. Si comienza sólo con otra persona inicialmente y después contrata los demás miembros, tendrá que ver cómo hace para poner esos nuevos miembros al día con lo que se ha hecho e integrarlos al equipo de trabajo. En resumen, tiene una serie de labores administrativas o gerenciales que no tenía antes y sencillamente encontrará que tiene menos tiempo para dedicarle a la programación, por lo que su productividad personal en términos de escritura y depuración de líneas de código bajará. Sus colegas también invierten tiempo en reuniones o en consultarle y en comunicarse entre ellos,por lo que también tendrán menos tiempo para programar. En resumen, hay todo un mundo de gestión y dinámica interpersonal que no está presente cuando usted trabaja en solitario.

A medida que crece la envergadura del software, crecen las dificultades gerenciales. Al pasar de software mediano a software grande, ya no son cuatro personas trabajando juntos; en un proyecto de gran envergadura pueden haber 20, 50, 100 o más desarrolladores. Se forman múltiples equipos y por ende a la dinámica interpersonal se agregan dinámicas intergrupales.

Pero, ¿las únicas diferencias en el desarrollo de software de pequeña envergadura y software de mediana o gran envergadura es el tamaño del producto, el número de personas que intervienen en el desarrollo y la gestión del desarrollo?

¿Qué ocurre en la práctica?

Distintos estudios coinciden en concluir que muchos proyectos de desarrollo de software fracasan pues:

- No llegan a entregar un producto (*proyectos abandonados*);
- Entregan un producto pero mucho más tarde que lo prometido, a un costo mucho mayor que el acordado inicialmente, o que hace mucho menos que lo que se pidió originalmente *(proyectos parcialmente exitosos).*
- Entregan productos que no se usan, bien sea porque no cumplen con las expectativas y/o las necesidades de los usuarios finales o los clientes o bien sea porque no son confiables, es decir,

4 Y vaya si habrán decisiones importantes, decisiones sobre qué va a hacer el software y cómo lo va hacer, sobre qué componentes va a tener, si se van a desarrollar de cero o si van a aprovechar componentes previamente desarrollados o disponibles, sobre cómo se van a repartir las tareas entre los miembros del equipo y para cuándo tienen que estar listas esas tareas, sobre cómo alcanzar una calidad adecuada del software y cómo y cuándo se sabrá que la han alcanzado, sobre la vocería y la relación con el cliente, etc.

presentan tasas de falla inaceptables.

Así por ejemplo, el Grupo Standish reporta las siguientes tasas de éxito, fracaso e indefinición[5] para proyectos de desarrollo de software[6]:

	1994	1996	1998	2000	2002	2004	2009
Tasa de éxito	16%	27%	26%	28%	34%	29%	32%
Tasa de fracaso	31%	40%	28%	23%	15%	18%	24%
Tasa de indeterminación	53%	33%	46%	49%	51%	53%	44%

Una búsqueda de "software failures" o similar por la red le proporcionará ejemplos específicos de fracasos, tanto de productos de software como de proyectos de desarrollo del mismo. Las fallas de software han sido responsables de muertes, sobre-exposición a radiación, diagnósticos médicos errados, apagones, fallas en controladores de todo tipo de dispositivos, la pérdida de reservaciones y equipaje, así como retrasos en vuelos, paralización de operaciones, el corte de servicios y todo tipo de pérdidas monetarias. Varias empresas han ido a la quiebra a consecuencia de los efectos de tales fallas o el fracaso de proyectos críticos de desarrollo. A continuación una breve descripción de los casos recientes más sonados:

Año	**Caso**
2013	Una falla de software ocasionó la interrupción de los servicios de correo electrónico de *Hotmail (Outlook.com)* durante 16 horas, afectando a un estimado de 2 millones de usuarios. La falla afectó el sistema de enfriamiento del centro de datos causando un incremento repentino de temperatura que, a su vez, causó una falla de los servidores. No sólo se interrumpieron los servicios *Microsoft* sino que también fue afectado el servicio de almacenamiento en la nube de *Skydrive.*
2012	El proyecto para desarrollar un sistema de gestión de cadena de suministros para la fuerza aérea estadounidense (*Expeditionary Combat Support System*) fue cancelado en Noviembre 2012, después de tres años de desarrollo, dejando una pérdida de alrededor de un millardo de dólares. La alternativa, corregir lo hecho para que el sistema entrara en operación en el 2020 -con un retraso de ocho años a lo requerido- hubiera costado otro US$ 1,1 millardos de dólares. Este sistema es uno de seis, de los nueve sistemas de software planificados por el Pentágono para mejorar deficiencias en su gestión financiera, que llevan un retraso de hasta 12 años y que están costando US$ 6,9 millardos por encima de su estimación original. En enero 2013, un Comité del Senado estadounidense inició una investigación sobre este

5 Un proyecto en estado *indefinido* (en Inglés "*a challenged project"*) es un proyecto que tiene tantos problemas que no se puede considerar que está en camino de convertirse en exitoso, a la vez ni se termina de cancelar ni genera suficiente motivación para hacer un esfuerzo para reorganizarlo. Típicamente (a)el proyecto ya está atrasado, (b) la mayor parte de los miembros del equipo desarrollador están descorazonados, desmotivados y estresados, (c) el código está acumulando defectos y la documentación está desatendida o desactualizada., (d) los miembros del equipo pasan mucho tiempo en reuniones improductivas centradas en la asignación de culpas y (e) hay mucha presión para que se entregue algo para acabar de una vez y (f) hay una creciente sensación de desconexión entre el equipo y el resto de la organización. Véase http://softwareandotherthings.blogspot.com/2012/10/challenges-of-challenged-projects.html

6 *Fuente:* http://www.galorath.com/wp/software-project-failure-costs-billions-better-estimation-planning-can-help.php

	caso.[7]
2012	La baja calidad de la nueva aplicación para iOS6 que lanzó Apple para competir con Google Maps. En la primera semana de operación las quejas de los usuarios sobre errores en direcciones, ubicaciones e instrucciones para llegar a diferentes localidades, además de una seie de fallas francamente bizarras, obligaron al presidente de Apple a disculparse públicamente y le costó el empleo al gerente encargado del desarrollo de la aplicación[8].
2011	En septiembre, el gobierno del Reino Unido abandonó un proyecto de desarrollo de un sistema para llevar los registros digitales de salud de todos sus ciudadanos. El proyecto había comenzado a desarrollarse en el 2002, costó US$ 18,7 millardos y no llegó a producir un sistema capaz de usarse. Se consideró el proyecto de tecnología informático más costoso acometido por este gobierno[9].
2011	*AXA Rosenburg Group*, un grupo dedicado a los servicios de inversión, no le informó a sus clientes-inversores que el software que les proporcionaba para ayudar a gestionar sus carteras de inversión y sus activos tenía varios defectos. Los clientes se quejaron de algunas fallas, pero la empresa les dijo que los problemas eran debido a la volatilidad del mercado financiero. Una investigación por parte del ente regulador de este mercado en Estados Unidos, la *Securities and Exchange Commission* (SEC) determinó que hubo fraude. Como consecuencia de ello, la empresa tuvo que pagar una multa de US$ 25 millones y US $217 millones como compensación a los clientes, por las pérdidas en que incurrieron[10].

El recurso del método

La complejidad y envergadura que alcanza el software y las exigencias que necesita cumplir obligan a prestar particular atención a la gestión del proceso de su desarrollo. Sin embargo, la gestión por si misma no es suficiente. En 1968, ante la preocupación de muchos especialistas en relación a la calidad del software que se ponía en producción, se acuñó el término de *Ingeniería del Software.* Los especialistas argumentaban que se podían adoptar y adaptar las técnicas y los enfoques propios de la Ingeniería a la problemática de la construcción de software fiable en plazos predecibles y costos razonables.

> La ingeniería desarrolla y aplica, en forma disciplinada, métodos y técnicas fundamentadas en conocimientos científicos a la elaboración de artefactos *útiles* sujeto a *restricciones de recursos.*

El software es un artefacto "útil" y el desarrollo de software está sujeto a restricciones de recursos como tiempo, esfuerzo y dinero, por lo que la propuesta de los especialistas lucía -y sigue luciendo- prometedora[11].

7 *Fuente:* http://www.bloomberg.com/news/2013-01-24/senators-to-probe-air-force-s-1-billion-failed-software.html

8 *Fuente:* http://www.crn.com/slide-shows/applications-os/240144057/the-10-biggest-software-stories-of-2012.htm;jsessionid=LK3dtH1K6dENn5toPu-ktw**.ecappj03?pgno=2

9 *Fuente:* http://www.computerworld.com/s/article/9222864/10_biggest_ERP_software_failures_of_2011

10 *Fuente:* http://www.kualitatem.com/9-latest-software-failures-in-enterprise-applications-2011-2012/

11 ¿Qué conocimientos científicos pudiesen sustentar técnicas y métodos ingenieriles para desarrollar software? Debido a la naturaleza literalmente *intangible* del software, no parecía tener sentido fundamentar a esta nueva disciplina en ciencias

Pueden distinguirse actividades características en el desarrollo de cualquier proyecto de ingeniería. Por ejemplo:

- Se dan actividades de *construcción* o producción del artefacto.
- Se dan actividades de inspección, *verificación* o prueba para cerciorarse que el artefacto que se está construyendo cumpla con ciertas especificaciones pre-establecidas y tenga las propiedades que requiera tener.
- Se dan actividades de *diseño*, donde se exploran, calculan, simulan y evalúan cuidadosamente distintas opciones para que el artefacto cumpla satisfactoriamente con su propósito y las restricciones legales, ambientales, de tiempo, dinero, esfuerzo u otros recursos que puedan aplicar o exigirse.
- Se dan actividades de *mantenimiento* del artefacto.
- Se dan actividades previas donde se busca precisar el propósito del artefacto, realizar el *análisis* su factibilidad y plantear una idea del costo de su desarrollarlo.
- Se dan actividades gerenciales para asegurar la buena marcha del proyecto de desarrollo del artefacto.

Todos estos tipos de actividades se han venido adaptando al software. Adicionalmente se han desarrollado varios *modelos de procesos de desarrollo*, tales como el modelo de la cascada, *Rational Unified Process (RUP)* y *Scrum*. Estos modelos proponen disciplinas de trabajo y formas de intercalar y llevar a cabo estas actividades. Escapa del alcance de este curso presentar y comparar tales modelos; por limitaciones de tiempo, introduciremos algunos aspectos de análisis y gestión, pero centraremos nuestra atención en las actividades y técnicas de diseño, construcción y verificación en el marco de un proceso *agil* de desarrollo. En particular desarrollaremos el curso con las siguientes preguntas en mente:

1. ¿Cómo se diseña software flexible? Esto es importante dado que algunas experiencias sugieren que los requerimientos de un software cambian a una tasa promedio de 1% *mensual* durante el período de desarrollo de un proyecto: así, al finalizar un proyecto de tres años, la tercera parte de los requerimientos finales surgieron o se modificaron en paralelo con el desarrollo del mismo.
2. ¿Cómo podemos desarrollar varios diseños que cumplan con unos requerimientos dados y cómo se puede escoger entre ellos?
3. ¿Qué podemos hacer para reducir el número de defectos en el software?
4. ¿Cómo facilitamos el mantenimiento futuro del software desarrollado?
5. ¿Cómo se puede desarrollar eficientemente software en equipo?

Lectura adicional

Lea y contraste esta introducción con la entrada de Ingeniería de Software que se encuentra en

como la Física, la Química o la Biología. Claramente existía y se podía desarrollar cierta fundamentación matemática, como lo mostraba claramente la teoría de complejidad y el desarrollo de la teoría que subyace las técnicas de compilación de lenguajes de programación. En sus clases en la Universidad Simón Bolívar, el profesor Nagib Callaos propuso que la diferencia de Ingeniería de la Computación respecto a las ingenierías más tradicionales era que se debía fundamentar en las ciencias sociales como Psicología, Sociología y Política, una aguda intuición de particular relevancia para el área de Sistemas de Información.

Wikipedia o con el capítulo introductorio de algún texto del área tal como lo son:

- Roger S. Pressman: *Software Engineering: A Practitioner´s Approach*. 7° Edición, McGraw-Hill, 2010
- Ian Sommerville: *Software Engineering.* 8° Edición, 2007 (también puede consultar la novena edición).
- Bernd Bruegge, Allen H. Dutoit: *Object-Oriented Software Engineering Using UML, Patterns and Java.* Prentice-Hall, 2010.