Acercamiento a la metodología y filosofía de trabajo Getting Real

Approach to the methodology and philosophy of work Getting Real

Luis Alberto Cabra Rativa

lacabrar@correo.udistrital.edu.co

Universidad Distrital Francisco José de Caldas

Tipo: Artículo de Investigación

Para citar este artículo: Para citar este artículo: Cabra L.A., (2015). Acercamiento a la metodología y filosofía de trabajo Getting Real. Revista TIA, pp 40-45

Fecha de recepción: 24 de noviembre de 2014
Fecha de aceptación: 5 de junio de 2015

Revista Digital TIA
Tecnología Investigación y Academia

Resumen

Getting Real es una metodología y filosofía de trabajo para el desarrollo de soluciones web de la empresa estadounidense Basecamp, que se ha ido construyendo desde finales de la década de los noventa y que rompe por completo metodologías de trabajo tradicional; además, se acerca de un modo más agresivo a las metodologías de desarrollo de software ágiles tratando con la realidad y no con abstracciones, buscando llegar a resultados de la forma más rápida deshaciéndose de lo innecesario.

Palabras clave: desarrollo de software, desarrollo web, getting real, metodologías ágiles.

Abstract

Getting Real is a work methodology and philosophy for web base solution development from American company Basecamp, which has been developing it since late 1990's and irrupts completely classic work methodologies and gets close aggressively to agile software methodologies working with reality and not with abstraction getting results faster decluttering.

Key words: agile methodologies, getting real, software development, web development.

I. Introducción

El desarrollo de aplicaciones y servicios web ha tenido un fuerte crecimiento en la última década y una buena acogida por parte de los usuarios que pueden acceder a ellos a través de múltiples dispositivos electrónicos que tienen acceso a la red, ya no se manejan versiones anuales que se entregan en medios como CD o DVD, ni un manejo de versiones hacia el cliente. Esto requiere un enfoque diferente de metodología de desarrollo, que ha venido haciendo 37SIGNALS, como se le conocía a la empresa antes de ser renombrada bajo su producto estrella BASECAMP [1]. Desde el año 1999 se ha refinado y comprobado una forma de trabajo completamente distinta a la tradicional, donde la planeación a largo plazo y la documentación son esenciales para el éxito de los proyectos.

El acercamiento de la metodología y filosofía de trabajo Getting Real, versa sobre lo real y no en lo abstracto, llegando a una forma más pequeña, más rápida y mejor de crear software, olvidándose de cosas que abstraen la realidad, como gráficos, esquemas, tablas y creando objetos reales. Getting Real es menos, menos masa, menos software, menos características, menos papeleo, menos de todo lo que no es esencial.

Getting Real comienza con la interfaz, las pantallas reales que el usuario va a usar. Comienza con lo que el cliente experimenta realmente y se construye hacia atrás a partir de ahí. Getting Real entrega solo lo que los clientes necesitan y elimina todo lo que no. [2]

II. 37Signals

Signal es una empresa ubicada en Chicago, Illinois, cofundada por Jason Fried, Carlos Segura y Ernest Kim en el año 1999, la cual ofrecía inicialmente el servicio de rediseño de sitios web; en el año 2003 desarrollan su primer producto como respuesta a no encontrar una herramienta de manejo de proyecto que los satisficiera, BASECAMP. Más adelante desarrollan otra serie de aplicaciones web, pero es el framework RUBY ON RAILS, creado por David Heinemeier Hansson (uno de los desarrolladores de la empresa) el otro desarrollo que los hace resaltar en la industria, cuando es liberado como un proyecto de código abierto a la comunidad en el año 2005. [3]

Desde sus inicios comparten sus vivencias en el blog Signal Vs Noise, en el que comienzan a tratar temas que consolidan inicialmente en el libro *Getting Real* en el año 2006 y luego en 2010 en el libro *Rework*, ambos de las manos de Jason Fried y David Hansson.

A la fecha, más de tres millones de personas usan sus productos y cientos de miles son asiduos lectores de sus publicaciones. [4]

II. Getting real

La metodología Getting real es trata varias partes de las etapas de desarrollo: el inicio de un

proyecto, los procesos, la organización, el personal, la selección de características, la interfaz de usuario, el código, la documentación, el lanzamiento y soporte; todos con algo en común, mantenerlo de la forma más sencilla y liviana eliminando lo innecesario.

Ahora bien, se nos recomienda iniciar construyendo algo que sea para nosotros, y no hay mejor forma de entender al cliente si el cliente somos nosotros, así como ellos desarrollaron una aplicación para la gestión de proyectos porque la necesitaban. Cuando se soluciona un problema propio uno realmente se preocupa por hacerlo bien.

Iniciar con los recursos justos es uno de sus puntos al inicio de un proyecto, no preocuparse por tener las herramientas más costosas, ya que estas no son las que hacen finalmente la diferencia; muchas veces estar restringidos implica que se encuentren soluciones creativas, como los desarrolladores de la aplicación para teléfonos inteligentes o Instagram. Al ver que la resolución de la mayoría de las cámaras de los celulares era baja decidió incluir filtros en sus fotos y terminó siendo una característica importante. [5]

No se puede perder tiempo esperando a lanzar la mejor versión de un producto, hay que priorizar qué es lo más importante del producto y enfocarse en ello, siendo flexive en que más adelante se irá mejorando el mismo haciendo pequeñas entregas, muy parecido a la metodología XP, trabajar en iteraciones. De esta forma también se celebran pequeñas victorias y se empieza a desarrollar un *momentum* que animará al equipo.

También se recomienda ser rápido en los procesos, desde que se tiene una idea, se prototipa y se programa, no hay necesidad de modelos muy detallados cuando se puede invertir ese tiempo en tareas más productivas. No hay que abundar en los detalles al inicio del proyecto, esto solo causa demoras y reduce la oportunidad de éxito.

Es importante tener un equipo pequeño al inicio, que convenientemente podría estar integrado por un diseñador, un programador y una persona con conocimiento en ambas áreas, al igual que estar restringido por el personal se intentará priorizar sin desgastar; más adelante si es requerido se irá ampliando el equipo, pero sin crecer sin medida. Harvard no es la mejor universidad por tener cedes por todo Estados Unidos. [6]

Al ser pequeños se estará más cerca del cliente, esto hace que se puedan comunicar de una forma directa y personal, se podría usar un lenguaje familiar y darle una voz humana al producto. No se puede perder tiempo en problemas que no son aún un problema, no se puede pensar en contratar más personal del que se necesita, ni pensar en una infraestructura para

miles de clientes si apenas se tienen unos pocos. Hay que tomar decisiones cuando es el momento justo y se tiene acceso a toda la información que se necesita, mientras tanto es necesario prestarle atención a las cosas que requieren inmediato cuidado, pensar en ello como un plan no es realmente planear, es intentar adivinar.

Es indispensable satisfacer un nicho de mercado, si se intenta satisfacer a una gran cantidad de clientes con sus necesidades no se podrá realizar dicha tarea; cuando se enfoca en un grupo pequeño es más fácil hacerles llegar información sobre el producto y tener retroalimentación. Aunque al tener dicha retroalimentación es importante estar siempre atento en que no se puede hacer todos los cambios o mejoras que son solicitados, hay que responder no como primera instancia porque varias de estas peticiones no son realmente mejoras y lo que se quiere es siempre mantener todo lo más sencillo y liviano posible.

El mejor sustituto para probar el desarrollo de nuestro producto son los clientes reales, así que esta es una de las ventajas de lanzar pronto y ver cómo se comporta todo en un ambiente verdadero.

Las reuniones son tóxicas, una reunión de diez personas por una hora no es otra cosa que diez horas de productividad perdidas o hasta quince si se tiene en cuenta el tiempo que tarda una persona en volver a concentrarse en lo que debe hacer; esto sucede porque no se tiene una agenda clara, se programa para demasiadas personas con un periodo de tiempo largo y además se programan demasiadas reuniones. Se recomienda empoderar al personal para que tome decisiones por cuenta propia, de esta forma no se desgasta en alguna reunión esperando una confirmación de algo que pudo haber decidido solo en un inicio.

III. Interfaz de usuario

Lo primero que se debe desarrollar es la interfaz del usuario, porque esta es la parte más liviana del desarrollo y por lo mismo consume menor tiempo y esfuerzo, de esta forma también evaluamos si lo que queremos tiene sentido, si va a ser fácil de usar o tiene características innecesarias que ya no se deben programar, diseñando inicialmente la interfaz mantiene el desarrollo flexible.

Se debe saber cuál va a ser el epicentro del producto porque es con lo que va a interactuar más el cliente, por lo mismo va a ser el elemento crítico dentro del desarrollo; de igual forma, pensar en los tres estados que puede llegar a ver el cliente en la interfaz, la pantalla en blanco, como cuando recién abrimos una cuenta de correo, la pantalla con un uso regular y la pantalla de error.

La pantalla en blanco se puede usar para generar expectativa al cliente, ya que es la

primera impresión y generalmente se pierde, allí se pueden tener tutoriales de inicio de uso o muestras de cómo lucirá cuando se empiece a usar, creando expectativas se reduce la intimidación, frustración y confusión.

Se tiene que pensar que no todo puede salir bien y se debe estar preparado cuando las cosas van mal, para ello se encuentra el diseño defensivo, que no es otra cosa que estar preparado para ese 10% de posibilidad que las cosas fallen. [7]

IV. Código

El código debe ser también tan sencillo como sea posible, cada vez que aumenta el código aumenta exponencialmente su complejidad, la forma de evitar esto se relaciona con tener menos software, pero esto solo significa menos características. La clave está en disminuir un problema grande en uno pequeño y resolver el 80% del problema con el 20% de esfuerzo [8]. Menos código significa que es más fácil de mantener, reduce los costos al ser adaptable fácilmente, menos defectos y menor soporte.

Hay que hacer felices a los desarrolladores, no hay que escoger estándares, prácticas y métricas de la industria porque sí, que ellos escojan el lenguaje con el que se sientan cómodos, así van a disfrutar y llegar a soluciones creativas, como sucedió con el desarrollo del framework RAILS; además no deben trabajar más de 40 horas a la semana.

Se piensa que la deuda solo es económica pero puede venir de otras formas, se debe mejorar código poco eficiente para saldar esa deuda. Hay que estar abiertos a que los clientes puedan hacer también desarrollo sobre el producto, esto se logra con APIs.

V. Documentación

No se deben escribir documentos muertos que consumen el tiempo en su generación, pero son desechados rápidamente, estos documentos restan flexibilidad al desarrollo, no evolucionan y se convierten en una carga, un documento de las especificaciones de una característica que más adelante es desechada es una mala idea, en vez de esto en una sola página se debe escribir una historia en lenguaje sencillo sobre lo que el producto debe cumplir, de ahí se comienza a trabajar en la interfaz.

Se debe prevenir el exceso de papeleo, a menos que un documento se haga para algo que realmente se va a producir. Hay que desarrollar, no documentar, si algo se debe explicar es mejor acudir a prototipos sencillos. Los documentos que viven a parte de la aplicación no sirven y no llevan a ningún lado. Muchos de estos documentos no van a ser leídos.

Se recomienda escribir historias, no detalles. Hay que escribirlas de la forma más humana posible, como si se tratara de una conversación real, no tiene que ser un ensayo. La idea principal es remover la abstracción y trabajar con lo real.

VI. Compartir

Otra forma de publicidad para el producto que se desarrollo es compartir, pueden ser experiencias de desarrollo, como con los libros y blog de 37 Signals, también dando pequeñas porciones gratis del producto para enganchar a posibles clientes, algo muy parecido al modelo *freemium*.

VII. Conclusiones

La diferencia del resultado de las cosas se halla en las personas y su ejecución, la metodología parece un poco fuera de la norma, pero si se enmarca solo en el desarrollo de soluciones web tiene sentido implementarla, ya que se puede realizar cantidad indefinida de entregas y estar mejorando sobre estas mismas y preocuparse por crear comunidad alrededor del producto.

VIII. Referencias

[1] J. Fried, Two big announcements, 5 de febrero de 2014, [en línea]. Consultado el, disponible en: http://37signals.com/

[2] 37 Signals, "Getting Real", Chicago, 2006, Introduction, Ruby on Rail*s*, 17 de noviembre de 2014, [en línea]. Consultado el, disponible en: http://en.wikipedia.org/w/index.php?title=Ruby_on_Rails&oldid=634254845

[3] J. Fried and D. Hansson, *Rework*, Chapter 1 – First, Chicago 2010

[4] K. Rose, Episode 16 w/ Kevin Systrom, 30 de junio de 2012, [en línea]. Consultado el, disponible en: http://foundation.bz/16/

[5] J. Fried, D. Hansson, *Rework*, Chapter 9-Hiring, Chicago, 2010.

[6] 37 Signals, *Getting Real*, Interface Design, Chicago, 2006.

Pareto principle, 3 de noviembre de 2014, [en línea]. Consultado el, disponible en: http://en.wikipedia.org/w/index.php?title=Pareto_principle&oldid=632327346