Recibido 7 Sep 2016 *ReCIBE, Año 6 No. 1, Mayo 2017*
Aceptado 15 Mar 2017

Aplicando Scrum y Prácticas de Ingeniería de Software para la Mejora Continua del Desarrollo de un Sistema Ciber-Físico

Applying Scrum and Software Engineering Practices to Continuously Improve the Development of a Cyber-Physical System

Gabriela Sobrevilla[1]
gabriela.sobrevilla@cimat.mx

José Hernández[1]
pphdez@cimat.mx

Perla Velasco-Elizondo[2]
pvelasco@uaz.edu.mx

Silvia Soriano[3]
silvia_soriano@lasec.mx

[1] Centro de Investigación en Matemáticas, CIMAT A.C., Unidad Zacatecas, México.
[2] Universidad Autónoma de Zacatecas, Unidad Académica de Ingeniería Eléctrica, México.
[3] Lasec Telecomunicaciones S.A.P.I de C.V., México.

Resumen: En la actualidad, presiones competitivas fuerzan a las compañías a desarrollar productos y servicios en el menor tiempo posible. Una alternativa para lograrlo es el uso de métodos de desarrollo ágiles como Scrum. A pesar de que Scrum ha sido usado exitosamente en muchos dominios de aplicación, no existe un claro entendimiento sobre cómo aspectos específicos del desarrollo de Sistemas Ciber-Físicos deben ser tratados cuando se utiliza este método sin que afectar la agilidad. En este artículo se describen algunos problemas técnicos y organizacionales que surgieron durante el desarrollo de un Sistema Ciber-Físico y cómo se resolvieron incorporando algunas prácticas de ingeniería de software. Compartiendo esta experiencia se pretende ayudar a otros equipos de desarrollo a ganar un mejor entendimiento sobre algunos de los problemas relacionados a este tipo de sistemas y cómo enfrentarlos con las prácticas descritas.

Palabras clave: Sistemas Ciber-Físicos, Scrum, Arquitectura de Software, Métodos Ágiles.

Abstract: Competitive pressures force companies to develop products and services in less time. An alternative to achieve this is using agile methods such as Scrum. Although Scrum has been successfully utilized in many application domains, there is not a clear understanding about how specific aspects of the development of Cyber-Physical Systems should be addressed when using Scrum without lost of agility. This article describes some technical and organizational problems that arose during the development of a Cyber-Physical System and how they were addressed by incorporating some software engineering practices. Sharing this experience pretends to help development teams to gain a better understanding of some of the problems related to this type of systems and how to deal with them by adopting the described practices.

Keywords: Cyber-Physical Systems, Scrum, Engineering Practices, Agile Methods.

1. Introducción

Los métodos ágiles de desarrollo de software son procesos basados en iteraciones cortas las cuales generalmente producen un entregable correspondiente a una versión operable y evolutiva del sistema. Los métodos ágiles han ganado popularidad en los últimos años debido a la mejor capacidad de respuesta que presentan con respecto a los métodos tradicionales, a las demandas de mercado. En relación a esto, The Chaos Report (The Standish Group International, Inc., 2016) indica que la tasa de proyectos fallidos en ambientes de desarrollo ágil ha sido en 2016 del 9%, mientras que en los métodos tradicionales este porcentaje sube hasta el 27%. El reporte indica además, que utilizar un enfoque ágil de desarrollo incrementa a un 91% las probabilidades de finalizar un proyecto sin que este sea fallido.
Scrum es uno de los métodos ágiles más populares en la actualidad y se ha convertido en el método ágil de preferencia de equipos de desarrollo de software superando a otros que habían dominado la escena (Meyer, 2014). El proceso de Scrum está definido por una serie de iteraciones de duración fija llamadas sprints, que pueden durar desde una hasta cuatro semanas según las preferencias y/o necesidades del equipo. Scrum considera un equipo auto organizado y multifuncional integrado desde tres hasta nueve miembros. Dentro de este equipo, llamado Equipo Scrum, se definen los siguientes roles: el Dueño del Producto, quien es la voz que se encarga de que el equipo entienda las necesidades de los usuarios finales del sistema así como de otros involucrados. El Scrum Master que actúa como mentor encargándose de que Scrum sea entendido y llevado a la práctica como lo marcan sus principios, así como de eliminar los impedimentos que surjan durante el proyecto. El Equipo de Desarrollo que es el responsable de construir el producto de software. Debido a su naturaleza multifuncional, además de desarrolladores el equipo puede incluir miembros con diferentes perfiles como analistas, diseñadores o personal de pruebas aunque no reconoce roles entre miembros del equipo. El proceso de Scrum además involucra una serie de eventos, artefactos y reglas que los relacionan, información detallada sobre estos puede ser encontrada en la Guía de Scrum (Sutherland & Schwaber, 2013) diseñada por sus creadores.

Otro elemento importante de este caso son los Sistemas Ciber-Físicos o CPS (Cyber-Physical Systems), que emergen de la integración de componentes de software, hardware y procesos físicos conectados por una infraestructura de telecomunicaciones. En los CPS las operaciones son monitoreadas, coordinadas y controladas por un núcleo de software, hardware y telecomunicaciones (Rajkumar, Lee, Sha, & Stankovic, 2010). A menudo su desarrollo recae en arquitecturas de Internet de las Cosas, que permiten recolectar y procesar datos de componentes heterogéneos.

El desarrollo de CPS requiere colaboración de varios equipos de diversas disciplinas como ingeniería de software, electrónica y telecomunicaciones. Así, una simple instanciación del proceso Scrum a menudo no es suficiente para mantener la agilidad entre los equipos involucrados en su desarrollo.

En este artículo se describen las experiencias de un equipo que utilizó Scrum en el desarrollo de un CPS para la localización de vehículos en minas subterráneas. Específicamente se describen algunos problemas técnicos y organizacionales que surgieron durante su desarrollo y cómo fueron resueltos utilizando algunas prácticas de ingeniería de software. Específicamente, prácticas de arquitectura de software, comunicación con los involucrados y aseguramiento de la calidad del software. Compartiendo esta experiencia se pretende ayudar a otros equipos de desarrollo a ganar un mejor entendimiento sobre algunos de los problemas relacionados a este tipo de sistemas y cómo enfrentarlos con las prácticas descritas.

El artículo se ha organizado de la siguiente forma: en la Sección 2 se describen los detalles relacionados al Sistema de Localización Subterránea, que es el CPS de interés. En la sección 3 se mencionan los problemas experimentados así como las prácticas adoptadas para resolverlos. Finalmente se termina en la sección 4 con una discusión y conclusiones sobre el caso y el trabajo futuro.

2. Sistema de Localización Subterránea

El Sistema de Localización Subterránea (SLS) fue un proyecto desarrollado por una PYME (*Acrónimo utilizado para nombrar a la pequeña y mediana empresa*), dedicada al negocio de las telecomunicaciones en México. El objetivo del proyecto es proveer una solución accesible y con soporte especializado, para automatizar algunos procesos en la industria minera.

Como primer paso, la compañía decidió desarrollar un Producto Mínimo Viable o MVP (*Minimum Viable Product*) para distribuirlo entre algunas compañías mineras y así identificar posibles compradores. El concepto de Producto Mínimo Viable especifica una estrategia por la cual se pueden identificar las características de un producto que los clientes encontrarán de valor (Moreira, 2009). El MVP debería tener suficiente funcionalidad para permitir recolectar la mayor cantidad de información posible del uso del producto y sus potenciales compradores.

El SLS involucra un conjunto de componentes de hardware y software desarrollados internamente por la compañía que, mediante una infraestructura de telecomunicaciones provee la localización en tiempo real de vehículos bajo

tierra como cargadoras o camiones de remolque, entre otros. La Figura 1 muestra de manera general los elementos del SLS.

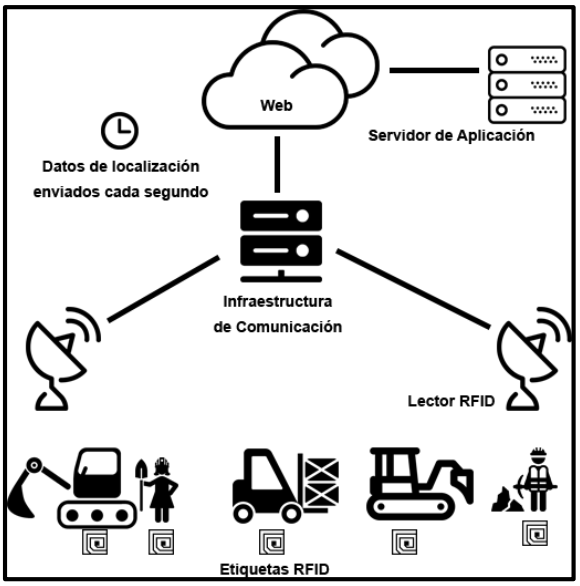

Figura 1. *Elementos del SLS.*

El SLS utiliza un conjunto de dispositivos de Identificación de Radiofrecuencia (*RFID tags*) cuya función es intercambiar información con una serie de lectores RFID a través de ondas de radio. Estos dispositivos cuentan con una fuente de poder local y pueden operar a cientos de metros de un lector RFID. Cada vehículo que circula en los túneles de la mina tiene asociado uno de estos dispositivos.

Los RFID transmiten datos de ubicación que son recibidos por los lectores RFID instalados en la red de túneles. Luego, los datos son enviados un servidor en el que se encuentra instalado un "artefacto de software" que los procesa para su visualización en tiempo real. Se utilizará el término artefacto de software para referirse de manera simple al conjunto de componentes de software del SLS. El artefacto de software es utilizado por uno o más operadores como soporte para la toma de decisiones.

Uno de los retos más importantes del SLS era la confiabilidad, la transmisión de datos que realiza el RFID cada segundo debe ser procesada de manera que no se pierda ningún paquete de datos. Esto con la finalidad de proporcionar una imagen de la ubicación de los vehículos en tiempo real al operador. Así, en caso de siniestro se podría localizar el vehículo y las personas de manera más rápida y precisa.

Otro requisito, éste relacionado a la escalabilidad, era que el sistema debería soportar cientos de dispositivos de manera concurrente sin degradarse. Dado que del SLS pueden depender vidas humanas, se esperaba que también

tuviera una alta disponibilidad. Sin embargo para este atributo una medida precisa no había sido especificada por el cliente o los usuarios finales.

Para dar una perspectiva de la complejidad del SLS, en el momento en que se realizó este análisis (sprint 10 del desarrollo), el visor 3D del sistema estaba conformado por 23 componentes de software que sumaban 54,485 líneas de código correspondientes al 10% del total del sistema aproximadamente. Se esperaba que al término del desarrollo el SLS estaría conformado por alrededor de 500,000 Líneas de Código O LOC (*Lines Of Code*). El cálculo de LOC fue realizado utilizando la herramienta SLOCCount (Wheeler, 2016) que permite obtener la cantidad de líneas físicas de código fuente y sus componentes.

2.1 El Contexto del Proyecto

El SLS fue desarrollado utilizando Scrum. Un esfuerzo fallido previo dió a la compañía algunas experiencias que determinaron en gran medida la selección de este enfoque. En el proyecto anterior, mientras el equipo de hardware interno desarrollaba los dispositivos RFID, la construcción del SLS fue asignado a una compañía externa ubicada en una ciudad diferente a las instalaciones de la empresa en cuestión. La compañía contratada adoptó un enfoque tradicional, que aunado a la falta de experiencia en el dominio de aplicación, la poca comunicación con el equipo de hardware y entregas esporádicas, contribuyeron a que algunos requerimientos fundamentales no fueran completamente entendidos. Debido a lo anterior, el prototipo construido después de un periodo de 10 meses de desarrollo presentó graves problemas de confiabilidad y escalabilidad al ser llevado a un ambiente real de operación.

Luego de esta experiencia la compañía decidió desarrollar el SLS por su cuenta en el plazo de un año, adoptando un enfoque ágil utilizando Scrum. Entre los factores determinantes para utilizar Scrum en el desarrollo de este CPS se pueden mencionar: el auge que ha tenido este método en México en los últimos años, el impulso que le ha dado el gobierno en la comunidad tecnológica para la formación de capital humano (PROSOFT, 2016), así como la ventaja que representan sus principios sobre entregas y retroalimentación frecuentes producto del desarrollo en iteraciones cortas e incrementales que involucra. Para lograr su objetivo, la compañía contrató a un equipo de desarrollo con la finalidad de crear el nuevo SLS. La experiencia de los miembros iba desde 2 hasta 10 años en el desarrollo de software, siendo los más experimentados el Scrum Master/Arquitecto de Software y el Dueño del Producto.

El equipo de hardware que fue el mismo involucrado en el desarrollo anterior, estaba conformado por 10 integrantes, incluyendo al líder del equipo. Se

desconoce su metodología de trabajo y experiencia de los miembros, ya que aunque ambos equipos pertenecían a la misma compañía, políticas organizacionales limitaban la comunicación e intercambio de información entre ellos. Cada equipo trabajaba de manera separada para mantener la confidencialidad del proyecto, lo que complicaba establecer cualquier mecanismo de sincronización. Para realizar adecuadamente la puesta en marcha de Scrum, el Scrum Master entrenó a los miembros del equipo de desarrollo, en el uso de este método. El Director general quien era el principal interesado participó también como *Dueño del Producto*, manteniendo comunicación constante con cada equipo ya que dadas las políticas mencionadas esta fluía principalmente a través de él. Con excepción del Scrum Master, todos los miembros del equipo aplicaron Scrum por primera vez.

2.3 El Proyecto y los Involucrados

Dado que el SLS es una solución tecnológica que busca explorar una oportunidad de mercado, el sistema no ha sido desarrollado para un cliente en particular. A pesar de que se contó con un Dueño del Producto desde el inicio, este no sería usuario del software ni conocía los procesos que el software debía automatizar de manera profunda. Así, con la finalidad de recabar más requerimientos de los usuarios finales, el Director General/Product Owner se encargó de visitar a posibles clientes a través del país acompañado del Líder de Proyecto/Product Owner quien después pasaría estos requerimientos al Equipo de Desarrollo.

Además, el Director General aprovechó la versión anterior del SLS para recabar nuevos requerimientos y utilizarlo como prueba en diferentes empresas mineras de la región, que eran compradores potenciales. Esto le permitiría probar el software en ambiente real de operación y obtener retroalimentación frecuentemente de los usuarios finales. Una vez que la nueva versión del SLS pudo suplir a la versión inicial, se hizo el reemplazo en las empresas mineras donde se encontraba.

El Dueño del Producto decidió que, de manera regular y casi siempre al final de cada sprint, los miembros del Equipo de Desarrollo y de Hardware acudirían con los usuarios finales para actualizar la versión del software y obtener retroalimentación sobre el comportamiento del SLS.

3. Problemas Experimentados y Prácticas Adoptadas

A pesar de que la decisión de desarrollar el software y hardware en la compañía ayudó, el uso de Scrum no fue tan ágil como se esperaba cuando algunos problemas aparecieron. La Figura 2 muestra el momento en que estos problemas aparecieron en el periodo comprendido por los primeros 10 sprints del proyecto.

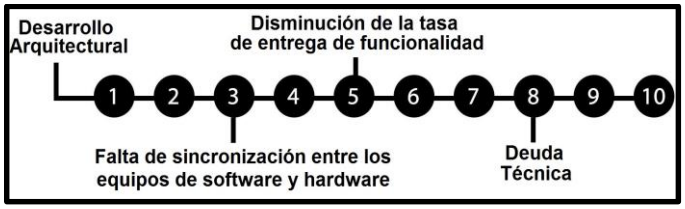

***Figura 2**. Problemas que aparecieron en los primeros 10 sprints del desarrollo del SLS.*

En las siguientes secciones es descrito cómo estos problemas fueron enfrentados usando las siguientes prácticas de ingeniería:

Problemas Enfrentados	Prácticas Adoptadas
Desarrollo Arquitectural	Hacer el trabajo de arquitectura explícito anexando un evento al flujo de Scrum.
Desarrollo Arquitectural	Realizar trabajo de arquitectura y refactorización de código explícito a través de historias de usuario.
Falta de sincronización de los equipos de software y hardware	Sincronizar equipos de software y hardware ubicándolos en el mismo lugar y estableciendo un nuevo canal de comunicación.
Falta de sincronización de los equipos de software y hardware	Introducir integración continua al proceso de desarrollo.
Falta de sincronización de los equipos de software y hardware	Introducir pruebas de usuario cada tercer día.
Disminución de la tasa de entrega de funcionalidad	Hacer el trabajo de arquitectura explícito a través de historias de usuario.
Deuda Técnica	Incluir en la Definición de Terminado, un requisito de documentación de cualquier cambio significativo en la arquitectura derivado del proceso de desarrollo.

***Figura 3.** Prácticas implementadas para mitigar los problemas surgidos durante el desarrollo del SLS.*

3.1 Desarrollo Arquitectural

El propósito del desarrollo arquitectural de software es proveer una estructura para el sistema a construir y reducir riesgo técnico. Sin embargo, el valor del desarrollo arquitectónico y el desarrollo ágil ha sido sujeto a debate por mucho tiempo. Un aspecto clave de los métodos ágiles es su principio dirigido a entrega de incrementos de software funcionales en un periodo corto de tiempo. Existe una postura muy popular en los practicantes de los métodos ágiles de evitar el gran diseño por adelantado o *"big design up front"* (Meyer, 2014), porque el evitarlo permite a los equipos a enfocarse en diseñar e implementar solo la funcionalidad que provee valor inmediato (y no todo el diseño). Por otra parte, adoptar esta postura en el desarrollo de un sistema complejo puede ser un factor determinante para incurrir en deficiencias arquitecturales o deuda técnica. Dada la naturaleza de este dominio de aplicación, donde el corto tiempo para desarrollar el producto y la habilidad de adaptarse al cambio de requerimientos eran una necesidad competitiva, fue de vital importancia la definición de un conjunto de estructuras que permitieran forjar las bases del sistema y elegir el conjunto de tecnologías adecuado para implementarlas.

Considerando la poca experiencia del equipo de desarrollo en el dominio de aplicación, este se dio cuenta que definiendo una arquitectura para el SLS tomaría más de un sprint. Así, con el respaldo del Dueño del Producto, el equipo decidió incluir en el proceso Scrum el desarrollo de arquitectura como un evento que se llevaría a cabo antes de la reunión de planificación del sprint. A este evento se le llamó Desarrollo de la Arquitectura. El objetivo del evento fue diseñar e implementar un *architecture runway* (Leffingwell, 2008) que permitiera el desarrollo de funcionalidades a corto plazo sin incurrir en un esfuerzo grande de refactorización posterior.

El desarrollo de un *architecture runway* fue altamente dirigido por los drivers arquitectónicos del proyecto: requerimientos de alto nivel, atributos de calidad así como restricciones técnicas y de negocio (Bass, Clements, & Kazman, 2012). Durante la construcción del *architecture runway* el Arquitecto de Software orientó al equipo de desarrollo, invitándolo a probar en código las decisiones tomadas con la finalidad de que no se vieran fuertemente afectadas en el futuro. La salida de este evento fue una primera e incompleta versión de la arquitectura, que proveía un conjunto básico de servicios para soportar la recepción de altos volúmenes de mensajes provenientes de una variedad de dispositivos.

3.2 Disminución de la Tasa de Entrega de Funcionalidad

Tan pronto como una parte de la arquitectura estuvo disponible, el equipo de desarrollo empezó a construir funcionalidad del SLS. Con la finalidad de responder a la presión del mercado cada miembro del equipo empezó a

desarrollar características específicas utilizando la infraestructura arquitectural disponible. Aunque la velocidad del equipo era estable, en el quinto sprint fue evidente que había muchas características incompletas (véase Figura 4). Debido a esto el Dueño del Producto no estaba muy satisfecho con la situación, y de nuevo, la presión sobre el Equipo de Desarrollo aumentó.

Contar con una arquitectura casi completa habría contribuido a mantener la tasa de entrega de funcionalidades durante los primeros sprints sin embargo, no sucedió en este caso. Así, la necesidad de promover el desarrollo de arquitectura para habilitar la implementación de funcionalidades, llevó al equipo a cambiar a una manera desorganizada de trabajo entre el desarrollo de ambos, la arquitectura y la funcionalidad. Además su poca experiencia en el dominio de aplicación influyó en que subestimaran el trabajo de arquitectura.

Para contrarrestar esta problemática el Equipo de Desarrollo decidió hacer el trabajo de arquitectura explícito y para ello decidieron especificarlo utilizando historias de usuario. De esa manera el equipo pudo obtener estimaciones más precisas del trabajo de arquitectura y también comunicó de forma adecuada la importancia y complejidad de su desarrollo al Dueño del Producto. El trabajo de refactorización de código fue también especificado usando este enfoque de hacerlo explícito a través de historias de usuario. El desarrollo de arquitectura y refactorización de código contribuyeron a contrarrestar la deuda técnica. Estas prácticas fueron de valor porque como se muestra en la Figura 4, el trabajo de ambos fue permanente durante este periodo del proyecto.

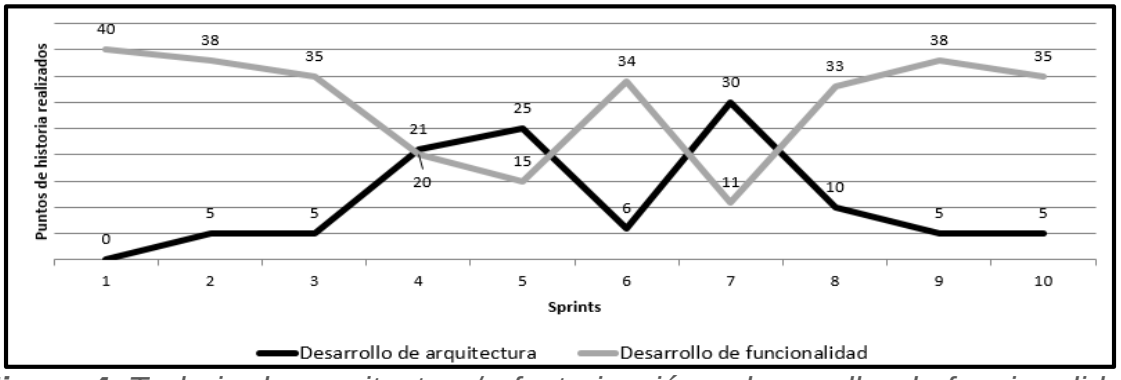

Figura 4. *Trabajo de arquitectura/refactorización y desarrollo de funcionalidad en el periodo de los primeros 10 sprints del desarrollo.*

3.3 Falta de Sincronización de los Equipos Software y Hardware

Los CPS involucran componentes de software y hardware, los cuales son a menudo desarrollados por equipos separados. Idealmente, en Scrum estos equipos deberían estar trabajando en el mismo lugar y estar altamente

sincronizados. En este proyecto los equipos estuvieron hasta el tercer sprint, localizados en diferentes instalaciones e incomunicados. Además, el equipo de hardware estuvo utilizando una forma de trabajo diferente en la cual el único punto de sincronización con el equipo de desarrollo era una reunión semanal. Los lanzamientos del equipo de hardware ocurrían en esa reunión. Sin embargo la planificación de estos lanzamientos no estaba propiamente manejada lo que propiciaba retrasos frecuentes.

Debido a esta falta de sincronización, el equipo de desarrollo tuvo que construir muchas funcionalidades simulando los componentes de hardware en lugar de utilizar los componentes reales. Adicionalmente, la falta de acceso rápido y sencillo a los componentes de hardware y la presión de mercado promovieron la implementación de los componentes de hardware simulados usando información incompleta sobre su comportamiento. El Equipo de Desarrollo se percató de que estos factores contribuyeron a tener un número significante fallos en las revisiones del sprint.

Para subsanar el problema, el Equipo de Desarrollo y el equipo de hardware fueron colocados en un mismo edificio en pisos contiguos. A demás, el Equipo de Desarrollo inició prácticas de integración continua y promovió la realización de pruebas de usuario de manera interna como mecanismos para el aseguramiento de la calidad del software. La integración continua es una práctica de desarrollo de software donde los miembros del equipo integran su trabajo con frecuencia (...) cada integración es comprobada por una construcción automática para detectar errores tan rápido como sea posible (Fowler, 2006). Los procesos de integración fueron programados en el trabajo diario, lo que ayudó a soportar el trabajo de pruebas de características funcionales cada tercer día. De esta forma se empezaron a encontrar errores de manera más temprana durante el desarrollo y las entregas de software en la revisión del sprint fueron más efectivas.

Finalmente, el Scrum Master gestionó la comunicación del Equipo de Desarrollo hacia el líder del equipo de hardware al preguntar en las reuniones diarias si alguno de los miembros necesitaba ayuda o información específica del otro equipo. Así, en caso de ser requerido y una vez notificado el líder del equipo de hardware, los miembros de ambos equipos se comunicaban directamente con quien se necesitara obteniendo la ayuda esperada con mayor rapidez. Aunque ambos equipos seguían trabajando de manera confidencial sus partes del proyecto, se comunicaban entre sí con más frecuencia y la falta de comunicación dejó de ser un impedimento constante.

3.4 Deuda Técnica

A pesar de la disciplina del equipo de desarrollo para actualizar los diseños arquitecturales, después del quinto sprint las presiones de mercado los llevaron

a tomar un nuevo hábito donde creaban, actualizaban y diseñaban sólo si una nueva funcionalidad era activada. Sin embargo, en lugar de actualizar el diseño principal empezaron a generar pequeños bocetos de diseños para consulta rápida. Esta práctica fue útil y suficiente durante varios sprints.

En el octavo sprint (véase Figura 2) nuevos miembros se unieron al Equipo de Desarrollo. Cuando el equipo estaba familiarizando a los nuevos miembros con el sistema y su arquitectura, se dieron cuenta de lo difícil que resultaba utilizar un diseño principal desactualizado en conjunto con todos estos pequeños bocetos de diseños derivados de las características funcionales y no funcionales que iban implementando.

Para subsanar este problema, la *definición de terminado* fue utilizada como mecanismo para garantizar que la documentación de la arquitectura se mantuviera actualizada a través del tiempo. En el contexto ágil, la definición de terminado es un acuerdo por medio del cual el equipo provee transparencia y una forma de manejar la calidad del software (Visser, Rigal, Wijnholds, & Lubsen, 2017). Considerando lo anterior, fue acordado que cualquier cambio en la arquitectura o refactorización que involucrara un cambio significante en la arquitectura debería ser documentado antes de que cualquier otro miembro lo utilizara.

4 Discusión y Conclusiones

Responder adecuadamente a las presiones de mercado y su competencia depende altamente de identificar continuamente un conjunto de problemas y sus causas, así como adoptar las prácticas correspondientes para contrarrestarlos. Esto no es un trabajo simple porque cada proyecto y equipo de desarrollo es diferente. Adicionalmente, la adopción de algunas prácticas podría involucrar desventajas que es importante identificar y evaluar su impacto.

En este proyecto la decisión de hacer el trabajo de arquitectura explícito añadiendo un evento al flujo de Scrum fue altamente influenciado por el conocimiento de la experiencia de un desarrollo previo fallido. Sin esta experiencia, el equipo podría haber adoptado la postura de evitar el *“big design up front”*. El *architecture runway* producido en combinación con el hecho de hacer el trabajo de diseño arquitectural subsecuente explícito con historias de usuario fueron factores que soportaron la entrega de características (funcionales y no funcionales) de una manera más predecible.

Además, se detectaron y arreglaron algunos problemas de comunicación al reunir a los equipos en el mismo lugar y estableciendo un nuevo canal de

comunicación. Por otro lado, tratar con entregas frecuentes e incrementales fue complicado al inicio, esta situación desmotivó al equipo y los miembros quisieron abortar su práctica. Afortunadamente en un periodo corto empezaron a percibir el valor de las mismas.

La arquitectura de software está presente en todos los sistemas, y el hecho de que un proyecto se desenvuelva en un ambiente ágil no implica que no se deba hacer trabajo de arquitectura. Algunos autores como Brown (2016) y Fairbanks (2010) apoyan la realización de suficiente "*big design upfront*" como para que se habilite la entrega de valor al cliente y a su vez, no se caiga en el extremo de realizar trabajo de arquitectura sin valor.

Como trabajo futuro se encuentra realizar una revisión a la literatura sobre prácticas de arquitectura de software para CPS que ha demostrado ser eficaces aplicando métodos ágiles. Adicionalmente, se debe evaluar la posibilidad de adoptar otras prácticas de ingeniería de software como "*release train*" (Leffingwell & Reinertsen, 2012) para lograr una mejor sincronización de los equipos. En este mismo contexto, se podría considerar el proceso descrito por Wagner (2014) en el que propone una variación de Scrum para CPS que se busca manejar la integración de los equipos de hardware y software involucrados en el desarrollo de este tipo de sistemas.

Referencias

Bass, L., Clements, P., & Kazman, R. (2012). Software architecture in practice (1st ed.). Upper Saddle River, N.J.: Addison-Wesley.

Brown, S. (2016). Software Architecture for Developers (1st ed., p. 84). Leanpub.

Fairbanks, G. (2010). Just Enough Software Architecture (1st ed., p. 10). Colorado: Marshall & Brainerd.

Fowler, M. (2006). Continuous Integration. martinfowler.com. Retrieved 28 February 2017, from https://www.martinfowler.com/articles/continuousIntegration.html.

Leffingwell, D. (2008). Scaling software agility (1st ed.). Upper Saddle River, N.J: Addison-Wesley.

Leffingwell, D. & Reinertsen, D. (2012). Agile software requirements (1st ed.). Upper Saddle River, N.J: Addison-Wesley.

Meyer, B. (2014). Agile! (1st ed., p. 139). Zurich: Springer International publishing.

Moreira, M. (2009). Being Agile (1st ed., p. 30). New York: Apress.

Rajkumar, R., Lee, I., Sha, L., & Stankovic, J. (2010). Cyber-physical systems: the next computing revolution. Design Automation Conference, (p. 47)

Sutherland, J. & Schwaber, K. (2013). The Scrum Guide (1st ed.). Retrieved from http://www.scrumguides.org/scrum-guide.html

The Standish Group International, Inc.. (2016). The chaos report (p. 3).

Visser, J., Rigal, S., Wijnholds, G., & Lubsen, Z. (2017). Building Software Teams: Ten Best Practices for Effective Software Development (1st ed., p. 23). California: O´REILLY.

Wagner, S. (2014). Scrum for a Cyber-Physical Systems: A Process Proposal. En Proceedings Of The 1St International Workshop On Rapid Continuous Software Engineering, (p. 1).

Wheeler, D. (2016). SLOCCount. Retrieved 25 February 2017, from http://www.dwheeler.com/sloccount/

Notas biográficas:

Dra. Perla Velasco-Elizondo es profesor-investigador en la Universidad Autónoma de Zacatecas. Desde el 2001 realiza actividades de investigación, desarrollo tecnológico, docencia y coaching/consultoría en sus temas de especialidad: Ingeniería de Requerimientos y Diseño e Implementación de Arquitecturas de Software. Ella es SOA Architect Professional y ATAM Evaluator certificado por el Software Engineering Institute. Igualmente, es Scrum Master certificado por la Scrum Alliance. Ha apoyado a equipos de desarrollo a incorporar prácticas tradicionales y ágiles de diseño e implementación de arquitectura. La Dra. Velasco-Elizondo es coautora de uno de los pocos libros de arquitectura de software en español, “Arquitectura de Software: Conceptos y Ciclo de Desarrollo” y ha publicado algunos capítulos en libros y varios artículos en revistas científicas y/o de divulgación. En 2012 fundó una de las primeras Escuelas de Verano en Ingeniería de Software en México, la cual sigue realizando. Ha colaborado como mentor en eventos como Startup Weekend Zacatecas y Hackathon de Campus Party México. Igualmente es miembro de la comunidad Epic Queen en la ciudad de Zacatecas. La Dra. Velasco-Elizondo obtuvo el doctorado en Ciencias de la Computación en The University of Manchester, en El Reino Unido y fue investigadora post-doctoral en el Institute for Software Research de Carnegie Mellon University, en Estados Unidos.

Maestro José Hernández es profesor-investigador del Centro de Investigación en Matemáticas, A. C. Desde el 2010 se ha especializado en el Área de Gestión Ágil de Proyectos de Software tanto como docente, formador y consultor en estos temas. Es Scrum Master por Scrum Alliance, así como Kanban Certified Trainner y Enterprise Service Planning Trainer por Lean Kanban University (http://edu.leankanban.com/users/pepe-hernandez). El Maestro José Hernández cuenta con la Maestría en Administración de Tecnologías de Información y la Maestría en Innovación para el Desarrollo Empresarial, ambas por el Tecnológico de Monterrey.

Ingeniera Gabriela Sobrevilla, es alumna de la Maestría en Ingeniería de Software ofrecida por el Centro de Investigación en Matemáticas. Cuenta con experiencia como líder de proyectos y desarrolladora de software en diversos proyectos para la industria privada. Sus intereses principales incluyen la administración de proyectos de software, métodos de desarrollo ágiles y Lean así como la ingeniería de requerimientos, el análisis y diseño de sistemas de software.

Licenciada Silvia Soriano es líder técnico en desarrollo de software. Laboró en el Instituto Nacional de Astrofísica, Óptica y Electrónica durante 8 años en proyectos de seguridad nacional; fungiendo como desarrollador, analista y finalmente como líder en integración y desarrollo de sistemas multidisciplinarios. A partir del 2015 inició relación laboral con la empresa Lasec Telecomunicaciones S.A.P.I de C.V., donde se desempeña como líder en desarrollo de software. En esta última faceta ha trabajado con un equipo de desarrollo junior utilizando metodologías ágiles.