

Título del Documento

Martin Borgo, Isaías Reniero, Leandro Molina

Universidad Nacional de Entre Ríos

Facultad de Ciencias de la Administración

Licenciatura en Sistemas

martinborgo8@gmail.com, isa.reniero001@hotmail.com, LeandroRodrigoMolina@gmail.com

Abstract. Resumen hasta 200 palabras.

Keywords: Máximo 5 palabras claves.

1 Introducción

En el desarrollo de software, ágil (Agile) es una filosofía de desarrollo que incorpora una serie de prácticas, herramientas y una nueva cultura que permiten al equipo de desarrollo y a las partes interesadas colaborar estrechamente y prosperar en un entorno de requisitos en constante cambio, entregando un programa funcional de manera incremental e iterativa en periodos cortos de tiempo, obteniendo así una retroalimentación continua por parte del usuario. La filosofía ágil es intrínsecamente ligera y fomenta el trabajo en equipo, autoorganizados y empoderados para entregar software de alta calidad. En la actualidad existen distintas metodologías populares que siguen los valores y principios ágiles, tales como: XP (Beck & Andres, 2005) Scrum (Sutherland & Sutherland, 2014), Kanban (Anderson, 2010), Lean (Poppendieck & Poppendieck, 2003), Crystal (Cockburn, 2004), DSDM (Stapleton, 1997) y FDD (Palmer & Felsing, 2001). Los valores son expresados y explicados en el manifiesto ágil (Beck et al., 2001).

Este artículo se centra en dos de las metodologías ágiles más prominentes: Scrum y eXtreme Programming (XP), ya que según estudios realizados por Fuor (2019) Scrum es una de las metodologías más utilizadas para el desarrollo de sistemas, mientras que XP es una de las menos utilizadas, este mismo resultado fue obtenido por el State of Ágil Report¹ del 2022, donde XP ocupa una de las últimas posiciones, mientras que Scrum se encuentra liderando la lista. Estas metodologías gozan o han gozado de gran popularidad y se han sido utilizadas ampliamente en la industria del software como a nivel académico, proporcionando una abundante fuente de información, artículos científicos y casos prácticos. Este trabajo tiene como objetivo explorar las ventajas y desventajas individuales de cada metodología, proporcionando una visión de sus características distintivas y áreas de aplicación. Posteriormente, se revisarán diversas herramientas y estudios que han llevado a cabo comparando estas metodologías, con el objetivo de sintetizar los hallazgos y ofrecer una perspectiva equilibrada sobre cuándo es más apropiado utilizar una sobre la otra. Todo esto con la finalidad de brindar recomendaciones sólidas y basadas en evidencia para la selección de metodologías en situaciones específicas.

2 Desarrollo de Trabajo

En esta sección se realizará una breve introducción a cada una de las metodologías bajo estudio, describiendo tanto ventajas como desventajas, y se finalizará realizando una recopilación de estudios comparativos entre ambas metodologías, con el fin de comprobar los puntos expuestos y dar una visión más amplia sobre el tema.

2.1 eXtreme Programming

Desarrollado en 1999 por Kent Beck y Andres Cynthia, posteriormente presentado en su libro “*Extreme Programming Explained: Embrace Change*” (Beck & Andres, 2005), XP es un marco de trabajo liviano orientado en su totalidad al desarrollo y entrega rápida de un producto de software, basada en un conjunto de buenas prácticas que debían ser llevadas a cabo para lograr los objetivos de manera correcta. Entre estas buenas prácticas se destaca la programación entre pares, la cual consiste en ubicar a dos programadores en un solo equipo, donde uno de ellos se encargará de escribir código y el otro observará dicho código corrigiendo posibles errores, estos roles se van rotando e incluso se puede ir alternando las parejas. Por otro lado está la refactorización, que a grandes rasgos consiste en hacer cambios en el código para hacerlo más legible,

¹16th State of Agile Report

eficiente y mantenible, eliminando duplicaciones, mejorando la claridad del código y haciendo ajustes para seguir buenas prácticas de programación. Por último, se encuentra el desarrollo basado en pruebas o Test Driven Development (Beck, 2003) por sus siglas en inglés, esta práctica se basa en un ciclo de desarrollo en el que se escribe una prueba, se escribe la cantidad mínima de código para que pase, se ejecutan todas las pruebas, para posteriormente refactorizar el código si es necesario. Este ciclo se repite hasta que todas las funcionalidades requeridas estén implementadas. Esta práctica, que en sus inicios fue propia de XP, resultó ser tan beneficiosa al momento del desarrollo que posteriormente fue implementada por otras metodologías de desarrollo como RUP, Kanban o incluso Scrum.

Otra característica innovadora que propuso XP fue la idea de integrar al cliente o partes interesadas directamente en el proceso de desarrollo de software, esto con el fin de asegurar en primera instancia el verificar cumplimiento de los requerimientos del usuario o realizar los cambios que sean necesarios. Además este enfoque otorga ciertas ventajas a la hora de realizar correcciones en alguno de los requerimientos planteados, como también resulta más fácil el adaptarse a nuevos requerimientos que vayan surgiendo a lo largo del ciclo de vida del producto.

Glass (2001) en su artículo menciona los aspectos positivos de XP, entre los cuales se encuentra el testeo unitario, que es probablemente una de las mejores prácticas en producción, la cual nos permite asegurar el buen funcionamiento de un componente del sistema, lo que evita futuros problemas en el desarrollo. Otro aspecto positivo destacado por el autor es la integración continua cuyo propósito es detectar los problemas que pueden surgir por la interacción entre las distintas partes que componen el sistema, garantizando de esta manera que el código escrito se puede combinar de manera efectiva. El último aspecto positivo que enfatizó Glass es el hecho de que el cliente está incluido desde el primer momento en el equipo de desarrollo y siempre es escuchado, presentando una ventaja por sobre Scrum, ya que solo el Product Owner es el único que interactúa con el cliente y la retroalimentación se produce recién en el Sprint Review, en cambio XP permite una retroalimentación más temprana y continua.

Glass también destaca dos aspectos negativos presentes en XP. Por un lado nombra a la refactorización, si bien menciona que la práctica de este método es buena y permite mejorar el código escrito e incentiva a los programadores a utilizar estándares a la hora de codificar, esta práctica aplicada de forma incorrecta, muchas veces puede conducir al mal diseño de un componente o pieza del sistema, lo cual en proyectos medianos y grandes, esto se puede traducir en una gran cantidad de horas para reconstruir el componente por completo o en gran medida. El otro aspecto negativo que es mencionado en el artículo es la programación a pares. Este último aspecto resulta ser sobre todo una opinión personal, ya que en un artículo publicado por Cockburn y Williams (2000) demostró que si bien el tiempo de desarrollo de un equipo que emplea programación a pares es 15 % mayor en comparación con los grupos de desarrollo que programan de forma individual, la cantidad de errores en el código eran un 15 % menor en el caso de la programación a pares, esto representa un gran ahorro de tiempo y recursos, sobre todo de cara al mantenimiento. Además, los autores mencionan una gran cantidad de beneficios extras, tales como una mejora en la comunicación entre los miembros del equipo, una mejora significativa en cuanto al diseño del sistema y la mayor velocidad a la hora de resolver problemas, asimismo resulta útil de aplicar la programación a pares cuando en el grupo de trabajo existe personal que recién se incorpora al sector laboral, ya que estos pueden aprender mucho al programar de forma conjunta ya sea con personas más capacitadas o con el mismo nivel, gracias a la retroalimentación continua y mutua entre ambas partes.

2.2 Scrum

Scrum nace de un artículo publicado en el Harvard Business Review en 1986, titulado “*El nuevo juego de desarrollo de nuevos productos*” (Takeuchi & Nonaka, 1986), en primera instancia este artículo estaba orientado al mundo industrial y no fue hasta que en 1993, Jeff Sutherland y su equipo en Easel Corporation tomaron y combinaron conceptos de ese artículo y crearon el proceso Scrum aplicado al desarrollo de software.

Scrum se presenta como un marco de trabajo bien definido, más enfocado en la gestión del grupo de trabajo y el planeamiento de las diferentes tareas que se deben llevar a cabo en el proceso de desarrollo. El proceso de Scrum está compuesto por múltiples eventos (Sprint planning, Sprint review, Daily, Sprint retrospective), artefactos (Product Backlog, Sprint Backlog e Incremento) y roles (Product Owner, Scrum Master, Equipo de desarrollo) que son utilizados y llevados a cabo por distintas personas dentro del equipo de desarrollo. El aporte significativo de Scrum al campo de la ingeniería de software radica en su enfoque en la mejora continua del proceso de desarrollo. Al concluir cada iteración, o Sprint en el contexto de Scrum, se realiza un análisis del proceso de desarrollo llevado a cabo. Esto permite identificar áreas donde se puede mejorar y/o perfeccionar, dado que cada proyecto es único y puede requerir enfoques distintos para alcanzar el éxito y la correcta satisfacción del cliente.

Rodríguez y Vicente (2015) en su artículo dedican una sección a nombrar algunos aspectos en los que Scrum tiene algunas falencias con respecto a las demás metodologías ágiles que existen, entre esas falencias se encuentra el tiempo, el cual comparado con metodologías como XP donde el desarrollo y puesta en marcha del proyecto es mucho más veloz, Scrum se encuentra en desventaja. Por el lado de los requerimientos, la prioridad u orden en el que estos deben ser implementados dependen del Product Owner, lo cual implica que este puesto debe ser ocupado por una persona experimentada o con experiencia en la actividad. Por último se menciona como puede ser complicado adoptar Scrum en equipos de desarrollo con poca o nula experiencia en el desarrollo, ya que se requiere una sinergia muy grande entre el equipo de desarrollo, esto es respaldado por Barrios et al. (2012) que en su artículo estudia cómo fue la aplicación de Scrum en una pequeña firma desarrolladora de software. El equipo de desarrollo estaba integrado en su mayoría por estudiantes universitarios con poca experiencia en el sector y por algunos graduados con experiencia previa en desarrollo, esto dio como resultado una mala coordinación inicial y estimación de los tiempos de desarrollo, aunque esto se fue corrigiendo en las siguientes Sprints, gracias a las Sprint retrospective. La estimación de los tiempos en Scrum también es una tarea difícil o cuanto menos complicada de realizar, ya que los plazos de tiempo estipulado para la implementación de una tarea es decidido arbitrariamente, ya sea por experiencias previas o suposiciones del equipo de desarrollo, lo que muchas veces se puede traducir en un incumplimiento de los plazos y por consiguiente una demora general en la planificación de las siguientes iteraciones.

Una de las cosas positivas de Scrum es el hecho de que no define un proceso claro a la hora de su adopción, dejando esta decisión a manos del Scrum Master, esto brinda una gran flexibilidad, permitiendo adoptar tanto métodos como procesos de otras metodologías logrando adaptar mejor el desarrollo de acuerdo al tipo de sistema que se está construyendo, aunque este tipo de decisiones siempre deben tomarse con cautela. Una de las ventajas más destacadas de esta metodología es su capacidad de escalabilidad, como mencionamos al principio Scrum se enfoca en la gestión y organización de equipos de trabajo. Esto explica por qué empresas como Spotify eligen Scrum para el desarrollo de sus servicios. Sin embargo, es importante destacar que muchas empresas, incluyendo a Spotify, adaptan y personalizan la aplicación de Scrum para que se ajuste a sus necesidades específicas y al número de desarrolladores con los que cuentan (Kniberg & Ivarsson, 2012).

2.3 Estudios Comparativos

Gill y Henderson-Sellers (2006) utilizaron una herramienta de análisis de 4 dimensiones (4-DAT) para evaluar y comparar las características de ambas metodologías. Esta herramienta permite analizar de manera formal las cualidades de cada una de las metodologías en base a 4 perspectivas llamadas dimensiones, las cuales son:

1. **Caracterización del alcance del método (Method Scope Characterization):** Esta dimensión se enfoca en evaluar varios aspectos como el tamaño del proyecto, el estilo de desarrollo, el entorno tecnológico, etc., para determinar el alcance de un método ágil específico.
2. **Caracterización de la agilidad (Agility Characterization):** En esta dimensión, se evalúa el grado de agilidad de un método ágil en términos de características como flexibilidad, velocidad, leanness, aprendizaje y capacidad de respuesta. Es la única dimensión que tiene un enfoque cuantitativo dividida en dos (fases y prácticas).
3. **Caracterización de los valores ágiles (Agile Values Characterization):** Aquí se evalúa cómo un método ágil soporta los valores ágiles establecidos en el Manifiesto Ágil y otros identificados por los investigadores.
4. **Caracterización del proceso de software (Software Process Characterization):** Esta dimensión se centra en evaluar cómo un método ágil soporta diferentes procesos involucrados en el desarrollo de software, incluyendo el proceso de desarrollo, el proceso de gestión de proyectos, etc.

Tres de estas dimensiones son cualitativas, solamente una (Agility Characterization) es cuantitativa. Se usan estas dimensiones para analizar y comparar las metodologías a través de tablas², ya que cada dimensión contiene diferentes aspectos de estas.

La primera dimensión se utiliza para analizar y comparar el alcance de XP y Scrum en varios aspectos como el tamaño del proyecto, el tamaño del equipo, el estilo de desarrollo, entre otros. Se encuentran que ambos métodos son adecuados para proyectos pequeños y medianos, pero Scrum puede escalar para proyectos grandes, este mismo resultado fue obtenido por Quiñónez-Ku (2017) en su artículo donde se comparaban las distintas metodologías ágiles.

La segunda dimensión mide el grado de agilidad en términos de flexibilidad, velocidad, leanness³, aprendizaje y capacidad de respuesta, asignando una calificación de 1 o 0, donde 1 es positivo y 0 es negativo. Este análisis se realiza en dos niveles: fases y prácticas. Las "fases" se refieren a las etapas del ciclo de vida del desarrollo del software. Y las "prácticas" se refiere a las técnicas específicas empleadas dentro de cada método, como "The Planning Game."^{en} XP, o "Product Backlog."^{en} Scrum. A partir de este análisis, se pueden sacar las siguientes conclusiones: La tercera dimensión analiza cómo las prácticas de XP y Scrum representan los valores ágiles del Manifiesto Ágil. Ambas metodologías respaldan firmemente los primeros cuatro valores a través de varias prácticas. No obstante, XP no respalda los valores de "mantener el proceso ágil" y "mantener el proceso rentable". Scrum respalda el primero, pero, igual que XP, no sostiene el compromiso con la rentabilidad del proceso.

²Los autores además comentan que han desarrollado un prototipo para la implementación de medición de agilidad (Gill & Henderson-Sellers, 2006, p. 1)

³En el contexto de los métodos ágiles se refiere a la eficiencia y economía del proceso de desarrollo. Un proceso "lean" (delgado, en español) busca eliminar el desperdicio, es decir, cualquier cosa que no agregue valor al producto final, y se enfoca en proporcionar exactamente lo que el cliente necesita, ni más ni menos.

La cuarta dimensión examina las prácticas de XP y Scrum para el desarrollo y gestión de proyectos de software. Ambas metodologías presentan prácticas sólidas para estos aspectos, pero no abordan directamente el "Proceso de Control de Configuración/Soporte" ni el "Proceso de Gestión del Proceso". XP destaca por su amplia gama de prácticas que respaldan el desarrollo. Scrum, en cambio, pone un mayor énfasis en el trabajo en equipo y las iteraciones regulares para impulsar el desarrollo. Ambos ofrecen enfoques distintos pero efectivos para la gestión de proyectos.

Si bien esta herramienta resulta útil para la toma de decisiones tiene una serie de limitaciones. Primero, la asignación binaria (0 o 1) para cada característica puede ser demasiado simplista, no capturan la variabilidad y complejidad de cada fase y práctica. Segundo, el análisis es en gran parte cualitativo, lo que puede introducir subjetividad y limitar la precisión del análisis. Tercero, la cantidad de prácticas evaluadas en cada metodología puede influir en los resultados, potencialmente favoreciendo a la metodología con menos prácticas.

Por otro lado, Fernandes y Almeida (2010) realizan una comparación de XP y Scrum en base a un modelo propuesto por Sol (1983) en donde se realiza un análisis cuasi formal de cada una de las metodologías en base a su propuesta y su aplicación práctica. El análisis se divide en dos partes, la primera evalúa cómo se satisface los requerimientos de software en base a 5 atributos. La segunda parte se centra en los aspectos relacionados a la construcción de software, que a su vez es evaluado en base a 4 atributos. A cada uno de estos atributos se le asigna una nota conceptual (satisfecho adecuadamente, parcialmente satisfecho y no satisfecho) basándose en la teoría existente sobre la metodología y su implementación en casos prácticos. Lo más interesante del análisis realizado por Fernandez y Almeida es que dado que Scrum fue planteado desde un inicio como un marco de trabajo orientado a la organización de las distintas tareas y actividades, en el apartado de construcción de software concretamente en todos sus sub-atributos fueron puestos como no satisfechos, aunque en la síntesis de los resultados se lleva un gran puntaje en el apartado de gestión.

3 Resultados Obtenidos

4 Conclusión

Referencias

- Anderson, D. J. (2010). *Kanban: successful evolutionary change for your technology business*. Blue Hole Press.
- Barrios, W. G., Guglielmone, M. V. G., Fernández, M. G., Mariño, S. I., Ferreira, F. M., & Zarrabeitia, C. T. (2012). SCRUM: application experience in a software development PyME in the NEA. *Journal of Computer Science and Technology*, 12(03), 110-115.
- Beck, K. (2003). *Test-driven development: by example*. Addison-Wesley.
- Beck, K., & Andres, C. (2005). *Extreme programming explained: embrace change* (2nd ed). Addison-Wesley.
- Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). Manifesto for Agile Software Development. <https://agilemanifesto.org/>
- Boral, S. (2016). *Ace the PMI-ACP® exam*. Apress. <https://doi.org/10.1007/978-1-4842-2526-4>
- Cockburn, A. (2004). *Crystal clear: A human-powered methodology for small teams: A human-powered methodology for small teams*. Pearson Education.

- Cockburn, A., & Williams, L. (2000). The costs and benefits of pair programming. *Extreme programming examined*, 8, 223-247.
- Fernandes, J. M., & Almeida, M. (2010). Classification and comparison of agile methods. *2010 Seventh International Conference on the Quality of Information and Communications Technology*, 391-396.
- Fuior, F. (2019). Key elements for the success of the most popular Agile methods. *Romanian Journal of Information Technology & Automatic Control/Revista Română de Informatică și Automatică*, 29(4).
- Gill, A., & Henderson-Sellers, B. (2006). Comparative evaluation of XP and scrum using the 4d analytical tool (4-DAT). *Proceedings of the European and Mediterranean Conference on Information Systems, EMCIS 2006*.
- Glass, R. L. (2001). Extreme programming: The good, the bad, and the bottom line. *IEEE software*, 18(6), 112.
- Kniberg, H., & Ivarsson, A. (2012). Scaling agile@ spotify. *online*], *UCVOF, ucvox. files. wordpress.com/2012/11/113617905-scaling-Agile-spotify-11.pdf*.
- McKenna, D. (2016). *The Art of Scrum*. Apress. <https://doi.org/10.1007/978-1-4842-2277-5>
- Palmer, S. R., & Felsing, M. (2001). *A practical guide to feature-driven development*. Pearson Education.
- Poppendieck, M., & Poppendieck, T. (2003). *Lean software development: an agile toolkit*. Addison-Wesley.
- Quiñónez-Ku, X. (2017). Análisis Comparativo de metodologías Agiles de desarrollo de software: una revisión bibliográfica. *Education*, 2018.
- Rodríguez, C., & Vicente, R. D. (2015). ¿Por qué implementar Scrum? *Revista Ontare*, 3(1), 125-144.
- Sol, H. G. (1983). A Feature Analysis of Information Systems Design Methodologies: Methodological Considerations. *CRIS*, 1-8.
- Stapleton, J. (1997). Dynamic Systems Development Method. The method in practice (DSDM), DSDM Consortium 1997.
- Sutherland, J., & Sutherland, J. (2014). *Scrum: the art of doing twice the work in half the time*. Currency.
- Takeuchi, H., & Nonaka, I. (1986). The New New Product Development Game. *Harvard Business Review*. <https://hbr.org/1986/01/the-new-new-product-development-game>