UNIVERSIDAD DEL BÍO-BÍO

Tópicos Avanzados en Ingeniería de Software

Desarrollo Ágil de software

María Antonieta Soto Ch.
Magíster en Ciencias de la Computación
2019

Traducido de transparencias que acompañan el texto: Sommerville, Ian (2015). “Software Engineering”, 10th. Ed., Pearson.

1

Temas tratados

- ✧ Métodos ágiles
- ✧ Técnicas de desarrollo ágil
- ✧ Administración de un proyecto ágil
- ✧ Escalamiento de métodos ágiles

2

Desarrollo rápido de software

- En la actualidad la entrega y el desarrollo rápidos son por lo general el requerimiento fundamental de los sistemas de software
  - Los negocios funcionan en un entorno altamente cambiante, siendo prácticamente imposible producir un conjunto de requerimientos de software estable
  - El software tiene que evolucionar rápidamente para reflejar las necesidades cambiantes de los negocios.
- El desarrollo conducido por plan es esencial para algunos tipos de sistema, pero no satisface estas necesidades de negocio.
- Los métodos ágiles aparecen a fines de los 90s cuyo objetivo era reducir radicalmente el tiempo de entrega de sistemas de software funcional

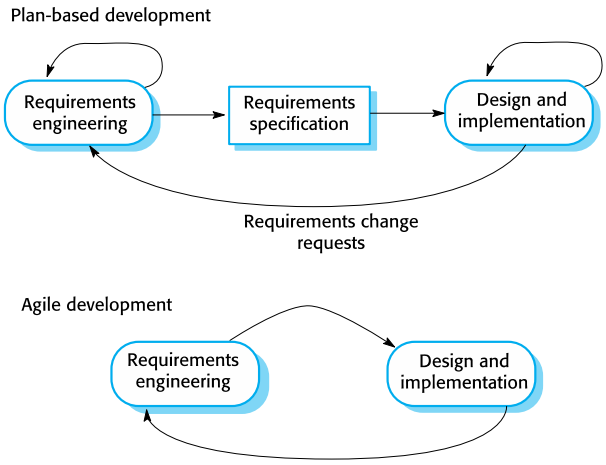
3

Desarrollo Ágil

- La especificación, diseño e implementación de programas se intercalan
- El sistema se desarrolla como una serie de versiones o incrementos con stakeholders que participan en la especificación y evaluación de las versiones
- Entrega frecuente de nuevas versiones para su evaluación
- Extenso uso de herramientas (p.e. herramientas de prueba automatizadas) para apoyar el desarrollo.
- Mínima documentación – foco está en código funcional

4

Especificación dirigida por plan y ágil



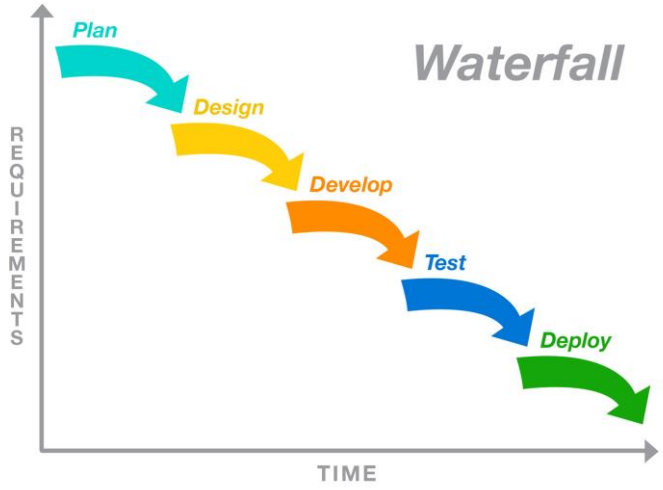
5

Desarrollo dirigido por un plan y desarrollo ágil

- ✧ Desarrollo conducido por plan
  - Un enfoque conducido por plan para la ingeniería de software identifica etapas separadas en el proceso de desarrollo con salidas asociadas a cada etapa. Las salidas de una etapa se usan para planear la siguiente actividad del proceso.
  - No necesariamente corresponde al modelo cascada, también soporta el desarrollo y la entrega incremental.
  - La iteración ocurre dentro de las actividades.
- ✧ Desarrollo ágil
  - La especificación, diseño, implementación y prueba están intercaladas y las salidas se deciden a través de un proceso de negociación durante el proceso de desarrollo de software.

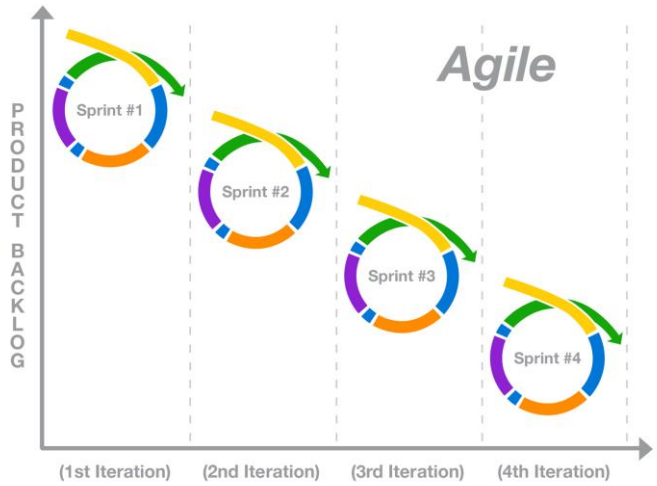
6

¿Cuán diferente es Ágil de Cascada?



7

¿Cuán diferente es Ágil de Cascada?



8

La mentalidad ágil

Hacer Ágil

- Enfocarse en todas las reglas & procedimientos.
- Decir a otros qué hacer.
- No es mi labor o responsabilidad.
- Saltarse las pruebas para entregar rápido.
- Codifico esto en 2 semanas, por lo tanto comenzaré la próxima semana y trabajaré 10 horas diarias.

Ser Ágil

- Enfocarse en entregar valor al cliente.
- Trabajar juntos para hacerlo.
- Como miembro del equipo es nuestra responsabilidad.
- Construyamos scripts de prueba para garantizar la calidad.
- Trabajamos a una velocidad sostenible.

Mentalidad Ágil

- Flexible.
- Adaptable.
- Actitud positiva.
- Objetivo de éxito del equipo.
- Apertura y aceptación de los fracasos como una etapa del proceso de aprendizaje.

9

Métodos ágiles

10

Métodos ágiles

- Descontento con los costos involucrados en los métodos de diseño de software de los 80s y 90s llevó a la creación de métodos ágiles. Estos métodos:
  - Se enfocan en el software en lugar del diseño
  - Se apoyan en el enfoque incremental para el desarrollo de software
  - Son adecuados para el desarrollo de aplicaciones cuyos requerimientos cambian rápidamente.
- El objetivo de los métodos ágiles es reducir los costos del proceso de desarrollo de software (p.e. limitando la documentación) y responder rápidamente ante requerimientos cambiantes sin re-trabajo excesivo.

11

Manifiesto Ágil

- Sus principios son:
  1. La mayor prioridad es satisfacer al cliente
  2. El software funcionando es la medida del progreso
  3. Se acepta que los requerimientos cambien
  4. Ritmo de desarrollo sostenible
  5. Entrega continua de software con valor
  6. Atención continua a la excelencia técnica
  7. Los responsables del negocio & los desarrolladores juntos diariamente
  8. Simplicidad
  9. Los proyectos se desarrollan en torno a individuos motivados
  10. Equipos auto-motivados
  11. La conversación cara a cara es mejor
  12. Reflexión & adaptación a intervalos regulares

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** *OVER* processes and tools
- **Working software** *OVER* comprehensive documentation
- **Customer collaboration** *OVER* contract negotiation
- **Responding to change** *OVER* following a plan

12

Aplicabilidad de los métodos ágiles

- ✧ Desarrollo de un producto donde una compañía de software elabora un producto pequeño o mediano para la venta.
  - Virtualmente todos los productos de software y apps se desarrollan ahora usando un enfoque ágil
- ✧ Diseño de sistemas a la medida dentro de una organización, donde hay un claro compromiso del cliente por intervenir en el proceso de desarrollo, y donde no existen muchas reglas ni regulaciones externas que afecten el producto.

13

Técnicas de desarrollo ágil

14

Métodos/Técnicas de desarrollo ágil

- Las empresas que utilizan el desarrollo ágil de software, normalmente combinan varios métodos/técnicas:
  - Lean
  - Kanban
  - Scrum
  - XP

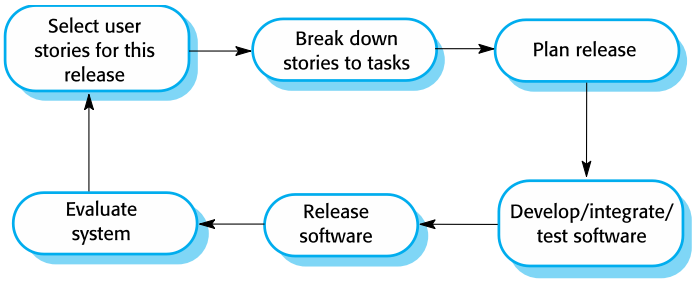
15

Programación extrema

- Un método ágil muy influyente, desarrollado a finales de 1990, que introdujo una serie de técnicas de desarrollo ágil.
- La programación extrema (XP, eXtreme Programming) lleva a niveles 'extremos' el desarrollo iterativo.
  - Se pueden construir nuevas versiones varias veces al día;
  - Se entregan incrementos a los clientes cada 2 semanas;
  - Se deben ejecutar todas las pruebas a cada nueva versión y las versiones se aceptan solo si las pruebas se ejecutan exitosamente.

16

Ciclo de versiones de la programación extrema



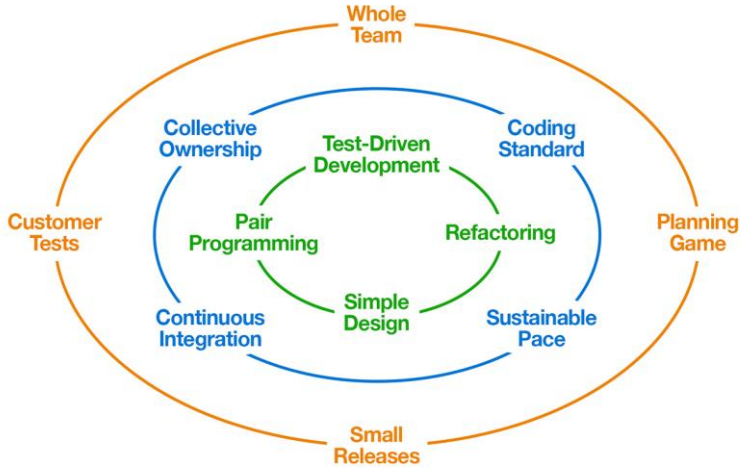
17

XP y los principios ágiles

- El desarrollo incremental se apoya en pequeñas y frecuentes entregas (releases) del sistema.
- La participación o inclusión del cliente significa un compromiso de tiempo completo de este con el equipo.
- Las personas, no los procesos, se basan en la programación en pares, la propiedad colectiva del código y en un proceso de desarrollo sustentable que evite largas jornadas de trabajo.
- Los cambios son admitidos a través de entregas regulares del sistema
- Mantener la simplicidad a través de la refactorización constante del código.

18

XP Practices



19

Prácticas XP Influyentes

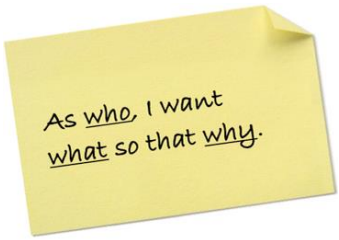
- ✧ XP tiene se enfoca en lo técnico y no es fácil de integrar con las prácticas de gestión de la mayoría de las organizaciones.
- ✧ En consecuencia, el desarrollo ágil usa las prácticas de XP, mientras que el método definido originalmente no se utiliza ampliamente.
- ✧ Prácticas claves:
  - Historias de usuario para la especificación
  - Refactorización
  - Pruebas primero, luego desarrollo
  - Programación en pares

20

Historias de usuario para los requerimientos

- ✧ En XP, un cliente o usuario es parte del equipo XP y es responsable de decidir acerca de los requerimientos.
- ✧ Los requerimientos de usuario se expresan como historias de usuario o escenarios.
- ✧ Est@s se escriben en tarjetas y el equipo de desarrollo l@s divide en tareas de implementación. Estas tareas son la base de la calendarización (programación) y la estimación de costos.
- ✧ El cliente escoge las historias a incluir en la siguiente liberación basado en sus prioridades y la calendarización estimada.

Historias de usuario

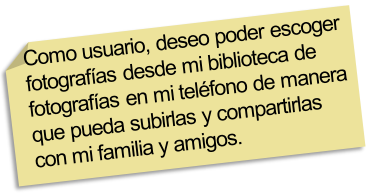


- ✧ ¿Qué es una historia de usuario?
  - Una historia de usuario representa un pequeña porción del valor del negocio que un equipo entrega en una iteración.
  - Cada historia de usuario se escribe usando la siguiente plantilla.

    Plantilla (o template):

Como un <tipo de usuario>, quiero <objetivo> de modo que <beneficio>.

- ✧ Ejemplo:



Refactorización

- La creencia convencional en ingeniería de software es diseñar para el cambio. Vale la pena invertir tiempo y esfuerzo anticipándose a los cambios ya que esto reduce los costos más adelante en el ciclo de vida.
- XP, sin embargo, sostiene que esto no vale la pena en la medida que los cambios no se pueden anticipar con fiabilidad.
  Más bien, se propone mejorar el código constantemente (refactorización) para facilitar los cambios cuando se tienen que implementar.

23

Refactorización

- El equipo de programación busca posibles mejoras del software y las hace incluso cuando no hay necesidad inmediata de ellas.
- Esto mejora la comprensión del software y de este modo se reduce la necesidad de documentación.
- Los cambios son más fáciles de realizar porque el código está bien estructurado y es claro.
- Sin embargo, algunos cambios requieren refactorizar la arquitectura y esto es mucho más costoso.

24

Ejemplos de refactorización

- ✧ Re-organizar una jerarquía de clases para eliminar código duplicado.
- ✧ Ordenar y cambiar el nombre de los atributos y métodos para hacerlos más fáciles de entender.
- ✧ La sustitución de código con llamadas a métodos que se han incluido en una biblioteca de programas.

Pruebas en XP

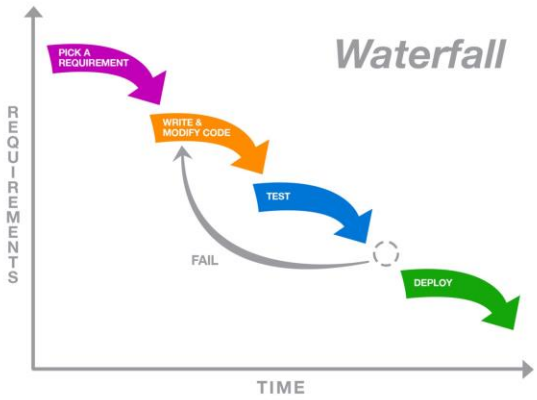
- ✧ Las pruebas son centrales para XP que ha desarrollado una aproximación donde el programa es probado después de realizar cada cambio.
- ✧ Características de las pruebas en XP:
  - Definición de pruebas primero.
  - Desarrollo incremental de pruebas a partir de escenarios.
  - Participación del usuario en la definición y validación de las pruebas.
  - Uso de herramientas de prueba que permitan ejecutar todas las pruebas cada vez que se libera una nueva versión.

Desarrollo conducido por pruebas - TDD

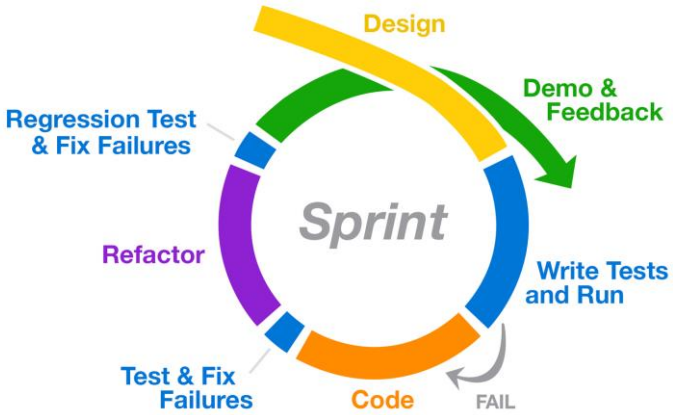
- ✧ Escribir las pruebas antes de codificar aclara los requerimientos a implementar.
- ✧ Las pruebas se escriben como programas en lugar de datos de modo que se puedan ejecutar automáticamente.
  - Por lo general se basa en un marco de pruebas tales como Junit.
- ✧ Todas las pruebas anteriores y los nuevos se ejecutan automáticamente cuando se añade una nueva funcionalidad, comprobando así que la nueva funcionalidad no ha introducido errores.

TDD - Test Driven Development

- ✧ TDD es un enfoque al revés para que los desarrolladores no TDD. A continuación se muestra el enfoque tradicional no-TDD:



TDD - Test Driven Development (& Diseño)



29

Descripción de caso de prueba para comprobación de dosis

Test 4: Dose checking
Input: 1. A number in mg representing a single dose of the drug. 2. A number representing the number of single doses per day. **Tests:** 1. Test for inputs where the single dose is correct but the frequency is too high. 2. Test for inputs where the single dose is too high and too low. 3. Test for inputs where the single dose * frequency is too high and too low. 4. Test for inputs where single dose * frequency is in the permitted range. **Output:** OK or error message indicating that the dose is outside the safe range.

30

Problemas con TDD

- Los programadores prefieren programar en lugar de probar y, a veces toman atajos al escribir pruebas. Por ejemplo, pueden escribir pruebas incompletas que no comprueban todas las posibles excepciones que pueden ocurrir.
- Algunas pruebas pueden ser muy difíciles de escribir de forma incremental. Por ejemplo, en una interfaz de usuario compleja, a menudo es difícil escribir pruebas unitarias para el código que implementa la "lógica de visualización" y flujo de trabajo entre pantallas.
- Es difícil juzgar la exhaustividad de un conjunto de pruebas. Aunque es posible que tenga una gran cantidad de pruebas de sistema, el conjunto de pruebas no puede proporcionar una cobertura completa.

31

Gestión de proyectos ágiles

32

Scrum

- ✧ Scrum es un método ágil que se centra en la gestión del desarrollo iterativo en lugar de prácticas ágiles específicas.
- ✧ Hay tres fases en Scrum.
  - La fase inicial es una fase de planificación general en la que se establecen los objetivos generales del proyecto y el diseño de la arquitectura de software.
  - Esta es seguida por una serie de ciclos sprint, donde en cada ciclo se desarrolla un incremento del sistema.
  - La fase de cierre del proyecto concluye el proyecto, completa la documentación requerida, tales como marcos de ayuda del sistema y manuales de usuario y evalúa las lecciones aprendidas del proyecto.

33

Terminología Scrum (a)

Scrum term	Definition
Development team	A self-organizing group of software developers, which should be no more than 7 people. They are responsible for developing the software and other essential project documents.
Potentially shippable product increment	The software increment that is delivered from a sprint. The idea is that this should be 'potentially shippable' which means that it is in a finished state and no further work, such as testing, is needed to incorporate it into the final product. In practice, this is not always achievable.
Product backlog	This is a list of 'to do' items which the Scrum team must tackle. They may be feature definitions for the software, software requirements, user stories or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation.
Product owner	An individual (or possibly a small group) whose job is to identify product features or requirements, prioritize these for development and continuously review the product backlog to ensure that the project continues to meet critical business needs. The Product Owner can be a customer but might also be a product manager in a software company or other stakeholder representative.

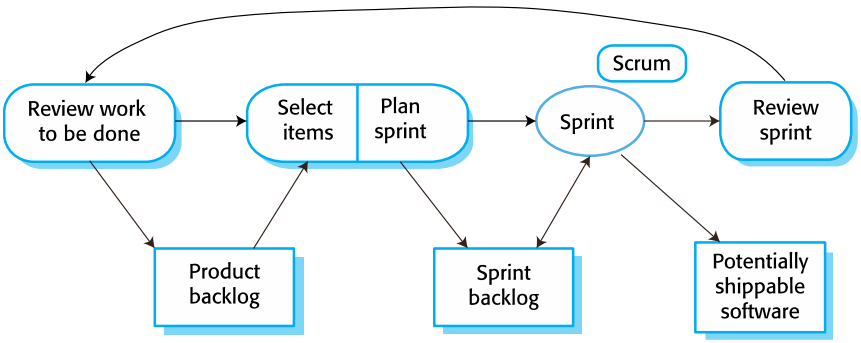
34

Terminología Scrum (b)

Scrum term	Definition
Scrum	A daily meeting of the Scrum team that reviews progress and prioritizes work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team.
ScrumMaster	The ScrumMaster is responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. He or she is responsible for interfacing with the rest of the company and for ensuring that the Scrum team is not diverted by outside interference. The Scrum developers are adamant that the ScrumMaster should not be thought of as a project manager. Others, however, may not always find it easy to see the difference.
Sprint	A development iteration. Sprints are usually 2-4 weeks long.
Velocity	An estimate of how much product backlog effort that a team can cover in a single sprint. Understanding a team's velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance.

35

Ciclo sprint de Scrum



36

Ciclo sprint de Scrum

- ✧ Los sprints tienen una duración fija, generalmente 2–4 semanas.
- ✧ El punto de partida para planificar es el product backlog, que es la lista de trabajo que se debe realizar durante el proyecto.
- ✧ La fase de selección involucra a todo el equipo del proyecto quienes trabajan con el cliente para seleccionar las características y funcionalidad del product backlog que se debe desarrollar durante el sprint.

El ciclo Sprint

- ✧ Una vez que se llega a acuerdo respecto de las funcionalidades, el equipo se organiza para desarrollar el software.
- ✧ Durante esta etapa el equipo se separa del cliente y de la organización, canalizándose todas las comunicaciones a través del ‘Scrum master’.
- ✧ La tarea del Scrum master es proteger al equipo de desarrollo de distracciones externas.
- ✧ Al término del sprint, el trabajo realizado se revisa y presenta a los stakeholders. A continuación, comienza el siguiente sprint.

El trabajo en equipo en Scrum

- El 'Scrum Master' es un facilitador que organiza reuniones diarias, realiza un seguimiento del backlog de trabajo a realizar, registra las decisiones, mide el progreso contra el backlog y se comunica con los clientes y la administración fuera del equipo.
- Todo el equipo asiste a reuniones diarias cortas (Scrums), donde todos los miembros del equipo comparten información, describen su progreso desde la última reunión, los problemas que han surgido y lo que se planea para el día siguiente.
  - Esto significa que todos en el equipo saben lo que está pasando y, si surgen problemas, pueden volver a planificar el trabajo a corto plazo para afrontarlos.

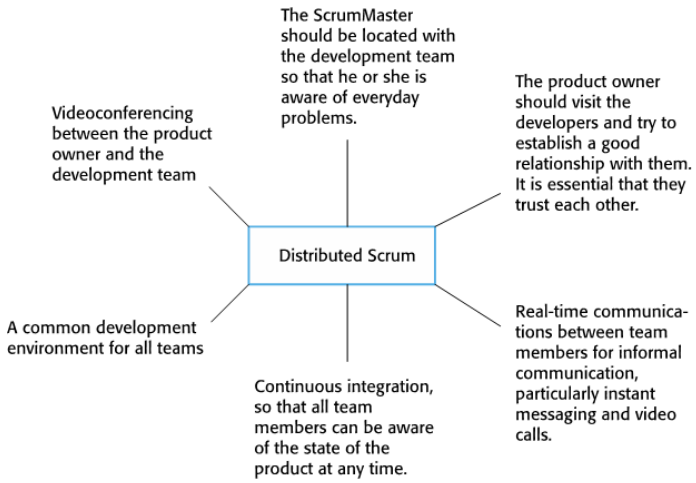
39

Beneficios de Scrum

- El producto se descompone en un conjunto de partes manejables y comprensibles.
- Los requisitos inestables no retrasan el progreso.
- Todo el equipo tiene visibilidad de todo y, por lo tanto, se mejora la comunicación del mismo.
- Los clientes ven incrementos entregados a tiempo y obtienen retroalimentación sobre cómo funciona el producto.
- Se establece una relación de confianza entre clientes y desarrolladores y se crea una cultura positiva en la que todo el mundo espera que el proyecto tenga éxito.

40

Scrum distribuido



41

Escalamiento de métodos ágiles

42

Escalamiento de métodos ágiles

- Los métodos ágiles han demostrado tener éxito en proyectos pequeños y medianos que pueden ser desarrollados por un pequeño equipo ubicado en el mismo lugar.
- A veces se afirma que el éxito de estos métodos se produce debido a la mejora de las comunicaciones, que es posible cuando todos están trabajando juntos.
- Escalar los métodos ágiles supone cambiar estos para hacer frente a proyectos más grandes, más extensos, en los que hay varios equipos de desarrollo, tal vez trabajando en diferentes lugares.

Escalamiento por expansión y ampliación

- La perspectiva de “expansión” (scaling up) se interesa por el uso de métodos ágiles para el desarrollo de grandes sistemas de software que no pueden ser desarrollados por equipos pequeños.
- La perspectiva de “ampliación” (scaling out) se interesa por que los métodos ágiles se introduzcan en una organización grande con muchos años de experiencia en el desarrollo de software.
- Al escalar métodos ágiles es esencial que se mantengan los fundamentos ágiles
  - Planificación flexible, liberaciones frecuentes del sistema, integración continua , desarrollo dirigido por pruebas y buena comunicación del equipo.

Problemas prácticos de los métodos ágiles

- ✧ La informalidad del desarrollo ágil es incompatible con el enfoque jurídico de definición de contratos que se utiliza comúnmente en las grandes empresas.
- ✧ Los métodos ágiles son los más apropiados para el desarrollo de software nuevo en lugar de mantenimiento de software. Sin embargo, la mayoría de los costos de software en las grandes empresas provienen del mantenimiento de sus sistemas de software existentes.
- ✧ Los métodos ágiles están diseñados para pequeños equipos ubicados en el mismo lugar, sin embargo, mucho desarrollo de software ahora involucra equipos de distintas regiones del mundo.

Aspectos contractuales

- ✧ La mayoría de los contratos de software para sistemas personalizados se basan en una especificación, que establece lo que tiene que implementar el desarrollador para el cliente del sistema.
- ✧ Sin embargo, esto impide la intercalación de especificación y desarrollo como es norma en el desarrollo ágil.
- ✧ Se requiere un contrato que paga por el tiempo de desarrollo en lugar de la funcionalidad.
  - Sin embargo, los departamentos jurídicos consideran esto muy riesgoso porque no se puede garantizar lo que se tiene que entregar.

Métodos ágiles y mantención de software

- La mayoría de las organizaciones gastan más en mantener software existente que en el desarrollo de nuevo software. Por lo tanto, para que los métodos ágiles sean exitosos, deben apoyar la mantención tanto como el desarrollo original.
- Dos aspectos claves:
  - ¿Se pueden mantener los sistemas que se desarrollan usando una aproximación ágil, dado el énfasis en el proceso de desarrollo de minimizar la documentación formal?
  - ¿Se pueden usar los métodos ágiles efectivamente para evolucionar un sistema como respuesta a requerimientos de cambio de un cliente?
- Pueden aparecer problemas si el equipo de desarrollo no se puede mantener como tal.

Mantención ágil

- Los problemas fundamentales son:
  - La falta de documentación del producto
  - Mantener a los clientes participando en el proceso de desarrollo
  - Mantener la continuidad del equipo de desarrollo
- El desarrollo ágil se basa en que el equipo de desarrollo conoce y entiende lo que tiene que hacer.
- Para los sistemas de ciclo de vida largo, esto es un problema real, ya que los desarrolladores originales no siempre trabajarán en el sistema.

Métodos ágiles y conducidos por plan

- La mayoría de los proyectos incluyen elementos de procesos conducidos por plan y ágil. Decidir sobre el equilibrio entre estos enfoques depende de:
  - ¿Es importante tener una especificación y un diseño muy detallados antes de pasar a la implementación? Si es así, probablemente usted tenga que usar un enfoque conducido por plan.
  - ¿Es práctica una estrategia de entrega incremental, donde se dé el software a los clientes y se obtenga una rápida retroalimentación de parte de ellos? De ser así, considere el uso de métodos ágiles.
  - ¿Qué tan grande es el sistema que se desarrollará? Los métodos ágiles son más efectivos cuando el sistema se puede desarrollar con un pequeño equipo que se comunique de manera informal. Esto sería imposible para grandes sistemas que precisan equipos de desarrollo más grandes, en el que se tenga que usar un enfoque conducido por plan.

49

Principios ágiles y práctica organizacional

Principle	Practice
Customer involvement	This depends on having a customer who is willing and able to spend time with the development team and who can represent all system stakeholders. Often, customer representatives have other demands on their time and cannot play a full part in the software development. Where there are external stakeholders, such as regulators, it is difficult to represent their views to the agile team.
Embrace change	Prioritizing changes can be extremely difficult, especially in systems for which there are many stakeholders. Typically, each stakeholder gives different priorities to different changes.
Incremental delivery	Rapid iterations and short-term planning for development does not always fit in with the longer-term planning cycles of business planning and marketing. Marketing managers may need to know what product features several months in advance to prepare an effective marketing campaign.

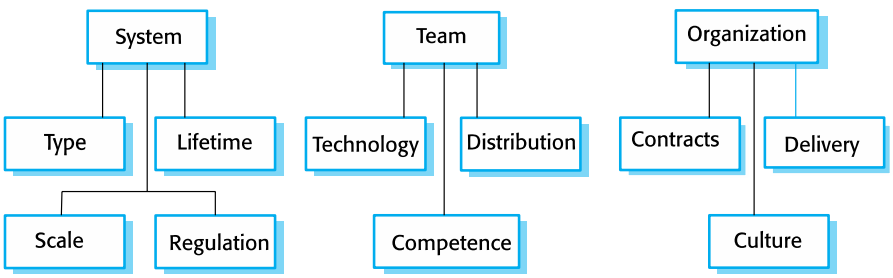
50

Principios ágiles y práctica organizacional

Principle	Practice
Maintain simplicity	Under pressure from delivery schedules, team members may not have time to carry out desirable system simplifications.
People not process	Individual team members may not have suitable personalities for the intense involvement that is typical of agile methods, and therefore may not interact well with other team members.

51

Factores ágiles y basados en plan



52

Aspectos del sistema

- ¿Cuán grande es el sistema que se desarrollará?
  - Los métodos ágiles son más efectivos con un equipo relativamente pequeño y ubicado en el mismo lugar, quienes puedan comunicarse informalmente.
- ¿Qué tipo de sistema se desarrollará?
  - Los sistemas que requieren una gran cantidad de análisis antes de la implementación necesitan un diseño bastante detallado para llevar a cabo este análisis.
- ¿Cuál es el tiempo de vida que se espera del sistema?
  - Los sistemas con tiempos de vida largos pueden requerir más documentación de diseño para comunicar al equipo de apoyo los propósitos originales de los desarrolladores del sistema.
- ¿El sistema está sujeto a regulación externa?
  - Si un regulador externo tiene que aprobar el sistema (p.e. la FAA de EEUU aprueba el software que es crítico para la operación de una aeronave), entonces, probablemente requerirá generar documentación detallada como parte del sistema de seguridad.

Personas y equipos

- ¿Qué tan buenos son los diseñadores y programadores del equipo de desarrollo?
  - A veces se argumenta que los métodos ágiles requieren niveles de habilidad superiores a los necesarios en los enfoques conducidos por plan, en los que los programadores simplemente traducen un diseño detallado a código.
- ¿Cómo está organizado el equipo de desarrollo?
  - Si el equipo de desarrollo está distribuido o si parte del desarrollo se subcontrata, entonces tal vez se requiera elaborar documentos de diseño para que se comunique todo el equipo de desarrollo.
- ¿Qué tecnologías están disponibles para apoyar el desarrollo del sistema?
  - Si la documentación de diseño no está disponible, es esencial un IDE para apoyar la visualización y análisis del programa.

Aspectos organizacionales

- ✧ Las organizaciones de ingeniería tradicionales tienen una cultura de desarrollo basado en plan, ya que esta es la norma en ingeniería.
- ✧ ¿Es una práctica estándar de la organización desarrollar una especificación detallada del sistema?
- ✧ ¿Estarán los representantes de los clientes disponibles para proporcionar retroalimentación acerca de los incrementos del sistema?
- ✧ ¿Puede el desarrollo ágil informal encajar en la cultura organizacional de documentación detallada?

Métodos ágiles en el desarrollo de grandes sistemas

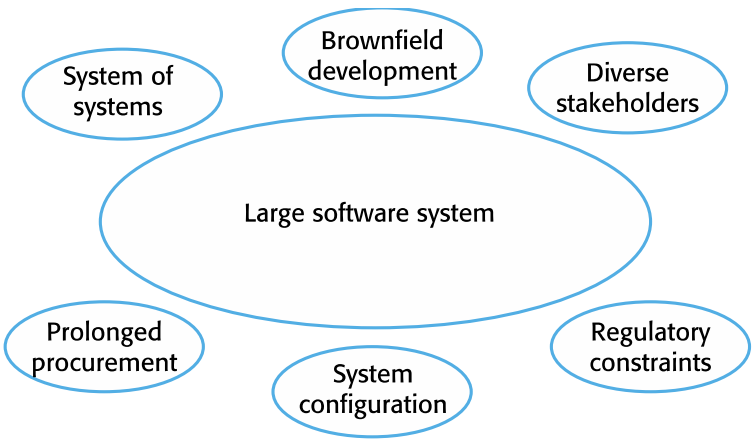
- ✧ Los grandes sistemas son, generalmente, colecciones de sistemas separados que se intercomunican, donde equipos diferentes desarrollan cada sistema. Comúnmente, estos equipos trabajan en distintos lugares, en ocasiones en diversas zonas horarias.
- ✧ Los grandes sistemas son ‘sistemas antiguos (abandonados)’, esto es, incluyen e interactúan con un número de sistemas existentes. Muchos de los requerimientos del sistema dicen relación con esta interacción y por ello no son proclives a la flexibilidad y al desarrollo incremental.
- ✧ Donde se integran varios sistemas para crear un sistema, una fracción significativa del desarrollo tiene relación con la configuración del sistema más que con el desarrollo de nuevo código.

Desarrollo de grandes sistemas

- Los grandes sistemas y sus procesos de desarrollo a menudo están restringidos por reglas y regulaciones externas que limitan la manera en que se puede desarrollar.
- Los grandes sistemas tienen un tiempo prolongado de adquisición y desarrollo. Es difícil mantener equipos coherentes que conozcan el sistema durante dicho período, pues resulta inevitable que las personas se cambien a otros trabajos y proyectos.
- Los grandes sistemas tienen generalmente un conjunto variado de participantes (stakeholders). Es prácticamente imposible involucrar a todos estos participantes en el proceso de desarrollo.

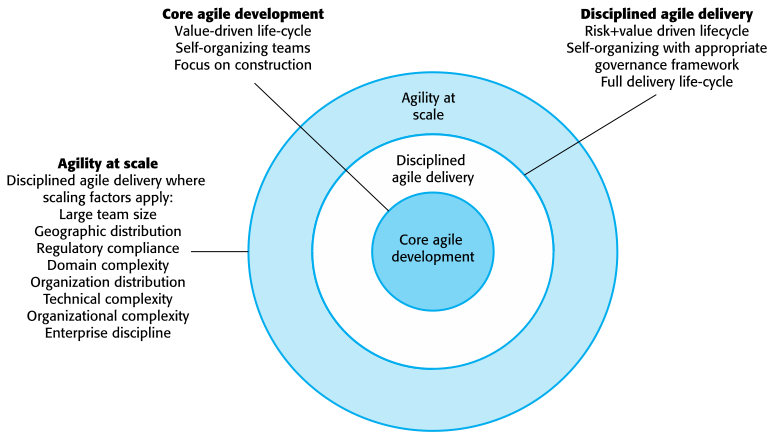
57

Factores presentes en grandes sistemas



58

Esquema de agilidad de IBM



59

Escalamiento (por expansión) de grandes sistemas

- ✧ Es imposible un enfoque completamente incremental para la ingeniería de requisitos.
- ✧ No puede haber un único product owner o representante del cliente.
- ✧ En el desarrollo de grandes sistemas, es imposible enfocarse solo en el código del sistema.
- ✧ Se tienen que diseñar y usar sistemas de comunicación entre equipos de desarrollo.
- ✧ La integración continua es prácticamente imposible. Sin embargo, es esencial mantener builds frecuentes y liberaciones regulares del sistema.

60

Scrum multi-equipo

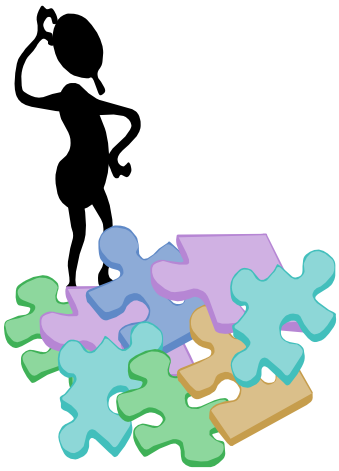
- Replicación de rol
  - Cada equipo tiene un Product Owner para su parte del trabajo y un Scrum Master.
- Arquitectos de producto
  - Cada equipo elige un arquitecto de producto y estos arquitectos colaboran para diseñar y desarrollar la arquitectura general del sistema.
- Alineación de releases
  - Las fechas de releases del producto de cada equipo se alinean de forma que se genera un sistema demostrable y completo.
- Scrum de Scrums
  - Hay un Scrum diario de Scrums donde representantes de cada equipo se reúnen para discutir el progreso y plan de trabajo de lo que queda por hacer.

61

Métodos ágiles en las organizaciones

- Los administradores de proyectos que no tienen experiencia en métodos ágiles pueden ser reacios a aceptar el riesgo de un nuevo enfoque.
- Las grandes organizaciones suelen tener procedimientos y normas de calidad que se espera todos los proyectos sigan y, debido a su carácter burocrático, éstos tienden a ser incompatibles con los métodos ágiles.
- Los métodos ágiles parecen funcionar mejor cuando los miembros del equipo tienen un nivel relativamente alto de habilidad. Sin embargo, en grandes organizaciones, es probable que haya una amplia gama de habilidades y capacidades.
- Puede haber resistencia cultural a los métodos ágiles, especialmente en aquellas organizaciones que tienen una larga historia de uso de procesos convencionales de ingeniería de sistemas.

62



63