PROCESOS ÁGILES

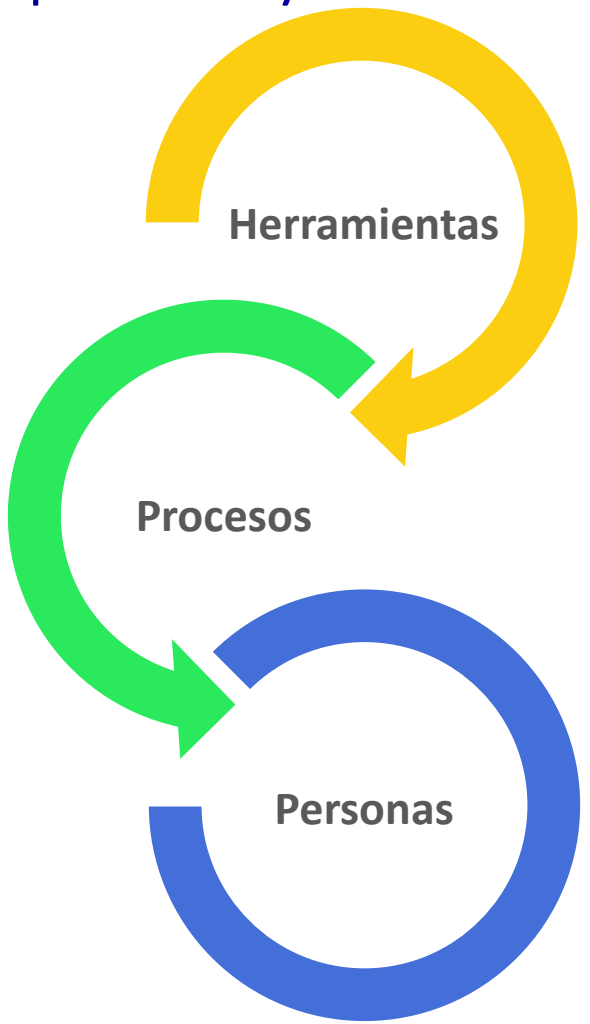
MANIFIESTO POR EL DESARROLLO ÁGIL (2001)

- Hemos aprendido a valorar:
  - personas e interacciones más que procesos y herramientas
  - software funcionando más que documentación extensiva
  - colaboración con el cliente más que negociación contractual
  - respuesta ante el cambio más que seguir un plan.

PERSONAS E INTERACCIONES

… personas e interacciones más que procesos y herramientas

Para producir software se necesita:



Hasta década de los 80

A partir de la década de los 80

A partir del 2001

SOFTWARE FUNCIONANDO

… software funcionando más que documentación extensiva

- Podemos tener una especificación de requisitos o un diseño maravilloso, pero si otro tiene el producto funcionando…
- Lo prefiero, porque
  1. puedo evaluar atributos de calidad externos del sw funcionando. El cliente nos puede dar una retroalimentación.
  2. El sw funcionando (no solo construido), pone el criterio de verificación bastante alto: no dice que sea perfecto, pero tiene que funcionar al punto tal que pueda ser evaluado: criterio de calidad muy alto. Las versiones B no son versiones en condiciones de ponerse en producción, sino de ser evaluadas. Se tarda mucho y lleva mucho esfuerzo tener sw funcionando.
  3. Permite reducir el riesgo de que haya una variación significativa entre la percepción del avance del proyecto y el avance real. P. ej. si tengo 95 % de avance y el cliente me dice que no le sirve…

COLABORACIÓN CON EL CLIENTE

…colaboración con el cliente más que negociación contractual.

El manifiesto ágil se presenta en un momento en que hay fuerte presión por reducir el tiempo de salida al mercado, hay muchos proyectos innovadores (despegue de la web), y hay mucha competencia. Difícil definir con precisión los requisitos al comienzo del proyecto. Es común que cambien y que sean imprecisos.

Cuando el cliente tiene muy claro qué precisa, se puede elaborar un contrato con el cliente. En ambiente de incertidumbre ese enfoque rinde poco. Entonces precisamos que cliente esté colaborando con nosotros y nosotros con él. La gracia de generar sw funcionando es que el cliente lo pueda evaluar.

RESPUESTA ANTE EL CAMBIO

… respuesta ante el cambio más que seguir un plan

En procesos tradicionales el grueso de la planificación se lleva adelante al comienzo. Esto no implica que no tengamos que planificar después (incluso en proceso en cascada), por cambios en los requisitos o en las condiciones (productividad, cantidad de defectos, tecnologías incompatibles).

En el enfoque ágil, en ambiente de incertidumbre: ¿para qué vamos a sobreplanificar? Planifiquemos solo a corto plazo (es como lo meteorológico); al alejarse en el plazo, el nivel de incertidumbre aumenta y planificar a lago plazo es inútil.

MANIFIESTO POR EL DESARROLLO ÁGIL (2001)

- Hemos aprendido a valorar:
  - personas e interacciones más que procesos y herramientas
  - software funcionando más que documentación extensiva
  - colaboración con el cliente más que negociación contractual
  - respuesta ante el cambio más que seguir un plan.
- Aunque valoramos los elementos de la derecha, valoramos más los de la izquierda.

PRINCIPIOS DEL MANIFIESTO ÁGIL

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcionando es la medida principal de progreso.
8. Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos autoorganizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

PROCESOS GUIADOS POR PLANES Y ÁGILES

- Procesos guiados por planes (pesados):
  - P. ej. cascada, RUP, MUM
  - El plan se construye fundamentalmente al comienzo.
  - La comunicación se basa en documentos.
- Procesos ágiles:
  - P. ej. XP, SCRUM, FDD, DSDM
  - El plan se completa a medida que avanza el proyecto.
  - La comunicación está basada en conversaciones cara a cara.

FACTORES RELEVANTES PARA EL GRADO DE FORMALIDAD/AGILIDAD CONVENIENTE

- Características del proyecto:
  - Riesgos (del proyecto o asociados al uso del producto):
    - Requisitos oscuros o cambiantes (>, > agilidad)
    - Impacto fallas (>, > formalidad)
  - Relación con otros proyectos (>, > formalidad)
- Características de la aplicación
  - Criticidad (>, > formalidad)
- Características del equipo de trabajo
  - Competente (>, > agilidad)
  - Tamaño (>9..., > formalidad)
  - Scrum de scrums, no toda la comunicación es cara a cara
- Características del cliente
  - Cultura, capacidades, posibilidad de involucramiento
- Características de la relación con el cliente
  - Desarrollo interno / contratación externa (+confianza, > agilidad)

PROCESOS GUIADOS POR PLANES

- Cuando usar un proceso guiado por planes:
  - Cuando se trata de una aplicación crítica
  - En un proyecto muy grande con distintos equipos de desarrollo distribuidos.
  - Cuando los que lo van a mantener no son los mismos que los desarrollaron.

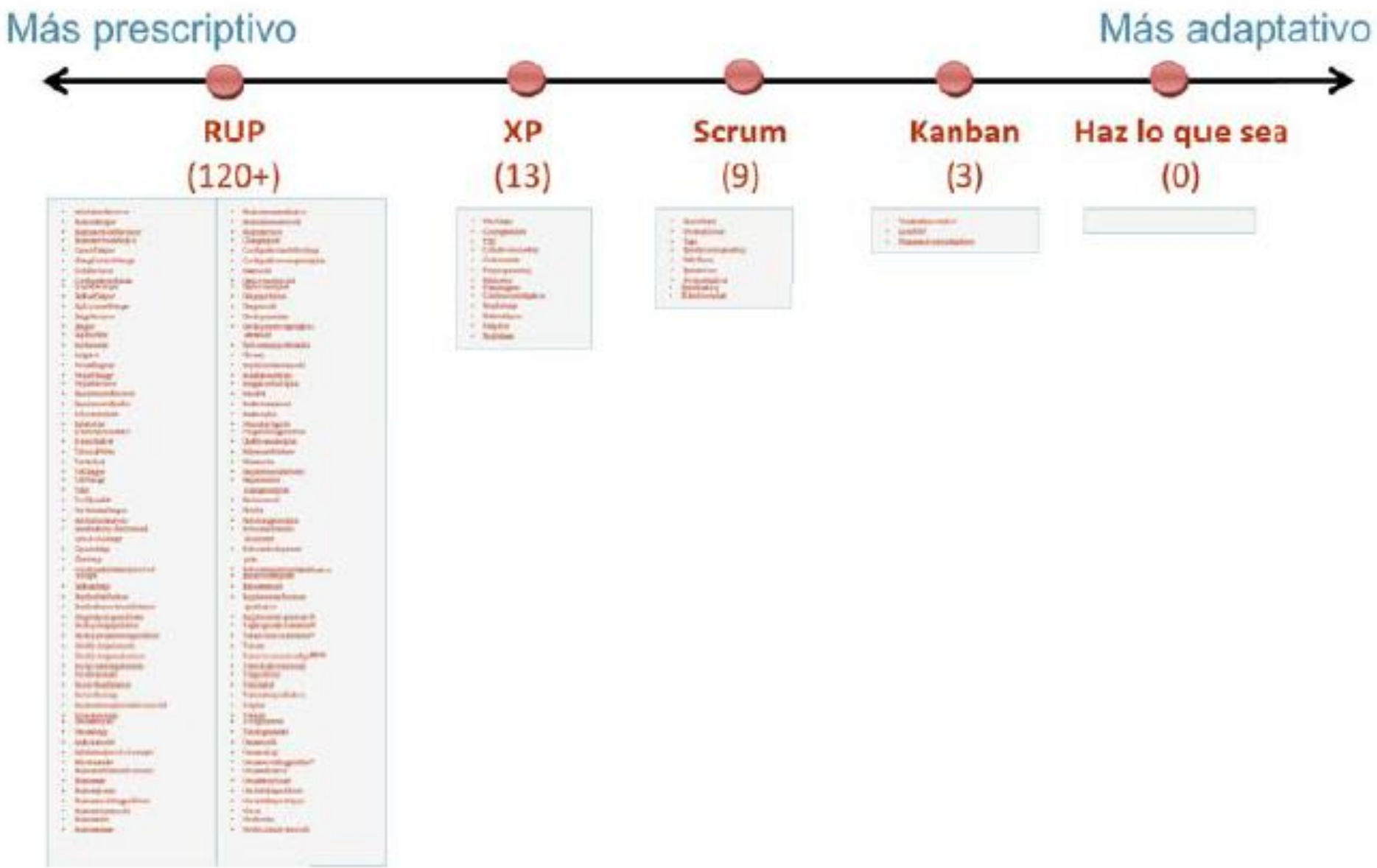
ENFOQUE ÁGIL

- Enfoque apropiado para:
  - Proyectos pequeños (< 10 desarrolladores)
  - No hay condiciones contractuales formales
  - Los requisitos están cambiando continuamente y se debe entregar funcionalidad a los usuarios de forma frecuente.

- *Agile* son una serie de principios y no una metodología o *framework* en sí.

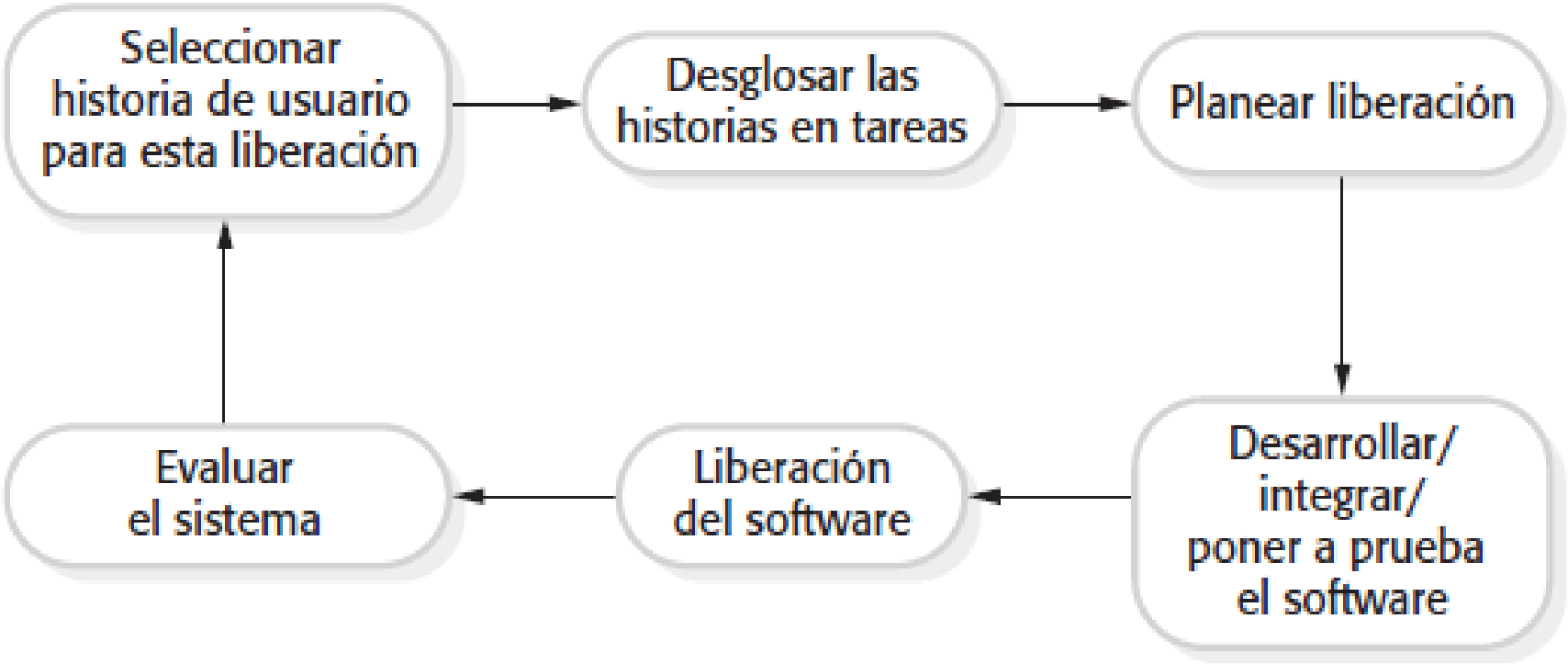
PRINCIPALES PROCESOS ÁGILES

- XP (Extreme Programming)
- Crystal (Cockburn 2002)
- Scrum (1994)
- ASD (Adaptative Software Development)
- Kanban
- FDD
- APF
- APM


Más prescriptivo
Más adaptativo
RUP
(120+)
XP
(13)
Scrum
(9)
Kanban
(3)
Haz lo que sea
(0)

eXtreme Programming (XP)

PROCESO DE XP



* Figura tomada de [Sommerville, 2011].

EXTREME PROGRAMMING (XP)

- Destinado a mejorar la calidad con objetivo de satisfacer las necesidades en constante evolución del cliente.
- Concepto original de Agile
- XP incluye
  - *sprints* cortos,
  - iteraciones frecuentes y
  - colaboración constante con los *stakeholders*.

Extreme Programming (XP)

ROLES

- Cliente
- Equipo de trabajo
  - Desarrollador
  - Verificador
    - Se focaliza en ayudar al cliente a diseñar y escribir pruebas funcionales
    - Responsable de ejecutar las pruebas regularmente y dejar los resultados disponibles para el equipo
  - Coach
    - Encargado de la ejecución técnica y de la evolución del proceso
  - Tracker
    - Registra las estimaciones y las mediciones

XP – EXTREME PROGRAMMING

- El costo del cambio no debe crecer con el tiempo
- Desarrollo en base a lo actual, sin preocuparse por el mañana
- 4 valores que deben guiar al equipo:
  - comunicación
  - simplicidad
  - retroalimentación
  - coraje
- 12 prácticas ayudan al equipo a lograr esos valores

PRÁCTICA DE XP

1. Cliente en el lugar
2. El juego de la planificación
3. Pruebas automatizadas
4. 40 horas semanales
5. Integración continua
6. Pequeñas liberaciones
7. Metáfora
8. Diseño simple
9. *Refactoring*
10. Programación en parejas
11. Estándares de codificación
12. Propiedad colectiva

CLIENTE (O SU REPRESENTANTE) EN EL LUGAR

- Disponible para contestar preguntas y resolver prioridades de pequeña escala
- Escribe los requisitos en historias
- Escribe las pruebas funcionales

EL JUEGO DE LA PLANIFICACIÓN

- Los requisitos se registran como historias en tarjetas.
- Planificación de la liberación
  - Se determina qué requisitos van en una liberación según el tiempo disponible y la prioridad relativa.
  - Participan el cliente y el equipo de desarrollo
  - Cliente: prioridad de cada historia
  - Equipo de desarrollo: Estimación de esfuerzo/tamaño
- Planificación de la iteración
  - Participa solo el equipo de desarrollo
  - Se incluyen historias en función de:
    - Prioridad dada por el cliente y
    - Estimación de esfuerzo/tamaño de quienes vayan a implementarla
  - Se divide la iteración en tareas de pocos días
  - Cada tarea tiene un responsable de que se complete

PRUEBAS AUTOMATIZADAS

- El desarrollo es guiado por el *testing* (*Test Driven Development*)
- Las pruebas son automatizadas (por software)
  - deben codificarse **antes** de escribir el código de la funcionalidad
  - en el momento de diseñarlo
- La implementación de un programa se da por concluida si todas las pruebas automatizadas corren sin defectos

40 HORAS SEMANALES

- La regla es trabajar 40 horas semanales
- Nunca trabajar extra dos semanas seguidas
  - Preocupación por las personas y por el equipo (ver desarrollo sostenible del Manifiesto Ágil)
- Ritmo sostenible.
  - De lo contrario se reduce calidad y productividad.

INTEGRACIÓN CONTINUA

- El software se integra varias veces al día
- Incrementos e iteraciones
  - Se integran para
    - detectar defectos
    - obtener productos que funcionan (tienen comportamiento) y pueden ser evaluados por el cliente
- Todos las pruebas deben correr exitosamente antes de que integrar un nuevo incremento al sistema.

PEQUEÑAS LIBERACIONES

- Liberaciones frecuentes y pequeñas.
- Poner un sistema rápidamente en producción
- Liberar versiones nuevas en ciclos cortos (desarrollo en fases)

METÁFORA

- Guiar el desarrollo compartiendo una idea simple de cómo funciona el sistema.
- Visión común, nombres y acuerdo sobre forma de abordaje.
- Ejemplo para un software que permita manejar prioridades de tarea: «Debo poder manejar las tareas como las manejo con Post-It en una pizarra».

DISEÑO SIMPLE

- Dado un conjunto de requisitos, existen muchos diseños posibles para satisfacerlos.
- XP establece que hay que elegir el más simple.
- Si se detecta complejidad innecesaria, esta debe ser removida.
- ¿Para qué complicar?
  - Los requisitos pueden cambiar y
  - un diseño simple permite obtener más rápidamente software que funcione, y que, por lo tanto, pueda ser evaluado.
- Solo de necesidades actuales
- Un caso de la política:
  - «Postergar decisiones tanto como sea responsable hacerlo» (Metodología LEAN)

REFACTORING

- Mejora continua del sw con *refactoring*.
- Reestructurar el código para
  - simplificar
  - agregar flexibilidad
  - permitir incorporar nuevos requisitos
- Va de la mano con el diseño simple

PROGRAMACIÓN EN PAREJAS

- El diseño, las pruebas automatizadas, la codificación, es realizada en parejas
  - Compartiendo un teclado y un *mouse*
- Las parejas van cambiando
  - Por ejemplo es una en la mañana y en la tarde otra
- Dentro de una pareja uno hace y el otro revisa
- Estos roles en la pareja también se rotan

ESTÁNDARES DE CODIFICACIÓN

- XP enfatiza el código como herramienta de comunicación
- Todos pueden modificar el código (propiedad colectiva) por lo que debe ser comprensible para todos
- La documentación en XP está compuesta por:
  - historias
  - pruebas automatizadas
  - pruebas funcionales
  - código

PROPIEDAD COLECTIVA

- Propiedad colectiva del código.
- Todos pueden cambiar cualquier parte del código en cualquier momento.
- Cada integrante del equipo es responsable por el sistema completo.
- Si un par está trabajando y ve la oportunidad de mejorar una porción de código, debe hacerlo si considera que vale el esfuerzo.

SCRUM

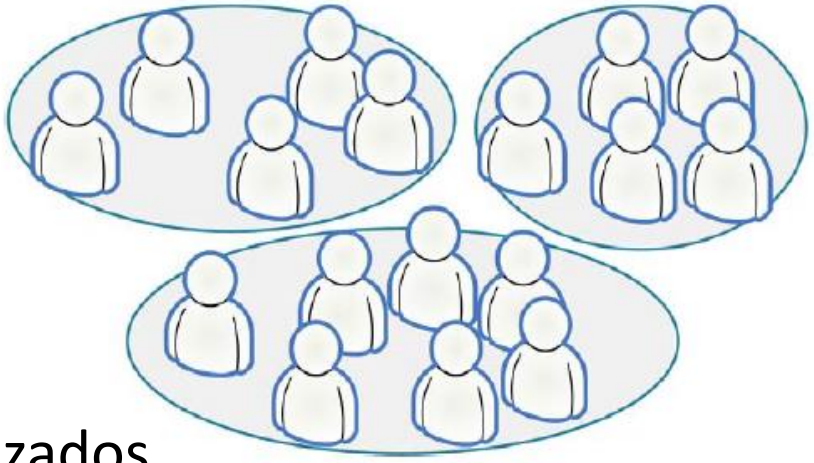
MARCO DE TRABAJO

- Scrum es un marco de trabajo de procesos:
  - no es un proceso, una técnica o método definitivo
  - es un marco de trabajo dentro del cual se pueden emplear varios procesos y técnicas
- Usado para gestionar el trabajo en productos complejos desde principios de los años 90.
- Pone el foco en la gestión ágil del proyecto.
- Surgió como metodología de desarrollo de productos nuevos, no para sw, pero ha sido muy exitosa en sw.

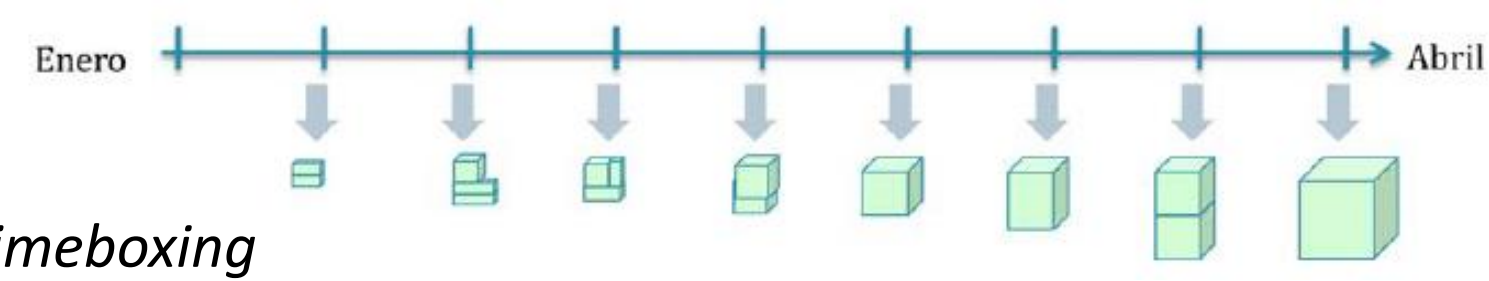
NORMAS

- No establece ninguna práctica específica de ingeniería.
- Prescribe iteraciones de duración fija (*sprints*) y equipos interdisciplinarios.
- Los *sprints* duran entre una y cuatro semanas y permiten entregar software de forma regular.
- Las responsabilidades son compartidas entre el equipo.
- Diferencias entre Scrum y RUP:
  - En RUP tienes demasiado, y se supone que quitarás aquello que no necesites.
  - En Scrum tienes demasiado poco, y se supone que añadirás el material que falta.

SCRUM

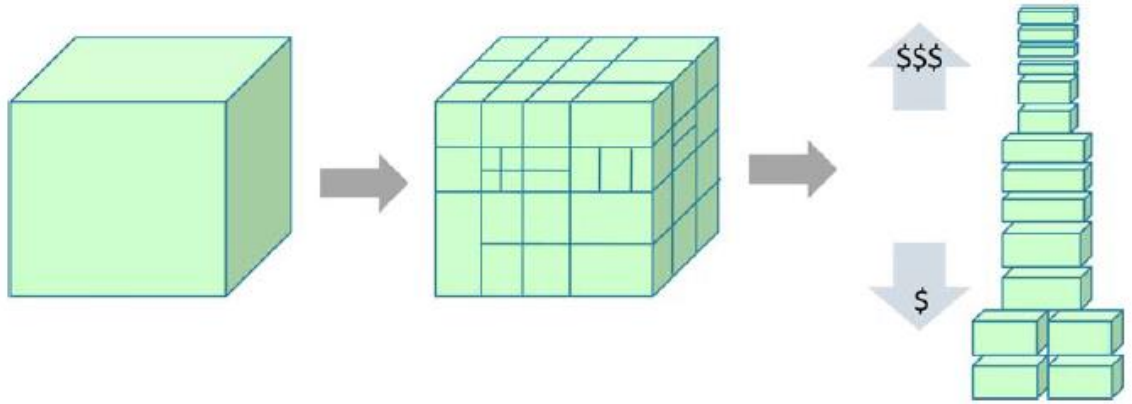


- Equipos pequeños, interdisciplinarios y auto-organizados
- Divide el tiempo en iteraciones cortas de longitud fija (*sprint*) (generalmente de 1 a 4 semanas), con código potencialmente entregable y demostrado después de cada iteración.



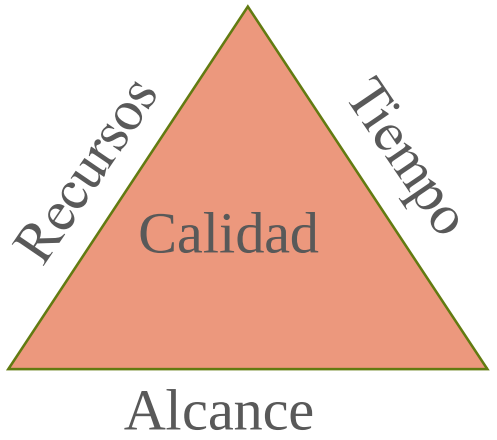
- *Timeboxing*

- Divide el trabajo en una lista de entregables pequeños y concretos. Ordena la lista por orden de prioridad y estima el esfuerzo relativo de cada elemento.



- Optimiza el plan de entregas y actualiza las prioridades en colaboración con el cliente, basada en los conocimientos adquiridos mediante la inspección del entregable después de cada iteración.
- Optimiza el proceso teniendo una retrospectiva después de cada iteración.

TIME-BOXING

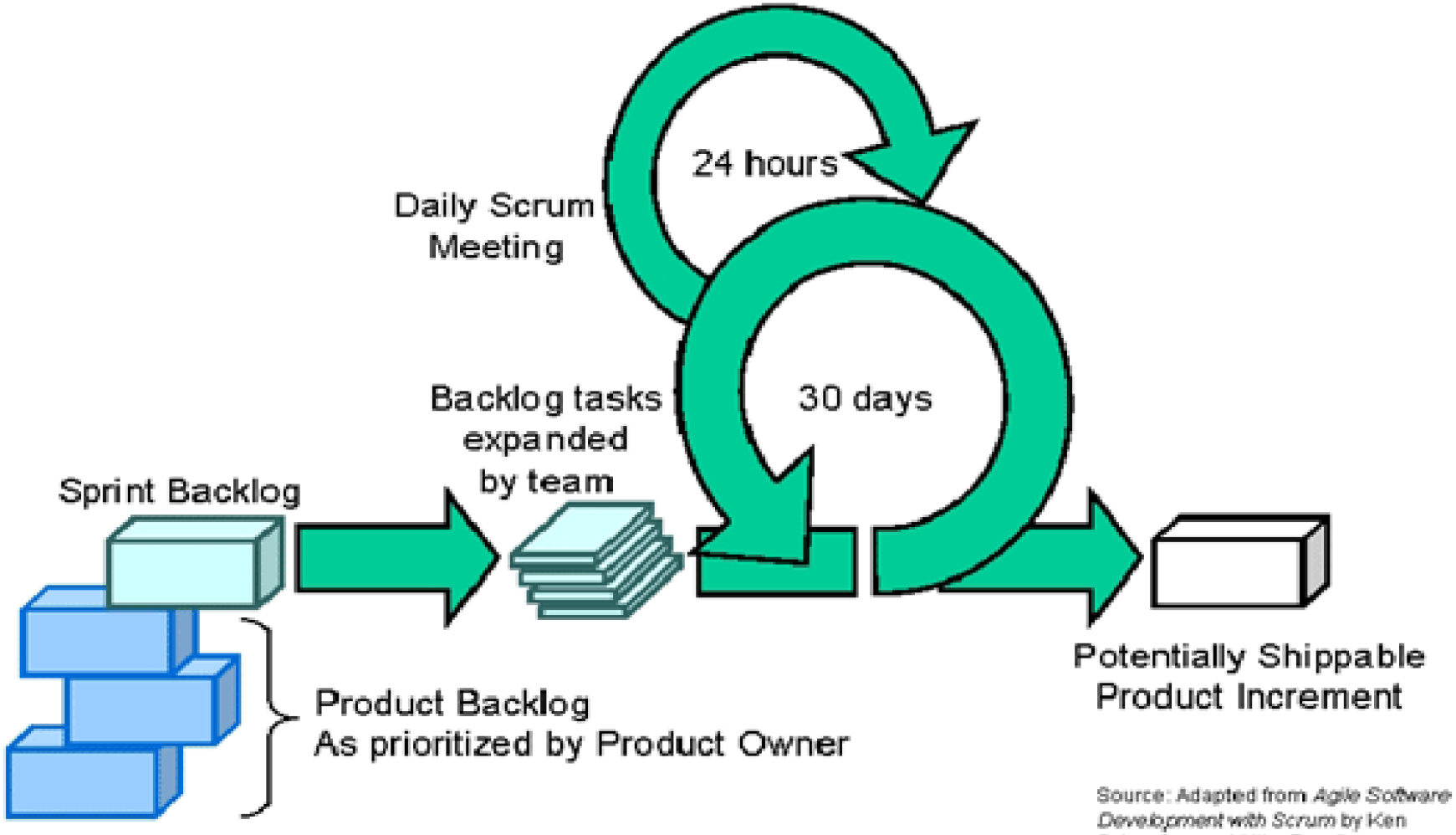


- En gestión de proyectos tenemos la triple restricción:
  - Alcance, recursos, tiempo
- Cambiar una afecta el resto de forma poco predecible. Los proyectos se gestionan a menudo por alcance fijo
  - Si no alcanza el plazo, se extiende o se agregan recursos
- Bueno trabajar en fases, pero …
  - difícil estimar cuánto debe durar cada una para cierto alcance
  - Si se atrasa ¿cuánto permitiremos que se atrase?
  - Para evitar que se estire el proyecto sin tomar medidas...

HAGAMOS AL REVÉS...

- Fijemos el tiempo
  - Dados los recursos
  - Veamos qué alcance logramos.
- Con fases de corta duración
  - La presión del cronograma está siempre presente.
  - El alcance normalmente tiene relevancia dispar (principio de Pareto) y el equipo puede poner foco en lo más relevante y olvidarse de los cambios.
  - Los cambios se pueden incluir en una fase siguiente.

SCRUM

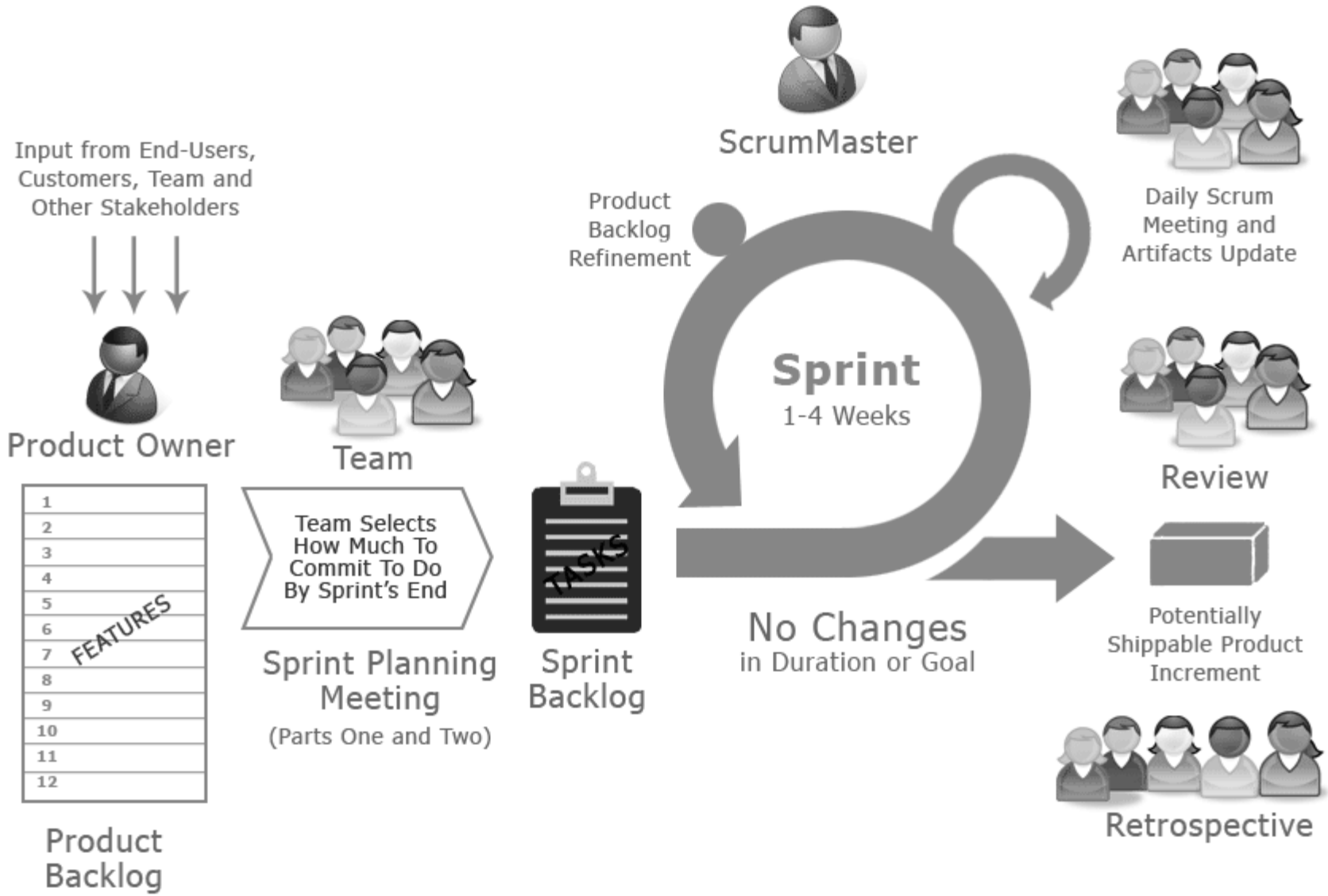


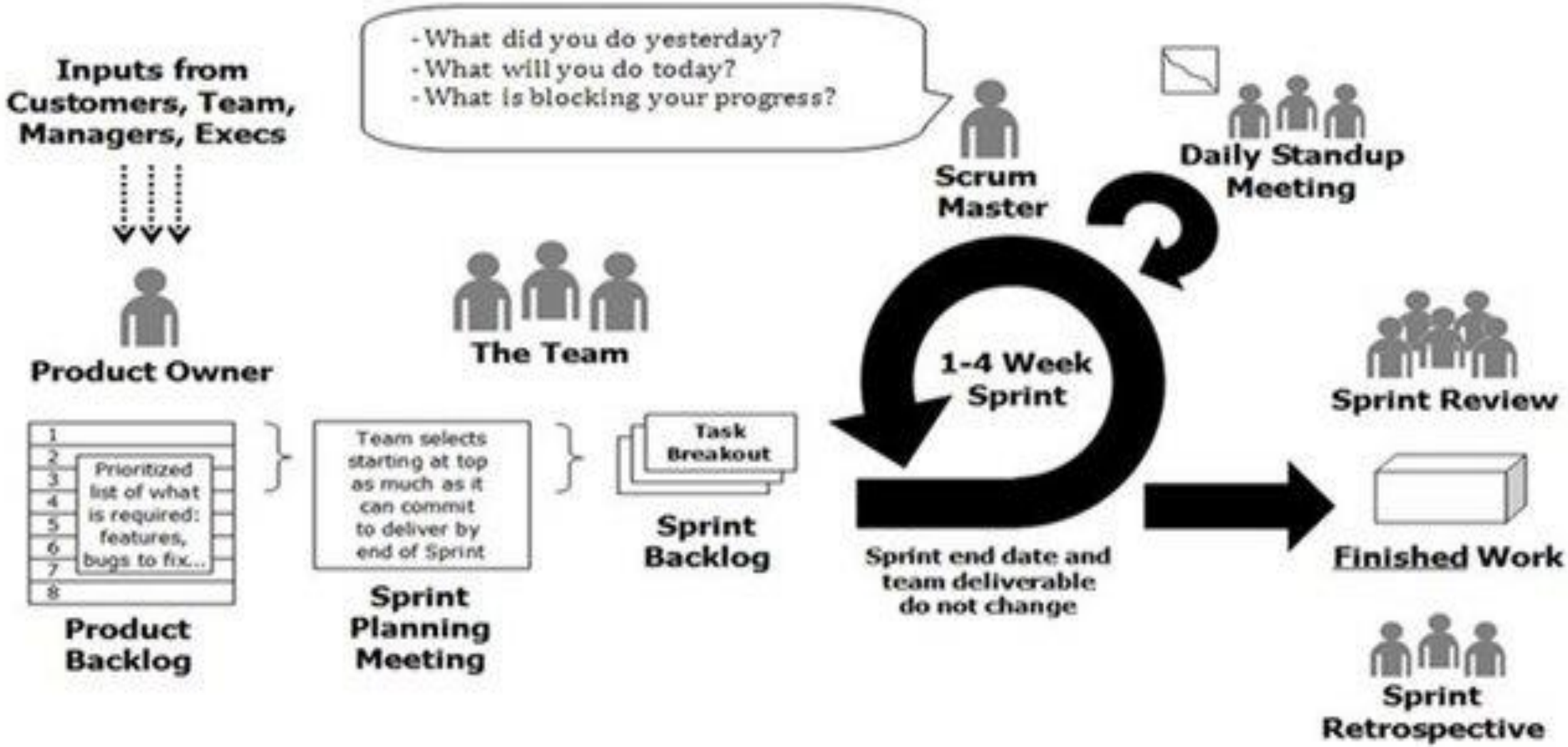
Source: Adapted from *Agile Software Development with Scrum* by Ken Schwaber and Mike Beedle.

Scrum Workflow

Bringing It All Together


The Agile Scrum Framework at a Glance
Inputs from Executives, Team, Stakeholders, Customers, Users
Product Owner
The Team
Scrum Master
Burndown/up Charts
Daily Scrum Meeting
Every 24 Hours
1-4 Week Sprint
Sprint Review
Finished Work
Sprint Retrospective
Product Backlog
Sprint Planning Meeting
Sprint Backlog
Sprint end date and team deliverable do not change
AGILEFORALL


Input from End-Users, Customers, Team and Other Stakeholders
Product Owner
Team
1
2
3
4
5
6
7
8
9
10
11
12
FEATURES
Product Backlog
Team Selects How Much To Commit To Do By Sprint's End
Sprint Planning Meeting
(Parts One and Two)
TASKS
Sprint Backlog
ScrumMaster
Product Backlog Refinement
Sprint
1-4 Weeks
No Changes
in Duration or Goal
Daily Scrum Meeting and Artifacts Update
Review
Potentially Shippable Product Increment
Retrospective


Inputs from Customers, Team, Managers, Execs
- What did you do yesterday?
- What will you do today?
- What is blocking your progress?
Scrum Master
Daily Standup Meeting
Product Owner
The Team
1-4 Week Sprint
Sprint Review
1
2
3
4
5
6
7
8
Prioritized list of what is required: features, bugs to fix...
Product Backlog
Team selects starting at top as much as it can commit to deliver by end of Sprint
Sprint Planning Meeting
Task Breakout
Sprint Backlog
Sprint end date and team deliverable do not change
Finished Work
Sprint Retrospective

COMPONENTES

- Los equipos Scrum y sus roles,
- eventos,
- artefactos y
- reglas asociadas,

EL EQUIPO SCRUM Y SUS ROLES

- El equipo Scrum consiste en
  - un dueño del producto (*product owner*),
  - el equipo de desarrollo (*development team*) y
  - un *Scrum master*.
- Los equipos Scrum son
  - autoorganizados
    - Los equipos autoorganizados eligen la mejor forma de llevar a cabo su trabajo y no son dirigidos por personas externas al equipo.
  - multifuncionales
    - Los equipos multifuncionales tienen todas las competencias necesarias para llevar a cabo el trabajo sin depender de otras personas que no son parte del equipo.
- Eres libre de añadir los roles adicionales que necesites

ROLES
PRODUCT OWNER

- Es el responsable de maximizar el valor del producto resultante del trabajo del equipo de desarrollo.
- Es la única persona responsable de gestionar el *product backlog:*
  - expresar claramente sus elementos;
  - ordenar sus elementos para alcanzar los objetivos de la mejor manera posible;
  - optimizar el valor del trabajo que el equipo de desarrollo realiza;
  - asegurar que la lista del producto es visible, transparente y clara para todos y que muestra aquello en lo que el equipo trabajará a continuación; y,
  - asegurar que el equipo de desarrollo entiende los elementos de la lista del producto al nivel necesario.
- Puede hacer el trabajo o delegarlo en el equipo de desarrollo. Pero sigue siendo el responsable de dicho trabajo.
- Es una única persona, no un comité. Podría representar los deseos de un comité, pero aquellos que quieran cambiar la prioridad de un elemento deben hacerlo a través de él.
- Toda la organización debe respetar sus decisiones. Estas se reflejan en el contenido y en la priorización de la lista del producto.

ROLES
PRODUCT OWNER

- Establece la visión del producto y las prioridades:
  - Es responsable de todas las cuestiones estratégicas de proyecto.
  - Representa la voz del cliente
  - Responsable por que el equipo aporte valor al negocio
  - Escribe las historias de usuario
  - Las prioriza y las agrega a la lista de pendientes (*backlog*)
  - Toma decisiones sobre el producto y equilibra todas las prioridades.
  - Da el foco del proyecto

ROLES
EQUIPO DE DESARROLLO

- Los profesionales que realizan el trabajo de entregar un incremento de producto «terminado» que potencialmente se pueda poner en producción al final de cada sprint (obligatorio en la revisión del sprint).

- Características:
  - Son autoorganizados. Nadie (ni siquiera el scrum master) indica al equipo de desarrollo cómo convertir elementos de la lista del producto en incrementos de funcionalidad potencialmente desplegables;
  - Son multifuncionales, esto es, como equipo cuentan con todas las habilidades necesarias para crear un incremento de producto;
  - Scrum no reconoce títulos para los miembros de un equipo de desarrollo independientemente del trabajo que realice cada persona;
  - Scrum no reconoce subequipos en los equipos de desarrollo, no importan los dominios que requieran tenerse en cuenta, como pruebas, arquitectura, operaciones o análisis de negocio; y,
  - Los miembros individuales del equipo de desarrollo pueden tener habilidades especializadas y áreas en las que estén más enfocados, pero la responsabilidad recae en el equipo de desarrollo como un todo.

ROLES
EQUIPO DE DESARROLLO

- Implementa el producto
  - Responsable por brindar un resultado de valor al final de cada *sprint*
  - Gestiona cómo lograr los objetivos marcados
  - Mejora continuamente como equipo

ROLES
EQUIPO DE DESARROLLO

- **Tamaño del equipo de desarrollo**
- Tamaño óptimo: lo suficientemente pequeño como para permanecer ágil y lo suficientemente grande como para completar una cantidad de trabajo significativa.
- Menos de tres miembros:
  - reduce la interacción y resulta en ganancias de productividad más pequeñas.
  - Los equipos de desarrollo más pequeños podrían encontrar limitaciones en cuanto a las habilidades necesarias durante un sprint, haciendo que el equipo de desarrollo no pudiese entregar un incremento que potencialmente se pueda poner en producción.
- Más de nueve miembros:
  - requiere demasiada coordinación.
  - Los grandes equipos de desarrollo generan demasiada complejidad como para que un proceso empírico les sea de utilidad.
- Los roles de dueño del producto y scrum master no cuentan en el cálculo del tamaño del equipo, a menos que también estén contribuyendo a trabajar en la lista de pendientes del sprint (*sprint backlog*).

ROLES
SCRUM MASTER

- Responsable de promover y apoyar Scrum, ayudando a todos a entender la teoría, prácticas, reglas y valores de Scrum.
- Líder al servicio del equipo scrum:
  - ayuda a las personas externas al equipo scrum a entender qué interacciones con el equipo scrum pueden ser útiles y cuáles no.
  - ayuda a todos a modificar estas interacciones para maximizar el valor creado por el equipo scrum.

ROLES
SCRUM MASTER

- **El servicio del scrum master al dueño del producto**
  - Asegurar que los objetivos, el alcance y el dominio del producto sean entendidos por todos en el equipo Scrum de la mejor manera posible;
  - Encontrar técnicas para gestionar la lista de producto de manera efectiva;
  - Ayudar al equipo scrum a entender la necesidad de contar con elementos de la lista del producto claros y concisos;
  - Entender la planificación del producto en un entorno empírico;
  - Asegurar que el dueño de producto conozca cómo ordenar la lista de producto para maximizar el valor;
  - Entender y practicar la agilidad; y,
  - Facilitar los eventos de Scrum según se requiera o necesite.

ROLES
SCRUM MASTER

- **El servicio del scrum master al equipo de desarrollo**
  - Guiar al equipo de desarrollo en ser autoorganizado y multifuncional;
  - Ayudar al equipo de desarrollo a crear productos de alto valor;
  - Eliminar impedimentos para el avance del equipo de desarrollo;
  - Facilitar los eventos de Scrum según se requiera o necesite; y,
  - Guiar al equipo de desarrollo en entornos organizacionales en los que Scrum aún no haya sido adoptado y entendido por completo.

ROLES
SCRUM MASTER

- **El servicio del scrum master a la organización**
  - Liderar y guiar a la organización en la adopción de Scrum;
  - Planificar las implementaciones de Scrum en la organización;
  - Ayudar a los empleados e interesados a entender y llevar a cabo Scrum y el desarrollo empírico de producto;
  - Motivar cambios que incrementen la productividad del equipo Scrum; y,
  - Trabajar con otros scrum masters para incrementar la efectividad de la aplicación de Scrum en la organización.

ROLES
SCRUM MASTER

- Elimina los impedimentos y proporciona liderazgo en el proceso
  - Asegura que el proceso Scrum se use de forma adecuada y que el equipo mantenga el foco durante el *sprint*
  - Ayuda a los miembros del equipo a obtener los resultados más efectivos y actuar juntos
  - Facilitador, elimina obstáculos para que el equipo genere los resultados esperados para el *sprint*
  - Rastrea el progreso
  - Habilita el debate constructivo
  - Facilita las reuniones marcadas en la metodología
  - «*Servant-leader*» del equipo
  - Se preocupa que el equipo mantenga el foco (dado por el *product owner*.)

EVENTOS

- Eventos:
  - *Sprint*
  - Planificación del *sprint*
  - *Scrum* diario
  - Revisión del *sprint*
  - Retrospectiva del *sprint*
- Todos tienen una duración máxima:
  - Una vez que comienza un *sprint*, su duración es fija y no puede acortarse o alargarse.
  - Los demás eventos pueden terminar siempre que se alcance el objetivo del evento, asegurando que se emplee una cantidad apropiada de tiempo sin permitir desperdicio en el proceso.

REUNIONES

- *Scrum* diario
- Revisión y ajuste del *backlog*
- Planificación del *sprint*
- Revisión del *sprint*
- Retrospectiva del *sprint*

SCRUM DIARIO

- En el mismo lugar y hora y comienza en hora.
- Es abierto a roles auxiliares, pero normalmente solo hablan los roles principales.
- Duración fija de 15 minutos, normalmente de pie.
- Cada integrante informa de 3 cuestiones:
  - Qué hice ayer,
  - qué pienso hacer hoy,
  - impedimentos o problemas
- El objetivo de la reunión no es resolver impedimentos o problemas, sino comunicarlos. La resolución requiere más tiempo y esfuerzo de personas más dedicadas.
- Es responsabilidad del scrum master resolver los impedimentos o problemas (en general, por fuera de la reunión).

REVISIÓN Y AJUSTE DEL *BACKLOG*

- Estimar esfuerzo/tamaño de cada historia
- Refinar los criterios de aceptación para cada una
- Descomponer historias en historias más pequeñas
- Estas reuniones
  - No pueden durar más de una hora
  - No incluye la descomposición de una historia en tareas
  - El equipo decide cuántas reuniones de este tipo precisa por semana

PLANIFICACIÓN DEL *SPRINT*

- Al comienzo de cada *sprint*
- Elegir qué trabajo será realizado
- Preparar el *sprint backlog*
- 8 horas como máximo
  - 4 h *product owner* + equipo para priorizar el *backlog*
  - 4 h del equipo para desarrollar el plan para el *sprint*

REVISIÓN DEL *SPRINT*

- Revisar el trabajo completado y no completado.
- Presentar el trabajo completado a los *stakeholders* (demo).
- 4 h como máximo.

RETROSPECTIVA DEL *SPRINT*

- Todos los miembros del equipo reflexionan acerca del *sprint* anterior
- Mejora continua del proceso:
  - ¿Qué funcionó bien?
  - ¿Qué se podría mejorar para el próximo *sprint*?
  - 3 h como máximo

SCRUM
ARTEFACTOS

- Historias:
  - pequeña descripción para documentar los requisitos funcionales y eventuales requisitos no funcionales (podría considerarse un caso de uso informal).
- *Backlog* del producto:
  - lista de pedidos pendientes
- *Backlog* del *sprint*
  - lo que vamos a incorporar durante el *sprint*
- Incremento
  - lo completado desde el comienzo
- *Burn down chart*
  - diagrama que compara el avance respecto a lo previsto
  - Burn up, cumulative flow

BACKLOG DEL PRODUCTO

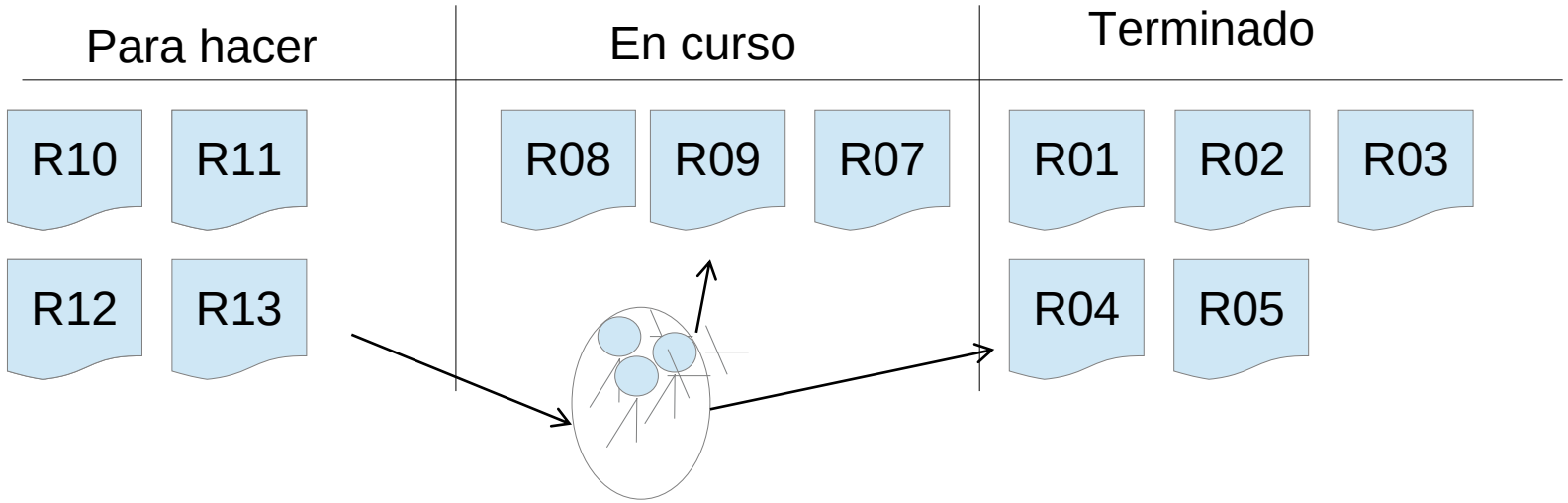
- Conjunto ordenado de requisitos pendientes para un producto
- El *product owner* es el responsable del *backlog* y establece el orden considerando:
  - riesgo,
  - valor para el negocio,
  - dependencias,
  - fecha en la que es requerido,
  - esfuerzo estimado para completarlo, etc.
- El valor para el negocio es estimado por el *product owner*
- El resto es suministrado por el equipo.

BACKLOG DEL *SPRINT*

- Conjunto de requisitos que el equipo debe atacar en el *sprint*.
- Es propiedad del equipo de desarrollo.
- El equipo va tomando las historias que están en el tope del *product backlog*, mientras piense que puede hacerlas.
- El equipo detalla para cada historia las tareas necesarias con duración de entre 4 a 16 h.
- Las tareas no se asignan, los integrantes del equipo las van tomando en el *Scrum* diario de acuerdo a las prioridades y a las habilidades de cada miembro.
- Esto promueve la autoorganización del equipo.

BACKLOG DEL *SPRINT* (CONT.)

- Se suele representar en una pizarra en 3 columnas:
  - *To do* (para hacer)
  - *In progress* (en curso)
  - *Done* (hecho, terminado)
- Resulta cómodo usar Post-it



INCREMENTO

- Es la suma de lo hecho en el *sprint* y en todos los *sprints* anteriores,
- Al fin de un *sprint* el incremento debe estar completo de acuerdo a la definición del equipo para «terminado».
- Debe estar utilizable, independientemente de que el *product owner* decida liberarlo o no.

KANBAN

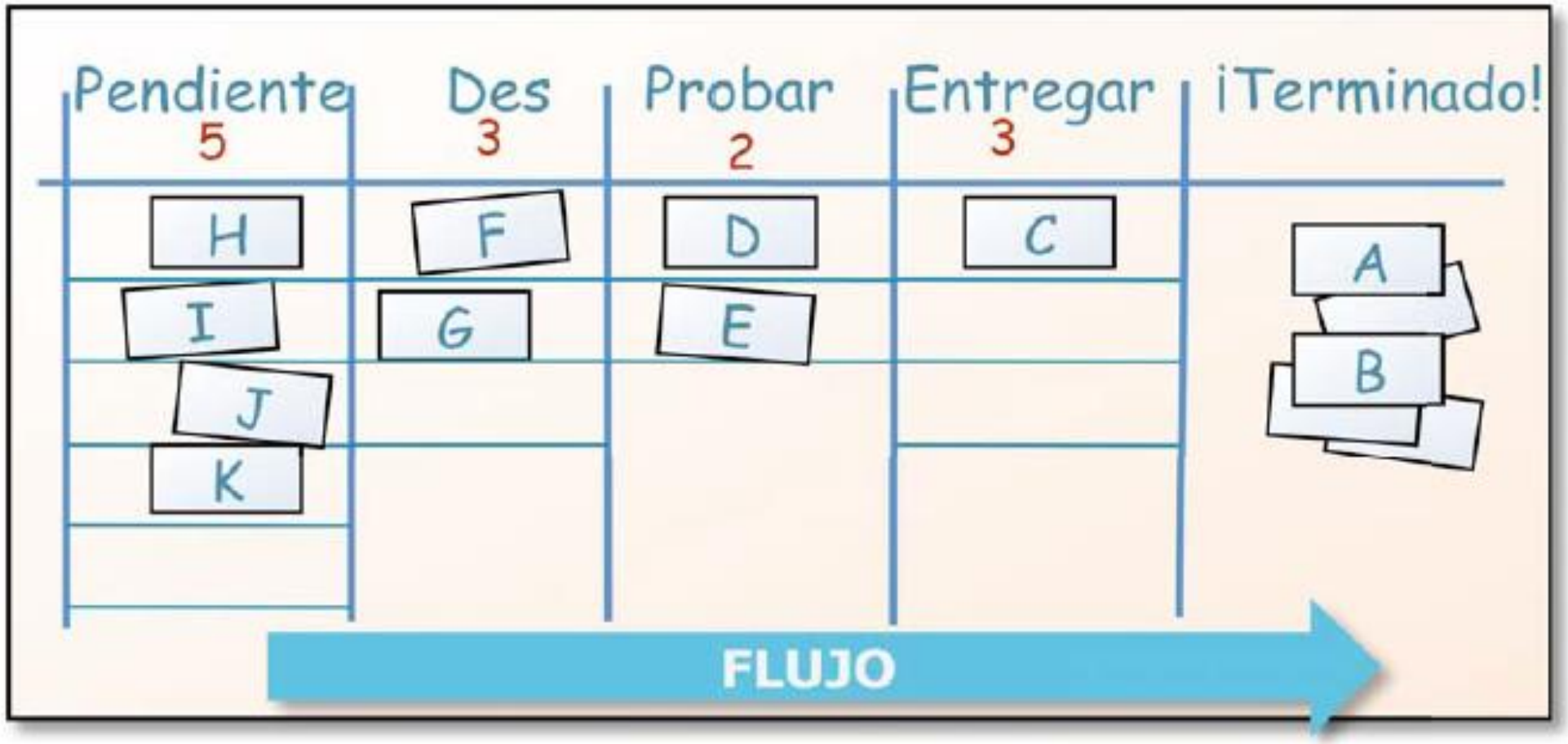
NORMAS

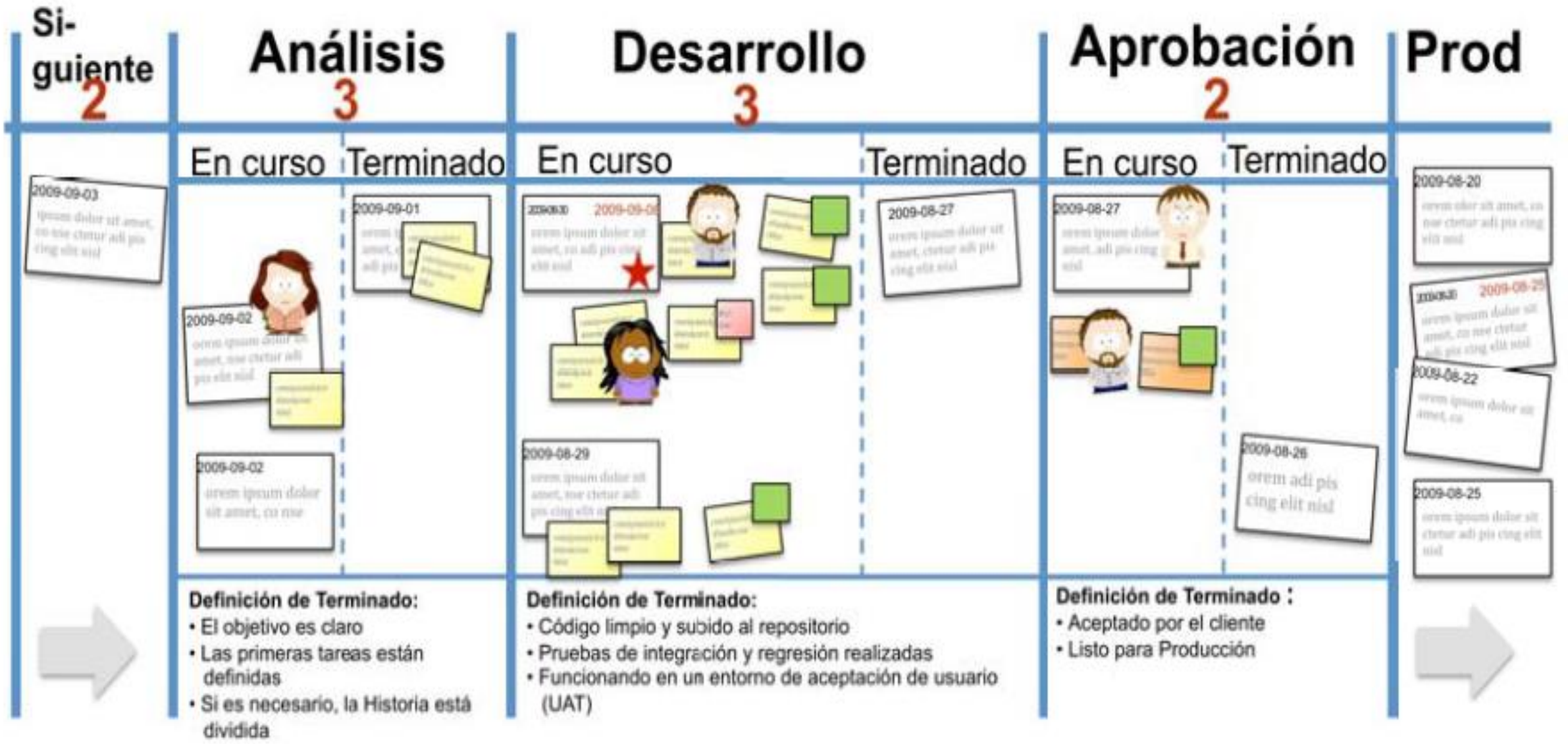
- Kanban deja casi todo abierto.
- Las únicas normas son
  - Visualizar el flujo de trabajo
    - Partir el trabajo en pedazos, anotar cada uno en una tarjeta y ponerla en la pared
    - Usar columnas con nombre para mostrar en qué etapa del flujo está cada ítem
    - Usar representaciones visuales de sus tareas con notas adhesivas de colores o utilizando servicios como Hygger, Jira o Trello**.**
  - Limitar el trabajo en curso (*work in progress*)
    - Asignar límites explícitos a la cantidad de elementos en cada estado del flujo del trabajo (limita el tamaño de las colas).
    - En trabajo intelectual tener más de una tarea en curso afecta negativamente la productividad.

KANBAN

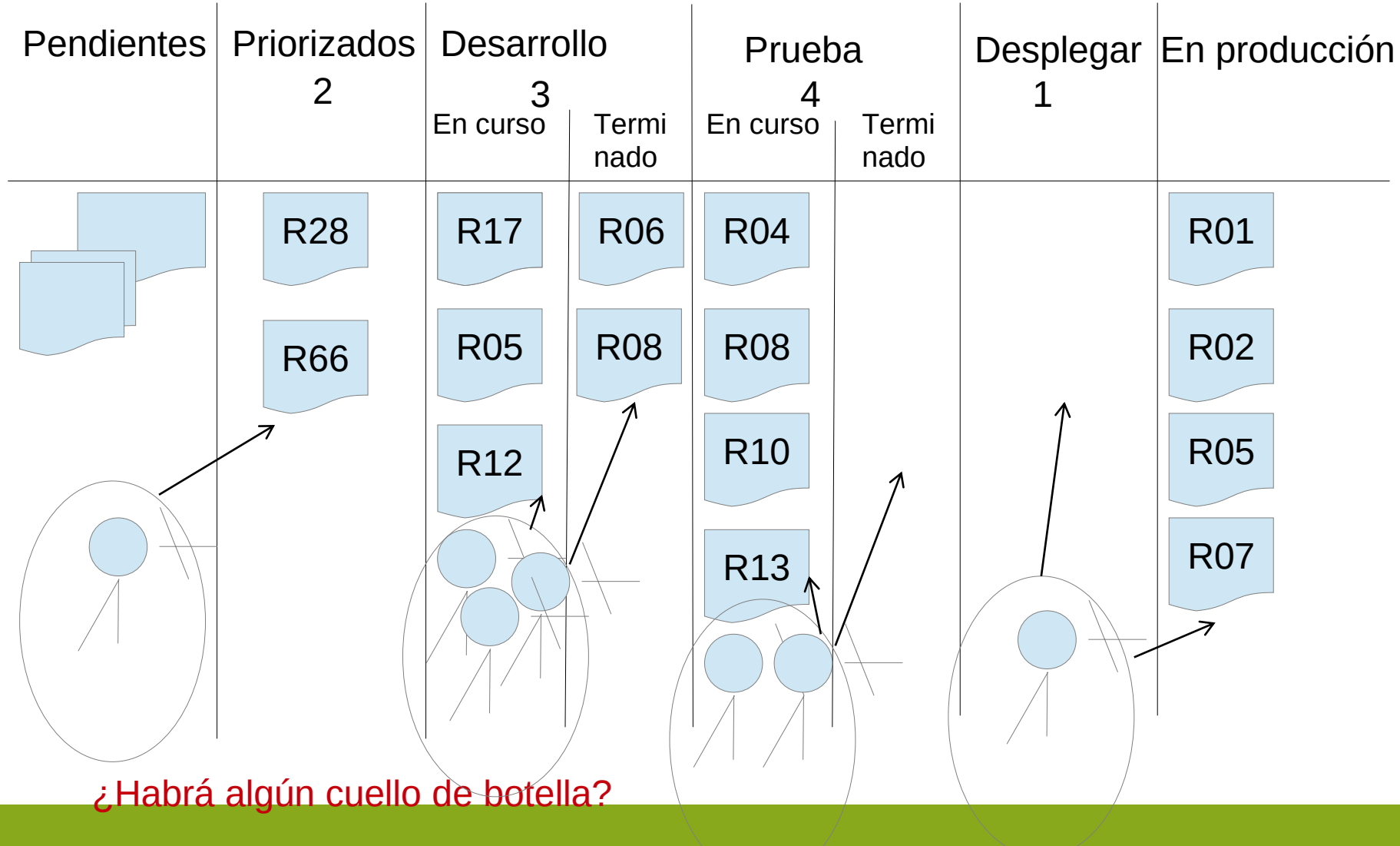
- Kanban (japonés) = tarjeta visual o valla publicitaria.
- Apropiado para gestionar
  - sistemas en producción, con objetivo claro de promover una mejora continua.
  - el trabajo que requiere un rendimiento constante.
- Medir el *lead time* = el tiempo que le lleva a un ítem desde que surge hasta que termina el trabajo (tiempo medio para completar un elemento, a veces llamado «tiempo de ciclo»)
- Ajustar el proceso para volver ese tiempo tan pequeño y predecible como sea posible
- Las pizarras Kanban son también la solución ideal para que los jefes de producto mantengan el *product backlog*.

CUADRO DE SEGUIMIENTO


Pendiente
5
Des
3
Probar
2
Entregar
3
¡Terminado!
H
I
J
K
F
G
D
E
C
A
B
FLUJO


Si-
guiente
2
Análisis
3
Desarrollo
3
Aprobación
2
Prod
En curso
Terminado
En curso
Terminado
En curso
Terminado
2009-09-03
2009-09-01
2009-09-02
2009-09-02
2009-08-27
2009-08-27
2009-08-26
2009-08-20
2009-08-22
2009-08-25
Definición de Terminado:
• El objetivo es claro
• Las primeras tareas están definidas
• Si es necesario, la Historia está dividida
Definición de Terminado:
• Código limpio y subido al repositorio
• Pruebas de integración y regresión realizadas
• Funcionando en un entorno de aceptación de usuario (UAT)
Definición de Terminado:
• Aceptado por el cliente
• Listo para Producción

CUADRO DE SEGUIMIENTO. EJEMPLO



¿Habrá algún cuello de botella?

ROLES

- Kanban no prescribe ningún rol
- Eres libre de añadir los roles adicionales que necesites

CONCLUSIONES

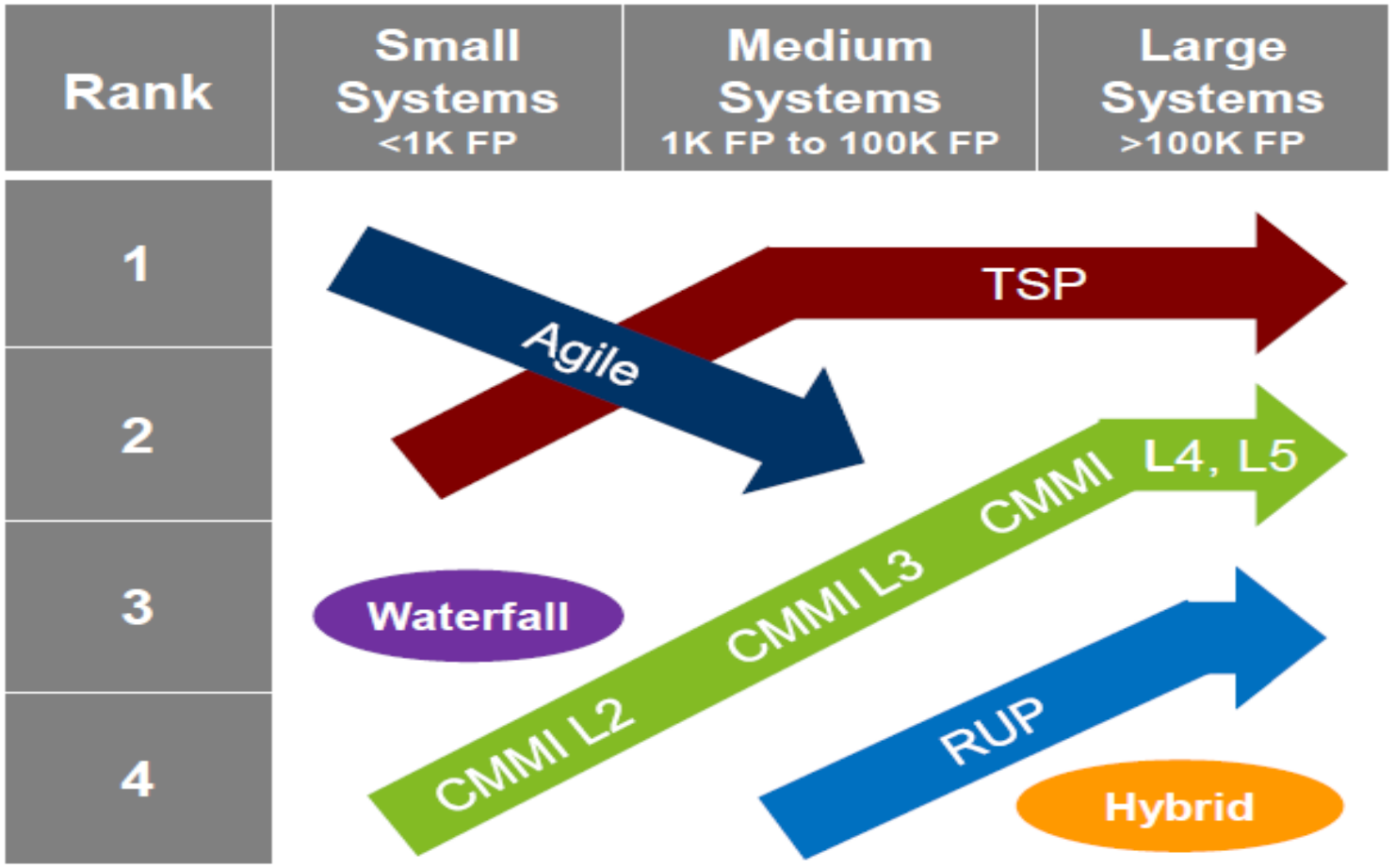
VENTAJAS

- Muy adecuadas :
  - Para equipos pequeños (hasta 7 integrantes)
  - En caso de requisitos oscuros o cambiantes
  - Para innovación en productos
- Manejan muy bien el cambio
- Permiten obtener rápidamente resultados de valor
- Eficientes
  - poco desperdicio
  - en especial en comunicación

PROBLEMAS Y LIMITACIONES

- Requiere personal competente
- Dificultades si la escala es mayor
- Si bien existen Scrums de Scrums, la gestión se complica...
- El mantenimiento de lo desarrollado con metodologías ágiles puede plantear problemas
- Depende de la tecnología que se use
  - P. ej. la automatización de pruebas de XP no es viable con cualquier herramienta de desarrollo
- No siempre es fácil conseguir un «cliente en el lugar» o un «representante del cliente»
- Puede plantear dificultades contractuales
  - El cliente ¿qué contrata?

DESEMPEÑO RELATIVO DE «AGILE»



Development practices by size of application in function points (FP) [1] [2]

(1FP ≈ 30 to 50 SLOC)

CUESTIONES QUE CAMBIAN

- La modalidad de la gestión
  - Rol del gerente
  - Rol del equipo de trabajo
  - «Empoderamiento» del equipo
- La relación con el cliente
  - Cuestiones contractuales
  - Confianza
- La conformación del equipo de trabajo
  - Mayor importancia de la capacidad de aprendizaje
  - Los proyectos son menos previsibles, por lo que la asignación de especialistas es más dinámica, se requiere a ellos en función de la evolución del proyecto

REFERENCIAS

- http://agilemanifesto.org
- Kent Beck (1999). Extreme Programming Explained: Embrace Change;
- Takeuchi, Hirotaka et al. (1986). The New New Product Development Game. Harvard Business Review, www.scrumalliance.org.
- Henrik Kniberg & Mattias Skarin (2010). Kanban and Scrum – making the most of both.
- http://www.nebulon.com/articles/fdd/latestfdd.html.
- A.Cockburn (2002). Agile Software Development.
- Jim Highsmith (2010). Agile Project Management- Creating Innovative Products, 2nd edition.
- Capers Jones (2010). Software Engineering Best Practices: Lessons from Successful Projects in the Top Companies.

PMI

FASES

- Fases secuenciales (iteraciones):
  - Cada iteración tiene como entregable código operativo.
  - Todos los procesos (análisis, diseño, implementación, verificación) se realizan dentro de una sola iteración para producir un incremento de código.
- Fase más grande llamada *release:*
  - Su entregable principal es un conjunto de los incrementos de código que puedan ser puestos en producción o entregados el cliente.

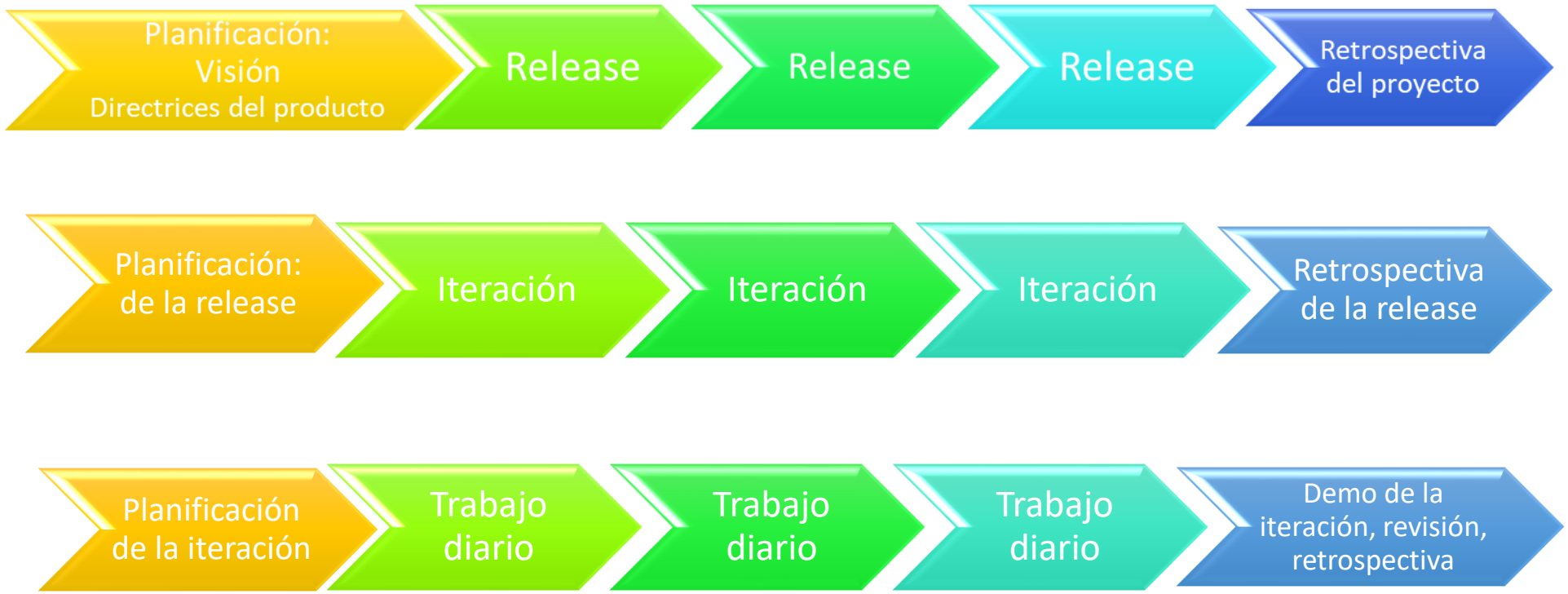
ITERACIONES

- Cantidad de iteraciones
  - determinada por el cliente, con base en lo que este defina como la mínima cantidad de funcionalidad aceptable para una *release*.
- Longitud de las iteraciones
  - Generalmente entre 1 y 4 semanas.
  - Está determinadas por el cliente.
  - Factores que afectan la duración de las iteraciones:
    - La volatilidad de la industria
    - La cantidad de riesgo
    - La claridad de la visión

ITERACIONES

- Primera iteración:
  - En proyectos ágiles en la primera iteración hay un proceso de planificación.
  - El entregable es un plan del proyecto a alto nivel.
  - Tmb. puede haber un primer incremento de código entregado.
- Iteración final (de una *release* o de varias):
  - se preparar al producto para su entrega,
  - retrospectiva final del proyecto y
  - otros procesos de cierre.
  - [Pero esto tmb lo hago en cada iteración]
- Cada fase (iteración) debe tener
  - una iniciación formal en la que se defina cuáles son los entregables esperados de esa fase. Las iteraciones ágiles comienzan con una reunión de planificación para definir qué se va a completar en la iteración una revisión formal al final para concluir la fase en la que se obtenga el permiso de continuar o se tome la decisión de detener el proyecto.
  - y terminan con una revisión para aprender de los eventos y obtener la aceptación del cliente por los *features* que se entregaron. Durante la revisión, el proyecto puede ser cancelado o aprobado para continuar, o se puede pedir una liberación que se puede implementar inmediatamente o en la siguiente iteración.

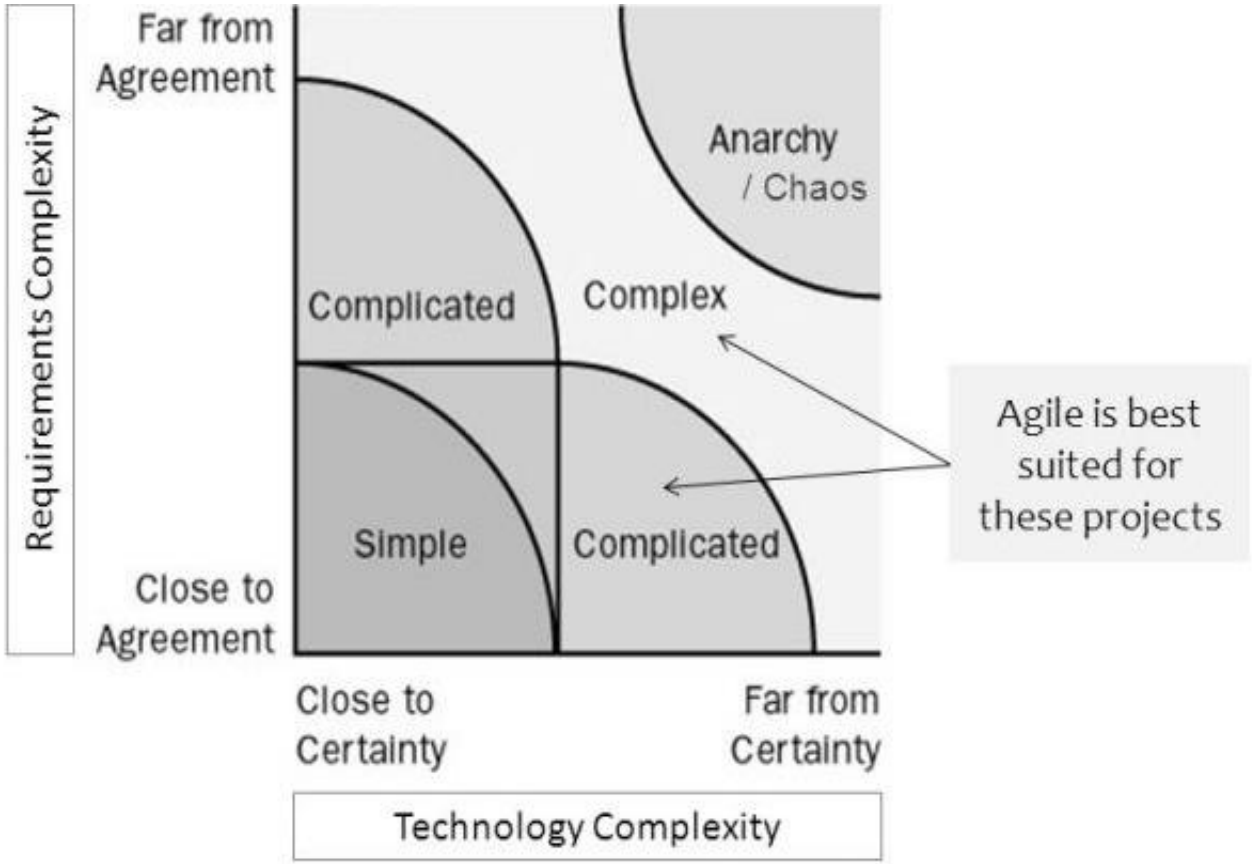
EL FRACTAL ÁGIL



MAPEO DE LOS PROCESOS DE PMI AL FRACTAL ÁGIL

	Grupos de procesos de gestión de proyectos				
Fractal ágil	**Inicio**	**Planificación**	**Ejecución**	**Seguimiento y control**	**Cierre**
Proyecto	Caso de negocio o estudio de factibilidad	Kickoff del proyecto y reunión de visionamiento. Planificación del hoja de ruta de la *release*	Entrega iterativa e incremental de sw operativo	Revisiones periódicas de los entregables, de avance y del proceso	Retrospectiva del proyecto
Release	Definición de la hoja de ruta y de la *release*	Reunión de planificación de la *release*	Entrega iterativa e incremental de sw operativo.	Revisiones periódicas de los entregables, de avance y del proceso.	Retrospectiva de la *reléase.*
Iteración	Reunión de planificación de la iteración	Reunión de planificación de la iteración	Desarrollo y verificación de los *features*	Panel de tareas, gráficas de burn down, reuniones diarias de pie, aceptación de los *features* terminados.	Demostración, revisión y retrospectiva de la iteración.

LA INCERTIDUMBRE INFLUYE EN LA METODOLOGÍA



Source: "Strategic Management and Organizational Dynamics " by Ralph D Stacey in Agile Software Development with Scrum by Ken Schwaber, Mike Beedle

- Los proyectos más adecuados para ser abordados por la metodología ágil son aquellos que:
  - Requieren de investigación
  - Tienen altas posibilidad de sufrir cambios
  - Presentan requisitos desconocidos o poco claros
  - Tienen un propósito difícil de describir

LAS PRÁCTICAS AGILES MÁS IMPORTANTES

- Planificación de la iteración
  - La capacidad de cada equipo es diferente. Se debe hacer una estimación de la capacidad.
  - Prácticas de ejecución que ayudan a los equipos a entregar valor:
    - Integración continua
    - Test en todos los niveles
    - ATDD test de aceptación.
    - TDD (test driven development) y BDD (behavior driven development)
    - *Spikes*. Útiles para aprender sobre elementos técnicos o funcionales críticos.
- Preparación del *backlog* (lista del trabajo presentado en forma de historias de usuario)
  - No es necesario tener todo el trabajo que el proyecto requiera, sino solo el que se va a desarrollar en siguiente sprint o iteración. Esta es tarea del *product owner.*
- Refinamiento del *backlog*
  - Es la preparación de historias de usuario para la siguiente iteración.
  - Si no se conocen las dependencias entre tareas, el *product owner* puede solicitar al equipo la realización de un *spike*, para conocer los riesgos.
  - El equipo no debería de emplear más de una hora a la semana en refinar historias. Lo contrario podría indicar que al equipo le faltan habilidades técnicas necesarias para evaluar y refinar el trabajo.

LAS PRÁCTICAS AGILES MÁS IMPORTANTES

- Reuniones diarias
  - Las mini reuniones diarias (de no más de 15 minutos) se usan para
    - declarar lo trabajado,
    - lo que se va a trabajar,
    - hacer aflorar posibles problemas.
  - No se deben convertir en sesiones de análisis de situación del proyecto
  - ni tampoco se deben usar para tratar de resolver problemas.
- Demos/reviews
  - Para proporcionar *feedback* al *product owner*, se organizan demos de lo desarrollado en cada iteración.
  - Deben ser frecuentes para así poder dar con la dirección adecuada.
- Retrospectivas
  - Permite al equipo aprender, mejorar y adaptar su proceso.
  - No es necesario que sea al final de una iteración de dos semanas; se puede solicitar en cualquier momento.
  - Busca las causas raíz de los problemas sufridos y se desarrolla un plan de acción para solucionarlos.