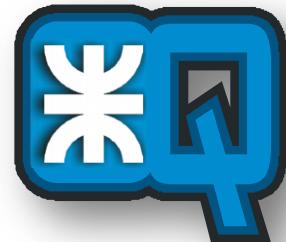


Sprint release: Avanzados

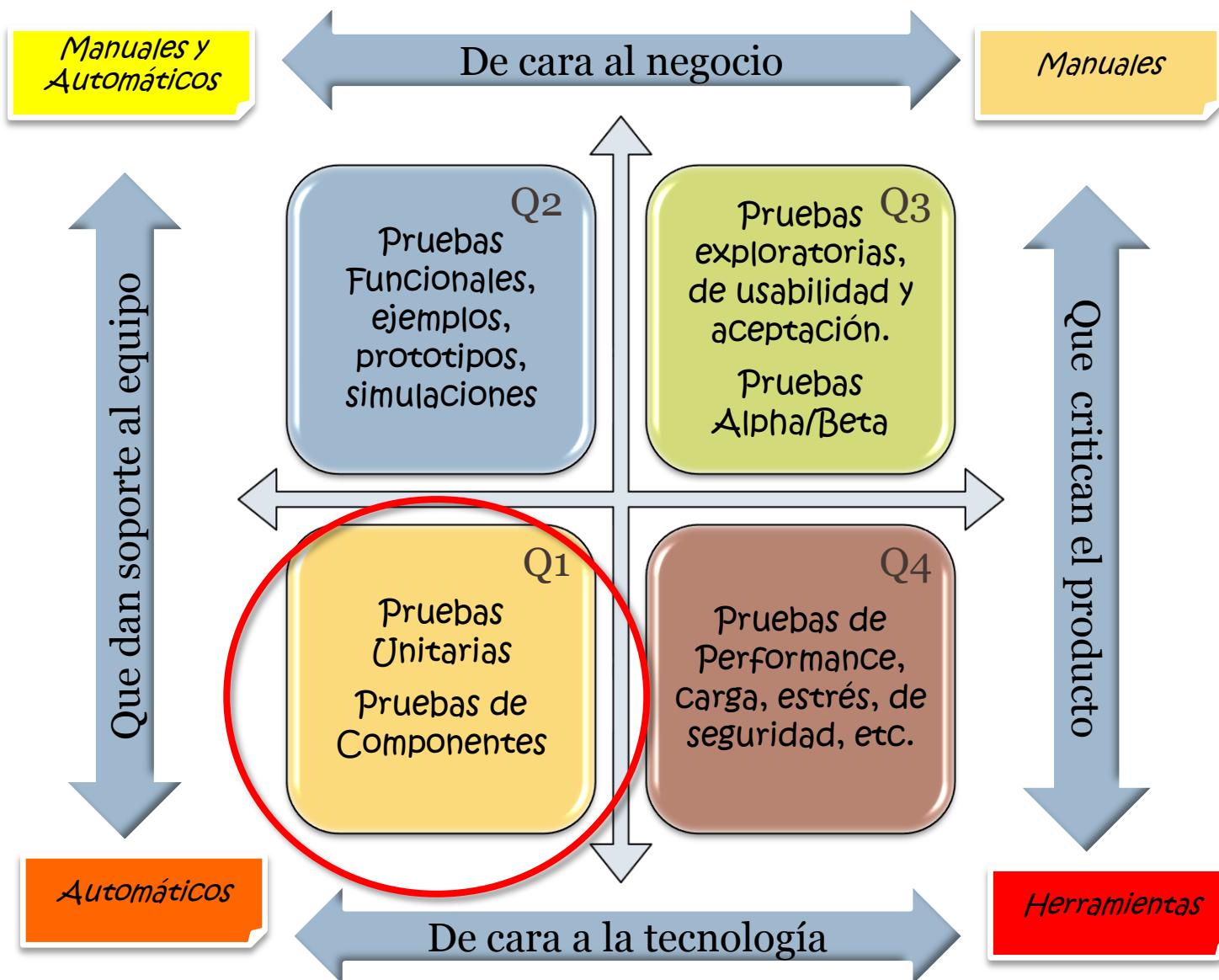
- Mgtr. Diego Rubio
- Mgtr. Natalia Andriano
- Ing. Juan Pablo Bruno
- Ing. Mauricio Silclir



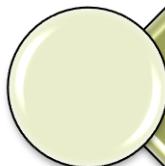
Objetivos específicos

- Desarrollar conceptos avanzados acerca de una técnica existente para la **ingeniería de requerimientos** siguiendo una metodología ágil. TDD.
- Introducir al estudiante a aspectos relacionados con la **integración continua**
- Permitir al estudiante conocer cuáles son las actividades relacionadas al **final de una iteración, entrega del producto** y las características a tener en cuenta. Retrospectiva. Release.
- Introducción a otras metodologías ágiles. **Lean, Kanban, XP.**

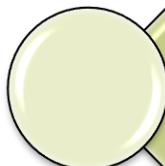
Los cuadrantes del Testing Ágil



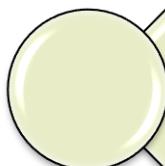
Beneficios del Q1



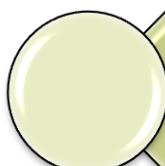
Foco en la calidad del código interno



Provee retroalimentación instantánea

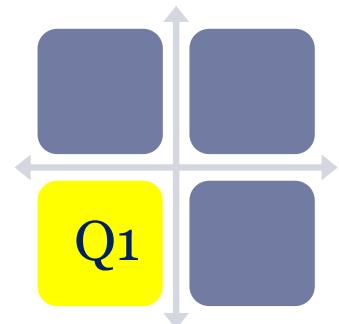


TDD incrementa la confianza en el diseño



Construye la “testeabilidad” dentro del código

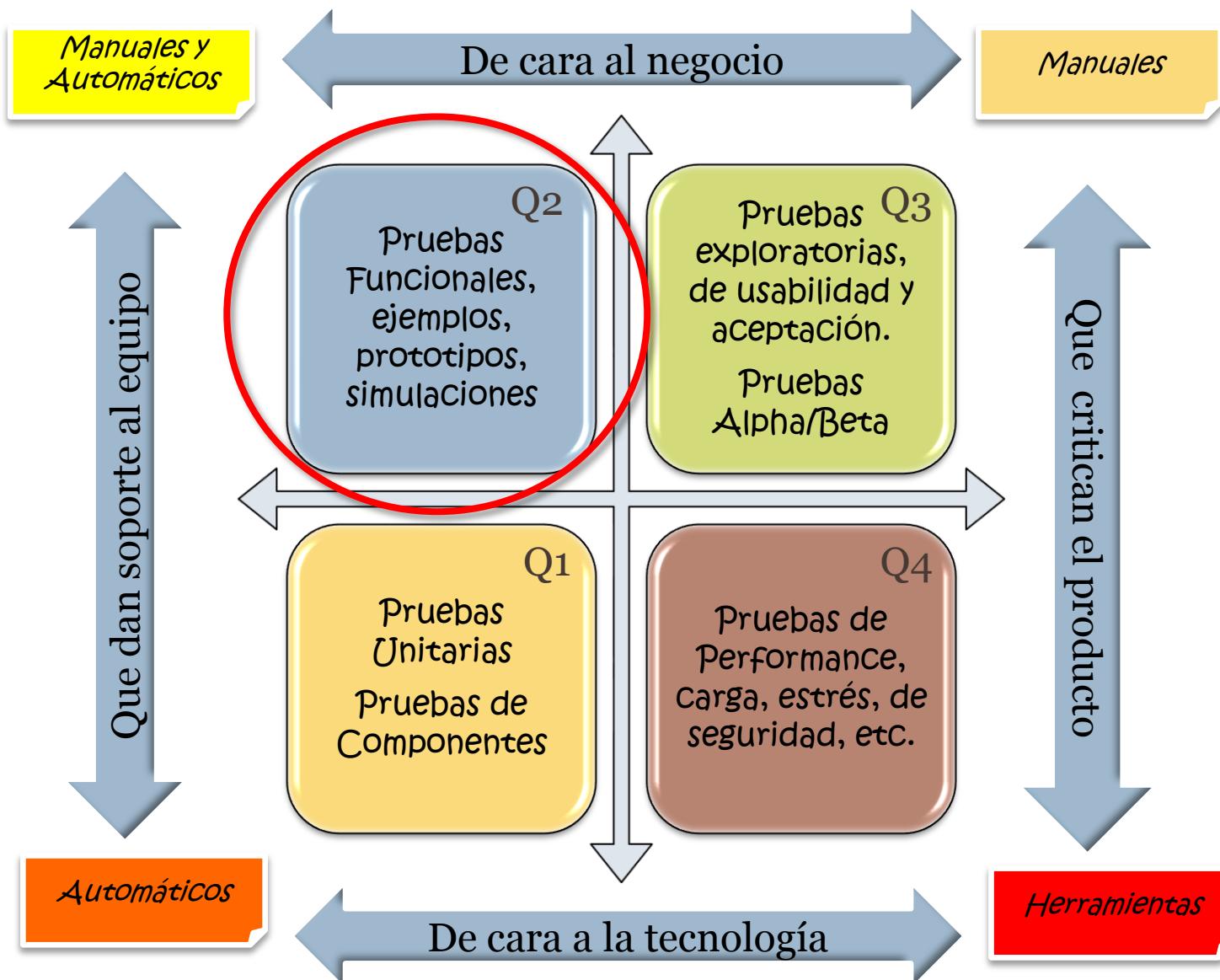
Primer Cuadrante (Q1)



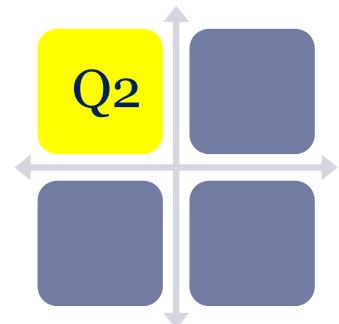
- Pruebas Unitarias
 - Desarrollo/Diseño guiado por Pruebas, más conocido como Test Driven Development/Design (TDD)
- Pruebas de Componentes
- *Estas pruebas deben automatizarse*



Los cuadrantes del Testing Ágil



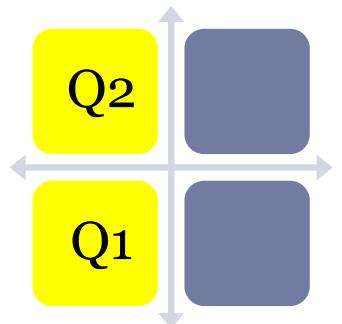
Segundo Cuadrante (Q2)



Desarrollo guiado por comportamiento (BDD)

Prácticas de BDD

- Involucrar a los interesados en el proceso
- Usar ejemplos para describir el comportamiento de la aplicación, o de unidades de código
- Automatizar estos ejemplos para proveer un rápido feedback y tests de regresión

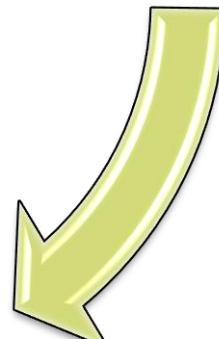
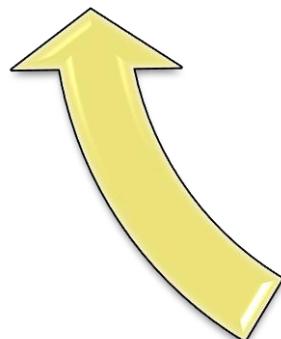


Desarrollo Guiado por Pruebas

1. Escriba la prueba que va a fallar



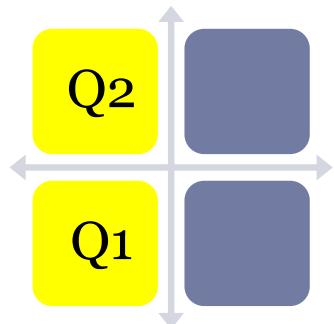
2. Implemente la funcionalidad para que la prueba pase



Refactorice

Test-Driven Development (TDD)

- NO es un método de **testing**, sino de **desarrollo**
- NO reemplaza a las pruebas de performance, rendimiento, ni usabilidad
- El objetivo es: ***“Código limpio que funciona”***
- Escribir los tests **antes** que el código, y **refactorizar** incrementalmente



No hay código sin pruebas asociadas

El código se origina y permanece sólido

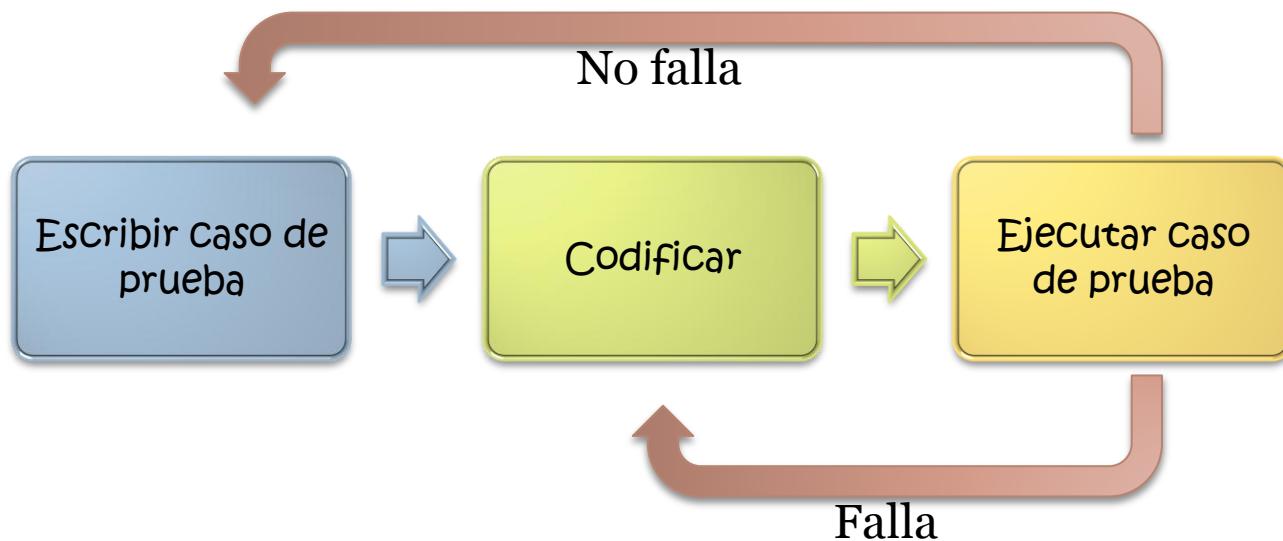
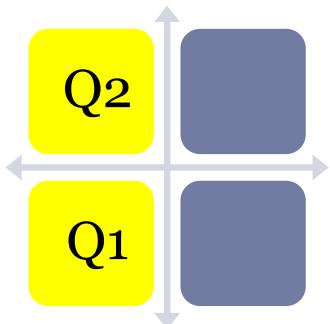
Las pruebas perduran

Las pruebas son documentación

Efecto psicológico

Test-Driven Development (TDD)

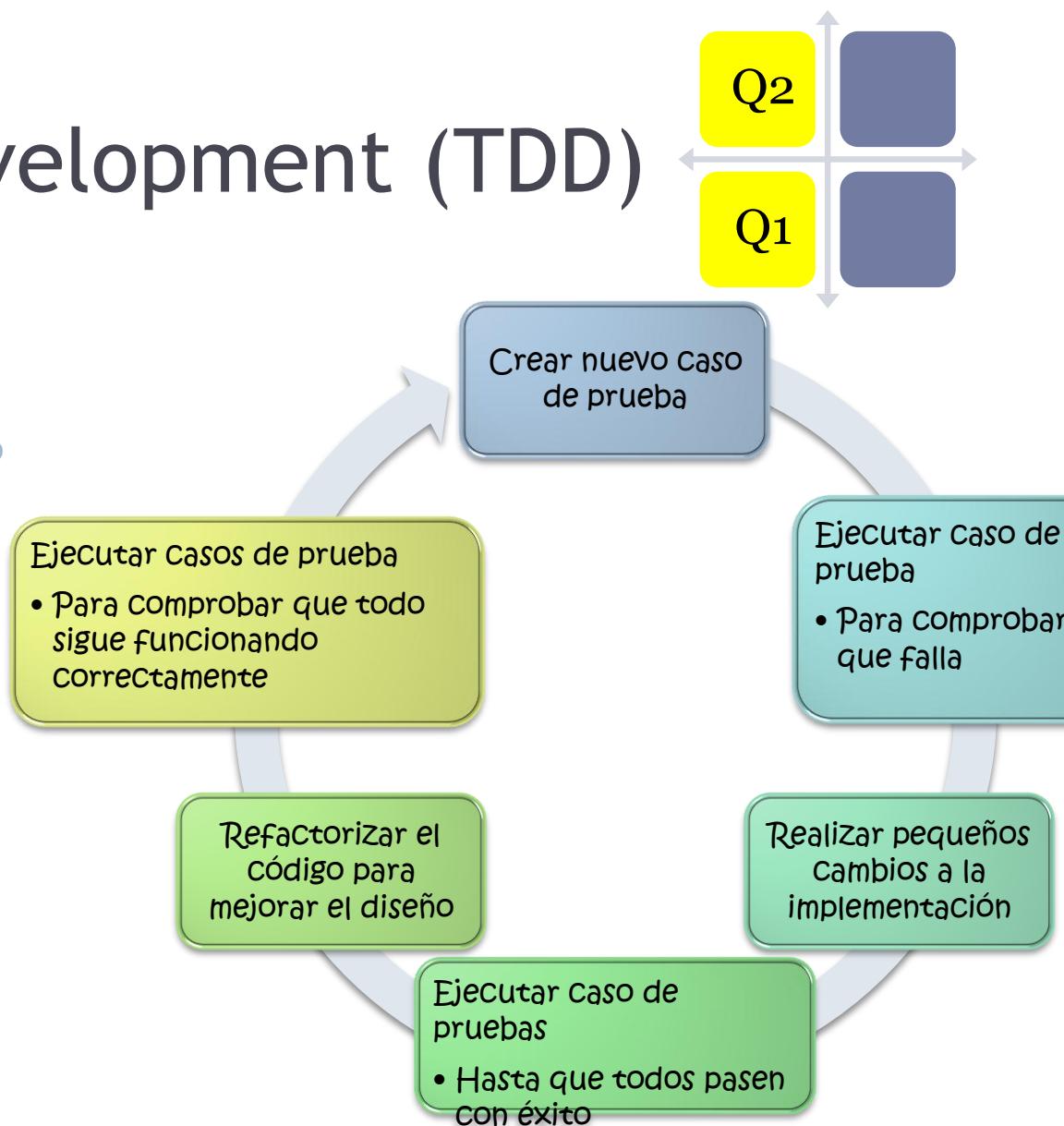
Consiste en implementar las pruebas de unidad antes incluso de comenzar a escribir el código del módulo



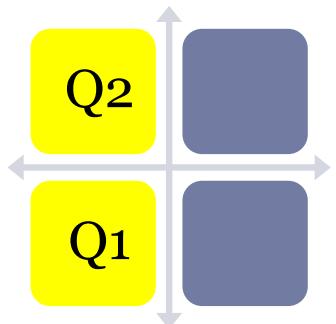
Las pruebas de unidad consisten en comprobaciones manuales o automatizadas que se realizan para verificar que el código correspondiente a un módulo concreto de un sistema de software funciona de acuerdo a los requerimientos

Test-Driven Development (TDD)

- Beneficios:
 - Los casos de prueba sirven como documentación del sistema
 - Se hace hincapié en el diseño de la interfaz de un módulo antes de pensar en la implementación
 - La ejecución de los casos de prueba se realiza en forma automatizada



Test-Driven Development (TDD)



Difícil de utilizar en
situaciones que
requieran test
funcional completo

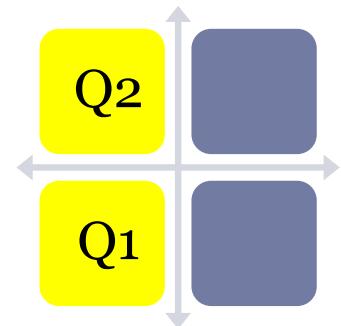
El nivel de cobertura y
el detalles del testing
logrado pueden no ser
creados tan fácilmente
en ocasiones
posteriores.

Desventajas

El soporte
gerencial es
esencial.

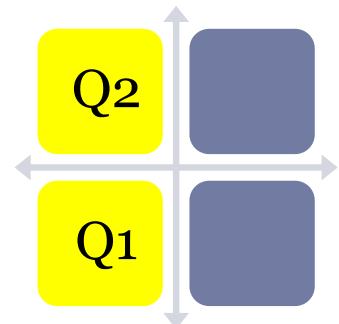
Los unit test son
creados por el
desarrollador que
es el que genera
también el código

Jbehave en acción



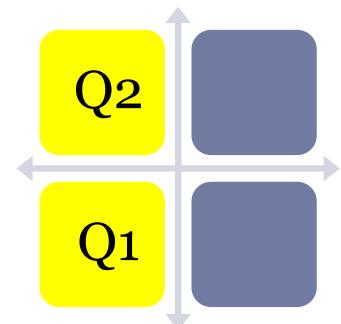
- **Historias**
 - Descripciones textuales de un conjunto de escenarios
- **Pasos (Steps)**
 - Implementación Java de cada una de los pasos de los escenarios.
- **Reportes**
 - Después de cada ejecución podemos acceder a un completo conjunto de reportes para evaluar los resultados

Jbehave ejecución



- El motor de ejecución es **JUNIT** lo que facilita la integración con el proceso de desarrollo.

Historias



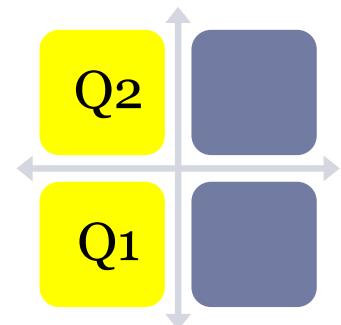
Narrative:

In order to show the browsing cart functionality
As a user
I want to browse in a gallery

Scenario: Browsing around the site for items

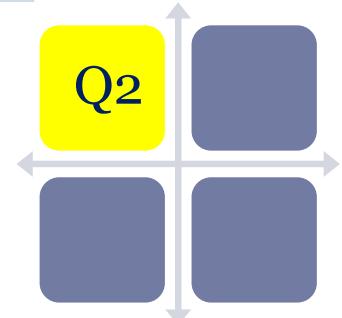
Given I am on etsy.com
When I want to browse through a treasury gallery
!-- We don't care for the results, just the gallery
Then results will be displayed in the gallery

Pasos



- Jbehave brinda un conjunto de anotaciones que nos permiten especificar la implementación de los pasos:
 - `@Given`: determina la implementación de una precondición
 - `@When`: determina la condición a validar
 - `@Then`: determina la implementación de una evaluación

Desarrollo Guiado por Comportamiento (BDD)



Discusión.

- Desarrolladores, testers, expertos de dominio y el dueño de producto se juntan y discuten la historia, gradualmente descomponiendo y destilando el comportamiento en un conjunto de especificaciones simples.

Decisión.

- El dueño de producto decide cuando la especificación cumple suficientemente el comportamiento esperado y está de acuerdo con los casos de ejemplo y demostración de la historia y cierra el alcance de la misma.

Desarrollo.

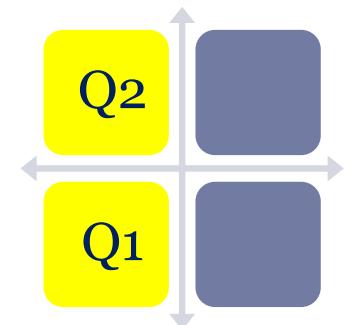
- Los testers refinan los ejemplos de la especificación y los desarrolladores instrumentan las especificaciones creando primero pruebas que fallan y después implementando la funcionalidad.

Demostración.

- Una vez que todos los tests pasan la historia puede ser demostrada al dueño de producto.

Acerca de la automatización

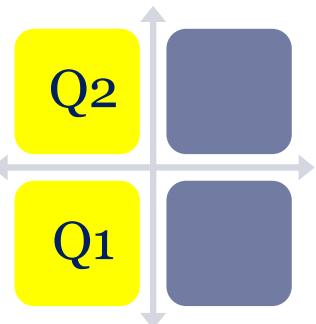
- ¿Por qué automatizar?



Automatizacion

- Las pruebas manuales llevan mucho tiempo
- Las pruebas manuales son propenso a errores
- La automatización permite a las personas focalizarse en otras cosas
- La automatización de las pruebas de regresión proveen seguridad
- Las pruebas automatizadas proveen información rápida y frecuentemente

Acerca de la automatización



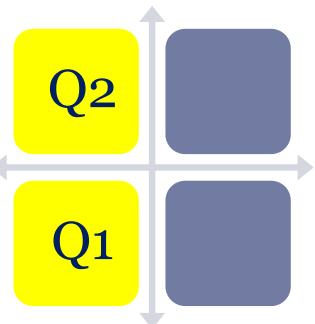
¿Qué automatizar?

- Proceso de Build
- Despliegue
- Tareas REPETITIVAS

¿Qué NO automatizar?

- Pruebas de usabilidad
- Pruebas exploratorias
- Las pruebas que nunca van a fallar
- Cuidado con la automatización de pruebas de interfaz

Acerca de la automatización



- **¿Dificultades?**

- Capacitación
- Inversión
- El código cambia constantemente
- Sistemas heredados
- Resistencia

Behaviour Driven Development (BDD)

- Herramientas
 - Jbehave!!!
 - SpecFlow

Ejemplos...

- Requerimientos para una aplicación de productos al por menor
 - “Como un encargado de deposito, quiero que los artículos devueltos puedan ser re ingresados al stock, para poder conocer la disponibilidad de artículos”.
- BDD
 - Escenario 1: Los artículos devueltos deberían ser re ingresados al stock”.
 - Dado que un cliente quiere devolver un sweter
 - Y actualmente en stock existen 3 sweters
 - Cuando se registra la devolución del sweter
 - Entonces deberá haber 4 sweters en stock.

Ejemplos...

- “Los artículos devueltos o intercambiados deberían ser re ingresados al stock”.
- BDD (desarrollador o QA)
 - Escenario 2: Los artículos intercambiados deberían ser re ingresados al stock”.
 - Dado que un cliente compra un vestido azul
 - Y actualmente en stock existen 2 vestidos azules
 - Y actualmente en stock existen 3 vestidos negros
 - Cuando el cliente intercambie el vestido azul por uno negro
 - Entonces deberá haber 3 vestidos azules y 2 vestidos negros en stock .

Ejemplos...

- Ejemplo1: Las listas nuevas están vacías
 - **Dada** una nueva lista
 - **Entonces** la lista debería estar vacía.
- Ejemplo2: Las listas que contengan cosas no deberían estar vacías.
 - **Dada** una nueva lista
 - **Cuando** se agrega un objeto
 - **Entonces** la lista NO debería estar vacía.
- Ambos ejemplos son requeridos para describir el comportamiento de
 - `list.isEmpty()`

TDD VS. BDD???

Ejemplos...

- **TDD - Unit Test for Creating a Customer**

[TestMethod]

```
public void PostCreateShouldSaveCustomerAndReturnDetailsView  
    with()  
    {  
        var customersController = new CustomersController();  
        var customer = new Customer {  
            Name = "Hugo Reyes",  
            Email = "hreyes@dharmainitiative.com",  
            Phone = "720-123-5477"  
        };  
  
        var result = customersController.Create(customer) as ViewResult  
        ;  
        Assert.IsNotNull(result);  
        Assert.AreEqual("Details", result.ViewName);  
        Assert.IsInstanceOf(result.ViewData.Model, typeof(Customer));  
  
        customer = result.ViewData.Model as Customer;  
        Assert.IsNotNull(customer);  
        Assert.IsTrue(customer.Id > 0);  
    }
```

Creación de test case al principio

LA CLAVE DE LA DIFERENCIA ESTÁ EN EL DISEÑO INICIAL Y LA CREACIÓN DE TEST CASES

- ▶ **BDD - Feature-Level Specification**

Feature: Create a new customer

In order to improve customer service and visibility

As a site administrator

I want to be able to create, view and manage customer records

Scenario: Create a basic customer record

Given I am logged into the site as an administrator

When I click the "Create New Customer" link

And I enter the following information

| | |
|-------------------------------------|--|
| Field Value | |
| Name Hugo Reyes | |
| Email hreyes@dharmainitiative.com | |
| Phone 720-123-5477 | |

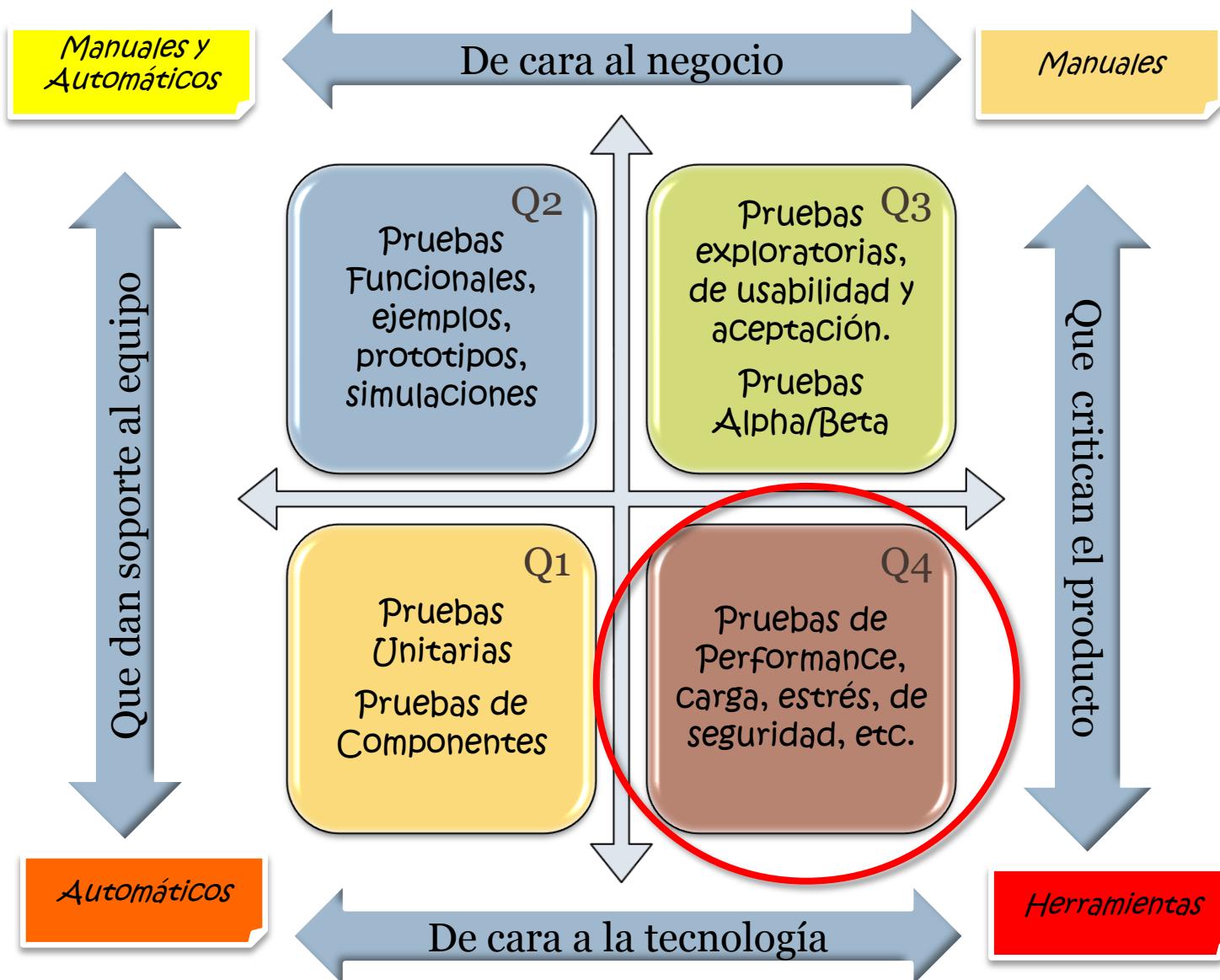
And I click the "Create" button

Then I should see the following details on the screen:

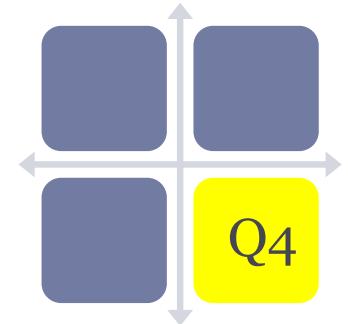
| | |
|-----------------------------|--|
| Value | |
| Hugo Reyes | |
| hreyes@dharmainitiative.com | |
| 720-123-5477 | |

**Se eleva el foco de la
funcionalidad, luego se usa la
descripción en cada unit code**

Los cuadrantes del Testing Ágil



Beneficios del Q4

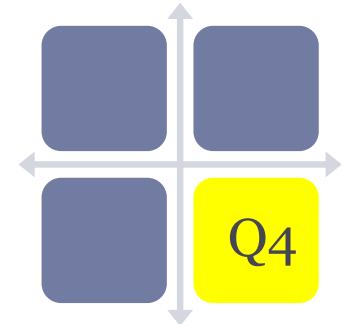


- Does your application deliver the 'right' value???

Requerimientos no
funcionalidad pueden tener
mayor prioridad que los
funcionales

Especialización puede ser
requerida

Test No Funcional



- Prueba los **atributos** de componentes o sistema que no están relacionados a la funcionalidad.
- Probamos “que tan bien anda el sistema”
- Se realiza **en todos los niveles** y puede incluir pruebas de: performance, carga, stress, etc.

Atributos de Calidad del Software

- El estándar **ISO 9126** identifica 6 atributos clave de calidad del Software:
 - **Funcionalidad:** La capacidad del producto de software de proveer funciones cuando el sistema es usado bajo condiciones específicas. Incluye:
 - Corrección
 - Interoperabilidad
 - Seguridad
 - **Confiabilidad:** Cantidad de tiempo que el software está disponible para su uso. Incluye:
 - Tolerancia a fallos
 - Tiempo de recuperación

Atributos de Calidad del Software

- **Usabilidad:** La capacidad del sistema de ser entendido, aprendido, usado y atractivo al usuario cuando se lo usa bajo condiciones específicas
- **Eficiencia:** La capacidad del sistema de proveer la performance apropiada con respecto a la cantidad de recursos usados bajo ciertas condiciones.
- **Mantenibilidad:** Facilidad con que una modificación puede ser realizada. Incluye:
 - Facilidad de análisis, de introducir un cambio
 - Estabilidad
 - Facilidad de prueba
- **Portabilidad:** La facilidad con que el software puede ser llevado de un entorno a otro

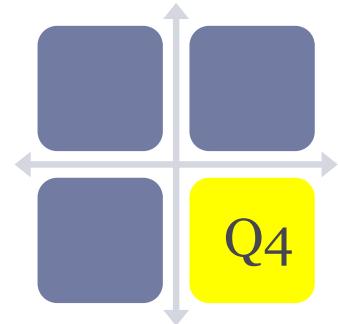
Atributos de Calidad del Software

- **Performance**: Determino el rendimiento del software en cuanto al uso de los recursos
- **Carga**: Mido el comportamiento del sistema al aumentar la Carga
- **Stress**: Evalúo un sistema o componente llevándolo a los límites especificados en sus requerimientos



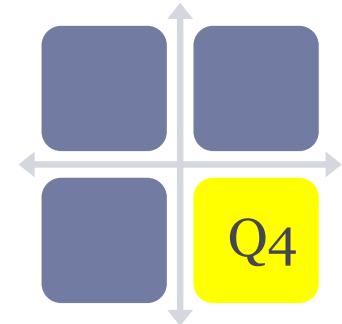
¿Cómo hago!?

Cuarto Cuadrante (Q4)



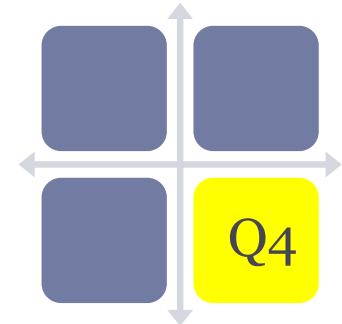
- Seguridad
 - Dos enfoques
 - De afuera para dentro (hackers). Imitar el comportamiento.
 - De adentro hacia afuera
 - Herramientas de análisis estático de código.

Cuarto Cuadrante (Q4)



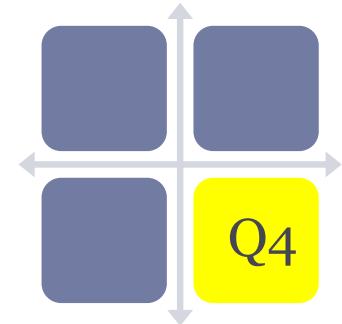
- Mantenibilidad
 - No es sencillo probar la mantenibilidad.
 - Revisiones e inspecciones de código.
 - Programación de a pares.
 - Utilización de guías y estándares (convención de nombres).

Cuarto Cuadrante (Q4)



- Interoperabilidad
 - Diversos sistemas y organizaciones trabajando conjuntamente y compartiendo información.
 - Pruebas end to end entre dos o más sistemas.

Cuarto Cuadrante (Q4)



- Compatibilidad
 - En aplicaciones web: Múltiples navegadores, sistemas operativos.
- Confiabilidad
 - Tiempo entre fallas
- Instalaciones

Ya casi llegamos...?



Integración Contínua

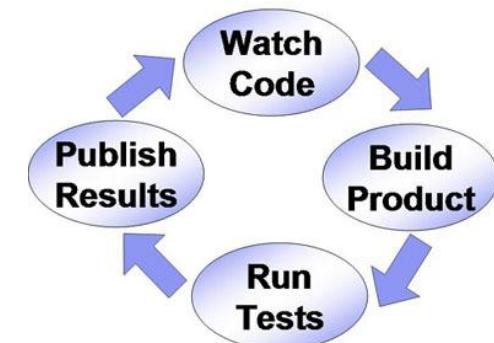
La Integración Continua (Continuous Integration - CI) es una práctica de desarrollo de SW donde los miembros de un equipo integran su trabajo frecuentemente, lo que lleva a múltiples integraciones por día.

Objetivo Principal

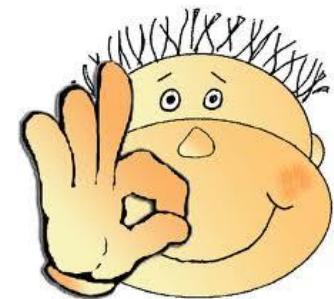
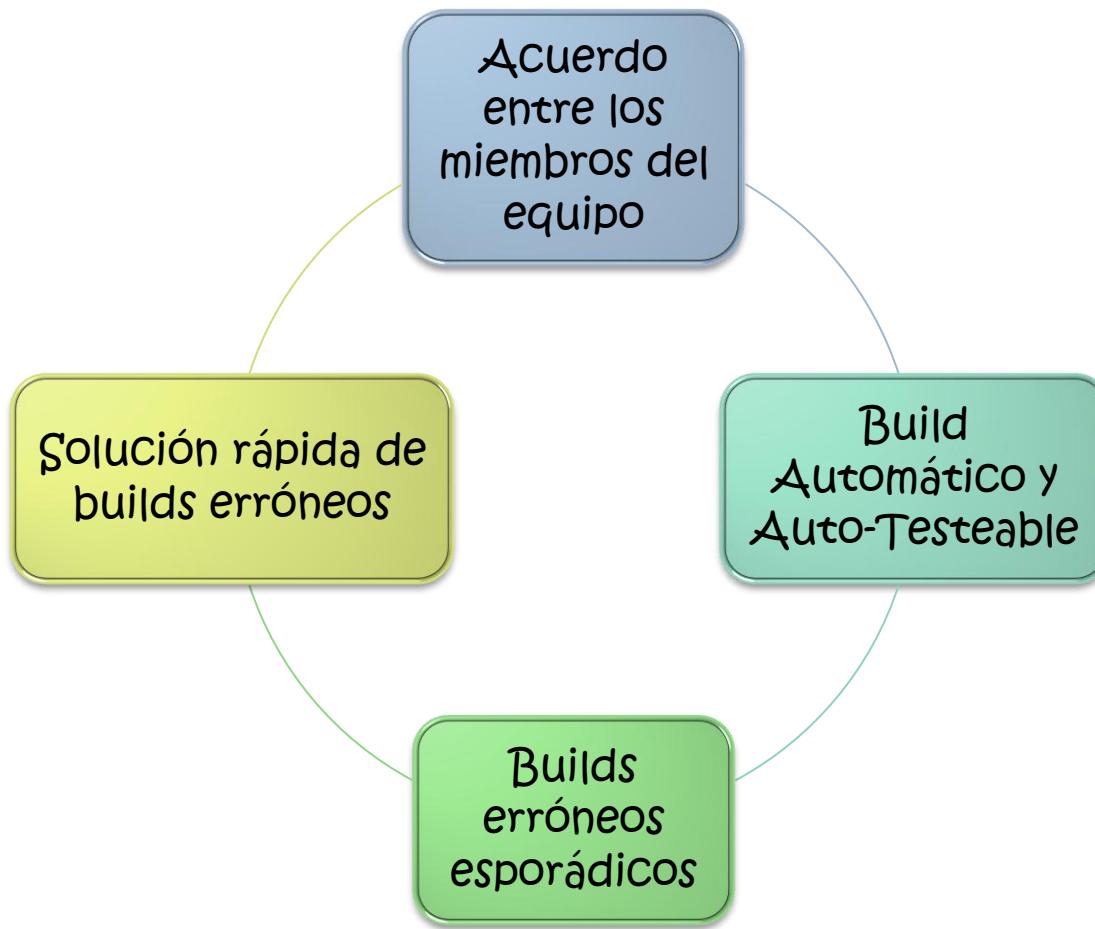
- Detectar errores de integración lo más rápido posible.
- Estar listos para entregar SW que funcione

Principal Problema

- Demoras entre que se “termina” el desarrollo y la aplicación puede liberarse



Factores Críticos de Éxito



Prácticas

Mantener un único repositorio de fuentes

- *Source Control*

Hacer el Build Auto-Testeable

- *SW que funciona y SW que funciona correctamente*

Automatizar la construcción del Build

- *Simplificar y agilizar*
- *Ambiente separado similar al de producción*

Commits diarios

- *Comunicación entre los desarrolladores*
- *Los miembros del equipo hacen commits todos los días*

Commit => Build

- *Cada commit debería involucrar un build sobre la rama correspondiente en una PC dedicada para fines de integración*

Prácticas

Builds rápidos

- *No debería tomar más de 10 minutos*
- *Estrategia 1: 2 builds (Integración en etapas – Síncrona y asíncrona)*
- *Estrategia 2: Mejorar tests*
- *Estrategia 3: Grupos de acuerdo al diseño*

Ambiente separado igual al de producción

- *Probar en un ambiente clonado del ambiente de producción*
- *Usar VMs*

Obtención de ejecutables

- *Cualquiera debe poder obtener el último ejecutable de la aplicación y ayuda a validarla*

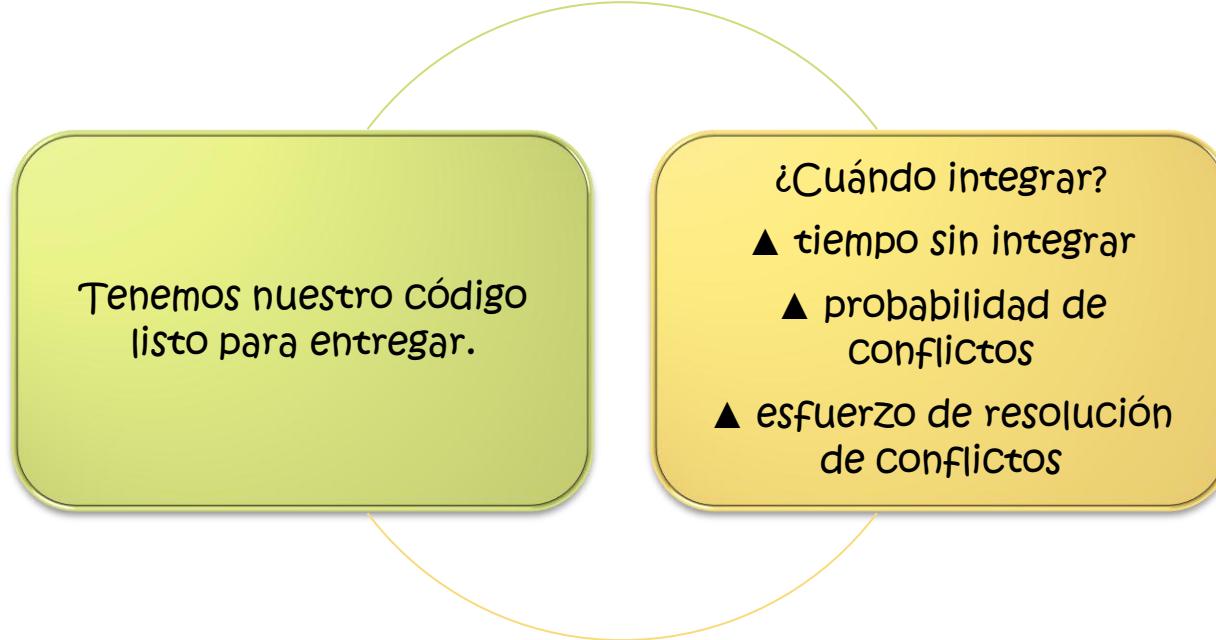
Cualquiera puede ver que está pasando

- *Buscar formas fáciles de comunicar el estado de los builds*

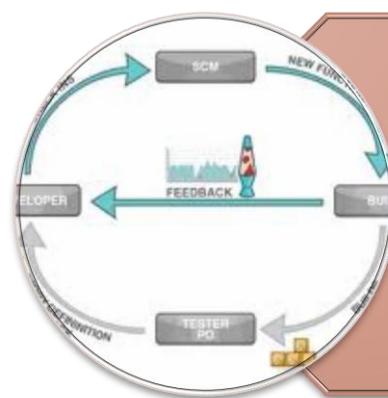
Automatizar el despliegue

- *Rápido despliegue en diferentes ambientes*

Frecuencia de Integración

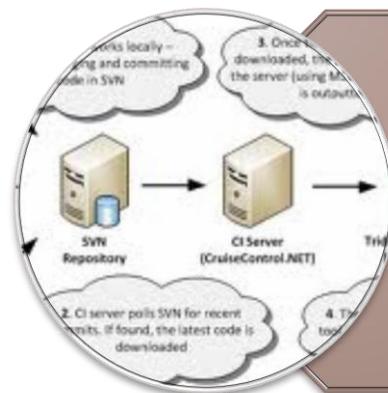


Builds Manuales y Servidores de Integración Continua



Manual o Síncrona

- Check Out
- Build
- Chequear resultados
- Check In
- Check Out en la PC de Integración
- Build
- Chequear resultados



Server o Asíncrona

- Check Out Automático
- Iniciar el build
- Avisar al desarrollador
- Si hay errores, se deben resolver
- Si no hay errores, se considerada terminada la integración

Beneficios

-  *Reduce el riesgo de fallas en la integración*
-  *Integraciones frecuentes*
-  *Conocimiento del estado de la aplicación*
-  *Menos bugs*
-  *Los bugs se encuentran y corigen más rápido*



Diferencias

Integración Continua

- Builds frecuentes que pasan los tests
- Problema: NO tener una PC de integración

Entrega Continua

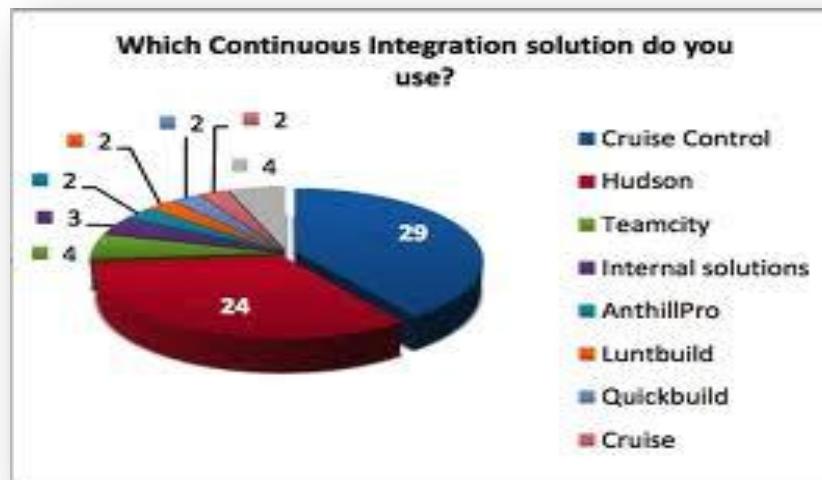
- Se hacen entregas al ambiente de testing
- Clave: Automatizar el despliegue

Despliegues Continuos

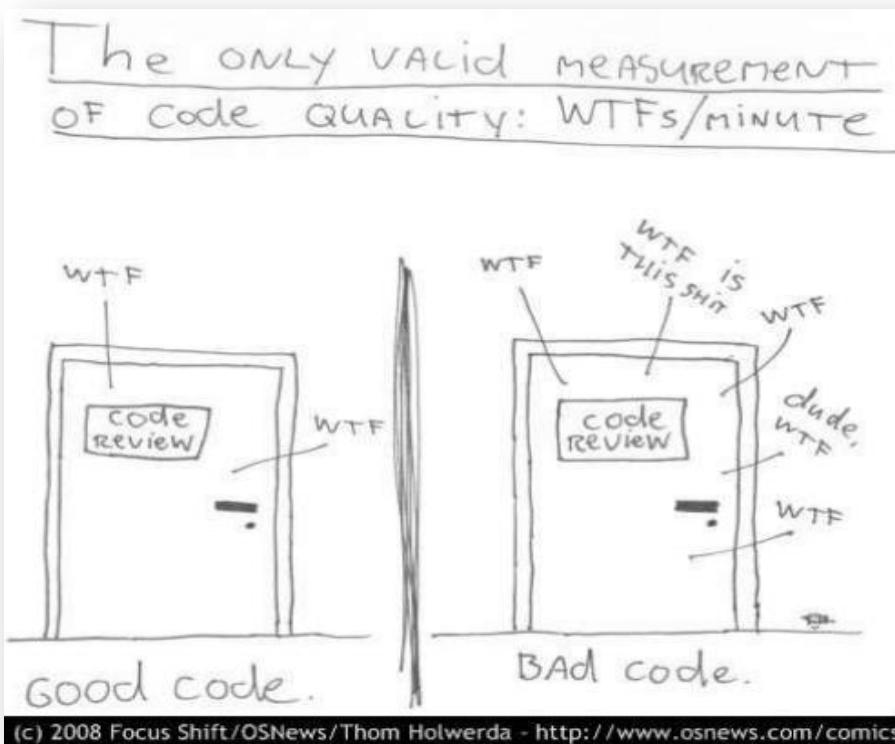
- La aplicación está lista para entregarse en el ambiente de producción
- Más riesgo
- ▲ TESTS ▼ RIESGO

Métricas

- Tests Unitarios
 - Xunit Technologies
- Servidores de integración continua



Métricas



Herramientas

- RTC – Rational Team Concert
- CruiseControl
- CruiseControl.NET
- CruiseControl.rb
- Cruise
- CI Factory
- Drumbeat CI
- Tinderbox & Tinderbox2
- BuildBot
- Anthill Professional
- Anthill
- Bamboo
- Luntbuild professional
- LuntBuild
- Gump
- Continuum
- Sin
- OpenMake Meister
- OpenMake Mojo
- Parabuild
- Tinderbox3
- Pulse
- TeamCity (EAP)
- Hudson
- FinalBuilder Server
- Zed
- easyCIS
- RedJack
- ElectricCommander

<http://confluence.public.thoughtworks.org/display/CC/CI+Feature+Matrix>



Y después que
integro...qué hago?

Test de confirmación

Ocurre cuando una nueva versión del sistema incluye corrección de defectos

Se necesita ejecutar una prueba para confirmar que los defectos realmente han sido eliminados

Es muy importante asegurarse que el sistema ha sido ejecutado exactamente de la misma manera que cuando el defecto fue encontrado

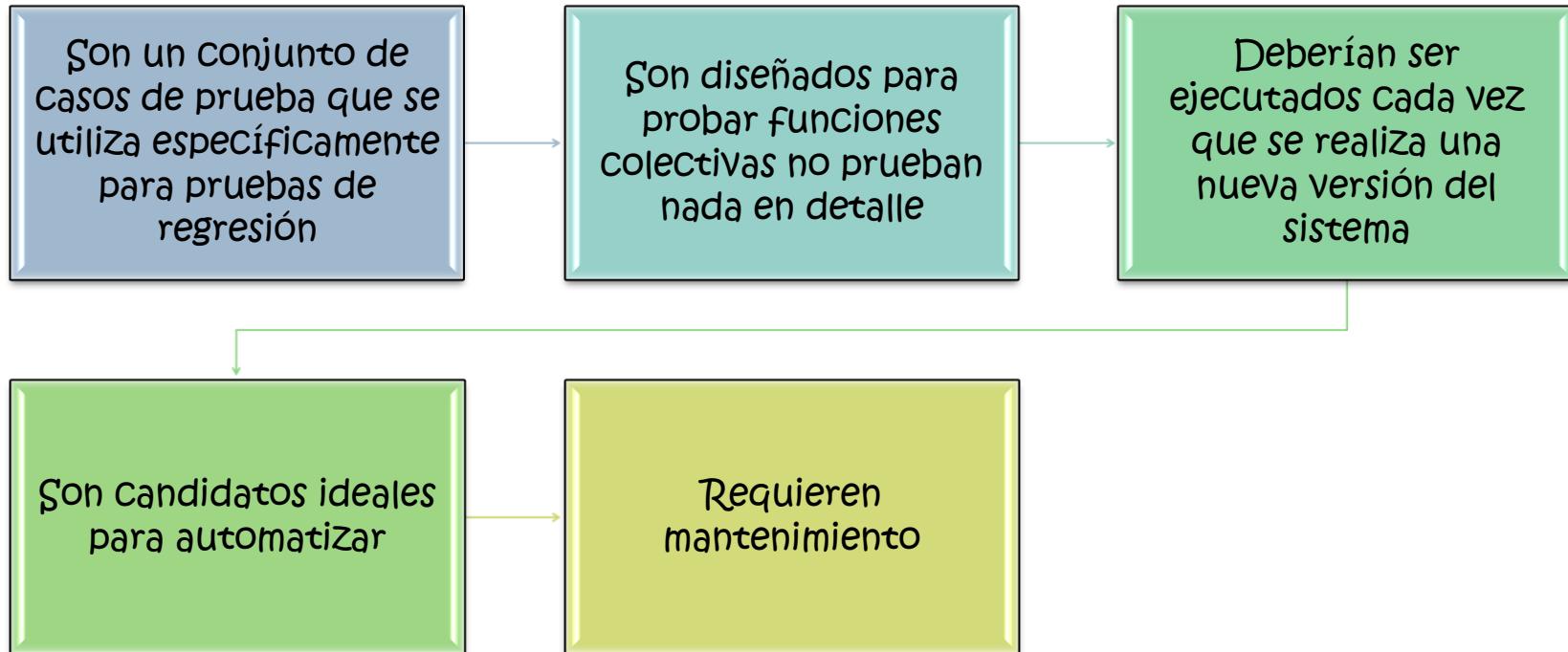
Test de regresión

Pruebas que se realizan para asegurarse que no se introdujeron nuevos defectos en áreas no modificadas con las últimas modificaciones realizadas

Se llevan a cabo cuando el sistema o el ambiente es modificado

Las pruebas de regresión involucran casos de prueba que han sido ejecutados antes

Suites de pruebas de regresión



Antes de continuar....

...Intentemos algo



Ejercicio en clase...

- En grupo generar un release plan basándose en el backlog del equipo contrario...

| Team | Product backlog actual | Nuevo Product Backlog |
|------------------|------------------------|-----------------------|
| chewbacca-agiles | Action tracker | Release Burndown |
| Sonic Boom | Release Burndown | Action tracker |
| Maestro Spr | Sprint burndown | timesheet |
| Jobniak | Action tracker | Sprint burndown |
| Los Quatro | Timesheet | Release burndown |



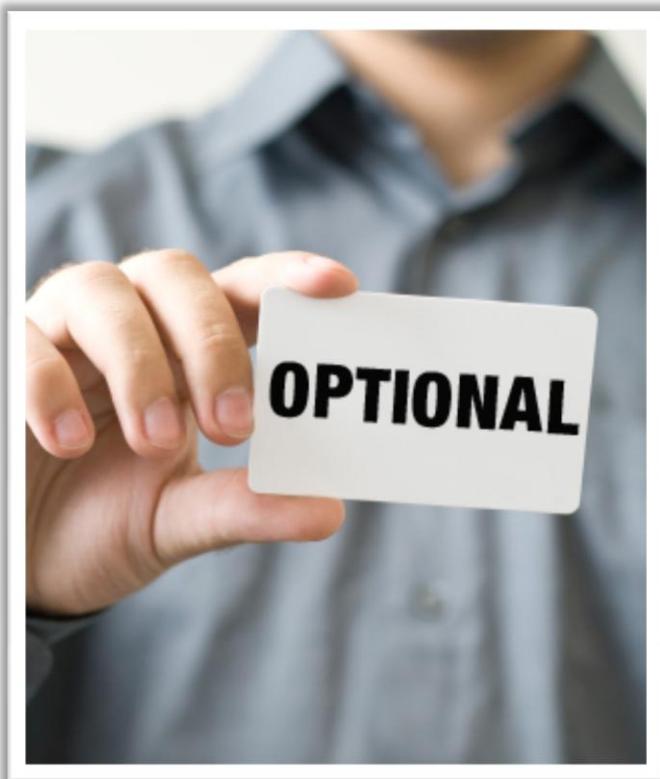
- Esto es todo????

Entrega del producto



¿Cada vez que se termina una iteración es obligatorio entregar el producto?

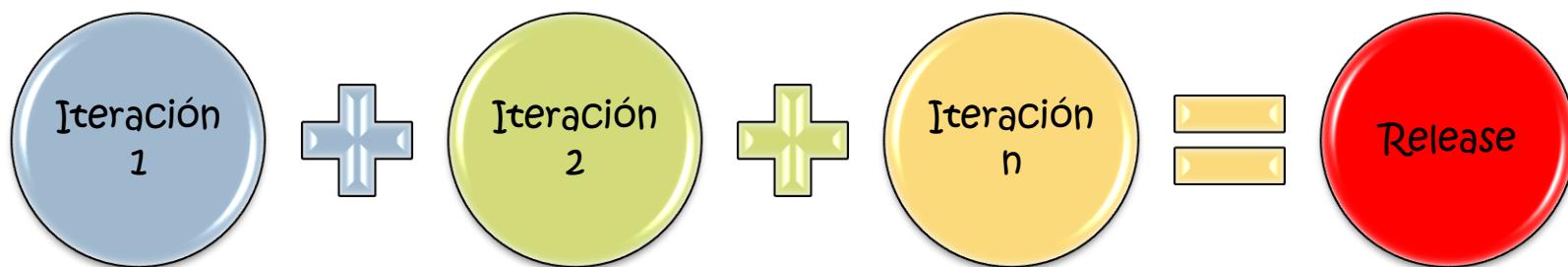
Entrega del producto



- En cada iteración el equipo transforma los requerimientos en software **potencialmente entregable**
- ↑
- Esto significa que el producto tiene la calidad suficiente para ser entregado al cliente

Entrega del producto

- Release?



- Ninguna funcionalidad nueva puede desarrollarse en una sola iteración...

Retrospectivas?

¿Cómo?

- Periódicamente se “mira” a lo que está y lo que no está funcionando
- Típicamente toma 15–30 minutos
- Después de cada iteración
- Todo el equipo participa
 - ScrumMaster
 - Dueño del Producto
 - Equipo
 - Clientes y otros..

Retrospectivas

¿Por qué fallan?

Falta de preparación

Falta de foco

Error en la recolección de datos

Una o dos personas dominando la conversación

Enfoque sólo en impedimentos que están fuera del alcance del equipo

Elegir planes de acciones para los que el equipo no tiene esfuerzo

Mantener un “plan de mejora” por separado.

Retrospectivas

¿Cómo las
hago
fructíferas?

- Revisión de los resultados de retrospectivas anteriores contra las mejoras de la iteración actual
- Ganar el “patrocinio” de alguien que pueda tomar acciones
- Hacerse el tiempo para discusiones cara a cara
- Estar dispuesto a realizar ajustes
- Cada participante debe prepararse correctamente

Retrospectivas - Técnicas

Diagrama Ishikawa

Pasos

- 1) Ponerse de acuerdo sobre el problema
- 2) Identificar las categorías mayores de causas
- 3) Escribir las categorías como “espinas” del pescado
- 4) Preguntar para cada una de las causas “¿Porqué pasa esto?”
- 5) Volver a preguntar lo mismo para cada causa identificada
- 6) Cuando el grupo no tenga más ideas, enfocarse en las “espinas” en donde haya menor cantidad de Causas

Es una forma de organizar y representar las diferentes teorías propuestas sobre las causas de un problema

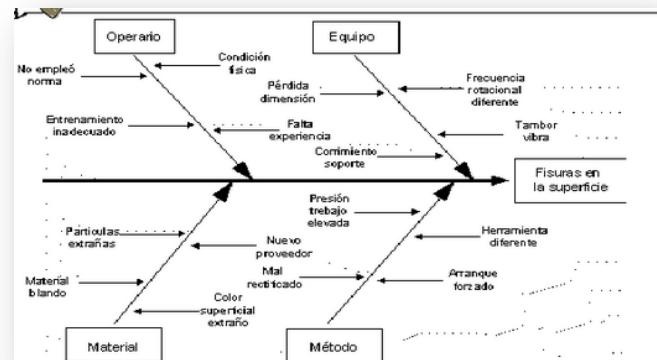
Identifica múltiples causas para un problema.

Retrospectivas - Técnicas

Diagrama Ishikawa

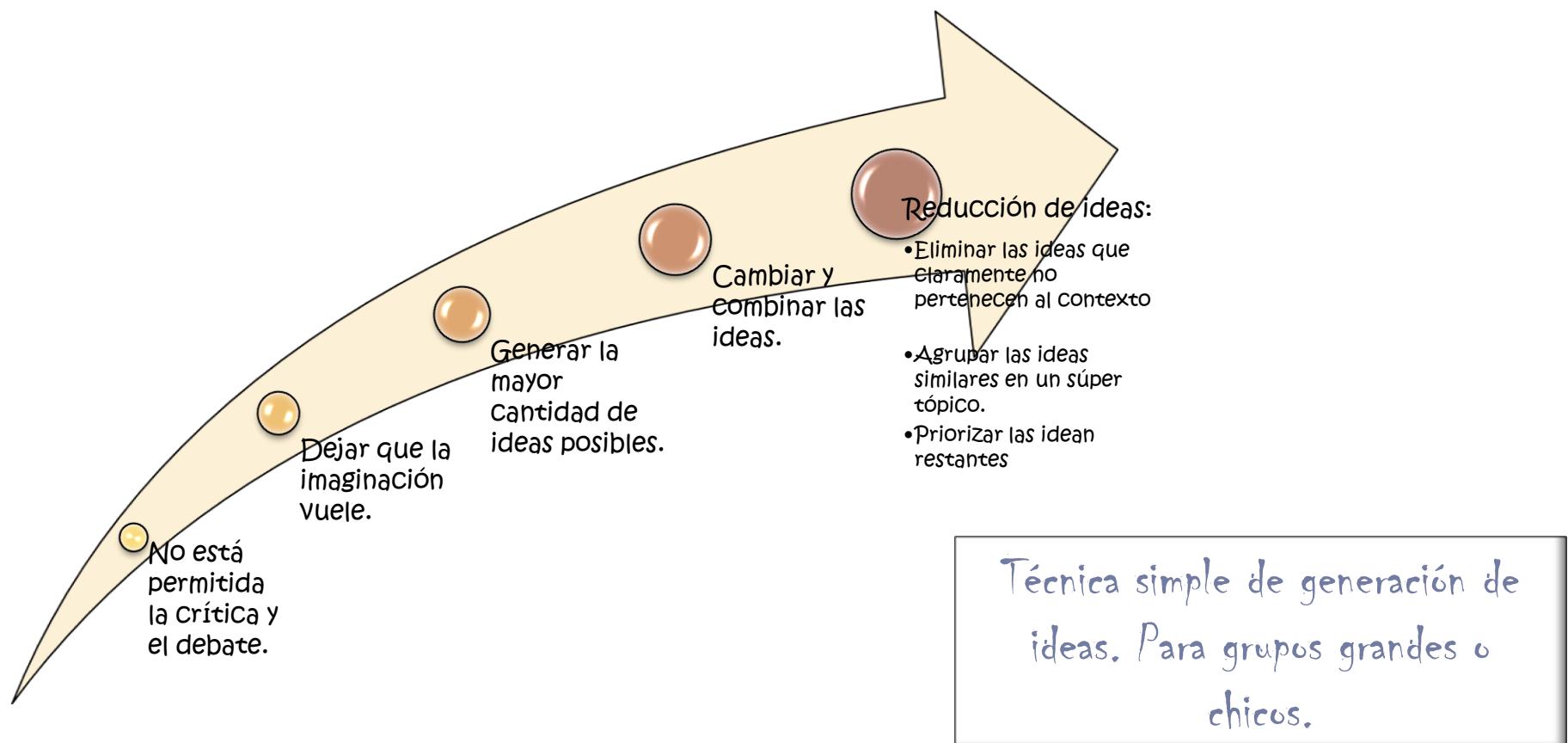
Desventajas

- Sólo cuando estas teorías son contrastadas con datos podemos probar las causas de los fenómenos observables
- Construir el diagrama antes de analizar globalmente los síntomas
- Limitar teorías propuestas enmascarando involuntariamente la causa raíz.
- Errores tanto en la relación causal como en el orden de las teorías (involucra in gasto de tiempo importante)
- No tienen una base estadística



Retrospectivas - Técnicas

Tormenta de Ideas



Retrospectivas - Técnicas

Tormenta de Ideas

Ventajas:

- No se necesitan expertos para participar.
- Es una técnica fácil de utilizar.
- No es costosa.
- Si se controla correctamente, es una manera rápida de generar ideas.
- Promueve la creatividad

Desventajas:

- Pueden llevar mucho tiempo si el grupo no es controlado apropiadamente
- Puede generar falsas expectativas a los participantes ya que muchas de las ideas nunca serán implementadas



Retrospectivas ¿vale la pena?

¿Se puede cancelar una iteración?



OBsoleto

Si el objetivo de la iteración
se vuelve obsoleta

Sii! ¿cuándo?!

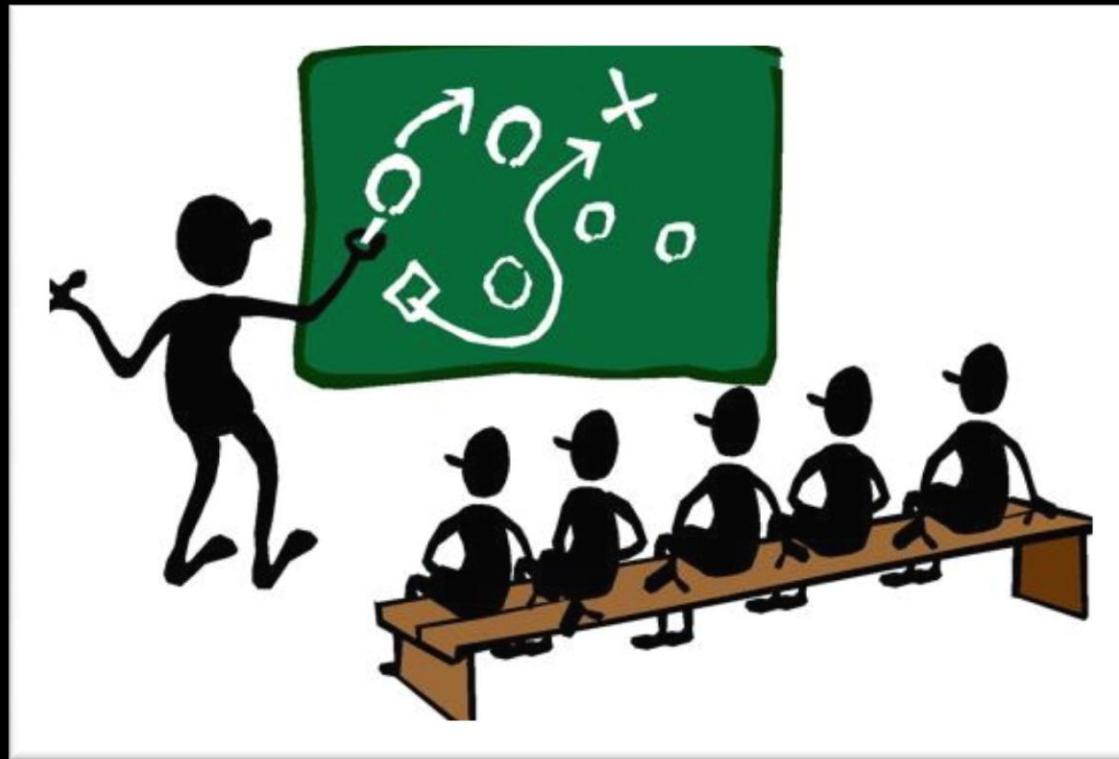
Se revisan todos los PBIs completos

Si parte del trabajo es potencialmente “entregable” el PO lo acepta.

Todos los PBIs incompletos son reestimados y vueltos a la pila del producto.

Sólo el Dueño del producto tiene la autoridad para cancelarlo!!!!

¿Qué más podemos aprender sobre metodologías ágiles...?

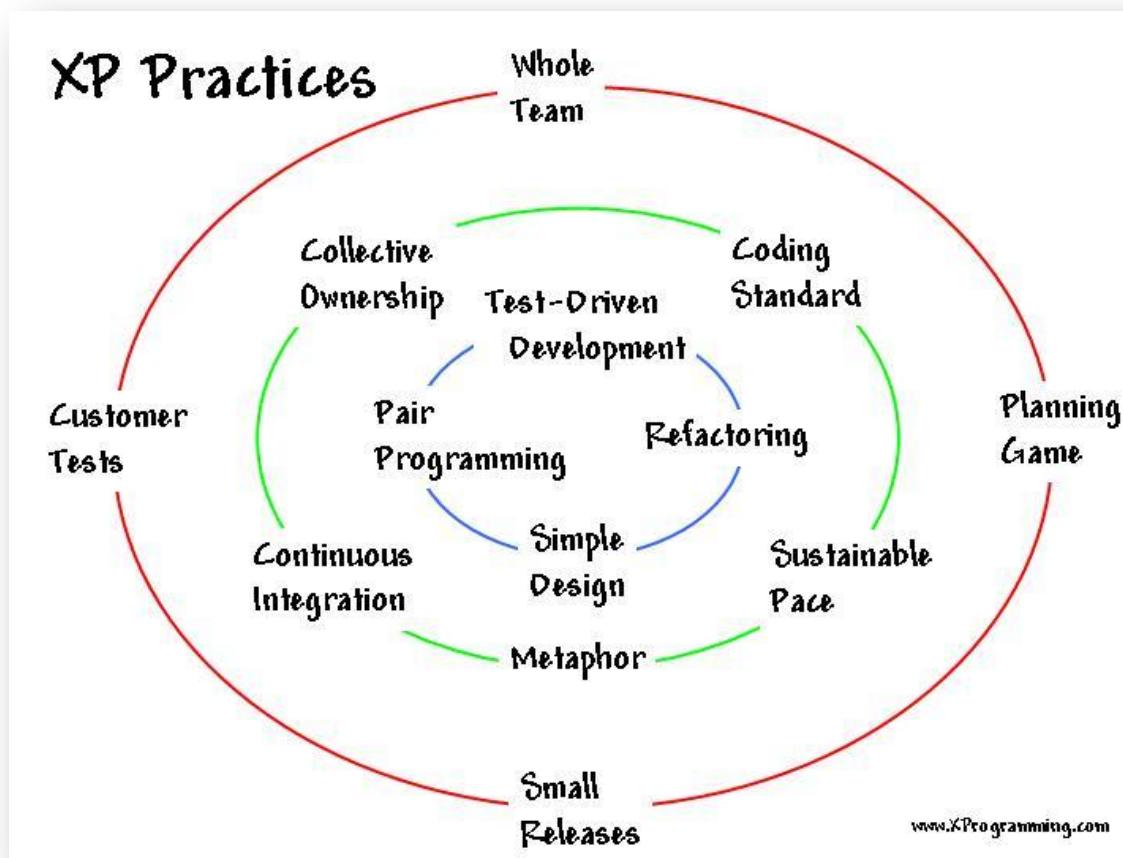


Extreme Programming

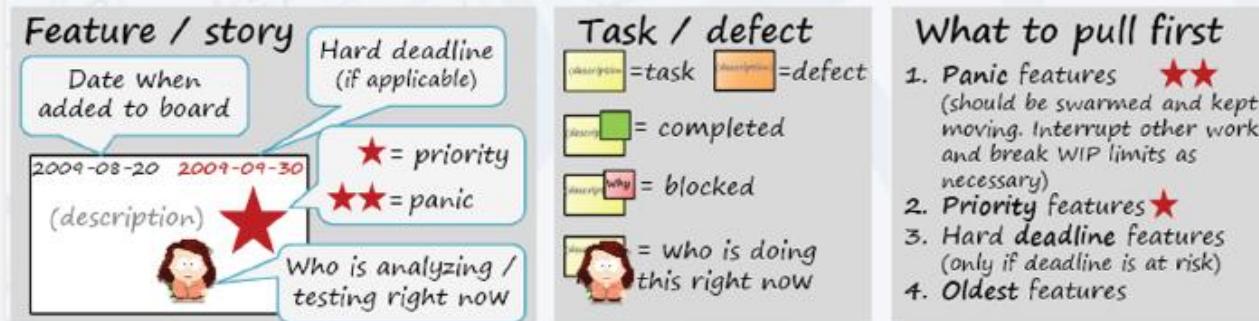
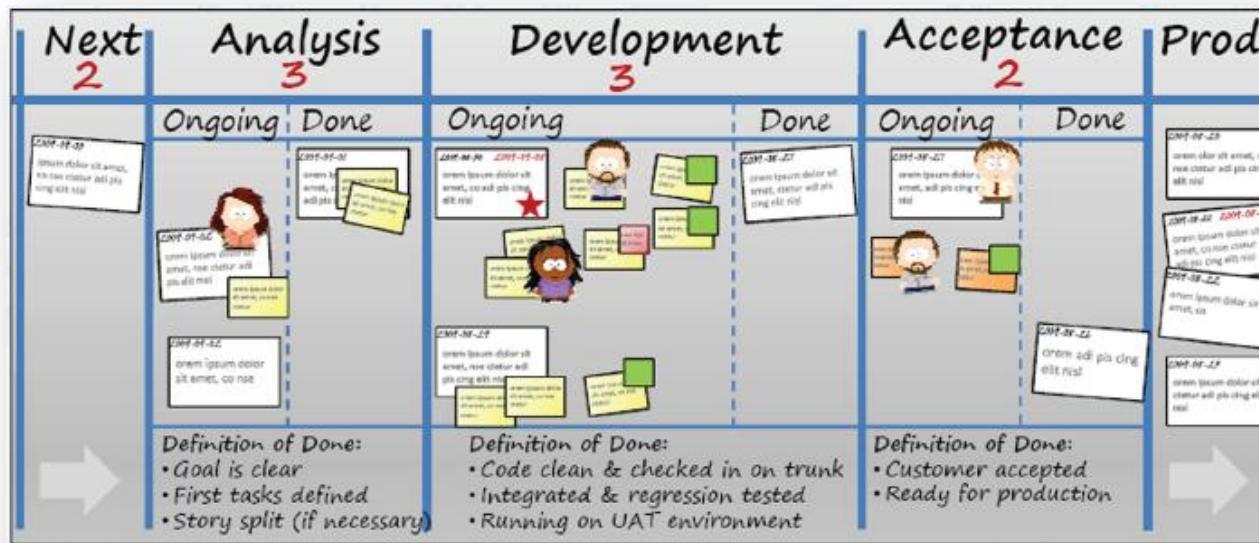
- Es la única metodología que se enfoca principalmente en la parte programática del desarrollo del software.
- Los roles de “administradores” son llevados a cabo también por programadores → fortaleza
- XP no puede ser implementado sin testing.



Extreme Programming

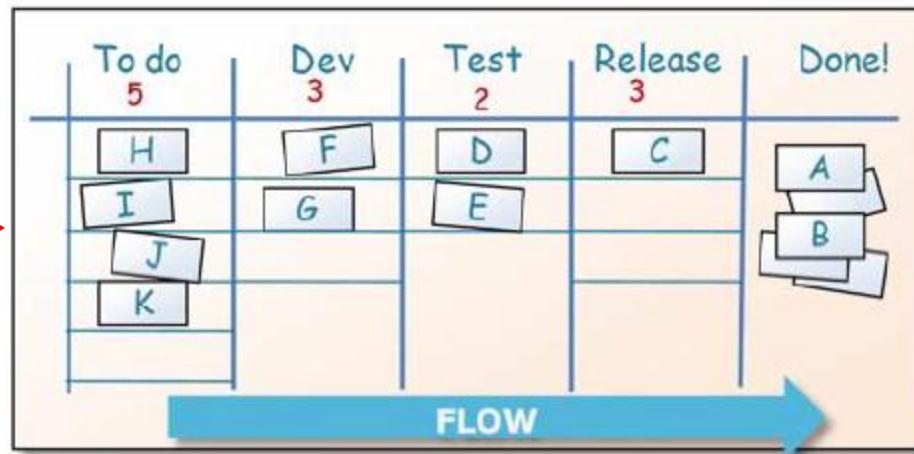


Kanban



Kanban

- Visualizar el “workflow”



- Limitar el “trabajo en progreso” (WIP)

- Medir el tiempo de entrega (lead time)

Comparando...

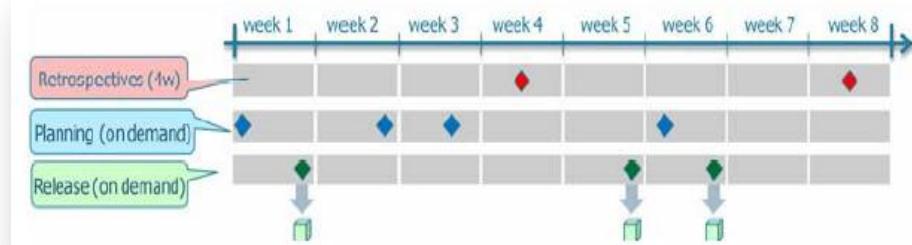
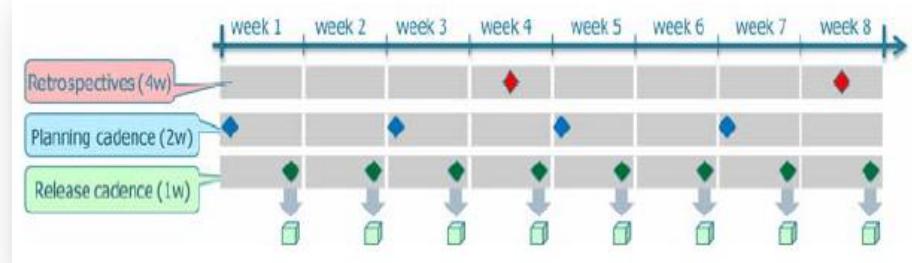
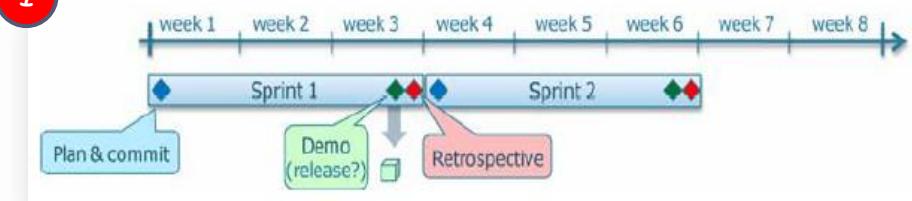
SCRUM

- 1) Iteraciones con tiempo fijo

Kanban

- 1) Se puede elegir cuando hacer planificación, mejora de proceso y release.

1



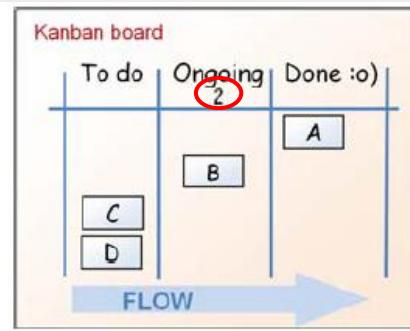
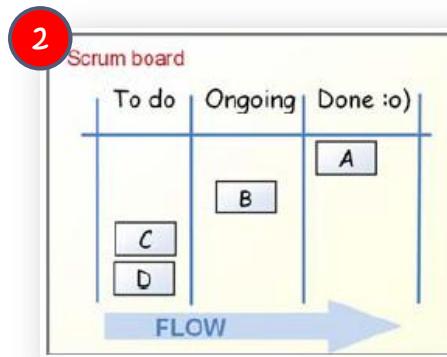
Comparando...

SCRUM

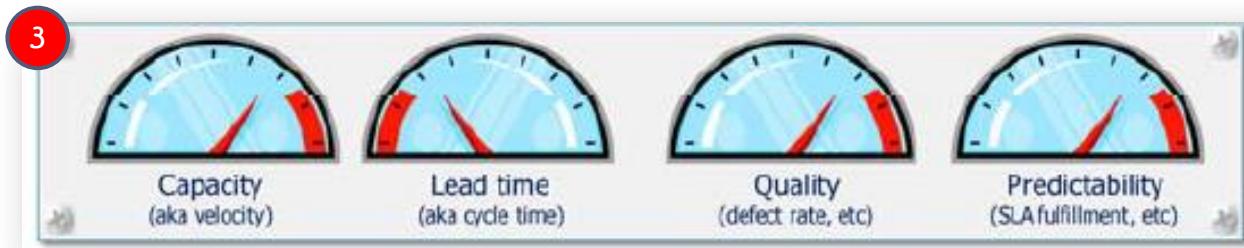
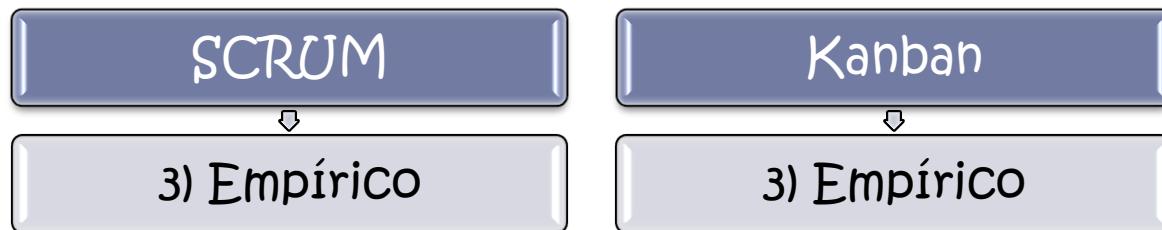
Kanban

2) Limita los WIP por iteración

2) Limita los WIP por estado del workflow



Comparando...



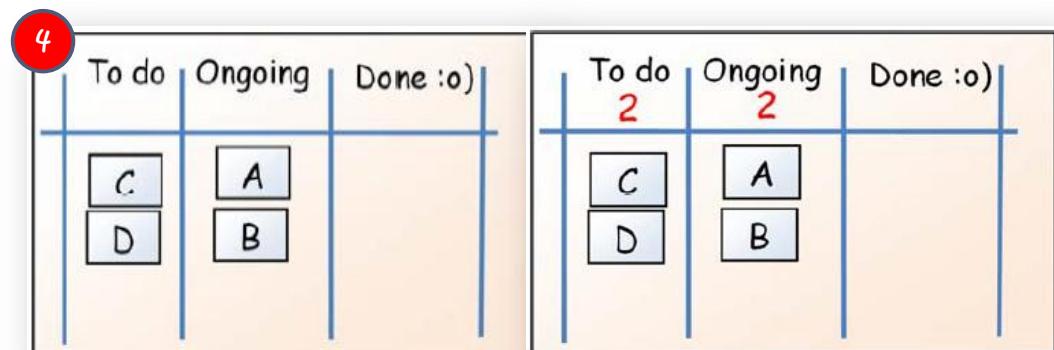
Comparando...

SCRUM

Kanban

4) Resiste al cambio dentro de la iteración

4) Dependiendo de la capacidad (Límite dado por la Cantidad de WIP)



Comparando...

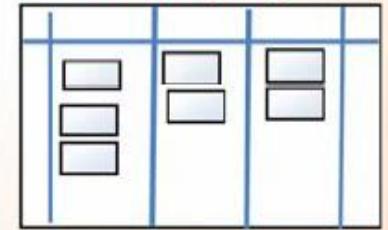
SCRUM

5) La “pizarra” es
reseteadas entre
iteraciones

Kanban

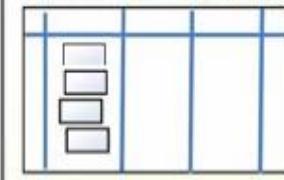
5) La “pizarra” es algo
persistente...

Kanban: Any day

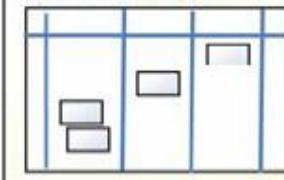


5

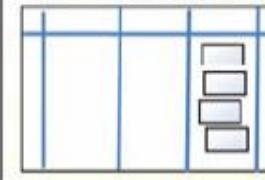
Scrum: First day of sprint



Scrum: Mid-sprint



Scrum: Last day of sprint



Comparando...

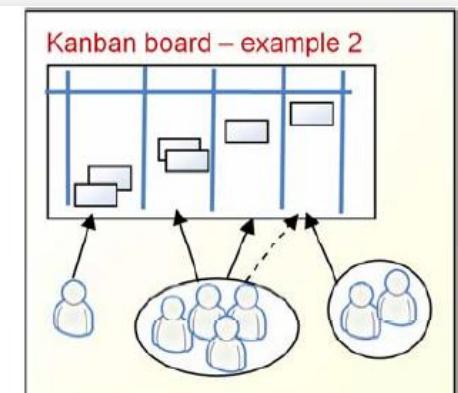
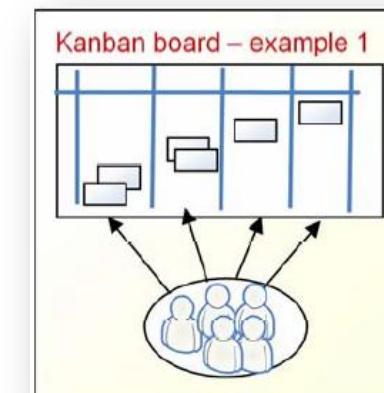
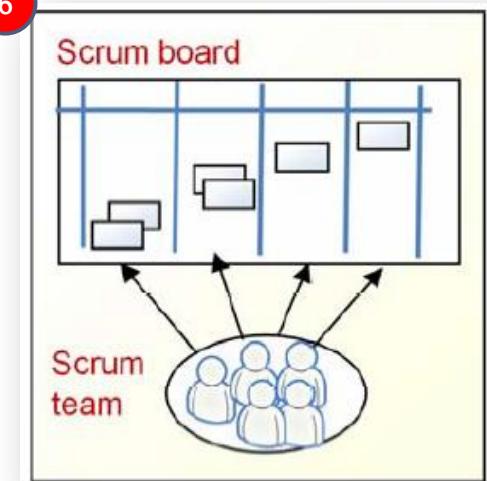
SCRUM

6) Prescribe equipos multifuncionales

Kanban

6) Equipos multifuncionales son opcionales.

6



¿Lean?

- Lean: “*with little or no fat*”
 - [American Heritage® Dictionary of the English Language]

→ ¿y en castellano?

- No es una metodología como las descriptas anteriormente.

- No propone una serie de prácticas a seguir.

- Es un “toolkit” de principios (qué, no cómo)

Principios del desarrollo Lean

Evitar despilfarro (*eliminate waste*)

- Componentes, documentos, funcionalidades innecesarias

Amplificar el aprendizaje (*amplify learning*)

- Proceso de desarrollo vs. proceso de producción

Decidir lo más tarde posible (*decide as late as possible*)

Entregar lo antes posible (*deliver as fast as possible*)

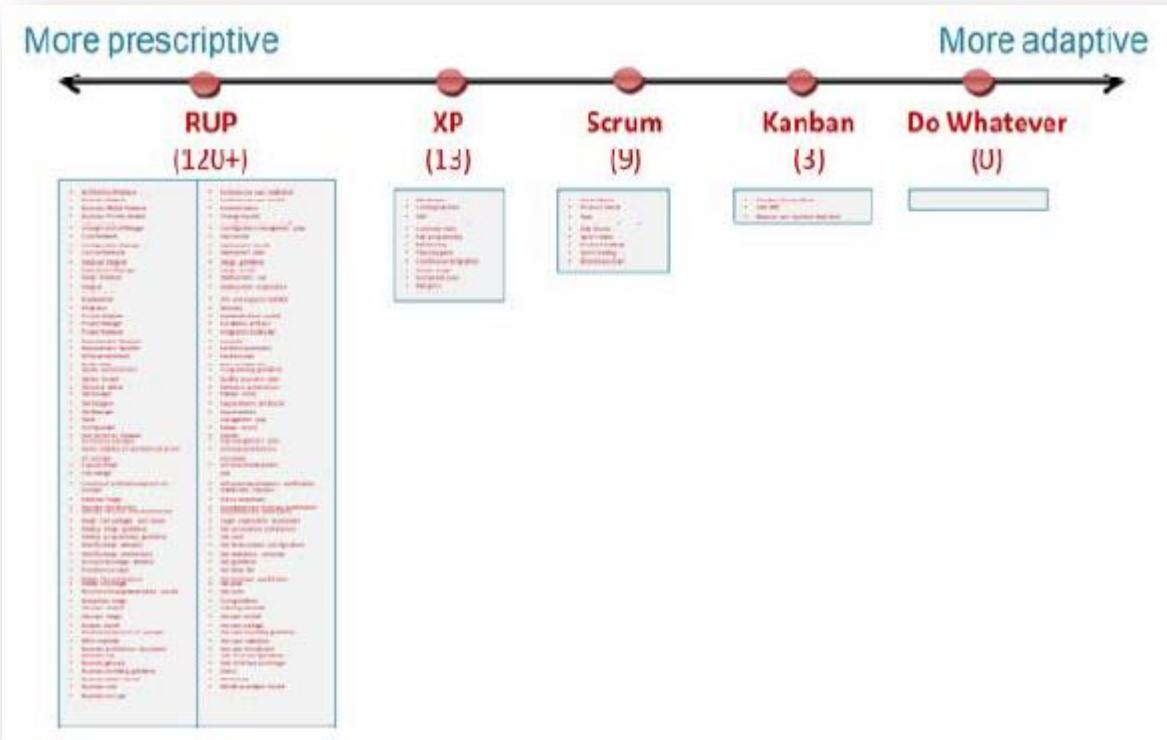
Dar poder al equipo (*empower the team*)

Integridad (*build in integrity*)

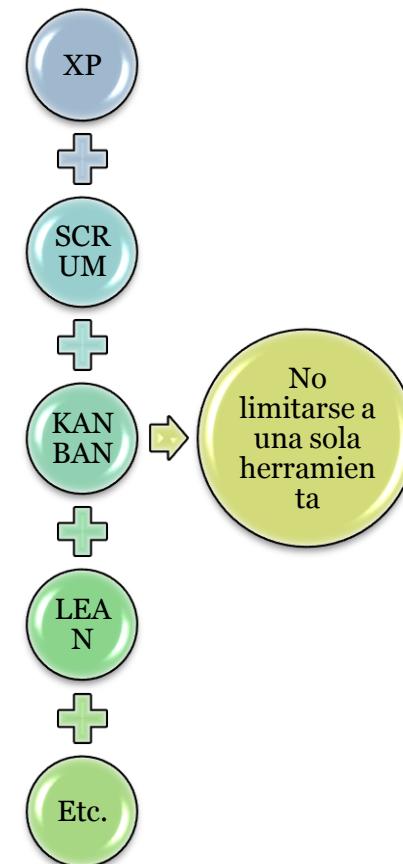
- El sistema funciona correctamente, es útil a través del tiempo, puede ser extendido y mantenido fácilmente.
- Se da por → Liderazgo, conocimiento del dominio, comunicación y disciplina

Tener una visión global (*see the whole*)

Comparando...



Lo importante???



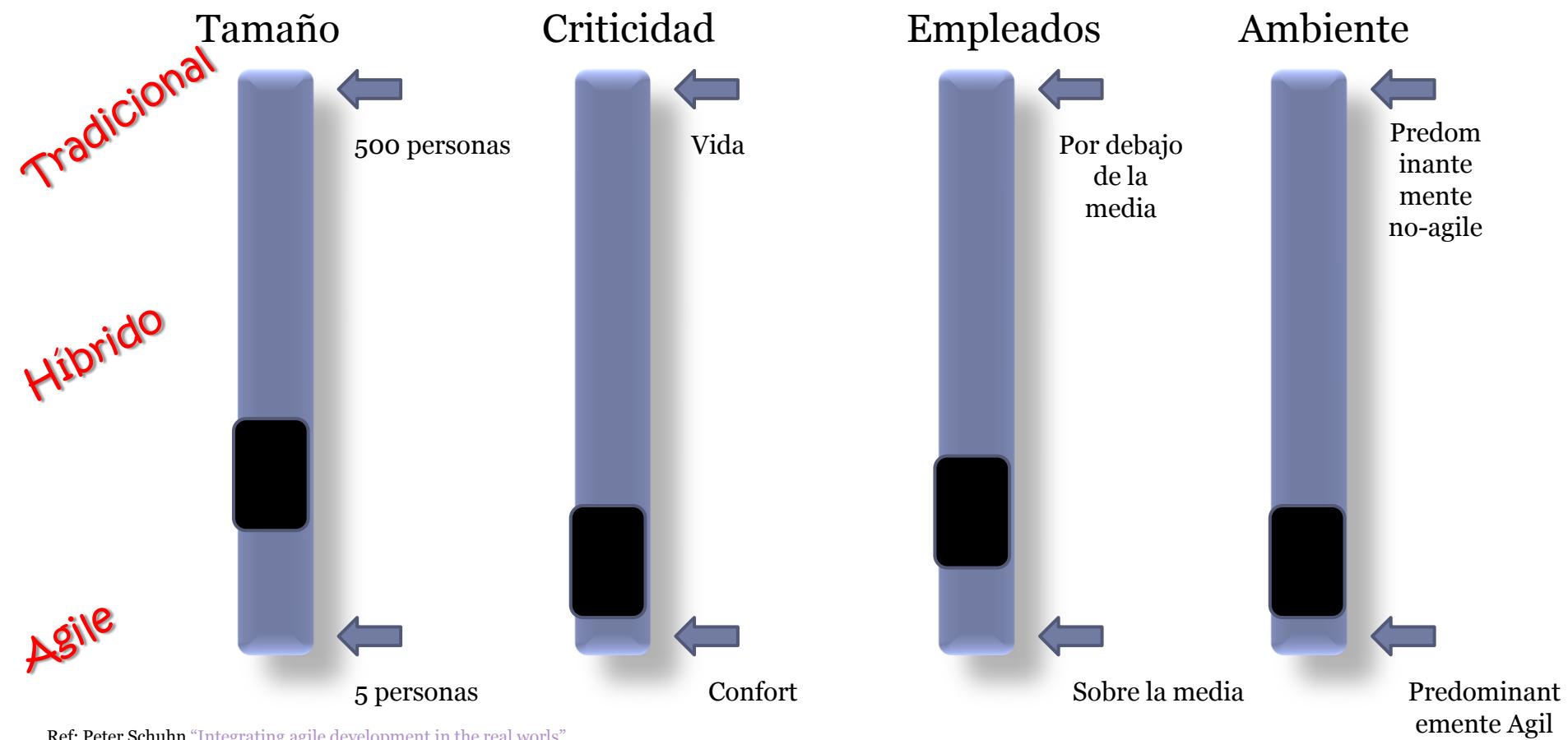
¿Cómo sabemos qué necesitamos?

Proceso tradicional

PROCESO ÁGIL?

Ambos?

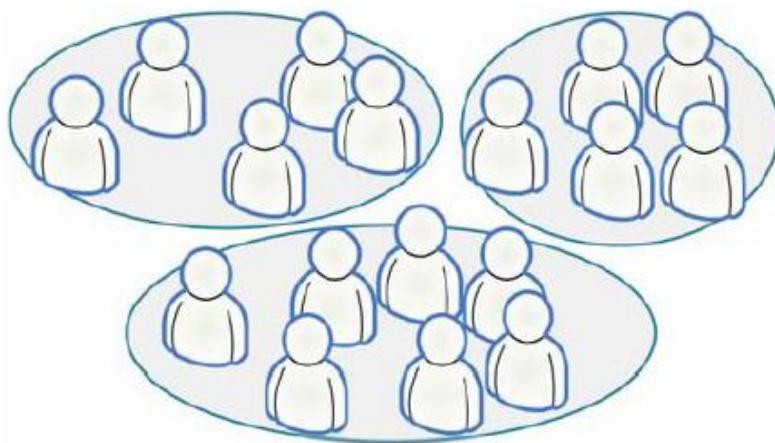
¿Cómo sabemos qué necesitamos?



Resumiendo....

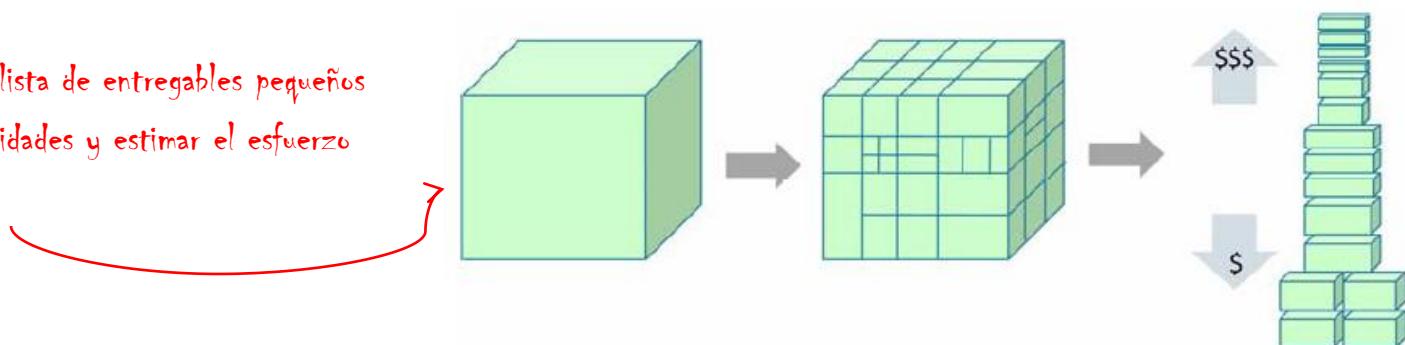


SCRUM in a nutshell

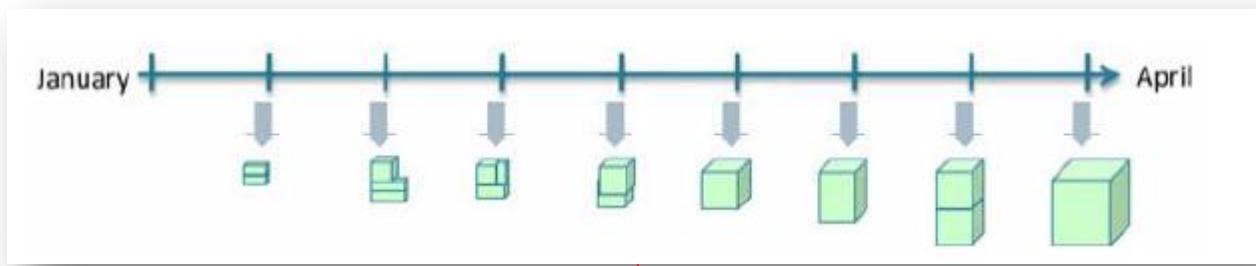


- Dividir la organización en equipos pequeños y auto organizados

- Dividir el trabajo es una lista de entregables pequeños
- Ordenar la lista por prioridades y estimar el esfuerzo para cada uno de ellos.



SCRUM in a nutshell



- Dividir el tiempo en iteraciones cortas de tiempo específico (1-4 semanas), con código potencialmente entregable.
- Optimizar el plan de release y actualizar las prioridad (en colaboración con el cliente), basándose en revisiones del release después de cada iteración
- Optimizar el proceso a través de retrospectivas después de cada iteración

SCRUM in a nutshell

entonces, en vez de tener un solo grupo grande invirtiendo mucho tiempo en la construcción de algo grande...

...Tenemos un equipo pequeño invirtiendo poco tiempo en algo pequeño. Pero con Integraciones regulares para “ver” el todo

“Tips” para planificar

Involucrar a todo el equipo

Planear a diferentes niveles

Mantener las estimaciones de tamaño y duración separadas utilizando diferentes unidades

Expresar la incertidumbre en la funcionalidad o en la fecha.

Re planificar frecuentemente

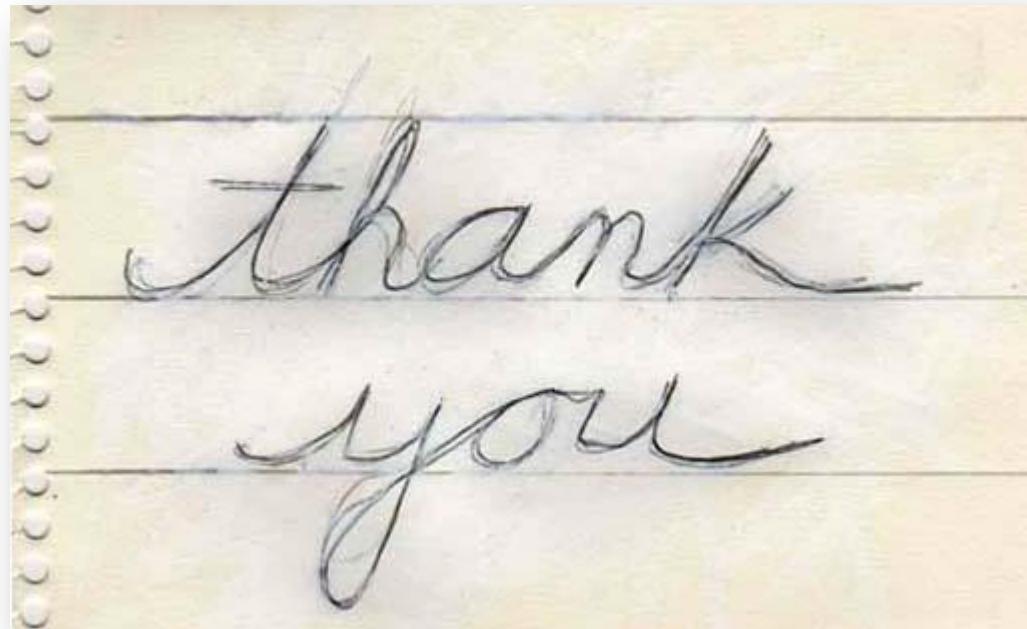
Planear las funcionalidades de acuerdo al tamaño

Priorizar las funcionalidades

Basar las estimaciones y los planeas en hechos.

Dejar algún tiempo extra

Coordinar los equipos mirando las progresiones de los planes



- Queremos agradecer a:
 - Paula Izaurrealde,
 - Claudio Gonzalez,
 - Sebastián Araudo,
 - Marcela Garay Moyano y
 - Luciano Marzo

Lectura obligatoria

| Autor | Título | Editor | Referencia |
|-----------|---|--------|------------|
| Kent Beck | Embracing change with Extreme Programming | IEEE | 0018-9162 |

Lectura recomendada

- http://www.scrumalliance.org/pages/what_is_scrum
- <http://www.mountaingoatsoftware.com/presentations/an-introduction-to-scrum>
- <http://www.mountaingoatsoftware.com/presentations/agile-estimating>
- <http://agilemanifesto.org/iso/es/>
- <http://www.scrumalliance.org/articles/151-scrum-in-a-nutshell>
- <http://www.mountaingoatsoftware.com/system/resources/5/EmptySprintTracker.xls> (un excel para aprender a hacer scrum tracking)

Bibliografía

- Ken Schwaber; Scrum Development Process; 1995
- Ken Schwaber and Jeff Sutherland; Scrum Guide; Scrum Alliance; 2010
- Kent Beck; Embracing Change with Extreme Programming; IEEE; 1999
- Brent Barton et al.; Reporting Scrum Project Progress to Executive Management through Metrics; Scrum Alliance; 2005
- Victory Szalvay et al; Agile Transformation Strategy; Danube; 2005
- Jeff Sutherland et al.; Scrum and CMMI Level 5: The Magic Potion for Code Warriors; 2007;
- Mike Cohn; Agile Estimating and Planning ; Prentice Hall; 2006; 0-13-147941-5
- Mary and Tom Poppendieck; Lean Software Development: An Agile Toolkit; Addison-Wesley; 2003;
0-321-15078-3
- <http://www.infoq.com/news/2009/09/key-elements-agile-retrospective>

Bibliografía

- <http://continuousdelivery.com/>
- <http://www.agilejournal.com/articles/current-edition/6101-agile-journal-may-2011>
- <http://www.martinfowler.com/bliki/Xunit.html>
- <http://martinfowler.com/articles/continuousIntegration.html>
- <https://jazz.net/library/article/474>
- <http://www.itwriting.com/blog/4797-continuous-integration-vs-continuous-delivery-vs-continuous-deployment-what-is-the-difference.html>
- <http://www.extremeprogramming.org/rules.html>
- http://en.wikipedia.org/wiki/Extreme_programming_practices
- <http://jamesshore.com/Blog/Continuous-Integration-is-an-Attitude.html>
- <http://jamesshore.com/Blog/Continuous-Integration-on-a-Dollar-a-Day.html>
- <http://jamesshore.com/Blog/Forces-Affecting-Continuous-Integration.html>
- http://jamesshore.com/Agile-Book/continuous_integration.html

Bibliografía

- <http://martinfowler.com/delivery.html>
- <http://www.continuousintegrationtools.com/?opensource>
- <http://www.continuousintegrationtools.com/?commercial>
- <http://www.continuousintegrationtools.com/?reviews>
- <http://confluence.public.thoughtworks.org/display/CC/CI+Feature+Matrix>
- http://blogs.oracle.com/theaquarium/entry/continuous_integration_needs_and_solutions
- <http://www.javaworld.com/javaworld/jw-06-2007/jw-06-awci.html>
- <http://www.slideshare.net/gabrielspmoreira/software-product-measurement-and-analysis-in-a-continuous-integration-environment>
- [http://msdn.microsoft.com/en-us/library/cc948982\(v=office.12\).aspx](http://msdn.microsoft.com/en-us/library/cc948982(v=office.12).aspx)
- http://www.agilesherpa.org/agile_coach/
- http://www.scrumalliance.org/pages/what_is_scrum
- <http://www.mountaingoatsoftware.com/presentations/an-introduction-to-scrum>
- <http://www.mountaingoatsoftware.com/presentations/agile-estimating>
- <http://agilemanifesto.org/iso/es/>
- <http://www.scrumalliance.org/articles/151-scrum-in-a-nutshell>
- <http://www.mountaingoatsoftware.com/system/resources/5/EmptySprintTracker.xls>

Versión

| Versión | Fecha | Comentarios | Autor |
|---------------|-------------|-----------------|------------------|
| 1.0.0_Draft_A | 01-Oct-2012 | Versión inicial | Natalia Andriano |
| 1.0.0 | 25-Oct-2012 | Baseline | Natalia Andriano |

Back up slides

Integración Continua y RTC

Único repositorio de fuentes

- Componente de source control,
- Administración del flujo de cambios -> Compartir artefactos,
- Permite definir y crear branches (desarrollo, mantenimiento, integración, etc.),
- Componentes: código fuente, tests unitarios, librerías, etc.

Automatizar el build

- Componente **Team Build**.
- Provee soporte para la automatización y monitoreo y de los builds.
- Permite representar definiciones, máquinas y resultados de los builds.

Hacer el build auto testeable

- El **Team Build** permite ejecutar y analizar resultados de pruebas unitarias.

Commits diarios

- Funcionalidad **Personal Build**.
- Permite ejecutar un build en su workspace.
- El equipo no se entera de estas ejecuciones.

Integración Continua y RTC

Commit => build

- Build manual: el **Team build** permite a los desarrolladores solicitar la ejecución de un build.
- Build Automático: ejecutado después de un commit.
 - Follow up (Request Build Execution)
- Build Programado: diarios, semanales, etc.

Builds rápidos

- **Team Build:** permite detectar que actividades del build consumen demasiado tiempo.

Ambiente separado igual al de producción

- Respecto de este tema RTC no tiene demasiado para aportar.

Obtención de ejecutables

- Herramientas Ant:
 - Tareas Ant -> artifactLinkPublisher y artifactFilePublisher -> permiten publicar los artefactos.
 - Build Team crea un snapshot del branch usado en el build y esa información se envía con los resultados del build.

Integración Continua y RTC

Cualquiera puede ver que está pasando

- **Tablero de componentes (Dashboard):** componente Web que brinda información **rápida** del estado del proyecto.
- Provee drill down.
- Integra la información de todos los componentes,
- CI
 - Historia de ejecuciones exitosas y fallidas
 - Reportes

Automatizar el despliegue

- Team Build: Se puede automatizar el despliegue en múltiples ambientes: testing, producción