

MASTERCLASS

Automatización de
Pruebas





Victor Portugal

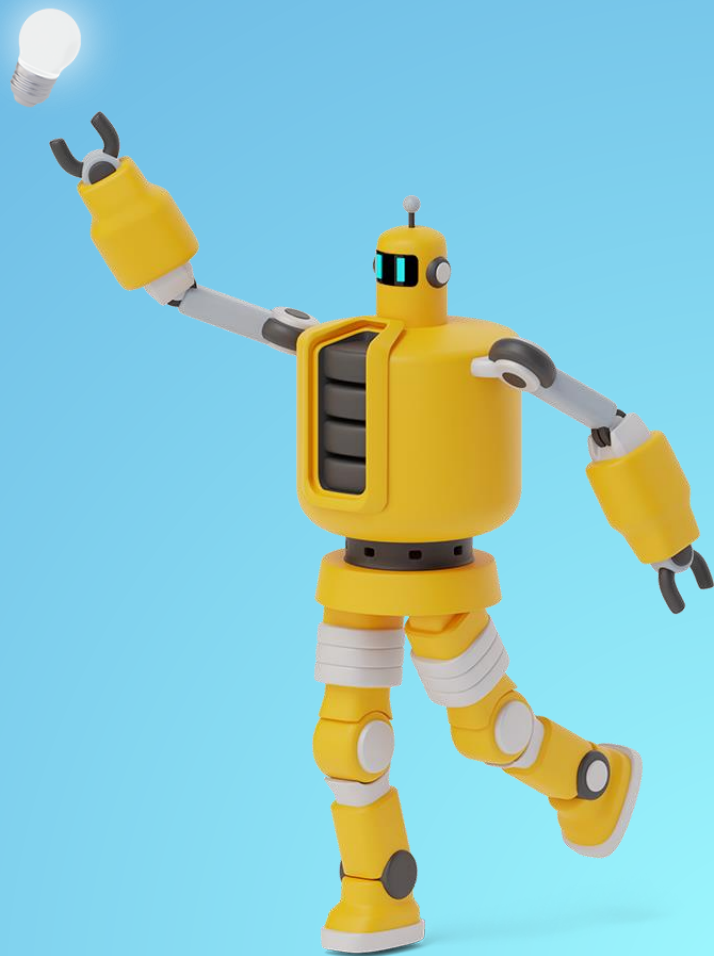
Microsoft Certified Technology Specialist (MCTS).
Certified Scrum Master (CSM).
Certified Scrum Developer (CSD).
Design Thinking Professional Certificate (DTPC)
Associate Android Developer (AAD).

Temario:



- ☒ ¿Qué significa desarrollo ágil?
- ☒ Buenas prácticas de pruebas dentro del sprint.
- ☒ TDD, BDD, ATD
- ☒ Prácticas de XP (Extreme Programming)
- ☒ Herramientas para automatización de pruebas.
- ☒ Test para métodos.
- ☒ Compiladores.
- ☒ Test de aceptación.
- ☒ Hojas de escenarios de pruebas.
- ☒ Pruebas funcionales de aceptación (front end)
- ☒ Unit test.
- ☒ Integración continua (Azure Devops)
- ☒ Códigos de ejemplos.
- ☒ Experiencias.

INTRODUCCIÓN A PRÁCTICAS DE DESARROLLO ÁGIL

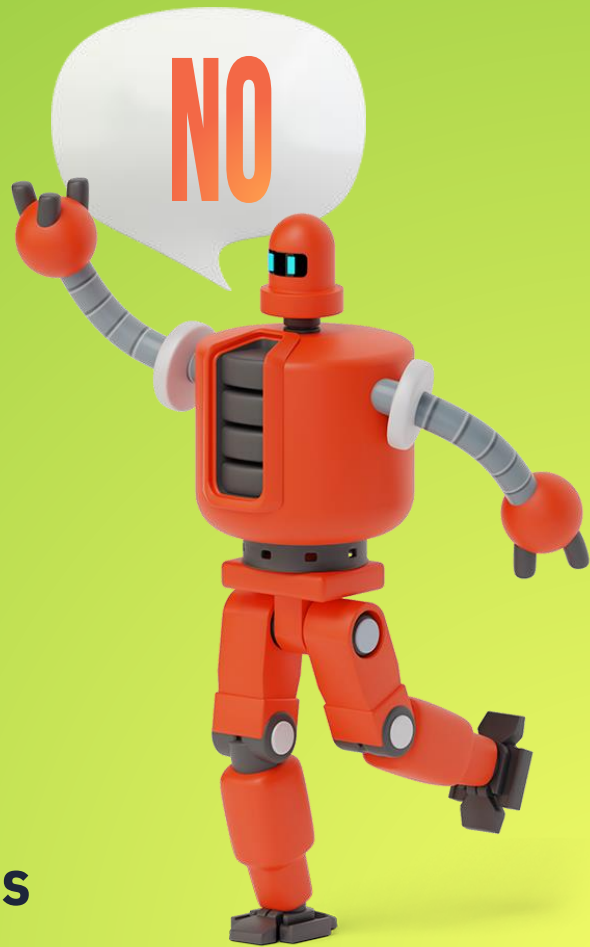


¿ Qué significa para
ti hacer “Desarrollo
ágil”?



AGILE DEVELOPER : QUE NO

- No es sólo programar
- No es ser un / una “gurú técnico/a”
- No es hacer muchas reuniones



AGILE DEVELOPER : QUE SÍ

- **Tener foco en la entrega de valor al negocio.**
- **Tener foco en la calidad del producto.**
- **Poder dar respuesta al cambio.**
- **Tener responsabilidad por el proceso de desarrollo de punta a punta.**
- **Trabajar en equipo “no rockstars”**



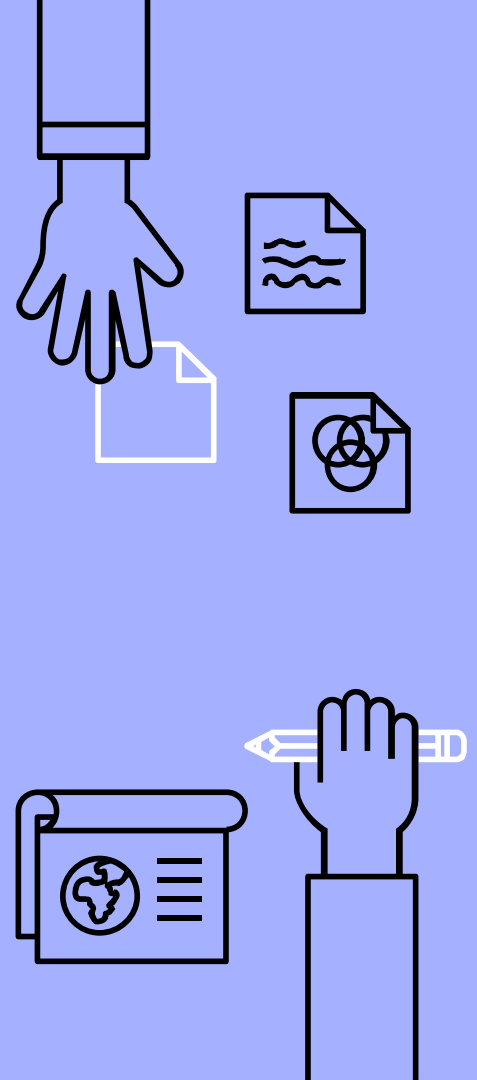
**Calidad en las
entregas**

**Disminuir
errores en
producción**

Sprint

**Entregas mas
rápidas de
productos**

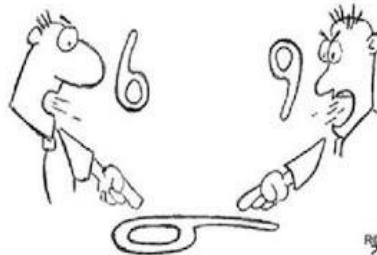
**Disminuir
tiempos en
pruebas**



Normalmente al realizar pruebas funcionales

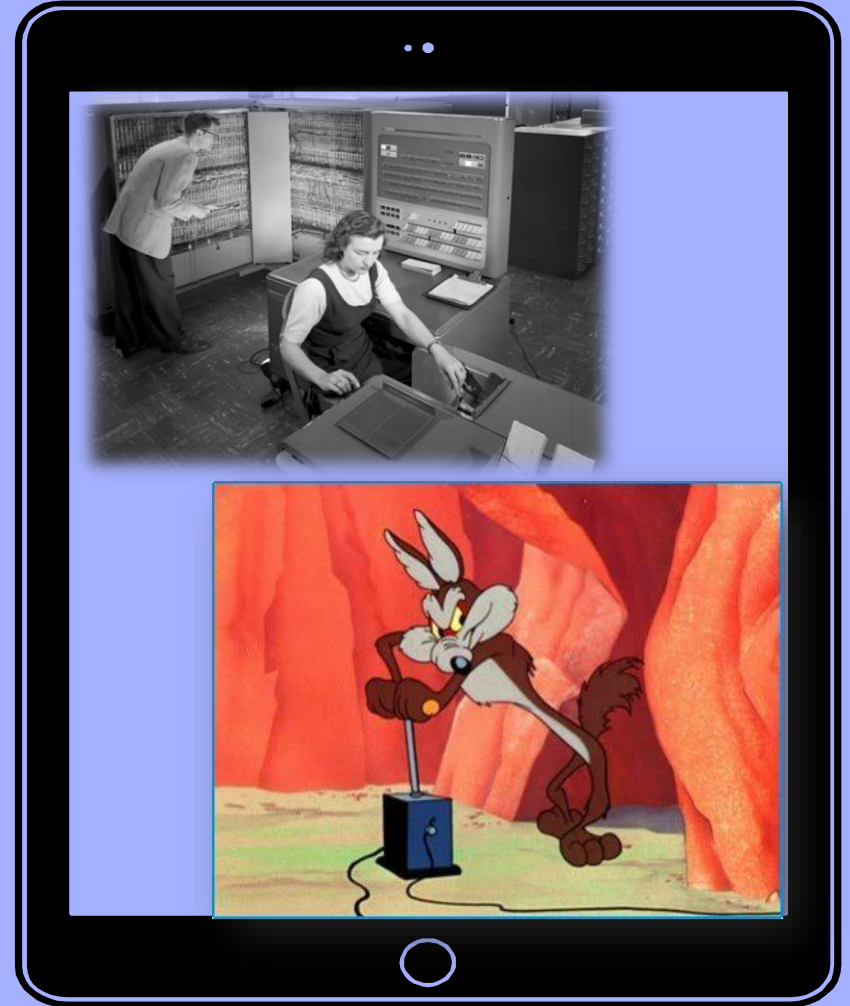
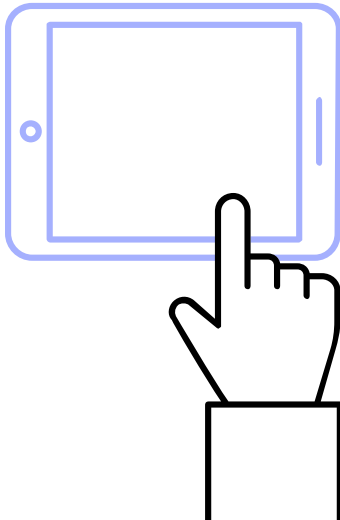


Developer V/s Tester



Clasificación del presente estudio	Clasificación por Singh y Chana (2012)	Alcance de la clasificación para los dos estudios
1) Infraestructura y plataformas como servicio	1) Arquitectura	Hace referencia a la estructura y los recursos de los sistemas en la nube, comprende servicios, software de comunicación entre aplicaciones (middleware) y aspectos de software.
2) Calidad	2) Desarrollo basado en componentes y reusabilidad 3) Calidad del servicio	Se incluyen los temas relacionados con rendimiento, pruebas funcionales, calidad del servicio ofrecido, calidad en los requerimientos del negocio, calidad en la ejecución de programas, paralelismo, seguimiento a los servicios, modelos de medición de servicios, procesos de desarrollo, reutilización y calidad en los datos.
3) Seguridad	5) Seguridad	En primera instancia se incluyen los temas de seguridad y privacidad, cumplimiento y los acuerdos legales. En segunda instancia se encuentran los temas relacionados con problemas de seguridad en proveedores y los que enfrentan los clientes y de tercera instancia, los problemas que se presentan a nivel de seguridad al utilizar la infraestructura como servicio.
4) Conceptualización	---	Esta clasificación incluye los temas que permiten definir aspectos conceptuales en la calidad de la computación en la nube, por ejemplo: definiciones, ventajas y desventajas, servicios en la nube, arquitectura base, entre otros.
5) Software como servicio	4) Diseño	Temas relacionados con la selección de servicios por parte de clientes, sistemas de búsqueda con tolerancia a fallos, requerimientos y retos del software como servicio y algoritmos para el aprovisionamiento de recursos en la prestación de servicios.
6) Caso aplicado	---	Incluye los temas que presentan resultados de aplicaciones reales que generan datos relevantes para soportar análisis y mediciones.
7) Acuerdos de niveles de servicio	---	Acuerdos de servicio entre clientes y proveedores, modelos y métricas para cuantificar violaciones en los acuerdos.
8) Móviles	---	Hace referencia al uso de la nube para aplicaciones móviles, inconvenientes en el desarrollo, análisis de los recursos limitados de un móvil y la ventaja de usar la nube.

Pero en mi
ambiente si
funciona.....



La automatización de pruebas de software es una poderosa estrategia que tiene importantes beneficios para el negocio.

Las pruebas automatizadas tienen como objetivo detectar fallas en el software evitando que una persona tenga que ejecutar las pruebas manualmente. En este caso, el experto en testing genera un caso a probar utilizando una herramienta para que luego la misma se realice automáticamente. No requiere la intervención del individuo en cada nueva ejecución. La prueba simula la interacción humana con el software.



Tecnicas

01

ATDD

...

02

BDD

...

03

ATDD

...





¿Qué es BDD?

BDD refiere a Behavior Driven Development, o sea, desarrollo dirigido por comportamiento. Como bien lo indica su nombre, no se trata de una técnica de testing, sino que es una estrategia de desarrollo (así como TDD, que es test driven development). Lo que plantea es definir un lenguaje común para el negocio y para los técnicos, y utilizar eso como parte inicial del desarrollo y el testing. Por esto es importante saber como parte del team entiendas bien qué es BDD.

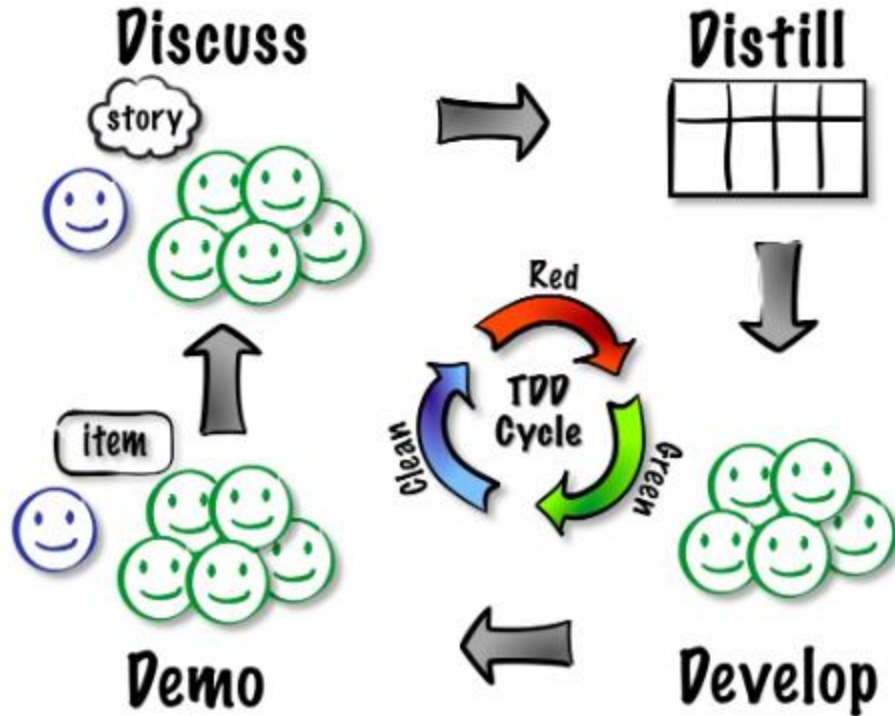


BDD y TDD

“BDD es una evolución del TDD. En TDD se enfoca a la prueba unitaria, en cambio en BDD se enfoca en la prueba de más alto nivel, la prueba funcional, la de aceptación, el foco está en cumplir con el negocio y no solo con el código. Ambos enfoques podrían convivir, ya que se podría especificar una prueba de aceptación, y luego refinar en pruebas unitarias al momento de codificar esa funcionalidad en las distintas capas. Tal vez una ventaja clara de BDD es que los desarrolladores se enfocan en ver qué es lo que tiene que funcionar y de qué forma a nivel de negocio..”



Acceptance Test Driven Development (ATDD) Cycle





ATDD (Acceptance Test Driven Development)

Es una práctica que fomenta la comunicación con el/la dueño/a de producto y permite al equipo garantizar que el desarrollo cumple los requerimientos de negocio





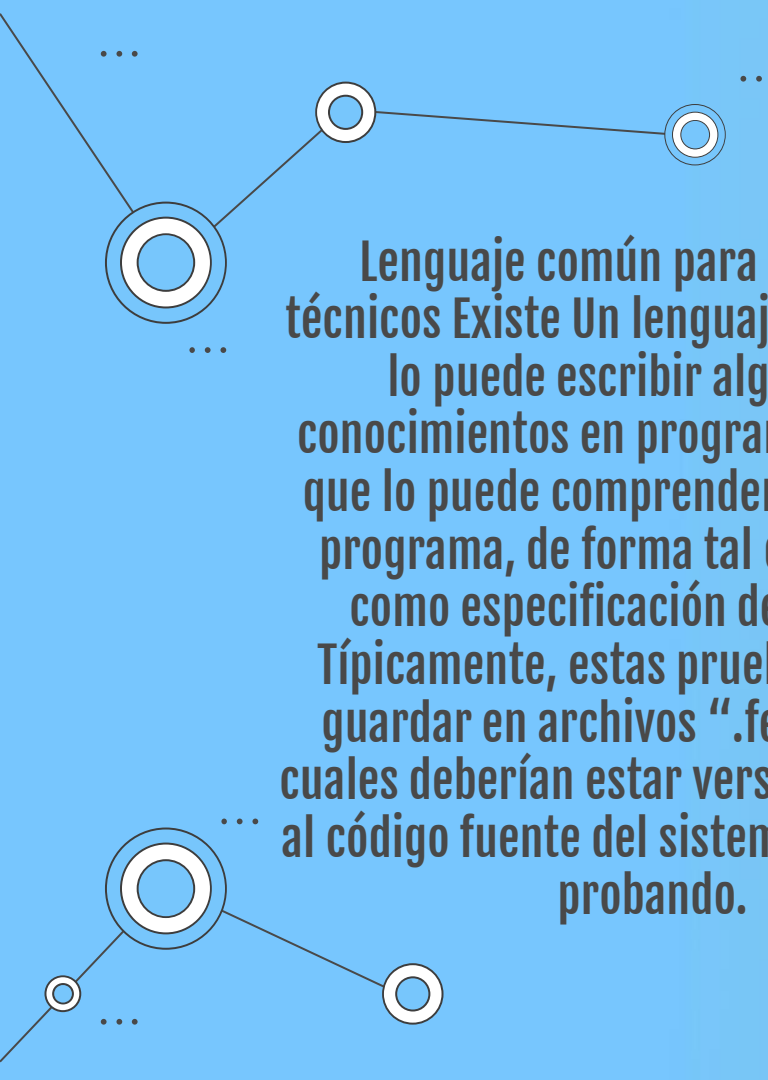
- Son pruebas de integración
- Garantizan que la aplicación cumpla con los requerimientos funcionales
- Una vez completas, representan un requerimiento y no deberían fallar (excepto que cambie el requerimiento)

ATDD



BDD estrategia que encaja bien en las metodologías ágiles, ya que generalmente en ellas se especifican los requerimientos como historias de usuario. Estas historias de usuario deberán tener sus criterios de aceptación, y de ahí se desprenden pruebas de aceptación, las cuales pueden ser escritas directamente en lenguaje Gherkin

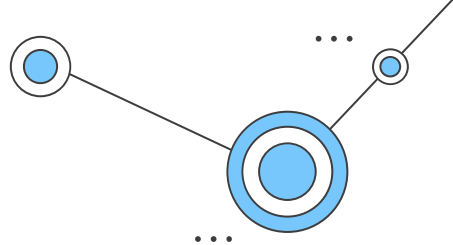




Lenguaje común para negocio y técnicos Existe Un lenguaje común, que lo puede escribir alguien sin conocimientos en programación, pero que lo puede comprender también un programa, de forma tal de utilizarlo como especificación de pruebas. Típicamente, estas pruebas se van a guardar en archivos “.feature”, los cuales deberían estar versionados junto al código fuente del sistema que se está probando.



```
1: Feature: Some terse yet descriptive text of what is desired
2:   Textual description of the business value of this feature
3:   Business rules that govern the scope of the feature
4:   Any additional information that will make the feature easier to understand
5:
6:   Scenario: Some determinable business situation
7:     Given some precondition
8:     And some other precondition
9:     When some action by the actor
10:    And some other action
11:    And yet another action
12:    Then some testable outcome is achieved
13:    And something else we can check happens too
14:
15:   Scenario: A different situation
16:     ...
```

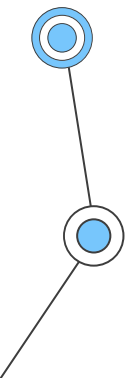


En estos archivos se especifica:

- **Feature:** nombre de la funcionalidad que vamos a probar, el título de la prueba.
- **Scenario:** habrá uno por cada prueba que se quiera especificar para esta funcionalidad.
- **Given:** acá se marca el contexto, las precondiciones.
- **When:** se especifican las acciones que se van a ejecutar.
- **Then:** y acá se especifica el resultado esperado, las validaciones a realizar.

Puede haber un *feature* por archivo y este contendrá distintos escenarios de prueba.

Nota: Gherkin soporta muchos lenguajes, así que si bien lo más común es verlo en inglés, se podría hacer en español también.



Herramientas para automatización de pruebas

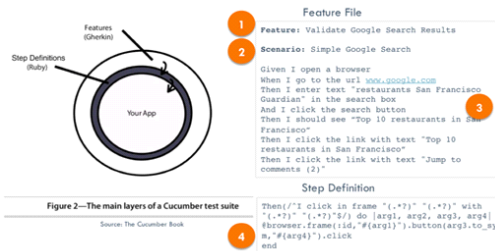




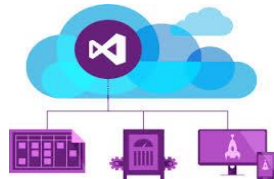
cucumber



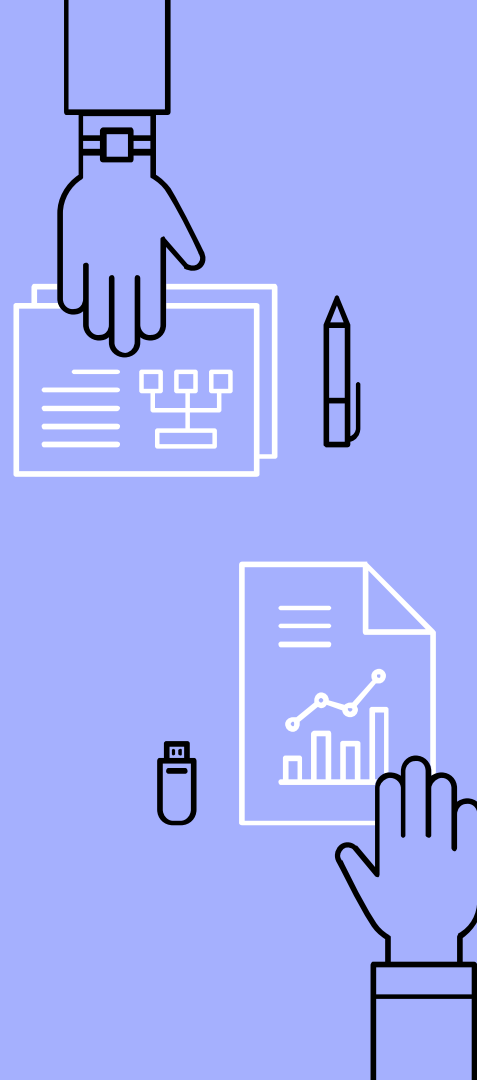
specflow
Cucumber for .NET

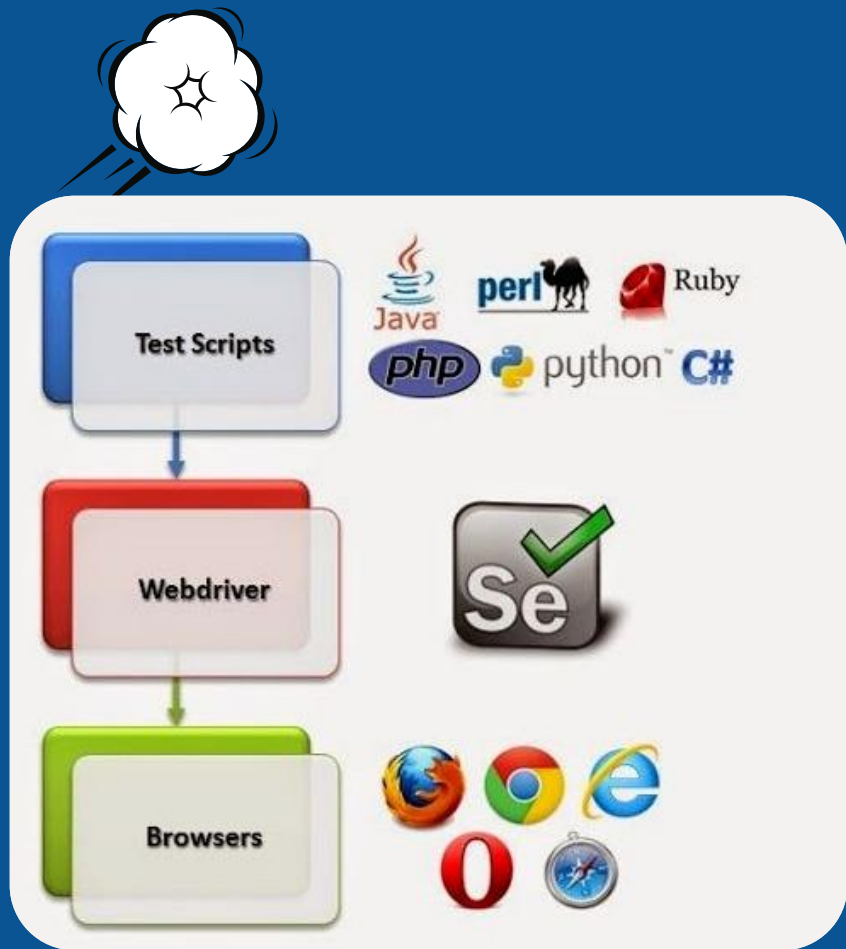


Empaquetados y Despliegues

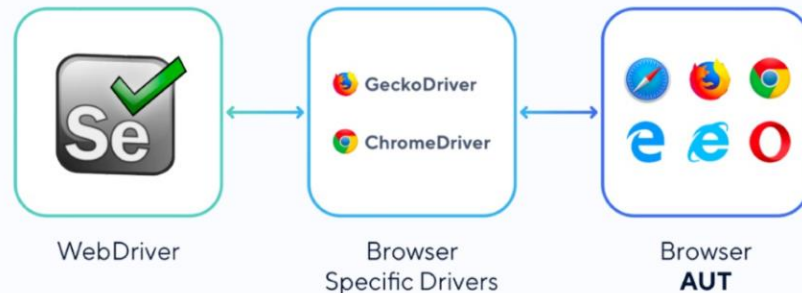


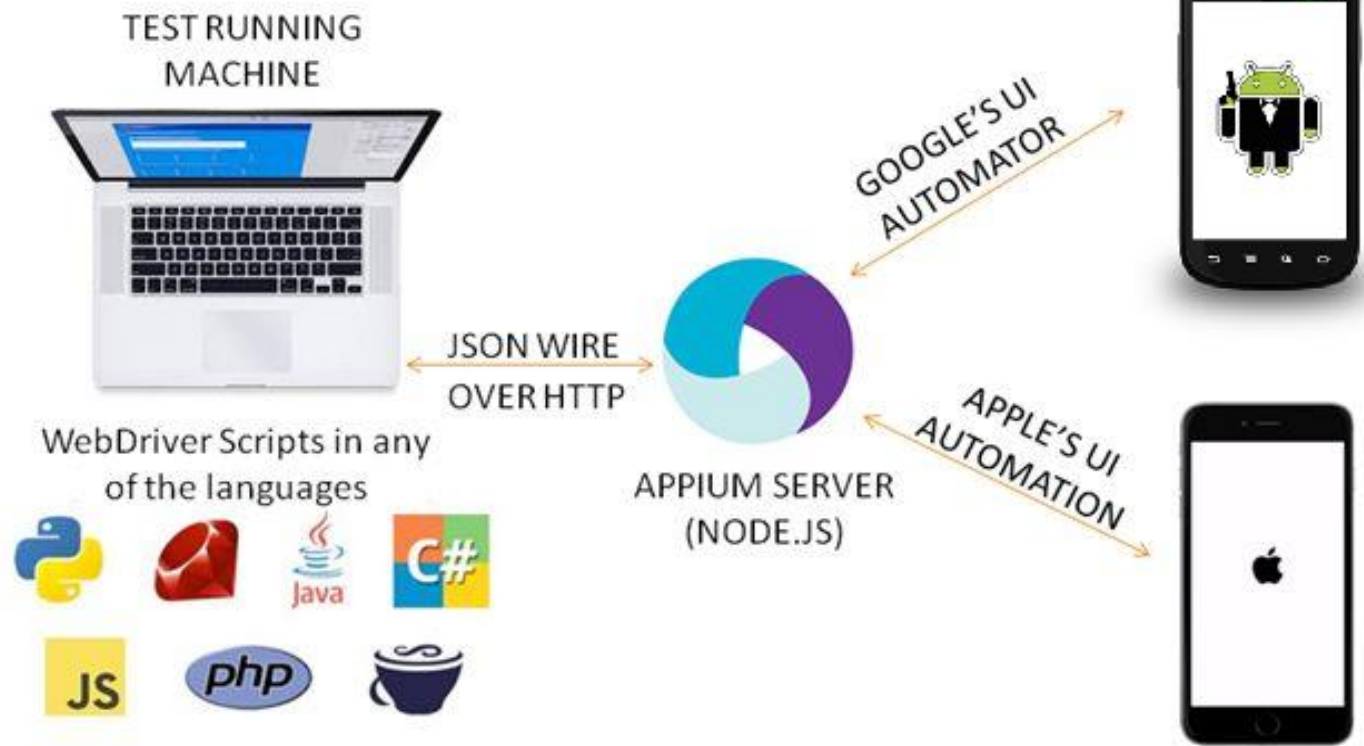
Nexus





Selenium WebDriver Architecture

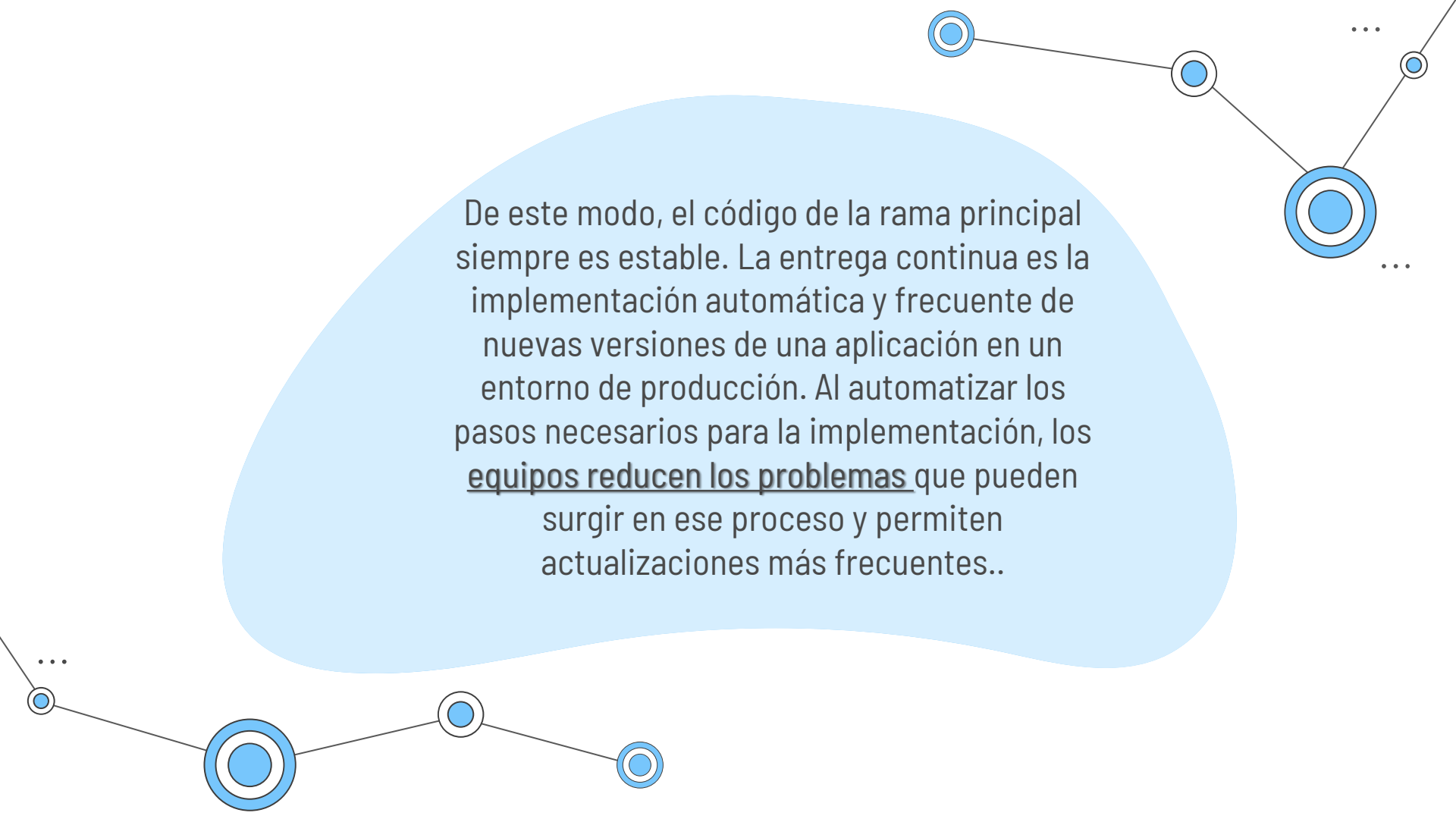




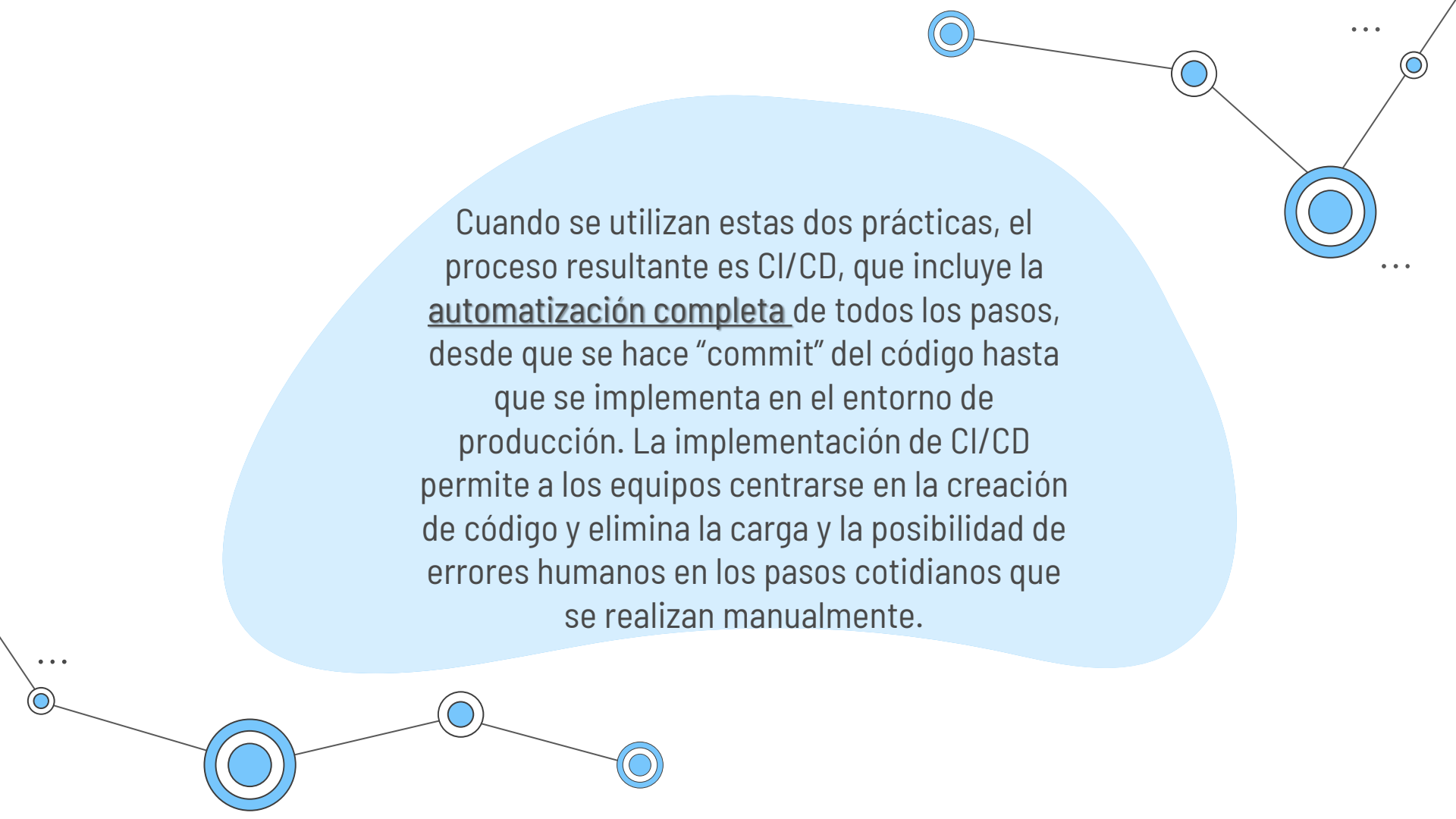


Integración y entrega continuas (CI/CD)

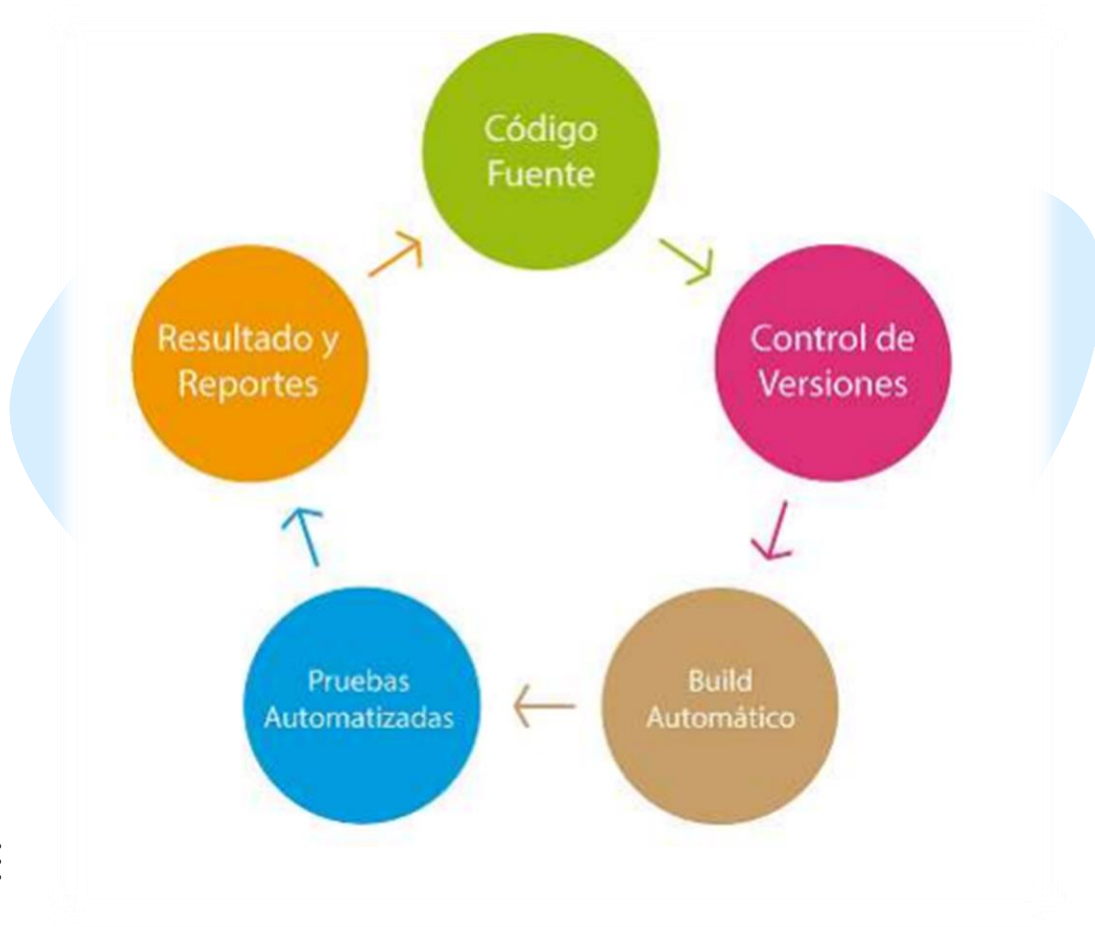
La integración continua es una práctica de desarrollo de software en la que los desarrolladores fusionan mediante combinación los cambios de código en la rama de código principal con frecuencia. En la integración continua se utilizan pruebas automáticas, que se ejecutan cada vez que se hace “commit” de código nuevo.



De este modo, el código de la rama principal siempre es estable. La entrega continua es la implementación automática y frecuente de nuevas versiones de una aplicación en un entorno de producción. Al automatizar los pasos necesarios para la implementación, los equipos reducen los problemas que pueden surgir en ese proceso y permiten actualizaciones más frecuentes..



Cuando se utilizan estas dos prácticas, el proceso resultante es CI/CD, que incluye la automatización completa de todos los pasos, desde que se hace “commit” del código hasta que se implementa en el entorno de producción. La implementación de CI/CD permite a los equipos centrarse en la creación de código y elimina la carga y la posibilidad de errores humanos en los pasos cotidianos que se realizan manualmente.



Visualización de la demostración - x Home - Microsoft Azure - x Overview dashboard - Overview - x

https://dev.azure.com/sccsoft/SCC2.0/_dashboards/dashboard/5ebf1514-04b9-4636-93cb-80aedf6c5b58

Azure DevOps

sccsoft / SCC2.0 / Overview / Dashboards

Search

VP

SCC2.0

SCC2.0 Team - Overview

Edit Refresh

Welcome

Get started using Azure DevOps to make the most of your team dashboard.

Manage Work
Add work to your board

Collaborate on code
Add code to your repository

Continuously integrate
Automate your builds

Visualize progress
Learn how to add charts

Work assigned to Victor Portugal

Last time you checked, there were no results.

SCC2.0 Team

G S MQ JA

Work

Backlog
Board
Task board
Queries

Sprint Burndown

Set iteration dates to use this widget.

Set iteration dates

New Work Item

Enter title

Bug

Create

Open User Stories

0 Work items

Visual Studio

Open in Visual Studio
Requires Visual Studio 2013+

Get Visual Studio
See Visual Studio downloads

Review settings

https://dev.azure.com/sccsoft/SCC2.0/_backlog/1/SCC2.0 Team?team=SCC2.0 Team

Type here to search

8:58 AM
3/22/2021

¿Qué es eXtreme Programming (XP) ?

“Es una metodología de desarrollo que se diferencia de las tradicionales por poner mas énfasis en la adaptabilidad que en la predictibilidad y pone foco en las prácticas técnicas para lograrlo.



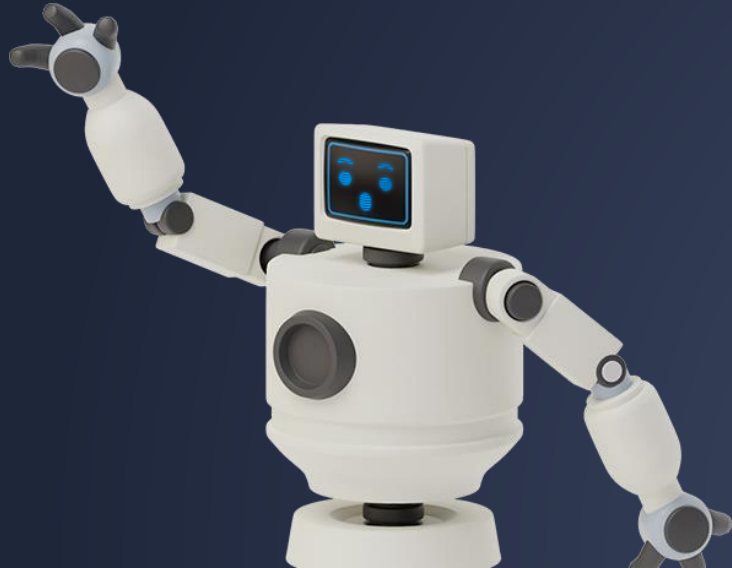
Promueve la aplicación de prácticas de ingeniería apropiadas para la creación de software. Tiene como objetivo principal que un equipo de desarrollo pueda producir software de mejor calidad de forma constante y a su vez busca promover una buena calidad de vida para el equipo.



A close-up photograph of a laboratory experiment. A hand on the left is pouring a bright green liquid from a test tube into a flask that already contains an orange liquid. Another hand on the right holds a test tube containing a blue liquid, positioned as if it will also be added to the mixture. The background is a plain, light-colored surface.

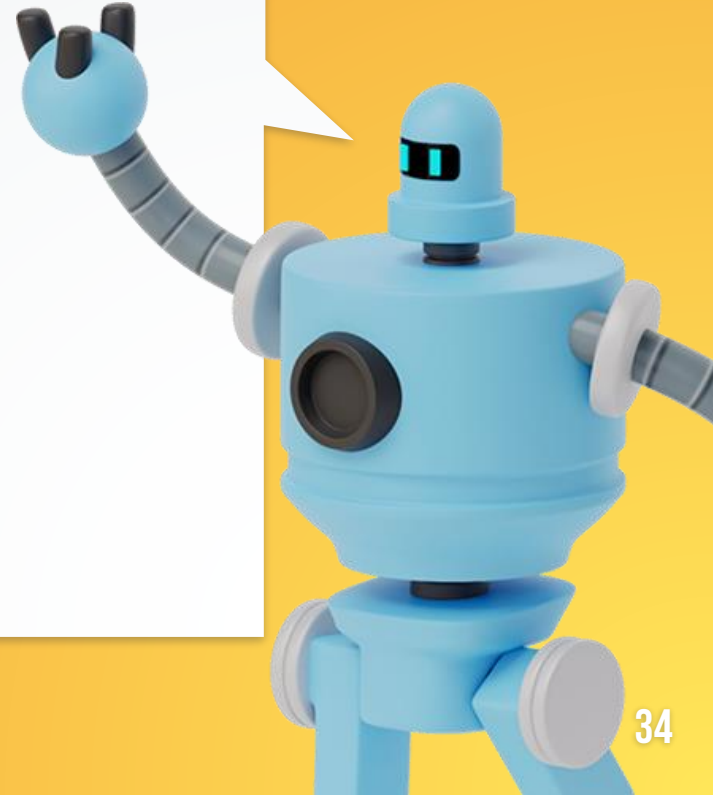
PRÁCTICAS DE XP

Propone buenas prácticas de planificación, organización, comunicación y de ingeniería de software, permiten crear una cultura de equipo de excelencia. Las prácticas de Extreme Programming (XP) más populares son:



***Desarrollo guiado por pruebas (TDD,
Test Driven Development):***

Es un enfoque evolutivo en la ingeniería de software que combina 2 prácticas que permiten crear código de calidad, pensar en la arquitectura del software que queremos desarrollar, escribiendo la prueba primero y luego mejorarla a través de la refactorización.



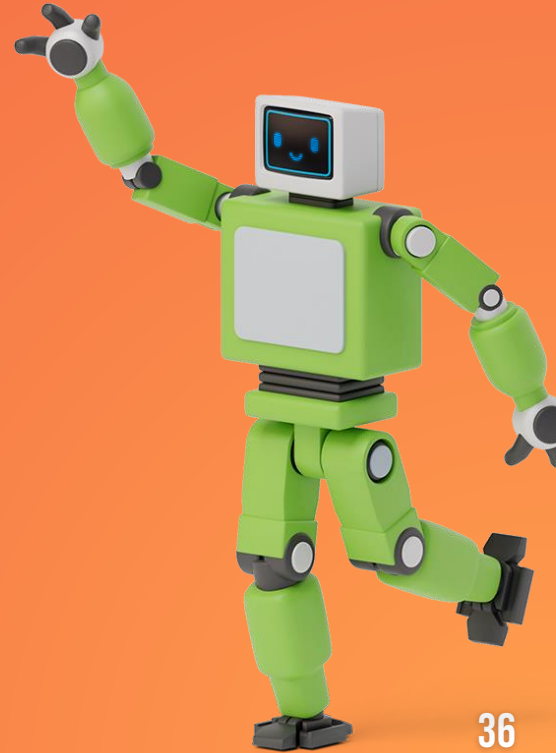
Refactorización (Refactoring):

Mejorar el diseño del código existente sin cambiar su comportamiento. Programación en parejas (Pair Programming): Dos programadores trabajando en pareja en una sola máquina, resolviendo el mismo problema.



Integración Continua **(Continuous Integration):**

Práctica especialmente diseñada para construir o integrar todas las etapas de desarrollo, identificar errores y eliminarlos durante el proceso de desarrollo, reduciendo así el tiempo de respuesta y mejorando la calidad del software que se lanza como resultado.

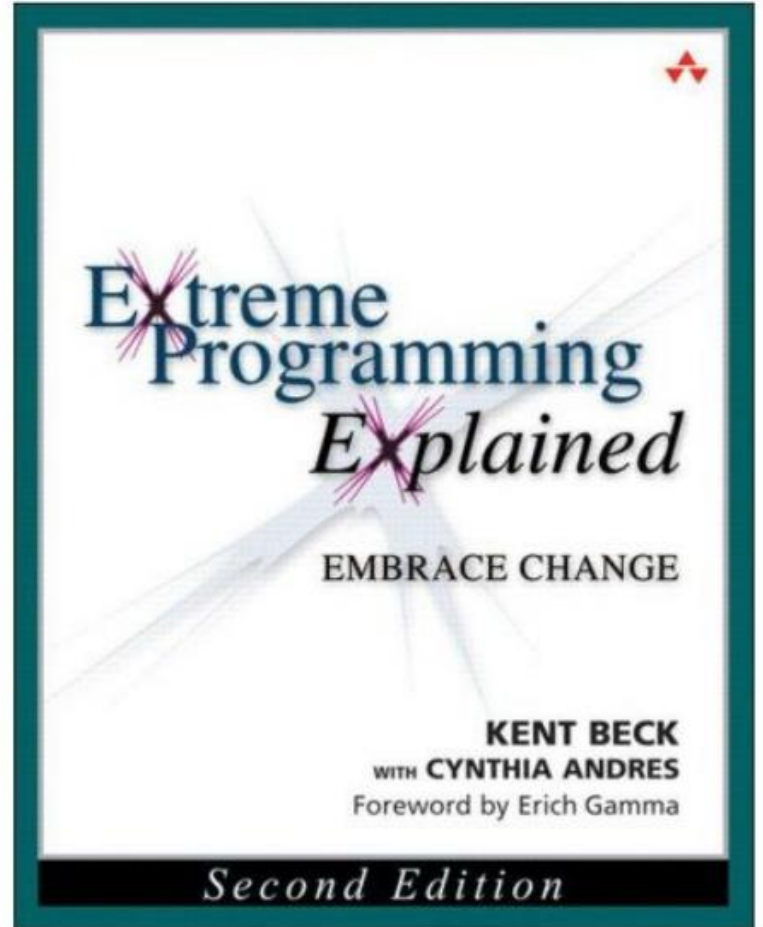


BIBLIOGRAFÍA

Extreme Programming Explained by Kent Beck

In order to build high-quality software, you need good agile management (e.g. Scrum) and strong technical practices such as XP.

- ☐ Involve the whole team
- ☐ Increase technical collaboration through pair programming and continuous integration
- ☐ Reduce defects through developer testing
- ☐ Align business and technical decisions through frequent planning and conversation
- ☐ Improve teamwork by setting up informative, shared workspaces



Veamos Código!

