Implementación de pruebas unitarias en Angular

Juan Pablo Gómez Restrepo

Ingeniero de sistemas

Asesor

Carlos Mauricio Duque Restrepo, Ingeniero de sistemas e informática

Universidad de Antioquia

Facultad de ingeniería

Ingeniería de sistemas

Medellín

2022

Cita	Gómez Restrepo
Referencia Estilo IEEE (2020)	[1] J. P. Gómez Restrepo, “Implementación de pruebas unitarias en Angular”, Presencial, Ingeniería de sistemas, Universidad de Antioquia, Medellín, 2022.

CENDOI

Repositorio Institucional: http://bibliotecadigital.udea.edu.co

Universidad de Antioquia - www.udea.edu.co

Rector: John Jairo Arboleda Céspedes.

Decano/Director: Jesús Francisco Vargas Bonilla.

Jefe departamento: Diego Jose Luis Botia Valderrama.

El contenido de esta obra corresponde al derecho de expresión de los autores y no compromete el pensamiento institucional de la Universidad de Antioquia ni desata su responsabilidad frente a terceros. Los autores asumen la responsabilidad por los derechos de autor y conexos.

Tabla de contenido

RESUMEN **7**

ABSTRACT **8**

I. INTRODUCCIÓN **9**

II. OBJETIVOS **10**

A. Objetivo general 10

B. Objetivos específicos 10

III. MARCO TEÓRICO **11**

IV. METODOLOGÍA **13**

V. RESULTADOS **14**

VI. CONCLUSIONES **19**

REFERENCIAS **21**

LISTA DE TABLAS

TABLA I: CASOS DE PRUEBA POR TEST 17
TABLA II: NÚMERO DE INCIDENCIAS DE COMPONENTE POR SPRINT 18

LISTA DE FIGURAS

Fig. 1. Resultado de pruebas unitarias en consola 16

Fig. 2. Resultado de pruebas unitarias en el navegador 16

Fig. 3. Set de pruebas del componente A 19

SIGLAS, ACRÓNIMOS Y ABREVIATURAS

AAA	Arrange Act Assert
BDD	Behavior Driven Development
SPA	Single Page Application
TDD	Test Driven Development
npm	Node package modules

RESUMEN

Gracias a la importancia del aseguramiento de la calidad en proyectos de software y los avances que ha venido teniendo la implementación de pruebas unitarias en estos proyectos, cada vez se hace más necesario incluir pruebas durante la construcción del software. De esta forma, el presente documento expone: una serie de conceptos teóricos clave para entender y abordar el desarrollo de pruebas unitarias; los resultados de un conjunto de pruebas unitarias implementadas usando Jasmine para un aplicativo web construido en Angular; y un análisis de los resultados de las pruebas que revela una mejoría sustancial en la calidad del proyecto en relación al estado anterior.

***Palabras clave* — Framework, Angular, Jasmine, software, web, test unitarios, calidad.**

ABSTRACT

Due to the importance of quality assurance and today's main advances in the implementation of unit tests in software projects, it is necessary to include those tests during the software construction. In this way, this document exposes a series of key theoretical concepts to understand and approach the development of unit tests, the results of a set of tests implemented in Jasmine for a web application in Angular and an analysis of some results which reveal a substantial improvement in quality in relation to the initial state of the project, just before the tests implementation.

***Keywords* — Framework, Angular, Jasmine, software, web, unit test, quality.**

I. INTRODUCCIÓN

En ingeniería de software, las pruebas unitarias suelen implementarse en proyectos de toda clase. Estas pruebas evalúan pequeñas piezas de código y validan que funcionen bien de forma aislada. La importancia de estas radica en que ayudan a validar el funcionamiento del código, antes de llevarlo a ambientes productivos. Por otro lado, ayudan a los desarrolladores a identificar y corregir errores de manera temprana o a refactorizar y mejorar su código inicial.

Por otra parte, las enormes capacidades que ofrecen los Framework de hoy en día, los hacen indispensables para el desarrollo de software de forma óptima y organizada [1]. Por ello existen una variedad Framework en la industria para distintos propósitos y unos de ellos permiten implementar pruebas unitarias para aplicaciones web SPA. Básicamente son una interfaz simple para el desarrollo de pruebas unitarias y ayudan a obtener información valiosa acerca del estado del proyecto.

Debido a que la implementación de estas pruebas supone la identificación oportuna de errores, las compañías las incluyen con más frecuencia en sus proyectos para garantizar calidad, reducir tiempo, esfuerzo y dinero.

De esta forma, se pretende implementar mediante la metodología Kanban [2], un conjunto de pruebas unitarias en un proyecto web en Angular [3], exponer los resultados de dichas pruebas y medir su impacto en la calidad del proyecto.

II. OBJETIVOS

A. Objetivo general

Implementar y documentar el proceso de creación de pruebas unitarias para una aplicación web, como parte del ciclo de vida de software en un proyecto empresarial.

B. Objetivos específicos

Implementar un conjunto de pruebas unitarias para nuevas funcionalidades dentro de una aplicación web en Angular.

Mostrar los resultados de las pruebas implementadas y analizarlos en aras de conocer su impacto en los procesos de calidad del proyecto.

III. MARCO TEÓRICO

Actualmente el tema de las pruebas es muy importante para la industria del desarrollo de software. Incluso existen estrategias de programación enfocados a las pruebas como lo es TDD.

TDD (Test driven development por sus siglas en inglés) es una estrategia para la construcción de software en donde primero los requerimientos son convertidos en casos de prueba y luego el software se construye para pasar las pruebas. Este enfoque se basa en la repetición de pequeños ciclos de desarrollo, en donde primero se desarrolla una prueba, luego se implementa el código para pasar dicha prueba y por último se acepta o rechaza el código dependiendo del resultado de la prueba [4].

Existen otros enfoques de programación en donde las pruebas también son importantes como lo es BDD (Behavior Driven Development), que es una estrategia de desarrollo por comportamiento. En BDD también las pruebas se deben definir antes del desarrollo, pero estas se centran en el usuario y el comportamiento del sistema, a diferencia del TDD [5].

Cabe destacar que, a pesar de que el departamento de calidad sigue haciendo pruebas manuales, las pruebas que se mencionan en la estrategia BDD y TDD, son diferentes, pues son pruebas unitarias automatizadas. La ventaja de las pruebas unitarias automatizadas es que resulta más fácil verificar la funcionalidad de cada componente de una aplicación pues estas se ejecutan una a una de forma automática, en poco tiempo.

La pruebas que se generan en conjunto a la construcción del software y están a cargo de los desarrolladores, se les conoce con el nombre de Pruebas Unitarias y no solo ayudan a encontrar errores oportunamente y a comprobar el correcto funcionamiento de una unidad lógica, sino para comprender el funcionamiento y los objetos principales del código.

En cuanto a la elaboración de las pruebas unitarias, es importante tener claro el concepto de Caso de Prueba. Un caso de prueba es un conjunto de condiciones bajo las cuales se determina si una unidad lógica dentro del software o una característica del mismo es parcial o

completamente satisfactoria y estos deben estar construidos a la luz de los Criterios de Aceptación, que son las condiciones que el software debe cumplir para ser aceptado por un usuario [6].

En este sentido, para evaluar la completitud de una unidad lógica, es recomendable crear por lo menos dos casos de prueba para cada requisito. Uno de estos casos debe demostrar que se ha alcanzado el requisito de manera satisfactoria, la Prueba Positiva. El otro caso de prueba, que se conoce como Prueba Negativa, debe demostrar que el requisito sólo se ha alcanzado en las condiciones deseadas.

Otro aspecto importante para la elaboración de test unitarios es el patrón AAA. El patrón AAA (Arrange-Act-Assert) se ha convertido casi en un estándar en la industria y sugiere que se debe dividir nuestro método de prueba en tres secciones: organizar, actuar y afirmar. Cada uno de ellos es el único responsable de la parte en la que lleva su nombre. Entonces, en la sección de Arrange, solo tiene el código necesario para configurar esa prueba específica. Aquí se crearán objetos, se instalarán simulacros (si se está usando uno) y se establecerán expectativas potenciales. Luego está el Act, que debería ser la invocación del método que se está probando. Y en Assert se comprueba si se cumplieron las expectativas. [7].

De esta forma, los conceptos de aislamiento de la prueba, prueba automatizada, patrón AAA, el desarrollo de pruebas positivas/negativas, la completitud de las pruebas y la creación de Stubs y Mocks son expuestas, entre otros, algunos de los conceptos principales tenidos en cuenta a la hora de construir pruebas unitarias durante las prácticas.

Ahora bien, para la variedad de lenguajes en aplicaciones web como Javascript, Php y Java, entre otros, existen framework o tecnologías que ayudan a integrar y automatizar las pruebas unitarias en los proyectos. Para el caso de javascript, existen alternativas orientadas a pruebas de interfaz y experiencia de usuario como Jasmine [8] y otros multipropósito como Jest [9]. Este tipo de Framework también ayuda a obtener información de la cobertura de las pruebas del proyecto y a correrlas de forma automática para analizar resultados e integrarlos al ciclo de vida del software.

IV. METODOLOGÍA

Para el desarrollo de las actividades de estas prácticas, se usó Kanban por ser un método visual que ha ganado gran popularidad en corporaciones y empresas de todo el mundo ya que permite gestionar el trabajo de forma eficiente y autónoma [2].

Como herramienta para el marco de trabajo, se creó un tablero en la plataforma Trello [10] para la visualización, creación y gestión de las tareas. Se fueron añadiendo y moviendo dichas tareas conforme a las necesidades del proyecto y sugerencias del asesor. Además, gracias a las ventajas de la metodología, las tareas se trabajaron de forma autónoma y se completaron de manera satisfactoria con una cantidad de dependencias casi nula gracias a la correcta división y asignación de funcionalidades de los desarrolladores en el proyecto.

Se realizaron reuniones de seguimiento periódicas, entre el profesor asesor, el asesor de la empresa y el estudiante; para la revisión de las entregas de valor del proyecto, en donde la implementación de las pruebas unitarias fue el insumo que permitió obtener los resultados y conclusiones de estas prácticas. De esta forma, el siguiente cronograma muestra el avance del proyecto y las entregas realizadas en el tiempo:

Ahora bien, con relación a la metodología para el desarrollo del software de la compañía dónde se hicieron las prácticas, no se siguió una metodología de forma estricta ya que a pesar de que hubo una reunión diaria para los avances e impedimentos, propias de metodologías ágiles como Scrum [11], no se tenían establecidas fechas de entrega y no se media el esfuerzo de las tareas. La falta clara de una metodología hizo que hubiera mucha incertidumbre en el equipo de desarrolladores a la hora de iniciar una tarea, pues muchas veces los criterios no quedaban establecidos en las tareas, no tenían suficiente completitud o eran ambiguos. Esto también afectó el orden y la cantidad de incidencias debido a requerimientos incompletos.

V. RESULTADOS

En primer lugar, es pertinente aclarar algunos aspectos para esta sección:

- Las pruebas unitarias se implementaron en una aplicación desarrollada en Angular con Karma y Jasmine [8], tecnologías aprobadas y utilizadas en los proyectos de la compañía. Esto condujo a la elección de estas tecnologías y, en general, el tema de Pruebas unitarias en SPA [12], para el desarrollo de esta práctica.
- Existen limitaciones para la exposición de algunos datos y resultados de esta sección debido a temas de confidencialidad pactados entre la empresa en dónde se desarrollaron las prácticas y el cliente para quien se entregaba el software. Así, datos como el nombre del cliente y la aplicación, código fuente que permita identificarlo, variables de ambiente y/o cualquier tipo de información sensible que pueda ser utilizada para vulnerar el sistema no serán registrados en este informe para evitar incumplir con las políticas de privacidad pactadas.
- Las pruebas unitarias se implementaron en una aplicación desarrollada en Angular con Karma y Jasmine, tecnologías aprobadas y utilizadas en los proyectos de la compañía. Esto condujo a la elección de estas tecnologías y, en general, el tema de Pruebas unitarias en SPA, para el desarrollo de esta práctica.
- Existen limitaciones para la exposición de algunos datos y resultados de esta sección debido a temas de confidencialidad pactados entre la empresa en dónde se desarrollaron las prácticas y el cliente para quien se entregaba el software.
- En su lugar, algunos resultados serán parcialmente mostrados y/o alterados para enseñar datos irrelevantes, ocultar los datos sensibles y así contextualizar y visualizar el resultado de la aplicación de las pruebas, que es lo importante.

Teniendo en cuenta lo anterior, con respecto al repositorio donde se aloja el código al que se le hicieron las pruebas unitarias (proyecto Angular), se analizó el porcentaje de cobertura del código, el número de casos de prueba por test en promedio, la percepción del equipo y el porcentaje de cobertura del código.

Es importante conocer las capacidades del software que se usa para recubrir herramientas que

puedan simplificar este tipo de análisis y ayudar a evaluar un proyecto de forma rápida. En este caso, mediante npm [13] es posible correr un comando que arroja un resumen de la cobertura del código en la consola y otro, mucho más ilustrativo, en el navegador. De esta forma, al ejecutar el comando: *npm run test* el resultado en consola es es el siguiente:

```
TOTAL: 34 SUCCESS
TOTAL: 34 SUCCESS
TOTAL: 34 SUCCESS

=============================== Coverage summary ======
Statements   : 35.2% ( 151/429 )
Branches     : 14.86% ( 26/175 )
Functions    : 28.48% ( 45/158 )
Lines        : 34.11% ( 132/387 )
======================================================
```

Fig. 1. Resultado de pruebas unitarias en consola

El número de líneas en el resumen (**Fig. 1**), expresa el porcentaje de las líneas de código del proyecto en general, que tienen pruebas. Siendo 34.11% un número no tan malo para el inicio de las pruebas relativamente cercano a la toma de estos resultados, pero bastante malo en términos de estándares de calidad en la industria del software.

```
Karma v4.1.0 - connected
Chrome 89.0.4389 (Mac OS X 11.2.3) is idle
Jasmine 3.4.0
••••••••••••••••••••••••••••••••••
34 specs, 0 failures, randomized with seed 10506

PreProfileService
  #getBankTabsBasedOnUserScore should be
    • kavakCapital=true and otherBanks=true when user qualification is R and there are no alerts
    • kavakCapital=true and otherBanks=false
    • kavakCapital=true and otherBanks=true when user qualification is P and there are no alerts
    • kavakCapital=true and otherBanks=true when user qualification is P and there are no KO alerts
    • kavakCapital=true and otherBanks=true when user qualification is R and there are no alerts
    • kavakCapital=false and otherBanks=true when user qualification is P and there are KO alerts
    • kavakCapital=false and otherBanks=true when user qualification is E
    • kavakCapital=false and otherBanks=true when user qualification is R and there are KO alerts
  • #getUserScore should return score value from observable
  • #generateUserCredentials should return credentials value from observable
  • should be created
CarsService
  • should be created
  • Should call getUrl method from apiService when use getCar
  • Should make a request to endpoint get car
CarrierService
  • Should call postUrl method from apiService when use getKikoyaUserData
  • Should call postUrl method from apiService when use postinsuranceEstimate
  • Should call putUrl method from apiService when use putUserData
  • Should call getUrl method from apiService when use getUserData
  • Should call postUrl method from apiService when use postUserAuth
  • should be created
  • Should call getUrl method from apiService when use getInsurance
InsuranceItemComponent
TableInfoComponent when pdf available
  • should call showPDF() when click button
  • should call syncInfo() when click button
TableInfoComponent without pdf
  • should call setTable() on component changes (ngOnChanges)
  • should create table-info component
  • should call setTable() on component init (ngOnInit)
  • should call switchDisabledButtons() when click button
  • should call financeRun() when click button
ModalWarningComponent
  • Should not create a component if the modal propeties are incorrect
  • Should not create a component if the modal propeties does not exist
  • Should call close modal method when acept button is press
  • should create a component with the correct text
  • should create
```

Fig. 2. Resultado de pruebas unitarias en el navegador

Se evidencia que los 34 casos de prueba implementados hasta ese momento (**Fig. 2**), pasaron exitosamente. Además, se puede ver un resumen de las descripciones por componente de cada uno de los casos de prueba ejecutados en la prueba.

El primer análisis tiene que ver con el número de casos de prueba por test en promedio, ya que este dato da nociones del nivel de completitud de las pruebas a través de los casos de prueba. En el caso particular de Angular, las pruebas por cada archivo (*describe*) equivalen a los casos de prueba generales y las pruebas específicas (*it*) son las que validan cada caso de prueba. Para conocer dicho promedio básicamente se debe contar la cantidad de pruebas específicas (*it*) y la cantidad de casos de prueba totales (*describe*) para obtener un promedio de los casos de prueba por cada test.

TABLA I
CASOS DE PRUEBA POR TEST

Factor	Cantidad
Test implementados	6
Casos de prueba	8
Casos de prueba promedio	1.333

Como se expuso anteriormente, basta con exponer 2 casos de prueba mínimamente (un caso negativo y otro positivo) para ver que los test se están construyendo de forma correcta pues abarcan diferentes flujos y alternativas. Por tal motivo, los 1,333 casos de prueba en promedio que muestra la **TABLA I** no se consideran una buena cantidad de casos de prueba para un proyecto por lo que es recomendable prestar mayor atención en la construcción y tener en cuenta alternativas de error entre otros flujos.

Otro punto es la percepción de los desarrolladores, ya que a modo de retroalimentación, durante una reunión en la que participó el equipo que implementó pruebas unitarias, algunos integrantes coincidieron en que durante el proceso de elaboración de las pruebas, se pensaron casos que normalmente no se hubiesen tenido en cuenta y que además se hicieron mejoras al código durante su construcción y también después del fallo de algunas pruebas.

Durante la sesión también se resaltó que es importante conocer las herramientas y los conceptos de manera clara ya que en muchas oportunidades, se añaden pruebas que no suman o aportan al porcentaje de cobertura de código y no son más que tiempo perdido.

Ahora bien, la otra sección de los resultados tiene que ver con un análisis de los test en Angular para una funcionalidad en particular. Se exponen resultados, no a nivel de proyecto sino respecto a una funcionalidad específica a la cual se le realizaron test. Esto con el propósito de mostrar de qué forma se realizaron los test y cómo afectan estos al reporte de incidencias. A pesar de que se hicieron varios test para varias funcionalidades dentro del tiempo de prácticas en diferentes repositorios, el análisis en esta sección tomará como referencia solo para un par de componentes.

Por una parte, se tiene el componente A en una de las aplicaciones. No se hará énfasis en describir en qué consiste la funcionalidad por los motivos de confidencialidad expuestos anteriormente y porque técnicamente las pruebas unitarias deberían dar suficiente contexto para entender la funcionalidad del componente en cuestión.

Además, se tendrá en cuenta el componente B al cual no se le hicieron pruebas unitarias, pero sí se pudo tener un seguimiento de incidencias al igual que el componente A.

Básicamente se escogieron estos componentes porque fueron unos de los pocos a los cuales se les pudo dar un seguimiento real de incidencias ya que tenían una complejidad y tamaño similares además de ser desarrollados de forma paralela. Por otro lado, cabe destacar que la comparación de solo 2 componentes evidentemente no proporciona un indicador del estado de las pruebas generales del proyecto, para ello se hizo el análisis global de la sección anterior (Análisis del proyecto Angular). En particular, esta sección busca evidenciar una real disminución de incidencias gracias a la implementación de pruebas unitarias en los componentes del aplicativo. La siguiente tabla muestra el número de incidencias reportadas por cada Sprint (ciclo de desarrollo de 20 días aproximadamente) posterior al Sprint donde se realizaron las pruebas:

TABLA II
NÚMERO DE INCIDENCIAS DE COMPONENTE POR SPRINT

Componente	Incidentes sprint 1	Incidentes sprint 2	Incidentes sprint 3
A (con pruebas)	5	2	0
B (sin pruebas)	7	3	0

De la **TABLA II** se puede ver que durante el Sprint posterior al desarrollo de las pruebas (Sprint 1), el componente A, que tenía pruebas unitarias, obtuvo 2 incidencias menos que B, el cual no tenía pruebas. En el siguiente (Sprint 2), a pesar de que la diferencia se redujo a 1, el componente A siguió presentando menos incidencias que B con lo que puede verse, aunque poco, una disminución sustancial de la cantidad de incidencias gracias a las pruebas unitarias.

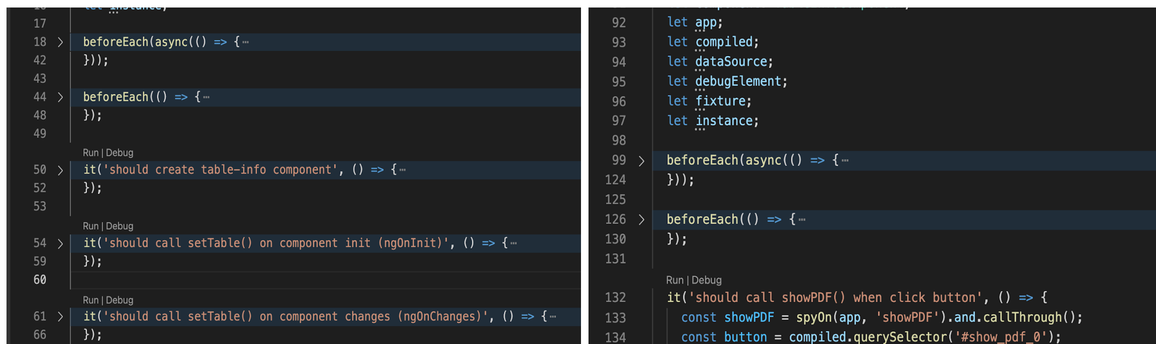

Fig. 3. Set de pruebas del componente A

Estas pruebas (**Fig. 3**), abarcan 2 casos de prueba que hacen referencia a una prueba negativa y positiva respectivamente. La primera prueba equivale a un caso de prueba dónde se debe evaluar el funcionamiento cuando falla algún servicio o no se tienen los datos suficientes para que el componente se comporte de manera normal. En segunda prueba, la prueba positiva, se tienen en cuenta las acciones que el sistema puede hacer mientras se tenga la información necesaria para que el componente presente un funcionamiento correcto. Ya que el código de las pruebas muestra de cierta forma de qué manera está hecho el componente y puede exponer datos sensibles (**Fig. 3**), el contenido de cada prueba se presenta semioculto (colapsado) dejando sólo la descripción de lo que se desea probar.

VI. CONCLUSIONES

Debido a que muchas de las pruebas implementadas para el desarrollo de esta práctica fueron las primeras que se añadían al repositorio del proyecto, los resultados y en general la apreciación de las personas que hicieron parte del proceso, superaron las expectativas iniciales. Sin embargo, fue evidente el hecho de que la forma de implementar las pruebas puede mejorar abarcando más casos de prueba ya que los 1.333 casos de prueba por test que se encontraron, no son suficientes para abarcar los diferentes flujos de ejecución de las funcionalidades desarrolladas.

Se notó que la implementación de pruebas unitarias significó un mayor tiempo y esfuerzo para cada desarrollo pues los tiempos de entrega se alargaron sustancialmente en los Sprints donde se implementaron pruebas unitarias en FE. De esta forma, es importante estimar un esfuerzo mayor para las tareas que requieren estas pruebas.

Aunque la curva de aprendizaje para el manejo de las tecnologías empleadas no fue muy pronunciada debido a experiencias previas a nivel académico y laboral. La cantidad de inyección de servicios e importaciones que deben añadirse a las pruebas y en general todo el proceso de aislamiento de la prueba, hacen que el manejo de la herramienta no sea muy intuitivo y requiere de cierta experiencia para trabajar con ella de manera fluida.

La implementación de las pruebas no solo permite evaluar la funcionalidad de las piezas de código con una mirada funcional-cualitativa ya que también se pueden obtener datos cuantitativos como el porcentaje de cobertura de código en los test añadidos y en todo el repositorio, lo cual fue bastante útil para medir la completitud e impacto de las pruebas implementadas en general.

Implementar las pruebas en Jasmine para Angular significó un gran reto técnico, en especial cuando la documentación oficial del framework se queda corta para ayudar a la solución de errores y más aún cuando el sistema de logs y mensajes de error Karma no es muy diciente. En

este caso, si no hubiese una alta demanda de solución de errores en plataformas online para desarrolladores, sería mucho más difícil construir estas pruebas.

A pesar de ya que se había tenido la oportunidad de trabajar un poco con dichas tecnologías en experiencias académicas y laborales anteriores. No se había podido medir y evaluar el impacto de las mismas en un proyecto, lo cual significa un aprendizaje que aporta positivamente al crecimiento profesional.

Generalmente, es difícil saber si una prueba puede subir o no el porcentaje de cobertura de nuestro código, para lidiar con esto fue de gran ayuda la herramienta de análisis de cobertura que viene con Angular, ya que con ella se puede revisar exactamente qué líneas de código no cubre nuestras pruebas y esto nos puede dar ideas para crear test que suban el porcentaje de cobertura.

Para algunos desarrolladores que hicieron parte de la implementación de pruebas unitarias, por primera vez en el proyecto, significó un aprendizaje importante que les ayudó a identificar mejoras en su código mientras lo construían.

.

La falta de adopción de una metodología y de detalle a la hora de definir tareas y sus criterios de aceptación, por parte de los dueños del producto, hace difícil medir la calidad real del proyecto, pues en muchos casos se reportan como incidentes detalles que no se especificaron en ningún requerimiento.

REFERENCIAS

[1] F. Grandjean, "¿Qué es un framework? - Wild Code School" [En línea]. Disponible en: https://www.wildcodeschool.com/es-ES/blog/que-es-un-framework

[2] J. Martins, "¿Qué es la metodología Kanban y cómo funciona?" [En línea]. Disponible en: https://asana.com/es/resources/what-is-kanban

[3] "Angular" [En línea]. Disponible en: https://angular.io/docs

[4] D. Rivera, "Cómo implementar Test-Driven Development (TDD) sin morir en el intento". [En línea]. Disponible en: https://bit.ly/3UEXIGg

[5] Vergara, S, "¿Qué es BDD (Behavior Driven Development)?" [En línea]. Disponible en: https://www.itdo.com/blog/que-es-bdd-behavior-driven-development

[6] A. Ballarin, "Como diferenciar la definición de done y criterio de aceptación" [En línea]. Disponible en: https://bit.ly/3G431em

[7] P. Gomes, "Unit Testing and the Arrange, Act and Assert (AAA) Pattern" [En línea]. Disponible en: https://bit.ly/3NRNNuJ

[8] "Jasmine" [En línea]. Disponible en: https://jasmine.github.io/

[9] "Jest" [En línea]. Disponible en: https://jestjs.io/

[10] "Trello" [En línea]. Disponible en: https://trello.com/

[11] K. Schwaber and J. Sutherland, "The 2020 Scrum Guide" [En línea]. Disponible en: https://scrumguides.org/scrum-guide.html

[12] J. Castro, "Single Page Applications vs. Server Side Rendering vs. Generadores de Sitios Estáticos" [En línea]. Disponible en: https://platzi.com/blog/spa-vs-ssr-vs-static-site-generators/

[13] "npm" [En línea]. Disponible en: https://www.npmjs.com/