S01: 01 Ago - 06 Ago

Lectura

Barker, Jacquie "Beginning Java Objects: From Concepts to Code". APress. 2005. Segunda edición.

1. Abstraction and Modeling

Investigación

Manifiesto Ágil. ¿Por qué? ¿Cuáles son sus valores? ¿Cuáles son sus principios?
Programación Extrema. ¿Para qué? ¿Cuáles son sus reglas?

BDD. ¿Qué significa? ¿Qué propone?

MDD. ¿Qué significa? ¿Qué propone?

SOLID. ¿Qué significa? ¿Qué propone?

UML. ¿Para qué? ¿Cuáles son algunos diagramas?
Java. ¿Para qué? ¿Cuáles son las principales ideas?

MANIFIESTO ÁGIL

¿POR QUÉ?

"Durante los fríos días del 11 al 13 de febrero del 2001, en la estación de esquí de Snowbird en las montañas de Utah, Estados Unidos; convocados por Kent Beck, se reunieron 17 reconocidos expertos de la ingeniería del Software.

El objetivo de la reunión fue debatir y buscar alternativas a los procesos tradicionales de desarrollo de software, caracterizados por la rigidez de su carácter normativo y su gran dependencia de la planificación detallada previa al desarrollo."

¿CUÁLES SON SUS VALORES?

"Los valores definidos en el Manifiesto Ágil abogan por un cambio de mentalidad, una nueva cultura organizada basada en cuatro pilares:

1. Individuos e interacciones sobre procesos y herramientas
2. "Software" funcionando sobre documentación exhaustiva
3. Colaboración con el cliente sobre negociación contractual
4. Respuesta ante el cambio sobre seguir un plan"

¿CUÁLES SON SUS PRINCIPIOS?

Los cuatro valores se concretan en 12 principios, que definen el marco de trabajo de cualquier equipo 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 proceso.
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."

2. Bibliografia
3. [1] https://muyagile.com/el-manifiesto-agil/
4. [2] https://www.bbva.com/es/agile-manifiesto-que-es/
5. [3] https://agilemanifesto.org/iso/es/principles.html

PROGRAMACIÓN EXTREMA

¿PARA QUÉ?

Se considera como un "enfoque de la ingeniería de software" en la que se da mayor importancia a la adaptabilidad del programa que a la previsibilidad. Lo que conlleva a que se va adaptando el programa a medida que se van viendo requisitos diferentes, mas no se hace el programa con unos requisitos ya previos. Esa forma de irse adaptando a los requisitos durante el transcurso de la realización del programa es llamada como programación extrema.

¿CUÁLES SON SUS REGLAS?

La programación extrema esta fundamentada en una serie de valores y principios que la guian. Los cuatro valores de XP (Extreme Programming) son:

- Comunicación
- Simplicidad
- Realimentación
- Coraje

Los devotos de la programacion extrema afirman que dichos valores son necesarios para lograr diseños y simples, métodos eficientes de desarrollo de software y clientes contentos. Estos valores, deben estar dentro del equipo de trabajo.

BDD

¿Qué significa?

Desarrollo dirigido por comportamiento. Estrategia de desarrollo que plantea
es definir un lenguaje común para el negocio
como parte del desarrollo y el testing
como
quiero
para que

¿Qué propone?

"A diferencia de TDD(Test Driven Development), BDD se define en un idioma común entre todos los *stakeholders*, lo que mejora la comunicación entre equipos tecnológicos y no técnicos, adicionalmente, en BDD, las pruebas se centran en el usuario y el comportamiento del sistema, a diferencia del TDD que se centra en funcionalidades.

Una de las caracteristicas BDD y la principal ventaja es que todas las definiciones BDD se escriben en un idioma común. El principal objetivo es que el equipo describa los detalles de cómo se debe comportar la aplicación a desarrollar, y de esta forma será comprensible por todos, por esto mismo, BDD te permite desarrollar, probar y pensar el código desde la perspectiva del usuario. Debes tener la mentalidad de implementar 'ejemplos del mundo real' en lugar de implementar solo 'funcionalidades'. Las ventajas son muy considerable para integrar a todo el equipo con un objetivo común, además de empatizar con el usuario final de tus desarrollos."

Webgrafia:

- https://www.itdo.com/blog/que-es-bdd-behavior-driven-development/#:~:text='Given%2DWhen%2DThen'%20como%20lenguaje%20com%C3%BAn%20con%20BDD,que%20se%20van%20a%20ejecutar.

BDD: responde las preguntas de como, quiero y para poder las cuales definen lo que se conoce como una historia de usuario.

Como: Se especifica el escenario, las precondiciones.

Quiero: Las condiciones de las acciones que se van a ejecutar.

Para Poder: El resultado esperado, las validaciones a realizar.

Ventajas:

1. Ya no estás definiendo 'pruebas', sino que está definiendo 'comportamientos'.
2. Mejora la comunicación entre desarrolladores, testers, usuarios y la dirección.
3. Debido a que BDD se especifíca utilizando un lenguaje simplificado y común, la curva de aprendizaje es mucho más corta que TDD.
4. Como su naturaleza no es técnica, puede llegar a un público más amplio.
5. El enfoque de definición ayuda a una aceptación común de las funcionalidades previamente al desarrollo.
6. Esta estrategia encaja bien en las metodologías ágiles, ya que en ellas se especifican los requisitos como historias de usuario y de aceptación.

https://www.itdo.com/blog/que-es-bdd-behavior-driven-development/#:~:text=%27Given%2DWhen%2DThen%27%20como%20lenguaje%20com%C3%BAn%20con%20BDD,que%20se%20van%20a%20ejecutar.

MDD

¿QUE SIGNIFICA?

MDD significa Model Driven Development , y está propone utilizar modelos gráficos y componentes de aplicaciones preconstruidos para poder construir visualmente aplicaciones complejas, esto nos permite implementar un software más rápido, efectivo y menos costoso.

Bibliografía:

"https://www.mendix.com/model-driven-development/"

"https://searchsoftwarequality.techtarget.com/definition/model-driven-development"

Modelo que se plantea en respuesta a la creciente complejidad de las aplicaciones de software y sus requisitos, dando mejores presentaciones en menores tiempos de desarrollo por medio de un alto nivel de abstracción de los modelos desde las primeras etapas del desarrollo.

modelos que se utilizan como elementos (reutilizables y con mantenibilidad)para ser procesados por un computador o una herramienta para el desarrollo de software.

Bibliografía:

http://ingenieriadesoftwarerigo.blogspot.com/2012/09/que-es-mdd.html

https://ingsoftwarei2014.wordpress.com/2014/05/25/desarrollo-de-software-dirigido-por-modelos-mdd/

¿QUE PROPONE?

Tiene bastantes beneficios, como por ejemplo: productividad, aumento de calidad, mejor compresión del sistema a desarrollar, facilita evolución y mantenimiento, facilita la reimplementación en otras tecnologías, y otras más.

"Etapas de MDD

MDD distingue al menos las siguientes etapas:

• La construcción de un modelo del dominio, denominado CIM, que expresa la lógica del negocio desde una perspectiva independiente de la computación.
• La construcción de los Modelos Independientes de la Plataforma (PIM), que expresa la funcionalidad del sistema en forma independiente de las características de plataformas de implementación específicas.
• La transformación de un Modelo Independiente de la Plataforma (PIM), en uno o más Modelos Específicos de la Plataforma (PSM).
• La transformación de modelos específicos de la plataforma (PSM) a modelos de implementación, denominados ISM."

SOLID

“Es un que representa cinco principios básicos de la programación orientada a objetos y el diseño.

Los principios SOLID son guías que nos permiten eliminar código sucio, provocando que el programador tenga que refactorizar el código fuente hasta que sea legible y extensible.

S -> Principios de responsabilidad única (SRP): Como su nombre lo indica, establece que una clase, componente o microservicio debe ser responsable de una sola cosa.

O -> Principio de Abierto/ Cerrado (OCP): Establece que las entidades de software (clases, módulos y funciones) deben estar abiertas para su extensión, pero cerradas para su modificación.

L -> Principio de sustitución de Liskov (LSP): Declara que la subclase de un programa debería ser reemplazable por su superclase sin alterar el correcto funcionamiento del programa.

I -> Principio de segregación de la interfaz (ISP): Muchas interfaces cliente específicas son mejores que una interfaz de propósito general, es decir, cuando un cliente depende de una clase que implementa una interfaz cuya funcionalidad este cliente no usa, pero que otros clientes si, este cliente estará siendo afectado por los cambios que fuercen otros clientes en dicha interfaz.

D -> Principio de inversión de la dependencia (DIP): Establece que las dependencias deben estar en las abstracciones y no en las implementaciones, es decir, los módulos de alto nivel no deberían depender de módulos de bajo nivel si no que ambos deberían depender de abstracciones. Así mismo los detalles deberían depender de las abstracciones “

Webgrafía:

- https://enmilocalfunciona.io/principios-solid/
- https://es.wikipedia.org/wiki/SOLID
- https://profile.es/blog/principios-solid-desarrollo-software-calidad/#:~:text=Los%205%20principios%20SOLID%20de,%E2%80%93%20Liskov%20Substitution%20Principle%20(LSP)

Los principios S y O

Entrando más en detalle estos primeros dos principios nos indican lo siguiente:

" Principio de Responsabilidad Única (S)

La S del acrónimo del que hablamos hoy se refiere a Single Responsibility Principle (**SRP**). Según este principio “una clase debería tener una, y solo una, razón para cambiar”. Es esto, precisamente, “razón para cambiar”, lo que Robert C. Martin identifica como “responsabilidad”.

El principio de Responsabilidad Única es el más importante y fundamental de SOLID, muy sencillo de explicar, pero el más difícil de seguir en la práctica.

El propio Bob resume cómo hacerlo: “Reúne las cosas que cambian por las mismas razones. Separa aquellas que cambian por razones diferentes”.

Este principio nos dirige hacia una cohesión más fuerte en la clase y un encaje más flojo entre la dependencia de clases, una mayor facilidad de lectura y un código con una complejidad menor.

Principio de Abierto/Cerrado (O)

El segundo principio de SOLID lo formuló Bertrand Meyer en 1988 en su libro “Object Oriented Software Construction” y dice: “Deberías ser capaz de extender el comportamiento de una clase, sin modificarla”. En otras palabras: las clases que usas deberían estar abiertas para poder extenderse y cerradas para modificarse.

Utiliza las clases de la manera que necesites, pero para modificar su comportamiento aparece cuando añades un código nuevo, nunca cuando lo modificas el viejo. El mismo principio puede aplicarse para módulos, paquetes y librerías.

Aplicando el principio de abierto-cerrado conseguirás un acople más relajado, mejoraras la lectura y finalmente, reducirás el riesgo de romper alguna funcionalidad ya existente. "

Webgrafía:

- https://profile.es/blog/principios-solid-desarrollo-software-calidad/#1_principio_de_responsabilidad_unica
- https://enmilocalfunciona.io/principios-solid/
- https://apiumhub.com/es/tech-blog-barcelona/principios-solid/

UML

UML (“Unified Modeling Language” o “Lenguaje Unificado de Modelado”) es una técnica estándar que se adoptó a nivel internacional por varias empresas y varios organismos para crear esquemas, diagramas y documentos de desarrollo de software para la especificación de los sistemas en todas sus fases.

- No es un lenguaje propiamente más bien es una serie de normas y estándares gráficos de cómo se debe representar los esquemas relativos de un software.
- Estos diagramas o esquemas ayudan a describir el comportamiento de “algo” dentro de un sistema.
- Este modelo de diagramas no solo sirve para grandes sistemas.

“*Los principales beneficios son:*

- *Mejores tiempos totales de desarrollo (de 50 % o más).*
- *Modelar sistemas (y no sólo de software) utilizando conceptos orientados a objetos.*
- *Establecer conceptos y artefactos ejecutables.*
- *Encaminar el desarrollo del escalamiento en sistemas complejos de misión crítica.*
- *Crear un lenguaje de modelado utilizado tanto por humanos como por máquinas.*
- *Mejor soporte a la planeación y al control de proyectos.”* (2)

Algunos diagramas son:

- Diagrama de clases: “*nos permitirá representar gráficamente y de manera estática la estructura general de un sistema, mostrando cada una de las clases y sus interacciones (como herencias, asociaciones, etc.), representadas en forma de bloques, los cuales son unidos mediante líneas y arcos. Los* ***diagramas de clases*** *son el pilar fundamental del modelado con UML, siendo ampliamente utilizados tanto para análisis como para diseño de sistemas y software en general.*” (3)
- Diagramas de casos de uso: “*es una secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema*”
- Diagrama de secuencia:” El diagrama de secuencia es un tipo de diagrama de interacción cuyo objetivo es describir el comportamiento dinámico del sistema de información haciendo énfasis en la secuencia de los mensajes intercambiados por los objetos.” (5)

Bibliografía:

1. Eriksson, Hans-Erik and PENKER, Magnus, http://profesores.fi-b.unam.mx/carlos/aydoo/uml.html.
2. https://culturacion.com/que-es-un-diagrama-de-clases/
3. https://www.ecured.cu/Caso_de_uso
4. Cillero Manuel, (01 de junio del 2017,), diagramas de secuencia, https://manuel.cillero.es/doc/metrica-3/tecnicas/diagrama-de-interaccion/diagrama-de-secuencia/

JAVA

JAVA el por que se retoma antes de la aparición de Java, la relación entre software y hardware era muy estrecha. El software era muy dependiente del hardware donde se ejecutaba. Es decir, cuando realizabas un programa software éste debía realizarse para las características concretas del hardware donde se iba a utilizar. Si un programador realizaba un programa para ejecutarse sobre un ordenador con un procesador Pentium, por ejemplo, éste programa debía modificarse si se quería ejecutar en un ordenador con diferentes características.

Todo esto cambió con la aparición de Java y su máquina virtual (JVM o Java Virtual Machine por sus siglas en inglés). Gracias a esta ‘máquina virtual’, un programa escrito en Java podía ejecutarse en diferentes plataformas, sin importar sus características hardware. “Write once, run anywhere”, es decir: “Escribe una vez, ejecutalo en cualquier sitio”.

Hoy en día existen multitud de JVMs para diferentes arquitecturas (32 bits, 64 bits) para todas las plataformas (Windows, Linux, iOS, Android, etc) que permiten que Java pueda ser ejecutado en multitud de dispositivos.

Las principales ideas o caracteristicas de java serian :

- Es orientado a objetos.
- Es interpretado y compilado
- Robusto. Está diseñado para disminuir el máximo de errores posible. Por ejemplo, como desarrollador no tendrás que preocuparte de reservar o liberar memoria como en C. La máquina virtual de Java lo hace por tí.
- Seguro.
- Portable.
- Multihilo: Esto significa que te permitirá ejecutar varias tareas a la vez.

Bibliografia : https://codinghub.es/que-es-java/

Al ahondar en las maquinas virtuales, las cuales convirtieron a Java en un lenguaje portátil y apto para sistemas operativos heterogéneos, se comprende mejor como logran adaptarse a la arquitectura sin necesidad de cambiar le código. Gracias a que los compiladores Java no generaban binarios para una arquitectura real sino para una virtual solo se necesita un emulador que traduzca las instrucciones del programa.

Fuente: http://di002.edv.uniovi.es/~lourdes/publicaciones/bt99.pdf

el codigo de java es mas robusto.

conocemos otros lenguajes como C++ que ofrecen más rendimiento y mayor control, pero que son mucho más difíciles de manejar sin meter la pata. Java es un lenguaje que ofrece manejo automático de la memoria y cuyos objetos no hacen referencia a datos fuera de sí mismos o de otros objetos de Java. Esto hace imposible que una instrucción de Java pueda corromper la memoria, ni "pisar" o comprometer los datos de otras aplicaciones o del propio sistema operativo. La máquina virtual realiza todo tipo de comprobaciones para asegurar la integridad que impiden que pueda romper el sistema operativo u otros programas."

podemos agregar que java no es un lenguaje complicado de aprender comparado con lenguajes clásicos como C o C++, desde luego. En general se puede decir que Java no es un lenguaje más difícil de aprender que cualquier otro, especialmente otros modernos como C# o Swift, pero sí que es más fácil de aprender que muchos otros.