PATRONES DE DISEÑO DE SOFTWARE PARA MODELOS ARQUITECTURALES EN SMART CAMPUS BASADO EN INCEPTION DECK E INGENIERÍA KAISEN

Manuel Alejandro Pastrana Pardo y Ana Milena Rojas Calero
Grupo de investigación GrinTic
Institución Universitaria Antonio José Camacho (UNIAJC)

Recepción: 05/06/2018. Aceptado: 22/10/2018.
Cómo citar este artículo:
Pastrana Pardo M.A., Rojas Calero A.M. (2018). Patrones de diseño de software para modelos arquitecturales en Smart Campus basado en Inception Deck e Ingeniería Kaisen. Revista Sapientía. 10 (20), 52-64.

RESUMEN

Inception Deck es una técnica de elicitación utilizada en marcos ágiles como SCRUM y adoptado por el proyecto Smart Campus de la Institución Universitaria Antonio José Camacho, que ofrece muchas posibilidades para el proceso de desarrollo. Gracias a ello, el proyecto Smart Campus ha comenzado a explorar cómo se puede mejorar la fase de diseño implementando la técnica mencionada y sumándole los principios de Kansei Engineering en la fase de análisis de requisitos para sugerir patrones de diseño de software a la hora de la implementación de sus proyectos.

ABSTRACT

Inception Deck is an elicitation technique use it in agile frameworks like SCRUM and adopted by Smart Campus which give a lot possibilities to the development process. Thanks' to it, The university Antonio Jose Camacho Smart Campus project has started to explore how the design phase can be improve using the technique mentioned and implementing the principles of Kansei Engineering into the requirement analysis phase to suggest software design pattern for the software construction.

PALABRAS CLAVES

Inception Deck, Ingeniería Kansei, Kansei, Sistemas de Información, Técnica de Elicitación, Patrones de Diseño de Software.

KEYWORDS

Inception Deck, Kansei Engineering, Kansei, Information Systems. Elicitation Technique, Software Design Patterns.

INTRODUCCIÓN

Una de las problemáticas más comunes en el proceso de desarrollo de software para cualquier tipo de proyecto consiste en que las soluciones requeridas por los usuarios distan de las expectativas de los usuarios finales como menciona (León Duarte, Romero Dessens & Olea Miranda, 2008). Diversos estudios como The Standish Group (2013) determinan que los usuarios finales reciben un producto que opera bajo sus necesidades básicas funcionales, pero no toma en cuenta otros aspectos relevantes como la usabilidad, la mantenibilidad, el rendimiento esperado e incluso la seguridad. Lo anterior, debido a la brecha que se presenta en la interpretación entre el levantamiento de requisitos escrito en lenguaje natural y la construcción del software (Pastrana, Ordóñez, Ordóñez & Merchan, 2017). Esta problemática es objeto de estudio y refinamiento al día de hoy para distintos investigadores, quienes buscan bajo enfoques tradicionales o ágiles minimizar esta brecha. Sin embargo, se siguen presentando inconvenientes al momento de plasmar lo especificado por el usuario en los casos de uso, historias de usuario, prototipos, etc.(Pastrana et al., 2017).

Es importante resaltar que aunque la interpretación del lenguaje natural en el que están escritos los casos de uso o las historias de usuario es un problema claro e identificado, no es el único. El involucramiento de todos los interesados del proyecto, la confrontación de sus opiniones y la mediación para adquirir una sola visión del producto son relevantes también dentro de los aspectos en los que actualmente el proceso de análisis de requisitos requiere ser refinado (Pastrana et al., 2017).

Para obtener un resultado cercano a las expectativas reales de los interesados del proyecto, es necesario un enfoque donde sea posible no solamente la unificación de la visión del producto como lo plantea (Pastrana, Ordóñez, Ordonez, Thom, & Merchan, 2018), sino la automatización de la toma de decisiones arquitecturales: aspecto en el que actualmente las entrevistas, como método principal de recopilación de información, se quedan cortas, en especial al dejar aspectos no funcionales sin identificar.

Con base en lo anterior, los autores proponen una alternativa basada en el trabajo de Pastrana et al. (2018) quien plantea la utilización de la técnica de elicitación de requisitos Inception Deck para identificar los elementos funcionales y no funcionales de un proyecto de desarrollo de software y recopilar dicha información a través del modelo de proceso de negocio o BP y el trabajo de Nagamachi (1995) que permite identificar factores constantes a la hora de implementar un diseño que impactan de manera positiva al usuario. El objeto de estudio es poder identificar constantemente atributos de calidad comunes a problemas similares para ser reutilizados creando así un modelo de referencia arquitectural común dentro del proyecto Smart Campus de UNIAJC.

PROCESO PARA EL DESARROLLO DE SOFTWARE

ANTECEDENTES

El proceso para el desarrollo de software, según Roger S. Pressman (2010), abarca un conjunto de actividades y artefactos dentro del ciclo de desarrollo tradicional, dividido en las siguientes fases:

- Planeación (Análisis de requisitos)
- Diseño
- Desarrollo (Implementación)
- Calidad
- Despliegue (Implantación)

Dentro de la fase de planeación, el equipo encargado del proyecto recopila toda la información relevante para entender el contexto, las necesidades funcionales y las restricciones implícitas en la elaboración de la solución, además de ser capaz de estimar el esfuerzo requerido y darle una prioridad de atención. El artefacto fundamental de esta fase es un documento escrito en lenguaje natural.

Para las metodologías tradicionales de desarrollo de software como Rational Unified Process-RUP y Microsoft Solution Framwork-MSF se utiliza el formato de caso de uso, estandarizado por Engineers (2008). En el caso de los marcos de trabajo ágiles como SCRUM y XP se utiliza el formato de historias de usuario, que actualmente no posee un estándar. Pese a la ausencia de un modelo de referencia universal para la elaboración de este artefacto, los dos autores más relevantes sobre cómo escribir correctamente historias de usuario son Cohn (2004) y Beck (1999).

En relación con las fases del ciclo de desarrollo, Sommerville (2005) indica que en la fase de diseño se determinan los parámetros que guían la arquitectura o drivers para un correcto diseño de la solución, basado en el resultado de la etapa anterior donde se ha identificado, tanto en el plano funcional como no funcional, las necesidades del producto a construir. Al respecto Roger S. Pressman (2010) muestra la relación de dependencia entre ambas fases, resaltando la importancia que hay en una buena elicitación para la correcta toma de decisiones en la etapa de diseño. Esto se evidencia en la figura 1.

Dentro de esta fase, el equipo se encargará de tomar decisiones que orientan la estructura principal del sistema a desarrollar. Esto se denomina atributos de calidad de una arquitectura y corresponden al comportamiento esperado del producto. Aspectos como mantenibilidad, rendimiento, integrabilidad, disponibilidad y seguridad resaltan entre los atributos más comunes a la hora de diseñar una solución de software.

Para garantizar el cumplimiento de los atributos de calidad propuestos dentro de una arquitectura, se recurre al uso de patrones de diseño que son soluciones recurrentes, efectivas y comprobadas a problemas comunes que ya existen. El uso de patrones de diseño, adicionalmente permite plantear de una manera más fácil y en términos formales una solución arquitectural para construir la solución requerida.

Por otra parte, la fase de construcción permite que las decisiones de diseño, producto del análisis, sean materializadas a través del lenguaje de programación y componentes requeridos. Esto va muy de la mano con la fase calidad en la que se comprueba si las funcionalidades descritas en el análisis responden adecuadamente a su descripción y si las decisiones arquitecturales fueron satisfechas correctamente a través de los patrones de diseño implementados.

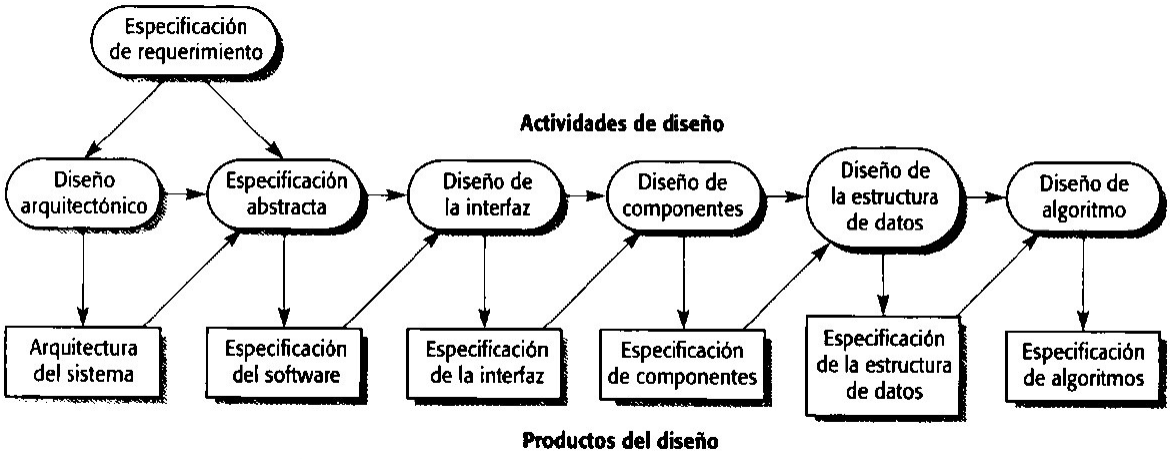

Figura 1. Modelo general del proceso de diseño. Imagen tomada de ingeniería de software un enfoque aplicado. Roger S. Pressman

Por último, en la fase de despliegue, se realiza la réplica en el ambiente de producción de lo expuesto en ambiente de pruebas para que los usuarios finales puedan operar con el aplicativo.

INCEPTION DECK Y SU APLICACIÓN EN EL ANÁLISIS DE REQUISITOS

Inception Deck es una técnica de elicitación de requisitos presentada a la comunidad por primera vez por Rasmusson (2006), posteriormente refinada por Rasmusson (2010). El objetivo fundamental de esta técnica es la unificación de la visión del producto a ser construido para que todos los interesados obtengan el resultado esperado.

La técnica conforme a la guía construida por Ripenhausen (n.d.) sugiere una forma de orientar la actividad basada en 10 dinámicas que alinean la visión del producto frente a todos los integrantes del proyecto a través de la interacción que se da entre stakeholders, equipo de desarrollo, gerentes de proyecto y patrocinador, permitiendo interactuar entre diversos puntos de vista y llegando a un consenso sobre las decisiones que afectan al proyecto. Adicionalmente la técnica permiteidentificar riesgos del proyecto, compromisos a asumir para lograr el éxito del proyecto y la descripción base del mínimo producto viable. A continuación se expone la técnica de Inception Deck con base en lo expuesto por Ripenhausen (n.d.).

Paso previo: identificar quién está en el auditorio y las reglas del juego

Esta dinámica conlleva a identificar a los involucrados en la ejecución de la técnica determinando rol, influencia y poder de sus opiniones. El tiempo máximo asignado a esta actividad es en promedio de unos 30 minutos.

Para realizar esta dinámica se requiere que se agrupen por parejas o tríos a los participantes y cada uno se presente a su compañero, para ser posteriormente presentado por él. Al final de esta presentación cada uno debe generar uno o varios Post It Notes, para ser pegados en un tablero con la siguiente información:

- Nombre
- Datos de contacto.
- Qué le gusta a la persona.
- Qué le disgusta a la persona

Lo anterior ayuda a romper la tensión entre los stakeholders, equipo de desarrollo y gerentes de proyecto; a identificar claramente si las personas involucradas en la reunión son las correctas y, por último, a familiarizarse con la gente que hace parte del proyecto.

Después de realizar el paso anterior se presenta a los involucrados las reglas del juego, lo que se traduce como las normas a seguir en la reunión. Un ejemplo de esto es:

- Cada actividad tiene un tiempo máximo para ser realizada, esto se le denomina timebox o caja de tiempo.
- Para maximizar la productividad de la reunión las personas que están en esa reunión no deben atender otro asunto distinto.
- No se utilizarán celulares a menos que sea para una presentación de algún tipo de información relevante.
- Todos los participantes deben aportar su opinión en cada dinámica realizada.
- Todas las opiniones expresadas bajo el marco del respeto son válidas y tomadas en cuenta por el equipo de trabajo.

Primera dinámica: ¿Por qué estamos aquí?

Esta dinámica permite identificar el contexto del proyecto, que normalmente es expuesto por quien tiene la visión del producto. El tiempo sugerido puede variar entre 15 a 45 minutos, dependiendo del tamaño del proyecto.

Segunda dinámica: Elevator Pitch

Esta dinámica consiste en identificar rápidamente la esencia del proyecto. Por lo tanto, los involucrados en la dinámica buscan detectar factores relevantes a través de preguntas varias o lluvia de ideas. Un tiempo prudencial para este momento es entre 20 y 40 minutos. Algunas preguntas sugeridas son:

- ¿Para quién está dirigida la solución?
- ¿Qué beneficio trae la construcción de este proyecto?
- ¿Qué no gusta del producto y qué alternativas se plantean?
- ¿Qué factor diferenciador trae este proyecto?

Al final todas las respuestas son socializadas al auditorio como mecanismo de unificación de puntos de vista, buscando llegar al consenso.

Tercera dinámica: Diseñar la caja del producto

Se plantea al auditorio que se imagine que el proyecto ya está construido y está a la venta por cualquier medio (catálogo físico o virtual, stock, etc.), por lo que es vital ver reflejada la marca y el objetivo de negocio. En un tiempo no superior a 30 minutos se solicita a los participantes responder a estos tres simples tópicos:

- Nombre del producto
- Frase que lo identifica
- Beneficios del producto que lo diferencia de los demás.

Una vez cada participante expone sus respuestas, se realiza una votación para escoger por consenso el nombre, frase y beneficios.

Cuarta dinámica: generar la lista del no

Delimita el alcance del proyecto indicando qué no es el producto. En este punto surgen aspectos tan relevantes como: tiempos de espera no aceptados, lenguajes de programación, bases de datos y frameworks no aceptados por el cliente o el equipo; políticas de seguridad, restricciones de acceso a datos dependiendo del rol, etc. El tiempo sugerido para este punto es aproximadamente de entre 45 minutos y 90 minutos dependiendo del tamaño del proyecto.

Quinta dinámica: Conoce el vecindario

Este punto permite identificar como las personas que están en el auditorio interactúan entre sí, además de potenciar esas relaciones para lograr el éxito del proyecto. El objetivo de este punto es elaborar una matriz de interesados identificando rol o responsabilidad, principales intereses o expectativas, nombre, teléfono y correo electrónico para contacto. El tiempo de ejecución de este paso es 30 minutos.

Sexta dinámica: ¿Qué nos impide dormir en las noches?

En este punto, el objetivo es identificar de manera temprana los posibles riesgos del proyecto y las estrategias que el equipo debe seguir para impedir que sucedan o, en caso contrario, minimizar su impacto. El tiempo para este paso es de entre 30 minutos hasta 1 hora y 30 minutos. Los resultados el equipo los divide en dos partes: los riesgos (¿Qué nos impide dormir en las noches?) y las estrategias identificadas (¿Qué nos permite dormir en las noches?).

Séptima dinámica: Mostrar la solución

Es el eje central de la técnica. Aquí se realiza la definición funcional del sistema. Existen varias formas de abordar este punto:

- Darle una personalidad a la aplicación. Utilizado cuando la categoría del sistema a construir es muy específica y los aspectos de usabilidad altamente relevantes para su construcción. Aplicaciones como catálogos de productos, guías de apoyo a procedimientos, atención al público específico y video juegos son los que más utilizan este paso.
- Hagamos que fluya. Es un modelado del paso a paso de los procesos del sistema. No requiere el formalismo de UML o de BPMN. Se construye entre todos los participantes.
- Prototipado. Entre todos los participantes se construyen todas las pantallas del sistema, se detallan qué roles pueden visualizar que componentes de cada pantalla, la navegación, los mensajes de información, de alerta y de error, los correos electrónicos del sistema, las integraciones con otras plataformas, etc.

- Mapa de historias. Es la opción más utilizada y difundida de esta técnica. Consiste en una división modular del sistema, donde se detallan las funcionalidades asociadas a cada módulo. Esto permite tener una visión global funcional de todo lo requerido para delimitar el mínimo producto viable.
- El equipo puede seleccionar una o varias opciones de las anteriormente planteadas para detallar la solución a construir.
- Octava dinámica: ¿Qué es lo que hay que dar?
- El objetivo de este paso es que todas las personas involucradas tengan claro los compromisos asumir desde el inicio para el éxito del proyecto. El tiempo sugerido es de aproximadamente 20 minutos.

Novena dinámica: ¿Qué es lo que costará hacerlo?

En este punto el equipo realiza sus apreciaciones sobre que costará hacerlo no en términos económicos, sino en aspectos técnicos, de proceso y de riesgos. El tiempo sugerido para este paso es de 30 minutos.

Décima dinámica: Resumen o sumarización

Es el paso final. Aquí y como buena práctica se realiza un resumen de los resultados donde se detalla todo lo que se ha hecho, se expone el mínimo producto viable definido, se recopilan opiniones y se cierra la actividad. Este paso toma entre unos 5 a 15 minutos aproximadamente.

DISEÑO COLABORATIVO

En el proceso de desarrollo de software se evalúan todas las necesidades en términos de comportamiento del sistema frente a su uso constante y cómo esto afecta la percepción del usuario sobre su calidad. En otras palabras, aspectos como qué tanto se demora una búsqueda de información y la presentación de los resultados en pantalla, cuántos usuarios máximo pueden estar operando el sistema al mismo tiempo sin afectar la operación, la ventana de disponibilidad del aplicativo, la capacidad de interactuar con otros sistemas, etc., son aspectos que moldean la estructura base del producto de software a construir.

A partir de la técnica expuesta por Rasmusson (2006), se determinan una serie de factores que bajo las técnicas tradicionales no se detectan tan fácilmente y esto es trasmitido al equipo para modelar las guías que orientan la arquitectura. Por lo tanto, lo primero que aborda el equipo dentro del proceso de diseño colaborativo es, sin duda, cuáles son los atributos que el sistema espera satisfacer y cómo lo hará.

Para responder esta pregunta, se revisan los aspectos no funcionales del sistema identificados en el análisis de requisitos y luego se procede a complementar desde el aspecto funcional los que no fueron mencionados en el proceso de análisis de una manera tan clara. Esto recibe el nombre de atributos de calidad. Los atributos de calidad se dividen en observables y no observables. Esta categorización responde a si se ven dentro de la estructura del código fuente donde serán implementados o si se ven presentes en tiempo de ejecución.

Lo segundo que se identifica dentro de un taller de diseño colaborativo son los patrones de diseño que corresponden a los atributos de calidad mencionados. Los patrones de diseño satisfacen los atributos arquitecturales a través de soluciones que ya existen, que fueron comprobadas como exitosas y que pueden ser claramente determinados dentro de los componentes o de los comportamientos del sistema, como indica (Larman, 2004). Una vez ha sido identificado los atributos de calidad y los patrones que los satisfacen se modelan las vistas arquitecturales a través de UML para formalizar la exposición de la solución, según lo indicado por Roger S. Pressman (2010) para la documentación formal de la fase de diseño.

KANSEI Y SU TRABAJO

Mitsuo Nagamachi (1995) aborda una forma diferente de mirar la creación de productos de manera innovadora, planteando un símil desde la industria automotriz. Aquí resalta la importancia de entender los sentimientos a través de las sensaciones (Kansei en japonés) para lograr el diseño exacto que cumpla con lo requerido, esto se denomina diseño ergonómico.

En este sentido, Nagamachi (1995) plantea un modelo en el que las palabras se convierten en requisitos de diseño que despiertan la idea de que el producto es exactamente lo que el posible cliente quiere. Dado este modelo a través de un sistema de inteligencia artificial aplicada al trabajo de Nagamachi para el tipo de industria especifico, que recibe el nombre de KES o Kansei Engenieering System, se logran plasmar los sentimientos e imágenes del posible cliente en detalles de diseño. Por medio de este sistema y de técnicas como la lógica difusa, que permite normalizar los datos obtenidos de las distintas percepciones de un grupo estudio que ha alimentado la base de datos de conocimiento previamente, se hace posible identificar las respuestas psicológicas positivas y negativas de acuerdo a los prototipos expuestos para lograr un efecto de diseño que explote el lado positivo en la mayoría de los receptores del diseño.

APLICACIONES DE LA INGENIERÍA KANSEI

Desde su invención en el año 1995, la Ingeniería Kansei ha sido aplicada a diversos modelos de negocio. El primer caso mencionado es el del carro Miata de la compañía Mazda. Según Nagamachi (1995), para probar su teoría decidió contactar al director de Mazda en esos momentos, el señor Miyamoto, y plantearle la posibilidad de implementar en un diseño de un automóvil dicho proceso. El resultado de esta prueba fue un éxito total en las ventas de ese año, posicionando a Mazda como una de las empresas líderes de la industria automotriz, adicionando mediante este método una ventaja competitiva sobre sus rivales comerciales al entender mejor las necesidades emocionales del cliente final.

Además del caso mencionado, también existen otros casos en los que la aplicación de la ingeniera Kansei ha sido todo un éxito. De acuerdo con Ishihara, Matsubara, Nagamachi & Matsubara (2011), debido a las características urbanas de Japón que imposibilitan tener un jardín real en las residencias de algunos ciudadanos, se presentó la necesidad de implementar un sistema de realidad virtual que simulara un jardín y que por medio de él generara la sensación de estar inmerso en un jardín real. Por tanto, surge la necesidad de resolver esta problemática imitando todas las características de luz, sombra, sonidos, viento, movimiento de las ramas y demás características del ambiente real. La implementación de este sistema es muy compleja y no puede ser modelado con un proceso de elicitación tradicional, porque no hay forma de levantar requisitos dentro de un grupo focal de interesados, por eso como indica Ishihara et al. (2011), se decidió utilizar la implementación de la Ingeniería Kansei para desarrollar la aplicación hecha en el lenguaje de programación Java que pudiera cumplir con las expectativas de un ambiente real.

Otro ejemplo se da en las interfaces de usuario de las redes sociales, ya que son otro gran marco de trabajo de estas teorías al requerir implementar mejoras en la experiencia de usuario mediante la interacción de estos con las aplicaciones, permitiendo determinar con solo su manipulación qué mejoras se pueden aplicar.

INCEPTION DECK E INGENIERÍA KANSEI APLICADOS

Dentro de este trabajo los autores buscan refinar el proceso de desarrollo de software de Smart Campus mediante la inclusión del Inception Deck, descrito anteriormente, y la aplicación de los principios de la Ingeniería Kansei como apoyo a la construcción de aplicaciones institucionales.

Nagamachi (1995) sugiere la construcción de un sistema KES que será alimentado por las palabras

principales Kansei o palabras que determinan una sensación subjetiva positiva o negativa en el usuario de estudio. Pero dentro del trabajo mencionado, no se indica una forma clara de realizar la recolección de esta información desde el punto de vista de un ingeniero de software. Partiendo de que se cuenta con un KES ya implementado que interprete la información, resta identificar las palabras con las que se alimentaría dicho sistema.

La mejor forma de encontrar las palabras claves para la KES es identificar los atributos de calidad recurrentes dentro de varios proyectos en ejecución o ya finalizados del Smart Campus. Luego realizar la identificación de los patrones de diseño observables y no observables que resuelven el atributo expuesto. Finalmente realizar un mapeo que sugiere a partir de los atributos como palabra clave los patrones que deben ser usados. Los proyectos evaluados para esta propuesta son:

- APYSCC: Sistema para la administración de los acuerdos pedagógicos, planes de curso y monitoreo de resultados académicos de los cursos
- CTSANDROID: App móvil creada para la comunicación entre estudiantes y profesores (Foro, chat, compartir archivos, etc)
- PASE: Sistema para el plan de mejoramiento académico.
- PSUNIAJC: Sistema para la gestión de las operaciones de la oficina de proyección social.
- SCRUM: Sistema para la administración de proyectos de desarrollo de software basados en SCRUM.
- SIGEG: Sistema para las operaciones de la oficina de gestión de egresados.

Los resultados obtenidos de la identificación de atributos de calidad recurrentes entre los proyectos evaluados son expuestos en las tablas 1 y 2.

Tabla 1. Atributos de calidad observables más frecuentes en proyectos Smart Campus

Atributos de Calidad "Observables"	
Atributo de Calidad	**Descripción**
Confidencialidad	Es la ausencia de acceso no autorizado a la información. (Barbacci & Kazman, 1997)
Confiabilidad	Es la medida de la habilidad de un sistema para mantenerse operativo a lo largo del tiempo. (Barbacci & Kazman, 1997)
Desempeño	Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria. (IEEE 610.12).

Tabla 2. Atributos de calidad no observables más frecuentes en proyectos Smart Campus

Atributos de Calidad "No Observables"	
Atributo de Calidad	**Descripción**
Modificabilidad	Es la habilidad de realizar cambios futuros al sistema.(Folmer, Gurp, & Bosch, 2005)
Mantenibilidad	Capacidad de modificar el sistema de manera rápida y a bajo costo. (Folmer et al., 2005)
Escalabilidad	Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental (Pressman, 2010)
Disponibilidad	La capacidad de que el sistema esté total o parcialmente operativo al mismo tiempo que es requerido para manejar eficazmente las fallas que puedan afectar la disponibilidad del sistema.(Barbacci & Kazman, 1997)
Facilidad de integración	Capacidad de comunicación con otros sistemas. (Barbacci & Kazman, 1997)

Una vez identificados los atributos de calidad más comunes, se hace el emparejamiento con los patrones de diseño correspondientes utilizados en la solución. Los resultados son expuestos en la tabla 3 y 4.

Tabla 3. Patrones de diseño por atributos de calidad observables

Atributos de Calidad "Observables"	
Atributo de Calidad	**Patrón de diseño**
Confidencialidad	**Patrón Autorizador-Autenticador:** Funcionalidad de autenticación o login que permite acceder al sistema **Patrón RBA o Rol Base Access:** Se realizará la implementación de un mecanismo de autorización que provee acceso granular a la información por medio de roles y permisos de los usuarios.
Confiabilidad	**Patrón Multi-Capa o Multi-Layer:** Permite que el sistema pueda ser manipulado específicamente en determinados puntos sin afectar a los demás componentes.
Desempeño	**Patrón Inyección de dependencias:** Los objetos son creados en tiempo de ejecución haciendo uso dinámico de los recursos de memoria del dispositivo que ejecuta la aplicación.

Tabla 4. Patrones de diseño por atributos de calidad no observables

Atributos de Calidad "No Observables"	
Atributo de Calidad	**Patrón**
Modificabilidad	**Patrón GRASP:** Planteamiento modular del sistema mediante división por capas lógicas de software con responsabilidades definidas de cada capa y protocolos de comunicación entre ellas a través de interfaces. En aplicaciones complejas esto se da a través de servicios.
Mantenibilidad	**Patrón Multi-Capa o Multi-Layer:** Mediante este patrón se permite que las modificaciones a los componentes del sistema sean más sencillas, dado que van a puntos específicos del código sin afectar otros componentes.
Escalabilidad	**Patrón Multi-Capa o Multi-Layer:** Se implementa un modelo arquitectural multicapa donde la capa de lógica tiene la habilidad de propender a una fácil modificación para pasar de una arquitectura web a una SOA por medio de la exposición de servicios web. **Patrón ORM:** El uso de un ORM como Hibernate o elocuent permite un desarrollo independiente del DBMS utilizado. Por tal motivo no hay dependencia con el mismo y puede ser modificado en cualquier momento sin que esto afecte en un alto grado a la aplicación.
Disponibilidad	**Patrón Sincronización de estado:** Mediante la verificación de disponibilidad del estado de los servicios web, el sistema sabe cuándo puede o no operar. En caso de no haber conexión, el sistema indica mediante un mensaje al usuario que la conexión con los servicios de autorización de ingreso o de validación de datos que no dependen de la aplicación, sino que son componentes externos, no están disponibles.
Facilidad de integración	**Patrón Integración orientada a servicios:** Mediante el uso de los servicios web que contiene la aplicación se plantea la comunicación con diferentes fuentes de información.

Lo anterior indica una base de palabras claves que permiten sugerir una serie de patrones de diseño a partir de su identificación desde el análisis de requisitos, a modo general, para diversos tipos de proyecto.

Toma relevancia para cumplir este objetivo que desde la ejecución del Inception Deck y dentro de las dinámicas 4 o Lista del no y 7 Mostrar la solución, se identifique entonces si el sistema requiere ser mantenible, modificable, escalable, integrable, confiable, confidencial, disponible y en qué grado, y el rendimiento esperado. Esto sugiere que dentro de la técnica se maximice a través de quien la dirige este enfoque en los dos puntos mencionados para que el resultado sea óptimo y el sistema sugiera los patrones de manera adecuada.

Por último, los resultados deben ser evaluados por parte del equipo para recategorizar la relevancia e importancia de las decisiones arquitecturales tomadas y que el sistema sugiera los patrones de diseño de una manera cada vez más acertada.

CONCLUSIÓN

La propuesta actual permite apoyar el proceso de desarrollo de software haciendo más fuerte la coherencia que existe entre las etapas de análisis y diseño, manteniendo unificada la visión del producto a través de ambas. Cabe resaltar que el impacto de identificar las necesidades no funcionales o restricciones del sistema desde una etapa temprana genera unas condiciones favorables para la construcción del sistema que al equipo le facilitan la identificación del modelo arquitectural a seguir.

Adicionalmente la identificación de patrones de diseño a implementar como solución a unas necesidades específicas de comportamiento del sistema a desarrollar, puede sugerirle al equipo no solamente cómo diseñar su solución, sino también basado en la experiencia del mismo qué tecnologías utilizar que cuenten con dichas implementaciones en sus frameworks de desarrollo.

Por lo anterior, el impacto que tiene elicitar a través de Inception Deck las palabras claves para un modelo KES es alto. La ejecución de la técnica no solo facilita la comunicación entre todos los interesados, facilita también la recolección de metas y restricciones arquitecturales, incrementando las posibilidades de refinamiento del diseño de una manera rápida lo que repercute en el tiempo de toma de decisiones.

TRABAJOS FUTUROS

Dentro del trabajo actual no es posible evaluar en los distintos tipos de arquitectura cómo impacta esta sugerencia general de patrones y la posibilidad de refinar las sugerencias dependiendo de si el sistema a desarrollar es móvil, web, híbrido o un sistema de integración, por lo que se sugiere experimentar con más proyectos a fin de refinar los resultados obtenidos hasta el momento.

REFERENCIAS BIBLIOGRÁFICAS

Barbacci, M. R., & Kazman, R. (1997). Software architecture evaluation panel. In Proceedings Twenty-First Annual International Computer Software and Applications Conference (COMPSAC'97) (pp. 160–161). http://doi.org/10.1109/CMPSAC.1997.624780

Beck, K. (1999). Extreme Programming Explained: Embrace Change. XP Series. http://doi.org/10.1136/adc.2005.076794

Cohn, M. (2004). User Stories Applied: For Agile Software Development (Addison Wesley Signature Series). Writing (Vol. 1). http://doi.org/10.1017/CBO9781107415324.004

Engineers, I. of E. and E. (2008). Especificación de requisitos segun el estándar de IEEE 830, 27.

Folmer, E., Gurp, J. Van, & Bosch, J. (2005). Software Architecture Analysis of Usability. Architecture, 38–58. Retrieved from http://www.springerlink.com/index/87jl9wkck2gwyyd4.pdf

Ishihara, S., Matsubara, T., Nagamachi, M., & Matsubara, Y. (2011). Kansei analysis of the Japanese residential garden and development of a low-cost virtual reality Kansei engineering system for gardens. Advances in Human-Computer Interaction, 2011(1). http://doi.org/10.1155/2011/295074

Larman, C. (2004). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Analysis. http://doi.org/10.1016/j.nec.2006.05.008

León Duarte, J. A., Romero Dessens, L. F., & Olea Miranda, J. (2008). Customer subjective perception as a main issue in conceptual product desing: a methodological proposal. Ingeniare. Revista Chilena de Ingeniería, 16(2), 301–309. http://doi.org/10.4067/S0718-33052008000200004

Nagamachi, M. (1995). Kansei Engineering: A new ergonomic consumer-oriented technology for product development. International Journal of Industrial Ergonomics, 15, 3–11. http://doi.org/10.1016/0169-8141(94)00052-5

Pastrana, M., Ordóñez, H., Ordónez, A., Thom, L. H., & Merchan, L. (2018). Optimization of the Inception Deck Technique for Eliciting Requirements in SCRUM Through Business Process Models. Business Process Management Workshops, 4928(January), 649–655. http://doi.org/10.1007/978-3-319-74030-0_52 Pastrana, M., Ordóñez, H., Ordóñez, A., & Merchán, L. (2017). Requirements elicitation based on inception deck and business processes models in scrum. In Communications in Computer and Information Science (Vol. 735, pp. 327–339). http://doi.org/10.1007/978-3-319-66562-7_24

Pressman, R. S. (2010). Ingeniería del software. Un enfoque práctico. México: McGraw Hill. Recuperado de http://artemisa.unicauca.edu.co/~cardila/Libro_Pressman_7.pdf

Rasmusson, J. (2006). Agile project initiation techniques - The inception deck & boot camp. In Proceedings - AGILE Conference, 2006 (Vol. 2006, pp. 337–341). http://doi.org/10.1109/AGILE.2006.14

Rasmusson, J. (2010). The Agile Samurai–How Agile Masters Deliver Great Software. Pragmatic Bookshelf, año.

Ripenhausen, E. C. (n.d.). Inception.pdf.

http://doi.org/http://zeus.inf.ucv.cl/~bcrawford/Modelado%20UML/Ingenieria%20del%20Software%207ma.%20Ed.%20-%20Ian%20Sommerville.pdf

Sommerville, I. (2005). Ingeniería del software. España: Pearson.

The Standish Group. (2013). The Standish Group Report - Chaos, 16.

AUTORES

Manuel Alejandro Pastrana Pardo: Miembro IEEE desde 2011. Nació en Santiago de Cali el 17 de Enero de1987. Egresado de ingeniería de sistemas de la Universidad Santiago de Cali, Esp. De procesos para desarrollo de software y MSc. en ingeniería de software de la Universidad San Buenaventura. Con diversos estudios y certificaciones por parte del Carnegie Mellon, Seotin, SENA y fedesoft. Miembro gold del Microsoft Virtual Academy o MVA por sus siglas en inglés. Experiencia como ingeniero de desarrollo de software, arquitecto de software e integraciones y consultor en diversas empresas nacionales e internacionales. Docente tiempo completo UNIAJC perteneciente a la Facultad de Ingenierías. Actualmente hace parte del grupo de investigación Grintic. Correo: mapastrana@admon.uniajc.edu.co

Ana Milena Rojas Calero: Tecnóloga en Sistemas de la Universidad del Valle, ingeniera de sistemas de la Universidad Antonio Nariño, Especialista en Gerencia en Informática Organizacional y Maestría en Informática y Telecomunicaciones de la Universidad ICESI, Docente de Carrera Adscrita a la Facultad de Ingenierías de la Institución Universitaria Antonio José Camacho; amplia experiencia en Gerencia de TIC. Miembro activo del grupo de investigación GRINTIC en el área de ingeniería de software. Correo: amrojas@admon.uniajc.edu.co