UNIVERSIDAD DE EXTREMADURA

CENTRO UNIVERSITARIO DE MÉRIDA

MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN INGENIERÍA Y ARQUITECTURA

REFORMULACIÓN Y DESARROLLO DE UNA APLICACIÓN PARA PROCESOS DE EVALUACIÓN ONLINE DE GOBERNANZA DEL RIESGO

Páblisson Araújo Silva

Mérida, Noviembre, 2017

UNIVERSIDAD DE EXTREMADURA

CENTRO UNIVERSITARIO DE MÉRIDA

MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN INGENIERÍA Y ARQUITECTURA

REFORMULACIÓN Y DESARROLLO DE UNA APLICACIÓN PARA PROCESOS DE EVALUACIÓN ONLINE DE GOBERNANZA DEL RIESGO

Autor: **Páblisson Araújo Silva**
Fdo:

Director: **Prof. Dr. Juan Ángel Contreras**
Fdo:

Codirector: **Prof. Dr. Urbano Fra .Paleo**
Fdo:

Agradecimientos

Al profesor Juan Ángel Contreras por aceptar este reto y acompañarme.

A Urbano Fra. por proponerme este reto y contribuir a que se hiciera realidad y, juntamente con Belén, por la constante acogida en España.

Al profesor José Ramón Figueiras por la generosidad y la atención en los momentos que necesitaba.

A mis compañeros de clase, especialmente Isidoro Arroyo por la disponibilidad y apoyo siempre que fue necesario.

A Renan y Daniela por la cercanía y la fuerza para que se materializara este reto.

A mis amigos de Brasil y España, especialmente a los amigos de la DDJ en la persona de Don José Antonio Adrio, y a mis amigos del grupo Caminantes en la fe por la presencia constante.

A Josélia por ser siempre mi gran amiga, compañera y el amor que me fortalece.

A Dios por la gracia de la vida siempre renovada en su amor.

Resumen

En general, el proceso de desarrollo de software sufre constantes retrasos, incrementos en los presupuestos y costes, por lo que, con frecuencia, finaliza sin satisfacer plenamente al cliente. Durante muchos años, el enfoque principal de la ingeniería de software se centró en los grandes proyectos, los considerados críticos. La universalización del acceso a la tecnología ha ampliado las soluciones basadas en el software, a la vez que ha visto incrementada su complejidad. Todo esto, asociado al avance de las técnicas de ingeniería de software, ha ampliado las posibilidades de mejora de la calidad final del producto. El objetivo de este trabajo ha sido desarrollar Risonance 2.0 a partir de una versión anterior con importantes debilidades. Se trata de un Policy Support System (PSS) que permite evaluación participativa de la gobernanza del riesgo de desastres. El desarrollo ha permitido reflexionar sobre la calidad del software a partir del dinamismo y los problemas que surgen en los pequeños proyectos llevados a cabo por un solo desarrollador. Hemos optado por los métodos ágiles, tanto por su perspectiva, que son más bien orientaciones para la construcción del software fuertemente basado en valores, como por los métodos, dirigidos por planes, con la intención de conocer un poco más sobre el Rational Unified Process (RUP) y sus posibilidades. Esto ha permitido la construcción de un software de mayor calidad que el anterior.

Abstract

In general, software development suffers from constant delays, increases in budgets and costs, and often ends without fully satisfying the customer. For many years, the main focus of software engineering were large, critical projects. The universalization of access to technology has broadened the solutions based on software, while increasing its complexity. All this, associated to the advancement of software engineering techniques, has enlarged the options of improvement of the final quality of the product. The objective of this work has been to develop Risonance 2.0, an application whose earlier version had important weaknesses. It is a Policy Support System (PSS) that allows participatory evaluation of disaster risk governance. The process allowed discussing on the quality of software from the perspective of the dynamism and the problems that arise in small projects carried out by a single developer. We have chosen an agile method, both for its focus on a construction of software strongly based on values, and for a method directed by plans, with the intention of learning about the Rational Unified Process (RUP) and the opportunities. This has made possible the construction of a software with much higher quality than the previous version.

Índice

Índice de figuras

Índice de tablas

… mi propósito no es el de enseñar aquí el método que cada cual debe seguir para guiar acertadamente su razón, sino solamente el de mostrar de qué manera he tratado de guiar la mía.

Descartes (Discurso del método)

1. Introducción

En la actualidad el desarrollo de software ha evolucionado mucho en técnicas, estándares y paradigmas respecto de lo que es un software de calidad. Autores como Beck y Andres (2005), Pressman (2010), Sommerville (2011), Ambler (2002), Kroll y Kruchten (2003), Boehm y Turner (2004) y muchos otros han discutido extensamente sobre estas cuestiones en los últimos años y ellas nos llevan a comprender la complejidad del tema.

Ambler (2002) señala que la importancia del desarrollo de software está en el hecho de su propia construcción y por consiguiente, todo lo que está alrededor, es decir, la propia Ingeniería de Software. De esta forma este trabajo aborda los desafíos de Ingeniería de Software aplicado al contexto de desarrollo con un solo desarrollador puesto en práctica en la versión 2.0 de Risonance, en un entorno muy interactivo y cambiante, siguiendo las orientaciones propuestas por Kroll y Kruchten (2003) y Ambler (2002).

Risonance 2.0 sigue los principios propuesto por Fra.Paleo (2015) respecto a una evaluación participativa de la gobernanza del riesgo de desastres, basada en criterios, asincrónica y no presencial. Este modelo de evaluación propone un avance en la gestión de la gobernanza del riesgo por medio de un proceso de evaluación que forma parte del proceso de gobernanza y que va más allá de la habitual medición del rendimiento. Esa estructura configura un proceso de aprendizaje continuo de los actores sociales participantes y hace que el proceso de evaluación sea también formativo.

Este modelo de evaluación que sigue fuertemente los procesos democráticos y mantiene una comunicación horizontal de modo que cada ciudadano pueda expresarse sin control vertical, son potencializados por medio de la internet[1]. Un reflejo claro es la capacidad que tiene cualquier ciudadano en las redes sociales para manifestar su opinión sobre las políticas públicas.

Así, el software es un instrumento que facilita la aplicación de un sistema de evaluación basado en procesos democráticos que tiene lugar a través de la evaluación

[1] Ver más detalles sobre un análisis sociológico de la influencia de internet en una sociedad en red en Castells (2014).

participativa de la gobernanza del riesgo. En este contexto, Risonance 2.0 sistematiza y amplía las opciones existentes en la versión 1.0, permitiendo conectarse con varios medios aprovechando sus capacidades, favoreciendo la construcción participativa de la gobernanza del riesgo, mediante la evaluación.

1.1. Justificación

La red a la que estamos conectados por medio de diversos dispositivos, teléfonos móviles, tabletas y PCs, amplía las relaciones sociales y la posibilidad de realizar procesos que anteriormente apenas era posible realizar sin la red. Internet posibilita la interacción colaborativa y en tiempo real, al mismo tiempo que permite que más personas estén conectadas realizando un mismo proceso sin necesidad de hacer grandes cambios en su rutina (Castells, 2014). Además de permitir conectarse a múltiples fuentes al mismo tiempo, es posible recoger, almacenar y tratar distintos formatos, como video, imágenes, textos, y datos abiertos, mientras se sigue un proceso de evaluación (Fra.Paleo, 2015).

De este modo, una aplicación web que permita este proceso, potenciaría la transparencia de las políticas y ampliaría la capacidad de conexión con las nuevas tecnologías, principalmente desde la perspectiva de la recogida y tratamiento de la información, añadiendo más valor a la información evaluada.

Para la evaluación participativa de la gobernanza del riesgo de desastres, Fra.Paleo (2015) presenta las bases para evaluar con múltiples objetivos, que van más allá de la medición del rendimiento. Al hacer una evaluación, además de generar información sobre los procesos de gobernanza de un territorio permite identificar el nivel de desarrollo de las políticas públicas y promover la participación pública. De esta forma la constante evaluación es al mismo tiempo un proceso de aprendizaje y un proceso de construcción de políticas públicas.

Un proceso de evaluación basado en el consenso, en el que están involucrados distintos actores sociales, genera más transparencia y refuerza la democracia (Fra.Paleo, 2015). En un mundo conectado en red el uso de tecnologías puede ampliar y fortalecer estos conceptos. Así se programó la aplicación Risonance 1.0 para realizar la evaluación, que antes

solo era posible de forma presencial y que ahora se puede hacer conectado en red, desde cualquier punto geográfico.

Por otro lado, aunque Risonance 1.0 tenía una gran potencialidad, la presencia de muchos errores graves en su codificación, como la mezcla entre códigos PHP, HTML, Javascript y CSS imposibilitaron su uso. Además, no tenía una arquitectura clara, el código estaba disperso sin ninguna estructura de orientación a objetos y, por lo tanto, basado en funciones, y con graves problemas de seguridad. Habría que decir además, que muchas funcionalidades implementadas no funcionaban correctamente, o presentaban alguna incompatibilidad con las nuevas versiones de PHP y por último, mantenía un diseño anticuado.

1.2. Objetivo

1.2.1. Objetivo general

El objetivo general del trabajo es desarrollar la versión 2.0 de Risonance como software online de evaluación participativa de la gobernanza del riesgo de desastres, para que pueda contribuir de manera efectiva al proceso de toma de decisiones que solucionen los problemas encontrados en la versión inicial, además de incrementar las capacidades del mismo con nuevos elementos, métodos y funcionalidades, por medio de la integración de Método Ágil (MA) y Método Dirigido por Planes (MDP), haciendo un ajuste al Rational Unified Process (RUP).

1.2.2. Objetivos específicos

- Utilizar métodos interactivos de procesos de desarrollo, como los MA y MDP, con el objetivo de conocer más sobre RUP y sus posibilidades.
- Reestructurar el software utilizando UML como un lenguaje para modelar sistemas orientados a objetos.
- Reestructurar la base de datos relacional.

- Codificar el software, trabajando con un modelo de programación orientada a objetos, manteniendo los estándares de codificación propuesto por PHP y el framework de desarrollo, y adoptar técnicas que contribuyan a garantizar la seguridad del sistema.

- Evaluar la calidad del software a partir de los problemas de los pequeños proyectos desarrollados por pequeños equipos o por un solo programador.

1.3. Estructura de la memoria

En el capítulo 1, se presenta una breve introducción bien como la justificación del trabajo y los objetivos. En el capítulo 2, se hace una descripción del estado de la cuestión en la ingeniería de software, prestando especial atención a la contribución de los métodos de desarrollo más interactivos. En este capítulo se examinan los modelos de proceso de desarrollo de software, el lenguaje de modelado unificado (*Unified Modeling Language* o UML), además de los patrones de diseño de desarrollo de software dentro de la estructura del lenguaje de desarrollo PHP. Al finalizar se hace una descripción del proceso de evaluación participativa de la gobernanza de riesgo haciendo referencia a conceptos generales como riesgo, desastre y gobernanza del riesgo y, en particular, la evaluación participativa de la gobernanza del riesgo. También se revisan las características de la versión 1.0 de Risonance y, particularmente, sus debilidades para identificar las opciones para la versión 2.0.

En el capítulo 3 se presenta la sistematización de la metodología utilizada para el desarrollo de Risonance 2.0, al final se expone en un quadro metodológico con la sistematización de las acciones emprendidas para el cumplimento de los objetivos.

En el capítulo 4 se aborda el uso de los conceptos de ingeniería de software en el desarrollo con alta interactividad con un solo desarrollador, utilizando Risonance 2.0 como objeto de estudio. Así se trata la integración de los MA y de los MDP, reflejada en la documentación por medio de un modelado orientado a objetos y, finalmente, el empleo de las recomendaciones en el desarrollo orientado a objetos en PHP.

En el capítulo 5 se hace un análisis de lo propuesto y de lo que hemos alcanzado en términos de calidad de software. Finalmente, en el capítulo 6 se habla de las posibilidades

que ofrece Risonance 2.0 en relación a las posibles aplicaciones, además del uso de técnicas de inteligencia artificial para ampliar las capacidades y funciones del software.

2. Marco teórico

Al avanzar en el estudio de la ingeniería de software y la búsqueda de la mejora en la calidad de software, se produjo un incremento en la complejidad (Sommerville, 2011). La medición de esa complejidad, por tener múltiples enfoque, es un tema amplio y sin un consenso en su definición, es considerada en ocasiones como "el imposible Santo Grial" (Pressman, 2010, p.527), y muchas veces su monitoreo dentro del proyecto depende de la experiencia del ingeniero de software.

Las primeras discusiones sobre calidad tuvieron lugar a partir del momento considerado la 'crisis del software', que llevó a la organización de dos grandes conferencias[2] , una en 1968, en Garmisch (Alemania), y otra en 1969, en Bruselas (Bélgica). Según Sommerville (2011), las discusiones influyeron para que en los años 70 y 80 se desarrollasen técnicas de ingeniería de software como la programación estructurada, el encubrimiento de información y el desarrollo orientado a objetos. Además de que, "se perfeccionaron herramientas y notaciones estándar que ahora se usan de manera extensa"[3] (Sommerville, 2011, p.5).

A medida que fue creciendo la complejidad del software, se empezaron a buscar respuestas que pudiesen satisfacer las necesidades específicas de su desarrollo, buscando hacer una integración en el ambiente existente, por medio de nuevas técnicas, métodos y software que pudiesen dar soporte al proceso de construcción de software (Sommerville, 2011). Los métodos y procesos que surgieron y que, como señalan Boehm y Turner (2004, p.10), "fitted well with the branch of academic computer science and software engineering

[2] Estas conferencias fueron organizadas por la *Organización del Tratado del Atlántico Norte* (OTAN) para analizar los problemas del desarrollo de software. "En esa época había grandes sistemas de software que estaban rezagados, que no ofrecían la funcionalidad que requerían los usuarios, que costaban más de lo esperado y que no eran fiables" (SOMMERVILLE, 2011, p. iii). Los documentos de los eventos pueden ser consultados en el siguiente enlace: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/index.html. Acceso en: 20/09/2016.

[3] **Herramientas** son comprendidas como el soporte necesario para el desarrollo de un software. Como ejemplo de herramientas tenemos: Un modelo de proceso, las herramientas CASE o hasta un sistema controlador de versión. **Notaciones estándares** se constituyen como patrones internacionales para orientar la creación o utilización de lenguajes, técnicas o modelos, como es el caso del UML (Unified Modeling language). Más información en https://es.wikipedia.org/wiki/Herramienta_CASE y en https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/History/ . Accedido en 12/08/2017.

that views software development as a process of formal mathematical specification and verification", contribuyendo a ampliar la discusión sobre el software de calidad.

2.1. Calidad en el desarrollo de software

Al respecto de los múltiples conceptos e interpretaciones sobre la calidad, Reeves y Bednar (1994, p.419) señalan que, "Regardless of the time period or context in which quality is examined, the concept has had multiple and often muddled definitions and has been used to describe a wide variety of phenomena". Así que, cuando pensamos en la calidad del software es posible que encontremos también un gran número de definiciones y perspectivas. Pressman (2010, p. 340) revisa la definición de calidad a partir de múltiples puntos de vista, como el que habla de un "proceso eficaz de software que se aplica de manera que crea un producto útil que proporciona valor medible a quienes lo producen y a quienes lo utilizan", y esta indica una visión de un MDP.

Estas definiciones, que algunas veces pueden ser contradictorias, nos permiten comprender el grado de complejidad del desarrollo de software y, por lo tanto, la propia calidad del mismo. Sommerville (2011) señala que, a pesar de que se habla mucho de la calidad en el desarrollo de software, raras veces se aplican procesos y técnicas que aseguren la misma. Esa perspectiva podría llevarnos a comprender que seguir un proceso de desarrollo de software de forma rígida, y solo por medio de él, nos garantiza la obtención de un software de alta calidad, dado que se tiene un alto nivel de control, supervisión y previsibilidad (Sommerville, 2011).

Sin embargo la dificultad del concepto de calidad y, por tanto, la calidad del software, viene del hecho de ser una noción subjetiva. No obstante, respecto del software, aunque no sea posible llegar a una única y clara definición de lo que es un software de calidad, sabemos cuándo nos encontramos con un software de mala calidad (Sommerville, 2011). Sea por el alto grado de incumplimiento con las especificaciones preestablecidas, la falta de comprensión de las características deseadas por el cliente y, por lo tanto, la ejecución de un desarrollo erróneo, además de la separación del cliente del proceso de desarrollo (Ambler, 2002).

En la calidad desde un enfoque ágil, las personas están por encima del proceso y, así, "they must have skill. The team must have skill in analysis, modeling, design, programming, testing - all of the things that are the basis of good software" (Ambler, 2002, p.xii). El modelado ágil tiene valores y principios en lugar de seguir procesos, lo que no excluye los procesos y su documentación, sino que pone el foco en la calidad mediante las periódicas entregas del software con funcionalidades hechas (Ambler, 2002).

Jeffries (2015, p.73) señala que, "for the best quality, smoothest progress, and greatest predictability, this is the best known way to work. Testing and refactoring are critical tools in feature-by-feature development. Don't leave home without them". Aunque sean importantes todas las técnicas y procesos, es necesario tener en cuenta que, para el cliente lo más importante es que el software funcione, ya que ver algo funcionando es importante para que el cliente pueda participar del proceso y valorar el trabajo mientras está siendo desarrollado (Ambler, 2002).

A continuación se describen los métodos de desarrollo a partir de dos perspectivas, MDP y MA, que interpretan la calidad en el desarrollo desde una forma de trabajo distinta (Pressman, 2010)

2.2. Método dirigido por planes (MDP)

Boehm y Turner (2004) caracterizan los métodos dirigidos por planes por mantener la rigidez del proceso, de modo que el software evoluciona mientras se avanza en las etapas del proceso hasta que los requisitos sean reflejados en el código. Según los autores, estos métodos proporcionan más control y previsibilidad al proceso, además de hacer que el producto sea repetible y comparable. Por ello estos métodos también son conocidos como modelos de proceso prescriptivos o modelos de procesos. Pressman (2010) señala que estos modelos son representaciones abstractas, y no tienen por objetivo ser un estándar que funcione para todos los proyectos de software en todas las situaciones sino, más bien, servir de orientación para la estructuración de su propio proceso.

Los MDP, proponen que manteniendo la rigidez del proceso y lo documentando a cada paso, se pueda obtener el mismo producto con la misma calidad. La claridad de las

actividades ejecutadas, además de una mejor gestión, control y previsibilidad en el proceso, posibilita un cambio rápido de personal cuando sea necesario (Sommerville, 2011). De acuerdo con Pressman (2010, p.33), aunque la verificación del cumplimiento de las funcionalidades ejecutadas de forma tardía pueden hacer el trabajo caótico, "la historia indica que estos modelos tradicionales han dado cierta estructura útil al trabajo de Ingeniería de Software y que constituyen un mapa razonablemente eficaz para los equipos de software". El primero de estos modelos prescriptivos, conocido por ciclo de vida clásico, es el modelo en cascada y fue propuesto por Royce en 1970 (Sommerville, 2011).

Hay una gran cantidad de modelos de procesos, por lo que revisaremos algunos relevantes para la discusión sobre el entorno de desarrollo con pequeños equipos. Aunque no funcionen como un estándar global, en general los modelos de procesos existentes poseen características que son actividades fundamentales que pueden ser encontradas en todos los modelos de procesos. Al tratar este tema, Sommerville (2011) señala que hay por lo menos cuatro actividades fundamentales: especificación del software, diseño e implementación del software, validación del software y evolución del software. De otro modo, Pressman (2010) propone una perspectiva a partir de cinco actividades fundamentales: comunicación, planificación, modelado, construcción y despliegue. En las dos propuestas mencionadas, y en muchas otras en la literatura, estas actividades estructurales tienen sus complejidades y subactividades individuales además de definir reglas y responsabilidades.

Así, para Pressman (2010, p.29) un modelo de proceso[4] o un patrón de un proceso "describe un problema relacionado con el proceso que se encuentra durante el trabajo de ingeniería de software, identifica el ambiente en el que surge el problema y sugiere una o más soluciones para el mismo". Los MDP que hemos seleccionado para discutir más ampliamente son el modelo de cascada, modelos de proceso incremental, modelo de proceso evolutivo, modelo espiral. Además trataremos el *Rational Unified Process* (RUP) que es un modelo de proceso que va más allá de las propuestas de los MDP y que funciona al mismo tiempo como un modelo, un proceso y un framework.

[4] Para más detalles ver Pressman (2010), especialmente el capítulo 2, donde el autor habla sobre otros modelos de procesos además de abordar particularidades de cada uno.

2.2.1. Modelo de proceso en cascada

El modelo de proceso en cascada nació de la necesidad original de resolver los problemas en el desarrollo en los primeros años de la ingeniería de software, con el fin de encontrar una solución a la falta de un patrón que pudiese aumentar la previsibilidad y reducir los errores de codificación (Pressman, 2010).

Sin embargo, aunque el modelo en cascada (Figura 1) ha contribuido a la mejora en el desarrollo de software, tiene muchos problemas derivados de su rigidez. El hecho de que cada fase tenga una fuerte dependencia de la anterior, al producirse un flujo de trabajo lineal, hace que no sea posible seguir adelante sin antes cumplir lo que fue propuesto. Eso generaba mucha ociosidad en el equipo, además de que el coste de descubrir o cambiar un requerimiento en medio del proceso es mucho más caro y se cae en el riesgo de identificar errores graves cuando el software está ya en una etapa muy avanzada (Pressman, 2010).

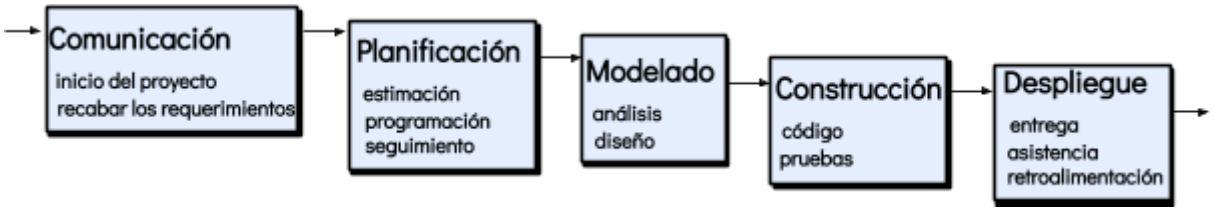


Figura 1: Modelo de proceso en cascada. Fuente: Pressman, 2010.

2.2.2. Modelo de desarrollo incremental

En el modelo de desarrollo incremental (Figura 2) el software es desarrollado por medio de secuencias en las cuales se hacen los incrementos. El modelo combina los flujos de trabajo lineal y paralelo, de modo que se van incrementando las funcionalidades mientras se avanza en el proceso, y se genera un producto simplificado. El desarrollo incremental, de acuerdo con Sommerville (2011, p.32), "se basa en la idea de diseñar una implementación inicial, exponer esta al comentario del usuario, y luego desarrollarla en sus diversas versiones hasta producir un sistema adecuado".

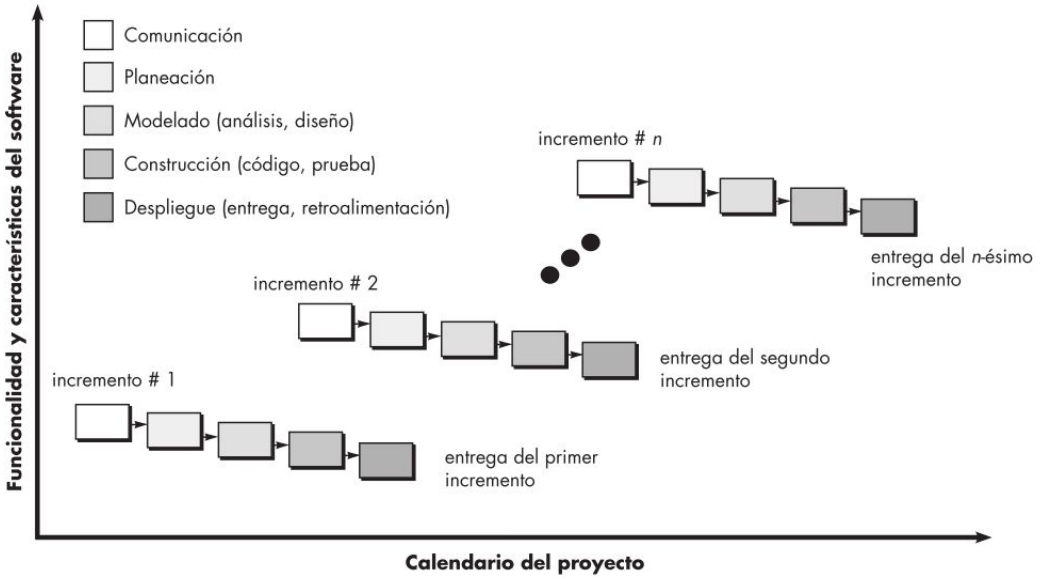


Figura 2: El modelo de desarrollo incremental. Fuente: Pressman, 2010.

Este tipo de modelo es útil principalmente desde la perspectiva del cliente, ya que se pueden valorar partes del producto que se está haciendo, corregir errores, proponer algún cambio y comprender mejor los requisitos que se solicitan desde los primeros incrementos (Pressman, 2010). La debilidad de este modelo, es que para grandes proyectos se torna más difícil el control y coordinación de un entorno con más burocracia. Eso se debe a que hay muchos equipos distintos desarrollando partes de un mismo sistema. (Sommerville, 2011).

2.2.3. El paradigma de prototipo

Este paradigma (Figura 3), que es un modelo de proceso evolutivo[5], propone crear prototipos con el fin de ayudar en el descubrimiento y documentación de los requisitos, en validar alguna funcionalidad o testear algún algoritmo. Es un paradigma utilizado en combinación con otros modelos, como por ejemplo, el modelo en cascada. La debilidad de este modelo es que el desarrollador, al crear un prototipo, busca ejecutar algo rápido con el fin de hacer un test, dejando muchas veces la calidad en un segundo plano. Muchas veces este paradigma no es entendido de la misma manera por el cliente que, creyendo que el software ya está muy adelantado solicita al programador añadir más funciones en el prototipo sin conocer el propósito específico del mismo (Pressman, 2010).

[5] "Los modelos evolutivos son iterativos. Se caracterizan por la manera en la que permiten desarrollar versiones cada vez más completas del software" (Pressman, 2010, p.36).

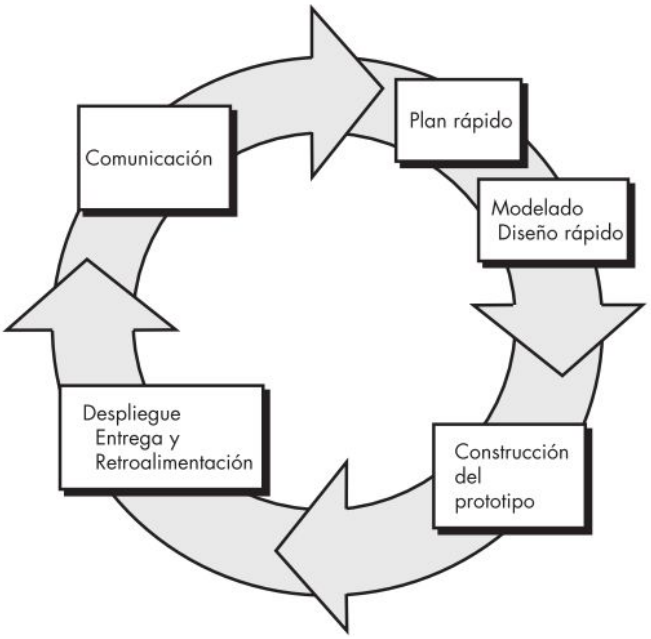


Figura 3: El paradigma de prototipos . Fuente: Pressman, 2010.

2.2.4. Modelo en espiral

El modelo en espiral (Figura 4), que también es un modelo evolutivo, fue propuesto por Boehm en 1988 y está orientado a los riesgos adaptando la interactividad y agilidad del modelo de creación de prototipos con la gestión y control del modelo en cascada (Pressman, 2010). Boehm (2001) define el modelo de desarrollo en espiral como un generador de modelos de proceso, que es orientado por los riesgos. Aunque otros modelos de procesos poseen análisis de riesgos, este modelo hace explícita su fundamentación de forma orientativa del trabajo. Así, tiene un enfoque cíclico que propone un crecimiento gradual de pequeños incrementos en el software mientras se va reduciendo el nivel del riesgo, además posee los hitos que aseguran el compromiso con las partes interesadas.

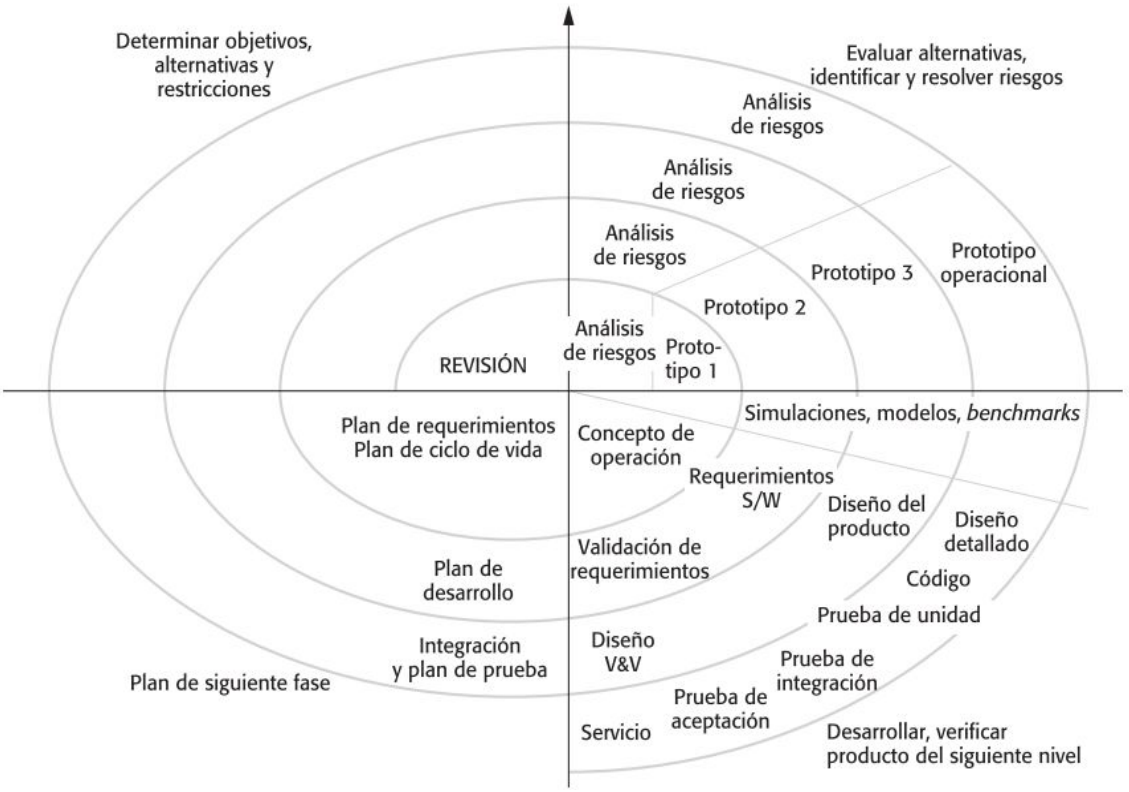


Figura 4: Modelo en espiral. Fuente: Sommerville, 2011.

2.2.5. El Proceso Unificado Racional o Rational Unified Process (RUP)

RUP fue desarrollado primero por Rational Software, aunque hoy pertenece a IBM[6] (Arlow y Neustadt, 2006). El ciclo de vida de RUP tiene 4 fases (Figura 5) con múltiples interacciones y es un modelo de proceso híbrido que contiene buenas características de los modelos evolutivos (Sommerville, 2011). Kruchten (2004) define RUP como un enfoque de desarrollo de software, un proceso de ingeniería de software bien definido y bien estructurado y, finalmente, como un producto, lo que le aporta un marco de proceso personalizable que le permite adaptarse al desarrollo en pequeños y grandes proyectos.

[6]Arlow y Neustadt (2006) considera RUP como una versión comercial y ampliada del Proceso Unificado *(Unified Process* o UP) pero con muchas similitudes.

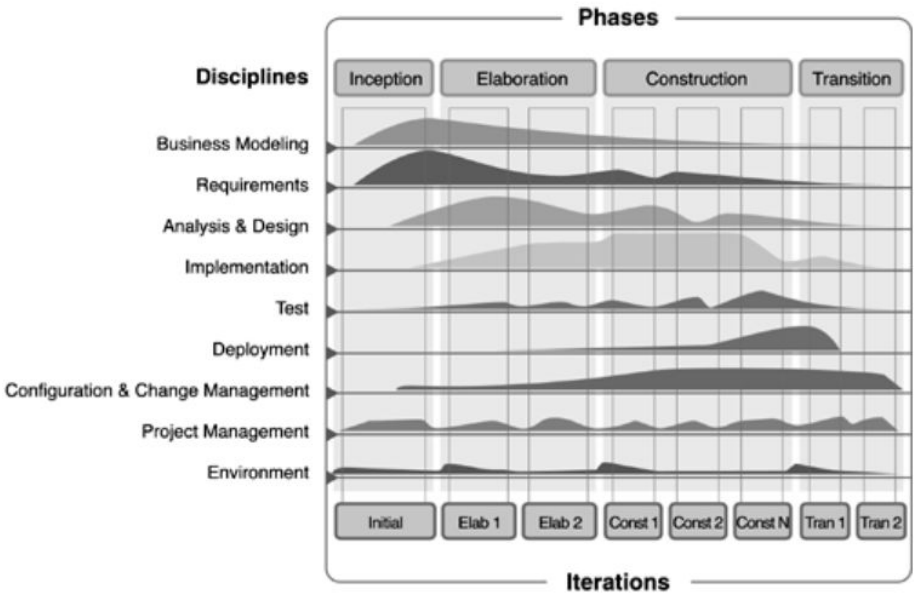


Figura 5: Dimensiones de RUP. Fuente: Kroll y Kruchten, 2003.

Con respecto al enfoque de desarrollo, RUP ofrece una perspectiva interactiva, basada en los siguientes principios: a) afrontar los riesgos mayores temprano y continuamente o ellos te atacan; b) asegurarse de entregar valor al cliente; c) mantenerse centrado en el software ejecutable; d) acomodar el cambio temprano en el proyecto; e) generar un ejecutable desde el principio, por medio de una organizada arquitectura; f) construir su sistema con componentes; g) trabajar juntos como un equipo; h) hacer de la calidad un modo de vida, no una idea posterior (Kroll y Kruchten, 2003).

Desde una perspectiva de un proceso bien definido y estructurado, el RUP ofrece una estructura basada en dos ejes, fases y disciplinas. En esa estructura global es donde se define la relación de los elementos, y por lo tanto, define *quién*, *qué*, *cómo* y *cuándo* dentro del proceso (Kroll y Kruchten, 2003).

Por último, RUP "is also a process product that provides you with a customizable process framework for software engineering. The RUP product supports process customization and authoring, and a wide variety of processes, or Process Configuration" (Kroll y Kruchten, 2003, p.3) y eso por poseer una estructura fuertemente basada en las buenas prácticas de desarrollo de software .

2.3. Métodos ágiles (MA)

A diferencia del MDP, los MA se basan en cuatro declaraciones de valores fundamentales, orientando sus relaciones como filosofía de trabajo[7].

El primero define que los individuos e interacciones están por encima de los procesos y herramientas. No se trata de descartar el proceso, sino, más bien de dejarlo como un soporte para que las personas ejecuten su trabajo. Al contrario que los MDP, que sostienen que cualquier desarrollador utilizando un buen proceso sería capaz de desarrollar un software de calidad, esta declaración propone que la calidad del software dependerá mucho de las características del personal involucrado en el desarrollo. Lo que señala que un equipo mejor preparado dará respuestas mejores al problema y será más capaz de construir un software de calidad (Ambler, 2002).

El segundo valor indica que es mucho más importante una parte del software funcionando que la generación de documentación indicando el seguimiento de un modelo muy bien elaborado. Si el proceso utilizado para construir el software no ayuda a desarrollar un software más rápido y con más calidad, debe descartarse (Ambler, 2002).

El tercero señala que debe haber colaboración con el cliente más que una negociación contractual, de modo que el cliente forma parte del proceso de desarrollo. Si el cliente se reúne con el responsable del desarrollo sólo al principio y al final del proyecto, puede ser que el mismo fracase (Ambler, 2002).

El cuarto valor tiene como premisa "la respuesta ante el cambio más que seguir un plan" (Manifiesto, 2001, s/p). Si se ha desarrollado una funcionalidad de acuerdo con lo especificado, pero el cliente se ha dado cuenta de que ha especificado mal, es mejor negociar el cambio y proponer una nueva solución, que entregar un software que haya cumplido exactamente con lo que fue propuesto en el plan inicial pero que no va atender al cliente tal cual podría hacerlo (Ambler, 2002). Además de estas declaraciones de valores descritas en el

[7] El Manifiesto Ágil fue propuesto en el año 2001 por un grupo de desarrolladores que estaban descontentos con las propuestas rigurosas de los modelos de desarrollo propuestos en aquel momento. Este grupo se autodenominó como *Agile Alliance*. Propusieron lo que hoy es conocido como *Manifesto Agile* que es el documento firmado por este grupo. Se puede consultar en: http://agilemanifesto.org/. Acceso en: 26/12/2016

Manifiesto Ágil, también propusieron doce principios para orientar el trabajo (Manifiesto, 2001).

Ambler (2002) analiza los valores de las metodologías ágiles considerando la comunicación, la simplicidad, la retroalimentación, la valentía y la humildad y justifica el uso de la humildad afirmando que frecuentemente cuando los desarrolladores están recolectando los requisitos creen saber más que el cliente, señalando que la fuente de los requisitos es el cliente.

Finalmente, los métodos ágiles tienen un enfoque en las personas al contrario que los métodos dirigidos por planes que mantienen el enfoque en el proceso y en la creación de una extensa documentación (Ambler, 2002). De esta forma, los MA se adaptan mejor a los modelos que proponen interacciones cortas de trabajo, modelando y diseñando solo lo que se hará hasta la siguiente entrega, entregando siempre algo que esté funcionando. Y eso justifica la necesidad constante del cliente en el proceso que contribuye para la validación de las entregas, la constante mejora de la comprensión del problema y la solución.

2.3.1. Extreme Programing (XP)

Beck y Andres (2005) indican que trabajar con XP exige un cambio personal en los hábitos. Al proponer valores como la comunicación, la simplicidad, la retroalimentación, el respeto y la valentía, acepta que dentro del negocio hay una relación entre las personas que necesita ser fortalecida para el cumplimento de los objetivos del proyecto. Así, centrándose en el rápido cambio de requisitos en cualquier fase del desarrollo de software, XP es una las metodologías ágiles más conocidas y más utilizadas.

XP está basada en el desarrollo por pares, con dos desarrolladores trabajando al mismo tiempo en un mismo ordenador codificando juntos, buscando minimizar errores, manteniendo un diseño simple, la refactorización[8], desarrollando un conjunto mínimo de funcionalidades y acercándose al cliente mediante constantes entregas. De acuerdo con Boehm y Turner (2003) una de las grandes ventajas de XP es la capacidad de escalar un problema con un equipo de hasta veinte personas, ya que todavía no hay datos que

[8] El *refactoring*, o refactorización, es una técnica de mejora u optimización del código sin cambiar su estructura o modelo. Disponible en https://martinfowler.com/books/refactoring.html acceso en 14/07/2017.

demuestren que ha funcionado con equipos más grandes. Además, Sommerville (2011) apunta que, en algunos casos, encontrar soluciones alternativas para los problemas generan con frecuencia códigos duplicados y partes del código reutilizados de forma inadecuada, lo que para Beck y Andres (2005), se resuelven utilizando la refactorización.

A pesar de que XP no trabaja con la misma filosofía de los MDP, XP posee un 'proyecto de ciclo de vida' que se inicia con la fase de exploración en la que, clientes y desarrolladores (Figura 6) trabajan en la captura de las historias de usuario (*user stories*) que tiene objetivos similares a la captura de requisitos en los MDP (Beck y Andres, 2005).

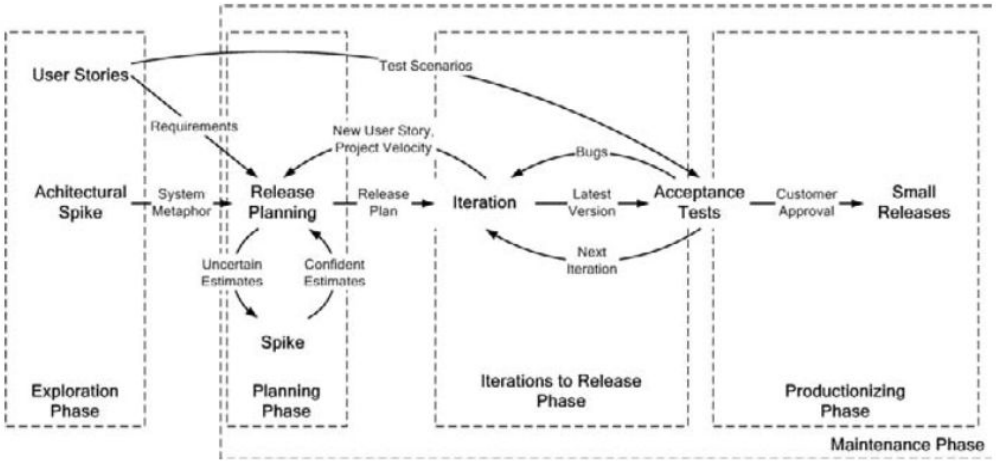


Figura 6: Proyecto de ciclo de vida XP. Fuente: Ambler, 2002.

2.3.2. Scrum

Scrum[9] es un sencillo modelo de desarrollo que se caracteriza por ser un proceso empírico, basado en cambios continuos, con ciclos cortos de desarrollo variando de una semana hasta un mes. En algunos casos se empieza con las historias de usuario a partir de las cuales se generan los '*product backlog*' y los *'sprints*'. Es posible que en el plan del '*product backlog*' sea donde se planea lo que hará el equipo. Probablemente la duración de las actividades exceda el ciclo corto propuesto en los '*sprints'*, por lo que en esta fase se detallan las actividades y se define quién desarrollará, qué se desarrollará y cuándo (Figura 7). Además se hacen reuniones diarias en las que se controla todo lo que está ocurriendo en el seguimiento del proyecto por medio de tres cuestiones: ¿qué hiciste desde la última reunión?;

[9] Ver en: http://www.scrumguides.org/history.html. Acceso en: 10/01/2017

¿has tenido algún tipo de impedimento? y ¿qué planeas hacer mientras llega la siguiente reunión del equipo? (Rubin, 2012).

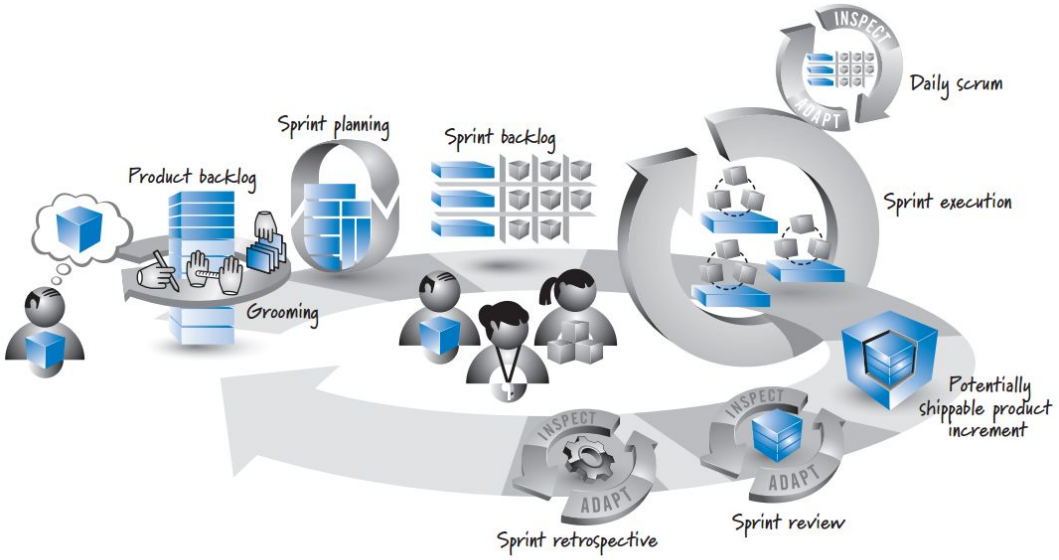


Figura 7: Scrum Framework.Fuente: Rubin, 2012.

La metodología Scrum es muy útil y se suele integrar con MDP ya que, como señala Sommerville (2011, p.72) mantiene el enfoque "en la administración iterativa del desarrollo, y no en enfoques técnicos para la Ingeniería de Software ágil".

2.4. Valoración de los métodos

Hay que subrayar que es muy importante que, al tratar de la calidad, se hable también de todo lo que hace y no hace el software. Además del diseño del software, de la gestión de los requisitos y del proceso, debemos tener en cuenta el problema a ser resuelto, el tamaño del proyecto y, sobre todo, el equipo involucrado (Sommerville, 2011).

Es muy probable que en un proyecto crítico en el que, por ejemplo, esté en riesgo la vida de las personas, se debe mantener mucho más el control y la previsibilidad, mediante un elevado grado de rigidez en el proceso, por lo que es necesario un enfoque rígido en el plan inicial. Por el contrario, un proyecto de una app que envía mensajes, por ejemplo, debería tener mucha más flexibilidad, permitiendo cambios durante el proceso y seguir una evolución más ágil. Lo que no impide que en los dos casos el software haya cumplido las exigencias de calidad, y finalizado con éxito (Kroll y Kruchten, 2003).

Kroll y Kruchten (2003) proponen un marco de comparación entre los procesos o proyectos de desarrollo de programas existentes basándose en un mapa de procesos con dos ejes (Figura 8), en el que el eje vertical mide el riesgo y la interacción y el eje horizontal mide el nivel de documentación y trazabilidad. Este marco de comparación, por medio del ajuste, tiene por finalidad proponer al desarrollo de software la documentación y formalización adecuada.

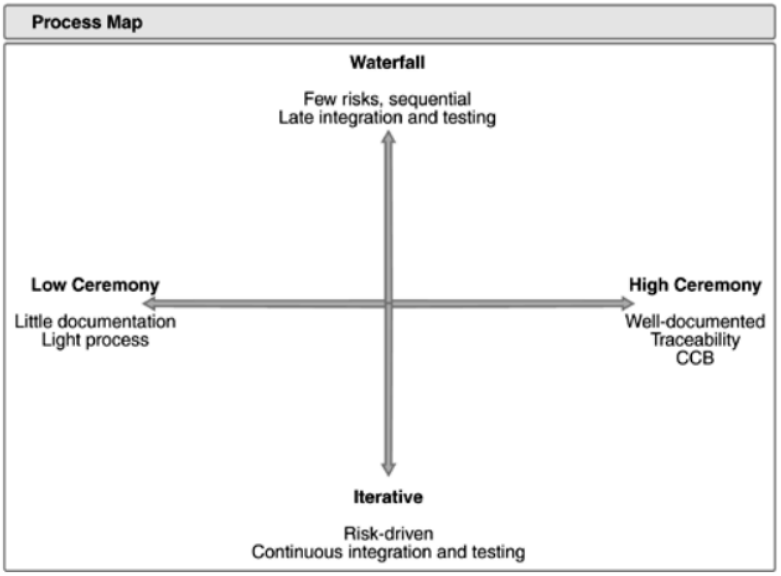


Figura 8: Mapa para la comparación de los procesos. Fuente: Kroll y Kruchten

Además proponen algunas configuraciones del RUP para que se puedan identificar las exigencias en el desarrollo del software. En este mapa los métodos más preceptivos, es decir, los dirigidos por planes, quedarían a la derecha en el eje horizontal y arriba en el eje vertical. Por otra parte los métodos ágiles quedarían más a la izquierda y abajo con baja ceremonia e interactividad.

Kroll y Kruchten (2003) propone una adaptación en RUP, que es un método dirigido por planes, que posibilita el desarrollo de un pequeño software con un desarrollador, con alta interactividad, y siguiendo de forma más orientativa el RUP. Esto respondería a la demanda de Sommerville (2011, p.5) de que "la Ingeniería de Software busca apoyar el desarrollo de software profesional, en lugar de la programación individual" ya que, de acuerdo con las propuestas de Kroll y Kruchten (2003) y Ambler (2002) es posible generar una documentación dinámica y objetiva al mismo tiempo que se desarrolla un software de calidad. Igualmente, Ambler (2002) propone una integración de UP y MA que denomina

Enterprise Unified Process (EUP). Esta adaptación permite, desde una perspectiva ágil, seguir un proceso claro y hacer que UP potencie la aplicación de las metodologías ágiles[10]. En EUP, Ambler (2002) añade a UP una fase de producción, y las disciplinas de operaciones y apoyo de gestión de infraestructuras con el objetivo de planificar el tiempo de salida del software al mercado.

2.5. Diagramas UML

Unified Modeling Language (UML)[11] es un lenguaje de propósito general que tiene como objetivo hacer una descripción visual del sistema y, por lo tanto, mostrar la conceptualización del mismo, lo que contribuye a revelar el patrón de diseño (Rumbaugh, Jacobson y Booch, 2005). A pesar de que muchos de los modelos creados antes de 1994 hayan sido la base para la creación de UML, él no fue aceptado por *Object Management Group* (OMG)[12] hasta 1997 cuando se convirtió en estándar para la industria de software (Arlow y Neustadt, 2006).

UML ha representado un gran avance para el modelado de sistemas orientados a objetos ya que, por medio de diagramas y un lenguaje visual, describe las decisiones y diseño del software en las diversas fases del desarrollo (Rumbaugh, Jacobson y Booch, 2005). Rumbaugh, Jacobson y Booch (2005, p.3) señalan que la utilización del UML ayuda a "understand, design, browse, configure, maintain and control information on such systems". Fowler (2004) añade que el principal valor de UML es la comunicación y la comprensión.

UML aporta un lenguaje de desarrollo tan sencillo como sea posible, organizando las buenas prácticas necesarias para estructurar una solución orientada a objetos y ayudar al analista a concebir una solución estructurada antes de empezar a desarrollarlo. El lenguaje no

[10] Ambler (2002, p.228) ofrece una tabla con una detallada descripción de prácticas ágiles asociadas a UP/RUP .

[11] El lenguaje tiene el objetivo de crear un estándar para la industria de software, ya que entre los años 70 y 90 surgieron un gran número de lenguajes de modelado que buscaban contribuir al desarrollo de un software orientado a objeto. Aunque algunos de estos lenguajes tenían un gran potencial, cumplían de forma incompleta sus propósitos y, a veces, eran consideradas más como una directriz que un lenguaje. Así UML propone un estándar que nació de la mezcla de muchos lenguajes, propuestas y estudios realizado hasta entonces (Rumbaugh, Jacobson y Booch, 2005).

[12] "Object Management Group es un consorcio, formado en 1989, dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XMI, CORBA y BPMN". Ver en https://en.wikipedia.org/wiki/Object_Management_Group. Acceso en 31/07/2017.

define en qué momento se debe utilizar cada diagrama sino que deja esa responsabilidad al profesional (Rumbaugh, Jacobson y Booch, 2005).

2.6. Ingeniería de requisitos en MA

La ingeniería de requisitos o requerimientos es una rama de la ingeniería de software que se estructura en dos partes: la gestión y el desarrollo de los requisitos además de una serie de técnicas para capturar, analizar, validar y especificarlos dentro del software. Otro rasgo que viene mejorar la comprensión de los requisitos es su clasificación en funcionales, no funcionales y de dominio (Sommerville, 2011). Sommerville (2011, p.83) define los requisitos como "descripciones de lo que el sistema debe hacer: el servicio que ofrece y las restricciones en su operación".

Así, a partir de los MA, la técnica utilizada para la captura de requisitos se denomina *user stories* o historias de usuario. Cohn (2010, p.239) señala que, "a user story is a short, simple description of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system". Además una *user story* puede tener distintos niveles de granularidad desde los más amplio hasta lo más bajo como *epic*, *feature*, *stories* y *theme* (Cohn, 2010). Para alcanzar la simplicidad y la objetividad necesaria en una *user story*, la misma se describe en la perspectiva del usuario desde lo más importante hasta lo menos importante. Es importante decir que una *user story* no encierra toda la estructura de los requisitos sino que trae la comprensión macro de una característica que debe ser fragmentada en tareas más específicas antes de ser implementada (Cohn, 2010).

2.7. Patrones de diseño

Mientras que el proceso de desarrollo atiende a los aspectos más amplios de la ingeniería de software, los patrones de diseño influyen directamente en cómo se desarrolla el software. Zandstra (2016), señala que el concepto tuvo su origen en los años 1970 en la arquitectura, aunque no fue utilizado por los desarrolladores de proyectos orientados a objetos (OO) hasta los años 1990. Mientras se desarrollaban estos conceptos, otros como *Extreme Programming* (XP) crecían simultáneamente fomentando la planificación y ejecución flexible orientada al diseño. Al mismo tiempo que este concepto comenzó a influir

en los entornos de desarrollo, recibía igualmente múltiples influencias. Es el caso de XP con propuestas tales como la defensa de que las pruebas constantes son esenciales, que los proyectos deben ser divididos en pequeñas interacciones en tanto cuanto sea posible, que el código y los requisitos necesitan ser constantemente revisados y que todo el equipo esté presente en el diseño (Zandstra, 2016).

Para Gamma, Helm, Johnson y Vlissides (1994, p.2) "design patterns help you choose design alternatives that make a system reusable and avoid alternatives that compromise reusability". Por seren técnicas ya utilizadas y experimentadas, los patrones de diseño mejoran la codificación incrementando la calidad y hace que el mismo sea mejor para los desarrolladores. En la tabla 1 vemos los patrones de diseño propuestos por Gamma, Helm, Johnson y Vlissides (1994).

Tabla 1: Aspectos quese pueden variar en función de los patrones de diseño. Fuente: Gamma, Helm, Johnson y Vlissides (1994)

Purpose	**Design pattern**	**Aspect(s) that can vary**
Creational	Abstract Factory	families of product objects
	Builder	how a composite object gets created
	Factory Method	subclass of object that is instantiated
	Prototype	class of object that is instantiated
	Singleton	the sole instance of a class
Structural	Adapter	interface to an object
	Bridge	implementation of an object
	Decorator	responsibilities of an object without subclassing
	Facade	interface to a subsystem
	Flyweight	storage costs of objects
	Proxy	how an object is accessed; its location
Behavioral	Chain of Responsibility	object that can fulfill a request
	Command	when and how a request is fulfilled

	Interpreter	grammar and interpretation of a language
	Iterator	how an aggregate's elements are accessed, traversed
	Mediator	how and which objects interact with each other
	Memento	what private information is stored outside an object, and when
	Observer	number of objects that depend on another object; how the dependent objects stay up to date
	State	states of an object
	Strategy	an algorithm
	Template Method	steps of an algorithm
	Visitor	operations that can be applied to object(s) without changing their class(es)

De esta manera consideramos que un patrón de diseño, es la descripción de un enfoque para resolver un determinado problema. Los patrones de diseño también considerados como buenas prácticas de desarrollo suelen ser soluciones en las que ya hay un consenso y han sido documentadas (Gamma, Helm, Johnson y Vlissides, 1994). Como están basados en la práctica, los problemas son fragmentados y resueltos desde la más pequeña estructura hasta la más grande. No se trata de una solución hecha, sino de un concepto de técnicas que nos pueden llevar a desarrollar software con más calidad. En ese contexto, Zandstra (2016, p.158) habla que "Design patterns inscribe approaches to particular problems. The details of implementation may vary enormously according to the wider context.".

2.8. Patrones MVC

Model View Controller (MVC)[13] es un patrón de diseño y una arquitectura de software muy utilizado, su uso va más allá de definiciones tecnológicas como la selección del lenguaje que se está utilizando. La principal aportación de MVC es que separa la lógica de

[13] Fowler (2006) hace una explicación más detalles en su página online. Ver en https://martinfowler.com/eaaDev/uiArchs.html . Acceso en 10/16/2017

dominio/aplicación/negocio de la interfaz de usuario y por lo tanto crea los componentes de modelo, vista y controlador (Fowler, 2006).

En este tipo de modelado (Figura 9) en el que el controlador media la comunicación entre el modelo de datos y la visualización del usuario, de manera que el modelo gestiona el comportamiento de los principales datos y responde a las solicitudes.

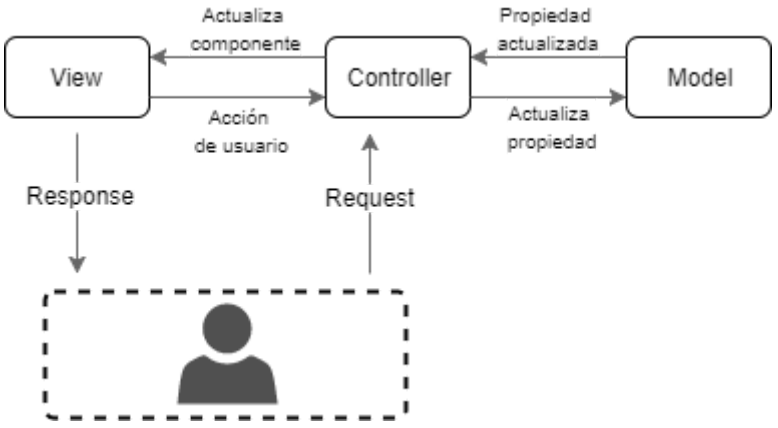


Figura 9: Esquema Model-View-Controller.

El componente de vista separa de manera efectiva la interfaz con el usuario de la aplicación para trabajar por separado las interacciones necesarias del usuario, lo que fortalece la modularización y permite a dos equipos, backend y frontend, trabajar al mismo tiempo en un mismo código. El componente control recibe la solicitud del usuario y establece la comunicación con la capa modelo respondiendo a las solicitudes del usuario. Por ello, el concepto de MVC está en constante evolución, buscando responder mejor a los problemas (Fowler, 2006). Estas arquitecturas son directamente aplicadas a los desarrollos de software web y muy utilizadas por los desarrolladores que programan con lenguajes Java, Python, .Net y PHP.

2.9. El lenguaje PHP

Hypertext Preprocessor (PHP) es un lenguaje de desarrollo interpretado y no ejecutable, con la ventaja de su fácil comprensión. Pero, en ocasiones es un problema, dado que es un lenguaje muy fácil de aprender, flexible y capaz de adaptarse a muchas formas de desarrollo, incluyendo las malas (Zandstra, 2016). A eso se suma el tiempo de aprendizaje que es bastante corto en comparación con otros lenguajes. Desde sus inicios, su sencillez y

potencial han contribuido a generar una gran y fuerte comunidad de desarrolladores, lo que ha facilitado el diálogo y constante aprendizaje, además de garantizar el avance continuo del lenguaje. Finalmente, es fácil encontrar servidores que puedan interpretar sus contenidos y, por lo tanto, lograr fácilmente un resultado (Lockhart, 2015).

Aunque su estructura ha evolucionado mucho, a lo largo de los años mantiene su facilidad y sencillez, lo que ha contribuido a que se generasen algunas veces códigos que no corresponden a técnicas muy utilizadas. De esta manera PHP tiene un gran potencial no solo por su sencillez sino también por su capacidad de producir un código orientado a objeto con alto grado de seguridad. Esto se produce principalmente a partir de PHP 5 en el que se han añadido un sólido modelo de objetos, funciones anónimas y *namespaces* en PHP 5.3, y *traits* en PHP 5.4 (Lockhart y Sturgeon, 2017).

La calidad y potencial del desarrollo con PHP se pueden ver reflejados en muchos frameworks que utilizan este lenguaje como Zend, Aura, Laravel, Yii y Symfony (Lockhart, 2015). En el caso de Symfony que es un framework PHP orientado a objetos, y que se define como "a set of PHP Components, a Web Application framework, a Philosophy, and a Community" (Symfony, 2017, s/p), muchas de las recomendaciones de buenas prácticas y estándares se ven reflejadas en su estructura. Potencier (2011, s/p) lo define como "a reusable set of standalone, decoupled, and cohesive PHP components that solve common web development problems" .

2.10. El proceso de evaluación participativa de la gobernanza de riesgo de desastres

Collins (2015) señala que la reducción del riesgo de desastres es una cuestión de gobernanza, de cómo las personas y no sólo los decisores políticos toman decisiones que afectan a la creación y mitigación del riesgo. Por otro lado, la toma de decisiones está influida por múltiples factores, tales como la percepción, el conocimiento, el poder, la cultura, el entorno en el que desarrolla su actividad, entre otros. De esa manera la acción humana afecta a la magnitud y la frecuencia de los desastres (Dickinson y Burton, 2015).

Los desastres, según Fra.Paleo (2015), pueden ser comprendidos como resultado de las decisiones individuales y colectivas tomadas por los actores sociales, como los gobiernos, el sector privado, los grupos de la sociedad civil y los ciudadanos que se exponen a los peligros, configurando situaciones de distinto grado de vulnerabilidad. Así, los actores se tornan clave en la toma de decisiones por lo que el objetivo debe ser una gobernanza más democrática.

En este punto Fra.Paleo (2015, p.238) considera que "evaluation processes can be recognized as opportunities for increasing societal interaction between groups and for the social construction of risk if we are interested in the process as much as in the outcome". Por consiguiente propone que la evaluación de la gobernanza del riesgo vaya más allá de la medición de resultados, teniendo en cuenta cuatro dimensiones: evaluación como participación más allá de la medición de resultados; evaluación como aprendizaje social; evaluación como la construcción del consenso; y evaluación como proceso de gobernanza.

Esto lleva a pensar que el proceso de evaluación no sea un fenómeno puntual en el tiempo y pase a ser un proceso dinámico de aprendizaje y de mejora continuada, conectado con las perspectivas de evaluación y monitorización. Hay que mencionar además que esta perspectiva favorece la diversidad de puntos de vista respecto de un problema y reconoce la democratización en el acceso, uso y control de la información confrontando directamente fuerzas que puedan manipularlos (Fra.Paleo, 2015).

Este modelo de evaluación se basa en criterios, y se apoya en evidencias documentales de las políticas y del marco normativo. Este enfoque favorece que la evaluación pueda ir más allá de la medición de resultados, ya que posibilita que entre los involucrados estén expertos, administradores públicos y actores sociales que puedan confrontar distintos puntos de vista en busca de una comprensión de lo que está siendo evaluado (Fra.Paleo, 2015).

De este modo, el proceso de evaluación fortalece la democratización al permitir que distintas interpretaciones examinen el problema y contribuye a la gestión del riesgo, generando un constante aprendizaje que retroalimenta el sistema generando mejores

intervenciones en el futuro. Además, la evaluación cualitativa mediante criterios contribuye a que el proceso se adapte al dinamismo de la gobernanza del riesgo (Fra.Paleo, 2015).

La evaluación de un criterio tiene por objetivo calificarlo de forma consensuada y argumentada en alguno de los cinco niveles de desarrollo, según Fra.Paleo (2015, 258):

1. No se han identificado avances en relación con el criterio;
2. Se han adoptado medidas aisladas sin responder a un programa estructurado;
3. Se han elaborado programas, pero no se han aplicado a través de proyectos;
4. Se han elaborado programas y los proyectos traducen sus principios. Tanto los planes como los proyectos se evalúan ocasionalmente;
5. Los programas y proyectos responden a una política diseñada para el diseño. Tanto programas como proyectos son evaluados regularmente.

Y, finalmente, para la construcción formal de la argumentación de los acuerdos se plantea dos cuestiones más;

1. ¿Están siendo evaluados los impactos del programa/política frente a los objetivos de reducción del riesgo de desastre?
2. ¿Están la sociedad civil y el mercado involucrados en el diseño, implementación, y evaluación de programa/política frente a esos objetivos?

Sabiendo que cuando un evaluador toma posición lo hace desde su punto de vista, y que la existencia de muchos actores en el proceso hace que se produzca una confrontación de argumentaciones y puntos de vista, es necesario buscar un consenso (Fra.Paleo, 2015). Una alternativa es la búsqueda de mayorías, pero esto no favorece la negociación y el consenso social sobre la distribución del riesgo, ya que las mayorías han producido una distribución del riesgo desigual (Fra.Paleo, 2015). Por ello es necesario negociar puntos de vista e identificar estrategias para esta negociación.

2.10.1. Otros procesos de evaluación

El sistema de evaluación participativa de la gobernanza de riesgo de desastres propuesto es integrador y completo como el instrumento *Assessment Instrument for*

Sustainability in Higher Education (AISHE)[14], puesto en marcha por un grupo de investigadores europeos entre los que se encuentra Fra.Paleo (Roorda, Rammel, Waara y Fra.Paleo, 2009) . Este sistema de evaluación de sostenibilidad de universidades está basado en el consenso, en el que están involucrados todos los actores universitarios, y la evaluación utiliza criterios cualitativos y no cuantitativos. Por ello contempla la existencia de un facilitador, que tiene una posición neutral y permite buscar soluciones que puedan satisfacer a todas las partes. AISHE 2.0 es basado en cinco módulos (operations, education, research, society y identity), cada módulo contiene seis criterios distintos, y su evaluación se fundamenta en cinco etapas de desarrollo (activity, process, system, chain y society). Como parte del proceso se ha desarrollado la aplicación informática 'AISHE 2.0 Reporter', que sirve para representar los resultados de la evaluación, el horizonte de objetivos buscado y el horizonte de objetivos alcanzable.

Aunque hay una gran cantidad de índices e indicadores (Tabla 2) que permiten medir el estado de distintos componentes y factores del riesgo de desastres que pueden ser utilizados como posibles criterios de evaluación, como el *disaster preparedness index* (DPi), *Disaster Deficit Index* DDI, el *Local Disaster Index* (LDI), el *Prevalent Vulnerability Index* (PVI), el *Risk Management Index* (RMI), el *disaster risk reduction* (DRR) y el *Hyogo Framework for Action Monitor* (HFA Monitor), los instrumentos completos de evaluación no son comunes. Así Fra.Paleo (2015, p.249) señala que "the evaluation of performance in risk management is also a recent development and instruments to measure are not abundant".

Tabla 2: Indicadores e índices de la gobernanza del riesgo. Fuente: Fra.Paleo (2015).

Indicador/index	**Strengths**	**Weaknesses**
Ibrahim Index of African Governance (IIAG)	Composite index of aggregated data	- African countries - Dependent on information from other organizations - Proxy measurement when data are missing - Criteria capture the quality of services provided to citizens by governments
Mo Ibrahim Foundation 2007		

[14] Los autores de AISHE lo definen como un instrumento de evaluación sostenible además de "a tool for the development of a policy towards sustainable development: by a university, a campus, a faculty, a school, an academy, an institute, or a separate education or research program". Se puede consultar en: https://niko.roorda.nu/books/aishe/. Acceso en: 22/09/2017.

Sustainable Governance Indicators	Qualitative and quantitative indicators	OECD countries
Bertelsmann Foundation 2009	Available data for evaluation	- Dependent on expert evaluation - Focuses on needs for reform and effectiveness of existing initiatives
Index of State Weakness in the Developing World	State weakness as a function of its effectiveness, responsiveness, and legitimacy	- Focuses on state function but neglects other political players
Brookes Institution 2008		
Worldwide Governance Indicators (WGIr)	- Based on survey respondents - Diverse group of respondents: citizens, entrepreneurs, and experts in the public, private sectors and NGO - It focuses on quality of governance	- Based on perception - The underlying principle associates good governance with economic development
Brookings Institution, World Bank Development Research Group, and World Bank Institute 1996		
World Governance Index (WGIx)	Synthetic index	- Synthetic index - Dependent on information from other organizations - Only quantitative indicators
Forum for a new World Governance (FnWG) 2008		
Global reporting initiative (GRI)	Takes into account the economic, environmental and social dimensions	- Oriented to reporting towards corporate social responsibility - Any kind of organization, but particularly corporations

2.10.2. Versión 1.0 de Risonance y sus debilidades

Risonance es un *Policy Support System* (PSS) que tiene dos propósitos a nivel macro: evaluar la gobernanza del riesgo de desastres de forma participativa y ayudar a la formulación de las políticas públicas. Adicionalmente promueve el aprendizaje social, incrementa la conciencia y favorece la coproducción de conocimiento sobre el riesgo de desastres.

La versión inicial del software fue diseñada por el profesor Urbano Fra Paleo juntamente con un equipo de tres desarrolladores en 2010. Aunque la primera versión de Risonance recogía los principios fundamentales de la evaluación como el registro de usuario, el envío de propuestas para la evaluación, la invitación de evaluadores, esta tenía graves problemas de codificación, desde las perspectivas del usuario, del cliente y del equipo de desarrollo. Desde la perspectiva del usuario el software no fue utilizado ni testeado de manera intensiva y, por lo tanto, no hay ningún registro de errores documentado. Desde la perspectiva del cliente, el software contemplaba funciones necesarias para la evaluación que no fueron finalmente implementadas como la generación de los informes parciales y finales además de la generación automática del consenso. Por otro lado, tenía poca adaptabilidad a los dispositivos móviles con un diseño anticuado. Podemos ver un ejemplo en la figura 10.

Figura 10:Pantalla para verificar los actores involucrados en un proceso de evaluación. Fuente: Risonance 1.0.

Aún así, los principales problemas aparecen en la parte técnica, o sea, desde la perspectiva del desarrollador. Algunas funciones del PHP que fueron utilizadas en el software estaban desactualizadas o en desuso, como la utilización de la función mysql_connect que ha sido declarada obsoleta en la versión 5.5 y eliminada en PHP 7.0. También había problemas graves de seguridad como la no encriptación de las contraseñas y el código contenía una mezcla de lenguajes PHP, HTML, Javascript y CSS (figura 11) con más de 125 archivos. Finalmente, el código no tenía una orientación clara y estaba disperso (figura 12).

```
<?php
session_start();
include ("funciones.php");
//Comprueba que se ha iniciado sesión
if (isset($_SESSION['id_usuario'])) {
    $arrUsuarios = listarUsuarios($_SESSION['id_usuario']);
    if (isset($arrUsuarios) && count($arrUsuarios) ) {
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<link rel="stylesheet" type="text/css" href="css/pro_dropdown_3.css" />
<link rel="stylesheet" type="text/css" href="css/style.css" />
<link rel="stylesheet" type="text/css" href="css/style_tabla.css" />
<script src="js/stuHover.js" type="text/javascript"></script>
<script type="text/javascript" src="js/jquery-1.4.4.js"></script>
<script type="text/javascript" src="js/tablesorter.js"></script>
<script type="text/javascript" src="js/jquery.tablesorter.pager.js"></script>
<script type="text/javascript">
    $(document).ready(function() {
        $("#myTable").tablesorter({
            debug: false,
            widgets: ['zebra'],
            headers: {
                3: { sorter: false }
            }
        })
        .tablesorterPager({ container: $("#pagerOne"), positionFixed: false })
        $("#myTable .header").click(function() {
            $("#myTable tfoot .first").click();
        });
    });

    function recargarPagina() {
        document.getElementById("form1").submit();
    }

    function redirigirAtras(action) {
        var form=document.forms.form1;
        form.action=action;
        document.getElementById("form1").submit();
    }
</script>
```

Figura 11: Mezclas de códigos en el archivo evaluador_processos.php. Fuente: Risonance 1.0.

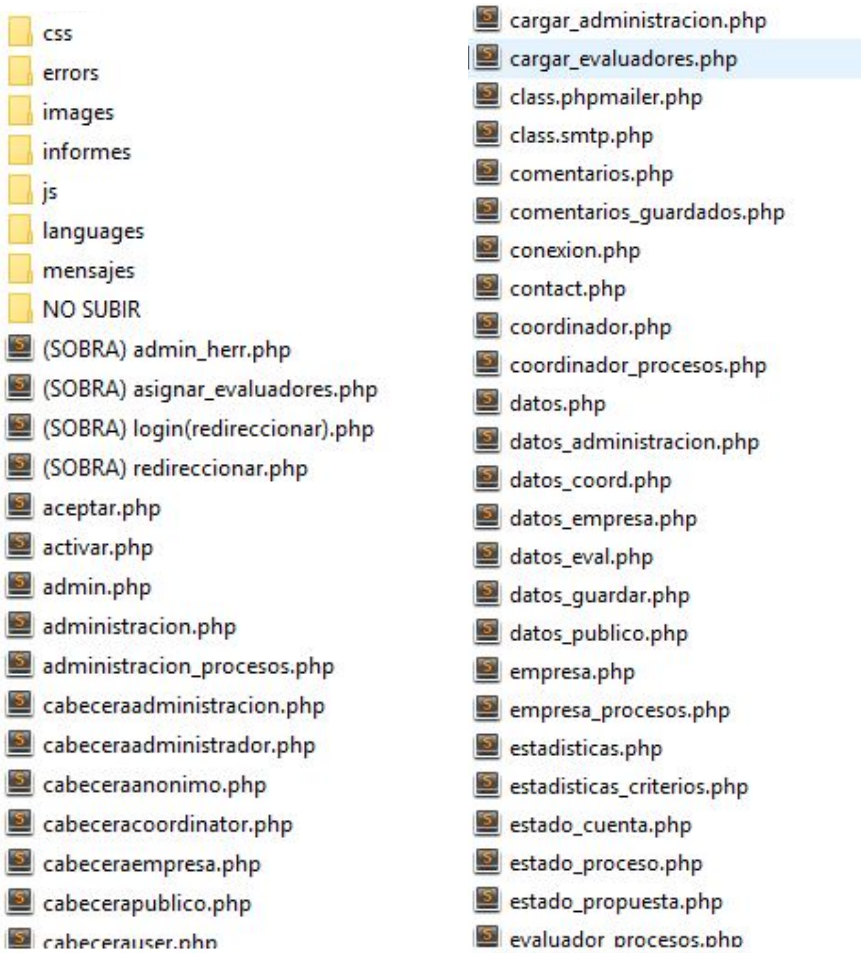


Figura 12: Estructura de archivos de la versión 1.0 de Risonance. Fuente: Risonance 1.0.

Este modo de desarrollar en PHP es una antigua herencia y viene todavía de algo que ha sido su fortaleza y debilidad desde el principio, ya que no exige que trabaje con orientación a objeto permitiendo rápidas implementaciones para testeo de funcionalidades. Sin embargo, en el ciclo de vida del software es inviable que se mantenga esta práctica ya que conduce a la desorganización y dificulta pensar en posibles cambios.

Habría que decir además que Risonance 1.0 no ha sido nunca hospedado en ningún servidor online. No hay tampoco ningún tipo de documentación que pueda auxiliar el desarrollo o la comprensión de su estructura, hecho que al principio ha dificultado verificar cómo funcionaba el software. Así, proponemos para la versión 1.0 un análisis basado en la matriz DAFO[15] (Tabla 3).

[15] La matriz DAFO ha sido diseñada con una perspectiva administrativa y es utilizada para realizar un análisis estratégico de una empresa. DAFO es un acrónimo formado a partir de las iniciales de Debilidades, Amenazas, Fortalezas y Oportunidades. Disponible en: https://es.wikipedia.org/wiki/An%C3%A1lisis_DAFO. Accedido en 02/08/2017.

Tabla 3: Análisis DAFO de Risonance 1.0.

	Aspectos Negativos	Aspectos Positivos
Origen Interno	**Debilidades**	**Forlalezas**
	- código disperso y duplicado. - estructura de archivos, ficheros y código lejos de las recomendaciones PSR[16]. - mezcla de códigos PHP, HTML, Javascript y CSS. - código sin orientación a objetos. - ninguna documentación que ayude al desarrollo. - el aspecto visual no responde las necesidades actuales y no funciona bien en todos dispositivos. - recoge parcialmente las funciones de evaluación previstas	- el uso del framework DCC para la evaluación de la gobernanza participativa del riesgo.
Origen Externo	**Amenazas**	**Oportunidades**
	- posibles propuestas de cambio, aunque sencillas. - cambio en el aspecto visual. - propuestas basadas en inteligencia artificial.	- soporte para que las instituciones públicas sean cada vez más transparentes en la toma de decisiones. - involucrar a los actores sociales en la toma de decisiones en las políticas públicas. - generar un constante aprendizaje por medio de un proceso de evaluación participativo. -implementación de soluciones más inteligentes en el proceso de evaluación.

Lo que buscamos por medio del análisis DAFO es identificar las debilidades que se quiere minimizar en la versión 2.0 de Risonance y la reducción del impacto de las amenazas, haciendo que las oportunidades sean mejor aprovechadas con el objetivo de potenciar las fortalezas de Risonance 1.0. En la tabla 4 se señalan los aspectos generales de esta primera versión que se relacionan con la Ingeniería de Software.

[16] PSR es un acrónimo para *PHP standards recommendation* publicadas por PHP-FIG. PSR está subdividido en cuatro grandes bloques: PSR-1 *Basic code style*, PSR-2 *Strict code style*, PSR-3 *Logger interface*, PSR-4 *Autoloading*. Más detalles en Lockhart (2015) y en http://www.php-fig.org/. Accedido en 31/07/2017.

Tabla 4: Características del Risonance versión 1.0.

Características	**Risonance 1.0**
Equipo involucrado	3 desarrolladores.
Uso de recomendaciones de buenas prácticas PSR	Han seguido de forma muy laxa alguna recomendación como en relación a nombres de variables y funciones.
Arquitectura utilizada	No hay demostración del uso de ninguna arquitectura o framework.
Uso de procesos de desarrollo	El código no ha seguido ninguna orientación de modelo de proceso o ninguna organización, aunque ágil.
Generación de documentación	No hay ningún tipo de documentación que pueda ser utilizada.

Hay que mencionar que, de acuerdo con Lockhart (2015), a la complejidad presente en el desarrollo PHP orientado a objetos, está involucrada la utilización de las buenas prácticas propuestas por la comunidad PHP, además de las características del lenguaje, como el uso de *Namespaces*, *Interfaces*, *Traits*, *Closures*. No quiere decir con eso que se deba usar todas las técnicas al mismo tiempo sino que pensar en el problema y en la solución, pero el total imcumplimiento de estas buenas prácticas puede demostrar malas técnicas de desarrollo y un código de mala calidad.

3. Metodologia

En este apartado detallaremos la metodología de desarrollo de la versión 2.0 de Risonance, que ha involucrado al cliente y al ingeniero de software. Con el objetivo de integrar MA y RUP hemos mantenido una constante comunicación entre los dos actores por medio de flujos de trabajo del software, expresados tanto verbal como gráficamente, construyendo una estructura de diagramas sencilla y manteniendo constante retroalimentación, lo que ha garantizado el seguimiento de los valores ágiles.

Por el entorno de trabajo de Risonance y sus características específicas, decidimos seguir la estructura de procesos de RUP de alta interactividad (Figura 5). Dentro de este proceso mantuvimos el enfoque principalmente en las tres primeras disciplinas propuestas, *Business Modeling*, *Requirements* y *Analysis & Design*, dado que son las fases de modelado (Ambler, 2002). Para la producción de la fundamental documentación se hizo necesario un ajuste del RUP, a partir de lo propuesto por Kroll y Kruchten (2003), que ofrecen una descripción de cuatro proyectos como ejemplos orientativos (Figura 13). Éstos poseen un grado distinto de ceremonia e interactividad que son resultado de la descripción de algunas características tales como el tiempo de desarrollo, la complejidad, la cantidad de personal técnico involucrado, o el grado necesario de adaptabilidad a los cambios.

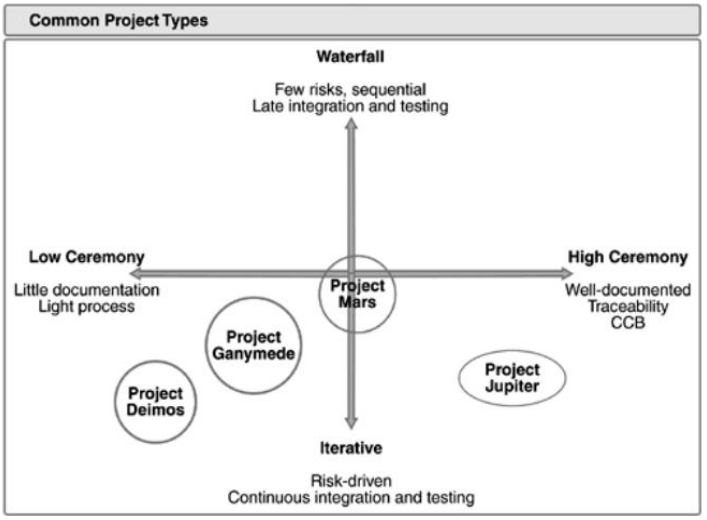


Figura 13: Proyectos de ejemplo en el Mapa de Procesos. Fuente: Kroll y Kruchten, 2003.

Para el ajuste propuesto utilizamos el '*Project Deimos*' y el '*Project Ganymede*'[17] por acercarse a las características de la versión 2.0 de Risonance. Algunas condiciones que se destacan en este sentido son el hecho de poseer uno solo desarrollador, un software con baja complejidad técnica, grandes probabilidades de cambios en los requisitos, además de poseer un equipo técnico con experiencia (Tabla 5).

Tabla 5: Proyecto Risonance versión 2.0.

Risonance	
Personas técnicas involucradas	1
Restricciones de tiempo	negociable
Tamaño del proyecto	pequeño
Duración	3 meses / corta
Base de datos	centrada
Probabilidad de cambios en los requisitos	cambia algo
Experiencia del equipo en desarrollo similares	miembro con experiencia
Despliegue de arreglos	permite despliegue
Resultado para posicionamiento en el gráfico	
Nivel de interactividad	altamente interactivo
Nivel de ceremonia	baja ceremonia

Estas características señalan la cercanía de Risonance con el '*Project Deimos*' y el '*Project Ganymede*' lo que sirvió de orientación para la elección de la documentación generada y la comprensión del proceso de desarrollo, que ha funcionado más como una orientación que como un proceso formal. Ese ajuste del RUP orientó al desarrollador para la construcción de un software de calidad siguiendo un modelo y una estructura atendiendo a la

[17] Los detalles de los cuatro proyectos se pueden ver en Kroll y Kruchten (2003) en las páginas 61 y 62.

flexibilidad e interactividad necesaria a los MA, según los valores propuestos por Ambler (2002).

La solución técnica para la nueva versión de Risonance, permitió la comprensión del problema, además de la propuesta de solución orientada a objetos que se refleja en el código generado. En el desarrollo se han utilizado las propuestas de Lockhart (2015) además de Lockhart y Sturgeon (2017) que se materializan en conceptos basados en la creación de componentes y buenas prácticas PHP, contempladas en Symfony. Esas propuestas, realizadas en primer lugar por la comunidad PHP para el desarrollo orientado a objetos, siguen la utilización de componentes, la reutilización de códigos y el uso extensivo de buenas prácticas. Se debe añadir también que son muy importantes para la construcción de un código seguro y de calidad y que su uso está garantizado por medio de un framework que las ha contemplado, como es el caso de Symfony, mencionado por Lockhart (2015, p.54) como "an excellent example of a modern PHP framework".

Dada la variedad y la extensión de las buenas prácticas, señalamos las que presentan las características más sencillas, como las propuestas por *PHP Standards Recommendation* (PSR), que se dividen en cuatro grupos.

- PSR-1 proporciona un estilo de código base para la integración entre códigos PHP independiente del framework. En esta estructura se definen aspectos como nombres de clases, variables y constantes, la forma de usar los PHP tags, etc. (Lockhart, 2015).
- El PSR-2 que es "a common strict code style lets developers write code that is easily and quickly understood by other contributors" (Lockhart, 2015, p.41), y define aspectos como la indentación general del código, la indentación de clases y métodos uso de namespace, etc.
- PSR-3 es una propuesta de interfaz de logueo y por lo tanto no es un guía como las recomendaciones anteriores.
- Finalmente, PSR-4 que funciona como una estrategia para encontrar y cargar las clases, interfaces o trait PHP.

La metodología y los métodos utilizados en este trabajo se han sistematizado en la tabla 6.

Tabla 6: Cuadro metodológico general.

Etapas	Acciones	Procedimientos	Resultados esperados
Revisión de literatura	Estudio de conceptos	Consulta: libros, artículos, portales web	-Estructuración y identificación de definiciones generales, de software, ingeniería de software y su entorno -La calidad de software en los MP y MA -Los procesos de desarrollo de software -Definiciones de los diagramas UML -UML en la práctica de desarrollo de acuerdo con las direcciones de los MP y MA -Descripción de orientación a objetos así como sus beneficios -Descripción de la potencialidad del desarrollo web -La estructura y potencialidad de los framework -La estructura MVC (Model, View y Control)
	Estudio de caso de aplicaciones prácticas		-Estructuración de los MDP y MA, así como sus potencialidades -Definiciones de RUP y sus potencialidades -Symfony como una aplicación de framework orientado a objetos que utiliza la estructura MVC
Reestructurar el software Risonance	Fundamentación conceptual	Consulta: libros, artículos	-Definiciones conceptuales sobre riesgos, desastres, gobernanza de riesgos -Comprensión del proceso evaluación participativa
	Replanteamiento	Filosofía de trabajo: MDP aproximándose MA	-Documentación necesaria para garantizar el seguimiento de un modelo
		Estructurar la comprensión del sistema: UML	-Diagramas para obtener la comprensión necesaria para el software -Discutir los diagramas generados
	Codificación	Desarrollo en los lenguajes: PHP, HTML, CSS, Javascript	-Código orientado a objeto con PHP -Código frontend con HTML, CSS, javascript
	Mantenimiento y acciones futuras	Controlador de versiones de código: git	-Añadir código de forma gradual

4. Risonance 2.0

Se presentan los resultados de la versión 2.0 de Risonance utilizando dos perspectivas. La primera, a través de las técnicas de ingeniería de software utilizadas dentro del análisis y modelado del sistema, buscando una comprensión del problema y la estructuración de la solución. La segunda a través de la estructura de desarrollo del software, de las técnicas utilizadas, los estándares y las buenas prácticas de desarrollo.

La tabla 7 describe los cambios realizados en la versión 2.0 de Risonance, a partir de los problemas detectados en la versión 1.0, haciendo un resumen de las condiciones, direcciones y acciones efectivas para el desarrollo de la nueva versión del software.

Tabla 7: Comparación de las características de las dos versiones de Risonance.

Características	Versión 1.0	Versión 2.0
Equipo involucrado	tres desarrolladores.	un desarrollador.
Uso de recomendaciones de buenas prácticas PSR	Han seguido de forma muy débil alguna recomendación o han sido solamente buenas prácticas en relación a nombres de variables y funciones.	Uso intensivo de las recomendaciones de buenas prácticas propuestas por la comunidad PHP empezando por la utilización de las recomendaciones PSR.
Arquitectura utilizada	No hay demostración del uso de ninguna arquitectura o framework.	Uso del framework Symfony pensado en conformidad con la arquitectura MVC.
Uso de procesos de desarrollo	No hay demostración de que el desarrollo haya seguido alguna orientación de modelo de proceso u organización, ni siquiera ágil.	El software ha sido estructurado y desarrollado buscando adecuarse a las MA con un acercamiento al RUP y utilizando su estructura de proceso para una comprensión del desarrollo.
Generación de documentación	No hay ningún tipo de documentación que pueda ser utilizada.	Se ha documentado basándose en las orientaciones de RUP y MA.

Esa tabla no tiene por objetivo realizar una comparación sistemática del desarrollo del software, sino una sistematización de las cuestiones planteadas en la primera parte de este trabajo, configurando el camino del desarrollo de la versión 2.0 de Risonance, manteniendo una alta interactividad. De esta manera hemos observado cómo el cambio en la orientación del trabajo, así como en la utilización de técnicas ágiles y buenas prácticas de desarrollo, pudieron generar una mejoría de la calidad.

4.1. Análisis y estructura general de la versión 2.0

La clave de Risonance está en operacionalizar la evaluación participativa de la gobernanza de riesgo a través de criterios mediante una evaluación cualitativa en la que se involucran distintos actores, lo que se configura como el punto central del software. Es importante destacar que estos actores que presentaremos a continuación son comprendidos dentro del software con un papel a ser desempeñado dentro de un proceso de evaluación específico. Los actores de Risonance 2.0 son:

- **Anónimo**: responsable de registrar un nuevo usuario y confirmar el registro además de visualizar la información general en la página.
- **Administración del sistema**: responsable de administrar todos los proyectos de evaluación y a los usuarios, siendo principalmente responsable de aceptar o no una propuesta o un usuario.
- **Administraciones de las instituciones**: responsable de enviar la solicitud de registro de una institución, aportando información, de invitar a un coordinador del proceso de evaluación y de representar a esta institución. Define cuáles son las dimensiones que serán evaluadas. Generalmente esto lo hace alguien que trabaja en los altos niveles de la institución. Es además responsable de aportar las evidencias documentales por parte de la administración que sirvan al proceso de evaluación de los criterios..
- **Coordinadores de las instituciones**: es el responsable de invitar a los evaluadores, facilitadores y de valorar la posible incorporación al proceso de evaluación de los actores sociales que lo soliciten. Al invitar a los evaluadores, se deben definir cuáles son las dimensiones de las que serán responsables y las fechas de inicio y fin del

proceso de evaluación. También deben resolver cualquier problema o conflicto, que pueda surgir en el proceso de evaluación.

- **Evaluadores**: responsables de evaluar los criterios dentro de los plazos establecidos, siendo su experiencia la que definirá la calidad de la evaluación. Están autorizados a visualizar las otras evaluaciones dentro del proceso en el que participa.
- **Facilitadores:** responsables de proponer y lograr el consenso en una o más evaluaciones, su trabajo empieza cuando los evaluadores terminan de evaluar.
- **Actor social**: es responsable de evaluar los criterios, al igual que un evaluador pero, a diferencia de éste, solicita participar en el proceso porque considera que puede aportar al proceso. No participa, como individuo, sino que representa a una organización social.

Las principales acciones ejecutadas por cada actor están contempladas en los casos de usos (Figura 14), y éstos orientan la relación de cada usuario con el sistema. La explicación del que se debe hacer en cada caso de uso, y los detalles de su expansión se muestra en el Anexo A, ítem 1.

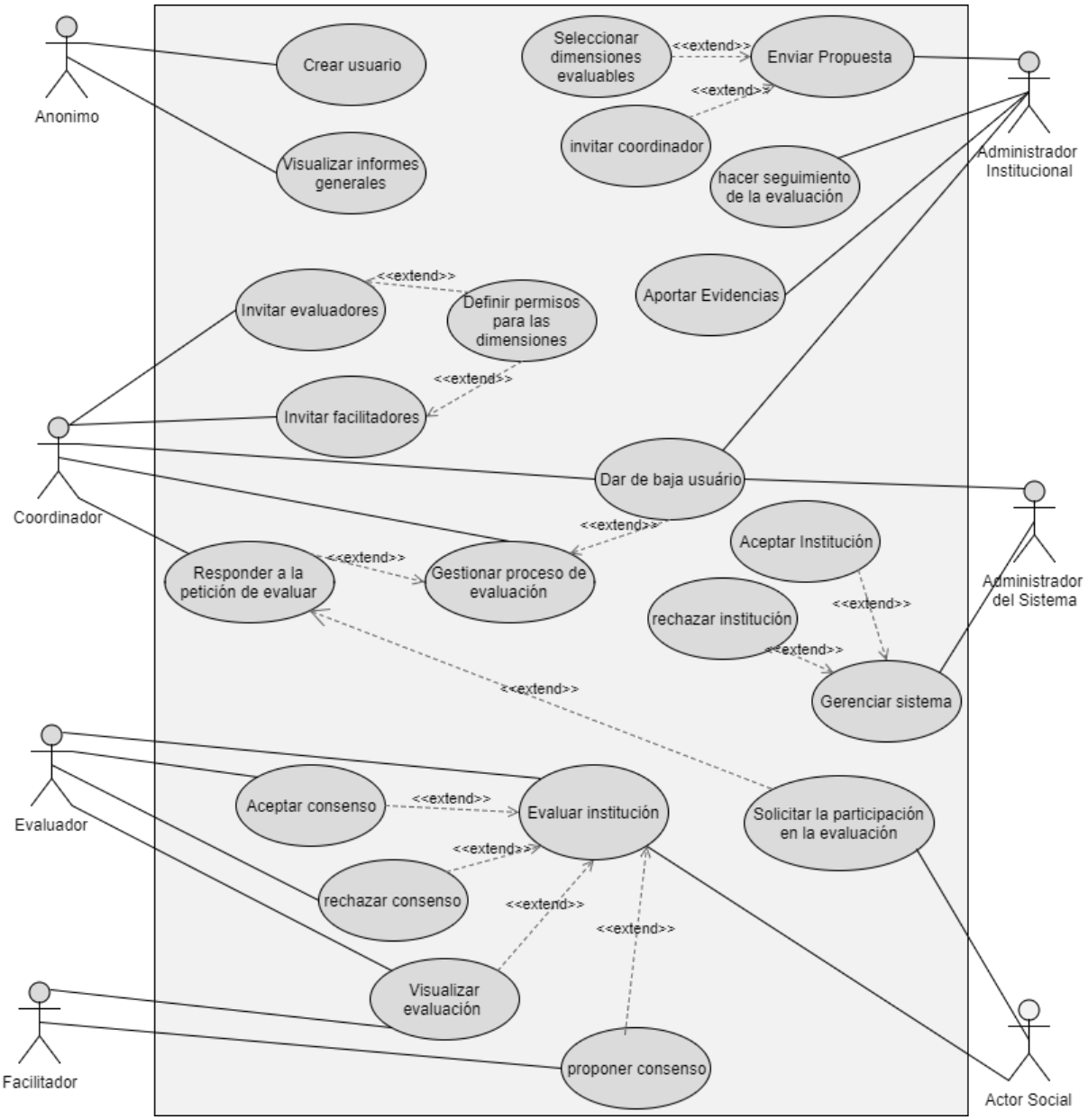


Figura 14: Diagrama de casos de uso.

A continuación sigue la descripción de cada caso de uso.

- **Crear usuario**: un usuario anónimo en Internet hace un registro de un nuevo usuario poniendo su nombre, apellidos, correo electrónico, nombre de usuario (login) y contraseña. Para entrar en el sistema el usuario deberá confirmar su registro mediante de un correo, lo que evitaría el registro de correos inválidos. En el caso de que el usuario no confirmase la recepción del correo, su usuario será borrado

automáticamente. El usuario sólo podrá registrar una institución si lo ha confirmado sus datos.

- **Visualizar informes generales**: todo lo que se desee informar para la comunidad interesada en las evaluaciones, o cualquier propósito general. Aquí se puede informar del método utilizado para evaluar, de los conceptos básicos de riesgo y gobernanza, de los procesos utilizados en el software o algún conocimiento necesario.
- **Enviar propuesta o Registrar institución**: registra la institución a ser evaluada y proporciona todas las informaciones y documentaciones necesarias para que eso ocurra. El usuario solo podrá invitar al coordinador si la institución es aceptada.
- **Seleccionar dimensiones evaluables**: al crear una institución se definen las dimensiones de la evaluación del riesgo de desastres que serán evaluadas.
- **Hacer el seguimiento de la evaluación**: supervisar el cumplimiento de los plazos, y añadir los datos que sean necesarios. También se podrá visualizar si los evaluadores han cumplido los plazos establecidos.
- **Aportar evidencias**: el usuario Administrador de la institución puede añadir muchos tipos de evidencias para la institución tales como: un archivo documental o la dirección de una página web, para que se pueda obtener información fundamental para evaluar los criterios.
- **Invitar coordinador**: hacer el envío de correo invitando a alguien a coordinar el proceso de evaluación. El envío será hecho por el administrador de la institución. En el caso de que la persona que haya recibido la invitación no tenga un registro en el sistema, recibirá la invitación y deberá hacer un registro para aceptar la invitación, en el caso contrário la invitación expirará en una semana. En el caso de que el usuario ya esté en el sistema recibirá una invitación por medio del sistema, además del correo. La invitación tiene validad de una semana, en el caso de que el usuario no conteste a la invitación será borrada automáticamente. El proceso se queda paralizado aquí hasta que el coordinador acepte la invitación.
- **Invitar evaluadores**: El coordinador hace un envío de un correo invitando a alguien a participar en el proceso de evaluación. En el caso de que no tenga un registro en el sistema recibirá la invitación y deberá hacer un registro para aceptar la invitación, en caso contrario la invitación expirará en una semana. Y en el caso de que el usuario ya esté en el sistema recibirá una invitación por medio del sistema además del correo. La

invitación tiene validez de una semana, en el caso de que el usuario no conteste la invitación será borrada automáticamente.

- **Definir permisos para las dimensiones**: Después de que cada usuario acepte participar en la evaluación, como facilitador o como evaluador, el coordinador definirá cuales son las dimensiones que éste evaluará. Además, el coordinador deberá definir un plazo límite. El coordinador deberá estar atento a la fecha inicial y final fijadas en el registro de la institución.
- **Gestionar el proceso de evaluación:** El usuario coordinador podrá identificar las evaluaciones con retrasos, criterios que van empezar a ser evaluados, y conocer cuál ha sido la última vez que un evaluador ha accedido el sistema. El usuario podrá dar de baja cualquier evaluador del proceso y justificarla, podrá aceptar a un actor social para participar en la evaluación además de visualizar todas las evaluaciones aunque el proceso esté inconcluso.
- **Invitar facilitadores**: El coordinador podrá invitar usuarios evaluadores para promover el consenso en un proceso de evaluación.
- **Eliminar usuario**: El coordinador y, el administrador del sistema pueden dar de baja a un usuario del proceso en el momento que consideren que ha roto las reglas del proceso o no realicen las tareas encomendadas.
- **Responder a la petición para evaluar:** El usuario podrá aceptar o rechazar la petición de un actor social. Para ello deberá contestar al usuario por medio de la plataforma virtual y justificar, lo que será visible para cualquier usuario de la página web.
- **Aceptar institución:** El usuario administrador deberá confirmar los datos de una institución después de contactar por teléfono o por cualquier otro medio y activar la aceptación.
- **Rechazar institución**: El usuario administrador podrá rechazar la utilización de la herramienta por parte de una institución después de contactar por teléfono o por cualquier otro medio y sin necesidad de dar ninguna justificación.
- **Gestionar sistema**: El responsable de gestionar el sistema recibirá un mensaje en la plataforma siempre que se reciba una solicitud de evaluación, lo que consideramos un registro de una institución. Además, podrá ver cuáles son las instituciones que están

siendo evaluadas, cuáles terminaron la evaluación y cuáles son las que están con retraso.

- **Evaluar institución**: Un evaluador propone una evaluación de un criterio y la envía. Después de que la evaluación sea salvada quedará disponible para los evaluadores, facilitador, coordinador, o administrador. El evaluador no puede editar la evaluación después de ser salvada.
- **Aceptar consenso**: Los evaluadores pueden aceptar o rechazar un consenso.
- **Visualizar evaluación**: Todos los usuarios que participan en una evaluación de la dimensión pueden acceder a los datos de esta evaluación para consultarla.
- **Proponer consenso**: El consenso, inicialmente propuesto por el facilitador, integrar las perspectivas de los evaluadores con opciones a ser aceptado por éstos. Más adelante se plantea que el sistema genere soluciones de consenso de forma automática con lo cual el propio software facilitará el proceso.
- **Solicitar la participación en la evaluación:** Un actor social puede hacer una petición para participar en la evaluación y convertirse en evaluador. Aunque el actor social sea considerado un usuario por el sistema, de hecho representa un grupo social que tiene interés en intervenir en la evaluación (asociaciones de vecinos, organizaciones no gubernamentales, etc).

Al mismo tiempo que se definieron los casos de uso se ha ido reestructurando la base de datos del software que se sistematiza en el modelo presentado en la figura 15, además de empezar a crear el diagrama de clases mediante las tarjetas CRC (Anexo A, ítem 3).

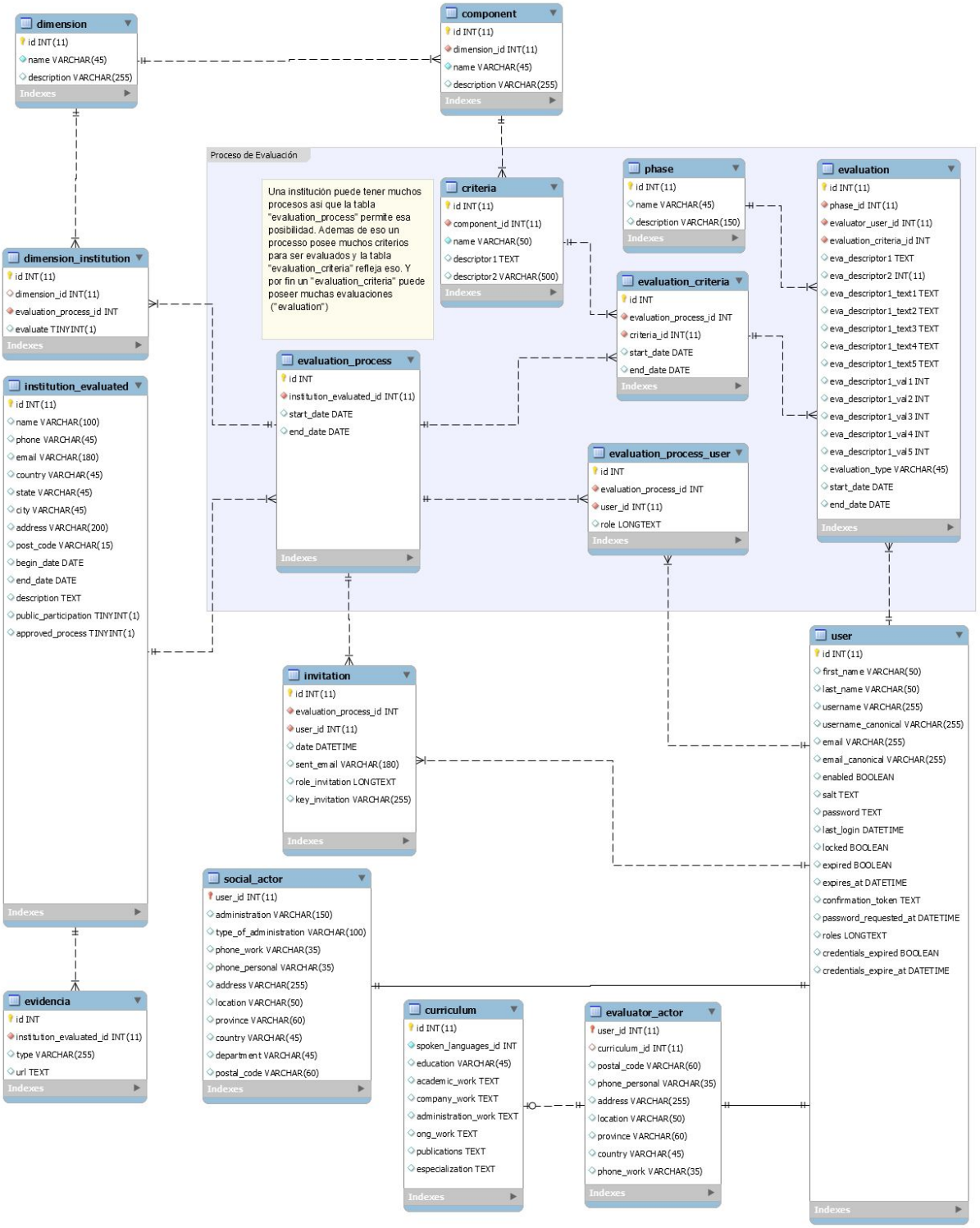


Figura 15: Diagrama Entidad-Relación de la base de datos.

4.2. Análisis y modelado del sistema

Se definieron cuatro *user stories* (Tabla 8), que son las características más amplias que deben estar contempladas en el software. A partir de la descripción de cada una, como sugieren los MA, se propusieron las interacciones así como la generación de resultados inmediatos para que el cliente pudiese acompañar e intervenir en el desarrollo. Así, por medio de UML, se hizo la descripción de la comprensión de cada una y la presentación de la propuesta de solución.

Tabla 8: User Stories definidas para Risonance versión 2.0.

#	**User Story**
1	Como usuario anónimo quiero hacer un registro de usuario y contraseña con alta seguridad para utilizar el software
2	Como administrador de una institución quiero hacer una petición para que la institución que represento sea evaluada
3	Como coordinador quiero crear un equipo para participar de todo el proceso de evaluación
4	Como actor participante de la evaluación quiero evaluar un criterio y contribuir para el proceso de evaluación

Para una mejor comprensión de cada *user story* fue igualmente importante la definición de las responsabilidades de cada uno de los usuarios dentro del software. Por lo tanto cada modelado buscó la descripción de las actividades al mismo tiempo que las definiciones de las responsabilidades.

Desde la perspectiva del proceso RUP (Figura 5) mantuvimos el enfoque principalmente en las tres primeras disciplinas, *Business Modeling*, *Requirements* y *Analysis & Design* y la documentación generada demuestran que estas disciplinas fueron contempladas. Como en todo desarrollo evolutivo la comprensión del sistema así como la documentación cambia y se actualizan constantemente, el resultado alcanzado en Risonance versión 2.0 se muestra en el modelado presentado en los próximos ítems. (más detalle en el anexo B).

4.2.1. User Story 1

El modelado de la *user story 1* se muestra en la figura 16. Además el diagrama de actividades del 'registro de usuario' se relaciona con otros casos de uso como:

- Crear usuario
- Visualizar informes generales
- Enviar propuesta
- Hacer el seguimiento de la evaluación
- Aportar evidencias
- Evaluar una institución

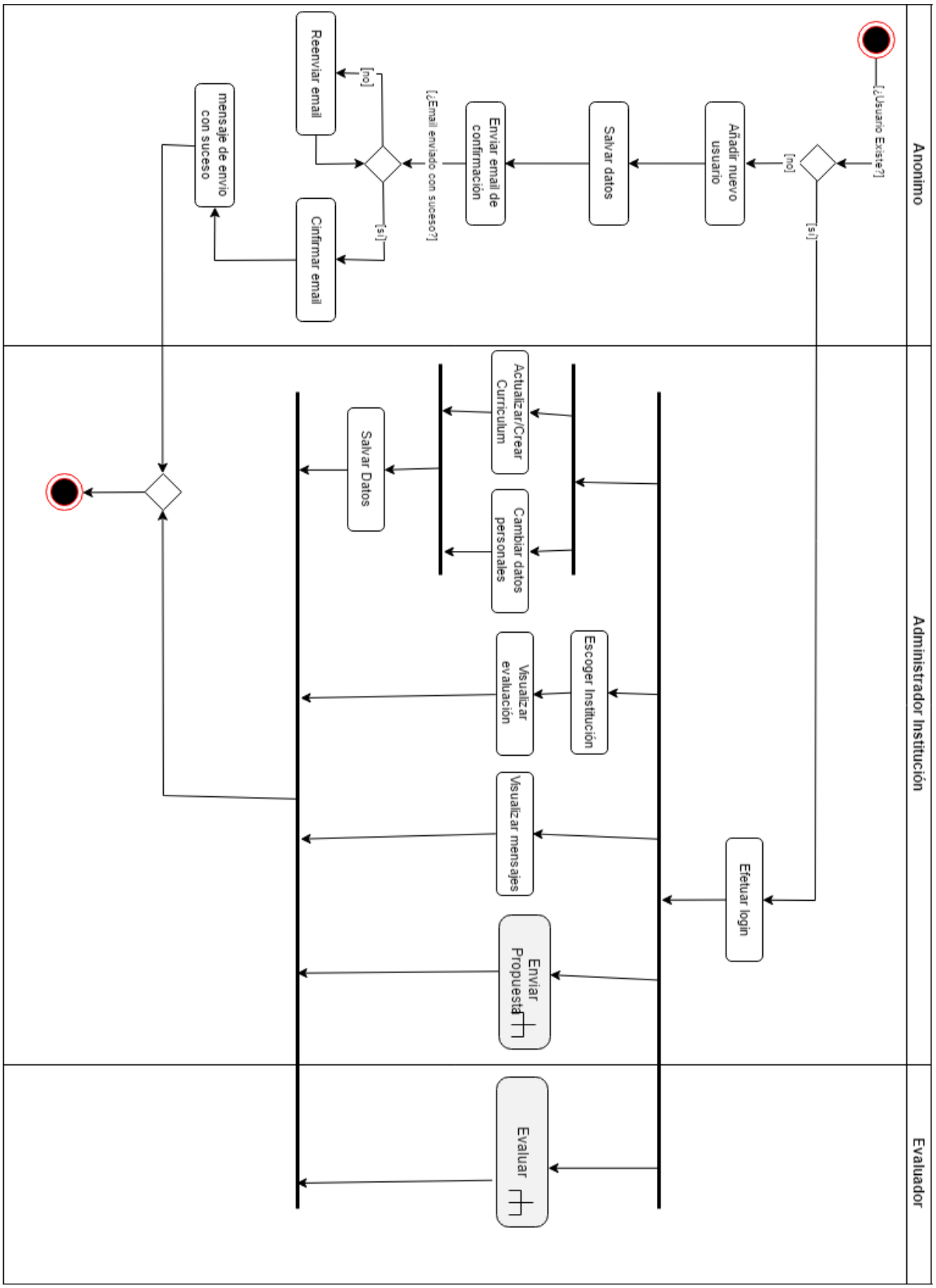


Figura 16: Diagrama de actividades descripción del login y creación de usuario.

Tras este modelado se crearon algunos diagramas de secuencia (descripciones en el Anexo A, ítem 4.c) y se ha aplicado al código PHP que muestra en las pantallas del software (figuras 17 y 18).

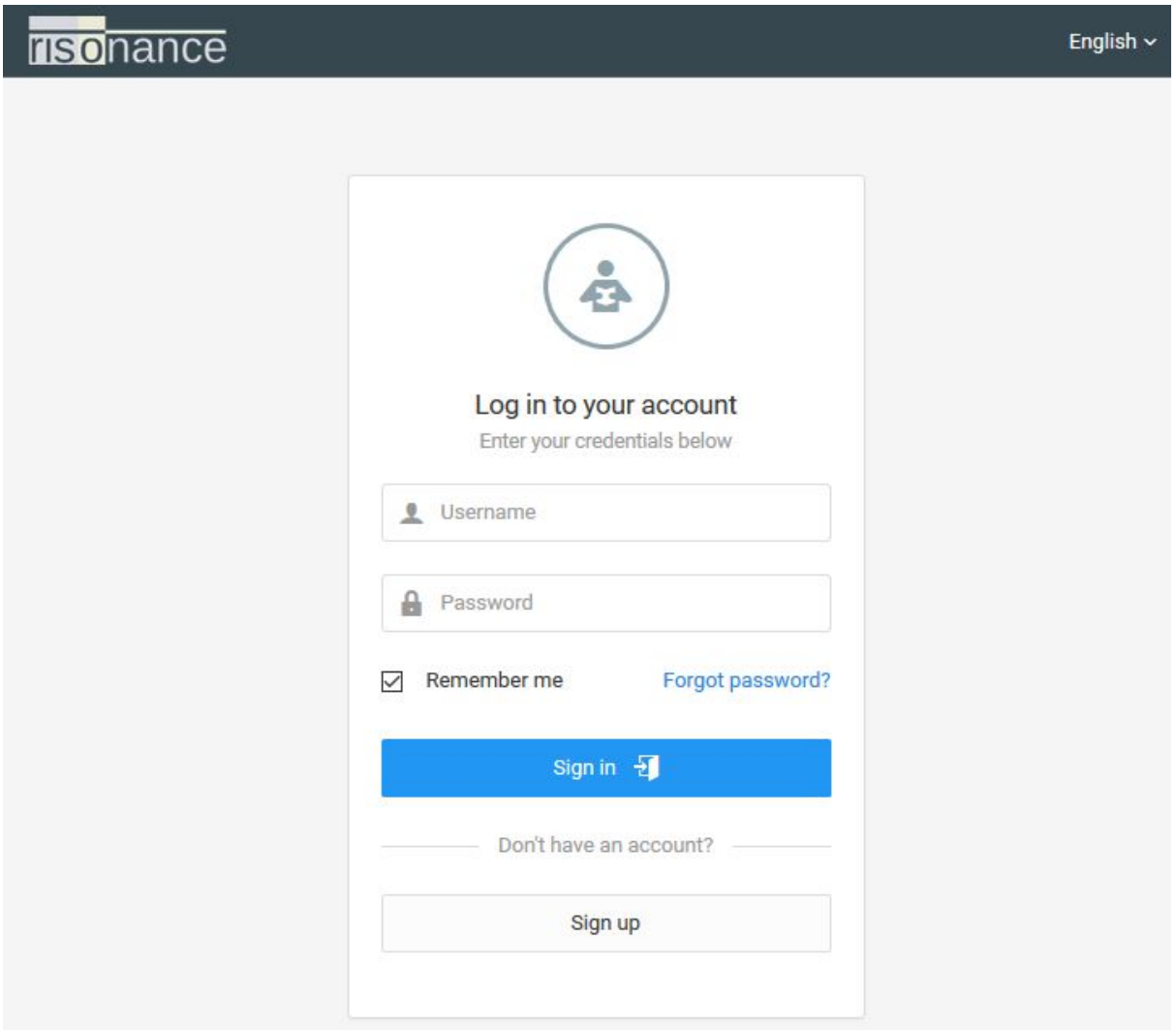


Figura 17: Pantalla de login.

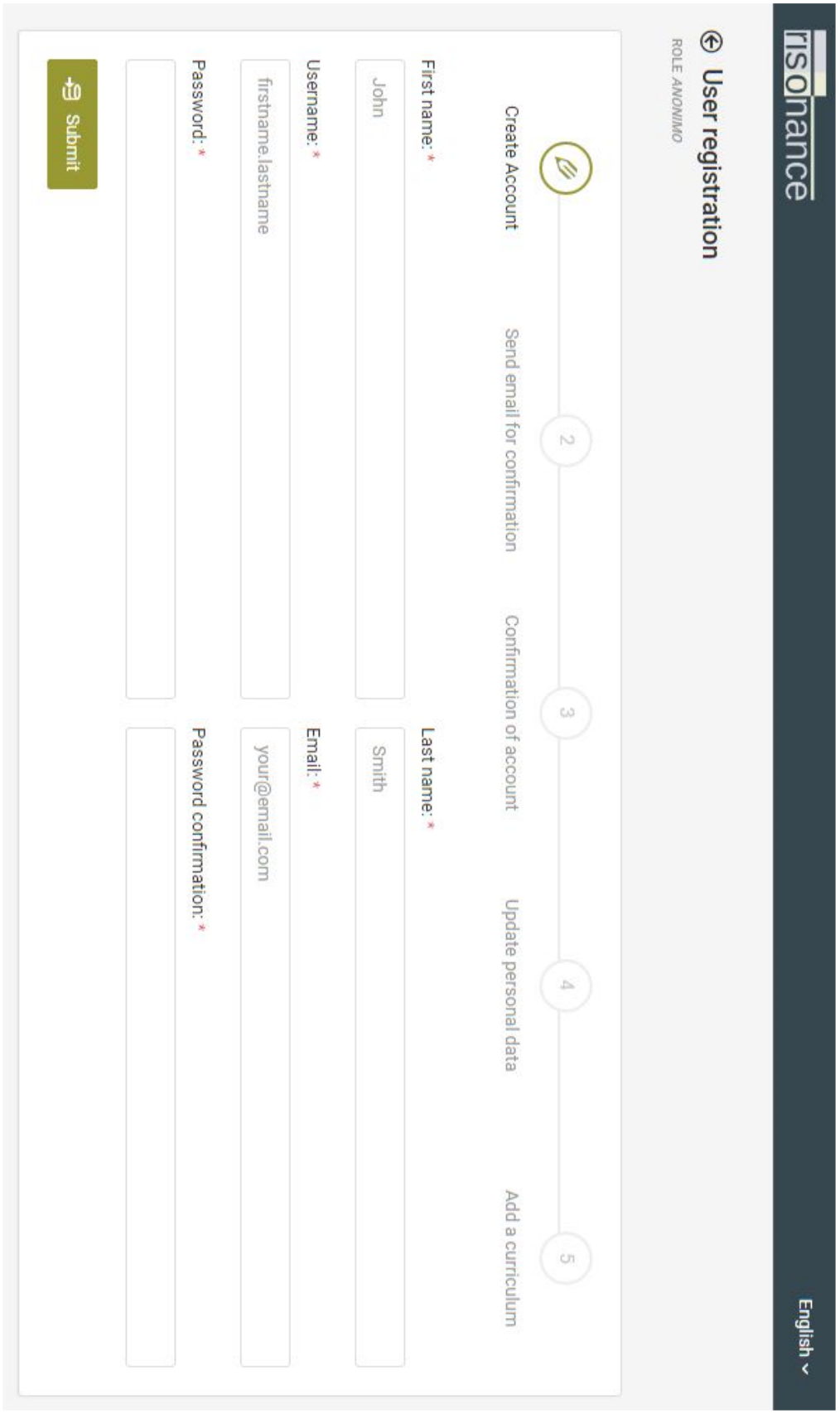


Figura 18: Pantalla de creación de usuario.

4.2.2. User Story 2 y 3

Las *user stories* 2 y 3 abordan los temas relacionados directamente con el caso de uso 'enviar propuesta' (figura 19), presentando su descripción. Los casos de uso asociados con este diagrama de actividades son:

- Crear Usuario
- Enviar Propuesta
- Invitar Coordinador
- Seleccionar dimensiones
- Invitar Evaluador
- Invitar Facilitador
- Configurar Módulos
- Aportar Evidencias

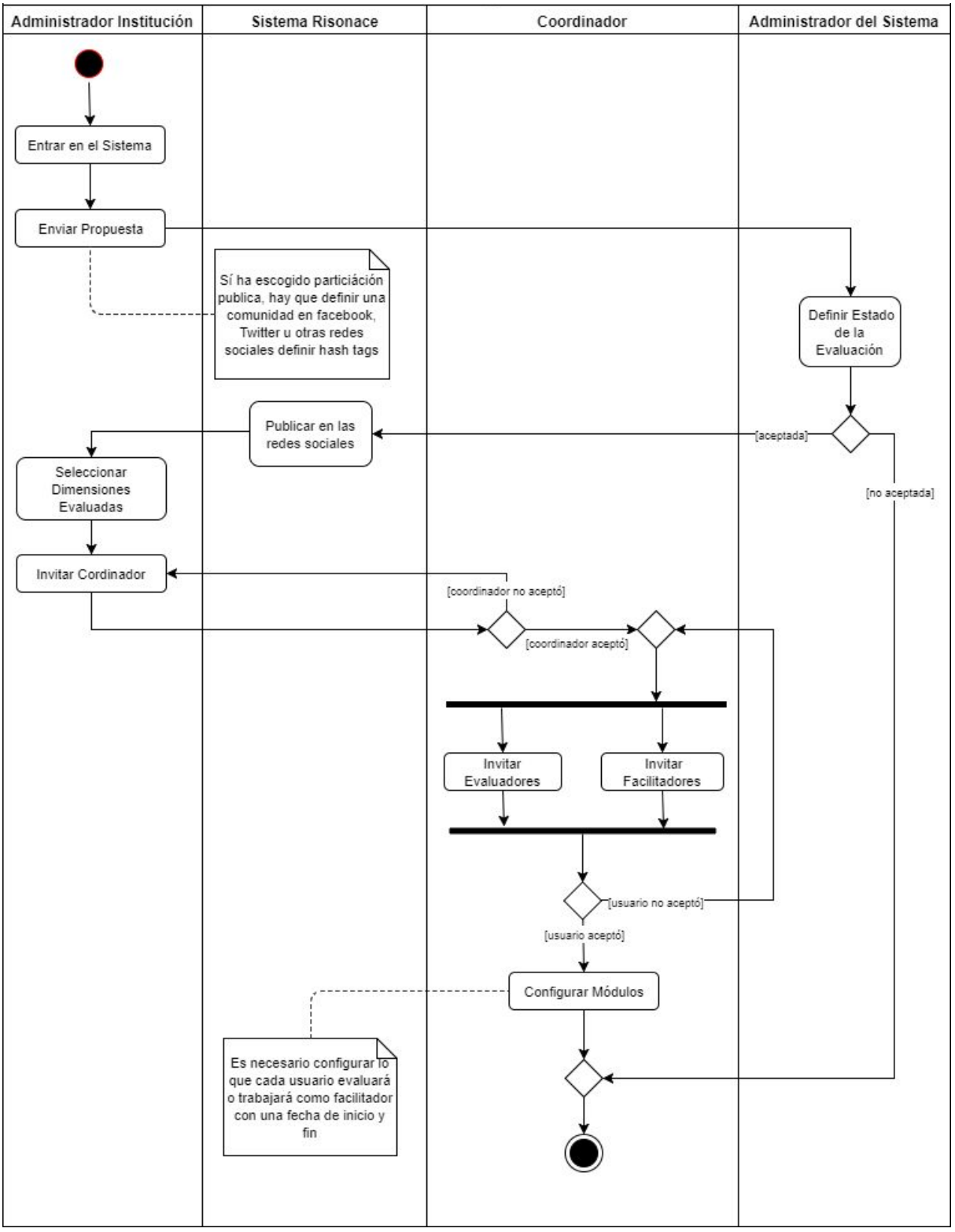


Figura 19: Diagrama de actividades enviar propuesta para evaluación.

Igualmente se estructuraron algunos diagramas de secuencia (Anexo A, ítem 4.a) que contribuyeron a una mejor comprensión de las funciones presentadas en el diagrama de actividades. En la secuencia se hizo la codificación en PHP y el resultado se muestra en la figura 20.

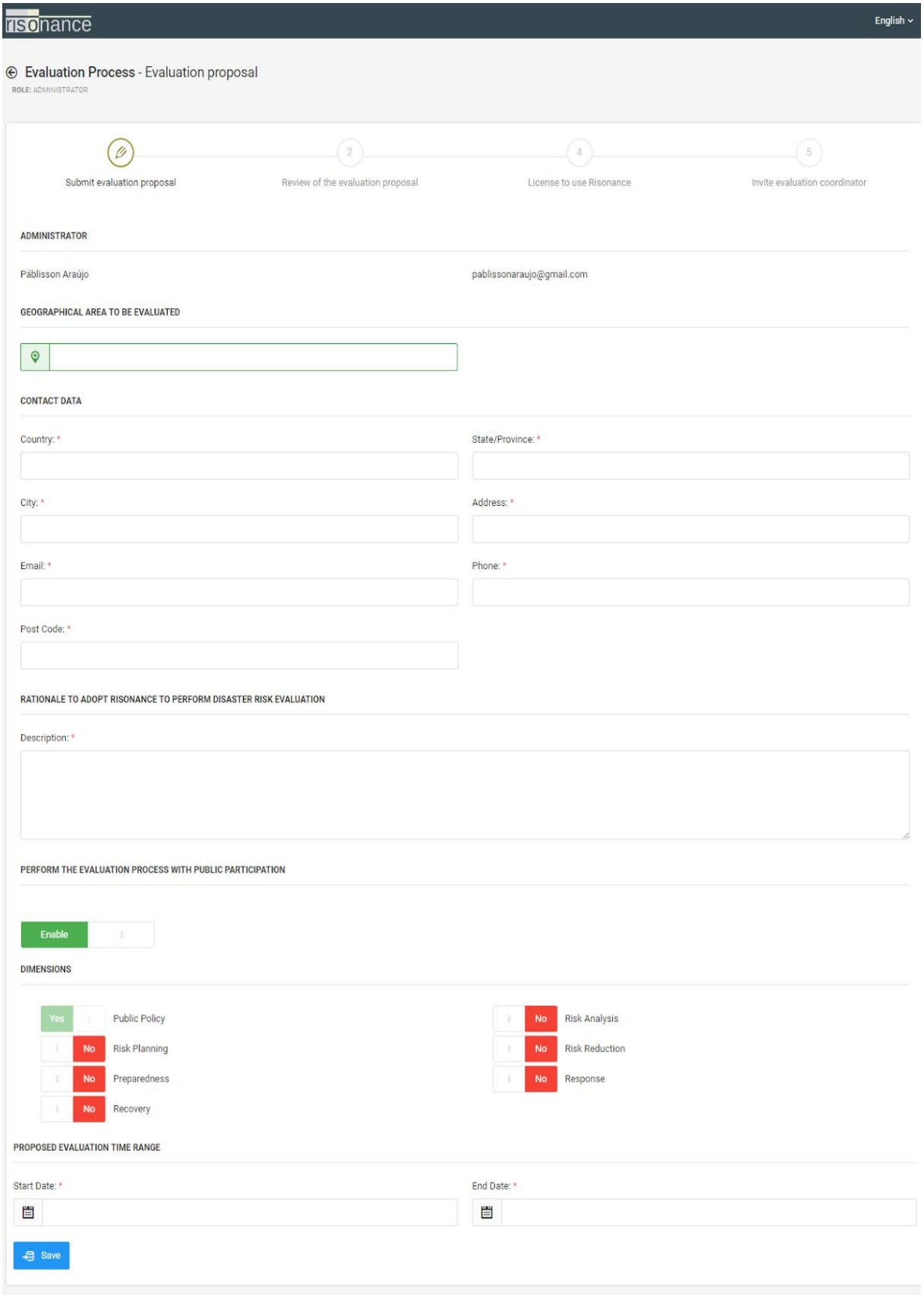


Figura 20: Pantalla para el registro de una nueva institución.

4.2.3. User Story 4

La *user story 4* contempla las actividades de la evaluación. Podemos ver el modelado de las actividades subdivididas por responsabilidades de cada actor (figura 21) y por las acciones del proceso de consenso (figura 22). Los diagramas se relacionan con los siguientes casos de uso:

- Evaluar institución
- Proponer consenso
- Aceptar consenso
- Rechazar consenso

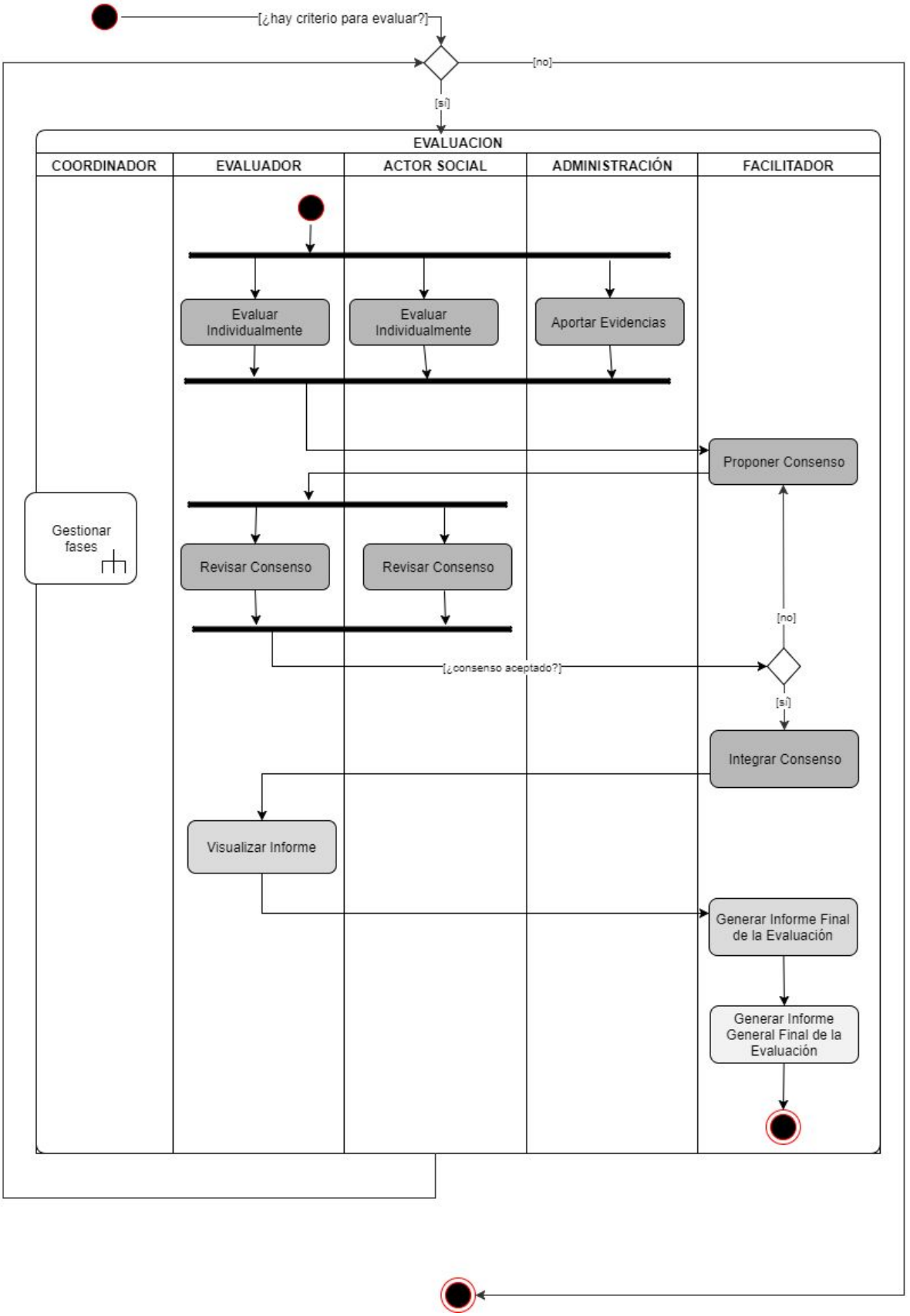


Figura 21: Diagrama de actividades de evaluación y logro del consenso.

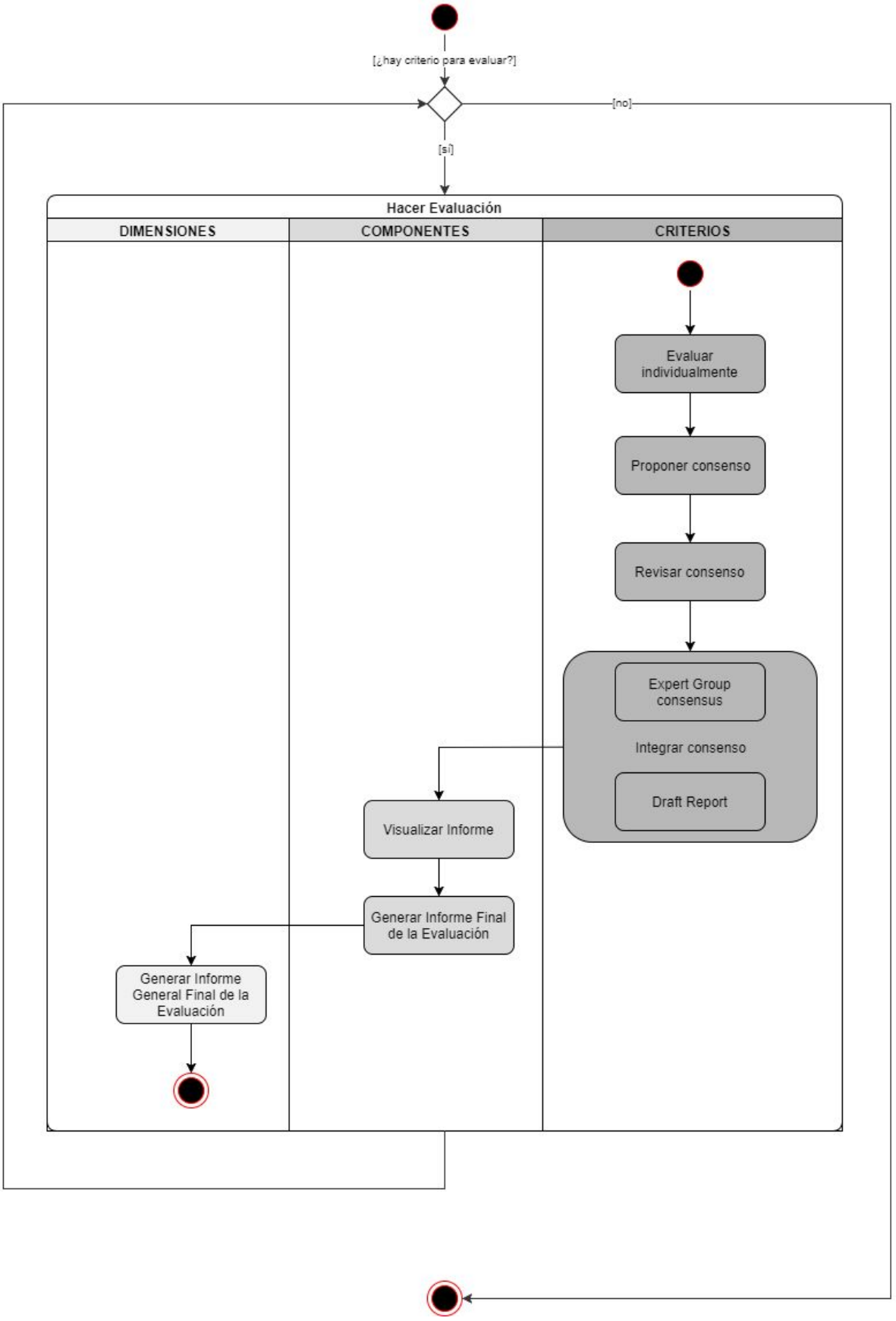


Figura 22: Diagrama de actividades para la producción del informe final de evaluación

La estructura de los diagramas de secuencia (Anexo A 4.b) y el código PHP utilizado se ilustran en la figura 23.

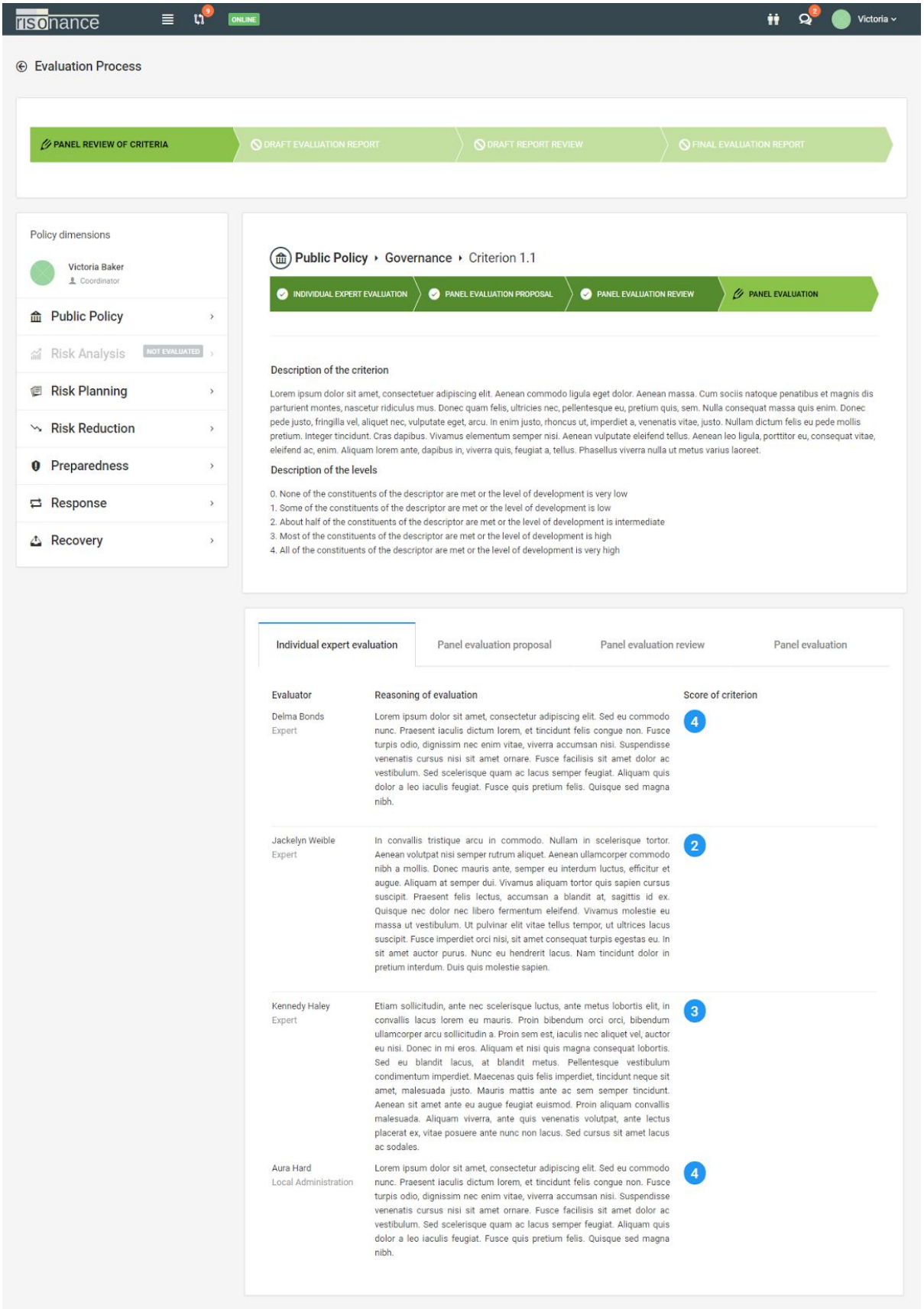


Figura 23: Pantalla de visualización de las evaluaciones.

4.2.4. Estructura final del modelado

Con el objetivo de obtener una mejor solución para las clases controladoras se han utilizado las tarjetas CRC (Anexo A, ítem 3) y también generado el diagrama de clase completo de la estructura (Figura 24). Igualmente se generó un documento que busca detallar los requisitos en tareas, de forma que se puedan implementar. Eso se ha aplicado a los conceptos de Scrum y generado el software por medio de interacciones.

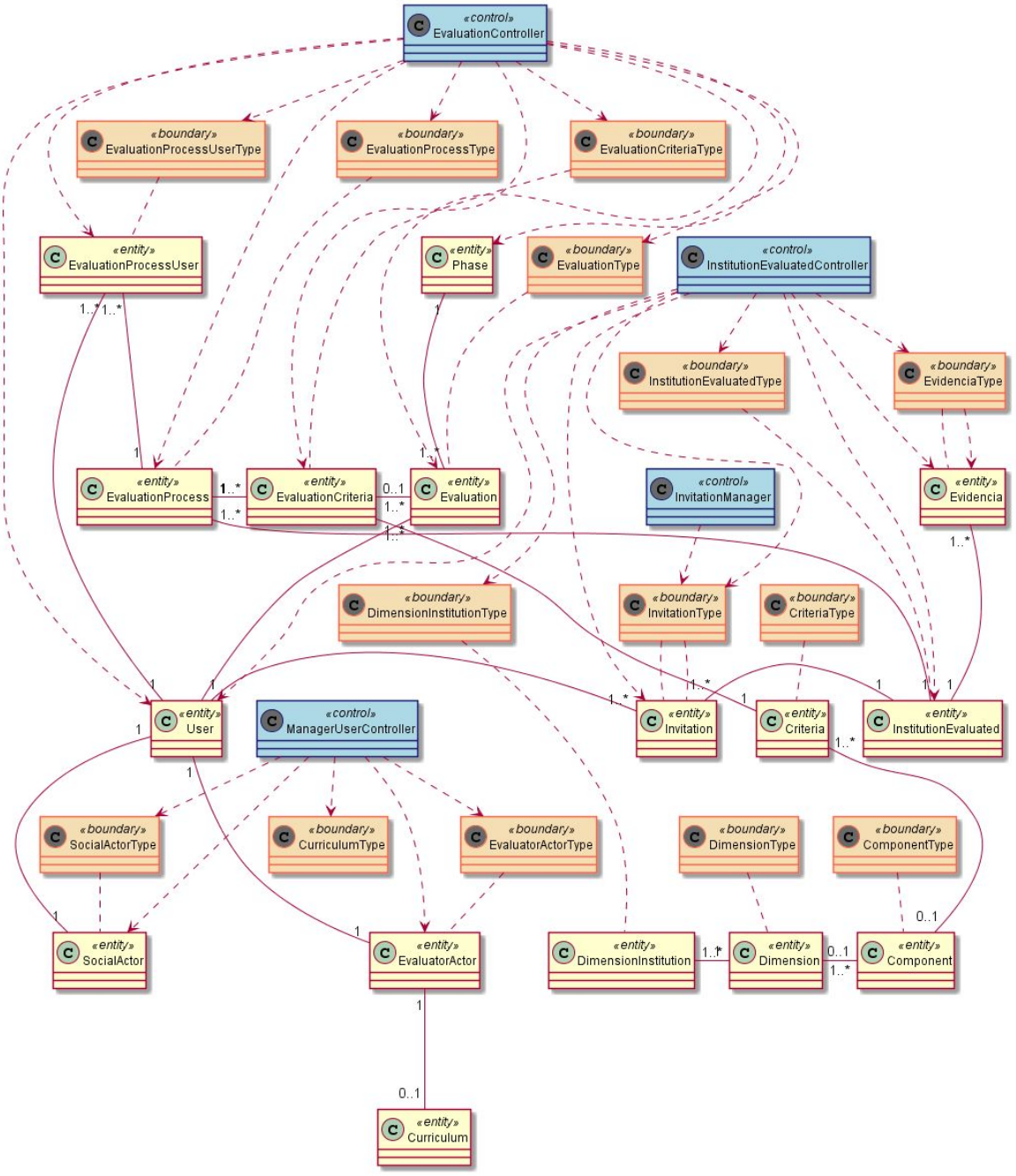


Figura 24: Diagrama de clases PHP de Risonance 2.0.

4.3. Estructura de desarrollo de software

Puesto que utilizamos UML para describir el problema y proponer una solución para el desarrollo, el entorno así como los conceptos, deben tener en cuenta el UML y viceversa. Así, los modelos generados están reflejados en un entorno de desarrollo basado en PHP con la utilización del framework Symfony.

Lo primero que vemos con el uso de Symfony es una estructura de archivos bien definida y organizada (figura 25) con una estructura de carpetas con app, bin, src, vendor y web[18]. Cabe destacar en esta estructura la separación del código backend y frontend, ya que el código frontend, HTML, CSS y javascript, queda en el archivo src/RisonanceBundle/Resources/views.

Risonance/
- app ---------➤ la configuración, plantillas y traducción del software
- src ----------➤ el código del proyecto
- vendor -----➤ los componentes de terceros usados en el proyecto
- web --------➤ el directorio web raíz

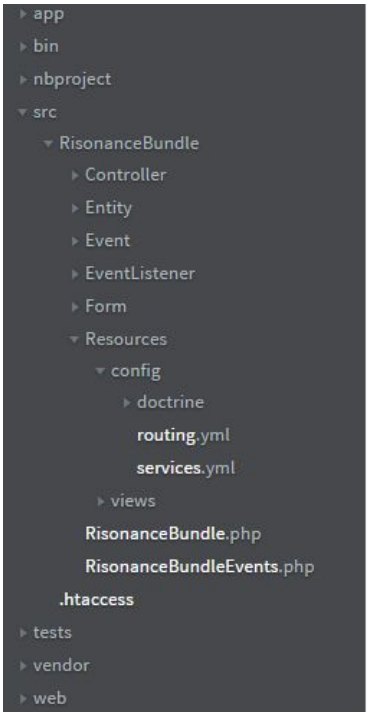


Figura 25: Estructura de archivos de la versión 2.0 de Risonance.

[18] Se puede ver más detalles al respecto de la estructura de Symfony en http://symfony.com/doc/2.8/quick_tour/the_architecture.html. Accedido en 25/07/2017

De manera general, en la versión 2.0 de Risonance se pueden ver algunas buenas prácticas que se materializan en el uso de namespace, código orientado a objetos, código divido por medio de componentes, y utilización de estructura MVC, entre otras. Un ejemplo aparece en la figura 26, que es un recorte aleatorio en el código de la clase KindUserInstitution.php. Además de hacerse presente las recomendaciones generales, se han adoptado buenas prácticas como el patrón *camelCase* para las definiciones de nombre de funciones y variables, además del patrón *CamelCase* para la definición de nombres de clases, que se encuentran en el PSR-1 y PSR-2.

```php
<?php

namespace RisonanceBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * KindUserInstitution
 */
class KindUserInstitution
{
    /**
     * @var integer
     */
    private $id;

    /**
     * @var string
     */
    private $roles;

    /**
     * @var \RisonanceBundle\Entity\InstitutionEvaluated
     */
    private $institutionEvaluated;

    /**
     * @var \RisonanceBundle\Entity\User
     */
    private $user;


    /**
     * Get id
     *
     * @return integer
     */
    public function getId()
    {
        return $this->id;
    }

```

Figura 26: El código de la Clase kindUserInstitution.php.

El código PHP informa de la estructura de codificación *backend*, así podemos observar que, al mismo tiempo que el código es orientado a objetos posee una estructura separada del *frontend*, que son los códigos HTML, CSS y javascript. Así, podemos ver la estructura de clases que refleja la base de datos (Figura 27). Por otra parte el código de la

clase InstitutionEvaluatedController.php (Figura 28) contiene todas las funcionalidades en relación a la institución.

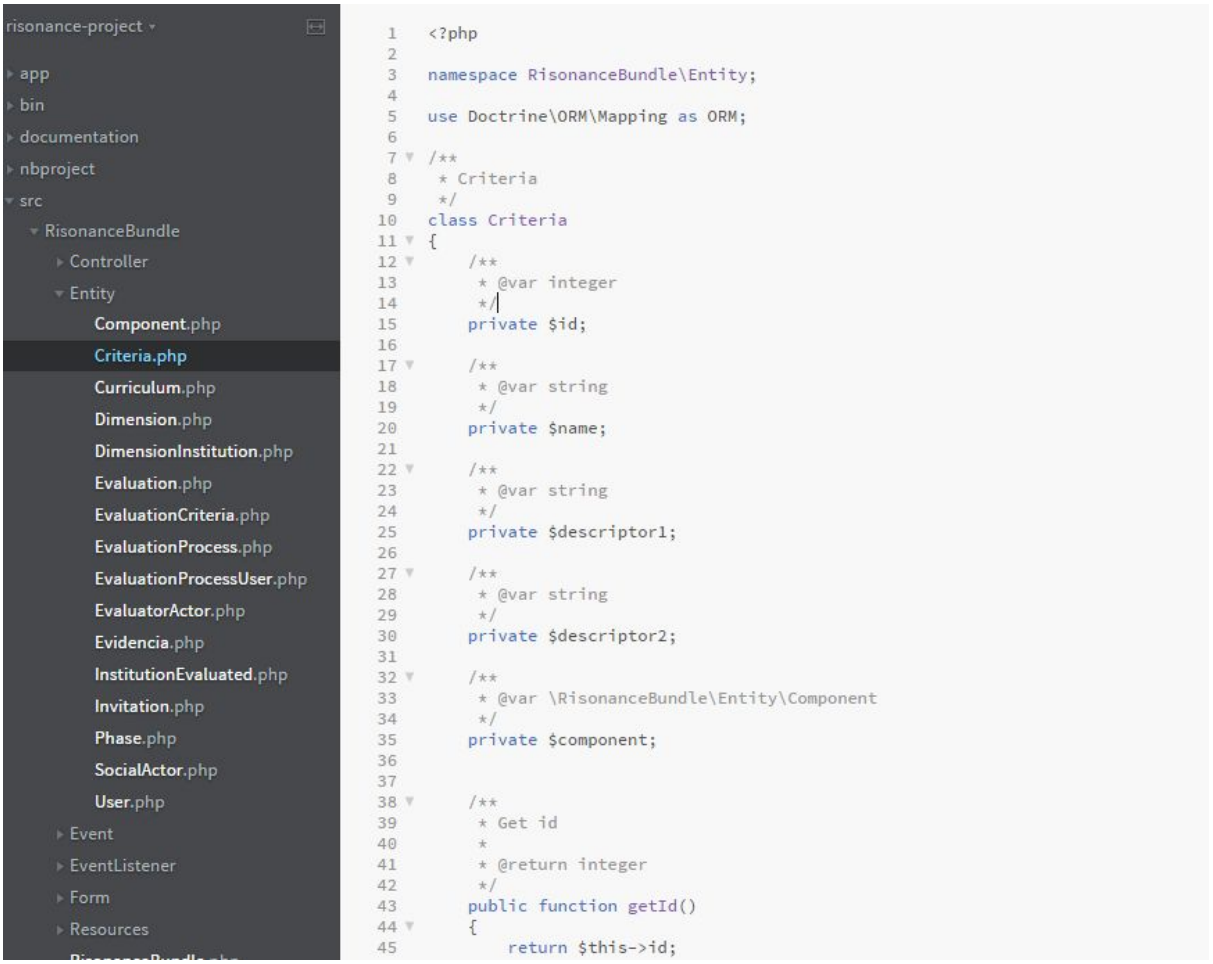
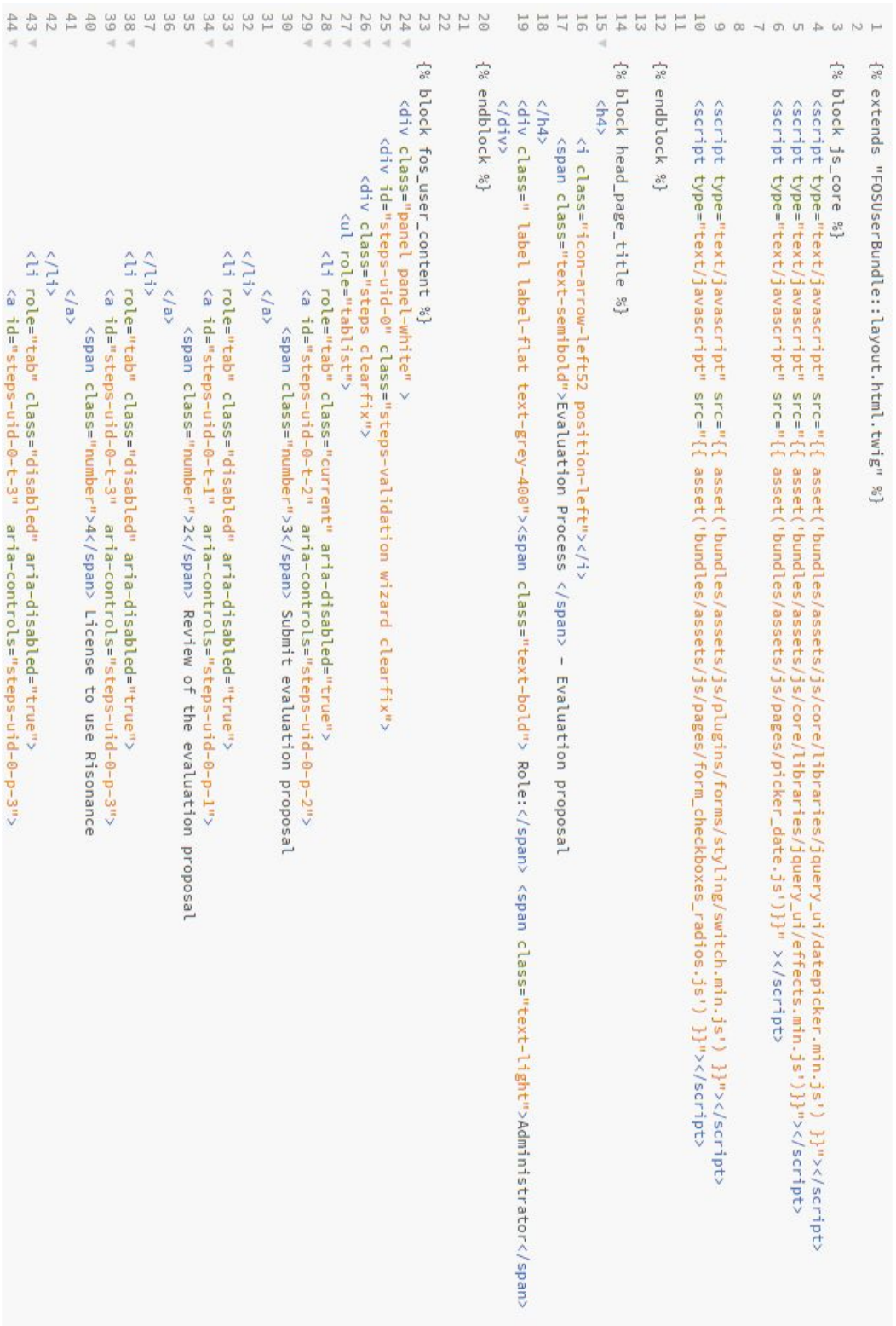


Figura 27: Estructura de clases que refleja la base de datos.

```php
    public function addInstitutionAction(Request $request){

        //verifica se el usuario esta conectado sin importar el tipo de permiso
        if (!$this->get('security.authorization_checker')->isGranted('IS_AUTHENTICATED_FULLY')) {
            throw $this->createAccessDeniedException();
        }
        $user = $this->getUser();

        $em = $this->getDoctrine()->getManager();

        $dimensionInstitution = new \RisonanceBundle\Entity\DimensionInstitution();
        $formDimensionInstitution = $this->createForm(\RisonanceBundle\Form\DimensionInstitutionType::class, $dimensionInstitution);

        $formDimensionInstitution->handleRequest($request);

        if($formDimensionInstitution->isSubmitted() && $formDimensionInstitution->isValid()){

            $institutionEvaluated = new InstitutionEvaluated();

            $institutionEvaluated->setName($formDimensionInstitution->get('institutionEvaluated')->get('name')->getData());
            $institutionEvaluated->setPhone($formDimensionInstitution->get('institutionEvaluated')->get('phone')->getData());
            $institutionEvaluated->setEmail($formDimensionInstitution->get('institutionEvaluated')->get('email')->getData());
            $institutionEvaluated->setAddress($formDimensionInstitution->get('institutionEvaluated')->get('address')->getData());
            $institutionEvaluated->setCountry($formDimensionInstitution->get('institutionEvaluated')->get('country')->getData());
            $institutionEvaluated->setState($formDimensionInstitution->get('institutionEvaluated')->get('state')->getData());
            $institutionEvaluated->setCity($formDimensionInstitution->get('institutionEvaluated')->get('city')->getData());
            $institutionEvaluated->setPostCode($formDimensionInstitution->get('institutionEvaluated')->get('postCode')->getData());
            $institutionEvaluated->setBeginDate($formDimensionInstitution->get('institutionEvaluated')->get('beginDate')->getData());
            $institutionEvaluated->setEndDate($formDimensionInstitution->get('institutionEvaluated')->get('endDate')->getData());
            $institutionEvaluated->setDescription($formDimensionInstitution->get('institutionEvaluated')->get('description')->getData());
            $institutionEvaluated->setPublicParticipation($formDimensionInstitution->get('institutionEvaluated')->get('publicParticipation')->getData());
            $institutionEvaluated->setApprovedProcess(false);
            $institutionEvaluated->setCoordinator($user);

            $dimensionAux = $formDimensionInstitution->get('dimension')->getData();

            foreach ($dimensionAux as $dim){
                $dimensionInstitution = new \RisonanceBundle\Entity\DimensionInstitution();
                $dimensionInstitution->setInstitutionEvaluated($institutionEvaluated);
                $dimensionInstitution->setDimension($dim);
                $em->persist($dimensionInstitution);
                $flush = $em->flush();
            }
```

Figura 28: El código de la Clase *InstitutionEvaluatedController.php*.

A continuación presentamos parte de un código *frontend* donde podemos ver la separación del código PHP (Figura 29).



```
{% extends "FOSUserBundle::layout.html.twig" %}

{% block js_core %}
    <script type="text/javascript" src="{{ asset('bundles/assets/js/core/libraries/jquery_ui/datepicker.min.js') }}"></script>
    <script type="text/javascript" src="{{ asset('bundles/assets/js/core/libraries/jquery_ui/effects.min.js') }}"></script>
    <script type="text/javascript" src="{{ asset('bundles/assets/js/pages/picker_date.js') }}" ></script>

    <script type="text/javascript" src="{{ asset('bundles/assets/js/plugins/forms/styling/switch.min.js') }}"></script>
    <script type="text/javascript" src="{{ asset('bundles/assets/js/pages/form_checkboxes_radios.js') }}"></script>

{% endblock %}

{% block head_page_title %}
    <h4>
        <i class="icon-arrow-left52 position-left"></i>
        <span class="text-semibold">Evaluation Process </span> - Evaluation proposal
    </h4>
    <div class=" label label-flat text-grey-400"><span class="text-bold"> Role:</span> <span class="text-light">Administrator</span>
    </div>
{% endblock %}

{% block fos_user_content %}
    <div class="panel panel-white" >
        <div id="steps-uid-0" class="steps-validation wizard clearfix">
            <div class="steps clearfix">
                <ul role="tablist">
                    <li role="tab" class="current" aria-disabled="true">
                        <a id="steps-uid-0-t-2"  aria-controls="steps-uid-0-p-2">
                            <span class="number">3</span> Submit evaluation proposal
                        </a>
                    </li>
                    <li role="tab" class="disabled" aria-disabled="true">
                        <a id="steps-uid-0-t-1"  aria-controls="steps-uid-0-p-1">
                            <span class="number">2</span> Review of the evaluation proposal
                        </a>
                    </li>
                    <li role="tab" class="disabled" aria-disabled="true">
                        <a id="steps-uid-0-t-3"  aria-controls="steps-uid-0-p-3">
                            <span class="number">4</span> License to use Risonance
                        </a>
                    </li>
                    <li role="tab" class="disabled" aria-disabled="true">
                        <a id="steps-uid-0-t-3"  aria-controls="steps-uid-0-p-3">
```

Figura 29: Código *frontend* con separación de código PHP.

Para finalizar es importante resaltar el alto nivel de seguridad que aportó Symfony, utilizando sus componentes y estructura, no solo por la encriptación de contraseñas e informaciones confidencial, sino también la visibilidad del código y URL por medio de los permisos de acceso que influyen en la visualización de información confidencial del software.

5. Conclusiones

La propuesta general de este trabajo ha sido reflexionar sobre la complejidad en el desarrollo de un software de calidad construido por un desarrollador, utilizando como objeto de estudio la versión 2.0 de Risonance. La solución de los problemas planteados inicialmente fueron propuestos a partir de diferentes metodologías de desarrollo de software, presentando resultados que nos llevan a concluir sobre las potencialidades reales de construir software de calidad con uno solo desarrollador.

Para reflexionar sobre las de potencialidades y límites se proponen tres ejes de análisis: la ingeniería de software, la codificación y la versión 2.0 de Risonance como aplicación de los conocimientos investigados.

Para la ingeniería de software la potencialidad está en las innumerables posibilidades de crear nuevas vías mediante la integración de metodologías existentes, superando las limitaciones específicas de cada una y potenciando sus puntos fuertes, aunque sean metodologías aparentemente contradictorias. Este trabajo ha demostrado que la integración de MDP y MA para solucionar los problemas técnicos identificados en Risonance 1.0 ha sido muy satisfactoria. Aún así, una debilidad en los entornos de alta interactividad y, en la nueva versión del Risonance, es la poca previsibilidad. Este es un punto importante ya que puede influir en la percepción del cliente sobre la calidad del software. Por consiguiente, es necesario buscar un término medio, haciendo el seguimiento del desarrollo del software para mantener un determinado nivel de control.

En cuanto a la codificación, la potencialidad se encuentra en la capacidad del equipo técnico además de las arquitecturas para construir códigos más limpios, objetivos y de mejor calidad. A partir de un código y una documentación organizados, una futura intervención en el software se realizará de forma más segura, tal como ha sido con la versión 2.0 de Risonance desarrollada en este trabajo. Desde el punto de vista del desarrollador, podemos afirmar que es posible desarrollar un software de calidad con uno solo desarrollador a partir del momento en el que el mismo demuestre la proactividad necesaria para aplicar el conocimiento técnico adecuado. Esto permitiría dar respuesta a los problemas planteados por el cliente, como se ha observado en la selección de las arquitecturas de desarrollo de la nueva

versión del software. Sin embargo, igualmente esto puede ser considerado un límite del proceso si el desarrollador no mantiene una disciplina y una autogestión adecuadas.

La organización general del software, presenta una documentación objetiva y un código organizado, lo que le permite evolucionar para la incorporación de nuevas tecnologías de manera más adecuada. Es decir, el código, la estructura y las nuevas tecnologías no actúan como barreras ante una nueva versión. Pero una debilidad de este entorno de desarrollo es el hecho de no mantener un equipo de desarrollo fijo, lo que pone en riesgo el trabajo. Eso se da porque cada nuevo desarrollador introduce sus propias características, bien como sus debilidades, y no da lugar a una evolución del conocimiento adquirido por su antecesor. Eso significa que, la madurez del sistema está vinculada a una única persona. Por lo tanto, aunque haya una documentación del software, ésta puede no ser suficiente para comunicar todo lo necesario, como ocurre en las MDP.

Los resultados alcanzados indican que manteniendo un enfoque disciplinado y constantes entregas de las funcionalidades, además de un ajuste en RUP acercándose a las MA ha permitido generar una documentación adecuada del proyecto y contribuir a una mejor gestión en escenarios de incertidumbre. De esta forma en la documentación generada, así como en el código, es posible identificar las técnicas de ingeniería de software utilizadas y en último término, la generación de un software seguro y con más calidad con un solo desarrollador.

6. Trabajos futuros

A partir del desarrollo del software Risonance 2.0 se plantean cuestiones que apuntan a nuevas investigaciones desde dos perspectivas, una teórico/conceptual y otra práctica.

Desde la primera perspectiva surge la cuestión de la comunicación más eficiente en proyectos de desarrollo con un solo desarrollador. Esa problemática nos lleva a pensar en la necesidad de investigar formas de mejoría de eficiencia en la comunicación, indicando debilidades y potencialidades en equipos que trabajan con MA. Igualmente interesante sería poder medir el grado de adecuación de un proyecto utilizando RUP con MA, y verificar cómo se desarrolla ese proyecto. Y, en relación a la codificación, se puede plantear la posibilidad de un testeo más eficiente y/o automatizado dentro de un entorno dinámico y de incertidumbre.

En relación a la práctica en el desarrollo de la versión 2.0 de Risonance, el proceso y los resultados nos llevan a preguntarnos cómo podría plantearse un consenso de forma automática, y cuáles serían las metodologías más adecuadas, ya que se trata del principal factor limitante, incrementando los tiempos del proceso.

Otra cuestión tiene que ver con la búsqueda de información y evidencias documentales para sostener la evaluación de un criterio y apoyar a los evaluadores, incluyendo las redes sociales y bases de datos gubernamentales y de organizaciones internacionales, así como la diversidad de formatos de documentos, imágenes y videos. Un problema complementario es la forma más eficaz de organizar el volumen de información que se puede incorporar a un proceso, así como en el conjunto de procesos sincrónicos e históricos. Por otro lado, esta acumulación de información es un activo ya que se convierte progresivamente en una rica base de conocimiento, en la cual se puede extraer información cada vez más valiosa para las distintas cuestiones que afecten a las políticas públicas.

7. Referencias bibliográficas

AMBLER, S. *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. New York: John Wiley & Sons, Inc, 2002. ISBN 0-471-20282-7.

ARLOW, J. y NEUSTADT, I. *UML 2*. Madrid: Addison Wesley, 2006. ISBN 84-415-2033-X.

BECK, K. y ANDRES, C. *Extreme Programming Explained: Embrace Change*. 2. ed. Boston: Addison-Wesley, 2005. ISBN 978-0321278654 .

BOEHM, B. The Spiral Model as a Tool for Evolutionary Acquisition. En *Crosstalk: The Journal of Defense Software Engineering*. Washington Dc. Mayo 2001, vol. 14, no. 5, pp. 4-11.

BOEHM, B. y TURNER, R. *Balancing agility and discipline: a guide for the perplexed*. 7. ed. Boston: Addison-Wesley, 2004. ISBN 0-321-18612-5.

CASTELLS, M. *The Impact of the Internet on Society: A Global Perspective* [en linea]. BBVA, 2014. [consulta en: 12 junio 2017]. Disponible en: <https://www.technologyreview.com/s/530566/the-impact-of-the-internet-on-society-a-global-perspective/>

COCKBURN, A. *Agile software development: the cooperative game*. 2. ed. Boston Ma: Pearson Education, 2006. ISBN 0-321-48275-1.

COHN, M. *Succeeding With Agile: Software Development Using Scrum*. Boston, Ma: Pearson Education, 2010. ISBN 0-321-57936-4.

COLLINS, A. E. Risk Governance and Development. En: FRA.PALEO, U. *Risk Governance: The Articulation of Hazard, Politics and Ecology*. 2. ed.. Berlin: Springer, 2015, pp. 477-479. ISBN 978-94-017-9327-8.

DAVOUDI, S. From Risk Society to Security Society. In: FRA.PALEO, U. *Risk Governance: The Articulation of Hazard, Politics and Ecology*. 2. ed. Berlin: Springer, 2015, pp. 465-467. ISBN 978-94-017-9327-8.

DICKINSON, T. y BURTON, I. The Disaster Epidemic: Research, Diagnosis, and Prescriptions. In: FRA.PALEO, U. *Risk Governance: The Articulation of Hazard, Politics and Ecology*. 2. ed. Berlin: Springer, 2015, pp. 185-199. ISBN 978-94-017-9327-8.

FOWLER, M., *MartinFowler.com* [en línea]. ©Martin Fowler, 2006.[consulta: 13 jun. 2017] Disponible en: <https://martinfowler.com/eaaDev/uiArchs.html>.

FOWLER, Martin. *UML distilled: a brief guide to the standard object modeling language*. 3. ed. Boston, Ma: Addison-Wesley, 2004. ISBN: 0321193687.

FRA.PALEO, U. Structure, Process, and Agency in the Evaluation of Risk Governance. In: FRA.PALEO, U. *Risk Governance: The Articulation of Hazard, Politics and Ecology*. 2. ed. Berlin: Springer, 2015, pp. 237-273. ISBN 978-94-017-9327-8.

GAMMA, E., HELM, R., JOHNSON, R. y VLISSIDES, J. *Design Patterns: elements of reusable object-oriented software*. Indianapolis, In: Addison-Wesley, 1994. ISBN: 0321700694.

JEFFRIES, R. *The Nature of Software Development: Keep It Simple, Make It Valuable, Build It Piece by Piece*. Raleigh: Pragmatic Bookshelf, 2015. ISBN-13: 978-1-941222-37-9

KROLL, P. y KRUCHTEN, P. *The Rational Unified Process Made Easy: a practitioner's guide to the RUP*. Boston: Addison-Wesley, 2003. ISBN 0321166094

KRUCHTEN, P. *The Rational Unified Process: an introduction.* 3. ed. Boston: Addison-Wesley, 2004. ISBN 0321197704.

LOCKHART, J. *Modern PHP: New Features and Good Practices*. Sebastopol, Ca: O'reilly Media, 2015. ISBN 978-1-491-90501-2.

LOCKHART, J. y STURGEON, P. *PHP: The Right Way* [en línea]. phptherightway.com [consulta en: 2 enero 2017]. Disponible en: <http://www.phptherightway.com/>.

Manifesto for Agile Software Development [en línea]. agilemanifesto.org, 2001. [consulta en: 26 diciembre 2016]. Disponible en: <http://agilemanifesto.org/>. .

MCENTIRE, D. A. An Evaluation of Risk Management and Emergency Management.: Relying on the Concept of Comprehensive Vulnerability Management for an Integrated Perspective. In: FRA.PALEO, U. *Risk Governance: The Articulation of Hazard, Politics and Ecology*. 2. ed. Berlin: Springer, 2015, pp. 201-209. ISBN 978-94-017-9327-8.

POTENCIER, F. *What is Symfony2.* Fabien Pontencier, 2007-2017. [consulta en: 01 julio 2017]. Disponible en: <http://fabien.potencier.org/what-is-symfony2.html>.

PRESSMAN, R. S.. *Ingeniería del software: un enfoque práctico*. 7. ed. México D.F: Mcgraw-Hill, 2010. ISBN: 978-607-15-0314-5.

REEVES, C. A. y BEDNAR, D. A.. Defining quality: alternatives and implications in *Academy Of Management Review*. Julio 1994, vol. 19, no 3, pp. 419-445.

ROORDA, N., RAMMEL, C., WAARA, S. y FRA.PALEO, U. *AISHE 2.0 Manual. Assessment instrument for sustainability in higher education.* 2009. [consulta en: 25 septiembre 2017]. Disponible en: <https://niko.roorda.nu/books/aishe/>.

RUBIN, K. S.. *Essential Scrum: a practical guide to the most popular agile process*. Michigan: Addison-Wesley, 2012. ISBN 0-13-704329-5.

RUMBAUGH, J., JACOBSON, I. y BOOCH, G. *The Unified Modeling Language reference manual*. 2. ed. Boston, Ma: Pearson Education, 2005. ISBN 0-321-24562-8.

RUSSELL, S. J. y NORVIG, P. *Artificial Intelligence: A Modern Approach.* 3. ed. Upper Saddle River: Prentice Hall, 2010. ISBN 0-13-604259-7.

SOMMERVILLE, I. *Ingeniería del software*. 9. ed. México: Pearson Educación, 2011. ISBN 978-607-32-0603-7.

Symfony [en línea]. symfony.com, 2017. [consulta: 05 julio 2017]. Disponible en: <http://symfony.com/what-is-symfony>.

THE PHP GROUP. *PHP* [en línea]. The PHP Group, 2001-2017 [consulta: 10 junio 2017]. Disponible en: <php.net/manual/es/history.php.php>.

ZANDSTRA, M. *PHP Objects, Patterns, and Practice*. 5. ed. Liverpool: Spring, 2016. ISBN-13 (pbk): 978-1-4842-1995-9

ANEXO A

Versión 2.0 de Risonance: Modelado del sistema

1. Expansión de los principales casos de uso

Caso de uso: Crear Usuario
ID: 01
Breve descripción: Este caso de uso permite que un usuario anónimo en Internet haga un registro de un nuevo usuario poniendo su nombre, apellido, correo, nombre de usuario (login) y contraseña. Para entrar en el sistema el usuario deberá confirmar su registro por medio de un correo, por lo que evitaría el registro de correos inválidos. Caso el usuario no confirme el recibimiento del correo su usuario será eliminado automáticamente. El usuario sólo podrá registrar una institución si ha confirmado sus datos.
Actores Principales: Usuario Anónimo
Actores Secundarios: ninguno
Precondiciones: ninguna
Flujo Principal: 1. Usuario rellena los datos iniciales como nombre, contraseña, username y email 2. Usuario selecciona si el usuario es un actor social, o representa alguna institución, empresa o administración pública o si el usuario es un evaluador que en este caso puede ser un evaluador experimentado o un facilitador. 3. Usuario salva los datos en la base de datos 4. Sistema envía un mensaje al correo para confirmar datos 5. Usuario confirma el recibimiento del correo 6. Sistema confirma el registro 7. Sistema habilita usuario para utilizar el sistema
Postcondiciones: 1. Loguear en el sistema 2. Actualizar los datos del usuario a. Si el usuario fuese un actor social y en este caso representar un grupo i. deberá actualizar el contacto de la institución bien como su dirección,

teléfono y todos los datos personales necesarios ii. salva los datos b. Si el usuario fuese un evaluador i. actualiza todos sus datos personales como teléfono personal, dirección teléfono de trabajo, localidad, país, etc. ii. Añade curriculum donde se puede poner su experiencia, lenguas de dominio y direccionar su perfil
Flujos alternativos: ninguno

Caso de uso: Visualizar informes generales
ID: 02
Breve descripción: El usuario anónimo, en el caso de no estar registrado en el sistema, podrá acceder a informes generales de todas las evaluaciones que permiten una evaluación participativa
Actores Principales: Usuario Anónimo
Actores Secundarios: Ninguno
Precondiciones: ninguna
Flujo Principal: 1. Acceder al sistema 2. Visualizar todos los procesos de evaluaciones que sean públicos 3. Visualizar localidades que están en un proceso de evaluación 4. Visualizar localidades que han finalizado el proceso de evaluación 5. Visualizar localidades que están empezando el proceso de evaluación
Postcondiciones: 1. Clicar en participar de evaluación pública en un proceso de evaluación 2. Recibe un correo 3. Confirma correo 4. Crear Usuario 5. Loguear 6. Confirma participación
Flujos alternativos: ninguno

Caso de uso: Enviar Propuesta
ID: 03
Breve descripción: El Administrador de la institución registra la institución a ser evaluada y proporciona las informaciones y documentaciones necesarias para que eso ocurra. El usuario solo puede invitar al coordinador después que la institución sea aceptada.
Actores Principales: Usuario Administrador de la institución
Actores Secundarios: Ninguno
Precondiciones: 1. Haber creado un usuario 2. Estar logueado en el sistema
Flujo Principal: 1. Insertar datos de la institución 2. Verificar la validez del correo 3. Seleccionar dimensiones a ser evaluadas 4. Añadir evidencias 5. Invitar Coordinador 6. Sistema envía correo al usuario coordinador
Postcondiciones: 1. Coordinador acepta invitación 2. Coordinador invita evaluador 3. Coordinador invita facilitador
Flujos alternativos: ninguno
Casos de uso asociados: -Seleccionar dimensiones a ser evaluadas -Acompañar el seguimiento de la evaluación -Invitar coordinador

Caso de uso: Acompañar el seguimiento de la evaluación
ID: 04
Breve descripción: Mantener el cumplimiento de los plazos, y añadir el conocimiento que haga falta editando los datos que sean necesarios. También se podrá visualizar si los evaluadores han cumplido los plazos establecidos.

Actores Principales: Usuario Administrador de la institución
Actores Secundarios: Ninguno
Precondiciones: 1. Haber creado un usuario 2. Estar logueado 3. Haber registrado una institución 4. Haber invitado Coordinador 5. Haber invitado Evaluador 6. Haber invitado Facilitador
Flujo Principal: 1. Clicar en una institución 2. Visualizar estado de todos los criterios 3. Visualizar las evaluaciones futuras 4. Visualizar evaluaciones en retraso 5. Visualizar todos los participantes 6. Visualizar redes sociales conectadas en el proceso de evaluación 7. Visualizar hashtags y detalles asociados 8. Puede eliminar un usuario del proceso y describir una justificación 9. Puede evaluar los usuarios 10. Puede enviar mensajes personales a los usuarios
Postcondiciones: ninguno
Flujos alternativos: ninguno

Caso de uso: Invitar Evaluadores
ID: 05
Breve descripción: El coordinador hace un envío de correo invitando a alguien a participar del proceso de evaluación, en el caso de que no tenga un registro en el sistema recibirá la invitación y deberá hacer un registro para aceptar la invitación, en el caso contrário la invitación expirará en una semana. Y en el caso de que el usuario ya esté en el sistema recibirá una invitación por medio del sistema, además del correo. La invitación tiene validez de una semana, en el caso en que el usuario no conteste la invitación será removida automáticamente.
Actores Principales: Usuario Coordinador
Actores Secundarios: Ninguno

Precondiciones: 1. Haber creado un usuario 2. Estar logueado 3. Haber registrado una institución
Flujo Principal: 1. Enviar correo para evaluador 2. Evaluador acepta/rechaza la invitación (expira en una semana) 3. Coordinador recibe un mensaje en el sistema con el estado de la invitación
Postcondiciones: 1. Usuario Evaluador crea usuario 2. Usuario Evaluador evalúa criterio 3. Coordinador visualiza evaluación
Flujos alternativos: ninguno

Caso de uso: Definir permisos para las dimensiones
ID: 06
Breve descripción: Después de que cada usuario acepte participar de las evaluaciones, como facilitador o como evaluador, el coordinador definirá cuales son las dimensiones que este evaluará. Además de eso el coordinador deberá definir una fecha para que se quede hecho. El coordinador deberá estar atento a fecha inicial y final que fue puesta en el registro de la institución.
Actores Principales: Usuario Coordinador
Actores Secundarios: ninguno
Precondiciones: 1. Haber invitado Evaluador 2. Haber invitado Facilitador 3. Usuario Evaluador/Facilitador haber aceptado participar de la evaluación
Flujo Principal: 1. Define criterio a ser evaluado 2. Define Fecha de inicio y fin
Postcondiciones: 1. Generar informes
Flujos alternativos: ninguno

Caso de uso: Gestionar Proceso de evaluación
ID: 07
Breve descripción: El usuario coordinador podrá visualizar las evaluaciones con criterios retrasados, criterios que van empezar, cuál ha sido la última vez que el evaluador ha accedido el sistema, usuario podrá eliminar a cualquier evaluador del proceso y justificar la acción, podrá aceptar ser actor social para participar de la evaluación además de visualizar todas las evaluaciones aunque el proceso esté inconcluso.
Actores Principales: Usuario Coordinador
Actores Secundarios: Ninguno
Precondiciones: ninguna
Flujo Principal: 1. Visualizar los distintos proceso de evaluación 2. Visualizar estado de todos los criterios 3. Visualizar las evaluaciones futuras 4. Visualizar evaluaciones con retraso 5. Visualizar todos los participantes 6. Visualizar redes sociales conectadas en el proceso de evaluación 7. Visualizar hashtags y detalles asociados 8. Puede remover un usuario del proceso y describir una justificación 9. Puede evaluar los usuarios 10. Puede enviar mensajes personales a los usuarios
Postcondiciones: niguna
Flujos alternativos: ninguno

Caso de uso: Gestionar sistema
ID: 08
Breve descripción: El responsable para gestionar el sistema recibirá un mensaje en la plataforma todas las veces en las que alguien pide envío de solicitud de evaluación o lo que consideramos un registro de una institución. Además podrá ver cuales son las instituciones que están siendo evaluadas, cuales ya terminaron la evaluación y cuáles son las que están con retraso.

Actores Principales: Usuario Administrador del Sistema
Actores Secundarios: Ninguno
Precondiciones: ninguno
Flujo Principal: 1. Aceptar institución 2. Rechazar institución 3. Generar informes 4. Visualizar los distintos procesos de evaluación 5. Visualizar los territorios que fueron evaluados 6. Visualizar todos usuarios del sistema
Postcondiciones: niguna
Flujos alternativos: ninguno
Casos de uso asociados: -Aceptar institución -Rechazar institución

Caso de uso: Evaluar Institución
ID: 09
Breve descripción: Un evaluador propone una evaluación y la guarda. Después que la evaluación sea salvada se quedará disponible para todos los que tienen permiso para acceder a la visualización. El evaluador no puede editar la evaluación después que ser salvada.
Actores Principales: Usuario Evaluador Usuario Facilitador Usuario Actor Social
Actores Secundarios: Usuario Coordinador Usuario Administrador del Sistema
Precondiciones: 1. Tener un usuario registrado 2. Estar logueado 3. Haber sido invitado para una evaluación o haber sido aceptado para participar en una evaluación

Flujo Principal: 1. Usuario Evaluador Analiza las evidencias 2. Usuario Evaluador escribe un texto con una valoración numérica 3. Usuario Facilitador genera una propuesta de consenso 4. Usuario Evaluador contesta al consenso propuesto
Postcondiciones: Genera un informe del criterio evaluado
Flujos alternativos: ninguno
Casos de uso asociados: -Aceptar consenso -Rechazar consenso -Visualizar evaluación -Hacer consenso

Caso de uso: Hacer pedido para participar de la evaluación
ID: 10
Breve descripción: El actor social puede hacer un pedido para participar en la evaluación y proponer una evaluación como un evaluador. El actor social aunque sea considerado por el sistema como un usuario, de hecho representa un grupo de personas.
Actores Principales: Usuario Anónimo Usuario Actor Social
Actores Secundarios: Usuario Coordinador
Precondiciones: 1. Visualizar los procesos de evaluaciones abiertos para la evaluación participativa 2. Escoger un proceso de evaluación 3. Enviar un pedido para participar del proceso 4. Aguardar respuesta
Flujo Principal: 1. Recibir contestación al respecto del pedido realizado
Postcondiciones: niguna
Flujos alternativos: ninguno

Casos de uso asociados: Contestar pedido para evaluar

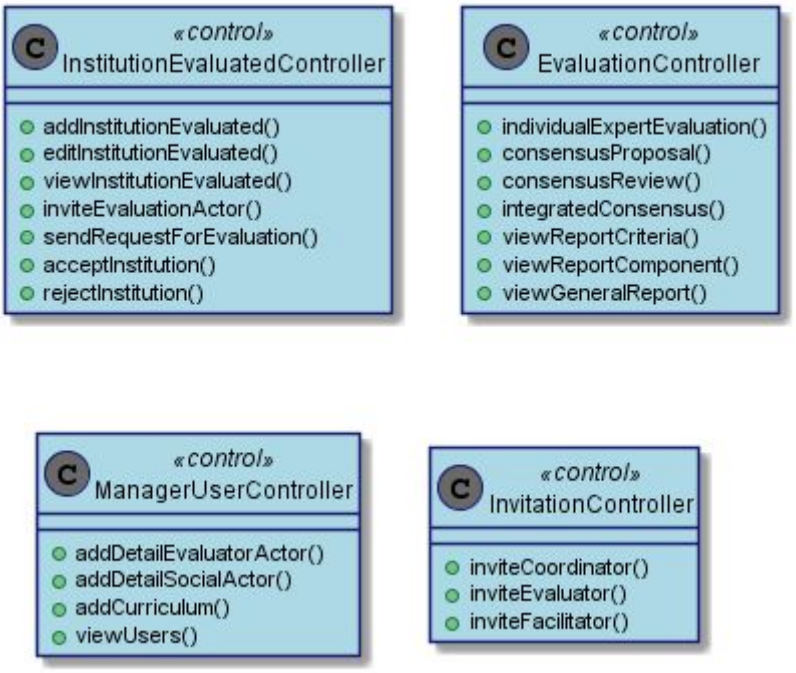
2. Tarjetas CRC

InstitutionEvaluatedController	
Gestionar institución	-DimensionInstitution -DimensionInstitutionType -InstitutionEvaluated -InstitutionEvaluatedType -Invitation -InvitationType -Evidencia -EvidenciaType -Dimension

InvitationManager	
Gestionar las invitaciones del sistema	-Invitation -InvitationType

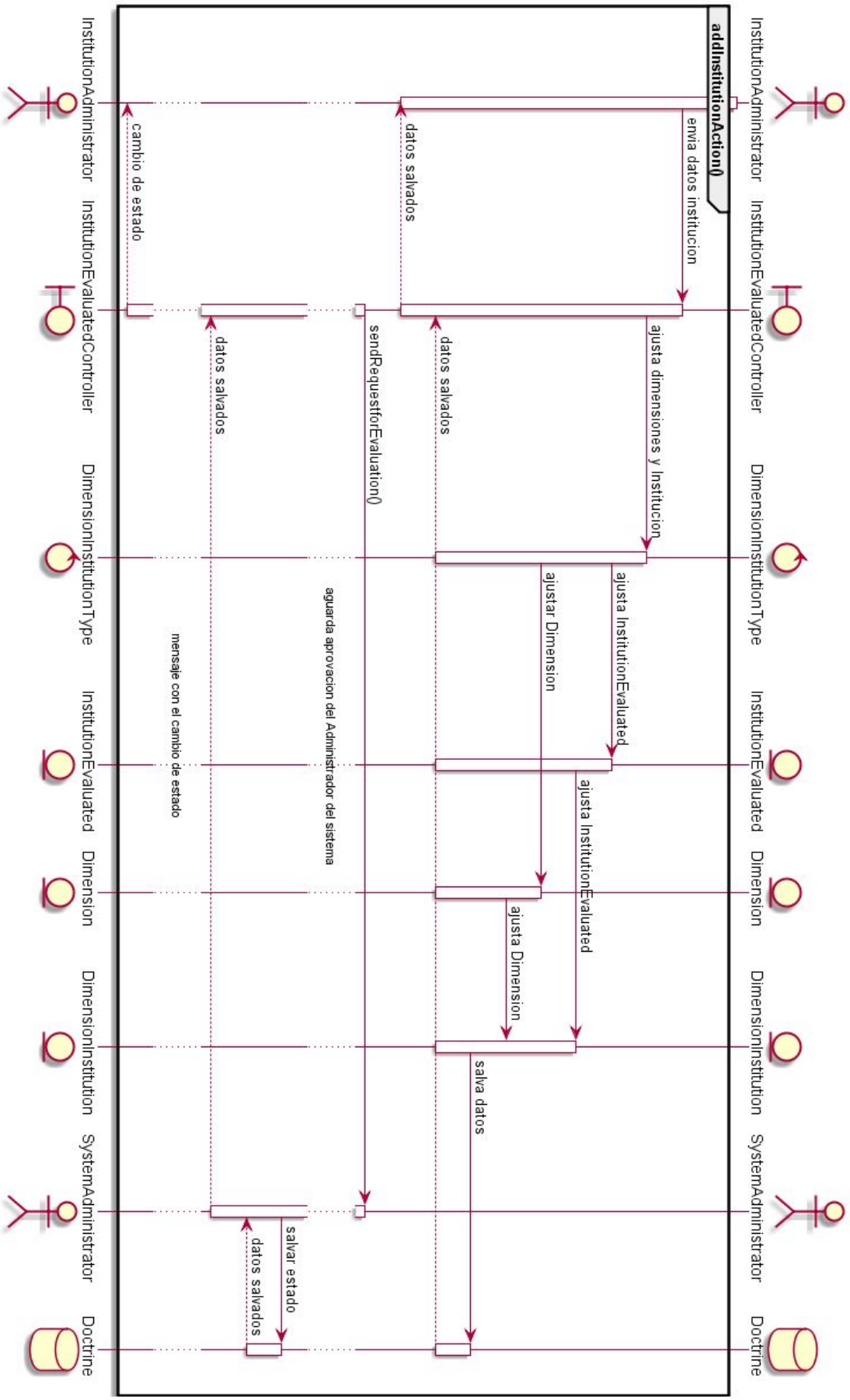
EvaluationController	
Gestionar el proceso de evaluación	-EvaluationType -Evaluation -EvaluationProcessUserType -EvaluationProcessType -EvaluationCriteriaType -EvaluationProcessUser -EvaluationProcess -EvaluationCriteria -EvaluationProcessUser

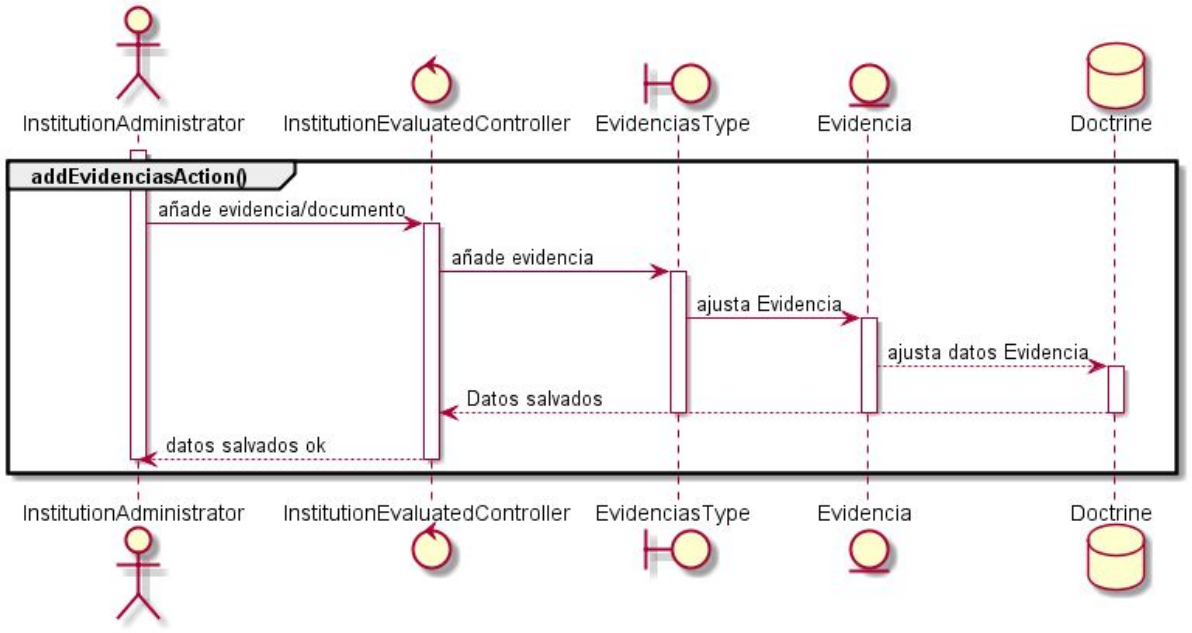
ManagerUserController	
Gestionar datos adicionales de usuarios	-SocialActorType -EvaluatorActorType -CurriculumType -SocialActor -EvaluationActor -Curriculum -User

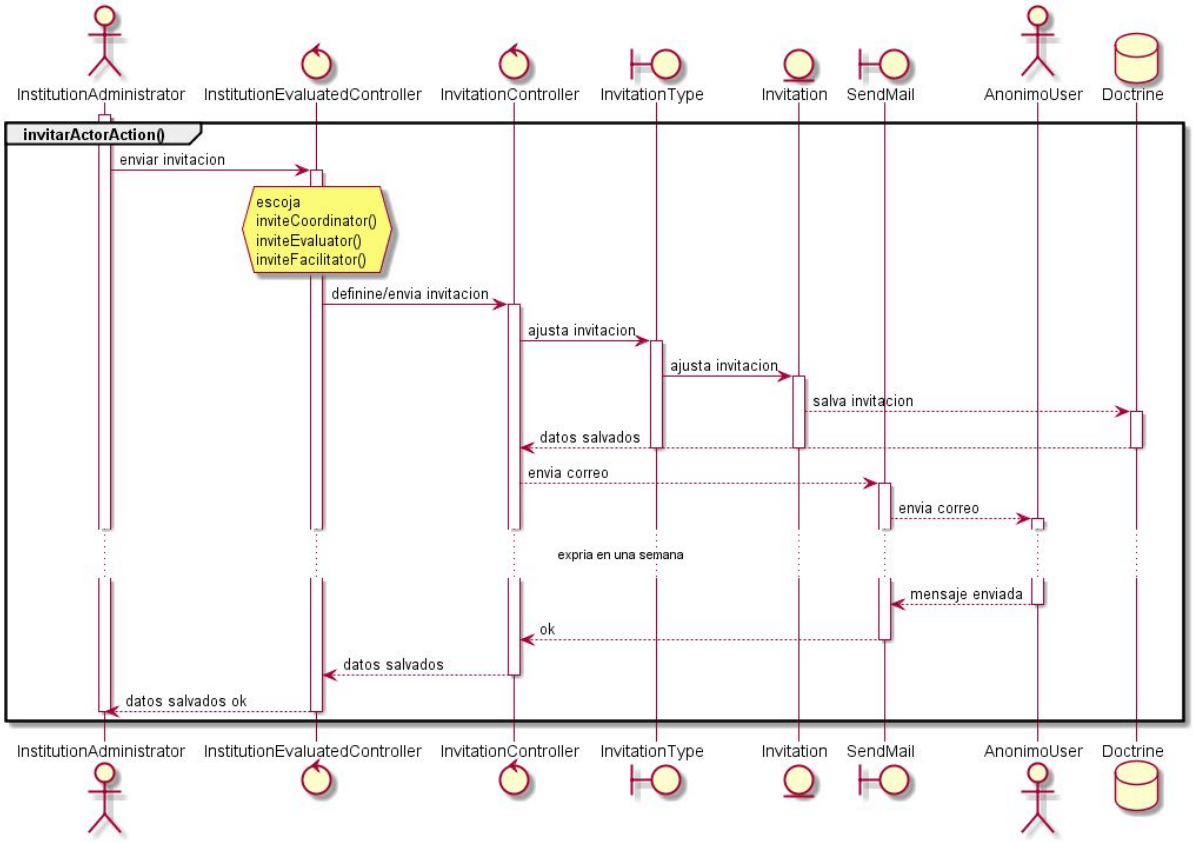

«control»
InstitutionEvaluatedController
addInstitutionEvaluated()
editInstitutionEvaluated()
viewInstitutionEvaluated()
inviteEvaluationActor()
sendRequestForEvaluation()
acceptInstitution()
rejectInstitution()
«control»
EvaluationController
individualExpertEvaluation()
consensusProposal()
consensusReview()
integratedConsensus()
viewReportCriteria()
viewReportComponent()
viewGeneralReport()
«control»
ManagerUserController
addDetailEvaluatorActor()
addDetailSocialActor()
addCurriculum()
viewUsers()
«control»
InvitationController
inviteCoordinator()
inviteEvaluator()
inviteFacilitator()

3. Diagramas de secuencia

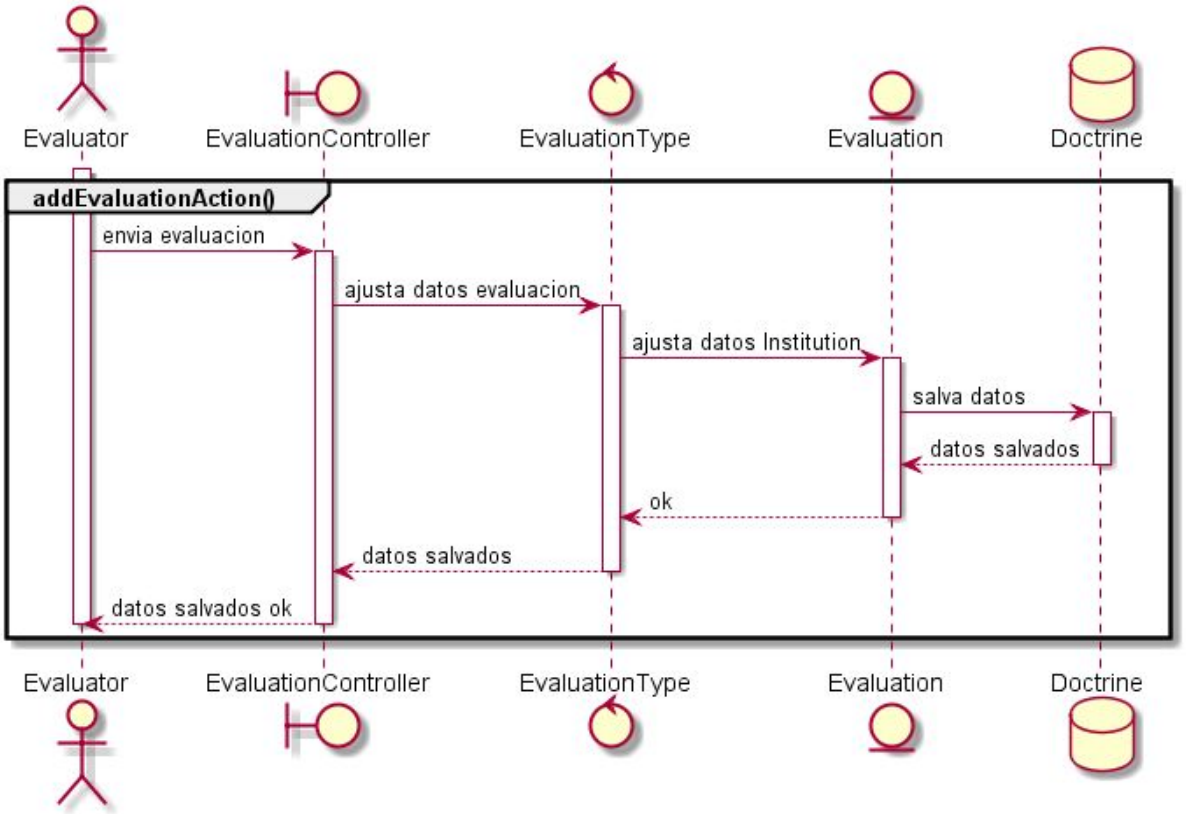
a. InstitutionEvaluationController

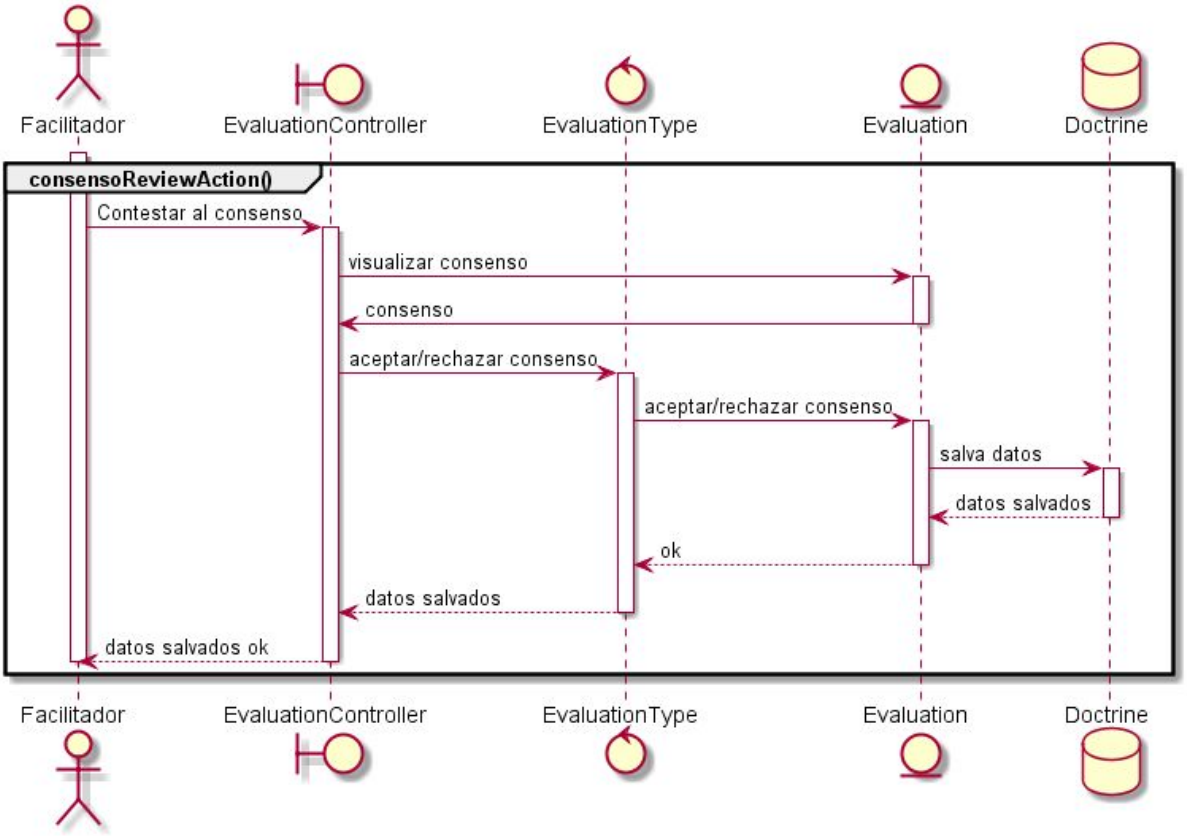

addInstitutionAction()
envia datos institucion
datos salvados
ajusta dimensiones y Institucion
datos salvados
ajusta InstitutionEvaluated
ajustar Dimension
ajusta InstitutionEvaluated
ajusta Dimension
salva datos
sendRequestforEvaluation()
aguarda aprovacion del Administrador del sistema
salvar estado
datos salvados
datos salvados
mensaje con el cambio de estado
cambio de estado
InstitutionAdministrator
InstitutionEvaluatedController
DimensionInstitutionType
InstitutionEvaluated
Dimension
DimensionInstitution
SystemAdministrator
Doctrine


InstitutionAdministrator
InstitutionEvaluatedController
EvidenciasType
Evidencia
Doctrine
addEvidenciasAction()
añade evidencia/documento
añade evidencia
ajusta Evidencia
ajusta datos Evidencia
Datos salvados
datos salvados ok

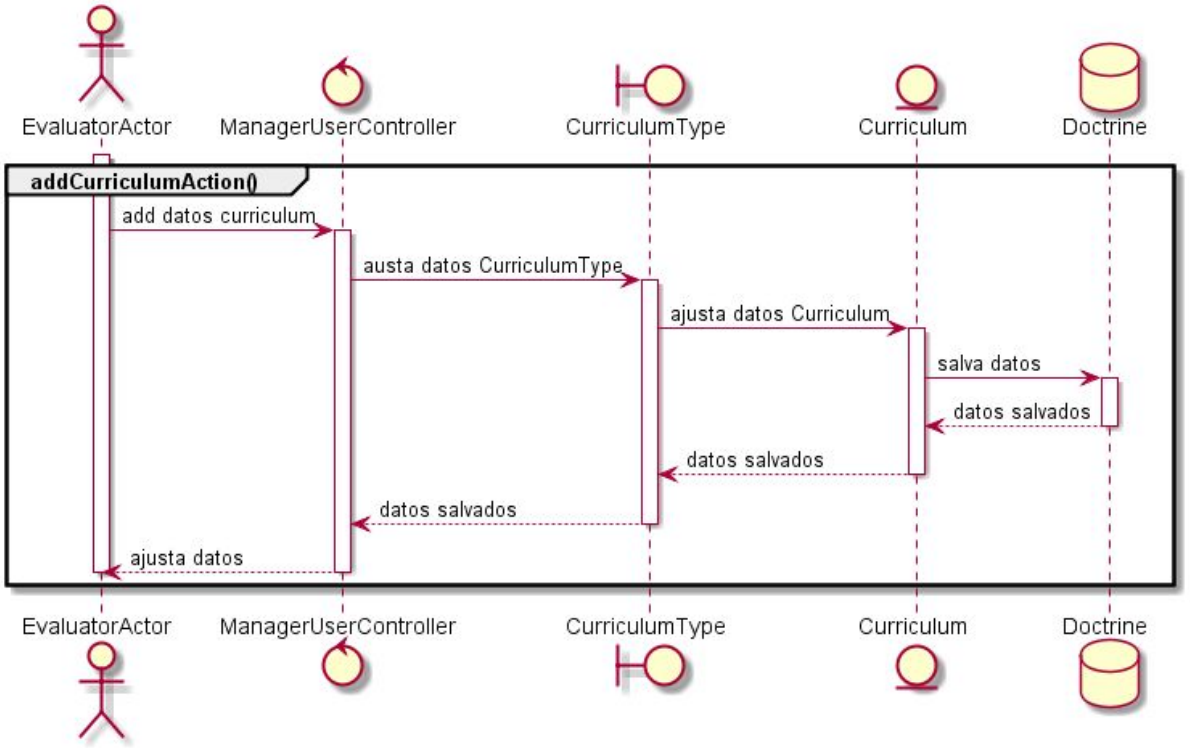

InstitutionAdministrator
InstitutionEvaluatedController
InvitationController
InvitationType
Invitation
SendMail
AnonimoUser
Doctrine
invitarActorAction()
enviar invitacion
escoja
inviteCoordinator()
inviteEvaluator()
inviteFacilitator()
definine/envia invitacion
ajusta invitacion
ajusta invitacion
salva invitacion
datos salvados
envia correo
envia correo
expria en una semana
mensaje enviada
ok
datos salvados
datos salvados ok

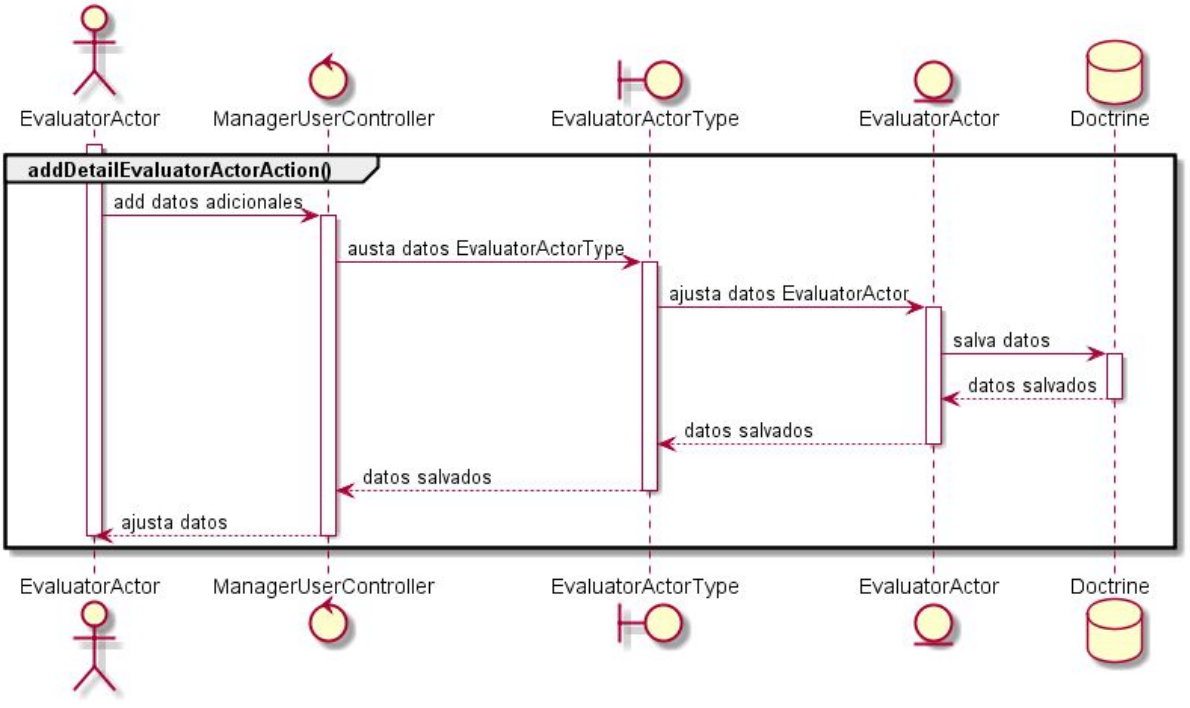
b. EvaluationController





c. ManagerUserController





ANEXO B

Risonance 2.0: Requisitos

1. User stories

01	**Como usuario anónimo quiero hacer un registro de usuario y contraseña con alta seguridad para utilizar el software**	**Epic**
F-01	-Como usuario anónimo quiero poder registrarme en el sistema utilizando mi correo -- crear las clases de usuario y las clases de control -- crear la estructura visual en html, css y javascript -- registrar datos iniciales -- confirmar la contraseña -- confirmar si el correo está correcto (el formato con algo@casa.com) -- confirmar si el correo existe por medio de un envío de email	story tasks tasks tasks tasks tasks tasks
F-02	-Como usuario anónimo quiero hacer login en el sistema utilizando un usuario y contraseña -- informar del correo y contraseña -- verificar si usuario existe	Story tasks tasks
NF-01	-Como usuario anónimo quiero mantener la seguridad de mi contraseña y mis datos -- definir un sistema de criptografía del almacenamiento de la contraseña -- reducir las exposiciones de datos, como contraseña, del banco de datos	Requisitos de dominio tasks tasks
NF-02	-Optimizar el tiempo que la página es cargada, optimizando la utilización de javascript y css	Requisitos de dominio
NF-03	-Página web deberá ser responsive adaptándose a los distintos dispositivos móviles	Requisitos de dominino
NF-04	-Los colores de la página deberán seguir la orientación de los colores del logotipo sin ser excesivos. Los colores son: 58 67 90 azul 153 153 51 verde	Requisitos de dominio
NF-04	-Al proponer un diseño al sistema se deberá llevar en cuenta los siguientes enlaces en la busca de un acercamiento. Diseño global del sitio ● http://softwarecriollo.com/ ● http://www.visionofhumanity.org/ estructuración de los temas: ● https://monocle.com/magazine/ ● http://www.yorokobu.es/	Requisitos de dominio

	visualización de resultados ● http://www.sgi-network.org/ ● http://thewebindex.org/ ● http://www.bti-project.org/en/home/ ● http://www.resourcegovernance.org/resource-governance-index ● http://www.oecdbetterlifeindex.org	

02	**Como administrador de una institución quiero hacer un pedido para que la institución que represento sea evaluada**	**Epic**
F-01	-Como administrador quiero informar los datos de la institución -- definir los datos individuales de la institución, como localidad, teléfono de contacto, etc. -- definir las dimensiones que serán evaluadas -- definir la fecha de inicio y fin del proceso de evaluación -- definir si la institución tendrá participación pública -- hacer una breve descripción justificando el pedido	Story tasks tasks tasks tasks tasks
F-02	-Como coordinador quiero definir evidencias para ayudar los evaluadores en las evaluaciones. -- enviar documentaciones -- definir descripciones de las documentaciones -- informar sitios web	Story tasks tasks tasks
F-03	Como administrador quiero invitar un coordinador -- informar un correo y enviarlo a un posible coordinador	Story tasks
F-04	Como administrador quiero gestionar los procesos de evaluación --visualizar los procesos de evaluación de la institución --visualizar usuarios pertenecientes a cada institución --visualizar los procesos de evaluación que están en retraso --visualizar nuevos actores sociales	Story tasks tasks tasks tasks
NF-01	-el coordinador confirma el correo y luego el administrador recibe una confirmación por la plataforma	

03	**Como coordinador quiero poder crear un equipo para participar en todo el proceso de evaluación**	**Epic**
F-01	-Como coordinador quiero invitar a un actor (evaluador o facilitador) -- informa por correo y enviarselo al posible actor escogiendo cuál será su posible rol (evaluador o facilitador)	Story tasks
F-02	-Como coordinador quiero poder definir lo que hará cada actor además de definir las fechas -- crear las clases de control y formularios -- envía un correo a un posible evaluador -- evaluador confirma y el coordinador recibe un mensaje -- coordinador define qué criterio en qué componente deberá evaluar	Story tasks tasks tasks tasks

	-- coordinador define la fecha de inicio y fin -- el coordinador define cuales son los criterios que el actor social podrá participar	tasks tasks
F-03	-Como coordinador quiero poder gestionar la evaluación y acompañar el seguimiento del proceso -- verificar cuales son los criterios que están con retraso -- cuáles son los usuarios que están con algún criterio con retraso -- verificar cuáles son las fases que están cumplidas -- contestar a pedidos de los actores sociales para participar de la evaluación	Story tasks tasks tasks tasks
F-04	-Como actor social quiero enviar un pedido para participar en los procesos de evaluaciones que permiten la participación pública -- actor social envía un pedido para participar en la evaluación	Story tasks
NF-01	-Optimizar el tiempo que la página es cargada, optimizando la utilización de javascript y css	

04	**Como actor participante de la evaluación quiero evaluar un criterio y contribuir para el proceso de evaluación**	**Epic**
F-01	-Como evaluador social quiero poder hacer una evaluación -- enviar una opinión para la evaluación -- informar un valor para la Descripción 1 -- informar los 5 textos referentes a la Descripción 2 -- informar un valor para cada texto de la Descripción 2 -- acceder a las evidencias de la institución	Story tasks tasks tasks tasks tasks
F-02	-Como facilitador quiero proponer un consenso entre las múltiples evaluaciones. -- visualizar cada evaluación y puntuación -- enviar una propuesta de consenso -- visualizar el relatório final	Story tasks tasks tasks