UNIVERSIDAD MAYOR DE SAN ANDRÉS - FACULTAD DE INGENIERÍA
INGENIERÍA ELECTRÓNICA

PERFIL DE PROYECTO DE GRADO

TUNKUNIA: DESARROLLO DE UN PAQUETE FOSS PARA LA IMPLEMENTACIÓN Y SEGUIMIENTO DE TRÁMITES ADMINISTRATIVOS. CASO DE ESTUDIO: SISTEMA SIAI

POSTULANTE: ERNESTO CARLOS ARENA ALARCON

ASESOR: JORGE ANTONIO NAVA AMADOR

DOCENTE D.A.M.: JORGE LEÓN

La Paz, 24 de junio de 2024

Índice

1. Introducción

El presente proyecto plantea como producto un paquete de software como se describirá más adelante y al cual nos referiremos en adelante bajo el nombre de “Tunkunia”, o simplemente .[el] producto”.

Para evitar el uso inadecuado de términos, se prefiere la utilización de términos en idioma inglés cuando el concepto es ambiguo en su traducción, no existe una traducción oficial ampliamente utilizada o sencillamente para evitar que este documento herede también esta ambigüedad, como es en el caso de una librería de software, proveniente de ”software library”, cuya traducción también podría ser ”Biblioteca de software”.

2. Antecedentes

3. Situación Actual

4. Planteamiento del Problema

Gobiernos de todo el mundo buscan digitalizar sus procesos para facilitar la interacción con la población y aumentar la eficiencia de los mismos, apuntando a lo que la CEPAL llama Gobiernos Digitales y Gobiernos Inteligentes.

Dentro de todos los procesos administrativos la figura de trámite y el seguimiento del mismo se encuentra siempre presente a todo nivel y suele ser motivo o parte del motivo de la creación de muchos sistemas de software gubernamentales.

El desarrollo de sistemas de software es un proceso largo que requiere por sí mismo la toma de varias decisiones profesionales para brindar a los usuarios el mejor producto en el menor tiempo posible y garantizando que se cumplan ciertos requerimientos funcionales y no funcionales, implicando una gran complejidad que sería limitante si no fuera por algunas buenas prácticas de desarrollo.

A menudo, los equipos de desarrollo, para evitar que la complejidad de los sistemas sea excesiva, buscan herramientas de terceros que les permiten no reinventar la rueda. Estas herramientas, llamadas paquetes de software, que pueden ser librerías o incluso frameworks de desarrollo, ofrecen una serie de ventajas a los proyectos de desarrollo, así como algunas limitaciones que, en la mayoría de los casos, y dependiendo del proyecto, no son relevantes, pero requieren la atención del equipo de profesionales.

Estas herramientas, que forman parte fundamental del desarrollo moderno, suelen ser de código abierto, o al menos así se prefieren por muchos profesionales, dadas sus ventajas de seguridad y versatilidad.

Es importante notar también que en algunos países se demanda o prefiere el uso de software libre en el ámbito público. Tal es el caso de Bolivia, que en su artículo 77 de la ley general de telecomunicaciones, tecnologías de información y comunicación indica que se debe promover y priorizar el uso del mismo dentro de los órganos ejecutivo, legislativo, judicial y electoral.

Por lo anteriormente descrito podemos encontrar una gran cantidad de tecnologías abiertas usadas para el desarrollo de sistemas, como la librería Carbon de PHP, el estándar EcmaScript o el popular framework de desarrollo Laravel, las cuales, a pesar de proporcionar utilidades que facilitan la creación de proyectos pequeños y medianos de software, no proporcionan de manera específica un marco de trabajo confiable y específico para trabajar con los tan frecuentes trámites administrativos y su seguimiento, obligando a los desarrolladores a reinventar la rueda en cada proyecto de este tipo.

El caso que se toma para este proyecto, que es el sistema SIAI del Viceministerio de Desarrollo Productivo, desarrollado por la empresa 2IES, tiene un módulo de seguimiento de trámites realizado desde cero para el mismo y, además de ser en el que se basará este proyecto y sobre el cual se aplicará

para medir el éxito del mismo, es un claro ejemplo de la necesidad de un paquete reutilizable para la creación de módulos de trámites y seguimiento de trámites.

Otros sistemas, cuyo código fuente, se escapan al conocimiento del autor de este documento, pero que se estima requirieron un desarrollo desde cero para su módulo de trámites son:

- Sistema de Matriculación de la UMSA
- Sistema para permisos de vidrios polarizados
- Sistema para permisos de circulación en pandemia
- Procedimiento para autorización de vidrios oscurecidos

Figura 1: Pasos del trámite para vidrios oscurecidos que se realizó en línea

Entre los sistemas anteriormente expuestos, se hace énfasis en los sistemas creados en época de pandemia, o el caso de los vidrios oscurecidos de la figura 1 , los cuales debían ser desarrollados con premura dadas las circunstancias, lo cual puede ser facilitado con una librería de software.

La carencia de una librería específica a los trámites y su seguimiento hacen visibles varias desventajas:

- Reinvención de la rueda en varios proyectos de características similares.

- Poca atención al detalle para un módulo de gran importancia.
- Tiempos de desarrollo mayores.
- Mayor coste.
- Sobre-ingeniería en proyectos pequeños.
- Menor adopción de sistemas digitales en el aparato público.
- Mayor susceptibilidad a desarrollos fallidos.
- Falta de características.
- Dificultad mayor en la redacción de la documentación.
- Poca presencia de proyectos open source bolivianos.

Actualmente existen librerías que ayudan con un concepto muy útil en el seguimiento de trámites que son las máquinas de estados. Sin embargo, no son específicas, son poco usadas y carecen de una buena documentación o facilidades para el desarrollador.

Además, sistemas similares a los de seguimientos de trámite también existen en el mercado, pero como productos cerrados y genéricos que no garantizan seguridad para un sistema gubernamental y no pueden ser reutilizados por varios proyectos de forma gratuita como es el caso de una librería de código abierto.

Por lo expuesto, se tiene una visión inicial que consiste en la premisa de que cuando una funcionalidad es común en distintos sistemas, una solución viable y común es la creación de una herramienta reutilizable de código abierto que ayude en su implementación. Esto proporciona varias ventajas y se suscribe al principio DRY [Don't repeat yourself] y al primer principio SOLID sobre la single responsibility.

Sin embargo, una librería open source acarrea varios desafíos en su realización, muchos de ellos académicos, de los cuales a continuación se listan algunos:

- Elección de un entorno sobre el cual aplicar este proyecto: Una librería puede estar limitada a un lenguaje de programación o incluso a un marco de desarrollo. Esta decisión es importante tomarla al inicio del proyecto.
- Metodología de desarrollo: Adoptar una buena metodología es importante en una librería, ya que ésta será utilizada por muchos proyectos que confiaran en la misma. Además, al ser open source, se espera que de tener éxito la misma siga mejorando.
- Arquitectura de Software: Este es un tema muy someramente estudiado durante el transcurso de la carrera de ingeniería Electrónica de la UMSA, pero es muy necesario para tomar las mejores decisiones en el desarrollo de un sistema.
- Documentación: Escribir la documentación de una librería que se espera sea utilizada por muchas personas es de mayor relevancia. Se requiere un buen lenguaje técnico y habilidades de redacción, pero a la vez la capacidad para que lo descrito sea fácil de entender por la mayor cantidad de personas.
- Knowhow open source: Se cuenta con muy poco conocimiento del desarrollo de software libre en el contexto local y mucho menos de sistemas con éxito.

- Versionado: Cualquier proyecto de software moderno requiere el uso de sistemas de versionado, pero en un proyecto de código abierto esto es especialmente importante para permitir colaboraciones externas.
- Buenas prácticas de desarrollo y uso de patrones de diseño: Para tener una buena librería existen libros de recomendaciones y patrones que pueden ser empleados, además de experiencias compartidas en internet por distintos desarrolladores. Las mismas pueden potenciar un proyecto y son importantes de aprender.
- Infraestructura y escalabilidad: La librería debe ser pensada para poder ser desplegada en sistemas grandes y poder ser escalable. El tema de escalabilidad es en sí mismo un campo de estudio dentro del despliegue de sistemas.
- Testabilidad: Los proyectos de software moderno tienen como proceso importante el del testing, el cual permite realizar desarrollos que cumplan con lo que se desea en su diseño y que no hagan algo distinto [1]. Sin embargo, el campo del testing no es ni minimamente explorado en instituciones universitarias, a pesar de su importancia.

5. Objetivos

Desarrollar la versión inicial de un paquete de software libre de código y concepto reutilizable, FOSS (free and open source software), que facilite la tarea de creación y seguimiento de trámites en base al uso de autómatas finitos o máquinas de estado finitas, la definición de lineamientos para su implementación y utilidades comunes a estos procedimientos administrativos. Todo esto en base a los requerimientos, experiencias y desarrollo del sistema "SIAI" (Sistema de Información Ambiental Industrial) del "Ministerio de Desarrollo Productivo y Economía Plural", que fue implementado por la empresa 2IES utilizando el framework Laravel.

La concreción de este objetivo involucrará de forma necesaria la realización de las actividades siguientes:

- Extraer los requerimientos específicos al módulo de trámites del sistema SIAI, que es un sistema que cuenta con varios módulos y los procesos administrativos son parte de uno de ellos.
- Definir una licencia de software libre que tenga coherencia con el objetivo general del proyecto, que armonice con los preceptos defendidos por la FSF (Free Software Foundation) y que sea compatible con la normativa boliviana.
- Estudiar y analizar casos de éxito en el mundo del software open source, considerando factores relevantes como el número de descargas, forks, etc [2].
- Adoptar prácticas que permitan entregar software de calidad.
- Escribir código autodocumentado y cuando sea necesario correctamente comentado para ser amigable a las contribuciones.
- Recibir al menos un pull request de un colaborador, como prueba de legibilidad del código.
- Redacción y despliegue de una documentación clara y moderna que sea fácilmente navegable y entendible, atendiendo a la reusabilidad del paquete a desarrollar.
- Poner en práctica patrones de diseño comunes como el patrón estado basado en máquinas de estados.

- Usar el paquete de software resultante de este proyecto en una implementación de un fork del sistema SIAI como caso de estudio.
- Implementar una interfaz para facilitar al desarrollador el uso de la herramienta.
- Publicar tanto este documento, realizado en Latex, como el documento final de proyecto y el código fuente en un repositorio remoto como GitHub.
- Añadir el paquete realizado a un repositorio de paquetes como el utilizado por el framework Laravel.
- Implementar medidas para garantizar la integridad de los datos de los trámites dentro del sistema.
- Integrarse con herramientas de software libre ya existentes dentro del entorno sobre el cual se realice el desarrollo para facilitar y mejorar la calidad de la implementación de este proyecto.
- Incentivar una nueva serie de proyectos de grado sobre software reutilizable open source en la carrera de Ingeniería Electrónica para posicionar a los estudiantes en el mercado laboral y como profesionales
- Crear una comunidad de proyectos de código abierto alrededor de éste.

6. Justificación

La solución propuesta tiene 3 pilares importantes que la componen y muestran relevancia cada una por su cuenta:

- La figura del paquete de software reutilizable.
- La figura del trámite en instancias gubernamentales y su modernización.
- El desarrollo de software libre.

De acuerdo a esto podemos ver la importancia de esta solución en distintos ámbitos:

6.1. Tecnológico

La creación de un paquete de software reutilizable para la creación y seguimiento de trámites ayuda en el proceso de digitalización de uno de los procedimientos más comunes en el ámbito público, brinda a los gobiernos la posibilidad de aprovechar de mejor manera los datos resultantes de un trámite y dan a la población herramientas que hacen más fáciles sus vidas.

Al usar máquinas de estados finitas logramos el aprovechamiento del conocimiento en sistemas secuenciales digitales dentro del ámbito de sistemas de software para lograr una abstracción del trámite.

El paquete no sólo facilitará la implementación de sistemas de software con módulos de trámites, sino que de forma más directa simplificará la tarea de los desarrolladores, siendo una pieza tecnológica dentro de proyectos más amplios.

6.2. Económico

El software reutilizable de código abierto suele se aprovechado hoy en día con el objetivo de disminuir costos en la producción de sistemas gracias a su naturaleza de comunidad y lo demandado de su funcionalidad.

De no existir el software reutilizable se deben invertir recursos para cada proyecto que requiera la misma funcionalidad. Recursos que, de usar un paquete de software, podrían conservarse, siendo que una parte del desarrollo ya estaría implementada. Existen excepciones a este caso en proyectos con necesidades muy específicas, pero en la mayoría de proyectos, una librería o framework bien implementado son esenciales para disminuir costos de producción.

6.3. Académico

Los proyectos de software libre a nivel regional y a nivel universidad son realmente escasos y al momento de redacción de este documento se desconoce de algún caso de éxito en la carrera de Ingeniería Electrónica de la UMSA.

Es por eso que este proyecto busca ser un punto de partida hacia la realización de más proyectos del mismo tipo, es decir, de software libre reutilizable. Todo esto a partir de esta travesía que a modo de ejemplo busca inspirar a más estudiantes de la carrera.

6.4. Político

La realización de este proyecto no sólo se alinea con la necesidad de los distintos gobiernos del mundo de digitalizar sus procesos administrativos - como se puede constatar en Bolivia por el decreto xxx donde se solicita a las distintas instituciones gubernamentales la realización de planes hacia un gobierno electrónico -, sino además apunta a la preferencia que los gobiernos tienen por el software libre, como se describe en el artículo xx de la ley xx.

6.5. Social

Los trámites son un dolor de cabeza para gran parte de la sociedad. Los gobiernos hacen el intento por digitalizar los mismos y así aliviar a la población, pero la existencia de herramientas reutilizables pueden acelerar drásticamente este proceso. Se espera que más y más trámites sean digitalizados en tanto más fácil sea hacerlo. De esta manera los distintos actores de la sociedad podrán aprovechar las bondades de las tecnologías de la información.

La digitalización de trámites, que sería facilitada por este proyecto, mejora además los mecanismos por los cuales la sociedad participa del gobierno. Impulsa la apertura de la información de los gobiernos hacia la gente y se adecúa a las nuevas necesidades de las personas.

7. Alcances y Limitaciones

De acuerdo a los objetivos planteados es menester delinear el campo de acción del proyecto.

7.1. Alcances

- Se desarrollará un paquete de software en su primera versión con funcionalidades para facilitar la creación y seguimiento de trámites.
- Tunkunia será distribuido mediante al menos un repositorio remoto de versionado como GitHub.

- El producto además será publicado en algún repositorio de paquetes de software coherente con la tecnología que use.
- Se usará una licencia de software libre que permita el uso de esta idea sin restricciones, pero mencionando al autor.
- El proyecto contará con la implementación de una documentación en línea para facilitar el uso de Tunkunia.
- Se demostrará el uso del paquete en una implementación de un fork del sistema SIAI.
- Se usarán buenas prácticas de programación para lograr código legible y sobre el cual sea sencillo colaborar.
- Los lineamientos, fruto de la creación conceptual de este software serán creados con la ayuda de herramientas para desarrolladores como un CLI.
- Se recibirá al menos un pull request en el repositorio para demostrar las bondades del software libre y se atenderá al menos un issue reportado.
- Se adoptarán medidas de testing para entregar un mejor código.

7.2. Limitaciones

- El desarrollo no será realizado desde cero y de forma completa, sino que siguiendo los mismos lineamientos sobre los cuales se piensa este proyecto de software, se usarán herramientas de software libre como ladrillos de construcción para éste. Esto puede ser considerado un alcance de acuerdo a cómo se lo vea.
- La versión 1 será funcional, pero no se garantiza que esté libre de bugs, lo cual es natural en muchos desarrollos de software a pesar de la aplicación de las mejores prácticas y el testing.
- El sistema, si bien podría ser utilizado por un amplio abánico de aplicaciones, en su versión 1, que es la que atañe a este documento, se centrará unicamente en los trámites públicos más comunes y sin cubrir necesariamente las necesidades de cualquier trámite que pueda existir. Se deben tomar como marco los trámites realizados en el sistema SIAI, que es el caso de estudio sobre el cual se desarrollará Tunkunia.
- Si bien este proyecto puede ser aplicado en distintos entornos de manera conceptual, debe quedar claro que la implementación práctica de cualquier paquete de software suele estar acotada a ciertas tecnologías. En el caso presente esta tecnología será Laravel por su presencia en el sistema SIAI y por su crecimiento reciente en popularidad.

8. Descripción de la Solución Propuesta

Dada la problemática expuesta en la sección 4, en la que se describe de manera breve el enfoque que se le piensa brindar, en esta sección se trata de describir la primera aproximación a su solución de forma detallada.

Sin duda, la realización de una herramienta de software libre reutilizable para la implementación de trámites y su correspondiente seguimiento a nivel de backend tiene muchas ventajas, pero su realización no es del todo trivial y demanda que se conceptualice en un modelo medianamente robusto, con metodologías claras y descripciones escuetas.

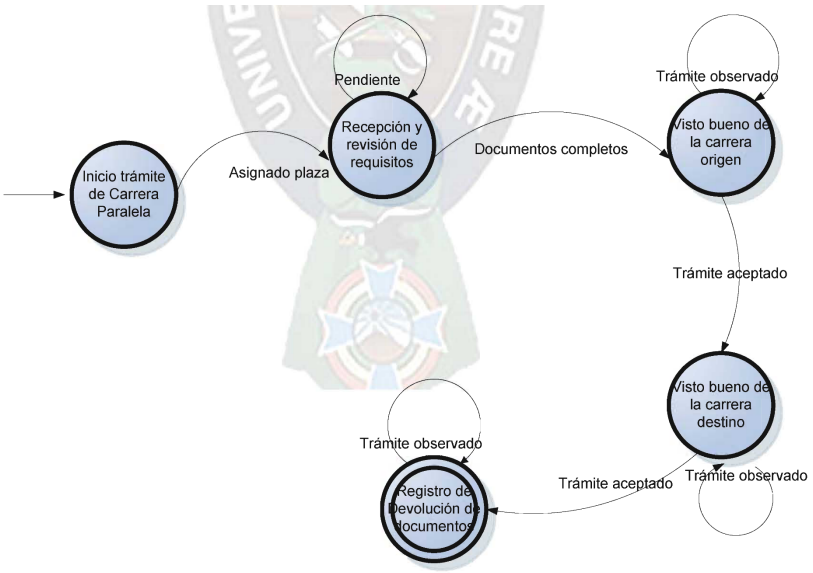

Figura 2: Diagrama de una máquina de Turing para carreras paralelas desarrollada en [4]

Cuando uno piensa en trámites, usualmente piensa en burocracia, en largas filas, una obligación muchas veces irrenunciable y una lista interminable de pasos a seguir. Según la Real Academia de la Lengua Española, se define como "Cada uno de los pasos y diligencias que hay que recorrer en un asunto hasta su conclusión" [3].

Si bien las definiciones de trámite son escasas y algunas pueden tratar su semántica desde una perspectiva más funcional, es indudable que un trámite es un procedimiento que consta de uno o más pasos a seguir. De este modo, se puede vislumbrar una manera casi obvia de modelarlo usando conceptos de teoría computacional, de matemáticas discretas o circuitos secuenciales. Esto es, usando máquinas de estado finitas.

Este enfoque no es precisamente nuevo y ya se puede ver en una aproximación al modelado en software de distintos trámites de la división de gestiones, admisiones y registros de la UMSA, donde se emplearon máquinas de Turing que a efectos prácticos se aproximan más a máquinas de estado finitas [4], como se puede ver en la figura 2.

Desde luego, las máquinas de estados finitas parecen ser una manera sencilla de modelar los procesos de trámite a nivel de software, principalmente por su paralelismo con los pasos y sus respectivas transiciones, asemejándose a un procedimiento administrativo, como se puede evidenciar en la definición matemática de este autómata:

Definición 1 (Máquina de Estados Finita) *Una máquina de estados es un conjunto de 5 elementos $M = (S,I,O,v,w)$, donde S representa a la colección de estados de M; I representa al alfabeto de entradas para M; O es el alfabeto de salidas de M; $v : SxI- > S$ es la función del siguiente estado; y $w : SxI- > O$ es la función de salida [5].*

Este paralelismo es más notorio en un diagrama de estados como el que se muestra en la figura 3, donde tenemos los estados de $S = (s_0, s_1, s_2, s_3, s_4, s_5, s_{error})$ y algunas transiciones entre los mismos, cada una con sus correspondientes entradas (i_{mn}) y salidas (o_{mn}). Las transiciones restantes se omiten porque apuntan a un mismo estado de error del trámite (s_{error}).

En la figura 4 podemos ver el mismo diagrama, pero simplificado y usando un lenguaje cercano a la naturaleza de los procedimientos administrativos de forma adrede para hacer más obvia la relación

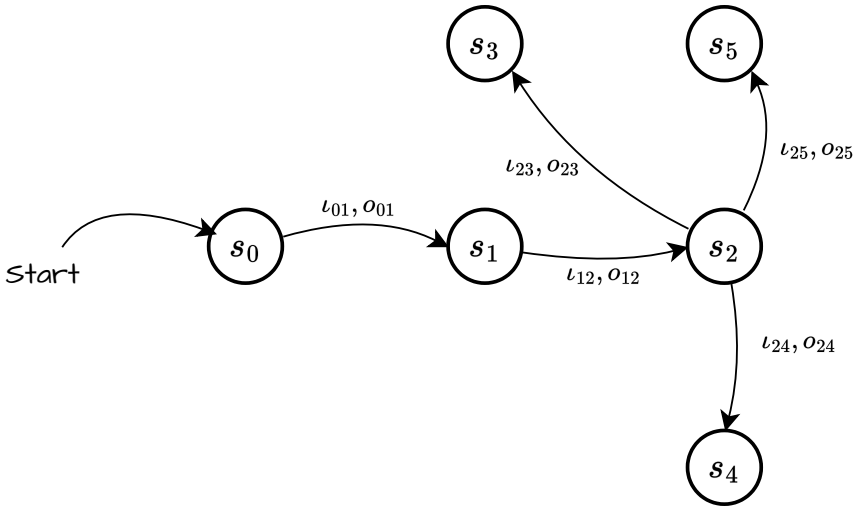

Figura 3: Diagrama de estados finitos con salidas

descrita. Nótese que se usan para los estados nombres cortos y no totalmente detallados para fines prácticos de ilustración. En un sistema real se esperan definiciones claras que reflejen la totalidad del trámite implementado.

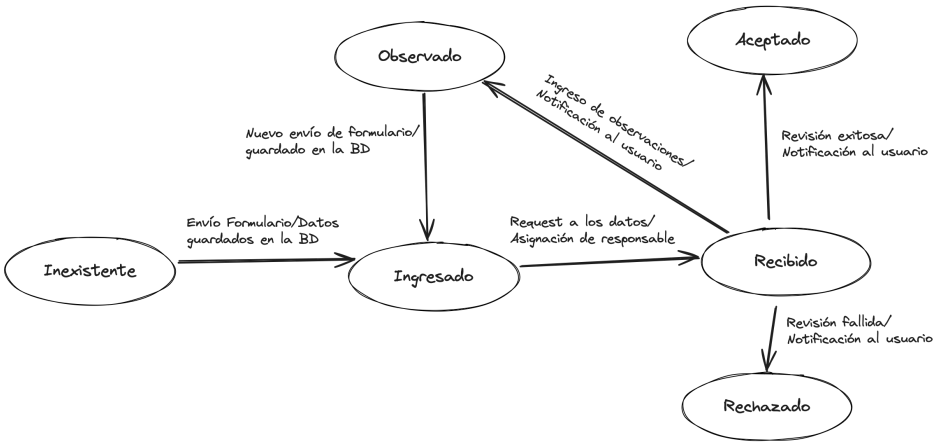

Figura 4: Diagrama de estados simplificado para un trámite

Se plantea utilizar este enfoque del uso de autómatas, como también son llamadas las máquinas de estado en contextos de teoría computacional, pero con ciertos lineamientos conceptuales que guíen al usuario de esta solución, que inicialmente serían:

- Descripción detallada de los estados, compuesta mínimamente por un nombre único, una descripción, el sujeto cuya perspectiva se usa para la descripción y su visibilidad.
- Preferencia del autómata de Moore sobre el autómata de Mealy. Los autómatas de Mealy son una simplificación en número de estados a partir de los autómatas de Moore, con estados que en las aplicaciones originales del concepto podían ser considerados innecesarios [6]. Si bien en muchas aplicaciones es lo deseado, en temas de auditoría se prefiere tener la mayor cantidad de

información relevante, por lo que sin restringir el uso de máquinas de Mealy, se priorizará el uso de máquinas de Moore.

- Respecto al anterior punto, incluso si el usuario usa Mealy en lugar de Moore, se recomienda la definición de tantos estados como sean posibles.

Como parte de la solución propuesta, se debe entender que los lineamientos conceptuales son muy importantes, ya que este proyecto no será solamente de código abierto, sino además de concepto abierto, es decir, estas ideas se pueden aplicar en otros entornos distintos al que se elegirá para esta solución [7, pág. 439], el cual se detalla más adelante.

Cualquier implementación de software implica una toma detallada de requerimientos sobre la cual trabajar. Quizá una desventaja del software libre en la práctica es que las metodologías no son aplicadas correctamente, los programadores definen los requerimientos y no hay objetivos medibles [8, Tabla 1]; esto debido a que no existen las figuras de cliente, inversión y beneficio económico, que precisamente son la fuente de los requerimientos más importantes para el proyecto.

Por esto mismo, para la solución planteada se aprovechará el ya existente proyecto SIAI (Sistema Integral Ambiental Industrial) del Viceministerio de Desarrollo Productivo, desarrollado por la empresa 2IES y en el cual, el autor de este documento, tuvo participación. Este sistema contempla requerimientos comunes en la realización de trámites gubernamentales y será una guía para la implementación actual.

Evidentemente, al ser un proyecto de código libre y al pretender dejar su desarrollo abierto desde el día 1 en Github, sería un despropósito no aprovechar y/o buscar más proyectos similares que puedan enriquecer el desarrollo actual.

El sistema SIAI, que será empleado como referencia, tendrá además otro papel importante en este proyecto, siendo el producto del mismo utilizado en una implementación de un fork del mencionado software para demostrar la usabilidad del paquete.

Esto nos obliga a tomar una decisión a priori sobre el desarrollo del producto que se propone como solución y es que, como cualquier otro paquete de software, su implementación suele estar restringida a alguna tecnología existente [7, pág. 444]. Dada la popularidad del framework Laravel de PHP y a que el sistema SIAI está desarrollado usando estas tecnologías se procederá a la creación de un paquete, precisamente, de Laravel.

La solución, al ser un paquete de software, contará no solamente con las ideas y marcos de trabajo descritos anteriormente, sino que además se piensa brinde herramientas que ayuden al proceso de trámites, como por ejemplo:

- Generación de códigos de trámite.
- Generación de códigos de verificación de correos (Que si bien hay alternativas para lo mismo, se facilitará el uso desde el paquete).
- Funcionalidad de notificaciones.
- Límites de tiempo asignables a los estados de trámite.
- Funcionalidad de persistencia de historial de estados incorporada.
- Funciones de recuperación de historia de trámite para seguimiento.
- Niveles de visibilidad de trámites para seguimiento.

- Mecanismos para dificultar el entorpecimiento de un trámite y facilitar la identificación de responsables.
- Interfaz sencilla para cambio de estados de una máquina (trámite).

Para concluir esta aproximación a la solución a realizar, quedan dos puntos importantes a tocar, uno de ellos opcional a la fecha de redacción de este documento, y es acerca de la naturaleza FLOSS o FOSS de este proyecto.

Para que un proyecto de software sea considerado de software libre debe cumplir con 4 libertades esenciales [9]:

- La libertad de ejecutar el programa como se desee, con cualquier propósito (libertad 0).
- La libertad de estudiar cómo funciona el programa, y cambiarlo para que haga lo que se desee (libertad 1). El acceso al código fuente es una condición necesaria para ello.
- La libertad de redistribuir copias para ayudar a otros (libertad 2).
- La libertad de distribuir copias de sus versiones modificadas a terceros (libertad 3). Esto le permite ofrecer a toda la comunidad la oportunidad de beneficiarse de las modificaciones. El acceso al código fuente es una condición necesaria para ello.

Esto debe ser claramente definido en un documento de licencia de Software que se encuentre junto al código fuente. Existen varias licencias que cumplen con los lineamientos antes señalados y son aprobados por la FSF (Free Software Foundation) [10].

Si bien la licencia predilecta del Software Libre es la GNU-GPL, de la cual existe incluso una versión boliviana llamada LPG-Bolivia [11], puede ser demasiado restrictiva para quien quiera usar los resultados de este proyecto. Por esto mismo se piensa licenciar no sólo el software sino también los documentos de proyecto asociados bajo alguna licencia similar a la del MIT [12], que brinda la posibilidad de generar software no libre a partir de software libre y es ampliamente usado por varios proyectos Open Source.

Un proyecto de software libre suele ser distribuido en la red y suele tener alguna denominación distintiva. Dada la naturaleza de los trámites, que consiste en una secuencia de pasos se piensa bautizar al paquete resultante de este proyecto como “Tunkunia”.

9. Temario

Figura 5: Proceso del Software - Pressman

Para el temario del documento final de proyecto se considerará una estructura que describa bien la naturaleza del proyecto y que además se adecúe a este que, después de todo, será un producto de software.

De acuerdo a la ingeniería de software, el software tiene un ciclo de vida o un “proceso del software”, el cual se modela de acuerdo a la metodología de desarrollo sobre la cual se realice. Sin embargo,

varios autores concuerdan en que existen ciertas etapas estructurales ajenas a cualquier metodología. Según Pressman [13, pág. 13], estas etapas serían las indicadas en la figura 5.

Por su lado, Sommerville las simplifica en las 4 etapas mostradas en la figura 6.

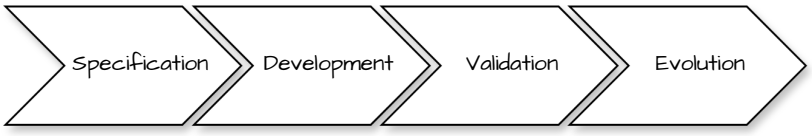

Figura 6: Proceso del Software - Sommerville

El temario del documento final del proyecto obedecerá entonces a esta definiciones del proceso de software para no depender estrictamente de la metodología usada.

Nótese, sin embargo, que este temario es tentativo, lo cual quiere decir que pueden surgir cambios durante la realización del proyecto.

Índice

0. Apartados Preliminares

- Presentación
- Agradecimientos
- Resumen
- Índice
- Glosario

1. Generalidades del proyecto:

1.1. Introducción: Describirá los antecedentes del proyecto, así como la problemática que se ha identificado, para la cual se plantea una solución a través del objetivo. De igual modo se hará referencia a la justificación del proyecto y los alcances y límites que se plantearon durante su gestación.

1.2. Marco Referencial: Se proporciona el contexto general en el cual se sitúa el proyecto. Se describen temas relevantes al tópico principal y que buscan delimitarlo en varios aspectos que pueden ser históricos, geográficos, sociales, políticos, económicos, tecnológicos, etc.

1.3. Marco Teórico: Se proporcionan las bases teóricas sobre las cuales se fundamenta el proyecto. se presentarán todos los conceptos, definiciones, técnicas y/o procedimientos que se han de tomar en cuenta para el desarrollo del proyecto y para su correcta comprensión.

2. Proceso del Software: Como se explica al inicio de la sección 9, esta parte del temario reflejará el ciclo de vida común del software. El orden no refleja el orden que se adopte realmente, el cual no será de tipo cascada, sino iterativo, pero como bloques fundamentales del proceso del software, tendrán la información relevante a cada etapa o cada esfuerzo realizado dentro de cada etapa.

2.1. Especificación de requerimientos de software [13, pág. 104]

2.2. Modelado y Diseño

2.3. Desarrollo y Construcción

2.4. Pruebas y Validaciones (Tests)

2.5. Despliegue

3. **Integración en el SIAI:** El paquete ya desplegado se integrará en un fork del SIAI. En este apartado se hará una bitácora del proceso y se expondrán los resultados del mismo.

4. **Resultados y Conclusiones:** El resultado general del proyecto y algunas recomendaciones para seguir trabajando en él, siendo especialmente importante considerando la naturaleza open source y colaborativa del paquete a desarrollar. Se deben dar algunas ideas para seguir desarrollando las siguientes versiones del software.

   - Bibliografía y Referencias
   - Anexos

10. Cronograma

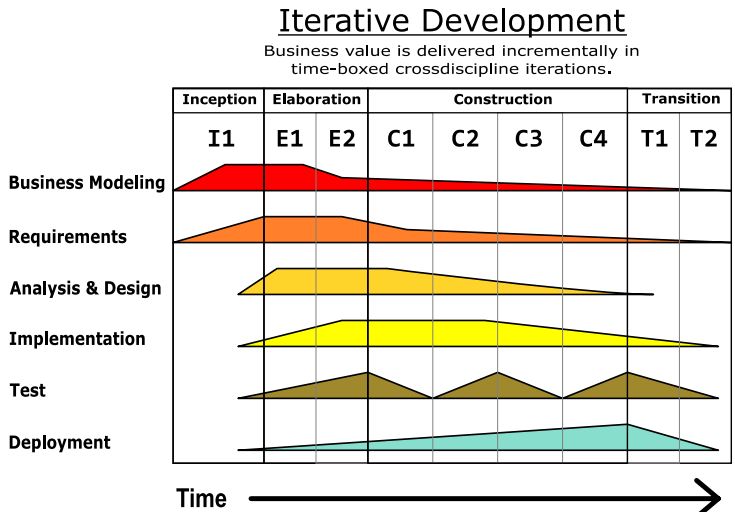

Figura 7: Fases del Proceso Unificado y los esfuerzos de cada actividad en las mismas

El cronograma de trabajo (figura 8) estará fuertemente influenciado por las fases de la metodología RUP (Rational Unified Process), la cual cuenta con 4 fases y 6 actividades principales que se realizan de forma iterativa en cada una de estas fases. La figura 7 muestra cuánto de cada actividad se debe realizar en cada etapa y permite entender el cronograma presentado.

Se debe notar que la redacción de la memoria se realizará de forma paralela al desarrollo del proyecto, al igual que el proceso de Investigación. Sin embargo, en el diagrama de Gantt se muestran las semanas en las que estas dos actividades tomarán prioridad.

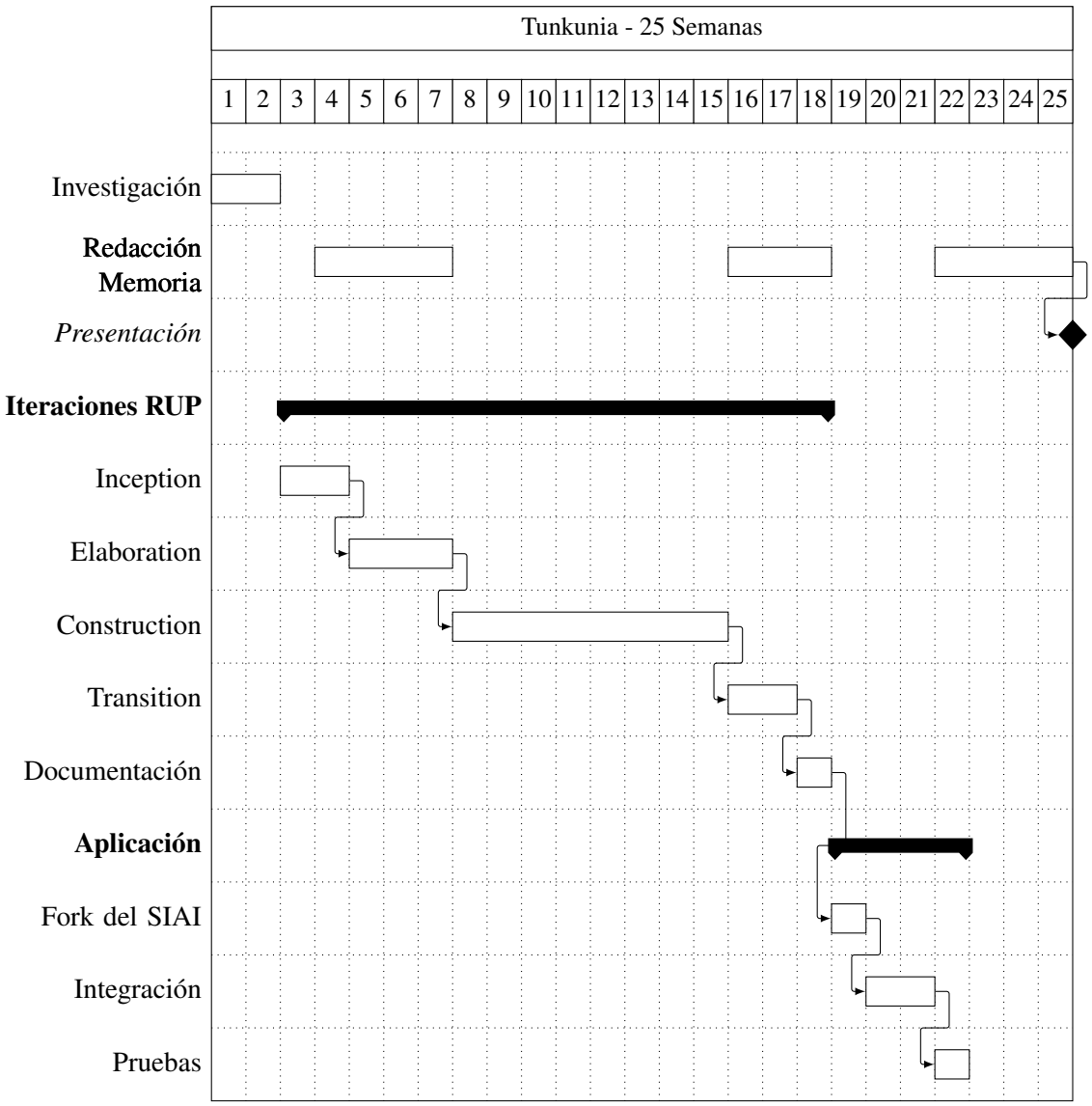

Figura 8: Diagrama de Gantt del proyecto

Referencias

[1] G. J. Myers, T. Badgett y C. Sandler, *The Art of Software Testing: Now Covers Testing for Usability, Smartphone Apps, and Agile Development Environments*, 3. ed. Hoboken, NJ: Wiley, 2012, 240 págs., ISBN: 978-1-118-03196-4.

[2] S. Mujahid, R. Abdalkareem y E. Shihab, «What Are the Characteristics of Highly-Selected Packages? A Case Study on the Npm Ecosystem,» *Journal of Systems and Software*, vol. 198, pág. 111 588, 1 de abr. de 2023, ISSN: 0164-1212. DOI: 10.1016/j.jss.2022.111588. dirección: https://www.sciencedirect.com/science/article/pii/S0164121222002643 (visitado 14-06-2024).

[3] R. ASALE y RAE. «Diccionario de la lengua española — Edición del Tricentenario,» «Diccionario de la lengua española» - Edición del Tricentenario. (), dirección: https://dle.rae.es/ (visitado 18-06-2024).

[4] P. J. Nacho, «SISTEMA DE CONTROL DE TRÁMITES UTILIZANDO MAQUINAS DE TURING CASO: DIVISIÓN DE GESTIONES ADMISIONES Y REGISTROS U.M.S.A.,» Proyecto de Grado, Universidad Mayor de San Andrés, La Paz, Bolivia, 2007, 91 págs. dirección: https://repositorio.umsa.bo/handle/123456789/1318.

[5] R. P. Grimaldi, *Discrete and Combinatorial Mathematics: An Applied Introduction*, 3. ed., [Nachdr.] Reading, Mass.: Addison-Wesley, 1998, ISBN: 978-0-201-54983-6.

[6] G. H. Mealy, «A Method for Synthesizing Sequential Circuits,» *The Bell System Technical Journal*, vol. 34, n.º 5, págs. 1045-1079, sep. de 1955, ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1955.tb03788.x. dirección: https://ieeexplore.ieee.org/document/6771467 (visitado 17-06-2024).

[7] I. Sommerville, *Software Engineering* (Always Learning), Tenth edition, global edition. Boston Columbus Indianapolis New York San Francisco Hoboken Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo: Pearson, 2016, 810 págs., ISBN: 978-1-292-09613-1.

[8] M. Aberdour, «Achieving Quality in Open-Source Software,» *IEEE Software*, vol. 24, n.º 1, págs. 58-64, ene. de 2007, ISSN: 0740-7459. DOI: 10.1109/MS.2007.2. dirección: http://ieeexplore.ieee.org/document/4052554/ (visitado 18-06-2024).

[9] «¿Qué Es El Software Libre? - Proyecto GNU - Free Software Foundation.» (), dirección: https://www.gnu.org/philosophy/free-sw.es.html (visitado 18-06-2024).

[10] «Various Licenses and Comments about Them - GNU Project - Free Software Foundation.» (), dirección: https://www.gnu.org/licenses/license-list.html (visitado 18-06-2024).

[11] R. Cayo. «LPG-Bolivia-ADSIB.» (), dirección: http://sistemas.armada.mil.bo/capacitacion/licencia_LPG.php (visitado 18-06-2024).

[12] «The MIT License,» Open Source Initiative. (31 de oct. de 2006), dirección: https://opensource.org/license/mit (visitado 18-06-2024).

[13] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 7th ed. New York: McGraw-Hill Higher Education, 2010, 895 págs., ISBN: 978-0-07-337597-7.