Título

Custom GUIs App

**Miembros del equipo desarrollador:**

**Raimon**

**Daniel**

**Cristian**

**Projectes de les Tecnologies de la Informació - PTIN**

**UPC - EPSEVG**

**2013**

**Abstract**

Needs

Features

Benefits

**Índice**

1. Introducción

2. El equipo

3. La Especificación:

Objetivos y requerimientos

Visión del sistema

Actores y componentes

State-of-the-art y la selección de las tecnologías

Características y funcionalidades

4. Metodología y Plan de trabajo

5. Propuesta de costes

6. Análisis de riesgos

7. Indicadores y seguimiento

8. Acciones correctivas

9.Referencias

10. Apéndices

**Introducción**

El proyecto CUSTOM GUIs nace de la idea de desarrollo de una herramienta que permita a los usuarios la personalización de interfaces. En la universidad UPC - EPSEVG, concretamente en la asignatura Proyectos de las Tecnologías de la Información (PTIN) se propone el desarrollo de una herramienta que sea capaz de adaptar una GUI o Interfaz gráfica de usuario al gusto y deseo del usuario de la misma herramienta, con el objetivo principal de satisfacer sus necesidades según la problemática o uso que el usuario vaya a poner en práctica.

Partiendo de la premisa anteriormente citada, se lleva a cabo el inicio del proyecto CUSTOM GUIs, una herramienta constructora de interfaces gráficas para el usuario. Es necesario mencionar la naturaleza de la herramienta, siendo ésta un medio de programación el cual simplifica y asiste en la creación de interfaces gráficas del usuario que permite a un desarrollador o diseñador la visualización y automatización del código fuente partiendo de cada parámetro, correspondiente por ejemplo a widgets o módulos.

La herramienta CUSTOM GUIs pretende facilitar la creación, y aún más la personalización, de interfaces gráficas que servirán para distintos propósitos designados por el desarrollador, y más allá de ello ofrecer una versatilidad para el amplio abanico de posibilidades que ofrece hoy en día el mundo del software. En vez de centrarse en un único campo, como puede ser el de Networking o Management, la herramienta puede ser flexible para poder adaptarse y ofrecer soluciones a múltiples campos de las Tecnologías de la Información. De esta manera, puede decirse que el resultado del proyecto supone un medio para un fin.

El proyecto para el desarrollo de la herramienta cuenta con un periodo de tiempo limitado a 12 semanas, y es por ello que el desarrollo se destina a ciertas funcionalidades y características dentro del mismo, siendo el resultado final del proyecto de naturaleza abierta y extensible.

A continuación, tras esta introducción, se da paso al resto de la documentación que conforman el proyecto en sí, donde todos los aspectos aquí documentados pueden estar sujetos a cambios y no son representativos del producto final.

**1. Especificación:**

En esta primera sección de la documentación del proyecto se muestran los objetivos tomados para el cumplimiento del producto, se muestra un acercamiento y visión del sistema, como también su arquitectura. Se detallan los actores y componentes que conforman el entorno así como las interfaces que lo conectan y permiten interactuar los actores con los componentes.

Se detallan aspectos tecnológicos y alternativas, así como el plan de trabajo que conforma el proyecto, y un primer acercamiento a la implementación.

**Objetivos**

En este apartado se muestran los principales objetivos que marcan el desarrollo de la herramienta. Los objetivos mostrados aquí se han fijado partiendo de las limitaciones que muestra el proyecto, por lo que otros fueron descartados, pero no implican que sean los únicos ni tampoco excluyentes. Además, cabe destacar que todos los objetivos han sido fijados para su cumplimiento de forma progresiva y pueden variar de un modo u otro en el desarrollo final.

Los objetivos son independientes de cualquier tecnología, y su enumeración no indica ningún tipo de orden o prioridad; las prioridades se detallan en el plan de trabajo.

1. Edición de GUIs: Otorgar a los desarrolladores las herramientas que permiten crear interfaces, de manera personalizada, eliminando las complejidades que pueden suponer los desarrollos de interfaces de uso final, facilitando la interacción con el usuario y con la meta de aumentar la productividad.

2. Versátil: Capacidad de permitir a los desarrolladores la implementación de interfaces, de forma multi-propósito permitiendo su desarrollo desde cero o a través de plantillas, con diversas opciones de personalización, exportación e importación.

3. Modular y extensible: Disponer de un entorno de trabajo sobre la aplicación que permita ofrecer la facilidad de interacción con el desarrollador y la capacidad de extensión con la cual se puedan incluir nuevas características al conjunto en un futuro si es deseado.

4. Independiente: La herramienta debe de ser capaz de funcionar de forma independiente a otro tipo de software aplicación y de sistema operativo en el cual se vaya a utilizar.

5. Personalizable: La herramienta debe tener la capacidad de adaptarse a la visión del desarrollador y a las necesidades de cada proyecto, ofreciendo sistemas de modificación de interfaz, módulos, widgets y entornos.

6. Importación y exportación: La herramienta debe permitir a los desarrolladores poder extraer el desarrollo de su interfaz a ciertos formatos que luego puedan ser utilizados con terceras aplicaciones para el proyecto designada, así mismo como importar y adaptar interfaces previamente desarrolladas.

7. Caja de herramientas

8. Las herramientas

9. Edición de propiedades

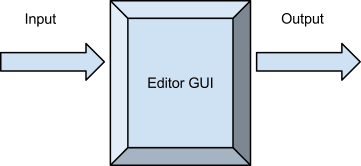
**Visión del Sistema y Arquitectura**

En este apartado se muestra una visión global del sistema y la estructura que conforma el entorno.

La arquitectura del sistema: con un enfoque global y en primera instancia el sistema (el conjunto de elementos que forman la totalidad de la herramienta) está compuesto por la herramienta en sí, además de la entrada y la salida de datos, que llamaremos Input y Output y que son los componentes que dan funcionalidad al sistema.

El Input serán todos aquellos datos que el usuario de la herramienta introduzca , ya sea código fuente, templates, importaciones, otro tipo de datos e incluso interacción.

El Output serán todos aquellos datos que el usuario reciba o extraiga de la herramienta, que mediante el entorno de usuario permita obtener datos guardados, exportados, así como también interfaces editadas con la herramienta, donde el usuario puede recoger en el formato deseado el producto de su desarrollo (por ejemplo en XML).



Imágen esquemática de interacción y flujo de datos de la arquitectura.

Dada la naturaleza abierta de la herramienta, y que uno de sus objetivos principales es la modularidad y extensibilidad, otros ejemplo de Input y Output son la capacidad de introducir nuevas funcionalidades, características, módulos, como también la exportación de éstos si a través de la herramienta se efectúa algún tipo de modificación, como es la personalización en sí.

A continuación se muestra un enfoque más concreto de la arquitectura del sistema con el objetivo de estructurar el entorno que presenta la herramienta.

El modelo de capas de la arquitectura muestra las distintas capas en orden jerárquico que forma el conjunto de la herramienta. Como se muestra a continuación las capas y el orden son los siguientes: Interfaz, Módulos, Aplicación, Sistema Operativo, Hardware. Cada capa actúa a la vez con la inferior y la superior, excepto la última (Hardware) y la primera (Interfaz). La capa de Interfaz es el enlace del usuario (o desarrollador) que conecta la interacción de éste con el resto del sistema, mientras que la capa de Hardware es la base donde se asienta el sistema.

A continuación una imágen ilustrativa de la arquitectura estructurada en capas y como el usuario interactúa con el sistema:





|  |
| --- |
| **Interfaz** |
| **Módulos** |
| **Aplicación** |
| **Sistema Operativo** |
| **Hardware** |

Modelo de capas de la arquitectura de la herramienta

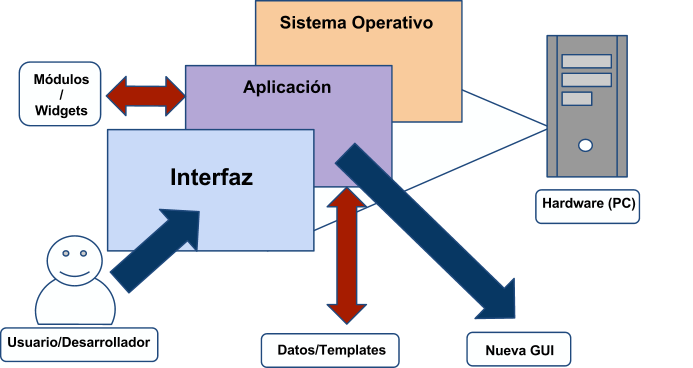
**Actores y componentes (sin terminar)**

Los actores y componentes que conforman el entorno de la herramienta son los siguientes:

|  |  |
| --- | --- |
| **Actores** | **Componentes** |
| Usuario/Desarrollador | Interfaz |
| Datos Importados/Exportados | Módulos |
| Templates | Extensiones/Widgets |
| Aplicación |  |
| Sistema Operativo |  |
| Hardware |  |

**Interfaces (sin terminar)**

Interfaces que relacionan e interactúan entre los distintos actores y componentes que conforman el entorno de la herramienta:



**El equipo (sin terminar)**

**State-of-the-Art y selección de tecnologías (sin terminar)**

**QT Creator**

He estado documentandome sobre Qt y alternativas debido a que por lo que he visto [Qt Designer puede funcionar para cualquier lenguaje de programación soportado por Qt](http://en.wikipedia.org/wiki/Qt_%28framework%29#Bindings).

Además Qt tiene restricciones en su licencia de uso, de los cuales no me he informado pero sé que por hacer determinadas cosas tienes que pagar. Como por ejemplo compilar las librerias de Qt dentro de tu mismo programa para crear solo un archivo binario sin dependencias.

**WxWidgets**

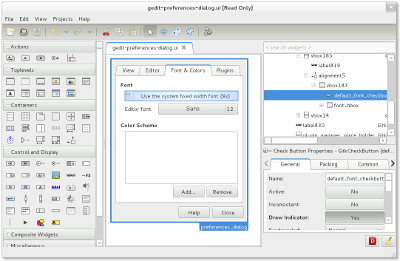
Descripción

**GTK+**

Sobre wxWidgets había mucha opinión variada, pero he pasado directamente a la [página](http://www.gtk.org/) de **GTK+**.

**Licencia**: licensing terms for GTK+, the [GNU LGPL](http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html), allow it to be used by all developers, including those developing proprietary software, without any license fees or royalties.

**Características**: “superb performance”, [language bindings](http://www.gtk.org/language-bindings.php), cross-platform, **Theme support**, **GtkBuilder** (creates your user interface from XML) y mobile noseque... parecen widgets para móvil como jquery mobile pero creo que solo son compatibles con determinados móviles.

GTK+ También dispone de una RAD para diseñar interfaces llamada [Glade](http://glade.gnome.org/):

Open source, pero solo permite diseñar los distintos tipos de ventanas y exportarlas a XML, luego cargarlas a tu programa mediante código llamando al GtkBuilder que construye una interfaz mediante XML. Esta aplicación nos permite añadir nuevos widgets e [integrar la “Glade UI Designer core library” en nuestra aplicación](http://developer.gnome.org/gladeui/3.6/) como ha hecho el IDE [adjunta](http://projects.gnome.org/anjuta/), que ha integrado Glade dentro de su aplicación. (INFORMACIÓN INCORRECTA)

Glade no dispone de proyectos ni edición de código, sólo crear nuevas ventanas y obtener su XML. Tampoco dispone de templates.

Propongo tres posibles opciones si acabamos escogiendo GTK+ en vez de Qt::

1. Programar nuestro propio Glade, aka reinventar la rueda (pues vaya)
2. Hacer un fork del código fuente para añadir las opciones de proyectos y templates, de forma que se puedan crear proyectos para un lenguaje específico y ya te cree el código base pudiendo elegir un “Theme” específico o importar otro.
3. Crear nuestra aplicación que permita edición de código, creación de proyectos, etc e integrar el “Glade UI Designer core library” para la edición de GUIs. (INCORRECTO)

Como editor de código existe un componente llamado [Scintilla](http://www.scintilla.org/), compatible con GTK+ que ya nos hace toda la faena de syntax highlighting, code completion, indicación de errores y esas cosas.

Por simplicidad, yo me quedaría con la segunda opción, aunque conlleva su faena de entender la estructura del código fuente de Glade.

EDIT: [Tabla de lenguajes](http://en.wikipedia.org/wiki/GTK%2B#Programming_language_bindings) por los que se han hecho bindings de GTK+.

En la mayoría de lenguajes **conocidos** se puede trabajar con GTK+ en cualquier plataforma, excepto el binding de java, que no funciona en Windows.

# **JUCE (Jules' Utility Class Extensions)**

JUCE es otra librería gráfica realmente potente y con un gran diseño. [INFO AQUÍ](http://wiki.gosub100.com/doku.php?id=juce:que_es_juce), [Más info y ejemplos](http://wiki.gosub100.com/doku.php?id=juce:inicio). (En castellano)

*JUCE (Jules' Utility Class Extensions) is an all-encompassing C++ class library for developing* ***cross-platform*** *software.*

*It contains pretty much everything you're likely to need to create most applications, and is particularly well-suited for building highly-customised* ***GUI****s, and for handling* ***graphics*** *and* ***sound****.*

## Aplicaciones comerciales que usan JUCE

### MAGIC: Multitrack Audio & Graphics Interactive Composer

Podéis encontrar un listado más amplio [aquí](http://www.rawmaterialsoftware.com/wiki/index.php/3rd-party_JUCE_Applications). [Ableton Live](https://www.google.es/search?q=ableton+live&oe=utf-8&aq=t&rls=org.mozilla:es-ES:official&client=firefox-a&um=1&ie=UTF-8&hl=en&tbm=isch&source=og&sa=N&tab=wi&ei=0uAwUeClEarR7AbT2oCwBA&biw=1440&bih=707&sei=0-AwUa_4Ks6WhQeInoCICQ#q=ableton+live&um=1&hl=en&client=firefox-a&rls=org.mozilla:es-ES:official&tbs=isz:lt,islt:xga&tbm=isch&source=lnt&sa=X&ei=4-AwUbnWCJOWhQeyiIDACw&ved=0CB0QpwU&bav=on.2,or.r_gc.r_pw.r_cp.r_qf.&bvm=bv.43148975,d.ZG4&fp=4cc2d6fcac02c097&biw=1440&bih=707), aunque no presente en la lista, también usa JUCE. Como podéis observar casi todas las aplicaciones están relacionadas con el audio debido a las potentes APIs de audio que ofrece JUCE.

## Demo ejecutable mostrando las funcionalidades de JUCE

[DEMO](http://188.165.195.47/public/JuceDemo.zip) con muchas de las funcionalidades de JUCE, el aspecto es pésimo ya que para demostrar su potencial colorea las cosas a su gusto sin seguir un patrón de diseño. Ésta demo la he compilado con Visual Studio 2010 SP1 Express mediante uno de los códigos fuente de ejemplo con los que viene JUCE.

Enlaces de interés:

<https://www.rawmaterialsoftware.com/wiki/index.php/ValueTree_Guide>

<http://code.google.com/p/juced/wiki/JuceTutorial>

**Metodología**

# **[ PROTOTIPO: 3 Fases ]**

### [ LEYENDA ]

**Fase inicial**: Objetivos que deberíamos cumplir en unas 2-5 semanas.

**Fase avanzada**: Objetivos a cumplir dentro del período de entrega.

**Fase extendida**: Objetivos que están **fuera del alcance del proyecto** debido al corto plazo de tiempo.

## Visión general

Nuestro objetivo principal tendría que ser el desarrollo de un conjunto de componentes que puedan ser añadidos a cualquier proyecto desarrollado en JUCE, particularmente, Introjucer. No podemos desarrollar estos componentes dentro del proyecto Introjucer ya que éste actualiza su código fuente con frecuencia y tendríamos que estar siempre adaptándonos a las nuevas versiones. De modo, que partiremos de un proyecto vacío con una ventana y su correspondiente contenedor de componentes. En este contenedor de componentes tenemos que inicializar nuestro editor y llamar a sus mètodos para obtener los distintos componentes necesarios para el editor, como las toolbox o la ventana de propiedades de un componente.

Entonces, tenemos un contenedor de componentes en nuestro programa principal, que es quien inicializa el editor, no el propio editor, su constructora tiene que ser parecido a:

//Inicializamos el editor

Editor editor = new Editor();

//Obtenemos una toolbox

Toolbox \*toolbox = editor->createToolbox(...);

//Añadimos componentes a la toolbox

toolbox->addItem(“juced\_Window”, “Window”, BinaryData::window\_png, ...);

toolbox->addItem("juced\_Label", "Text label", BinaryData::label\_png, ...);

...

//Añadimos la toolbox al contenedor de componentes actual

this->addAndMakeVisible(toolbox);

Y lo mismo vendría a ser para la ventana de propiedades, tiene que ser obtenida a través del editor para luego facilitar al desarrollador que incorpore el editor en su aplicación, así podrá poner dichos componentes en donde él crea que tienen que ir y no necesariamente dentro del componente editor.

## VALUETREE

## Características [Editor]

1. **Toolbox**: Permite seleccionar una herramienta que corresponde a un componente para ser dibujado en el editor.
2. **Interacción con el editor**:
   1. Pinta el área que va a ocupar el nuevo componente mientras el usuario lo dibuja en el editor. Crea un nuevo componente dentro del componente en el que empezó el drag&drop.
   2. Selecciona un componente en el editor para poder editar sus propiedades, redimensionar y desplazar..
3. **Propiedades**: Permite modificar los atributos de un componente
4. Componentes capaces de ser dibujados por el editor:
   1. **Window**
   2. **Label**
   3. **Textbox**
   4. **Button**
   5. **Listbox**
   6. **Combobox**
   7. **Tabs**
   8. **ImageButton**
   9. **TreeView**
   10. **Slider**
   11. **Viewports**
   12. **Tables**
5. **Historial de cambios**: Permite undo/redo mediante el componente ValueTree.
6. **Exportar**: Guarda la estructura de componentes y sus atributos en un archivo XML.
7. **Importar**: Carga la estructura de componentes y sus atributos de un archivo XML.
8. **Generar código**: genera los archivos de código fuente necesarios para poder compilar la interfaz.
9. Formas de editar el código por parte del usuario:
   1. **Gestionar eventos**: Proporciona funciones en archivos de código fuente separados del código generado para el tratamiento de eventos.
   2. **Edición directa de código**: Permite al usuario añadir líneas de código dentro del código generado, manteniendo un constante registro de cambios.
10. **Componentes externos**: Permite al usuario añadir librerías externas (\*.dll) al componente editor.

## Características [Aplicación]

Hay otras características que no forman parte del componente editor sino de la aplicación que alberga el editor, Introjucer:

1. **Plantillas**: Permite seleccionar un modelo de aplicación específico al crear el proyecto.
2. **Creación de componentes**: Permite crear un proyecto componente para ser compilado como una librería externa (\*.dll).
3. **Añadir componentes personalizados**: Permite al usuario incorporar librerías externas de componentes en el editor, así como mostrar su herramienta para dibujarlos en una de las toolbars.
4. **Interfaz para interactuar con el editor**: Proporciona componentes que permiten importar, exportar, gestionar los atributos del editor (p.e. ancho de la rejilla de fondo).

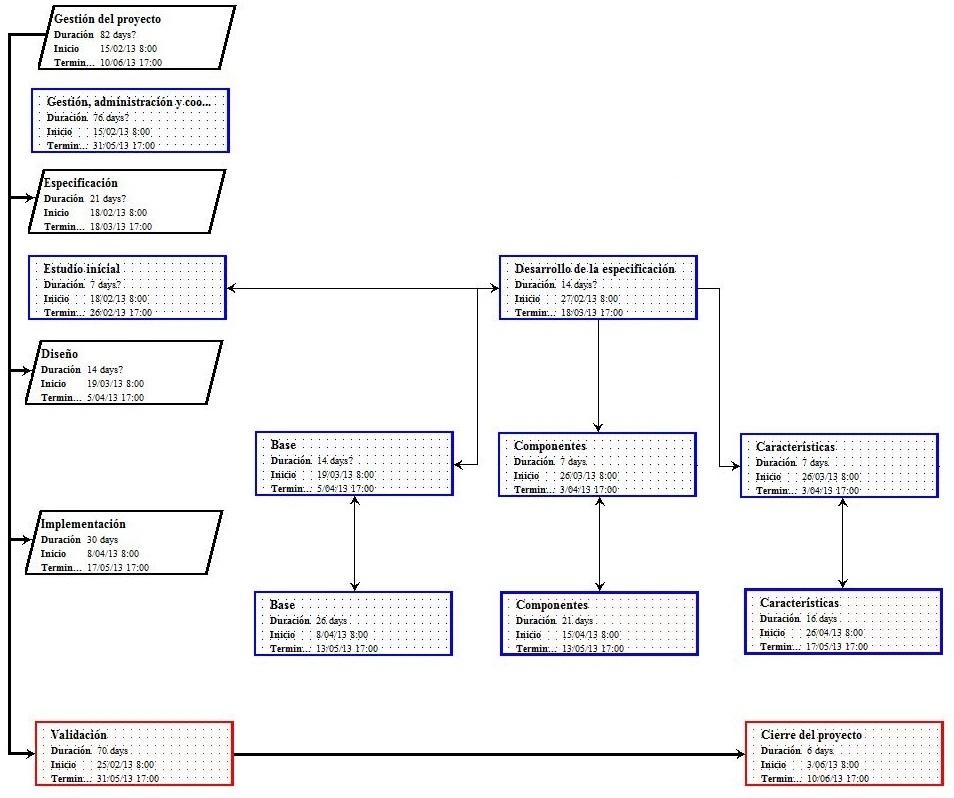
**Paquetes de Trabajo (sin terminar)**

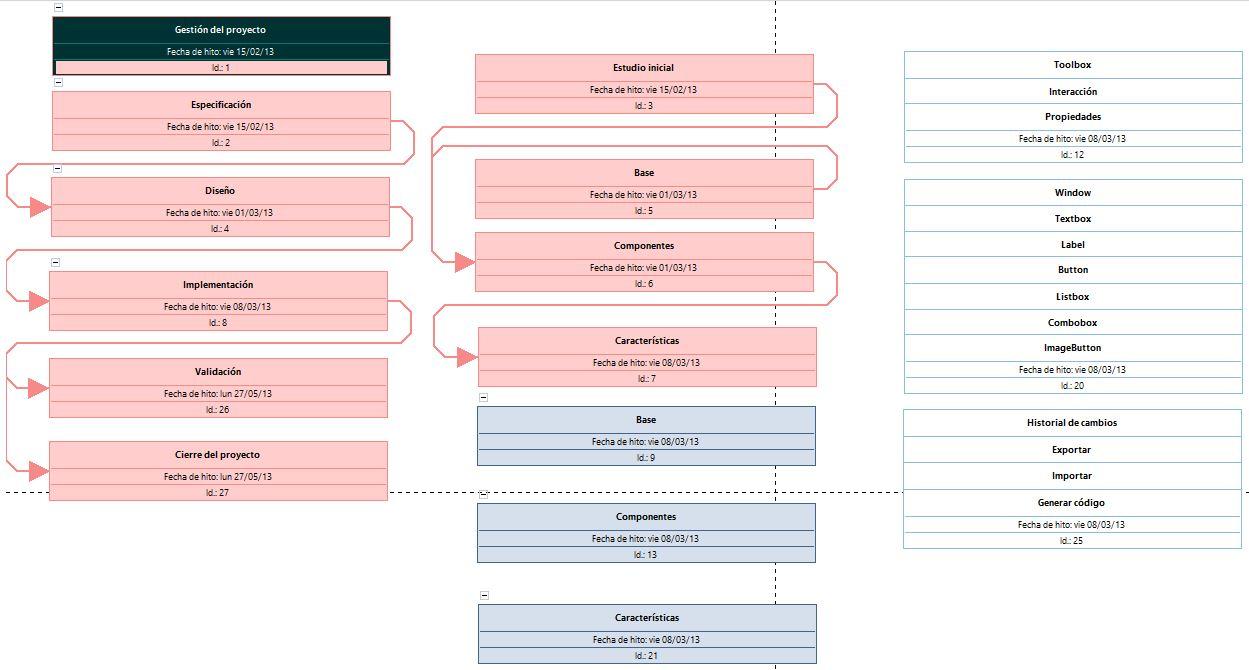
Descripción

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nº PT** | **Paquete Tarea** | **Semana inicio** | **Semana final** | **Implicación** |
| **1** | **Gestión del proyecto** | **S1** | **S12** | **4** |
| **2** | **Especificación:**  **- Estudio inicial** | **S1** | **S3** | **2** |
| **3** | **Diseño:**  **- Base**  **- Componentes**  **- Características** | **S3** | **S6** | **3** |
| **4** | **Implementación:**  **- Base**  **- Componentes**  **- Características** | **S4** | **S11** | **23** |
| **5** | **Validación** | **S3** | **S12** | **2** |
| **6** | **Cierre del proyecto** | **S11** | **S12** | **2** |

**Diagrama de PERT**

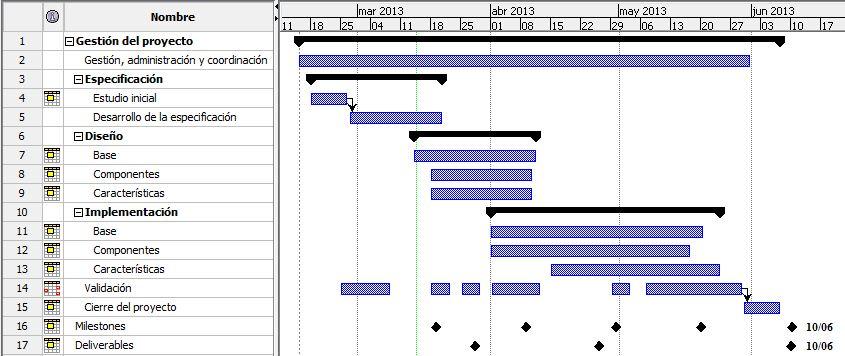
Descripción





**Diagrama de GANTT**

Descripción





**Propuesta de costes**

|  |  |
| --- | --- |
| **Costes directos** |  |
| **Nóminas** |  |
| *Project Manager (170 h)* | (22 €/h) 3.740 € |
| *Analista/Programador (480 h)* | (18 €/h) 8.640 € |
| *Programador (400 h)* | (15 €/h) 6.000 € |
| **Viajes** |  |
| Transporte (x7) | (60 €/u) 420 € |
| Dietas (x7) | (20 €/u) 140 € |
| **Equipamiento** |  |
| **Hardware** |  |
| *Portátil Toshiba Portégé Z830* | 999 € |
| **Software** |  |
| *Visual Studio Professional 2010* | 615 € |
| *Visual Studio Professional 2010 con MSDN* | 1.477 € |
| *Juce (x2)* | (891 €/u) 1.782 € |
| *Linux Redhat Enterprise Edition* | 138 € |
| *Microsoft Office Professional 2010* | 125 € |
| **Costes indirectos** |  |
| **Mantenimiento oficina** |  |
| *Servicio de limpieza (x3)* | (300 €/mes) 900 € |
| **Comunicaciones** |  |
| *Movistar Fibra Óptica 100Mb (x3)* | (60 €/u) 180 € |
| **Otros gastos** |  |
| *Luz (x3)* | (60 €/u) 180 € |
| *Agua (x3)* | (30 €/u) 90 € |
| *Comunidad (x3)* | (50 €/u) 150 € |
| *Alarma (x3)* | (99 €/u) 297 € |
| **Seguros** |  |
| *Generali Seguros (Garantías básicas + R.C. + Cristales + Robo)* | 204 € |

**Análisis de riesgos**

En este apartado se muestran los principales riesgos asegurables que están presentes desde el inicio del proyecto (incluyendo las negociaciones iniciales y la firma de contratos) hasta la finalización del proyecto y futuros sucesos y/o implicaciones del producto. En el análisis de los riesgos se muestran las medidas oportunas a tomar en caso de la materialización de tales sucesos:

Cambios en las condiciones del mercado interno y externo: Para el transcurso de tiempo estipulado para el desarrollo del proyecto cabe tener en cuenta otros productos que pudiesen aparecer en el mercado como competidores directos, para ello se tomarían acciones rectificatorias incluyendo estudios del mercado y la posible renegociación del contrato entre cliente / desarrolladora.

Accidentes y daños materiales: Para los posibles daños que afecten a las instalaciones o el material necesario para llevar a cabo el desarrollo del proyecto se cuenta con un seguro de cobertura.

Accidentes y daños personales: Para los posibles daños que afecten el personal involucrado en el proyecto, ya sea por accidente laboral o extralaboral se cuenta con un seguro de cobertura que permita solventar este tipo de sucesos.

Modificación de política y efectos legales: Existen métodos que se encargan de cubrir la necesidad de mantener dentro de la legalidad todo desarrollo tanto tecnológico (posibles patentes y/o licencias) así como el cumplimiento de los términos acordados para el producto final, tomando acciones sujetas a leyes y su actuación pertinente.

Cumplimientos de términos y contratos: Cobertura jurídica que protege y cubre el cumplimiento y gastos que puedan comportar tales sucesos, quedando sujeto a ley con la denuncia pertinente.

Errores, robo o intrusión externa: En el caso de posibles errores que puedan afectar al desarrollo del producto final existe una cobertura que permite proteger el desarrollo del proyecto sin implicar pérdidas para el cliente. En el caso de espionaje externo o robo de material se mantiene la cobertura total y metodologías jurídicas para perseguir tales sucesos tomando medidas sujetas a ley.

**2. Diseño**

**3. Implementación**