ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL

DEPARTAMENTO
de Ingeniería de Sistemas y Automática

ÁREA
Ingeniería de Sistemas y Automática

CONTROL EN POSICIÓN DE UN EJE DEL ROBOT SCORBOT-ER 4PC

Autor: Enrique Arrabal Almagro

Director:Víctor Torres López

Titulación: Graduado en Ingeniería en Tecnologías Industriales

MÁLAGA, septiembre de 2015

Índice General

[scale=0.25]./1-Portada/img/uma
[scale= 0.25]./1-
Portada/img/etsii

List of Figures

Lista de Tablas

1 Introducción

Se va a afrontar un proyecto de desarrollo de software desde el punto de vista de la ingeniería industrial. En mi experiencia profesional he podido ver cómo se aprecia el valor añadido que aporta la visión de nuestra profesión. Al fin y al cabo en la gestión de cualquier proyecto encontrar puntos en común, optimizar, estandarizar, documentar y sacar el mayor partido a los recursos de los que se dispone es el núcleo de lo que se enseña en esta escuela.

Los principales problemas del desarrollo comercial de software no difieren de la gestión de proyectos de cualquier industria. Vamos a analizarlo centrándonos en las bases más obvias:

- desafío técnico
- gestión de tiempo/presupuesto
- adaptación al cambio

Cuando surge una nueva disciplina el desafío técnico lo encara cada artesano e innova en su técnica dando lugar a los productos y resultados dispares, los clientes tienen complicado tener una referencia para saber lo que compran.

Cuando los proyectos son más grandes el artesano empieza a ser un factor de riesgo. La elección de un profesional con respecto a otro puede malograr ingentes cantidades de recursos y es difícil para el profesional garantizar la consistencia en sus resultados.

Quizá es en el último punto, la adaptación al cambio, donde la industria del software se diferencia un poco más del resto de industrias. El proyecto de software nunca termina. Es extremadamente volátil, caduco y cambiante.

En cada uno de estos puntos hay cantidad de métodos, herramientas, procesos y literatura que va encaminada a reducir la incertidumbre. La mayoría tienen más de 20 años y los más novedosos suelen ser re-formulaciones. A pesar de su utilidad demostrada y su antigüedad relativa no han penetrado en el software comercial a todos los niveles. Muchas veces porque el cliente no sabe de su existencia o no está dispuesto a pagar por ello. A nadie se le ocurriría comprar un coche sin homologación que garantice que ha pasado por las pruebas de seguridad correspondiente. En el software, sin embargo, se siguen vendiendo programas sin las más básicas pruebas de seguridad o garantía de ningún tipo.

Quizá la clave está en entender precisamente que el proceso de industrialización de una disciplina es un proceso más largo del que nos gustaría a veces. De hecho, la misma palabra encontrará un rechazo fuerte en el ecosistema del software, al considerarlo un arte que se escapa de la posibilidad de convertirlo en un proceso en cadena. No es esa la visión de industralización que se va a abordar si no el cómo el ofrecer unas garantías de calidad, seguridad, fiabilidad estándar y homogeneidad.

Las principales herramientas a día de hoy que responden a los diferentes puntos clave son:

- respecto a cómo se va entregando valor de forma iterativa: scrum, kanban, agile, extreme programing...
- respecto a cómo se garantiza cumplir la expectativa del cliente con el producto: Testing (TDD,BDD...) diseño de casos de uso, diagramas UML, UX, UI

- respecto a cómo se garantiza la posibilidad de cambio: arquitectura de capas o hexagonal, DDD...

De todas ellas, hemos escogido como relevantes para este trabajo las herramientas de diseño que garantizan la posibilidad de cambio en el futuro y el cumplimiento de las expectativas del cliente: arquitectura de capas, DDD y testing. frente a otras herramientas enumeradas.

Con respecto a la arquitectura de capas hago una reflexión: hablar en términos del negocio que se quiere controlar y separarlo de los detalles técnicos garantizan que lo que se construye es lo que se espera. Soy un firme defensor que la parte del código por la que realmente paga un cliente debería ser capaz de leerla él mismo, es decir, que sea legible por gente ajena al mundo del desarrollo de software. Para ello se debe hacer un esfuerzo en entender qué forma parte de la solución que se está ofreciendo y qué es una simple herramienta o detalle para conseguirlo. Pongo un pequeño ejemplo, pero profundizaremos más en esta reflexión a lo largo del proyecto: El funcionamiento de un semáforo, en su forma más simple, es que cada x segundos se enciende una luz y se apaga la otra, alternativamente. que haya un cableado por dentro, un microcontrolador o un señor apretando unos botones mirando un cronómetro no es lo relevante. Por lo que paga realmente el cliente es por la programación del comportamiento, y este debe ser legible para el mismo para poder evaluar si responde a sus expectativas. Los detalles técnicos, como no puede ser de otra forma,tienen su importancia, pero componen a mi parecer una segunda derivada.

Por último la reflexión con respecto a la gestión de la creación de software a la que le doy más importancia a día de hoy con mi experiencia actual es que el coste mas importante siempre es el mismo: tiempo de lectura de código. Además es un coste que no añade valor. Por el que el cliente no va a pagar de más. Es un coste oculto que dificulta el cambio. Dificulta encontrar profesionales que quieran hacerse cargo de él y otros efectos colaterales negativos. Es por esto que hay que reducirlo al máximo.

Es en gran parte la motivación de escoger Golang como lenguaje de programación, ya que se promete como uno de sus puntos fuertes. Se ha escogido por ser un lenguaje en crecimiento y con unas carácterísticas técnicas interesantes a investigar para determinados casos de uso, ya que al fin y al cabo es una herramienta y como tal hay muchas parecidas, al final la elección de una herramienta puede dar una ventaja en algún aspecto en particular pero el factor diferencial está en quién lo usa y cómo.

2 Objetivos

Crear un sistema capaz de albergar y gestionar tareas para su ejecucion en remoto.

Para ellos se requerira un servicio web API (application programming interface) y dos servicios clientes uno para manejarlo el frontend y otro para ejecutar las tareas

Para hacer uso en mayor medidas de la herramienta escogida y para explorar su uso en nuestro campo la tarea a ejecutar en remoto sera un control PID en velocidad y posición de un motor de corriente continua.

Tendremos la capacidad de:

- Añadir editar, eliminar y obtener las tareas contenidas en el sistema
- Añadir editar, eliminar y obtener los resultados de la ejecución de dichas tareas

- Añadir editar, eliminar y obtener las direcciones de los equipos remotos donde se ejecutarán las tareas

El servicio cliente remoto tendrá la capacidad de recibir un comando, ejecutarlo y devolver el resultado.

3 Estructura de la memoria

La estructura del presente documento va a perseguir mostrar el proceso como si de un proyecto real se tratara. Se va a comenzar presentado las herramientas que se pretende utilizar. Luego un dise;o a modo de proyecto tal y como se le presentaria a un cliente para que aprobara su comienzo de ejecucion. Se va a proceder a describir los hitos importantes en el desarrollo del mismo y las adaptaciones que surgen debido a imprevistos y descubrimientos de la propia investigacion contemplada en el proyecto

Proyecto básico: se va a diseñar el sistema objetivo tal para su presentación y aprobación por cliente. Está compuesto por una memoria descriptiva, constructiva y económica Ejecución del proyecto: se ha documentado el desarrollo del proyecto con sus distintos hitos y modificaciones requeridas por el devenir de la ejecución Aplicaciones del sistema: pruebas y resultados. Latencias! Control de luces Control del motor Conclusiones: se expondrán las distintas conclusiones que se han extraido con respecto a los puntos objetivo. Golang, protocolo de comunicación, diseño de software y testing Lineas Futuras