UNIVERSIDAD DE ALCALÁ DE HENARES

Escuela Politécnica Superior

Proyecto de fin de master

Marco de calidad mediante algoritmos evolutivos

Autor: Javier Ruano Ruano

Director: Luis Fernández Sanz

Madrid

Proyecto de fin de master

Universidad de Alcalá de Henares

Marco de calidad mediante algoritmos evolutivos

Madrid 2017

Quería agradecer a mi familia, su presencia en los malos y buenos momentos, y a los profesores del master de Dirección de proyectos informáticos, Jose Amelio Medina Merodio y Luis Fernández Sanz, la pasión por la asignatura de calidad que inculcan, sobre todo Luis Fernández, sabe guiar hacia unas pautas del buen hacer y el saber expresar.

Javier Ruano.

Índice

Contenido

Tabla Ilustraciones

1 Introducción

1.1 El objeto del estudio

El presente estudio versa sobre la aplicación de un marco de calidad. Para ello se ha recorrido el ámbito de la planificación como parte de la Ingeniería de Software y centrado en el conjunto de las pruebas para asegurar unos requisitos mínimos.

Es un tema muy interesante puesto que en la actualidad la Ingeniería del Software este muy influenciado por las metodologías ágiles, que permiten un desarrollo muy rápido y casi la integración de equipos de pruebas en los desarrollos.
Aún así he analizado el conjunto de métricas de calidad como un proceso laborioso, que no comienza, ni termina en la entrega del producto, sino en la atención plena de las metas que se quieren alcanzar en cuanto a costes, planificación sino en la monitorización y medidas correctores.

En la paciencia y constante verificación a través de la transparencia del conjunto de pruebas y pudiendo seguir las trazas, y con unos estándares destinados a seguir pautas de calidad.

La línea que se ha seguido es la de analizar la evolución en la Ingeniería del Software, sobre todo en el contexto de las pruebas de validación del producto, el por qué se requiere planificar en cuanto a unos recursos, en el mundo de la algoritmia cuán costoso es y marcar la diferenciación entre pruebas desde las etapas a pruebas iterativas.

Al nivel computacional costoso, se ha hecho un estudio de los algoritmos evolutivos, cuya inspiración biológica, añade simplicidad en la búsqueda de una planificación exitosa, con unos datos con incognitas impredecibles.

Los algoritmos evolutivos crean respuestas en ese tipo de problema con tiempos de espera inciertos y tiempos de terminación de las tareas también desconocidos.

Esas respuestas suelen ser buenas, y es lo que se intenta transmitir.

Para ello he elaborado un Planificador genético que recibe los resultados de unas pruebas, con información sobre los componentes Software y las incidencias y decide como se llevarán a cabo las modificaciones para mejorar la calidad en cuanto a defectos.

1.2 Ámbito

En la investigación se analizarán los modelos de planificación, desde el análisis de su complejidad y su labor en la ciencia de la ingeniería del Software y las técnicas de mejora de la calidad del software, con uno de los pilares como es la fase de pruebas, destinadas a la mejora del producto software, de cara al consumidor y a los requisitos

Y los algoritmos genéticos destinados a la planificación de tareas.

1.3 Objetivos

El objetivo general del presente trabajo es comprender el funcionamiento dentro de la Ingeniería del Software, de la fase de pruebas y la integración e importancia de esta en la mejora de la calidad Software.

Para ello se plantean como objetivos específicos:

- Estudio del ciclo de vida del software incidiendo en las pruebas y validación, de acuerdo con unos estándares.
- Estudio de los principales precursores en la utilización de los algoritmos evolutivos en la planificación de proyectos.
- Formas de representar las tareas de forma que resulten manejables por los algoritmos.
- Estudio de la eficiencia de las distintas formas de representación.
- Análisis del caso de estudio.
- Elaboración de estructuras de datos para elaboración de métricas.
- Empleo de algoritmos genéticos a través de la librería Galib para aplicación de las métricas en la ordenación de la planificación.

1.4 Metodología

Se ha realizado una introducción de carácter histórico sobre la Ingeniería del Software como de la Calidad. Esas empresas y esos pioneros dejaron su impronta en cuanto a ver el mundo del Software destinado a un tipo de clientes y señalando que ralentizaba, menoscababa o de que adolecía la calidad.

En el primer capítulo se realiza ese recorrido histórico, con unos precursores empresariales bastante claros, lo militar. Ese secretismo va dejando paso a un conjunto de profesionales bien preparados con una metodología que se empieza a compartir, y un público bien distinto.

A través de esa diversificación, analizo como las principales dificultades de los jefes de proyecto consisten en los recursos, las planificaciones, las medidas de calidad, y la predicción de defectos.

Es muy importante simplificar en que las pruebas añaden valor sobre el software, no quedando relegado a un espacio de consumidor de software carente de conocimiento que a través del manual no cometerá errores.

En el segundo capítulo he analizado la forma en que los algoritmos genéticos son la pieza clave en la planificación.

Distintos tipos de representación, distintos espacios de codificación por tanto y produciendo todo tipo de soluciones. Hay que recalcar que, sin hablar mucho de algoritmos, los algoritmos genéticos son un ejemplo de cómo una buena estructuración de los datos de cara a producir un cromosoma permite definir un marco de problema distinto, más sencillo o más complicado y conducir las respuestas a través de la función de aptitud, lo que sería la evolución o el azar hacia la solución.

En el último capítulo se estudian los datos aportados por el profesor Luis Fernández Sanz, de cara a su aprovechamiento en cuánto a la definición de indicadores para encauzar la planificación según unas métricas de calidad.

Esos datos los he manipulado desde la base de Datos MS SQL, transformándolos en un fichero mdf, y trabajando sobre dicha base de datos con Linqpad, para la generación de mis propios csv.

A través de R y de MS SQL he analizado los datos para ir generando distintos csv, que me fueran de utilidad.

Esto lo he podido llevar a cabo gracias al lenguaje de programación C++, de su creador Stroustrup y a Matthew Wall, creador de Galib, biblioteca que permite la abstracción de los algoritmos genéticos. Con estos dos elementos que desarrollado un Planificador genético que permite ordenar las incidencias de acuerdo con las métricas que se hayan aplicado.

2 Contexto teórico

2.1 Ingeniería del Software

2.1.1 Introducción

Al considerarse un proyecto como un esfuerzo temporal que se lleva a cabo para crear un producto, servicio o resultado único. Surge el esfuerzo del jefe de proyecto de optimizar tanto el presupuesto como mantener la calidad bajo control. Esta tarea se puede llevar sostener gracias al ciclo de vida del software, dónde se definen unas etapas para planificar, diseñar, implementar, probar, validar y entregar el producto al cliente.

Corresponde al jefe de proyecto el maximizar los esfuerzos destinado a la aceptación del producto final, ya sea mediante estrategias ya utilizadas con éxito, como con seguimiento y reestructuraciones en la planificación que se lleva sobre el proyecto.

El seguimiento es muy importante tanto para saber si se están cumpliendo objetivos, así como para tomar medidas correctoras.

La calidad es uno de los pilares dentro de la sociedad del bienestar y del libre comercio, casi más importante que el precio. Por ello que las medidas correctoras hayan sido un principio motor sobre el que mejorar el producto hacia los criterios del cliente, tanto en la eliminación de defectos como en la adecuación a sus requisitos, es vital.

Las formas de planificar las pruebas dentro de la aproximación a la conclusión del proyecto y entrega de los productos, debería confirmar el esfuerzo realizado hacia la satisfacción del cliente, evitando así quejas, reclamaciones y pérdida de prestigio como empresa.

Por ello se diseñan pruebas que cubran el mayor número de casos posibles que el producto se pueda encontrar, la finalidad es que se pueda garantizar la calidad.

Los algoritmos evolutivos tienen una fama contrastada en la búsqueda heurística, tanto para la producción de casos de pruebas como para la planificación evitando caer en óptimos locales.

Esto es un esquema a seguir dentro del control de la calidad, del que se plantea realizar un estudio, desde el análisis de la planificación de proyectos mediante algoritmos evolutivos, así como el desempeño de un caso de estudio, sobre los resultados de pruebas de validación de un producto.

Lo adecuado de la búsqueda de una solución global de los algoritmos genéticos destinada a la calidad.

Esto hace plantear el diseño de elementos de ayuda a la toma de decisión basados en los algoritmos evolutivos, haciendo bueno el esfuerzo siempre que produzca un acercamiento a la predicción y corrección de defectos.

2.1.2 Precursores

La forma de gestionar el proceso de creación del software intenta gestionar tanto las fases para su elaboración, el trato con el cliente, y supeditarlo a la dirección de la empresa, conocida como línea de negocio. Bajo la línea de negocio se intentarán seguir unas pautas para alinear la producción con las pretensiones de la empresa.

Pero desde la segunda guerra mundial, dónde las innovaciones estaban al servicio de la maquinaria de guerra, este ámbito de la ingeniería ha sufrido muchos cambios. En los años 50, por inercia de la segunda gran guerra, más del 50 % de la creación de software pertenecían a defensa y al ámbito científico, apenas un 10 % al gobierno.

Hacía los años 60 pasa al ámbito de la economía y nacen grandes compañías como IBM, y el gran ámbito de las bases de datos. La carrera de ingeniero de software

no existía y los programadores que se contrataban iniciaban su andadura en la materia en ese mismo momento. El número de lenguajes de programación comenzó como norma el ensamblador, y el número a gran velocidad. Esto trajo consigo la dificultad de continuar aplicaciones heredades, manejar los lenguajes de programación que hubieran caído en desuso, encontrar los fallos en una aplicación en la que se había utilizado varios lenguajes, sin un criterio en la selección de lenguaje,

El proyecto Apollo que aterrizó en la luna también requirió de la ingeniería del software.

En los años 70 junto a compañías como Apple, Microsoft y Oracle, la ingeniería del software da un gran salto, las razones pueden ser el inicio del entorno multiusuario, la aparición de los virus y las empresas de ciberseguridad.

Comienza a aparecer los patrones dentro de la industria como existen en otras industrias más antiguas. Los modelos de desarrollo en cascada de mano de Winston Royce, el modelo por prototipos.

Al final de los años 70 la carrera de Ingeniería del software está vigente en la mayoría de las universidades.

En 1981 con la introducción de los ordenadores personales por parte de IBM, se introducen grandes avances en el desarrollo estructurado del software. En 1984, el Instituto de Ingeniería del Software, SEI, por Watts Humphrey. En inicio de internet desde Arpanet. El grupo internacional de puntos de función (IFPUG) que se trasladó desde Canadá a Estados Unidos en 1986. El famoso Instituto de gestión de proyectos (PMI) en 1984.

Entre 1980 y 1989 más del 50 % del desarrollo de software corresponde a tecnologías de la información, software embebido y sistemas.

La usabilidad debido a destinarlo al público sin experiencia pasa a ser requerida.

El modelo en espiral por parte de Boehr en 1986.

Los años 90 junto con el lenguaje HTML e internet surge un abanico de diversas empresas de software VMWARE, AVG Antivirus, Apache, Google, Ebay, Amazon.

Aunque las pruebas han formado parte la ingeniería del software desde el principio, se ha hecho más énfasis en los últimos años en la predicción de errores en todas las fases de software, en lo necesario de la documentación y en unas pautas para garantizar la calidad de cara al propio producto y al consumidor. Para ello se utilizan métricas y modelos de madurez en el desarrollo del software donde se garantiza la evaluación contra los percances. Muchas de estas métricas han sido de gran ayuda de cara a la gestión no sólo de software sino también de servicios, donde los tiempos de respuesta y los fallos, los tiempos de respuesta pasan a ser garantizados por las empresas que venden dichos servicios.

Muchos de esos estándares se han homogeneizado, compatibilizado, aclarado y versados a mayor número de desarrolladores, se han eliminado puntos vacíos.

Según ha ocurrido esto, han surgido herramientas de automatización y gestión y aseguramiento tanto de la calidad como de corrección de defectos, bajo criterios de aceptación.

(Jones, 2014)

2.1.3 Planificación como NP-Dificil

Aproximaciones algorítmicas

Los problemas de planificación determinista son parte de una extensa clase de problemas de optimización combinatoria. Puesto que, la aproximación general al

análisis de estos problemas puede seguir unas líneas parecidas, pero uno debería tener en cuenta sus peculiaridades, las cuales son particularmente evidentes en las aplicaciones de ordenador. Es evidente que estas aplicaciones el tiempo de resolución del problema estará muy limitado por lo que un tiempo polinómico de bajo orden será utilizado.

Sin perderme en la explicación matemática, introduzco una serie de pautas

Blazewicz en su artículo, clasifica los problemas de planificación basados en unos recursos dentro de la categoría NP-Difícil (de un procesador, dos procesadores y tres procesadores basándose en

Existen una gran cantidad de problemas de optimización combinatoria para los que en su mayoría no existe un algoritmo de optimización eficiente. Algunos problemas cuya decisión se contrapone (por ejemplo, cuya respuesta es sí o no) son NP-completos. Los problemas de optimización son llamados NP-Hard en este caso.

Para el análisis de este tipo de problemas se aconseja los siguientes pasos:

En primer lugar, se debe relajar las restricciones impuestas en el problema original y resolver un problema más flexible. Esa solución puede ser una buena aproximación a la solución del problema original, en el caso de los problemas de planificación esta flexibilización puede consistir en:

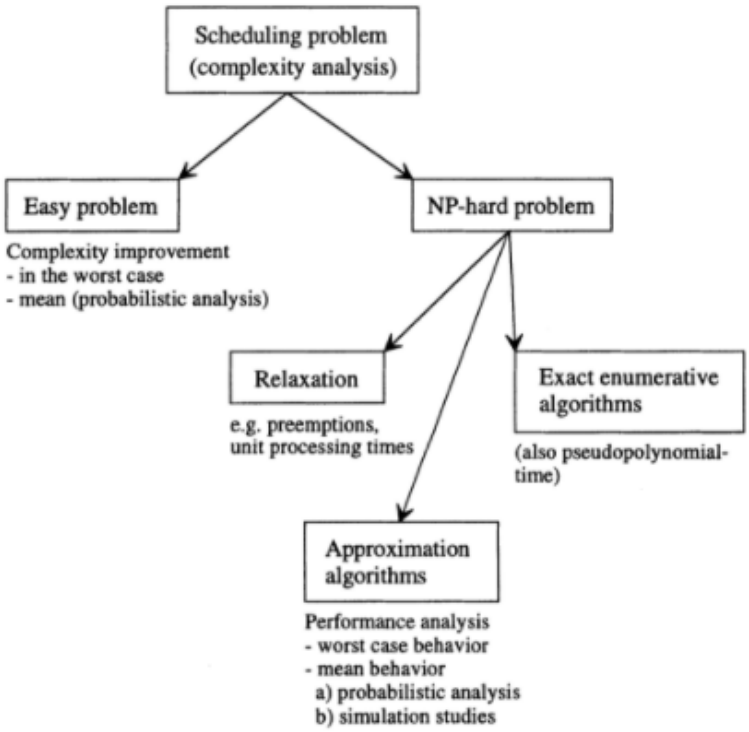


Ilustración 1Análisis de un problema de planificación (Błażewicz, Ecker, Pesch, Schmidt, & Węglarz, 2001)

- Permitir priorización, siempre que el problema original no contenga priorizaciones.
- Dar tiempos de finalización fijos, cuando las tareas tienen duración arbitraria en el problema original.
- Asumir ciertos tipos de grafos de precedencia, por ejemplo, árboles o cadenas, siendo en el problema original grafos arbitrarios.

(Błażewicz et al., 2001)

2.1.4 Planificación

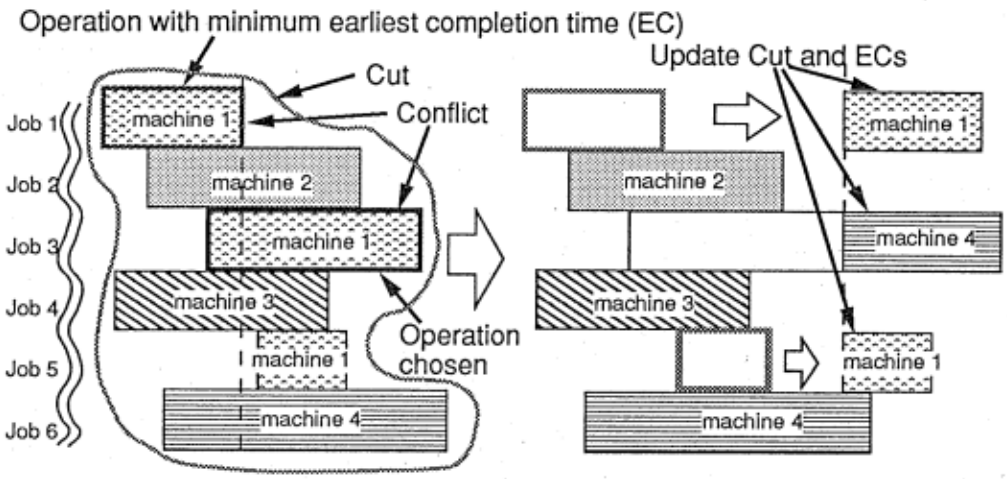


Ilustración 2 Algoritmo Clásico Griffer &Thompson.(Nakano & Yamada, 1992)

La gestión del plan de planificación es el proceso que establece unas políticas, procedimientos, y documentación para la planificación, desarrollo, gestión, ejecución y control de la planificación del Proyecto. Los beneficios clave de este proceso es que proporciona una guía y una dirección de como la planificación del Proyecto se gestionará a través del Proyecto.

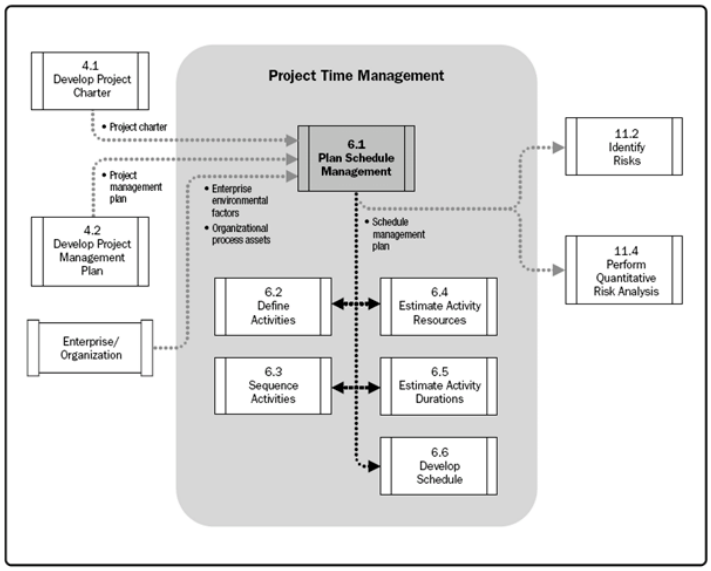


Ilustración 3Diagramam de flujo de datos de la gestión del plan de planificación (Project Management Institute, 2013)

El plan de gestión de la planificación es un componente del plan de gestión de proyectos. El plan de gestión de la planificación puede ser formal o informal, detallado de forma extensa o enmarcado de forma amplia, basado en las necesidades del Proyecto y se incluyen límites de control apropiados. La gestión de planificación define como los contingentes de planificación se informan y evalúan. La gestión de planificación puede ser actualizada para reflejar un cambio en el camino en que la planificación es gestionada. El plan de gestión de la planificación es la mayor entrada para el proceso del desarrollo del plan de gestión del Proyecto.

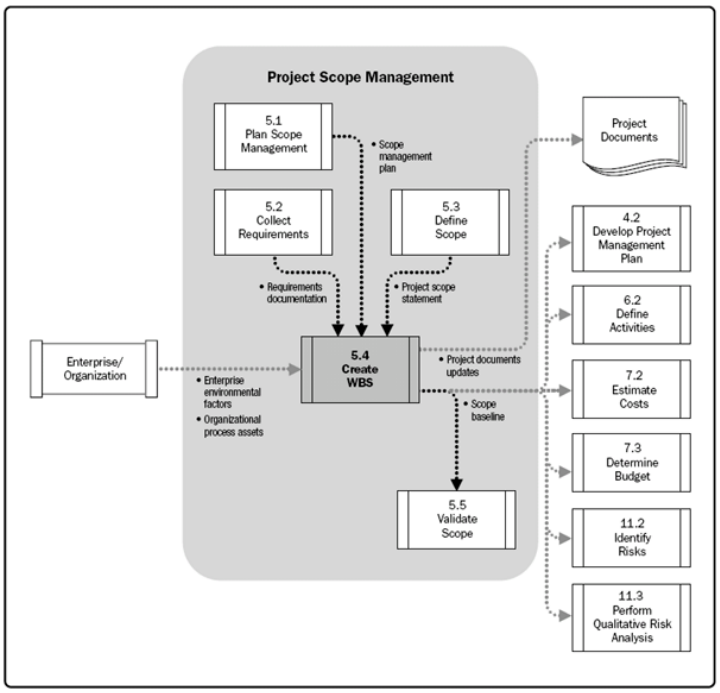


Ilustración 4Creación de un WBS (Project Management Institute, 2013)

El Proyecto deberá definir unos objetivos, de cara al producto o servicio. Y un objetivo para el propio Proyecto, que en muchas ocasiones incluye al propio objetivo de producto.

La gestión del tiempo de Proyecto, muy importante, incluiría la definición de actividades, la secuencia de actividades, la estimación de recursos de las actividades, estimación de la duración de las actividades, desarrollo de la planificación y el control de la planificación.

La gestión del coste, muy tratado por Boehm, sería la estimación de costos, determinar la cartera, el control de costos.

La gestión de calidad, realización del aseguramiento de la calidad, control de la calidad.

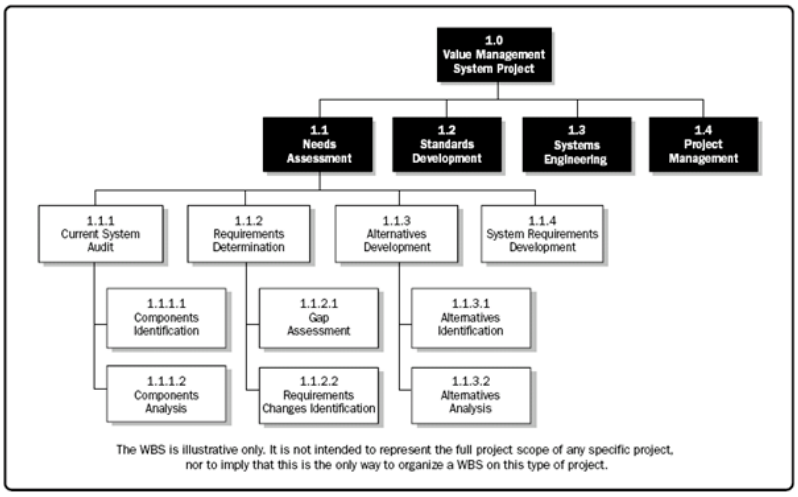


Ilustración 5 (Project Management Institute, 2013)

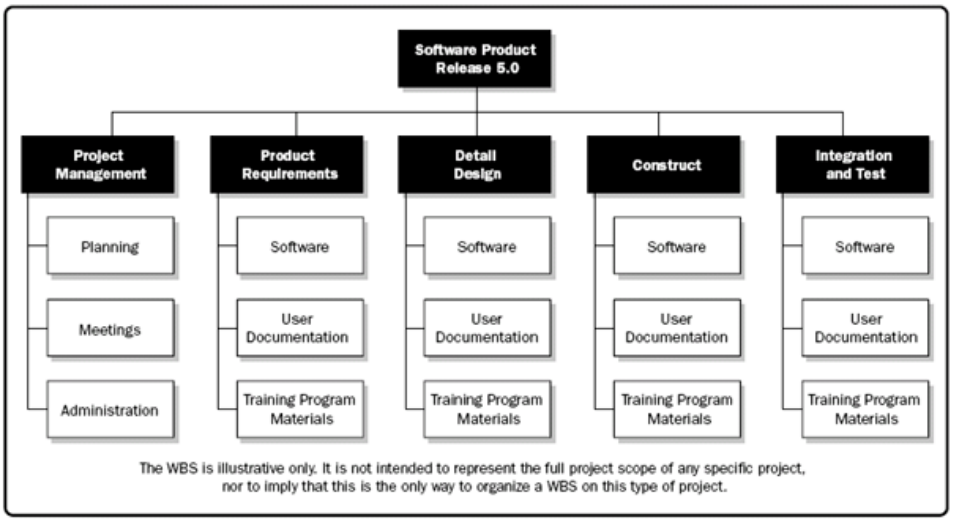


Ilustración 6 (Project Management Institute, 2013)

2.1.4.1 Estructura de descomposición del trabajo

Es una descomposición jerárquica del trabajo total objetivo que será llevado a cabo por el equipo del proyecto para cumplir con los objetivos de proyecto y crear los entregables necesarios. El EDT (WBS) organiza y define un total de objetivos del proyecto y representa un trabajo específico en la declaración de objetivos de proyecto que han sido aprobados en estos momentos.

2.1.4.2 Paquetes de trabajo

El plan de trabajo a más bajo nivel de los componentes EDT, se llamarán paquetes de trabajo. Un paquete de trabajo puede ser usado por un conjunto de actividades que son planificadas y estimadas, monitorizadas y controladas. En el contexto de WBS, el trabajo se refiere al trabajo de producto y entregables que resultan de las actividades que no son referidas la actividad en sí misma.

2.1.5 Planificación basada en unos recursos

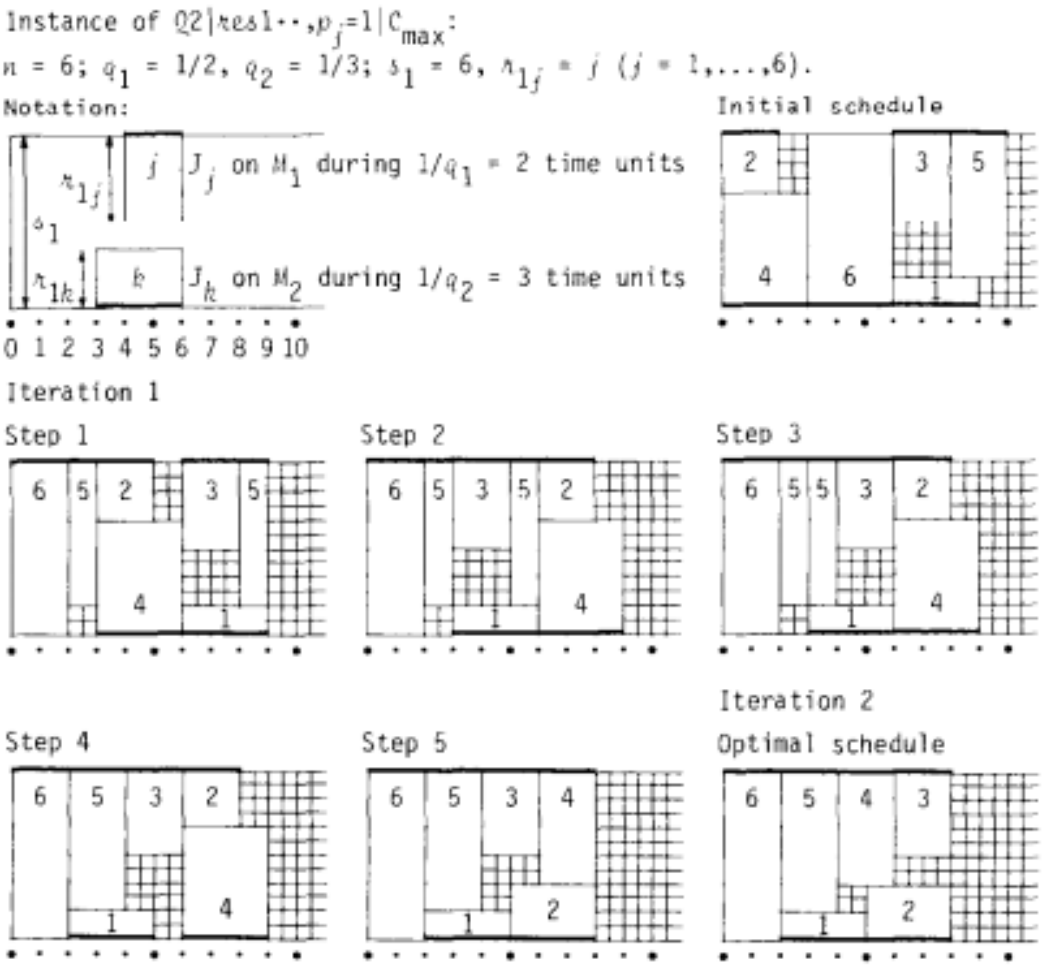


Ilustración 7Presentación de posibles soluciones (Blazewicz, Lenstra, & Kan, 1983)

La planificación de los proyectos consiste en la forma en que la organización lleva a cabo sus actividades consumiendo tanto tiempo como posiblemente otros recursos. Dado que estas actividades tienen varias formas de tener lugar, esto no plantea ningún problema. Existen muchas herramientas que permiten dentro de la categoría de planificación que ayudan a establecer su ejecución de forma paralela, ya que muchas de estas actividades no son triviales y se necesitan herramientas eficientes.

Históricamente este tipo de problema data de 1950. Cientos de actividades interrelacionadas y utilizando una gran cantidad de recursos. El conocido como Método del camino crítico CPM, de las corporaciones Du Pont de Nemours y Remington Rand Univac.

Una técnica similar de evaluación de proyectos y técnicas de revisión o PERT, que fue desarrollado por la US Navy (1959) para los sistemas de armas Polaris.

Red de representación

Actividades, Asociado a un macro nivel con proyecto. A un nivel medio se diferenciarían distintas tareas, y a un micro nivel del proyecto las tareas se subdividirían en actividades. Por lo que se pueden considerar átomos de un proyecto.

Duración de las actividades, Cada actividad se refiere a que tiene una cierta duración. Dado que esta actividad es complicada corresponde a alguien que toma las decisiones plantearse el ¿qué pasa sí? Que proporciona ciertos retrasos mientras se conserva la simplicidad de un modelo determinista.

Relaciones de precedencia, el que toma las decisiones ha aplicado su criterio para determinar el orden en que las actividades se deben llevar hasta buen término. Se tiene el axioma de transitividad.

(Eiselt & Sandblom, 2004)

2.1.6 Ciclo de vida: Pruebas

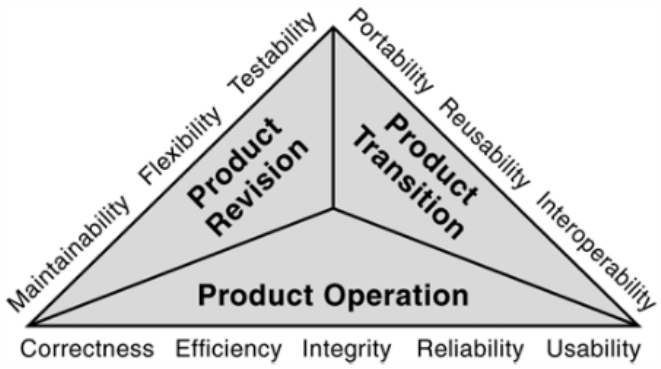


Ilustración 8 Restricción tripe en el Proyecto (Futrell, Shafer, & Shafer, 2002)

Los planes de proyecto son un elemento indispensable en el éxito de cualquier proyecto. Para proyectos complicados y laboriosos, los consumidores podrían requerir un plan donde se formulara todo el itinerario. Dicho plan puede servir como una guía durante el tiempo de vida del proyecto, que suele ser revisado al menos una vez al mes, dependiendo de las circunstancias y el tipo de proyecto.

	Flexibility		
	Least	Moderate	Most
Scope		X	
Schedule	X		
Cost			X

Ilustración 9Matrix de flexibilidad (Futrell et al., 2002)

De cara al modelo de pruebas, se analizarán varias metodologías de software, y su mécanica de pruebas.

(Futrell et al., 2002)

En cualquier caso, como apunta Boehr la calidad no debe mermar ni en coste ni en planificación, puesto que de forma clara resultará un producto software mermado en sus aseguramiento de calidad, tanto si se ampliaron los plazos por eliminación de controles de calidad como si se redujo la inversión en el aseguramiento de la calidad.

2.1.6.1 Modelo en V e Iterativo

Los modelos de desarrollo han tenido una enorme evolución desde los años ochenta con el modelo en cascada, donde cada tarea debía concluirse antes de iniciar la posterior y el modelo en espiral de Boehm, similar al desarrollo en cascada, pero retornando a la actividad primera al finalizar la última, este es el inicio de lo que paso a explicar a continuación, donde las pruebas y desarrollo de alguna forma se retroalimentan.

Paso a hablar de dos modelos que han sido incluidos por la Fundación Syllabus, por la relación entre desarrollo y pruebas.

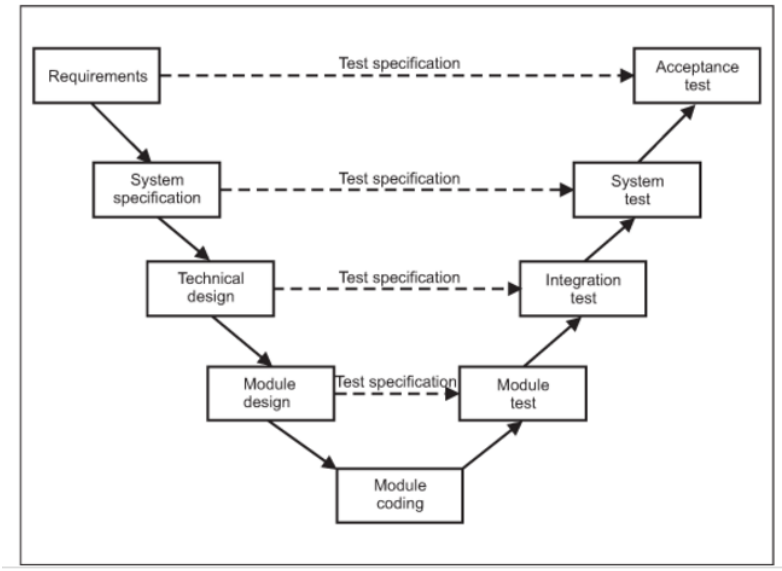


Ilustración 10Modelo en V (Hambling, Samaroo, & British Computer Society, 2009)

Este es un modelo idealista en muchas cosas y se da pocas veces en la práctica. Proporciona una forma clara y un camino simple de describir las relaciones entre pruebas y desarrollo, como se muestra en cada actividad de desarrollo que tiene asociada una actividad de pruebas y para demostrar que pruebas pueden y deberían ser especificadas al mismo tiempo que los entregables del desarrollo son creadas. ¿Por qué? Porque cada actividad cuenta con pruebas específicas lo que es una excelente forma de probar la calidad de los entregables de desarrollo.

Los problemas que generen las prueban tendrán una repercusión similar a nivel de señal en el próximo paso de desarrollo quedando dichos problemas bien ubicados en el tiempo de forma temprana. Para una implementación completa de este modelo se debería revisar cada etapa y las especificaciones de pruebas creadas antes de que la implementación de los módulos se inicie.

En la práctica cada modelo tiene algunos problemas fundamentales, el más importante de estos que el modelo entero depende de la calidad y la completitud de los requisitos, desde el punto de vista que pruebas y desarrollo producen un único

entregable. Los requisitos son muy difíciles de definir y generalmente no permanecen fijos a través de todo el proyecto software, por eso este problema tiene sus dificultades en la práctica.

(Hambling et al., 2009)

Si bien es cierto que este tipo de desarrollo puede no haberse llevado a cabo, es muy interesante como comentaba en la sección de NP-Hard, a modo de simplificación, es decir aquellos cambios en los requisitos, requiriendo ser urgentes, pueden ser tratados como la misma etapa en que se descubrieron, puesto que los errores de etapas posteriores, se supondrá que serán demorados por la función objetivo, al detectar una diferente fase de prueba.

Vuelta al análisis de modelos de desarrollo, no se va a seguir este modelo, pero podría hacerse, parando la ejecución de las pruebas para "proseguir" con el desarrollo, porque se supone que es un producto ya "finalizado".

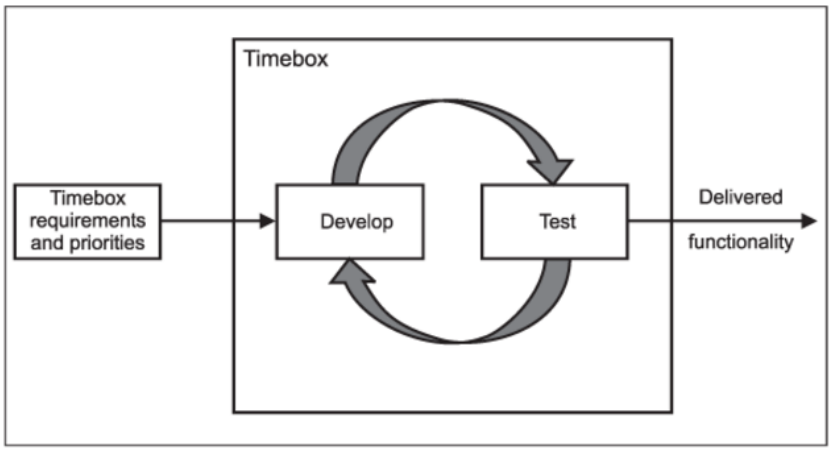


Ilustración 11Actividades de pruebas en el modelo iterativo

El modelo iterativo toma la idea que los requisitos no suelen ser estables y elimina como es lógico pensar la necesidad de establecer requisitos estables para

comenzar el proyecto. En lugar de ello, se identifican una serie de objetivos para el proyecto y se priorizan. El desarrollo está dividido en pequeñas cajas de series de tiempos cortos, durante los cuales los requisitos son desarrollados y probados, así que los progresos del proyecto como series de ciclos cortos de desarrollo, cada uno enfocado a la prioridad más alta de los requisitos todavía no desarrollados.

Desarrollo rápido de aplicaciones (RAD), prototipado, Ágil, metodología de desarrollo de sistemas dinámicos (DSDM)

(Hambling et al., 2009)

2.1.6.2 Métodos de estimación

A la hora de responder como va un proyecto se debe recurrir a unos parámetros de estimación, en las pruebas el esfuerzo requerido dependerá:

1. El esfuerzo del tamaño de desarrollo (por ejemplo, tiempo que ha llevado, número de líneas de código, complejidad del código que debe ser probado).
2. La cantidad de código reusado de proyectos previos. Requiere menos pruebas.
3. El rigor para la salida del producto.
4. Uso de herramientas automáticas.
5. Nivel de experiencia de los encargados de las pruebas
6. El entorno de pruebas.

Los métodos de estimación:

1. Basados en la intuición.
2. Grupo de consenso.
3. Basado en la métrica.
4. Estructura detallada de marco de trabajo.
5. Porcentaje de esfuerzo de desarrollo.
6. Análisis de puntos de pruebas.

(Hambling et al., 2009)

2.1.6.3 Idea Errónea sobre las pruebas

Una de las principales causas de las escasas pruebas de aplicaciones es que la mayoría de los programadores, comienzan con una idea equivocada en la definición de los términos Ellos suelen decir:

"Las pruebas es el proceso de demostrar que los errores no están presentes"

"El propósito de las pruebas es mostrar que un programa realiza aquellas funciones correctamente"

"La prueba es un proceso de establecer confianza con lo que un programa se supone que hace"

Cuando se prueba un programa se añade valor sobre él. Añadir valor a través de las pruebas significa elevar la calidad y la fiabilidad del programa. Elevar la fiabilidad de un programa significa encontrar y eliminar los errores.

(Myers, Sandler, & Badgett, 2012)

2.1.6.4 Reducción de defectos

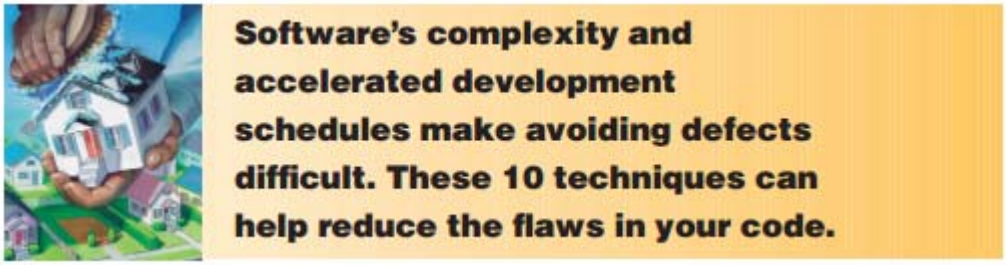


Ilustración 12Acerca de la generación de errores (Boehm & Basili, 2005)

Sobre la reducción de defectos en un artículo Boehm y Basili proporcionan diez consejos acerca como evitarlos.

1. Encontrar y corregir un problema de software después de la entrega suele ser 100 veces más caro que encontrarlo y arreglarlo durante la fase de requisitos y de diseño.
2. Los proyectos actuales dedican cerca del 40 al 50 por ciento de su esfuerzo a trabajo que se podía haber evitado
3. Cerca del 80 por ciento de trabajo evitable viene de un 20 por ciento de defectos
4. Cerca del 80 por ciento de los defectos viene de un 20 por ciento de los módulos y cerca que la mitad de los módulos están libres de defectos.
5. Cerca del 90 por ciento de bajar el tiempo proporciona cerca del 10 por ciento de los errores.
6. Las revisiones detectan el 60 por ciento de los defectos.
7. Las revisiones basadas en la perspectiva detectan 35 por ciento más de defectos que las revisiones no directas.
8. Las prácticas disciplinadas de personal pueden reducir la tasa de introducción de defectos hasta un 75 por ciento.
9. Todas las demás cosas son iguales, pero la introducción de instrucciones de código para desarrollo de productos software con alta dependencia cuesta el 50 por ciento más que el desarrollo de producto software con baja dependencia. En cualquier caso, la inversión será más rentable si el proyecto se encarga de operaciones significativas y costes de mantenimiento.
10. Cerca del 40 hasta 50 por ciento de los programas de usuario contiene defectos no triviales.

(Boehm & Basili, 2005)

En general existen tres estrategias

En un acercamiento a las estrategias de pruebas e inspección integrada, se usarán métricas de supervisión destinadas a la actividad de las pruebas. De forma que se garantice que los datos de verificación son válidos y monitorización de la calidad es desarrollada antes de las pruebas basándose en dichos datos.

Si la calidad de los datos de defectos de inspección son los suficientes, las diferentes métricas pueden ser aplicadas, por ejemplo, los defectos existentes (p.ej. número absoluto de defectos de inspección) o densidad de defectos (p.ej. número de defectos de inspección por unidad, p.ej. líneas de código).

Para poder priorizar es necesario saber la relación entre inspección y pruebas. De forma que se puede asumir que en la fase de pruebas se esperan encontrar más errores en aquellos módulos dónde la inspección los encontró. De forma que se podrá enfocar la tarea de pruebas sobre el código de las clases con mayor número de defectos descubiertos o mayor densidad de defectos. Así la priorización del código de las clases será cubierta por reglas de selección que se suponen servicios.

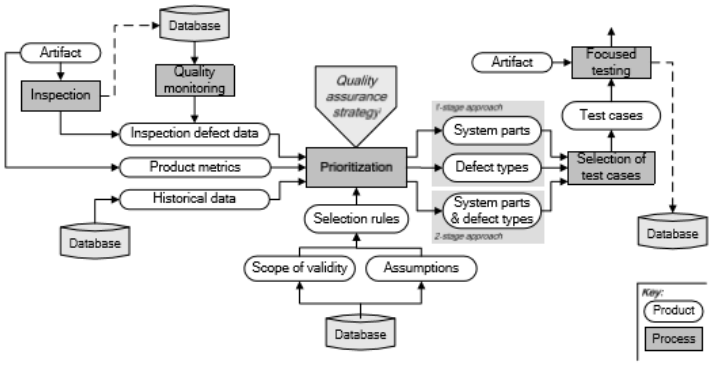


Ilustración 13Aproximación a las pruebas

Si cierto código de las clases se prioriza, los casos de prueba tienen que ser seleccionados o desarrollados, bajo una actividad enfocada a las pruebas que debe ser

guiada. Según la estrategia de QA, más o menos código de las clases podría seleccionarse para la actividad de pruebas. Por ejemplo, si el objetivo es reservar esfuerzo, sólo las clases de mayor prioridad serán seleccionadas. Si barajamos la opción de encontrar más defectos (p.ej. si la efectividad debe mejorarse), un mayor número de clases se seleccionará. Platónicamente, ambas estrategias tienen beneficios (p.ej. desarrollo de la eficiencia).

Además, las métricas de producto junto con datos históricos se pueden unir para métricas de inspección para el desarrollo de la prioridad. Cabe destacar que se debe reconsiderar si es válido en el número contexto puesto que muchas reglas de selección están ligadas a un contexto con un objetivo.

En conclusión, para priorizar las partes de un sistema, se debe priorizar según el tupo de defectos.

Subyace la idea para las diferentes actividades de QA contribuyen a una única actividad, con tres claros objetivos, reducción de esfuerzo, desarrollo de la efectividad y desarrollo de la eficiencia.

(Elberzhager & Münch, 2013)

2.1.7 Modelo de Pruebas

Not everything that counts can be counted and not everything that can be counted counts.

No todo lo que cuenta puede ser contado y no todo lo que puede ser contado cuenta

Albert Einstein

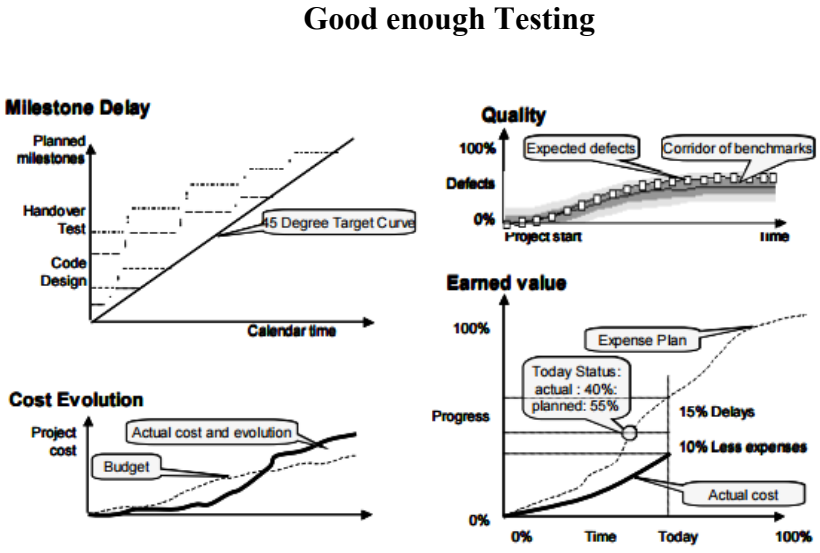


Ilustración 14Cuadro de mando de métricas, métricas sobre la planificación, coste, calidad y contenido o valor ganado

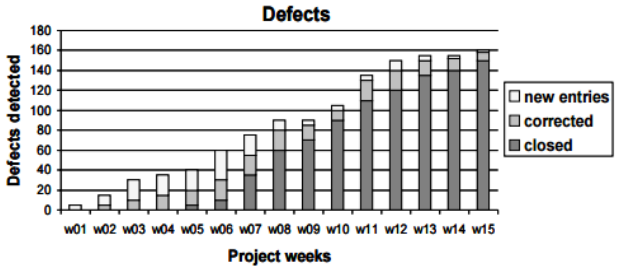


Ilustración 15Cuadro de mando de métricas, métricas de trabajo sobre entrega y calidad para el proyecto a través de tiempo

Para realizar una monitorización efectiva en el proceso y la implementación de acciones correctivas se requiere una organización fuerte. Las decisiones en muchos casos son contradictorias por la intromisión de los jefes de departamento y los gerentes. La prioridad de las tareas y la frecuencia en las tareas suele ser lo más afectado, y se modifican. La desmotivación y la ineficiencia suelen ser los resultados en dicha situación.

El trabajo de los jefes de proyecto debe estar basado en roles con responsabilidades definidas. Se requiere que desarrollen planes y objetivos. Se debe fijar en el proyecto la medición de la eficiencia y como objetivo la calidad, con un plan de calidad del proyecto más tarde se podrá monitorizar durante todo el ciclo de vida del software. Los jefes de proyecto informan sobre el proceso del proyecto y la calidad usando mecanismos estándar de informes. Como se puede ver en las imágenes.

(Ebert, 2005)

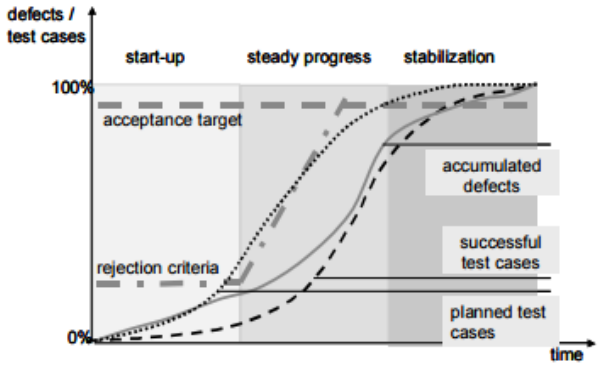


Ilustración 16 Típico ejercicio de control de proyecto

Para la identificación de los módulos propensos al error, se debe aplicar el teorema de Pareto. Un análisis de proyectos actuales revela que la aplicación de la regla de Pareto: 20% de los módulos son responsables del 80% de los funcionamientos no pretendidos, mal funcionamientos de todo el proyecto. Dichos módulos se deben

identificar lo antes posible, por ejemplo, después de la implementación. Para concentrar sobre dichos componentes la efectividad de la inspección de código e incrementar el número de pruebas sobre ellos y obtener menos errores durante la fase de pruebas.

Esto añade a todas las fases del desarrollo de software un añadido de inspección de código y pruebas de unidad.

El organismo que se encarga de la integración y el desarrollo es ISO 9001. Se compone una comunidad de desarrollo, pruebas, integración, y entregas que supera los 500 Ingenieros y ofrece distintas entregas a diferentes tipos de cliente y cada año.

(Ebert, 2005)

2.1.7.1 Modelos basados en la complejidad.

Las técnicas estadísticas multivariante proporcionan retroalimentación en cuanto,

Proveer retroalimentación acerca, la relación entre componentes (p. ej. Análisis de factor, análisis del componente principal), técnicas de clasificación para determinar las equivocaciones (p. ej. Componentes propensos al error). Finalmente, los diagramas detallados y las tablas proporcionan las razones por las que los distintos componentes tienen a tener un comportamiento equivocado y como mejorarlo. Una forma de predicción es el análisis de la complejidad temprana tomando la cuenta con criterios severos.

Un criterio es el análisis multifacética tomando la cuenta sobre un criterio grave. El único criterio de análisis de la complejidad de un módulo en la fase temprana.

Otro criterio atañe a la cantidad de código nuevo o modificado en un módulo, y la cantidad de faltas que el módulo tuvo en un proyecto anterior.

Todos estos criterios son usados para construer una predicción completa. Basado en el ranking de criticidad de todos los módulos que son usados para construir, diferentes mecanismos son aplicados para el desarrollo de la calidad, llamada rediseño, inspección del Código, o pruebas de módulos con alta cobertura. WC será en esta parte enfocado sobre la predicción de la criticidad basado en la complejidad y entonces mejorar el modelo.

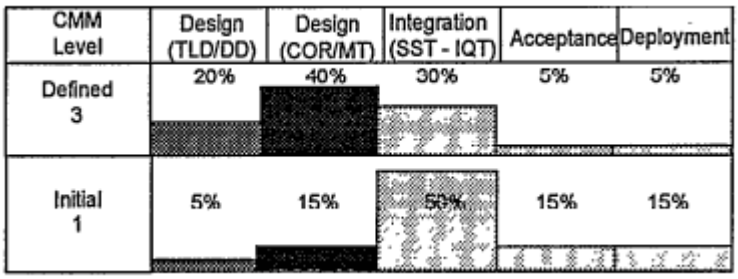


Ilustración 17 Típicos benchmark efectos de la detección temprana de defectos en el ciclo de vida (Ebert, 1997)

2.1.7.2 Metodologías de clasificación

- Clasificación de Pareto
- Clasificación mediante árboles
- Análisis discriminante basado en el factor
- Clasificación de Fuzzy.
- Aproximaciones de redes neuronales

El uso de los modelos predictivos de calidad basados en la complejidad puede compensar en las siguientes situaciones:

1. Recursos limitados que son asignados a tareas de alto riesgo o componentes.
2. Análisis del impacto y la evaluación del riesgo de cambios que son posibles según como afecten o cambien la complejidad.
3. Estrategias de caja gris que son aplicadas para identificar componentes de alto riesgo.
4. Cuando recibamos menos fallos del cliente.

(Ebert, 1997)

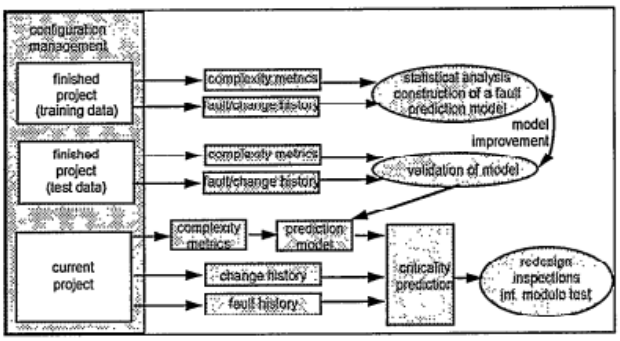


Ilustración 18Aproximación a predicción de criticidad (Ebert, 1997)

Dado que las pruebas se realizan desde el análisis matemático, desde unos casos que cubran la mayor parte de datos que el programa pueda requerir y caminos de ejecución que pueda seguir. Se requieren unas estrategias para afrontar las pruebas, y se crean dominios y subdominios dentro del programa para su análisis. Esto sugiere la posibilidad de paralelismo.

2.2 Algoritmos Evolutivos

2.2.1 Introducción

Los algoritmos evolutivos son una opción que permite un acercamiento a soluciones a problemas NP-Complejo o NP-difícil como es el caso de la planificación. Es por ello, la opción que permite acercarse a la solución óptima sin dejar de ser eficiente por óptimos locales. El hecho que se introduzca la aleatoriedad y una función de adaptación lo suficientemente buena, permite a la población de individuos que representa el problema ir evolucionando a una prole más efectiva en la resolución del problema.

Es muy interesante el hecho que se pueda emplear también los algoritmos genéticos como un control de la calidad, permitiendo mantener en los individuos ciertas características que se consideren esenciales a nivel de cliente y de estrategia comercial, cosa que muchos algoritmos pierden en su resolución y debe añadirse de alguna forma ya resuelto el problema, además de no violar ciertas condiciones para el producto.

Son por ello una herramienta muy bien considerada tanto en la planificación como en el control de la calidad.

2.2.2 Precursores

2.2.3 La planificación cómo proceso software

La planificación es un proceso de optimización para unos recursos limitados que son distribuidos durante el tiempo entre las actividades que se ejecutan en paralelo y en secuencia. Como las situaciones rutinarias en que se producen fábricas, editoriales, envíos, universidades, hospitales, aeropuertos etc. Para resolverlos como problema se debe realizar elecciones discretas para encontrar una solución óptima entre muchas o un número definido de infinitas alternativas.

En general, la tarea es compleja, lo que limita la utilidad práctica de la combinatoria, la programación matemática y otros métodos analíticos para resolver problemas de planificación de forma efectiva.

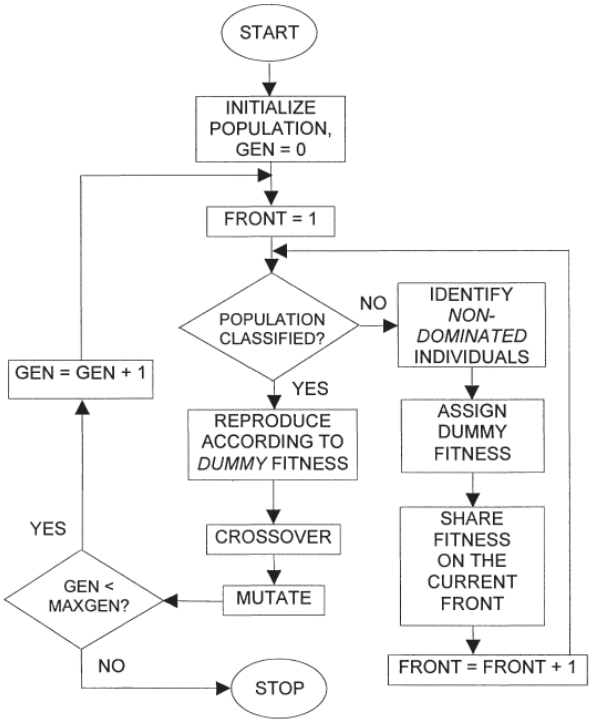


Ilustración 19 Ordenación GA no dominante (Bagchi, 1999)

Para encontrar soluciones exactas a los problemas se suele aplicar algoritmos de ramificación y poda, y algoritmos programación dinámica. El uso de información específica del problema en ocasiones reduce el espacio de búsqueda, aunque el problema suele resultar todavía difícil de resolver.

Algunos métodos de búsqueda local se inspiran en procesos de física estadística, evolución biológica y más recientemente en la neuropsicología. Para problemas de un único objetivo complejo, la efectividad de la inteligencia artificial basada en el respeto de las restricciones ha sido demostrada.

Las soluciones efectivas para problemas multi objetivo continúan siendo un enigma para planificadores de negocio, administradores de flotas, gerentes de tiendas y demás.

En los problemas de decisión multi objetivo se desea optimizar la eficiencia de más de un objetivo, como puede ser el tiempo total de terminación, la demora, el tiempo de ejecución de las tareas… La forma más sencilla de afrontarlo es combinar varios objetivos en un único objetivo a través de pesos que los definan y entonces optimizarlos.

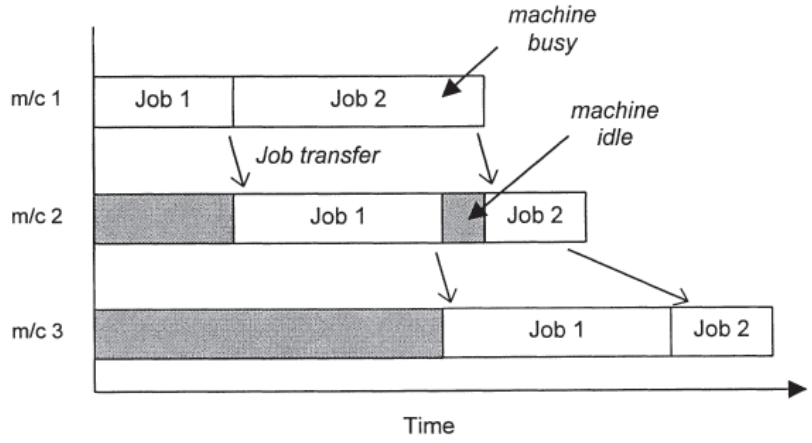


Ilustración 20The Gantt Chart (Bagchi, 1999)

Los principales inconvenientes a esto son

1. Se debe tomar una decisión previa de cuáles serán los pesos antes de combinar los objetivos para optimizarlos
2. La mejor alternativa es siempre probar los distintos pesos para los métodos solución
3. Qué sea produzca una suma de pesos no garantiza que la solución final este dominada por el sentido de optimalidad de Pareto, una clave para aceptar soluciones multiobjetivo es que sean no dominadas, cada solución es la mejor al mejor para una de las decisiones objetivo.

Los algoritmos genéticos tienen un método de búsqueda muy apropiado que se conoce como búsqueda de nicho que se asocia perfectamente a la noción de recurso y a la de función de aptitud compartida.

Además, suelen encontrar la mejor solución entre un número muy grande de soluciones de igual forma que imita la evolución natural.

Las soluciones por tanto son óptimas en Pareto o eficientes, un conjunto de soluciones no dominantes.

(Bagchi, 1999)

2.2.3.1 Planificación de proyectos con recursos limitados

El problema de la planificación clásica de proyecto con recursos limitados (RCPSP), consiste en que las actividades del proyecto deben ser ejecutadas de acuerdo con la minimización de su tiempo de finalización. Este problema afecta tanto a los proyectos de gestión del software igual que a los de planificación de la producción y ejecución.

La descripción de estos problemas suele consistir en un proyecto que consiste en J actividades etiquetadas j=1,….,J En el que los requisitos tecnológicos contienen relaciones de precedencia entre algunas de las tareas.

Estas relaciones de precedencia tienen dadas un conjunto de predecesores inmediatos P_j

Que indican cuando una actividad j no puede ser empezada antes de todas las anteriores predecesoras hayan finalizado, De forma similar, S_j es un conjunto de sucesores inmediatos de la actividad. La relación de cierre transitivo se da ante un conjunto (no necesariamente inmediatos) de sucesores S_j.

(a) permutation	2	4	1	6	3	5
(b) priority value	0.58	0.64	0.31	0.87	0.09	0.34
(c) priority rule	LST	GRPW	MTS	LST	MSLK	LFT

Ilustración 21Ejemplo de individuos(onke Hartmann & onke Hartmann, 1997)

Las relaciones de precedencia pueden ser representadas por una actividad en cada nodo de la red que se asumirá como acíclico. Esto puede considerarse como una actividad añadida j=0 que representa un único recurso y j=j+1 que representa una única pila por actividad en la red. Con la excepción de un recurso trivial y de una pila trivial de actividad, cada actividad requiere cierta cantidad de recursos que se renuevan para ser ejecutadas. Este conjunto de recursos suele ser referenciado como K. Para cada recurso k perteneciente a K, el periodo de disponibilidad es constante y dado por R_k. El tiempo de procesamiento (duración) de una actividad j es referenciado como una constante y que proviene de R_k.

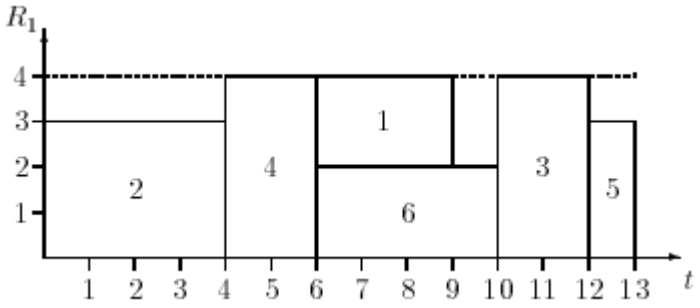


Ilustración 22Ejemplo de planificación (onke Hartmann & onke Hartmann, 1997)

Una vez comenzada, una actividad no debe ser interrumpida. Se considera que las actividades triviales tienen duración cero y no compiten por ningún recurso.

(onke Hartmann & onke Hartmann, 1997)

Voy a exponer las reglas que presenta en su artículo Sprecher, Hartmann y Drexl en cuanto a un nuevo algoritmo de ramificación y poda para la minimización del tiempo de finalización del proyecto, de un problema de recursos limitados multi modo. Establece un sistema tanto estático como dinámico de búsqueda a través del árbol.

En otro artículo establece cuatro reglas para la resolución de este tipo de problemas que constituyen el método del camino crítico, CPM y método de potencial metra (MPM), las reglas se establecen como

Regla de asociación 1: Regla de ventana de tiempo. Se denota como T a la mayor asociación sobre el tiempo de finalización del proyecto. Si hay una actividad a ejecutar j el tiempo asignado de finalización que exceda el tiempo mayor de finalización $LF_{j,}$ entonces la ejecución parcial actual no puede ser finalizada con el tiempo de finalización menor o igual a T.

$$LFj := LF\ 5 - (LFj - T+ 1) \text{ para } j = 1,..., J.$$

Regla de asociación 2: Reducción de información. Los datos del proyecto pueden ser ajustados para su uso de la forma siguiente:

Paso 1: Eliminar todos los modos no ejecutables de la información del proyecto.

Paso 2: Eliminar los recursos redundantes no renovables.

Paso 3: Eliminar todos los modos ineficientes.

Paso 4: Si algún modo ha sido borrado en el Paso 3, se va al paso 2.

Regla de asociación 3: Consumo de los recursos no renovables: Se asigna SJ_g a $\bar{S}J_g$ que se denota como un conjunto de actual de selección para ejecución y el conjunto de actividades no seleccionadas para ejecución. Respectivamente. Si hay recursos no renovables r pertenecientes a N con:

$$\sum_{j \in SJ_g} k^{\nu}_{jm_{jr}} + \sum_{j \in \overline{SJ}_g} kmin^{\nu}_{jr} > K^{\nu}_{r},$$

Entonces el problema no podrá ser completado.

Regla de asociación 4: Regla de multi modo. Se asigna j para ser la actividad que se concluye en el actual punto de decisión en el modo m_j. Si un multi modo se desplaza a la izquierda o un modo de reducción de j con el modo resultante m'_j, $1 \leq m'_j \leq M_j$, puede ser ejecutado de la siguiente forma parcial, y además si $k^{\nu}_{j\, mj\, r} \leq k^{\nu}_{j\, m'j\, r}$ mantiene para cada recurso no renovable r Є N, entonces la ejecución parcial actual no necesita ser completada.

j	m	d_{jm}	k^{ρ}_{jm1}	R	K^{ρ}_{1}	N	$\pi[3,1,j,m]$
1	1	0	0	{1}	4	∅	0
2	1	2	1				0
3	1	5	3				0
4	1	4	2				0
5	1	2	2				0
	2	3	1				1
6	1	3	2				0
	2	4	1				0
7	1	1	3				0
	2	2	2				0
8	1	0	0				0

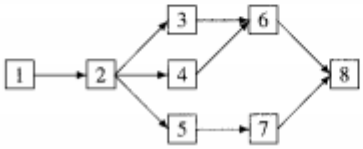


Ilustración 23 Red de trabajo (Sprecher, Hartmann, & Drexl, 1997)

Regla de asociación 5: Regla de desplazamiento local a la izquierda. Se asigna DA_g y DA_{g-1} como la selección de demora alternativa sobre el actual y nivel previo del árbol de ramificación y poda, respectivamente. Si existe una actividad j Є DA_g / DA_{g-1} que puede ser desplazada a la izquierda sin cambiar su modo. Entonces la planificación parcial actual no tiene porque ser desplazada.

Regla de Asociación 6: Selección inmediata – único trabajo. Asignar RDA_g, para ser de nuevo ajustado como conjunto de alternativas con demora mínima al nivel g del

árbol de ramificación y poda, eso es, ninguna alternativa de demora ha sido seleccionada o examinada todavía. Asigno $\overline{SJ}_{g \text{ a ser un conjunto de}}$ tareas no ejecutadas. De la misma forma, asignar JIP_g a ser el conjunto de actividades en proceso en el punto de la actual desición t_g. Si todas las actividades en el proceso en el tiempo t_g empiezan en el tiempo t_g,y, más todavía, existe una actividad $j \in J\ I\ P_g$ como

(a) $\Pi [\ i,m_i,j,m_j\ ] = 0$ para todas las actividades $i \in J\ I\ P_g \setminus \{\ j\ \}$,

(b) $\Pi [\ i,m_i,j,m_j\ ] = 0$ para todas las actividades $i \in \overline{SJ}_g$ y para todos los modos $m'_i,\ 1 \leq m'_i \leq M_i$

Regla de Asociación 7: Selección inmediata – dos trabajos. Asignar RDA_g, para ser de nuevo ajustado como conjunto de alternativas con demora mínima al nivel g del árbol de ramificación y poda, eso es, ninguna alternativa de demora ha sido seleccionada o examinada todavía. Asigno $\overline{SJ}_{g \text{ a ser un conjunto de}}$ tareas no ejecutadas. De la misma forma, asignar JIP_g a ser el conjunto de actividades en proceso en el punto de la actual desición t_g. Si todas las actividades en el proceso en el tiempo t_g empiezan en el tiempo t_g,y, más todavía, existe una actividad $j \in J\ I\ P_g$ como

(a) $\Pi [\ i,m_h,j,m_j\ ] = 1$ para todas las actividades $i \in J\ I\ P_g \setminus \{\ j\ \}$,

(b) $\Pi [\ i,m_i,j,m_j\ ] = 0$ para todas las actividades $i \in \overline{SJ}_g$ y para todos los modos $m'_i,\ 1 \leq m'_i \leq M_i$

(Sprecher et al., 1997)

2.2.3.2 Planificación Job Shop

Los problemas de planificación job shop consisten en un amplio sentido, en la asignación a través del tiempo, de recursos de capacidad reducida a operaciones mientras cumplen ciertas restricciones.

Las principales restricciones serían:

- Restricciones de capacidad
- Restricciones de precedencia
- Restricciones temporales
- Restricciones de preasignación
- Planificación de múltiples flujos.

(Falkenauer & Bouffouix, 1991)

2.2.4 Representación de individuos

Dado que el tipo de problemas requiere codificaciones más complejas, la codificación binaria deja de ser útil suponiendo una merma en la eficiencia. Al igual que los problemas de viajero que tiene que pasar por todas las ciudades (TSP), parece que se requiere de una permutación para modelar la forma en que se ejecutaran las tareas. Esto entraña otras dificultades como es la de las soluciones no reales o que no se pueden cumplir, algo así como imaginarias, pero eso ya será resuelto en los operadores genéticos.

A pesar de ello comienzo con una versión binaria de representación, por ser a lo que se está acostumbrado.

Se puede considerar que existen dos aproximaciones a la representación una directa y otra indirecta, la primera no requerirá un proceso de decodificación, pero se debe garantizar que los individuos son posibles.

Sobre la representación directa, cuando se emprende la tarea del desarrollo de operadores complicados, este esquema toma la ventaja de ser muy directo.

Y respecto a la representación indirecta necesitará un programa que interprete la forma de traducir a planificaciones válidas. La ventaja de este esquema será la simplicidad de los individuos tanto estructural como a nivel de operadores, y el inconveniente es que el algoritmo genético está restringido a la búsqueda en el espacio de todas las posibles permutaciones de los genes individuales.

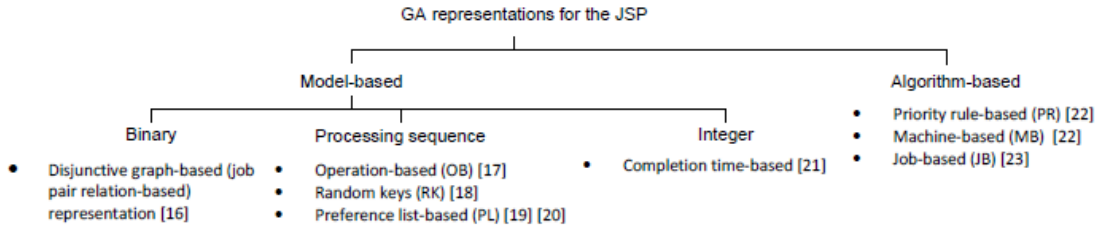


Ilustración 24 Tipos de representaciones para GA (Cheng, Gen, & Tsujimura, 1999, p. 3)

La representación en este tipo de problemas es un reto. Esta clase de problemas se caracteriza por variables discretas de decisión que suelen estar relacionadas a través de relaciones lógicas. Como resultado, se obtienen diferentes modelos matemáticos para el mismo espacio combinatorio de un problema, con la posibilidad de utilizar diferentes representaciones de algoritmos genético

(Cheng et al., 1999, p. 1)

2.2.4.1 Representación binaria

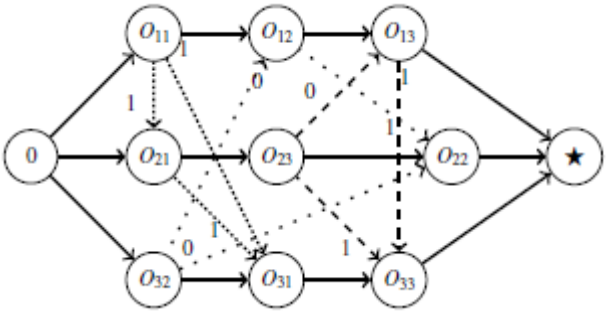


Ilustración 25Grafo disjunto etiquetado (Uyar, 2013)

Existen dos formas de representación la simbólica y la binaria. Los cruzamientos y la mutación no pueden aplicarse a la forma simbólica.

En la forma binaria utilizada se tienen de forma ordenada dos tuplas con los trabajos (j,k) por máquina. Sabemos cuál tienen prioridad porque tendrá o valor 1 o valor 0.

Por lo que se consigue un vector de pares de trabajo [j,k]

(Nakano & Yamada, 1991)

1	1	1	1	0	0	0	1	1
O_{11} O_{21}	O_{11} O_{31}	O_{21} O_{31}	O_{12} O_{22}	O_{12} O_{32}	O_{22} O_{32}	O_{13} O_{23}	O_{13} O_{32}	O_{23} O_{33}

Ilustración 26Representación binaria del grafo disjunto (Uyar, 2013)

2.2.4.2 Representación Basada en la prioridad de reglas

Basándose en un ámbito de job shop sin restricciones de precedencia, y con operaciones no interrumpibles. Se introduce dos formas de aprendizaje, en las cuales un algoritmo genético plantea una estrategia de control sobre las combinaciones de aprendizaje del proceso de planificación de los procedimientos locales.

Se intenta evitar los cuellos de botella descritos por Adams y utilizar la prioridad en las reglas de planificación de Panwalkar e Iskander.

Se analiza desde el algoritmo de Griffer y Thompson.

Descripción de reglas
SOT-rule (tiempo más corto de ejecución)
LOT-rule (tiempo más largo de ejecución)
LRPT-rule (tiempo más largo por terminar)
SRPT-ruIe (tiempo más corto por terminar)
LORPT-rule (Operación con mayor tiempo de procesamiento pendiente)
Ramdon
FCFS-rule (Primero llega, primero se atiende)
SPT-rule (Menor tiempo de procesamiento)
LPT-role (Mayor tiempo de procesamiento)
LOS-rule (operación posterior Más larga)
SNRO-rule (Menor número de operaciones pendientes)

LNRO-rule (Mayor número de operaciones pendientes)

Rule Index	Rule
1	Shortest processing time
2	Longest processing time
3	Shortest remaining time
4	Longest remaining time
5	Latest start time
6	Latest finish time

Ilustración 27 Cromosoma con la representación basada en prioridad (Dahal, Cowling, & Tan, 2007)

Se sugiere la idea que la eficiencia no es muy buena por el gran coste computacional. Además, individuos sin planificar podrían confundirse durante la planificación, la inversión de Holland tampoco es efectiva.

Sí que merecería la pena dentro de la resolución heurística de un algoritmo de simulación de alineación o de búsqueda tabú, usar un individuo de algoritmo genético que represente las decisiones en la secuencia que se toman a nivel local.

2.2.4.3 Representación basada en la preferencia de lista

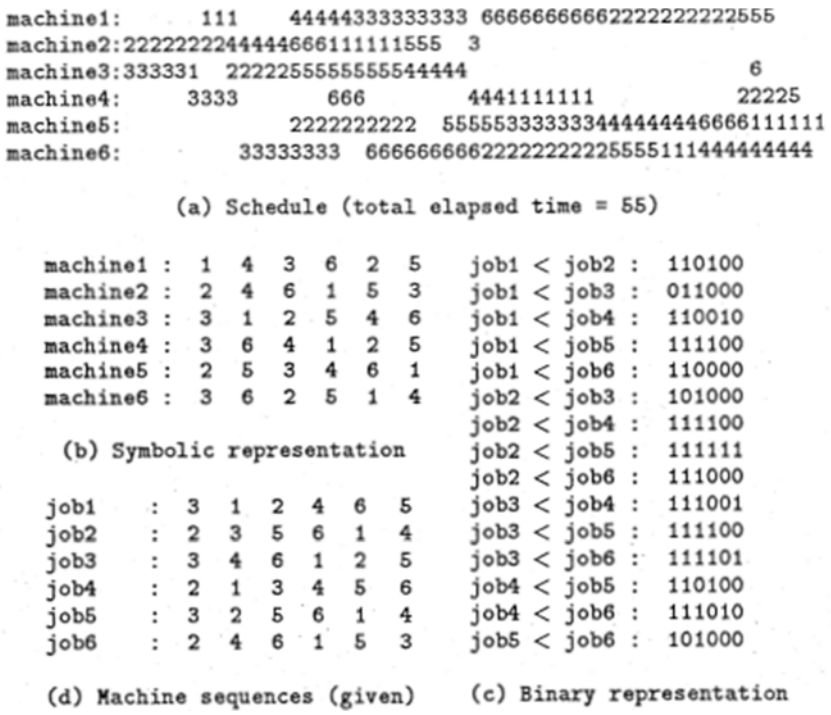


Ilustración 28 La buena planificación tiende a la prioridad de procesamiento (Nakano & Yamada, 1992)

Nakano en su artículo afirma que la representación simbólica es más efectiva para los problemas de planificación, aunque requiere la modificación de los operadores de cruce y de mutación. Pero el algoritmo genético convencional se ajusta menos que la representación simbólica.

En sus ejemplos aplica una representación de pareja de trabajos [j,k].

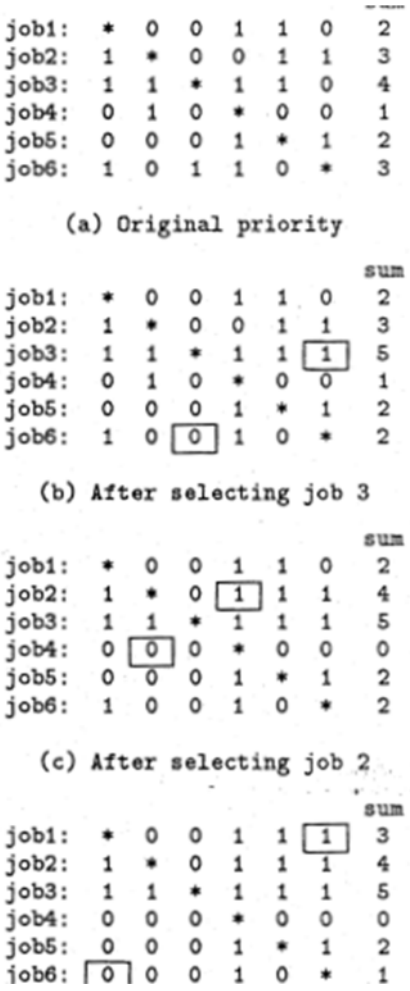


Ilustración 29Proceso de armonización por el que un genotipo ilegal pasa a ser legal (Nakano & Yamada, 1992)

(Nakano & Yamada, 1992)

El esquema que se aplica por debajo de este tipo de representación intenta usar una regla de asignación inherente sencilla, que se expresa en una lista de prioridades de precedencia, existe un taller de reglas por estación, hay una lista de preferencias en cada estación en el cromosoma.

Una lista de preferencia no es más que una lista de todas las operaciones que se deben ejecutar en la estación. En cualquier caso, en comparación con las codificaciones previas, la interpretación como lista es diferente. Cada vez que la estación termina una operación se elige un trabajo de los trabajos en espera de acuerdo con esta lista. …

Ejemplo.

Recurso	Operaciones
Machine 1:	O_{11}, O_{22}, O_{31}
Machine 2:	$O_{13}, O_{21}, O_{33}, O_{24}, O_{35}$
Machine 3:	$O_{12}, O_{23}, O_{32}, O_{34}$

Hay tres trabajos que tienen 3,4 y 5 operaciones cada uno para realizarse sobre tres máquinas. Los correspondientes cromosomas que los constituyen son tres listas de preferencia, una por cada máquina. Dos ejemplos:

Cromosoma 1:	Cromosoma 2:
O_{11}, O_{22}, O_{31}	O_{11}, O_{22}, O_{31}
$O_{13}, O_{21}, O_{33}, O_{24}, O_{35}$	$O_{13}, O_{21}, O_{33}, O_{24}, O_{35}$
$O_{12}, O_{23}, O_{32}, O_{34}$	$O_{12}, O_{23}, O_{32}, O_{34}$

La planificación por tanto es definida implícitamente por los cromosomas. De forma que la actual sucesión de operaciones en la “tienda”, deberá ejecutarse una simulación de la tienda. Dada el tiempo más temprano de inicio de las tareas y las longitudes de cada operación, la composición de las colas de espera en que cada estación puede pronosticar y cederlas y entregar reglas, las operaciones pueden ser terminadas a tiempo.

(Falkenauer & Bouffouix, 1991)

En otro artículo a raíz de un ejemplo se proporcionan más características para la representación, y explica que intenta evitar dicha representación, las planificaciones inválidas que se aplican de la traducción del genoma.

De dos tareas y dos máquinas:

Trabajo 1: op1(M1) op2(M2)

Trabajo 2: op3(M2) op4(M1)

Supón que el cromosoma es:

M1: op4 op1

M2: op2 op3

En este caso se define una situación de bloqueo, puesto que para ejecutar op4 se necesita ejecutar op1 primero y para ejecutar op3, se necesita ejecutar op2 primero.

Para solucionarlo se puede modificar en genoma para producir una planificación válida o codificar de otra forma los cromosomas para producir planificaciones válidas.

Cada cromosoma está compuesto por muchos subcromosomas, uno por máquina

Cada subcromosoma es una cadena de símbolos, cada símbolo identifica una operación que será procesada por una máquina en particular

Subcromosomas no describen la secuencia de operaciones sobre la máquina, debido a las listas de preferencia, donde cada máquina tiene su propia lista de preferencia que es usada al final de cada operación donde se tiene que elegir entre diferentes operaciones a la espera de ser planificadas, en ese caso la operación que aparezca primero en la lista de preferencia será elegida.

La planificación actual es elaborada desde el cromosoma a través del que se realiza una simulación que analiza el estado de las colas en espera que posee cada máquina, si es necesario usa la lista de preferencias para fijar una planificación. El procedimiento en que obtiene la planificación es el mismo sistema en que lo hace las reglas de prioridad con la diferencia son diferentes para cada máquina.

Los límites de este tipo de codificación serían:

Definición 1: Una planificación es activa si ninguna operación puede ser empezada antes sin retrasar otra operación violar las restricciones de precedencia. Esto demuestra que la planificación óptima se encuentra en funcionamiento

Definición 2: Una planificación es sin demora, si ninguna máquina permanece ociosa cuando hay al menos un trabajo esperando a ser planificado por la máquina. El conjunto de planificaciones sin demora es un subconjunto apropiado de planificaciones activas, por lo tanto, la solución óptima, es aquella que debe ser activa para planificar, no es necesario que exista demora, una instancia de este caso se informará

Además, se propone el uso de un algoritmo de predicción para mejorar sobre todo las planificaciones que tengan un peor rendimiento y se repiten en diversas ocasiones.

(Della Croce, Tadei, & Volta, 1995)

2.2.4.4 Representación Basada en el tiempo de terminación

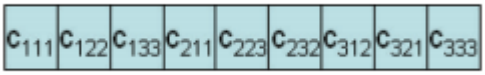


Ilustración 30 Siendo de Cjom, j=numero de tarea, o operacion de la tarea, m la máquina (Dahal et al., 2007)

Cada individuo psn representa una planificación activa, directamente usando los elementos $\{psn_{j,i,r}\}$ de la operación de finalización de acuerdo con la expresión A (por ejemplo, $psn_{j,i,r} = c_{j,i,r}$)

El tiempo total S_{psn} de la representación del individuo psn es calculada de la forma siguiente:

$$S_{psn} = \max \{psn_{j,i} \mid 1 \leq j \leq n, i = m\}$$

2.2.4.5 Representación basada en llaves aleatorias

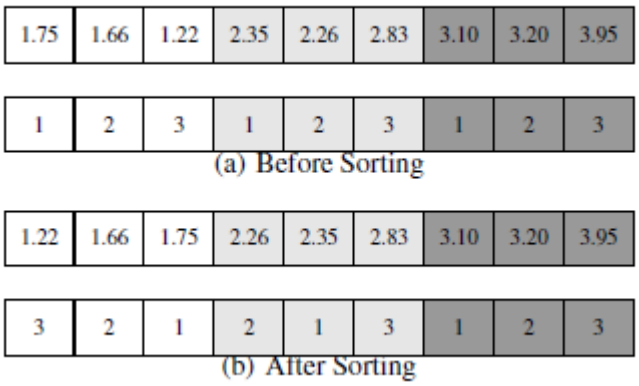


Ilustración 31 (Uyar, 2013)

Muy similar a la representación basada en operación que se verá más adelante.

Esta representación codifica una solución con números aleatorios. Estos valores son usados como una llave ordenada para decodificar la solución. Las llaves aleatorias eliminan el problema de la factibilidad de la descendencia por el uso de la codificación cromosómica que representa soluciones de una manera suave. Estas codificaciones son

interpretadas como la rutina evaluación objetivo de la forma en que afronta el problema de factibilidad.

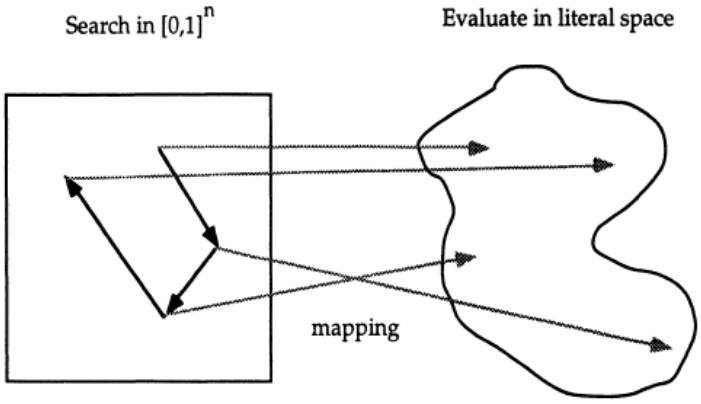


Ilustración 32 Proceso de llaves aleatorias (Bean, 1994)

(Bean, 1994)

2.2.4.6 Representación del trabajo basada en Permutación

Christian Bierwirth alude a su técnica de representación como permutación con repetición. Se intenta alinear en el ámbito de esquemas de solución del problema de vendedor viajero (TSP), con la ventaja de no producir cromosomas "ilegales", puesto que todo se encontraría ya ordenado dentro de las posibilidades reales.

Se recomienda el cruce OX, cruce ordenado.

n-trabajo

m-máquina

G indica que tareas son conectadas con reglas de producción tecnológica, describiendo su orden de procesamiento entre las máquinas. G es especificado como una matriz.

$$T= [T_i]_{i\leq i\leq n} \quad T_i=(M_{\Phi i[1]}, M_{\Phi i[2]}, \ldots M_{\Phi i[m]})$$

De vectores T_i, que representa una regla tecnológica del trabajo J_i, que es una permutación de todos (o un subconjunto de todos) máquinas. En caso de un subconjunto apropiado J_i es procesado por $m_i < m$ máquinas, de las cuales los últimos $m - m_i$ componentes de vector T_i son definidos como "máquinas zero" que indican que no tratarán como más trabajos de J_i. La operación de J_i que es procesada por M_j se caracteriza como O_{ij} y su tiempo de procesamiento como p_{ij}. El símbolo "*" llama a la medida de eficiencia de optimización

(Bierwirth, 1995)

La permutación se podría leer de la siguiente forma

Ecuación 1 (Bierwirth, 1995)

$$\{\underbrace{\pi_1, \ldots, \pi_{n_1}}_{\text{resource } 1}, \underbrace{\pi_{n_1+1}, \ldots, \pi_{n_2}}_{\text{resource } 2}, \ldots, \underbrace{\pi_{n_{m-1}+1}, \ldots, \pi_n}_{\text{resource } m}\}$$

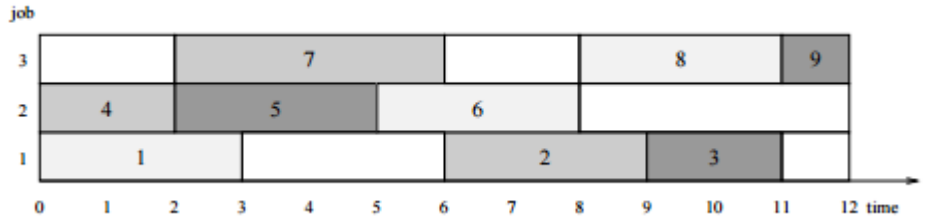


Ilustración 33Feonotipo de una solución JSP (Bierwirth, 1995)

Las conocidas como simple-GA sirve como una plantilla de algoritmo genético. En las tres versiones de SGA, se usan de forma alternativa GOX, GPMX y PPX. La tasa de cruce se sitúa a 0.6 mientas no se use la mutación. El tamaño de la población es situado a 500 individuos y el algoritmo termina después de 100 generaciones. Cada versión de los tres SGA se ejecutó un total de 10 iteraciones, sobre la marca de calidad. El error relativo al optimo y la desviación estándar fue registrada como media a través de cinco tipos de problemas.

operator	rel. err.	std. dev.
GOX	12.63	1.28
GPMX	4.01	1.10
PPX	1.62	0.50

Ilustración 34 Rendimiento de los principales cruces (Bierwirth, 1995)

(Bierwirth, 1995)

2.2.4.7 Representación basada en pareja de relación y trabajo

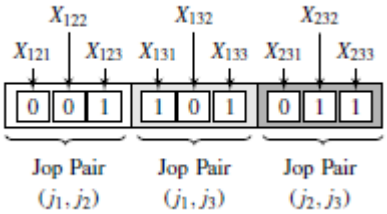


Ilustración 35(Uyar, 2013)

Es sencilla cada variable representará una operación. El domino de una variable representa los posibles tiempos de inicio asociados a una operación. Este dominio es inicializado para delimitar intervalos cerrados del tiempo más temprano y más tardío posible de inicio en la operación. Se hará efectivo desde la fecha de despliegue hasta la fecha que deba dentro de la duración de las operaciones que pertenezcan al mismo trabajo.

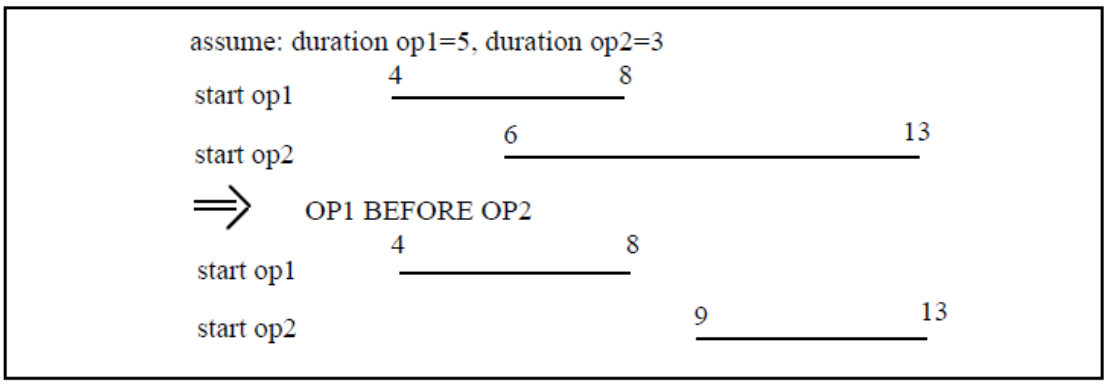


Ilustración 36Ejemplo de restricción de precedencia, op2 no puede preceder a op1 (Paredis, 1992, p. 5)

La operación inicial más temprana posible Oij. Por ejemplo, la suma de la fecha de despliegue del trabajo Ji y la duración de las operaciones de trabajo Ji que preceden

Oij. De igual manera, el comienzo más tardío posible de la operación Oij es el debido al trabajo Ji menos la suma de la duración de las operaciones de trabajo Ji que son posteriores a Oij. La restricción más importante es usada aquí es la restricción de precedencia. En la que problema inicial anticipa las restricciones en cuanto a precedencia que representa un Job Shop.

(Paredis, 1992, p. 4)

2.2.4.8 Representación basada en operación

Esta representación usa un modo indirecto de representación, los individuos y sus estructuras son sencillos, pero deben ser traducidas.

Puede ser vista como una gran lista de tareas, que cuenta con NxM genes, donde N es el número de tareas y M es el número de máquinas. Dado que cada tarea se procesará sólo una vez, por lo que cada tarea aparece en la lista de trabajos exactamente M veces.

Como planificación valida, el orden de procesamiento de las tareas es proporcionado en cada máquina y en los individuos indirectos que pueden ser traducidos en planificaciones validas de acuerdo con un orden de procesamiento determinado de las máquinas

Job No.	*Ordering of Machines*
Job 1	3 1 2 4
Job 2	2 3 1 4
Job 3	3 4 1 2
Job 4	1 2 4 3

Ilustración 37 Orden de las máquinas (Fang, Ross, & Corne, 1993)

Cada operación se asignará tan pronto como sea posible a la máquina disponible, siempre siguiendo el orden de ejecución, codificado en el genotipo.

Machine No.	*Ordering of Jobs*
$M1$	4 1 2 3
$M2$	2 4 1 3
$M3$	3 1 2 4
$M4$	3 4 2 1

Ilustración 38 Orden de las tareas (Fang et al., 1993)

Dado que no es una representación que consista en una permutación, no se podrán utilizar los operadores genéticos PMX, CX, OX y ER. Se propone un cruce de intercambio de planificaciones parciales.

En esta línea, se ofrece una opción basada en una idea de Grefenstette acerca de una lista circular de trabajos sin terminar. Pudiendo referirse a una tarea de acuerdo con su posición en la lista.

El genotipo para un problema de jxm es una cadena de caracteres que cuenta con jxm partes, cada parte será mayor cuanto mayor sean el número de trabajos a realizar (j). la parte es atómica tanto como concierne al algoritmo genético. Esto proporciona la forma en que se construirán planificaciones legales, la cadena de parte abc… significa, poner primero la desligada tarea en la posición a enésima tarea no completa

Sugiere buenos resultados usando selección basada en ranking con elitismo y una tasa de cruce fija, con cerca de 300 generaciones.

(Fang et al., 1993)

2.2.5 Representación más específica en planificación de proyectos

El autor Wall en su tesis sobre planificación de proyectos bajo restricciones de recursos, sugiere algunas representaciones.

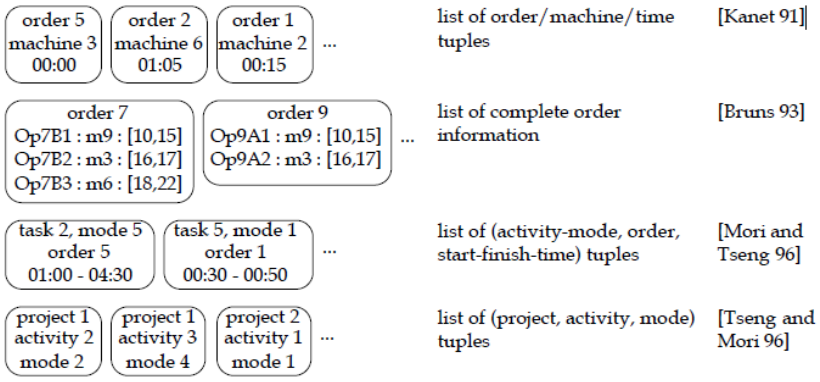


Ilustración 39Análisis de Matthew Wall(Wall, 1996)

2.2.5.1 Lista de tuplas de orden, máquina y tiempo

Se utiliza para problemas de un conjunto de únicas operaciones. En que cada una tiene un tiempo de llegada, un tiempo de procesamiento nominal, y cualquier función de coste general de forma que sea convexo al tiempo de finalización del trabajo.

En su artículo Kanet y Sridharan describen la solución de PROGENITOR, dirigida a este tipo de problema de planificación de la producción.

El enfoque más importante se establece en el proceso de acoplamiento, el cual se produce en dos pasos por parte de la generación anterior, en el primer paso una primera secuencia de cola es creada. Esta sencuencia de cola es una lista de tareas ordenadas de operaciones no planificadas/no ejecutadas. En el segundo paso, los miembros de secuencia de cola, son tomados de uno en uno, y pegados en un cuadro de planificación. para formar un hijo planificado.

Hay dos versiones diferentes para crear la secuencia de cola. Una es conocida como método global. En el que se utiliza el coste total de ejecución para guiar la planificación. Quedando en el método global, en que cada miembro del conjunto de acoplamiento como una asignación a una probabilidad de selección que es inversamente proporcional a su coste total. De forma que cada miembro del conjunto de acoplamiento, cada operación en eso, está asociada a una asignación de la máquina y a un tiempo de ejecución de inicio. Para crear la secuencia de cola del hijo, un miembro del conjunto de acoplamiento es seleccionado aleatoriamente (de acuerdo a la selección de probabilidades). El tiempo más temprano de ejecución como operación de inicio es eliminado del conjunto de la pareja elegida y situado en la secuencia de cola. La operación de inicio de ejecución desde la pareja seleccionada llega a sugerir inicio en el hijo. Esto será repetido hasta que todas las operaciones hayan sido seleccionadas. Las operaciones en la secuencia de cola del hijo son mantenidas no decrecientes de tiempos de inicio sugeridos. Después de todas las operaciones de asignación para la secuencia de cola, las operaciones serán pegadas al cuadro de planificación, en orden específico por la secuencia de cola. De forma que la asignación de máquina y el tiempo de inicio es inherente al conjunto de parejas. Este proceso se repite el número de veces que sea especificado en cada generación para crear la siguiente población.

(Kanet & Sridharan, 1991)

2.2.5.2 Lista de tuplas de modo de actividad, orden, tiempo de inicio y finalización.

Bruns en su artículo realiza un recorrido por las distintas representaciones, tanto las directas como las indirectas.

Diseña un tipo de representación directa, y como su investigación se centra en un

problema de planificación, intenta contener todas las operaciones relativas al orden con el que a los trabajos se asignan las máquinas, los intervalos de tiempo (inicio y final)

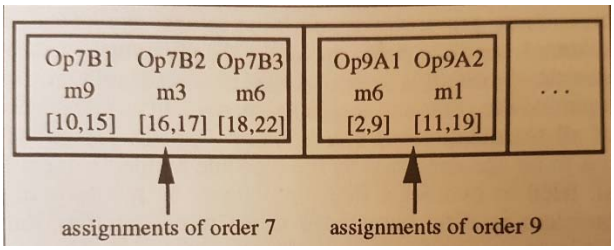


Ilustración 40 Planificación (Bruns, 1993)

La secuencia de los elementos en el cromosoma no es importante. Para determinar la calidad de un cromosoma no se requiere ninguna función de evaluación como se solía hacer en las planificaciones tradiciones, se puede hacer sin necesidad de una transformación.

(International Conference on Genetic Algorithms, Forrest, & USA, 1993)

2.2.5.3 Lista de tuplas de proyecto, actividad, modo

A1Mα	A2Mβ	.....	ANMγ
order 1	order 2		order Q
[start, finish]	[start, finish]		[start, finish]

Ilustración 41Representación directa

El planificador contendrá todas las actividades asociadas con un modo y una asignación de orden e intervalo de tiempo. El modo será elegido de forma aleatoria entre los modos disponibles y el orden de la planificación es asignado aleatoriamente con el intervalo para cada actividad.

Una vez que todos los modos y planificaciones son dados, el tiempo de finalización de cada actividad y el tiempo total del proyecto es obtenido si la planificación es realizable.

(Mori & Tseng, 1997)

2.2.6 Esquemas

2.2.6.1 Mapeo

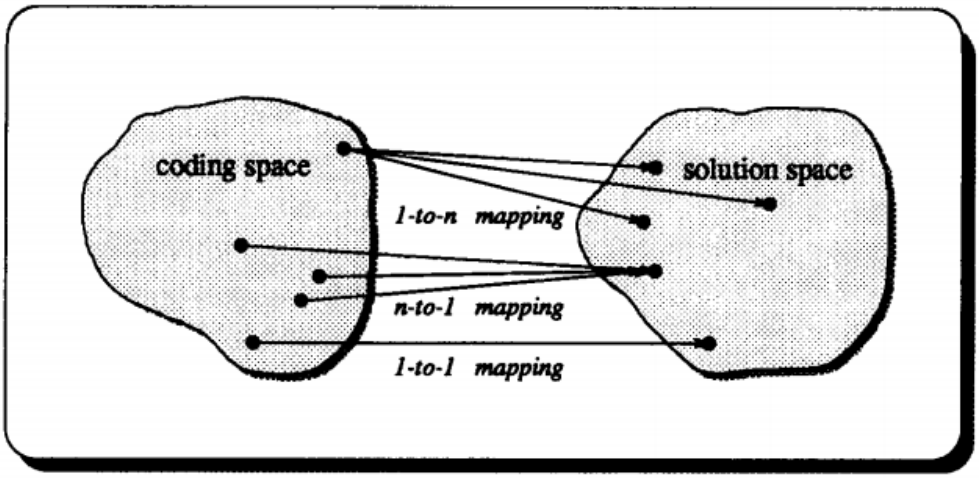


Ilustración 42Mapeo desde el cromosoma a la solución (Mitsuo Gen & Cheng, 1997)

Una de las características de los algoritmos genéticos es que se pude trabajar en el espacio de codificación y en el espacio de la solución alternativamente: las operaciones que operan sobre el espacio de la codificación (cromosomas), mientras la evaluación y la selección trabajan sobre el espacio de la solución. La selección natural crea una conexión entre los cromosomas y el rendimiento de su solución decodificada. Para las aproximaciones de codificación sin caracteres, hay tres problemas críticos a tener en cuenta entre la codificación y la decodificación entre cromosomas y soluciones (o el mapeo entre fenotipo y genotipo).

- Lo realizable del cromosoma
- La legalidad del cromosoma
- La unicidad del mapeo

Lo realizable se refiere a que una solución decodificada de un cromosoma mienta en la región del problema asignado.

Legalidad se refiere al fenómeno en que un cromosoma representa una solución a un problema.

Lo no realizable de los cromosomas proviene de la naturaleza del problema de optimización con restricciones. Todos los métodos, tanto los convencionales como los algoritmos genéticos, deben gestionar las restricciones. Para algunos problemas de optimización, las zonas realizables pueden ser representadas como un sistema de equivalencias y desigualdades (lineales o no lineales). Muchos métodos, se han propuesto para penalizar los cromosomas irrealizables. En los problemas de optimización con restricciones, el óptimo ocurre en los límites entre lo realizable e irrealizable. La penalización forzará al algoritmo genético a buscar la solución entre ambas áreas.

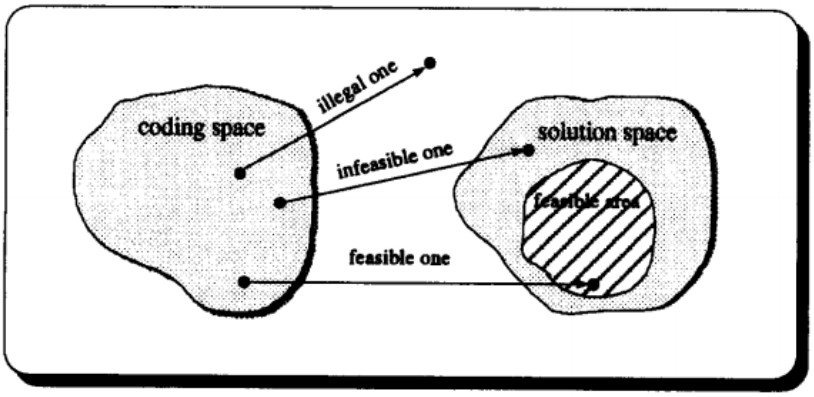


Ilustración 43Espacio de soluciones, la ilegalidad (Mitsuo Gen & Cheng, 1997)

La ilegalidad de los cromosomas proviene de las técnicas de codificación. Para muchos problemas de optimización combinatoria, el problema de codificación específica y como a través del cruce de un punto simplemente se llega a una solución ilegal. Porque la solución de un cromosoma ilegal no puede ser decodificada, esto significa que el cromosoma no será evaluado, por lo que la penalización es impracticable.

Hay técnicas de reparación para transformar un cromosoma ilegal en uno legal. Por ejemplo, el conocido como operador PMX.

Es una forma de cruce en dos puntos para representación de una permutación que después del corte posee un procedimiento de reparación para resolver la ilegalidad causada por ese simple cruce en dos puntos.

Ovorsh y Davis han mostrado que para algunos problemas de optimización combinatoria, es sencillo reparar los cromosomas irrealizables e ilegales, la técnica de reparación no supera a las técnicas de penalización o la estrategia de rechazo.

El mapeo de los cromosomas a las soluciones (decodificación) se enmarca en los siguientes casos:

Mapeo 1 a 1

Mapeo n a 1

Mapeo 1 a n

Esta última es la más temida.

(Mitsuo Gen & Cheng, 1997)

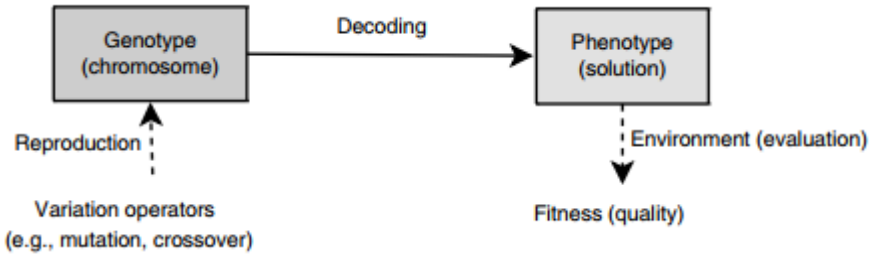


Ilustración 44Evolutionary Genotipo/fenotipo(Talbi, 2009)

2.2.6.2 Secuencia

Secuencia del algoritmo genético, para la búsqueda heurística de forma simple consiste:

Paso 1:

Seleccionar k=1

Seleccionar l secuencias iniciales $S_{1,1}$,…, $S_{1,l}$

Paso 2:

Seleccionar las dos mejores planificaciones a través de $S_{k,1}$,…, $S_{k,l}$

y llamar a estas S_k^+ y S_k^{++}

Seleccionar las dos peores planificaciones a través de $S_{k,1}$,…, $S_{k,l}$

y llamar a estas S_k^- y S_k^{--}

Generar dos descendientes S^* y S^{**} de los padres S_k^+ y S_k^{++}

Reemplazar S_k^- y S_k^{--} con S^* y S^{**}

Guardar todas las otras planificaciones por igual e ir al paso 3.

Paso 3

Incrementar k en 1.

Si k=N entonces PARADA

En otro caso ir al paso 2.

El uso de los algoritmos genéticos tiene sus ventajas y sus desventajas. Una de las ventajas es que puede ser aplicado a un problema sin saber demasiado acerca de las propiedades estructurales del problema.

Puede ser implementado de forma sencilla y proveer de soluciones completas. De todas formas, la enorme carga computacional lo hace pesado en comparación con estrategias más destinadas a un problema específico.

(Pinedo, 2009)

2.2.7 Operadores Genéticos

Los operadores genéticos permiten el desarrollo de unas características dentro de la población, que han sido definidas en un esquema.

En este acto de adaptabilidad, se producen soluciones y aproximaciones dentro del espacio de búsqueda definido en el esquema.

El hecho de poseer un esquema crea enorme potencial en cuánto al paralelismo, puesto varias poblaciones pueden seguir un mismo esquema, acotando sus soluciones con respecto al resto.

En el ámbito de la planificación de proyectos ciertos operadores son más útiles respecto a las características de representación del problema que el resto.

Dado que operan sobre una población que representa el problema, en este caso las tareas, para preservar ciertas características se requiere unos operadores particulares.

2.2.7.1 PMX

Desarrollado por Goldberg y Lingle, para la resolución del problema del viajero.

A = 9 8 4 5 6 7 1 3 2 10

B = 8 7 1 2 3 10 9 5 4 6

PMX procedería como se describe a continuación. Primero, dos posiciones son elegidas de una cadena de caracteres aleatoria. Las subcadenas definidas desde el primer número al segundo número conocidas como sección de mapeo. Las dos secciones de mapeo se intercambian. Por ejemplo, la selección aleatorio de 4 y 6, produciría el intercambio de A(5,6,7) con B(2,3,10), haría el intercambio secuencial. Posteriormente se situarían los elementos que no estuvieran en el intercambio y después los que sí, también con situando el elemento de nuestra cadena que hubiera sido intercambiado con elemento de la otra cadena

A' = 9 8 4 2 3 10 1 6 5 7

B' = 8 10 1 5 6 7 9 2 4 3

(Goldberg, Lingle, 1985)

En el libro de Goldberg aparece una versión modificada, dónde los puntos no encontrados en ambas cadenas pertenecientes al cruce, se sitúan en primera posición el cruce de la otra cadena, el cruce de nuestra cadena y por último de la parte derecha y los de la parte izquierda, ambos que no coincidieran con el cruce de la otra cadena.

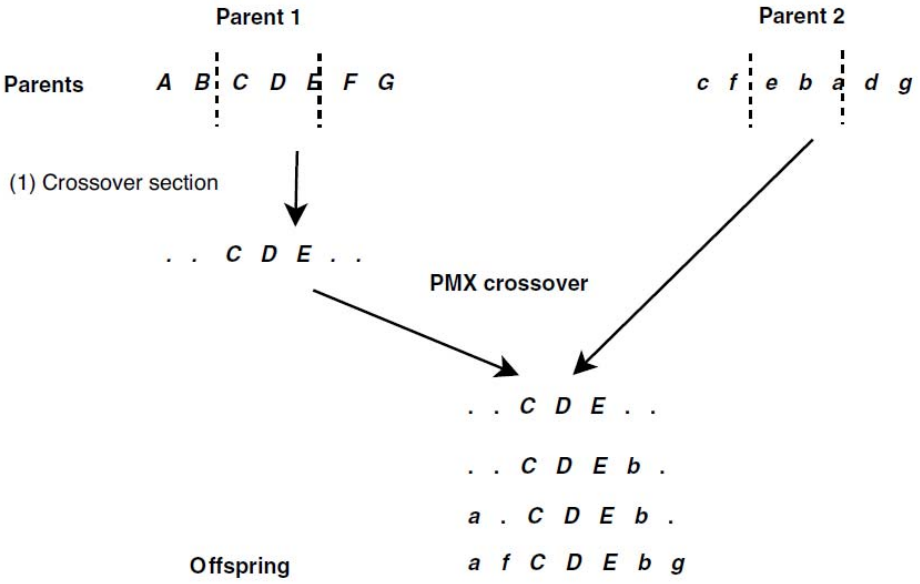


Ilustración 45 Cruzamiento parcialmente encontrado. (Talbi, 2009, p. 219)

2.2.7.2 OX

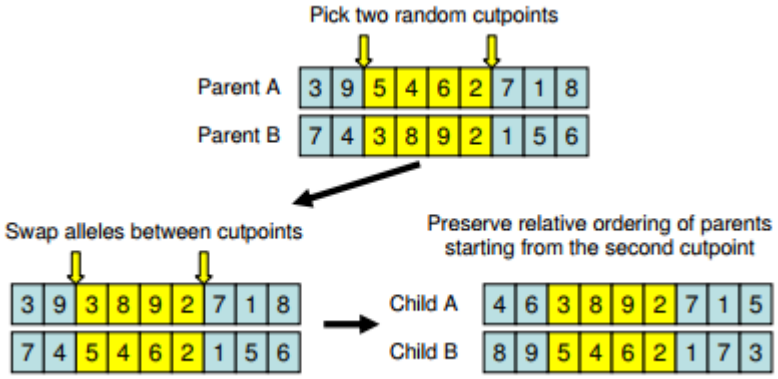


Ilustración 46Ejemplo de cruce ordenado (Dahal et al., 2007)

Davis propone una forma de búsqueda de una solución legal, a través de no evitar realizar el cruce. De esta forma introduce un paso intermedio en el intercambio de dos puntos y luego lo reordena según las preferencias que tuvieran los cromosomas padres.

(International Conference on Genetic Algorithms and Their Applications, Grefenstette, & Texas Instruments Incorporated, 1988)

Aunque en su libro Handbook of genetic algorithm Davis lo explica con claridad, el mecanismo sería el siguiente.

Esto es un mecanismo de reparación, y la solución siempre será real.

Paso 1: seleccionar dos puntos de corte aleatorios.

Paso 2: Intecambiar cadenas entre los dos puntos de corte para crear la prole.

Paso 3: En cada generación, determinar las posiciones de los símbolos en que están aquello que se ha recibido en el intercambio.

Paso 4: Empezando desde el segundo punto de corte, asignar los símbolos en las posiciones que se determinaron en el Paso 3.

(Uyar, 2013)

2.2.7.3 PBX

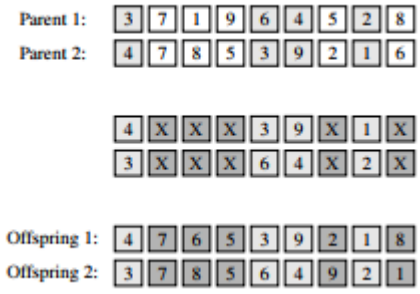


Ilustración 47(Uyar, 2013)

Lo propuso Sysweda en 1991.

Similar al operador de cruce uniforme y aplica un procedimiento de reparación de la prole ilegal.

Paso 1: Genera una máscara aleatoria ilegal y produce una generación por la aplicación del intercambio multi corte entre los padres de acuerdo a una mascara

Paso 2: Determinar la posición de los genes transferidos al otro individuo.

Paso 3: Situar los símbolos transferidos al otro padre, de izquierda a derecha.

(Uyar, 2013)

2.2.7.4 OBX

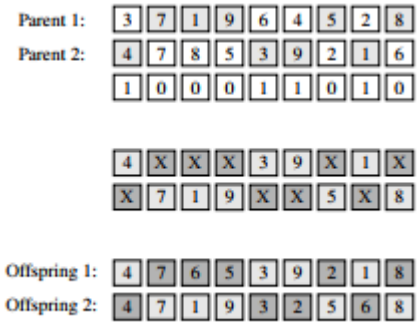


Ilustración 48(Uyar, 2013)

Propuesto por Syswerda en 1989, es una pequeña variación de PBX en el que el orden de los símbolos en la posición seleccionada en uno de los padres es impuesto sobre los correspondientes en los otros padres.

Paso 1: Generar una máscara aleatoria.

Paso 2: Tomar los símbolos del padre 2 donde la máscara es 1 para producir la primera generación y tomar los símbolos del padre 2 donde la máscara es 0 para generar el segundo hijo.

Paso 3: Situar los símbolos en el hijo 1 y el hijo 3 en el orden que tuvieran en el padre 1 y 2, respectivamente.

(Uyar, 2013)

2.2.7.5 CX

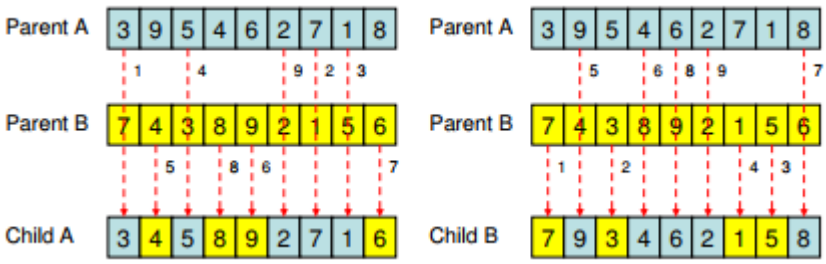


Ilustración 49Ejemplo de operador de Ciclo (Dahal et al., 2007)

El cruce de ciclo es un tipo de operador de cruce distinto, Realiza la recombinación bajo restricción de que cada nombre de ciudad viene de un padre u otro. Se puede ver en el ejemplo:

C= 9 8 2 1 7 4 5 10 6 3

D= 1 2 3 4 5 6 7 8 9 10

En lugar de elegir puntos de intercambio, elegimos empezar desde la ciudad del primer padre

C'= 9 - - - - - - - -

Como en la posición del otro padre, la posición del 9 la ocupa un 1, situamos el uno del primer padre en la misma posición en el hijo.

C'= 9 - - 1 - - - - - -

De la misma forma, completamos el patrón.

C'= 9 - - 1 - 4 - - 6 –

La cadena se completaría del padre D.

C'= 9 2 3 1 5 4 7 8 6 10

D'= 1 8 2 4 7 6 5 10 9 3

(Goldberg, 1989)

2.2.7.6 LOX

En su artículo Falkenauer y Bouffouix introducen el operador genético de cruce de orden lineal, partiendo del PMX de Goldberg, que permite la transmisión del esquema y el OX que Goldberg describe como lista circular.

Dado un cromosoma 1,2,3,4,5,6,7,8,9

Y suponiendo que existan dos individuos:

123456789

769432158.

El operador LOX difiere del original OX en que la información sobre las prioridades altas y bajas no se pierden durante el cruce, por ejemplo, la significancia entre el principio y el final del cromosoma se preservan. Esto es debido a lo siguiente: empezando con el mismo cromosoma con la sección de cruce de genes establecida aparte:

7H9|H32|1H8

Los genes son desplazados a la izquierda del punto cuando no queda espacio en la sección de cruce del lado izquierdo. Estos campos

793|HH2|1H8.

Entonces los genes son desplazados a la derecha, dejando sólo huecos en la sección de cruce, estableciendo

Finalmente, la sección de cruce es ocupada con el resto de genes, de la misma forma como el primer padre

793|456|218

Este nuevo cruce preserva tanto como es posible las posiciones relativas entre los genes y la posición absoluta entre los extremos del cromosoma, Además esta última información es transmitida entre ambos padres durante el cruzamiento como sigue. Asumiendo que el primer padre transmite el contenido de su sección de cruce, cuando la sección de cruce es definida en el centro del cromosoma (significa que el hijo heredará las operaciones de prioridad media del primer padre, tanto las operaciones de prioridad baja como la prioridad alta se obtendrán del segundo padre. De forma contraria, cuando la sección de cruce se sitúa en alguno de los extremos del cromosoma, el hijo heredará las operaciones tanto de prioridad alta como de prioridad baja del primer padre porque las zonas de cruce son la misma en ambos padres.

(Falkenauer & Bouffouix, 1991)

2.2.7.7 PSXX

Dado que la representación utilizada no es una permutación de trabajos. Se propone una planificación parcial. En primer lugar, la planificación parcial es elegida de forma aleatoria de los padres. La planificación parcial es identificada con el mismo trabajo tanto en la primera como en la última posición de la planificación parcial. Así intercambian las dos planificaciones parciales. Con frecuencia, las planificaciones parciales tienen varían en longitud (contienen diferente número de genes) La descendencia generada después se intercambie será mayor o más corta que la longitud que tuviera, de forma que se permite añadir y borrar genes para formar una descendencia legal.

p_1= 3 2 1 2 3 **4 1 2 4** 4 1 3 4 1 2 3

p_2= **4 1 3 2 1 3 4** 1 2 3 4 2 1 2 3 4

o_1= 3 2 1 2 3 **4 1 3 2 1 3 4** 4 1 3 4 1 2 3

o_2 = **4 1 2 4** 1 2 3 4 2 1 2 3 4

El orden de los genes en la planificación parcial es importante en la teoría de bloques constructivos. De forma que se debe preserver el orden del original.

En la planificación o_1, se deben borrar dos 3, y un 1. En la planificación p_2, el primer 3, de la planificación parcial es el primer 3 en la planificación entera. De forma que dos 3, el primer 3 y segundo 3, de o_1 pueden ser borrados. De la misma forma, el 1 de la planificación parcial en p_2 es el primer 1 en la planificación entera, dando lugar a que el primer 1 en o_1 sea borrado.

En la planificación o_2 , se deben añadir dos 3 y un 1. El 1 en la planificación parcial de p_1 es el segundo 1 de la planificación completa, de forma que un 1 se agregó como planificación parcial en o_2.

o_1= 2 2 **4 1 3 2 1 3 4** 4 1 3 4 1 2 3

o_2= 1 **4 1 2 4** 3 1 2 3 3 4 2 1 2 3 4

La mutación utilizada es la de intercambiar dos posiciones si son distintas y repetir el proceso si son iguales, resultando

p_1 = 1 2 4 **3** 2 1 4 2 2 3 1 **4** 1 4 3 3
o_1 = 1 2 4 **4** 2 1 4 2 2 3 1 **3** 1 4 3 3

(M. Gen, Tsujimura, & Kubota, 1994)

2.2.7.8 PPX

La principal ventaja de este cruce es que conserva de forma absoluta el orden de los cromosomas de los padres. La primera generación de cromosomas se inicializa vacía. Entonces el vector de longitud n es

parent 1	3	2	2	2	3	1	1	1	3
parent 2	1	1	3	2	2	1	2	3	3
gene of parent	1	1	2	2	2	2	1	1	1
PPX offspring	3	2	1	1	2	1	2	3	3

PPX se aplica como se muestra en la figura a un cruce en dos puntos estilizada, por ejemplo, la dirección de elección entre los padres cambia dos veces (tercera línea) Para aplicar PPX de una forma estilizada como cruce uniforme las opciones serían elegidas alternativamente de forma aleatoria. En cualquier caso, el orden absoluto entre dos genes en la descendencia al menos uno de los cromosomas tiene su origen en parental y se conserva. Todos los operadores se propones como adecuados para presentar una solución al final factible. Se espera que GOX para transmitir el orden relativo de los genes. GPMX tiende a transmitir las posiciones de los genes para respetar algunas de las ampliaciones.

PPX respeta el orden absoluto de los genes procurando la preservación de precedencia relativa entre genes.

(Bierwirth, Mattfeld, & Kopfer, 1996)

2.2.7.9 POX

Este cruce puede ser considerado como la columna vertebral de los algoritmos genéticos. Intenta portar de forma inherente la información sobre la solución de dos padres hacia una o más soluciones de los hijos. Para aplicar la operación de cruce de forma adecuada al JSP, se debería considerar la siguiente pauta, completitud, factibilidad, no redundancia y conservación.

El nuevo operador de cruce de precedencia, se propone para la representación basada en operación, la cual se adecua a la pauta de conservación, completitud y propiedad de factibilidad entre padre y sus hijos. El operador de cruce efectivo que se describe a continuación:

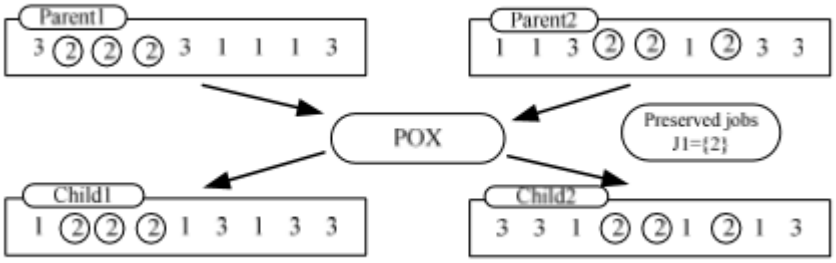
Dado un cromosoma, Padre1 y Padre2 que generan como hijos, Hijo 1 e Hijo 2, de la

Siguiente manera:

1. De forma aleatoria se elige un conjunto de trabajo del conjunto de número de trabajos {1,2,… n} en un subconjunto exclusivo y no vacío de J1.
2. Se copia aquellos números de J1 del Padre 1 al Hijo 1, y del Padre 2 al Hijo 2, preservando sus Locus.
3. Copia aquellos números en J1 que no han sido copiados en el paso 2, desde el Padre 2 al Hijo 1 y desde el Padre 1 al Hijo 2. Preservando su orden.

En la figura se muestra un ejemplo de operación de cruce de precedencia (POX) de un problema 3x3, el locus del trabajo {2} se conserva. El cruce del padre 1 y el padre 2 genera dos cromosomas hijos, hijo 1 {1 2 2 2 1 3 1 3 3} e hijo 2 {3 3 1 2 2 1 2 1 3}. Esto puede ser visto como la preservación de Hijo 1 preserva el Locus y el orden del trabajo {2} en el padre 1 y el orden de trabajo {1,3} en el padre 2, el Hijo 2 conserva el Locus y el orden del trabajo {2} en el Padre 2 y el orden del trabajo {1,3} en el Padre 1.

Por lo tanto, POX es un buen exponente en la conservación de características.



(Zhang, Li, Rao, & Li, 2005)

2.2.8 Problemas

2.2.8.1 Cuello de botella

La mayoría de los trabajos de planificación de job shop descritos en la literatura se basan en reglas de planificación por prioridad. Estas reglas eligen la siguiente operación de un conjunto específico de tareas. Pueden utilizar un criterio como el menor tiempo de procesamiento, SPT, más trabajo pendiente MWKR, primero en llegar/primero en ser atendido FCFS… El subconjunto de tareas que es posible elegir produce una ejecución activa (pasarán a ejecutarse sin demora, no habrá ninguna otra actividad pendiente) lo cual suele ser lo pretendido en las planificaciones sin demora (como ninguna máquina está ociosa en ningún momento cuando empiezan a ejecutar las operaciones). Los procedimientos avariciosos suelen tener un único procedimiento de paso, en el que se construye la solución en lo que parece ser a nivel local la mejor secuencia de operaciones que llevan hasta el final.

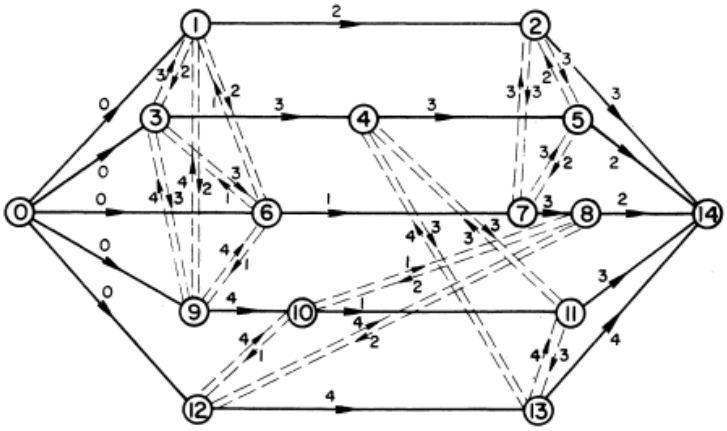


Ilustración 50Identificando cuello de botella (Adams, Balas, & Zawack, 1988)

Como la mayoría de los procedimientos de este tipo en otras áreas de optimización, aquellas heurísticas veloces suelen encontrar soluciones no muy malas. En algunas situaciones su uso está harto justificado. En cualquier caso, con el incremento de la velocidad de los computadores y la necesidad de una planificación eficiente, esto incrementa de forma importante los caminos de explorar para obtener planificaciones con algo de coste extraordinario computacional, comprobando de forma breve si se puede obtener una planificación optima garantizada.

Para obtener esta planificación optima se atenderá a ordenar las máquinas de la forma que debe establecerse la secuencia. Basándose en la clásica idea de dar prioridad a las máquinas del cuello de botella.

Para priorizar las máquinas. Es necesario expresar el concepto de calidad de cuello de botella como el grado en que importa una propiedad de si o no. Dicha calidad puede ser medida, como la utilidad marginal de reducir el tiempo total de ejecución, siendo difícil encontrar el para más tarde. Se puede medir la calidad de cuello de botella de una máquina k, con el valor de una solución óptima para ciertas planificaciones de un problema sobre una única máquina.

Paso 1. Identificar el cuello de botella m a través de las máquinas k Є M\ M_o y la secuencia óptima. Asignar M_o <- M_o U {m} e ir al paso 2

Paso 2: Re optimizar la secuencia para cada máquina crítica k Є M_o en activo, mientras guarda las otras secuencias bien formadas, por ejemplo M'_o := M_o – {k} y resolver P(k,M'_o). Entonces si M_o =M, parar, en otro caso ir a 1.

(Adams et al., 1988)

2.2.8.2 Lamarkian

Es un concepto que consiste en introducer a poblaciones donde se ha aplicado ya algún operador genético a través de generaciones. Tiene sus pros y contras, el principal beneficio es que puede traer lo mejor del conjunto de cromosomas, el peligro que solo proporcione un óptimo local.

2.2.9 Librerías y utilidades

2.2.9.1 Galib

Fue desarrollada por Matthew Wall, del Massachusetts Institute of Technology.

El autor desarrolla esta librería a raíz del uso de una librería de Kazu Saito en la MIT CADlab. De tener una licencia de MIT, paso a tener una licencia libre en el año 1999, tanto para usos educativos como usos económicos.

Cuenta con diversos tipos de genoma, dentro del conjunto de estructura de datos que hacen uso de ellos. Estas estructura spueden ser árboles, listas, cadenas binarias y arrays.

El paso de las generaciones se puede controlar de acuerdo a unos parámetros que se introducen al genoma, y permiten modificar la forma de cruce, de mutación, de reemplazo de las generaciones padre, …

3 Aseguramiento de la calidad.

"Measuring programming progress by lines of code is

like measuring aircraft building progress by weight."

El proceso de medición de la programación por líneas de código es cómo medir el progreso de construcción de un avión por su peso.

Bill Gates

3.1.1 Introducción

El concepto de calidad va unido a al desarrollo software. Comienza al igual que la Ingeniería del software en los años 70, cuándo se empiezan a cuantificar ciertos parámetros. Se establecen métricas oportunas para sacar conclusiones en los conocidos como atributos de calidad, que serían datos derivados de dichas métricas.

La mayoría de los ingenieros afirman que McCall publicó la primera definición de calidad de Software, 1977, afirma que algunos atributos son indispensables, corrección, comprensión, portabilidad, fiabilidad, mantenibilidad, reusabilidad, eficiencia, flexibilidad, interoperabilidad, integridad, complejidad, usabilidad, facilidad de pruebas, modularidad, documentación, coste, tiempo. La documentación, por ejemplo, no es un atributo de calidad, pero puede obtenerse una medida de calidad.

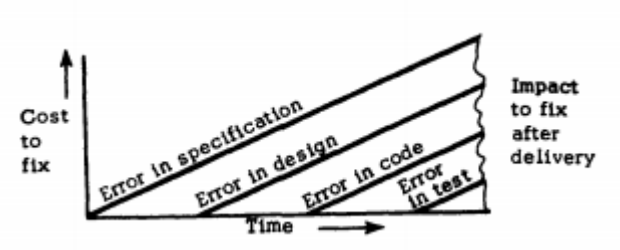


Ilustración 51 Impacto del error

También considera una ahora considerable encontrar errores en etapas tempranas del desarrollo del software.

En el manual de medición practica de software (Practical Software Measument), en base a casos prácticos advierte sobre los peligros en pérdida de calidad de acelerar las planificaciones, los recursos y los costes, cuándo se debería dar al usuario un producto sin errores. Aconseja para esto último extender la planificación de la calidad.

Boehm aporta con rotundidad en su artículo evaluación cuantitativa de la calidad del software, tres atributos que considera indispensables como son

- Claro y entendible que sea un programa y su dificultad para ser modificado.
- Usabilidad, la facilidad con la que se pueda usar un programa, incluso en estudios más recientes la usabilidad es un factor indispensable a la hora de elegir un proceso de desarrollo puesto que el personal necesitará formación y eso repercutirá en la motivación
- La portabilidad, lo dependiente que sea de la máquina.

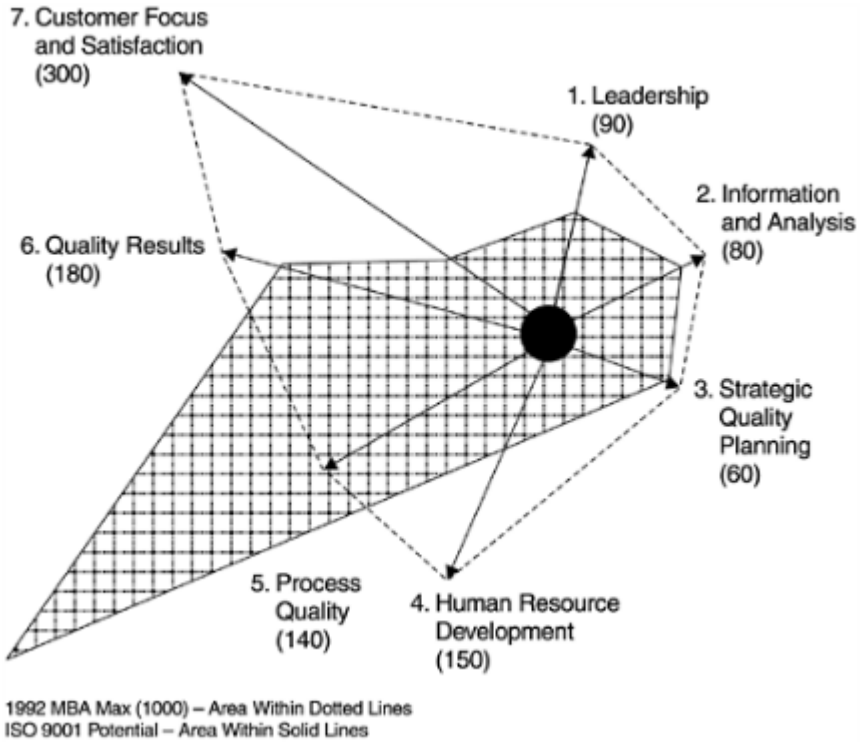


Ilustración 52Evaluación de Malcom Baldrige y ISO 9000 (Kan, 2003)

Además, incluye unas métricas de capacidad del cs-13 según las características en el programa como son el formato y tipo de datos, número de entradas, orden de entradas, unidades, rangos de aceptación, localización de almacenamiento asociados, recursos (lógicos y físicos) y acceso (lectura, acceso restringido)

Muy interesante también el concepto de Puntos de función desarrollado por Albretch, para IBM. Dando lugar a un análisis de la solución en lugar del programa. Asocia también las líneas de código y el lenguaje de programación a esta métrica.

A menor número de puntos de función más sencilla será la solución.

A tenor de la complejidad, sirve como referente Mccabe, que analiza la complejidad del flujo, a través de la complejidad cíclica. De forma que se puede establecer modelos de calidad en base a métricas de complejidad de cambio y modificación.

Otra métricas, en la misma línea que la anterior es la de Halstead, que analiza ciertos parámetros como las entradas a las funciones, de forma que es sencillo sobre todo en la fase de pruebas establecer pautas de calidad.

Asociando de forma más directa el tiempo y el esfuerzo está el modelo de Putnam, muy relacionado con esta aproximación está Cocomo de Boehm en 1981, la versión Cocomo II llega casi en el 2000.

El ajuste del esfuerzo en los proyectos es muy importante tanto para gestionar apropiadamente los recursos como no tener que modificar la planificación. Aunque se estima que la estimación manual comete errores por encima del 10%, las herramientas automáticas como Cocomo II, tienen aproximaciones casi aproximadas. Claro, son más pesimistas y en muchos casos los clientes no aceptan dichos plazos.

Junto el desarrollo de las métricas se pasa a planificar correctamente el Proyecto y en la etapa de pruebas, se puede realizar una labor más certera, como aconseja Boehm de control de la calidad, en otro caso tanto en recorte de costos como de tiempo requerirá un enorme esfuerzo.

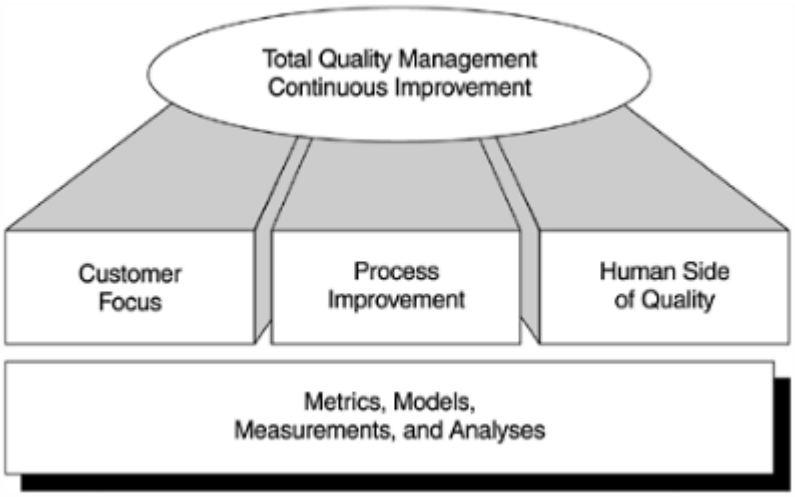


Ilustración 53Elementos llave de gestión de calidad total (Kan, 2003)

Esta disciplina se conoce como aseguramiento de la calidad, y consiste en hacer patentes durante el desarrollo del software el ajuste a unos requerimientos del usuario. Estas pruebas se automatizan para generar la mayor cobertura posible y detectar el

mayor número de errores posibles. La intensidad y número de fase de las pruebas dependerá de las métricas y en otras reglas de predicción de defectos que se comentarán. El equipo de aseguramiento de calidad, decidirá qué hacer, con objetivo en la aceptación del producto. Este camino se llama el de la mejora continua. Existiendo otras métricas como el tiempo entre fallas, tiempo de arreglo y densidad de fallos, número de pruebas y nuevos errores encontrados. Así se intentaría relacionar estas métricas con las primeras comentadas.

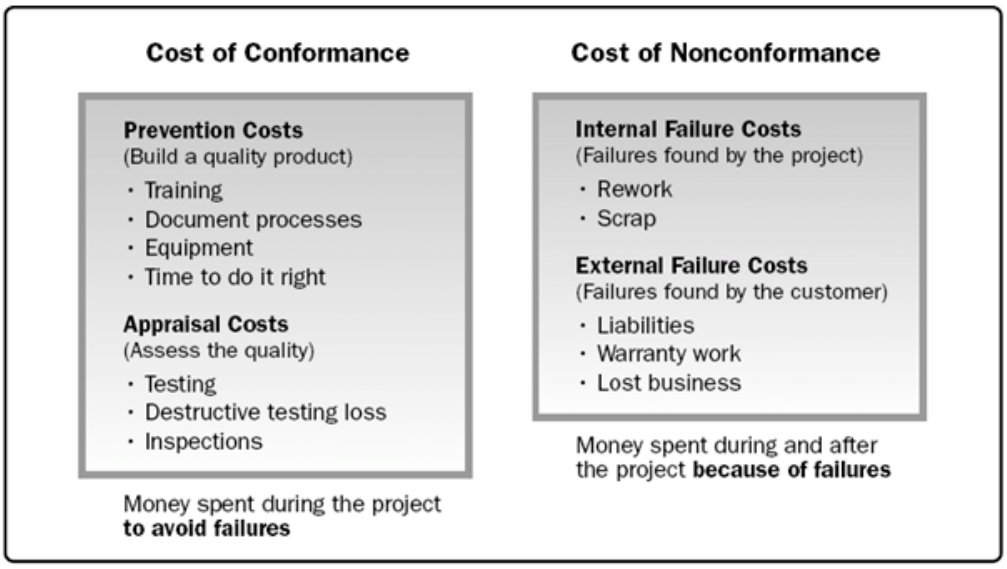


Ilustración 54 Coste de la Calidad(Project Management Institute, 2013)

Es muy importante también la influencia en los modelos de calidad de la cultura japonesa, con grandes aportes.

Para saber lo importante que es esta etapa de pruebas y validación hay que destacar el caso del Apollo dónde el 80% se destinó a pruebas.

Este ámbito está estrechamente ligado con la planificación, porque se debe controlar el trabajo que costará arreglarlo tanto en tiempo como en recursos, cómo económicos, y no introduciendo nuevos errores.

3.1.2 Métricas

Para definir este concepto retrocedo hasta el artículo Basili y Rombach de cómo aplicar el enfoque Taylor a los procesos de los objetivos y entorno de un proyecto.

Se concreta que la dificultad en la gestión del software es tres:

1. Encontrar un criterio para encontrar un proceso apropiado (Modelo de proceso global y las herramientas que soportarán dicho caso)
2. Evaluar lo positivo del proceso de software.
3. Desarrollarlo.

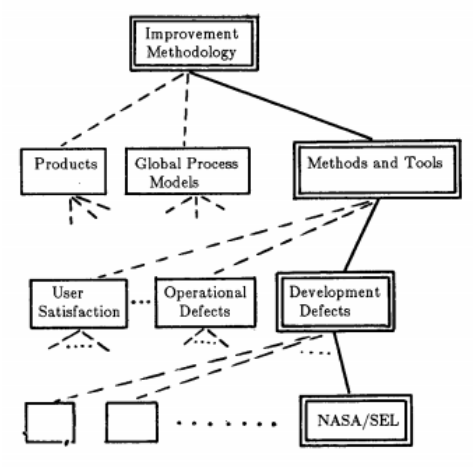


Ilustración 55 Desarrollo de un marco de trabajo de desarrollo

A modo de esquema para lo que se conoce como GQM, Modelo de calidad por objetivos, los pasos a seguir son:

1. Definir la aproximación o entorno.
2. Definir unos objetivos, cuestiones, datos para el éxito del desarrollo del proyecto y mejora a través de desarrollos de proyectos previos.
3. Elegir los métodos apropiados y las herramientas para el proyecto.
4. Llevar el desarrollo y mantenimiento de software, recabar la información que se determinó y validarla, y proporcionar información en tiempo real acerca del proyecto actual.
5. Analizar los datos y evaluar las prácticas actuales, determinar los problemas, grabar los descubrimientos y hacer recomendaciones para el desarrollo de futuros proyectos.

La caracterización conocida como GQM, el proceso de cuantificación del desarrollo de objetivos consiste en tres pasos.

1. Generar un conjunto de objetivos basados en los objetivos de la organización.
2. Derivar un conjunto de preguntas sobre los intereses o hipótesis para cuantificar dichos objetivos.
3. Desarrollar un conjunto de métricas y distribuciones que provean información necesaria para responder a las preguntas de interés.

Algunas de las preguntas de interés (pueden ser sobre el proceso, el producto, propósito de estudio, entorno, perspectiva), en cuánto al producto podrían ser:

- Atributos físicos
  - tamaño
    - Líneas de código
    - Número de unidades
    - Líneas ejecutables
  - Complejidad
    - Control
    - Datos
  - Características del lenguaje de programación.
  - Tiempo
- Coste
  - Tiempo
  - Fase
  - Actividad
  - Programa
- Cambios
  - Errores
  - Faltas
  - Fallos
  - Modificaciones
- Contexto
  - Comunidad del consumidor
  - Perfil operacional
  - Desarrollo de algún atributo de calidad en particular, corección. Qué características físicas son necesarias para analizar lo relativo a ese atributo de calidad.

3.1.3 Programas de gestión de la calidad

3.1.3.1 Sonar

Sonar es una plataforma de gestión de la calidad de software basada principalmente en el lenguaje de programación en Java, posibilita a los programadores el acceso y la monitorización del análisis de datos del código consistentes desde errores de estilo, errores potenciales, y códigos de defectos para diseñar ineficiencias, duplicación de código, carencia de cobertura en las pruebas, y exceso de complejidad.

Cada cosa que afecta al código base, desde mínimos errores de estilo hasta errores críticos de diseño, son inspeccionados y evaluados por Sonar.

- Se le pueden añadir plugin para la inspección del código.
- Se le pueden asociar perfiles de calidad.
- Medición de la complejidad del Software (Basado en las métricas de McCabe sobre complejidad ciclomática)

Sonar Severity Levels	
Severity Level	Severity Value (Weight)
Blocker	5
Critical	4
Major	3
Minor	2
Info	1

Ilustración 56Niveles de gravedad de Sonar.

Sonar puede contrastar la calidad de un código en base a un perfil de calidad.

Cuando una reglas se rompe, se anota genera la incidencia, y según se catalogue como muestra el cuadro según su peso, así afectará a la calidad.

Se puede extraer un indice de cumplimiento de las reglas, conocido como RCI basado en la severidad de la incidencia y las líneas de código.

(Arapidis, 2012)

3.2 Datos para la planificación y estudio

Los datos pertenecen a un grupo de desarrollo de Software de la universidad de Alcalá de Henares, a los cuales he tenido acceso gracias al profesor Luis Fernández Sanz.

Consisten en gestión de pruebas sobre el software antes de su entrega para el aseguramiento de la calidad.

Los datos provenían de un fichero csv, que se tuvo que portar a MS SQL, y para su manipulación he empleado linqpad, transformandolos en csv adecuados para su manipulación por parte del Programa Planificador Genético hecho en C++. Además incluyo en el anexo, los dos ficheros para la generación de csv a partir de una base de datos, y en el código del programa adjunto dicha base de datos en formato mdf.

```
> table(TFM_Dataset$`Fix Priority`)

 P1  P2  P3
819 450 210
> table(TFM_Dataset$`Fix by Date`)

01/Dec/15 01/Jan/16 02/Dec/15 03/Dec/15 04/Dec/15 04/Jan/16 04/Oct/15 05/Dec/15 05/Jan/16 06/Jan/16 07/Dec/15
       20         2        12        11        13        11         1         2         8        20        16
07/Jan/16 07/Oct/15 08/Dec/15 08/Jan/16 08/Oct/15 09/Dec/15 09/Oct/15 10/Dec/15 11/Dec/15 11/Jan/16 12/Oct/15
        3         1        16        12         2        13        23        21        22        21         3
13/Jan/16 14/Dec/15 14/Jan/16 14/Oct/15 15/Dec/15 15/Jan/16 15/Oct/15 16/Dec/15 17/Dec/15 18/Dec/15 18/Jan/16
       11        32        20         1        16        14         7         9        19        19        16
19/Jan/16 19/Oct/15 20/Jan/16 20/Oct/15 21/Dec/15 21/Jan/16 21/Oct/15 22/Dec/15 22/Jan/16 22/Oct/15 23/Dec/15
       12        17        14         6        16         4        17        10        22         6        14
23/Oct/15 24/Dec/15 25/Dec/15 25/Jan/16 26/Jan/16 26/Oct/15 27/Jan/16 27/Oct/15 28/Dec/15 28/Jan/16 28/Oct/15
       20         9         8        20         4        71         9        11        14         6        13
29/Dec/15 29/Jan/16 29/Oct/15 29/Sep/15 30/Dec/15 30/Oct/15 31/Dec/15     42310     42311     42312     42313
        5        11         8         1         8        21         1        45        14        12        12
    42314     42317     42318     42319     42320     42321     42324     42325     42326     42327     42328
       31        12        27         1        13        51        48        18        23        12        30
    42331     42332     42333     42334     42335     42338     42401     42402     42403     42404     42405
       26         8        22        10        10        17        11         9        14         7        14
    42408     42409     42410     42411     42412     42413     42415     42416     42417
       10        10        10        20        14         1         4         1         1
```

El conjunto de datos tiene un campo que también he incluido en la clase incidencia,

3.3 Datos relevantes y su significado para la calidad.

3.3.1 Tipos de Prioridades

```
> table(TFM_Dataset$Priority)

       Low      Major     Medium ShowStopper
       277        865        556        200
```

De acuerdo a su repercusión en el sistema.

3.3.2 Tipos de errores

```
> table(TFM_Dataset$`Issue Type`)

          Bug    Document Gap           Issue Requirement gap             Risk
         1546               4                            142                194                11
```

La asignación de tareas es bastante importante, pero dado que la amplia mayoría son Errores, no he considerado una significancia muy grande para la función objetivo, pero si que es cierto que en la clase class QualityIssue, he incluido el campo Type para su manipulación. Las de tipo Gap deberían tener una mayor prioridad. Las solicitudes

de gap, vendrían del analista de software, y no de una conjunto de pruebas, de forma que si se tuviera en campo Test Phase correctamente configurado, se podría "Obviar" esa importancia, puesto que según la fase de pruebas el resto perdería prioridad.

```
> table(TFM_Dataset$`Business Component`,TFM_Dataset$Priority)

                 Low Major Medium ShowStopper
  Administration   0     3      0           0
  Agent Desktop    0     0      1           0
  Claims           4    78     13           8
  Client Care      1     1      0           0
  Client File     70   121    166          17
  Despatch         1    18      1           4
  Diary           29   100     87          35
  Finance          9    33      6          13
  Follow up        7    25      8           2
  Front Office    11    12     33           2
  Maintenance      1     3      0           1
  Marketing        0     4      4           0
  Payroll & Bonus  1     3      0           2
  Reports          0    17      0           2
  Sales          108   347    180          82
  Segmentation     0     1      0           0
  Stock           15    44     18           8
  WebMethods       0     1      1           0
```

Esta relación basada en la cantidad de errores y su prioridad con relación a un Componente de Software, es lo que se pretende buscar en Pareto pero se aconseja no usar una respuesta dominante en cuanto a plasmarla directamente en el dominio de la solución, cómo se menciona en el punto 42

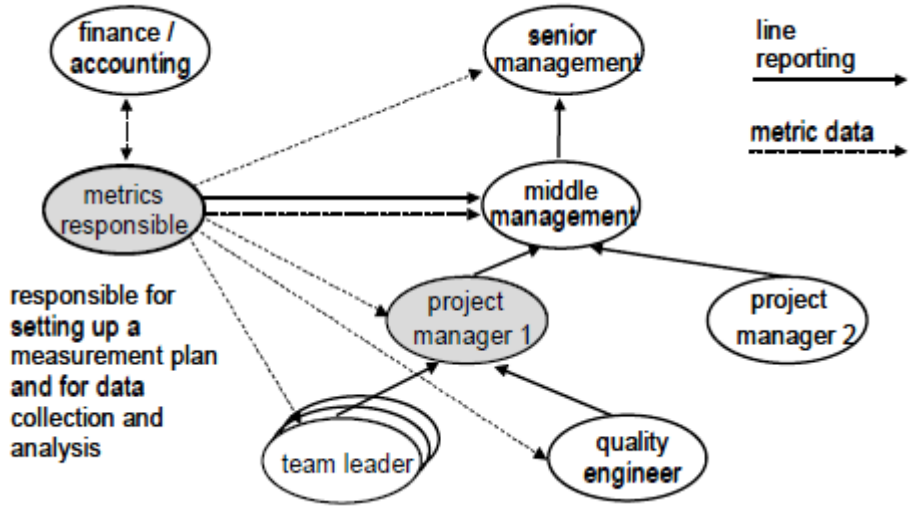


Ilustración 57 Como maneja el responsable de métricas la información (Ebert, Bundschuh, Dumke, & Schmietendorf, 2005)

```
> table(TFM_Dataset$`Business Component`,TFM_Dataset$`Influencing Sys`)

                EAI Fox Fox, VC JDE  VC Webmethods
  Administration    0   1       0   0   0          0
  Agent Desktop     0   0       0   0   0          0
  Claims            0  12       0   1   0          0
  Client Care       0   0       0   0   0          0
  Client File       0  82       0   0   0          0
  Despatch          0   5       0   0   0          0
  Diary             0  41       0   0   0          0
  Finance           0   5       0   0   0          0
  Follow up         0  28       0   0   0          0
  Front Office      0  19       0   0   1          0
  Maintenance       0   2       0   0   0          0
  Marketing         0   1       0   0   0          0
  Payroll & Bonus   0   1       0   0   0          0
  Reports           0  14       0   0   0          0
  Sales             0 189       0   2   0          0
  Segmentation      0   1       0   0   0          0
  Stock             0   9       0   0   0          0
  WebMethods        0   0       0   0   0          1
```

Este análisis es muy interesante, pero no deja de ser una constatación de Pareto, se debería favorecer la planificación de tareas con persistencia de algunas tareas destinadas a módulos y en caso de influenciar a un módulo, que esa persistencia fuera

compartida con el módulo influenciado. Pero si las métricas de líneas de código, complejidad y precio están bien hechas debería tener más significado, es deducible que si un módulo tiene más complejidad es porque sus entradas y salidas dependen de otros módulos y la integración es compleja. En la función objetivo analizo la complejidad del componente añadiendo y relajando el umbral de tiempo.

```
> table(TFM_Dataset$`Business Component`,TFM_Dataset$`Sys Components`)

                 Fox
  Administration   3
  Agent Desktop    1
  Claims         103
  Client Care      2
  Client File    374
  Despatch        24
  Diary          251
  Finance         61
  Follow up       42
  Front Office    58
  Maintenance      5
  Marketing        8
  Payroll & Bonus  6
  Reports         19
  Sales          717
  Segmentation     1
  Stock           85
  WebMethods       2
```

```
> table(TFM_Dataset$Priority,TFM_Dataset$`Business Component`)

             Administration Agent Desktop Claims Client Care Client File Despatch Diary Finance Follow up
  Low                     0             0      4           1          70        1    29       9         7
  Major                   3             0     78           1         121       18   100      33        25
  Medium                  0             1     13           0         166        1    87       6         8
  ShowStopper             0             0      8           0          17        4    35      13         2

             Front Office Maintenance Marketing Payroll & Bonus Reports Sales Segmentation Stock WebMethods
  Low                11           1         0              1       0   108            0    15          0
  Major              12           3         4              3      17   347            1    44          1
  Medium             33           0         4              0       0   180            0    18          1
  ShowStopper         2           1         0              2       2    82            0     8          0
```

```
> table(TFM_Dataset$Severity,TFM_Dataset$Priority)

           Low Major Medium ShowStopper
  Critical   0    60      2         175
  Major      0   767     29          25
  Medium    17    36    510           0
  Minor    260     2     15           0
```

Las categorías de severidad y de prioridad es una medida proporcional por lo que sólo se requiere una, en principio, al ser Sonar una herramienta fiable, se añadirá directamente a la estructura de datos del programa la variable PriorityFix. Ilustración 56Niveles de gravedad de Sonar. Como se puede apreciar en la clase class QualityIssue, de la librería QAPool.cpp

```
> table(TFM_Dataset_Total$LOC,TFM_Dataset_Total$`Critical issues`+TFM_Dataset_Total$`M
ajor issues`+TFM_Dataset_Total$`Minor issues`+TFM_Dataset_Total$`Info issues`)

        0 25 367 403 505 714 1299 2080 3252 6048 8035 20206 22515 65449
  0     5  0   0   0   0   0    0    0    0    0    0     0     0     0
  97    1  0   0   0   0   0    0    0    0    0    0     0     0     0
  219   0  1   0   0   0   0    0    0    0    0    0     0     0     0
  2256  0  0   1   0   0   0    0    0    0    0    0     0     0     0
  4008  0  0   0   1   0   0    0    0    0    0    0     0     0     0
  5258  0  0   0   0   1   0    0    0    0    0    0     0     0     0
  5556  0  0   0   0   0   1    0    0    0    0    0     0     0     0
  11528 0  0   0   0   0   0    1    0    0    0    0     0     0     0
  19006 0  0   0   0   0   0    0    1    0    0    0     0     0     0
  27821 0  0   0   0   0   0    0    0    1    0    0     0     0     0
  57625 0  0   0   0   0   0    0    0    0    0    1     0     0     0
  76517 0  0   0   0   0   0    0    0    0    1    0     0     0     0
  164952 0 0   0   0   0   0    0    0    0    0    0     1     0     0
  226059 0 0   0   0   0   0    0    0    0    0    0     0     1     0
  600902 0 0   0   0   0   0    0    0    0    0    0     0     0     1
>
```

En cuestión de Bugs, es perfectamente lineal en cuánto a las líneas de código presenta una proporcionalidad al número de bugs (en todas las categorías de errores)

Una excepción la presenta el componente Administración, pero por los demás componentes no hay problema.

4 El planificador evolutivo

4.1 Clases

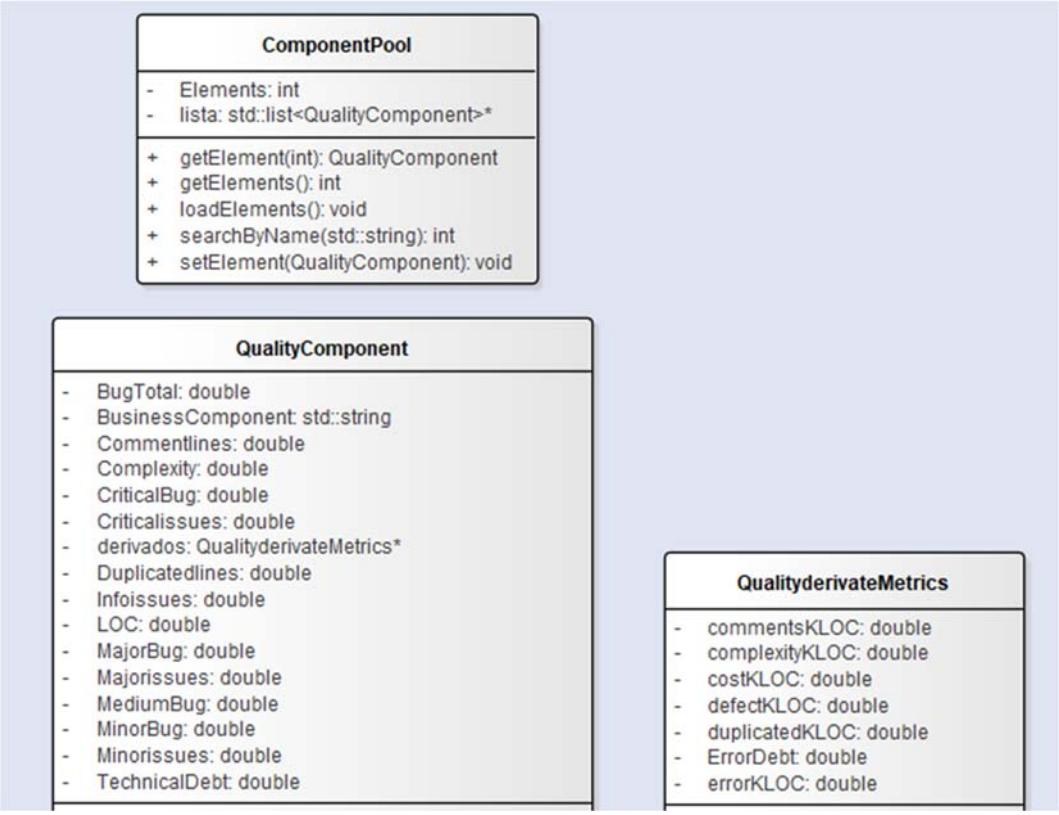
4.1.1 Clases Contenedor

Se compone de dos clases de almacenamiento, una para los componentes Software que componen el programa y otro para las incidencias que han reportado las distintas pruebas automatizadas.

La clase de Componentes consiste en una lista de Componentes y su número de componentes, Elements. La función de carga la realiza loadElements, el cual carga del fichero CSV, todas las líneas menos las cabeceras, transforma la información a un objeto llamado QualityComponent y la almacena en la lista.

La búsqueda sobre la lista de elementos almacenados se realiza con SearchByName que devuelve la posición del elemento a partir de su nombre.

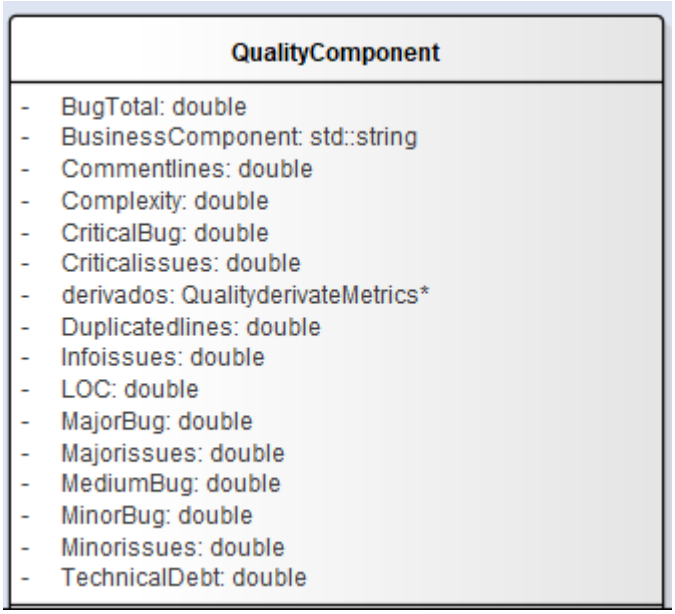
4.1.2 Clase ComponentPool



4.1.3 Clase QualityComponent

La clase QualityComponent es muy importante porque permite asociar la estrategia de negocio con la estrategia de planificación sobre los resultados de las pruebas. Para ellos también posee una clase agregada llamada QualityderivateMetrics, que realiza una traducción más o menos plausible gracias a LOC y a precio de los defectos para la lógica de negocio.

La clase se compone de información dividida conceptualmente en



1. Errores
   - Criticable.
   - MajorBug.
   - MediumBug.
   - MinorBug.

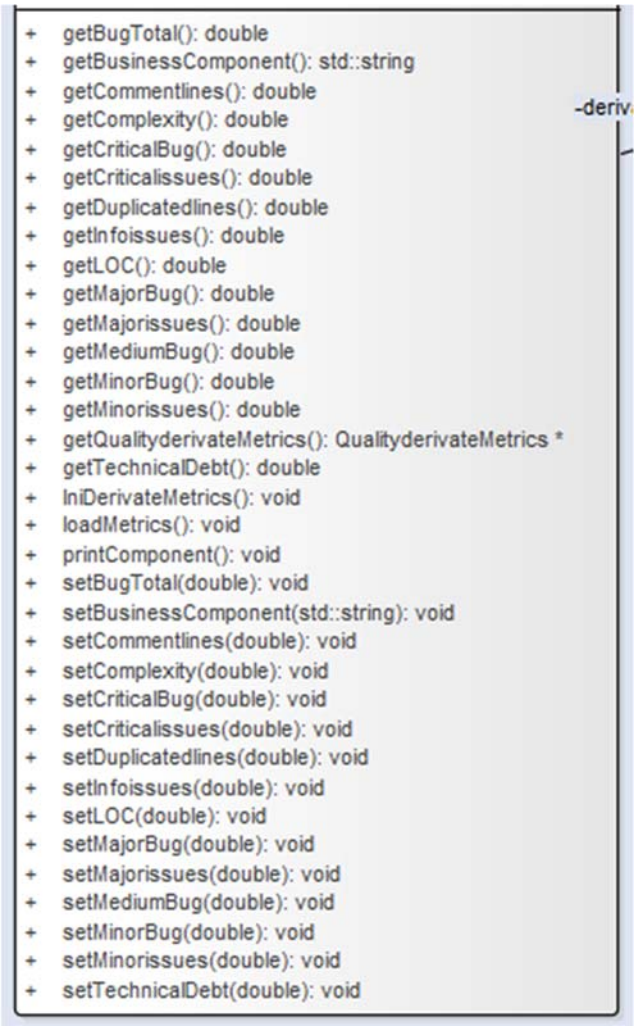
2. Incidencias
   - Criticalissues.
   - Majorissues.
   - Minorissues.
   - Infoissues.

3. Código
   - LOC.
   - Duplicatedlines.
   - Commentlines.

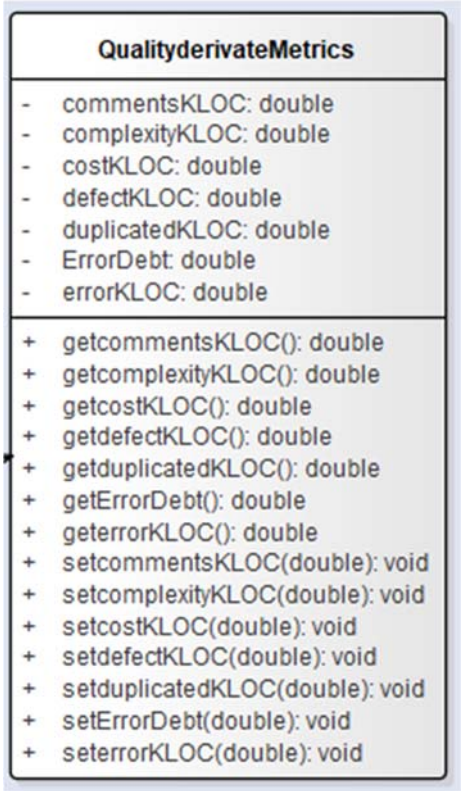
4. Complejidad.
   - Complexity.

5. Precio
   - TechnicalDebt.

Y las funciones del objeto, la clase derivada de QualityderivateMetrics permite mucha accesibilidad, puesto que todas demás clases podrían quedar encapsuladas y protegidas al programador de la línea de negocio. Como clases private.



4.1.4 Clase QualityderivateMetrics



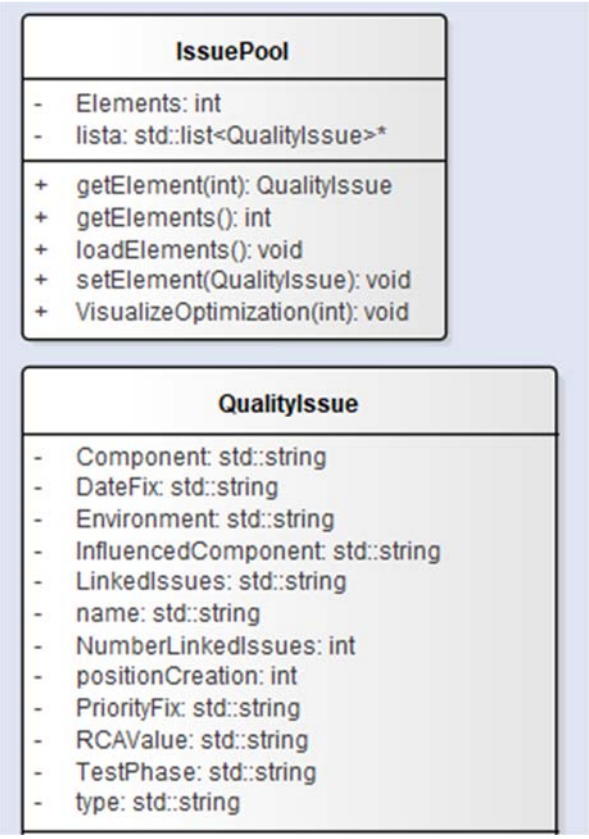
Al no ser estas medidas útiles por sí mismas, he favorecido la manipulación a través de una clase llamada QualityderivateMetrics, que realiza un seguimiento de las medidas mencionadas a través de LOC y del precio por defecto.

Está clase se crea al almacenar cada componente desde la creación del objeto QualityComponent, me ha parecido la mejor forma puesto que no se realiza una manipulación de la información fuera del contexto del objeto y puede ser interesante usar dicha información de cara a algún informe u otro planteamiento.

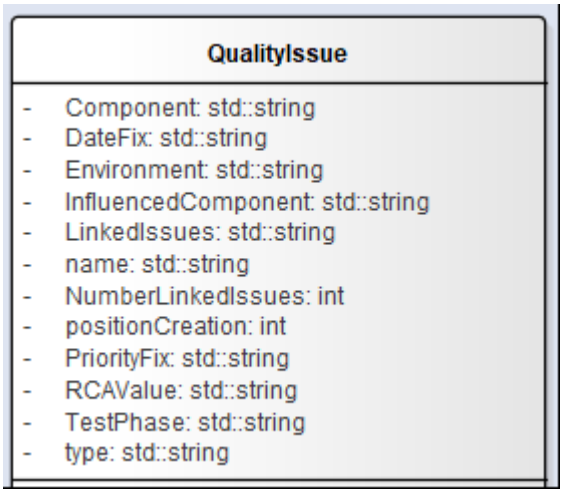
- defectKLOC.
- errorKLOC.
- costKLOC.
- commentsKLOC.
- duplicatedKLOC.
- ErrorDebt.
- complexityKLOC.

4.1.5 Clase IssuePool

Está clase permite la agrupación de las incidencias y su traducción a un número según el orden de llegada para su manipulación, al igual que la clase ComponentPool se realiza una carga de la información desde el fichero CSV, creando objetos de la clase QualityIssue, desde donde se trabajará, con el número de creación.



4.1.6 Clase QualityIssue



Está clase contiene información ya manipulada por un sistema de calidad en cuánto al orden de prioridad, necesidad de resolver la incidencia en un determinado tiempo no se debe derivar.

Además, debido a la falta de completitud de los datos de pruebas, en principio se trabaja con ella por sí misma, y se referencia sólo al nombre del componente que se podrá relacionar a través de la búsqueda del nombre.

Se compone del

De cara a referenciarlo

- Nombre, name;
- Tipo. Type

Necesarios para priorizar

- Fecha para cerrar la incidencia, DateFix
- Casos asociados, LinkedIssues
- Componente, Component
- Prioridad de cerrar la incidencia PriorityFix
- Componentes influenciados InfluencedComponent
- Número de casos asociados, NumberLinkedIssues.

Muy importantes, pero falta información, se puede hacer una agrupación menor de algunas incidencias de cara a la prueba que las origino.

- Caso de prueba TestPhase

- Valor RCA, RCAValue
- Entorno, Environment

Aunque el módulo de caso real, no se haya implementado una estrategia para reducir su complejidad puede ser demorar la planificación de una tarea según sus casos de incidencia asociados, pero trae la desventaja de que ante casos cíclicos como Reopened, la función objetivo pierda el sentido aplicando demora a casos que debería poner sin dudarlo en primera instancia.

4.1.7 UML de clases

4.2 Representación

Se usará la clase de Galib, GAListGenome, cada elemento en la lista tiene su elemento correspondiente según orden de llegadas de las incidencias que se almacenarán IssuePool *PopulationQAMetrics=new IssuePool();

La carga de los elementos se realizaría a través de

PopulationQAMetrics->loadElements()

4.3 Reproducción y mutación

Se usaría el cruce en un punto por defecto en galib, pero se ha usado PMX, incluido en uno de los ejemplos de Galib, ex 11.

Se usará un tipo de reproducción a través de las generaciones GASteadyStateGA, lo que significa que la población va siendo sustituida por su prole según los parámetros de que indiquemos, conviven distintas generaciones. Es una cosa muy positiva porque es una forma de "elitismo moderado" y de preservación de la diversidad, lo cual ayuda a encontrar los óptimos globales.

La mutación, sería el intercambio de dos elementos de la lista.

4.4 Función Objetivo

Dado que seguramente no sea un desarrollo en espiral o iterativo, dado que el número de desarrollos no es similar al de incidencias, en principio, se analiza lo coherente de las incidencias según el orden de llegada, en otro caso se deberá analizar el desarrollo de un módulo de coherencia para ver si incidencias relacionadas están en el orden correcto y crear una función de penalización a ese respecto.

IssuePool *PopulationQAMetrics=new IssuePool();

ComponentPool *PopulationQAComponent=new ComponentPool();

Va traduciendo según el entero que encuentre en GAListGenome, y utiliza las métricas para el elemento indicado de PopulationQAMetrics y PopulationQAComponent.

Recorre toda esa lista para ver si cada elemento se ajusta a una ventana de tiempo y aumentar la puntuación.

4.4.1 Ventana de tiempo TIME_SPLIT

Se crea una ventana de tiempo basada en los siguientes datos, donde las tareas según su creación tendrán que ajustarse a esa ventana creada a través de TIME_SPLIT

TIME_SPLIT

DATE_FIX se reducirá TIME_SPLIT a necesidad de terminar en una fecha

WINDOW_WORK_BASED_COMPLEXITY <- se reducirá a mayor complejidad, haciendo que la permisividad para terminar la tarea sea menor. Puesto que al ser más compleja necesita empezar antes la actividad.

- **ElementoComponent.getQualityderivateMetrics()->getcomplexityKLOC();**

Los factores de prioridad,

- **Elemento.getPriorityFix()**

FIX_HIGH 4

FIX_MEDIUM 2

FIX_LOW 1

Se puede variar para priorizar la corrección de otro tipo de casos.

4.4.2 #define MONEY_TALK 1

Crea la dimensión económica, se priorizará que la primera mitad de las incidencias cueste menos que la segunda mitad, en cuestión de calidad es lo mismo, y supone mejora de calidad sin pérdida de capital.

```
for(int i=1; i<genome.size(); i++){
```

```
    int valueCurrent = *genome.current();
    Elemento= matrixIssueQuality->getElement(valueCurrent);
    *genome.next();
  if(Elemento.getComponent()=="")
    continue;
    if(matrixComponentQuality->searchByName(Elemento.getComponent())>-
1)
    ElementoComponent=matrixComponentQuality-
>getElement(matrixComponentQuality-
>searchByName(Elemento.getComponent()));

    if(i<genome.size()/2){
     firsthalf+=ElementoComponent.getQualityderivateMetrics()-
>getErrorDebt();
    }else{
     secondhalf+=ElementoComponent.getQualityderivateMetrics()-
>getErrorDebt();
    }

}
```

4.4.3 #define TEST_PHASE 1

Muy interesante para el modelo en V, también para el modelo iterativo, permite crear la lógica de las ventanas de tiempo basándose en que prueba se ha ejecutado sobre el Software para producir las incidencias.

```
if(Elemento.getTestPhase().rfind("SIT C1")!=std::string::npos)
        premureWork=1200;
    else if(Elemento.getTestPhase().rfind("ST C2")!=std::string::npos)
            premureWork=400;
      else if(Elemento.getTestPhase().rfind("ST
C3")!=std::string::npos)
            premureWork=600;
              else if(Elemento.getTestPhase().rfind("ST
C1")!=std::string::npos)
            fix_tempo=200;
     else
```

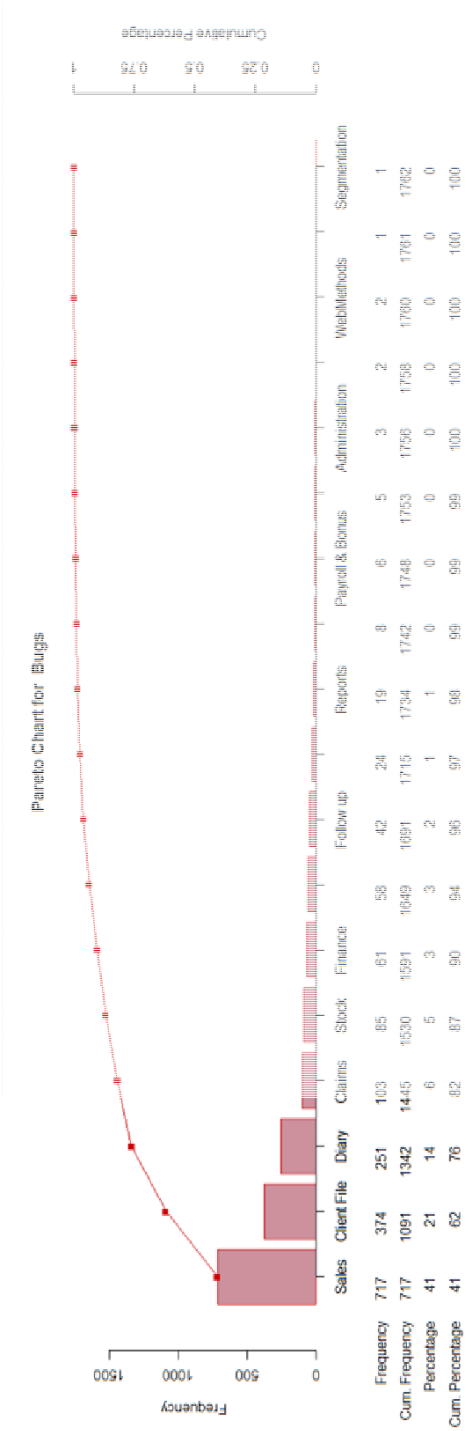
Está parte es muy interesante a nivel de un desarrollo ágil, iterativo, espiral o prototipo, es una priorización total.

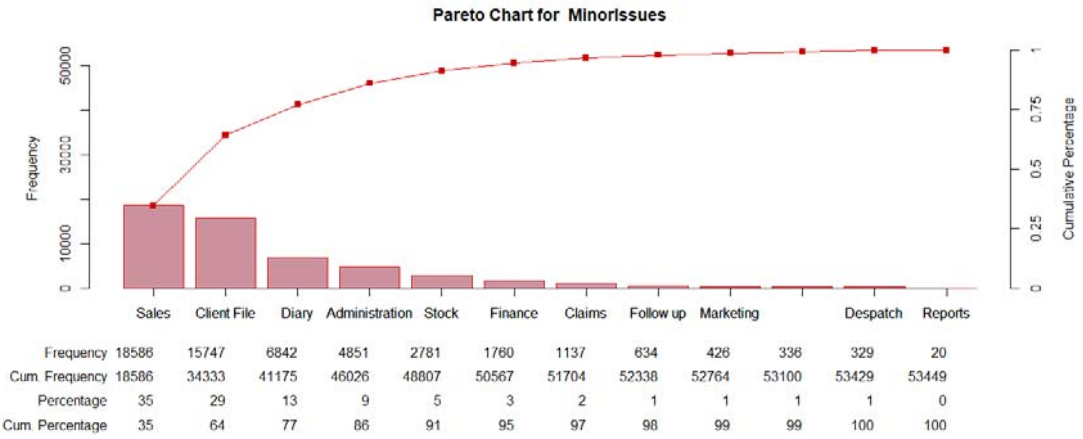
```
if(Elemento.getTestPhase().rfind("Functional")!=std::string::npos)
            fix_tempo=0;
     else
if(Elemento.getTestPhase().rfind("Design")!=std::string::npos)
            fix_tempo=0;
             else
if(Elemento.getTestPhase().rfind("Development")!=std::string::npos)
```

5 Anexo

5.1 Gráficas de Pareto

Los casos serán mejores cuanto más se parezcan a estas gráficas, corresponde al jefe de proyecto ajustar según crea necesario.





Frequency	18586	15747	6842	4851	2781	1760	1137	634	426	336	329	20
Cum Frequency	18586	34333	41175	46026	48807	50567	51704	52338	52764	53100	53429	53449
Percentage	35	29	13	9	5	3	2	1	1	1	1	0
Cum Percentage	35	64	77	86	91	95	97	98	99	99	100	100

(Roth, 2016) (Cano, Moguerza, & Prieto Corcoba, 2015)

5.1.1 Genración de CSV, métricas.linq

Este script hecho en linqpad permite la creación de los ficheros csv que se hayan en el Directorio QualityData

```
List<Myspacename.QualityMetrics> QualityMetrica=new
List<Myspacename.QualityMetrics>{};
List<Myspacename.QualityComponent> QualityComponent=new
List<Myspacename.QualityComponent>{};

// Revisar para los casos de reopen

IQueryable<Myspacename.QualityMetrics>  Lista= (from data in JiraBugs
 orderby data.Created
select new Myspacename.QualityMetrics(
data.Key,
data.IssueType,
data.FixPriority,
data.TestPhase,
data.FixbyDate.ToString(),
data.LinkedIssues,
data.BusinessComponent,
data.InfluencingSys,
data.RCAValue,
data.Environment
));

 QualityMetrica=Lista.ToList();

      //QualityMetrica.Dump();
IQueryable<Myspacename.QualityComponent> ListaQualityMetrics= (from
data in Resumes

select new Myspacename.QualityComponent(
data.BusinessComponent,
(double)data.BugTotal,
```

```
(double)data.CriticalBug,
(double)data.MajorBug,
(double)data.MediumBug,
(double)data.MinorBug,
(double)data.LOC,
(double)data.Complexity,
(double)data.Duplicatedlines,
(double)data.Commentlines,
(double)data.Criticalissues,
(double)data.Majorissues,
(double)data.Minorissues,
(double)data.Infoissues,
(double)data.TechnicalDebt
));
QualityComponent=ListaQualityMetrics.ToList();
QualityComponent.Dump();
var ListaDerivatedQualityMetrics= (from data in ListaQualityMetrics

select new {

});
QualityMetrica.Dump();
QualityComponent.Dump();

Util.WriteCsv (QualityMetrica, "Quality1.csv");
Util.WriteCsv (QualityComponent, "Quality2.csv");

}

public static class  Myspacename{
public class QualityMetrics
{
   public string IssueName {get;set;}
   public string IssueType {get;set;}
   public string FixPriority {get;set;}
   public string TestPhase {get;set;}
   public string TiempoRestriccion {get;set;}
   public string LinkedIssues {get;set;}
   public string BusinessComponent {get;set;}
   public string InfluenciedSys {get;set;}
   public string RCAValue {get;set;}
   public string Environment {get;set;}
   public QualityMetrics(string uno, string dos, string tres, string
cuatro, string cinco,string seis, string siete,string ocho,string
nueve,string diez){
   IssueName=uno;
   IssueType=dos;
    FixPriority=tres;
      TestPhase=cuatro;
   TiempoRestriccion=cinco;
   LinkedIssues=seis;
   BusinessComponent=siete;
   InfluenciedSys=ocho;
   RCAValue=nueve;
   Environment=diez;
   }
}
// Se podrï¿½a sacer una mï¿½trica derivada de estos atributos, puesto
que no tiene porque ser necesario, es decir el tiempo necesario
segï¿½n la complejidad y LOC, restando los comentarios.
```

```
public class QualityComponent{
      public string BusinessComponentName {get;set;}
      public double BugTotal {get;set;}
      public double CriticalBug {get;set;}
      public double MajorBug {get;set;}
      public double MediumBug {get;set;}
      public double MinorBug {get;set;}
      public double LOC {get;set;}
      public double Complex {get;set;}
      public double DuplicatedLines {get;set;}
      public double CommentLines{get;set;}
      public double CriticalIssue {get;set;}
      public double MajorIssue {get;set;}
      public double MinorIssue {get;set;}
      public double InfoIssue {get;set;}
      public double Deuda {get;set;}
public QualityComponent(string uno, double dos, double tres, double
cuatro, double cinco, double seis, double siete, double ocho, double
nueve, double diez, double once, double doce, double trece, double
catorce, double quince){
      BusinessComponentName=uno;
      BugTotal=dos;
      CriticalBug=tres;
      MajorBug=cuatro;
      MediumBug=cinco;
      MinorBug=seis;
      LOC=siete;
      Complex=ocho;
      DuplicatedLines=nueve;
      CommentLines=diez;
      CriticalIssue=once;
      MajorIssue=doce;
      MinorIssue=trece;
      InfoIssue=catorce;
      Deuda=quince;
   }

}
```

5.1.2 Generación de CSV, LinkedIssues NormalizeTablesTask.linq

NormalizeTablesTask.linq

Este script hecho en linqpad permite la creación de los ficheros csv que se hayan en el Directorio QualityData, sobre el análisis de LinkedIssues.

```
List<Myspacename.RecordLinkedIssues> Records=new
List<Myspacename.RecordLinkedIssues>{};
List<Myspacename.RecordLinkedIssues> RecordsTime=new
List<Myspacename.RecordLinkedIssues>{};
List<string> RecordsDefun=new List<string>{};
```

```
string busquedaAproxIssue;
DateTime testCalif;

var Lista= (from data in JiraBugs
where data.LinkedIssues!=null
select new {
clave = data.Key,
clave2 = data.LinkedIssues,
clave3 = (DateTime)data.Created
});
//var Lestado=(from ListadoIssues in JiraBugs where
var Lestado=(from data in JiraBugs
select new { clave = data.Key, Created = data.Created});
            //ListadoIssues.Key==items).SingleOrDefault();
foreach (var item in Lista){
//item.Dump(item.clave3.ToString());
string[] values = item.clave2.Split(',').Select(sValue =>
sValue.Trim()).ToArray();
//if(values.Count()==1)
 //values = item.clave2.Split(",".ToCharArray(),
StringSplitOptions.RemoveEmptyEntries);
if(values.Count()>1){

      foreach(var items in values){

      //DEBUG items.Dump();
// items.Dump();

      if((from ListadoIssues in Lestado where
            ListadoIssues.clave==items select
ListadoIssues.Created).Any()&&items.Count()<8){
      testCalif=(DateTime)(from ListadoIssues in Lestado where
            ListadoIssues.clave==items select
ListadoIssues.Created).FirstOrDefault();
             RecordsTime.Add(new
Myspacename.RecordLinkedIssues(item.clave,items,item.clave3,testCalif)
);
            }else{
            busquedaAproxIssue=items.Substring(0,6);
busquedaAproxIssue.Dump();
            if((from ListadoIssues in Lestado where
ListadoIssues.clave.StartsWith(busquedaAproxIssue) select
ListadoIssues.Created).FirstOrDefault().ToString().Any()){
                        testCalif=(DateTime)(from ListadoIssues in
Lestado where ListadoIssues.clave.StartsWith(busquedaAproxIssue)
select ListadoIssues.Created).FirstOrDefault() ;
             RecordsTime.Add(new
Myspacename.RecordLinkedIssues(item.clave,items,item.clave3,testCalif)
);
             }else{
            RecordsDefun.Add(items);
      }
            }
       Records.Add(new
Myspacename.RecordLinkedIssues(item.clave,items));
   // testCalif=(from joplines in Lista where joplines.clave==items
select joplines.clave3).SingleOrDefault();;
```

```
       //RecordsTime.Add(new
Myspacename.RecordLinkedIssues(item.clave,items,item.clave3,testCalif)
);
      //ComplexIssues(new
Myspacename.RecordLinkedIssues(item.clave,items));

      }
}else{
Records.Add(new
Myspacename.RecordLinkedIssues(item.clave,item.clave2));
if((from ListadoIssues in Lestado where
            ListadoIssues.clave==item.clave2 select
ListadoIssues.Created).Any()){
            testCalif=(DateTime)(from ListadoIssues in Lestado where
            ListadoIssues.clave==item.clave2 select
ListadoIssues.Created).SingleOrDefault();
             RecordsTime.Add(new
Myspacename.RecordLinkedIssues(item.clave,item.clave2,item.clave3,test
Calif));
      }else{
      busquedaAproxIssue=item.clave2.Substring(0,6);
      busquedaAproxIssue.Dump();
      if((from ListadoIssues in Lestado where
            ListadoIssues.clave.StartsWith(busquedaAproxIssue) select
ListadoIssues.Created).FirstOrDefault().ToString().Any()){

            testCalif=(DateTime)(from ListadoIssues in Lestado where
            ListadoIssues.clave.StartsWith(busquedaAproxIssue) select
ListadoIssues.Created).FirstOrDefault();

             RecordsTime.Add(new
Myspacename.RecordLinkedIssues(item.clave,item.clave2,item.clave3,test
Calif));
             }else{
RecordsDefun.Add(item.clave2);
}
      }


}
//Records.Dump();
}
Util.WriteCsv (Records, "out.csv");
Util.WriteCsv (RecordsTime, "outTime.csv");
Util.WriteCsv (RecordsDefun,"DefunKey.csv");
}

public static class  Myspacename{
public class RecordLinkedIssues
{
   public string Issue {get;set;}
   public string LinkedIssue {get;set;}
   public DateTime Tiempo {get;set;}
   public DateTime TiempoAjeno{get;set;}
   public RecordLinkedIssues(string uno, string dos){
   Issue=uno;
   LinkedIssue=dos;
   }
   public RecordLinkedIssues(string uno, string dos, DateTime
tres,DateTime cuatro){
   Issue=uno;
```

```
    LinkedIssue=dos;
    Tiempo=tres;
    TiempoAjeno=cuatro;
    }
}
```

5.1.3 QAPool.h

```
/*****************************************************************************
*                      TFM CALEVO, Universidad de Alcalá 2017
*
*
*
* Librería Planificador Genético basado en la libería GALIB desarrollada por Mathew Wall
http://lancet.mit.edu/ga/     *
*   linkedin@lancet.mit.edu
*
*    Director: Luis Fernández Sanz luis.fernandezs@uah.es
*
*    Autor : Javier Ruano Ruano  javier.ruano@edu.uah.es
*
*
*
*                                                  11/07/2017
*
*****************************************************************************/

#ifndef QAPOOL_H
#define QAPOOL_H

#include <iostream>
#include <sstream>
#include <fstream>
#include <cstdlib>
#include <list>
#include <iterator>
#include <vector>
#include <stdlib.h>
#include <string.h>
#define MAX_ISSUES 255

class QualityderivateMetrics;
class ComponentPool;

class QualityComponent{
private:

   std::string BusinessComponent;
   double BugTotal;
    double CriticalBug;
    double MajorBug;
    double MediumBug;
    double MinorBug;
    double LOC;
    double  Complexity;
double Duplicatedlines;
double Commentlines;
double Criticalissues;
double Majorissues;
double Minorissues;
double Infoissues;
double TechnicalDebt;
QualityderivateMetrics *derivados;
public:
    void loadMetrics();
    std::string getBusinessComponent();
    double getBugTotal();
    double getCriticalBug();
    double getMajorBug();
    double getMediumBug();
    double getMinorBug();
```

```
    double getLOC();
    double  getComplexity();
double getDuplicatedlines();
double getCommentlines();
double getCriticalissues();
double getMajorissues();
double getMinorissues();
double getInfoissues();
double getTechnicalDebt();


    void setBusinessComponent(std::string name);
   void setBugTotal(double bugs);
    void setCriticalBug(double bugs);
    void setMajorBug(double bugs);
    void setMediumBug(double bugs);
    void setMinorBug(double bugs);
    void setLOC(double bugs);
    void  setComplexity(double bugs);
void setDuplicatedlines(double bugs);
void setCommentlines(double bugs);
void setCriticalissues(double bugs);
void setMajorissues(double bugs);
void setMinorissues(double bugs);
void setInfoissues(double bugs);
void setTechnicalDebt(double bugs);
void printComponent();
void IniDerivateMetrics();
QualityderivateMetrics *getQualityderivateMetrics();
};


class ComponentPool{
  std::list<QualityComponent> *lista;

    int Elements;
    public:
  //  IssuePool();
    QualityComponent getElement(int n);
    void setElement(QualityComponent Element);
    int getElements();
    void loadElements();
    int searchByName(std::string Name);

};

class QualityderivateMetrics{
double defectKLOC;
double errorKLOC;
double costKLOC;
double commentsKLOC;
double duplicatedKLOC;
double ErrorDebt;
double complexityKLOC;
public:

double getdefectKLOC();
double geterrorKLOC();
double getcostKLOC();
double getcommentsKLOC();
double getduplicatedKLOC();
```

```
double getErrorDebt();
double getcomplexityKLOC();
void setdefectKLOC(double value);
void seterrorKLOC(double value);
void setcostKLOC(double value);
void setcommentsKLOC(double value);
void setduplicatedKLOC(double value);
void setErrorDebt(double value);
void setcomplexityKLOC(double value);

};

class QualityIssue{
    private:
     int positionCreation;
    std::string name;
    std::string type;
   std::string DateFix;
    std::string LinkedIssues;
    std::string Component;
    std::string PriorityFix;
    std::string InfluencedComponent;
    std::string TestPhase;
    std::string RCAValue;
    std::string Environment;
    int NumberLinkedIssues;

    public:
    void setpositionCreation(int numberLocation);
    void setName(std::string Nam);
    void setType(std::string Typ);
    void setDateFix(std::string DateFi);
    void setLinkedIssues(std::string Linkedissue);
    void setComponent(std::string Componen);
    void setPriorityFix(std::string PriorityFi);
    void setInfluencedComponent(std::string InfluencedComponen);
    void setTestPhase(std::string TestPhas);
    void setRCAValue(std::string RCAValue);
    void setEnvironment(std::string Environ);

    int getpositionCreation();

    std::string getName();
    std::string getType();
    std::string getDateFix();
    std::string getLinkedIssues();
    std::string getComponent();
    std::string getPriorityFix();
    std::string getInfluencedComponent();
    std::string getTestPhase();
    std::string getRCAValue();
    std::string getEnvironment();

    int getNumberLinkedIssues();
    void countLinkedIssues();

    void printIssue();

};
```

```
class IssuePool{
private:
    std::list<QualityIssue> *lista;
    int Elements;
    public:
  //  IssuePool();
    QualityIssue getElement(int n);
    void setElement(QualityIssue Element);
    int getElements();
    void loadElements();
    void VisualizeOptimization(int dataISSUE);
};

#endif
```

5.1.4 QAPool.cpp

```
/*********************************************************************************
*                      TFM CALEVO, Universidad de Alcalá 2017
*
*
*
* Librería Planificador Genético basado en la libería GALIB desarrollada por Mathew Wall
http://lancet.mit.edu/ga/     *
*   linkedin@lancet.mit.edu
*
*    Director: Luis Fernández Sanz luis.fernandezs@uah.es
*
*    Autor : Javier Ruano Ruano  javier.ruano@edu.uah.es
*
*
*
*                                                  11/07/2017
*
*********************************************************************************/

#include "QAPool.h"
void QualityComponent::IniDerivateMetrics(){
    this->derivados=new QualityderivateMetrics();

         this->derivados->seterrorKLOC(0);
    this->derivados->setcostKLOC(0);
    this->derivados->setdefectKLOC(0);
    this->derivados->setduplicatedKLOC(0);
    this->derivados->setcommentsKLOC(0);
    this->derivados->setcomplexityKLOC(0);

    this->derivados->setErrorDebt(0);

    if(this->getBugTotal()!=0)
    this->derivados->setErrorDebt(this->getTechnicalDebt()/this-
>getBugTotal());
 if(this->getLOC()!=0){
 double KLOC=this->getLOC()/1000;
 this->derivados->seterrorKLOC(this->getBugTotal()/KLOC);
this->derivados->setcostKLOC(this->getTechnicalDebt()/KLOC);
 this->derivados->setdefectKLOC((this->getMajorissues()+this-
>getMinorissues()+this->getInfoissues())/KLOC);
```

```
 this->derivados->setduplicatedKLOC((this-
>getDuplicatedlines()/1000)/KLOC);
 this->derivados->setcommentsKLOC((this-
>getCommentlines()/1000)/KLOC);
 this->derivados->setcomplexityKLOC((this-
>getComplexity()/1000)/KLOC);
 }


}

QualityderivateMetrics *QualityComponent::getQualityderivateMetrics(){
return this->derivados;
}
void   QualityComponent::setBusinessComponent(std::string name){
this->BusinessComponent=name;
}
void QualityComponent::setBugTotal(double bugs){
    this->BugTotal=bugs;
}
    void QualityComponent::setCriticalBug(double bugs){
    this->CriticalBug=bugs;

    }
    void QualityComponent::setMajorBug(double bugs){
    this->MajorBug=bugs;
    }
    void QualityComponent::setMediumBug(double bugs){
        this->MediumBug=bugs;
    }
    void QualityComponent::setMinorBug(double bugs){
        this->MinorBug=bugs;
    }
    void QualityComponent::setLOC(double bugs){
        this->LOC=bugs;
    }
    void  QualityComponent::setComplexity(double bugs){
        this->Complexity=bugs;

    }
void QualityComponent::setDuplicatedlines(double bugs){
this->Duplicatedlines=bugs;
}
void QualityComponent::setCommentlines(double bugs){
this->Commentlines=bugs;
}
void QualityComponent::setCriticalissues(double bugs){
this->Criticalissues=bugs;
}
void QualityComponent::setMajorissues(double bugs){
this->Majorissues=bugs;
}
void QualityComponent::setMinorissues(double bugs){
this->Minorissues=bugs;
}
void QualityComponent::setInfoissues(double bugs){
this->Infoissues=bugs;
}
void QualityComponent::setTechnicalDebt(double coins){
this->TechnicalDebt=coins;
}
```

```
std::string QualityComponent::getBusinessComponent(){
    return this->BusinessComponent;
}
    double QualityComponent::getBugTotal(){
    return this->BugTotal;
    }
    double QualityComponent::getCriticalBug(){
    return this->CriticalBug;
    }
    double QualityComponent::getMajorBug(){
    return this->MajorBug;
    }
    double QualityComponent::getMediumBug(){
    return this->MediumBug;
    }
    double QualityComponent::getMinorBug(){
    return this->MinorBug;
    }
    double QualityComponent::getLOC(){
    return this->LOC;
    }
    double QualityComponent::getComplexity(){
    return this->Complexity;
    }
double QualityComponent::getDuplicatedlines(){
    return this->Duplicatedlines;
}
double QualityComponent::getCommentlines(){
    return this->Commentlines;
}
double QualityComponent::getCriticalissues(){
    return this->Criticalissues;
}
double QualityComponent::getMajorissues(){
    return this->Majorissues;
}
double QualityComponent::getMinorissues(){
    return this->MinorBug;
}
double QualityComponent::getInfoissues(){
    return this->Infoissues;
}
double QualityComponent::getTechnicalDebt(){
    return this->TechnicalDebt;
}
int  ComponentPool::getElements(){
return this->Elements;
}

void ComponentPool::loadElements(){
 std::string linea,linea_aux;
 std::ifstream file ( "QualityData\\Quality2.csv" );

    std::string token;
        this->lista=new std::list<QualityComponent>();
    this->Elements=0;
    QualityComponent *ComponentsMetrics=new QualityComponent();
    //Elimino cabeceras;
getline (file,linea);
    int params=0,pos=0,end=0 ;
```

```
  while ( file.good()&& !file.eof())
  {
       getline (file,linea);
linea_aux=linea;
  for(params=0;params<15;params++){
     end = linea_aux.find(',');
      pos=0;
token = linea_aux.substr(pos,end);

  if(params==0){
      ComponentsMetrics->setBusinessComponent(token);
  }

  else if(params==1){
   ComponentsMetrics->setBugTotal(atol(token.c_str()));
  }

  else if(params==2){
      ComponentsMetrics->setCriticalBug(atol(token.c_str()));
  }

  else if(params==3){
   ComponentsMetrics->setMajorBug(atol(token.c_str()));
  }

  else if(params==4){
      ComponentsMetrics->setMediumBug(atol(token.c_str()));
  }
  else if(params==5){
   ComponentsMetrics->setMinorBug(atol(token.c_str()));
  }

  else if(params==6){
      ComponentsMetrics-> setLOC(atol(token.c_str()));
  }

  else if(params==7){
   ComponentsMetrics->setComplexity(atol(token.c_str()));
  }

  else if(params==8){
      ComponentsMetrics->setDuplicatedlines(atol(token.c_str()));
  }

  else if(params==9){
   ComponentsMetrics->setCommentlines(atol(token.c_str()));
  }

  else if(params==10){
      ComponentsMetrics->setCriticalissues(atol(token.c_str()));
  }

  else if(params==11){
   ComponentsMetrics->setMajorissues(atol(token.c_str()));
  }

  else if(params==12){
      ComponentsMetrics->setMinorissues(atol(token.c_str()));
  }

  else if(params==13){
```

```
     ComponentsMetrics-> setInfoissues(atol(token.c_str()));
    }

    else if(params==14){
        ComponentsMetrics->setTechnicalDebt(atol(token.c_str()));
    }

     pos=end+1;

    linea_aux=linea_aux.substr(pos,linea_aux.length());

    }
    ComponentsMetrics->IniDerivateMetrics();
     lista->push_back(*ComponentsMetrics);
this->Elements++;
}


}

QualityComponent ComponentPool::getElement(int n){
    std::list<QualityComponent>::iterator fastForward = lista-
>begin();
    std::advance(fastForward, n);
return *fastForward;
}
void QualityComponent::printComponent(){

std::cout << "Nombre " << BusinessComponent << "\n" << "Lineas " <<
LOC << std::endl;

}


void IssuePool::loadElements(){
    this->lista=new std::list<QualityIssue>();
    this->Elements=0;
    QualityIssue *IssueReport=new QualityIssue();
 std::string linea,linea_aux;
    int params=0,pos=0,end=0;
    std::ifstream file ( "QualityData\\Quality1.csv" );

    std::string token;
    //Elimino cabeceras
    getline (file,linea);
    while ( file.good()&& !file.eof())
    {

    getline (file,linea);
    linea_aux=linea;
    for(params=0;params<10;params++){

       end = linea_aux.find(',');
        pos=0;

     if(params==5&&end>7){

        end = linea_aux.find('"',1)+1;

        }
            token = linea_aux.substr(pos,end);
```

```
    if(params==0){
        IssueReport->setName(token);
    }
    else if(params==1){
     IssueReport->setType(token);
    }
        else if(params==2){
    IssueReport->setPriorityFix(token);
    }

    else if(params==3){
    IssueReport->setTestPhase(token);
    }
    else if(params==4){
    IssueReport->setDateFix(token);

    }
    else if(params==5){
    IssueReport->setLinkedIssues(token);
    }
    else if(params==6){


    IssueReport->setComponent(token);
    }
    else if(params==7){
          IssueReport->setInfluencedComponent(token);
    }
     else if(params==8){
          IssueReport->setRCAValue(token);
    }
     else if(params==9){
     IssueReport->setEnvironment(token);
    }
    pos=end+1;

    linea_aux=linea_aux.substr(pos,linea_aux.length());

    }
    IssueReport->setpositionCreation(this->Elements);
    IssueReport->countLinkedIssues();
    lista->push_back(*IssueReport);

    this->Elements++;
    }
}




int  ComponentPool::searchByName(std::string Name){
int i=0;
for (std::list<QualityComponent>::iterator watchflow = lista->begin();
watchflow != lista->end(); ++watchflow){
        i++;
     std::size_t
found=(*watchflow).getBusinessComponent().rfind(Name);
        if (found!=std::string::npos)
            return i;
}
```

```
return -1;
}


QualityIssue IssuePool::getElement(int n){
    std::list<QualityIssue>::iterator fastForward = lista->begin();
    std::advance(fastForward, n);
return *fastForward;
}


int IssuePool::getElements(){
 return this->Elements;
}


 void IssuePool::VisualizeOptimization(int DataISSUE){



        std::cout << this->getElement(DataISSUE).getName() << ", " <<
this->getElement(DataISSUE).getComponent() << ", " << this-
>getElement(DataISSUE).getPriorityFix() << std::endl;


     }

 void  QualityIssue::setpositionCreation(int numberLocation){
 this->positionCreation=numberLocation;
    }
    void  QualityIssue::setName(std::string Nam){
        this->name=Nam;
    }
    void  QualityIssue::setType(std::string Typ){
    this->type=Typ;
    }
    void  QualityIssue::setDateFix(std::string DateFi){
        this->DateFix=DateFi;
    }
    void  QualityIssue::setLinkedIssues(std::string Linkedissue){
        this->LinkedIssues=Linkedissue;
    }
    void  QualityIssue::setComponent(std::string Componen){
        this->Component=Componen;
    }
    void  QualityIssue::setPriorityFix(std::string PriorityFi){
        this->PriorityFix=PriorityFi;
    }
    void  QualityIssue::setInfluencedComponent(std::string
InfluencedComponen){
    this->InfluencedComponent=InfluencedComponen;
    }
    void  QualityIssue::setTestPhase(std::string TestPhas){
        this->TestPhase=TestPhas;
    }
  void QualityIssue::setRCAValue(std::string RCAValue){
  this->RCAValue=RCAValue;
  }
    void QualityIssue::setEnvironment(std::string Environ){
    this->Environment=Environ;
    }
```

```
    void QualityIssue::countLinkedIssues(){
    int count=0;
    char LinkedIssue[MAX_ISSUES];
    strncpy(LinkedIssue,this->LinkedIssues.c_str(),MAX_ISSUES);
    for(int i=0;i<MAX_ISSUES-1;i++){
        if(LinkedIssue[i]=='N'&&LinkedIssue[i+1]=='P')
            count+=1;

    }
    this->NumberLinkedIssues=count;

    }
     int QualityIssue::getpositionCreation(){
        return this->positionCreation;
     }
    std::string  QualityIssue::getName(){
    return this->name;
    }
    std::string  QualityIssue::getType(){
    return this->type;
    }
    std::string  QualityIssue::getDateFix(){
    return this->DateFix;
    }
    std::string  QualityIssue::getLinkedIssues(){
    return this->LinkedIssues;
    }
    std::string  QualityIssue::getComponent(){
    return this->Component;
    }
    std::string  QualityIssue::getPriorityFix(){
    return this->PriorityFix;
    }
    std::string  QualityIssue::getInfluencedComponent(){
    return this->InfluencedComponent;
    }
    std::string  QualityIssue::getTestPhase(){
    return this->TestPhase;
    }
     std::string  QualityIssue::getRCAValue(){
     return this->RCAValue;
     }
     std::string  QualityIssue::getEnvironment(){
     return this->Environment;
     }
     int QualityIssue::getNumberLinkedIssues(){
     return this->NumberLinkedIssues;
     }
    void QualityIssue::printIssue(){
        std::cout << "NOMBRE" << name << "\n" << "TIPO" << type <<
std::endl;
    }


  double QualityderivateMetrics::getcomplexityKLOC(){
  return complexityKLOC;
  }

  double QualityderivateMetrics::getdefectKLOC(){
  return defectKLOC;
  }
```

```
double QualityderivateMetrics::geterrorKLOC(){
return errorKLOC;

}
double QualityderivateMetrics::getcostKLOC(){
return costKLOC;

}
double QualityderivateMetrics::getcommentsKLOC(){
return commentsKLOC;
}
double QualityderivateMetrics::getduplicatedKLOC(){
return duplicatedKLOC;

}
double QualityderivateMetrics::getErrorDebt(){
return ErrorDebt;
}
void QualityderivateMetrics:: setErrorDebt(double value){
this->ErrorDebt=value;
}

void QualityderivateMetrics::setcomplexityKLOC(double value){
this->complexityKLOC=value;
}
void QualityderivateMetrics::setdefectKLOC(double value){
this->defectKLOC=value;
}
void QualityderivateMetrics::seterrorKLOC(double value){
this->errorKLOC=value;

}
void QualityderivateMetrics::setcostKLOC(double value){
this->costKLOC=value;

}
void QualityderivateMetrics::setcommentsKLOC(double value){
this->commentsKLOC=value;
}
void QualityderivateMetrics::setduplicatedKLOC(double value){
this->duplicatedKLOC=value;

}
```

5.1.5 Main

```
/*****************************************************************************
*                      TFM CALEVO, Universidad de Alcalá 2017
*
*
*
* Librería Planificador Genético basado en la libería GALIB desarrollada por Mathew Wall
http://lancet.mit.edu/ga/     *
*   linkedin@lancet.mit.edu
*
*    Director: Luis Fernández Sanz luis.fernandezs@uah.es
*
*    Autor : Javier Ruano Ruano  javier.ruano@edu.uah.es
*
*
*
*                                                11/07/2017
*
*****************************************************************************/
```

```
#include <iostream>
#include <fstream>
#include <typeinfo>
#include <list>
#include <iterator>
#include <vector>
#include <stdlib.h>
#include "library/QAPool.h"
#include <math.h>
#include "ga/GASStateGA.h"
#include <ga/ga.h>
#include <ga/std_stream.h>

#define cout STD_COUT
#define ostream STD_OSTREAM


//#define MAX_ISSUES 50 Posible mejora ventanas de prueba
#define TIME_SPLIT 80
#define DATE_FIX 30

#define WINDOW_WORK_BASED_COMPLEXITY 50
#define DEFAULT_COMPLEX 1

#define FIX_HIGH    4
#define FIX_MEDIUM  2
#define FIX_LOW     1

#define MONEY_TALK 1
#define TEST_PHASE 1

IssuePool *matrixIssueQuality;
ComponentPool *matrixComponentQuality;
int nissues = 0;

using namespace::std;
#define XOVER PMXover
int   PMXover(const GAGenome&, const GAGenome&, GAGenome*, GAGenome*);
float objective(GAGenome &);
void ListInitializer(GAGenome &);
void VisualizePhenotype(GAGenome & c);

int main(int argc, char *argv[])
{


    QualityIssue data;
    QualityComponent data2;

     IssuePool *PopulationQAMetrics=new IssuePool();
     ComponentPool *PopulationQAComponent=new ComponentPool();
    PopulationQAMetrics->loadElements();


    PopulationQAComponent->loadElements();
    nissues=PopulationQAMetrics->getElements();

    matrixIssueQuality=PopulationQAMetrics;
    matrixComponentQuality=PopulationQAComponent;
    cout << "Analizando Caso de Estudio ..." << endl;
    cout << "Tareas a planificar: " <<  nissues << endl;
```

```
    //  Prioridades P1,P2,P3 60,30,10   0.4
    // Fase de pruebas C1,C2,C3 Functional Technical Design
Development
    // 30,20,10,20,20  0.3


for(int ii=1; ii<argc; ii++) {
    if(strcmp(argv[ii++],"seed") == 0) {
      GARandomSeed((unsigned int)atoi(argv[ii]));
    }
  }

// Set the default values of the parameters.

  GAParameterList params;
  GASteadyStateGA::registerDefaultParameters(params);
  params.set(gaNpopulationSize, 150);     // population size
  params.set(gaNpCrossover, 0.6);   // probability of crossover
  params.set(gaNpMutation, 0.04);   // probability of mutation
  params.set(gaNnGenerations, 5000);      // number of generations
  params.set(gaNpReplacement, 0.6); // how much of pop to replace each
gen
  params.set(gaNscoreFrequency, 10);      // how often to record scores
  params.set(gaNnReplacement, 60);  // how much of pop to replace each
gen
  params.set(gaNflushFrequency, 10);      // how often to dump scores
to file
params.set(gaNscoreFilename, "reference.dat");
  params.parse(argc, argv, gaFalse);



  GAListGenome<int> genome(objective);
  genome.initializer(ListInitializer);
  genome.mutator(GAListGenome<int>::SwapMutator);
genome.crossover(XOVER);

// Now that we have our genome, we create the GA (it clones the genome
to
// make all of the individuals for its populations).  Set the
parameters on
// the GA then let it evolve.

  GASteadyStateGA ga(genome);
 // ga.crossover(GAListGenome<int>::PartialMatchCrossover);
  ga.parameters(params);
  ga.evolve();

// Assign the best that the GA found to our genome then print out the
results.

  genome = ga.statistics().bestIndividual();
  cout << "the ga generated the following list (objective score is ";
  cout << genome.score() << "):\n" << genome << "\n";

  cout << "Lista Planificación " << endl;
  VisualizePhenotype(genome);
 // cout << "best of generation data are in '" << ga.scoreFilename()
<< "'\n";
 // cout << ga.parameters() << "\n";
```

```
//matrixIssueQuality->VisualizeOptimization();
//  char *fn;
//  ga.get(gaNscoreFilename, &fn);
//  cout << "filename is '" << fn << "'\n";

    return 0;
}

void VisualizePhenotype(GAGenome & c){
 GAListGenome<int> & genome = (GAListGenome<int> &)c;
 *genome.head()==0;
for(int i=0; i<genome.size(); i++){
int valueCurrent = *genome.current();
    matrixIssueQuality->VisualizeOptimization(valueCurrent);
    *genome.next();
    }

    ofstream out("ValueInformation.txt");
    streambuf *coutbuf = cout.rdbuf();
    cout.rdbuf(out.rdbuf());
    *genome.head()==0;
for(int i=0; i<genome.size(); i++){
int valueCurrent = *genome.current();
    matrixIssueQuality->VisualizeOptimization(valueCurrent);
    *genome.next();
    }
cout.rdbuf(coutbuf);
}
float
objective(GAGenome & c)
{
QualityIssue Elemento;
QualityComponent ElementoComponent;
double firsthalf=0;
double secondhalf=0;
    int fix_tempo=0;
  int timeThreshold=0;
  int EnclosedThreshold=0;
  int CreationDistance=0;
  int testPhase=0;
  int timeBudget=0;
  GAListGenome<int> & genome = (GAListGenome<int> &)c;
#ifdef MONEY_TALK
*genome.head()==0;

for(int i=1; i<genome.size(); i++){
    int valueCurrent = *genome.current();
    Elemento= matrixIssueQuality->getElement(valueCurrent);
    *genome.next();
  if(Elemento.getComponent()=="")
    continue;
    if(matrixComponentQuality->searchByName(Elemento.getComponent())>-
1)
    ElementoComponent=matrixComponentQuality-
>getElement(matrixComponentQuality-
>searchByName(Elemento.getComponent()));

    if(i<genome.size()/2){
     firsthalf+=ElementoComponent.getQualityderivateMetrics()-
>getErrorDebt();
    }else{
```

```
     secondhalf+=ElementoComponent.getQualityderivateMetrics()-
>getErrorDebt();
    }

}

#endif
  float score = (*genome.head() == 0); // move to head of list
  for(int i=1; i<genome.size(); i++){

     int valueCurrent = *genome.current();

    CreationDistance=i-valueCurrent;
    //0.3
    Elemento= matrixIssueQuality->getElement(valueCurrent);
double premureWork=WINDOW_WORK_BASED_COMPLEXITY;
    if(Elemento.getComponent()=="")
     premureWork*=DEFAULT_COMPLEX;
        else{
        if(matrixComponentQuality-
>searchByName(Elemento.getComponent())>-1){
        ElementoComponent=matrixComponentQuality-
>getElement(matrixComponentQuality-
>searchByName(Elemento.getComponent()));
   premureWork*=ElementoComponent.getQualityderivateMetrics()-
>getcomplexityKLOC();
        }
        }

    if(Elemento.getDateFix()!="")
       timeBudget=TIME_SPLIT-DATE_FIX;
    else{
        timeBudget=TIME_SPLIT;
    }
if(Elemento.getPriorityFix().rfind("P1")!=std::string::npos)

        premureWork*=FIX_HIGH;

    else if(Elemento.getPriorityFix().rfind("P2")!=std::string::npos)
            premureWork*=FIX_MEDIUM;

    else{

    premureWork*=FIX_LOW;

    }


    /* Versión para el aprovechamiento del origen, se podrían usar
otro tipo de métricas basadas en las pruebas
    según el número de pruebas, la calidad, la cobertura que ofrezcan,
número de casos de prueba */
#ifdef  TEST_PHASE


    if(Elemento.getTestPhase()!=""){



if(Elemento.getTestPhase().rfind("SIT C1")!=std::string::npos)
```

```
        premureWork=1200;

    else if(Elemento.getTestPhase().rfind("ST C2")!=std::string::npos)
            premureWork=400;

      else if(Elemento.getTestPhase().rfind("ST
C3")!=std::string::npos)
            premureWork=600;
              else if(Elemento.getTestPhase().rfind("ST
C1")!=std::string::npos)
            fix_tempo=200;
     else
if(Elemento.getTestPhase().rfind("Functional")!=std::string::npos)
            fix_tempo=0;
     else
if(Elemento.getTestPhase().rfind("Design")!=std::string::npos)
            fix_tempo=0;
             else
if(Elemento.getTestPhase().rfind("Development")!=std::string::npos)
            fix_tempo=150;
    else{

    premureWork-=300;

    }
    }
#endif
 /*-----------------------------------------------------------------------
-----------------*/
#ifndef TEST_PHASE
    if(CreationDistance<timeBudget-premureWork&& 0<CreationDistance)
        score+=i;
#endif // TEST_PHASE

#ifdef TEST_PHASE
         if(fix_tempo<valueCurrent&& CreationDistance<timeBudget-
premureWork)
        score+=i;
#endif // TEST_PHASE
 *genome.next();
  }

  #ifdef MONEY_TALK
  if(firsthalf>secondhalf)
    score=0;
  #endif // MONEY_TALK

  return score;
}



void
ListInitializer(GAGenome & c)
{
  GAListGenome<int> &child=(GAListGenome<int> &)c;
  while(child.head()) child.destroy(); // destroy any pre-existing
list

  int n= nissues;
  child.insert(0,GAListBASE::HEAD); // the head node contains a '0'
```

```
  for(int i=1; i<n; i++)
    child.insert(i);          // each subsequent node contains a number
  for(int j=0; j<n; j++)
    child.swap(GARandomInt(0,n-1), GARandomInt(0,n-1));
}

int
PMXover(const GAGenome& g1, const GAGenome& g2, GAGenome* c1,
GAGenome* c2) {
  GAListGenome<int> &mom=(GAListGenome<int> &)g1;
  GAListGenome<int> &dad=(GAListGenome<int> &)g2;

  int a = GARandomInt(0, mom.size());
  int b = GARandomInt(0, dad.size());
  int h;
  if (b<a) { h=a; a=b; b=h; }

  int* index;
  int i,j,nc=0;

  if(c1) {
    GAListGenome<int> &sis=(GAListGenome<int> &)*c1;
    sis.GAList<int>::copy(mom);
    GAListIter<int> diter(dad);
    index = diter.warp(a);
    for(i=a; i<b; i++, index=diter.next()){
      if(*sis.head() == *index){
      sis.swap(i,0);
      }
      else{
      for(j=1; (j<sis.size()) && (*sis.next() != *index); j++);
      sis.swap(i,j);  // no op if j>size
      }
    }
    sis.head();          // set iterator to head of list
    nc += 1;
  }
  if(c2) {
    GAListGenome<int> &sis=(GAListGenome<int> &)*c2;
    sis.GAList<int>::copy(mom);
    GAListIter<int> diter(dad);
    index = diter.warp(a);
    for(i=a; i<b; i++, index=diter.next()){
      if(*sis.head() == *index){
      sis.swap(i,0);
      }
      else{
      for(j=1; (j<sis.size()) && (*sis.next() != *index); j++);
      sis.swap(i,j);  // no op if j>size
      }
    }
    sis.head();          // set iterator to head of list
    nc += 1;
  }

  return nc;
}


template <> int
GAListGenome<int>::write(ostream & os) const
```

```
{
  int *cur, *head;
  GAListIter<int> tmpiter(*this);
  if((head=tmpiter.head()) != 0) {
    os << *head << " ";
    for(cur=tmpiter.next(); cur && cur != head; cur=tmpiter.next())
      os << *cur << " ";
  }

  return os.fail() ? 1 : 0;
}
```

```
// force instantiations for compilers that do not do auto
instantiation
// for some compilers (e.g. metrowerks) this must come after any
// specializations or you will get 'multiply-defined errors when you
compile.
#if !defined(GALIB_USE_AUTO_INST)
#include <ga/GAList.C>
#include <ga/GAListGenome.C>
GALIB_INSTANTIATION_PREFIX GAList<int>;
GALIB_INSTANTIATION_PREFIX GAListGenome<int>;
#endif
```

6 Resultados

Incluidos en el directorio Datos, resultados basados en Rstudio, y ggplot2. Los datos se encuentran en la carpeta Datos, con los diferentes casos y una comparativa más detallada a nivel de incidencias y su prioridad. La información puede ser ampliada modificando la función IssuePool::VisualizeOptimization(int DataISSUE) dentro de la carpeta library, QAPool.cpp

6.1 Caso 0

6.1.1 Readme.txt

```
#define MAX_ISSUES 50
#define TIME_SPLIT 80
#define DATE_FIX 30

#define WINDOW_WORK_BASED_COMPLEXITY 50
#define DEFAULT_COMPLEX 1

#define FIX_HIGH    4
#define FIX_MEDIUM  2
#define FIX_LOW     1

#define MONEY_TALK 1
```

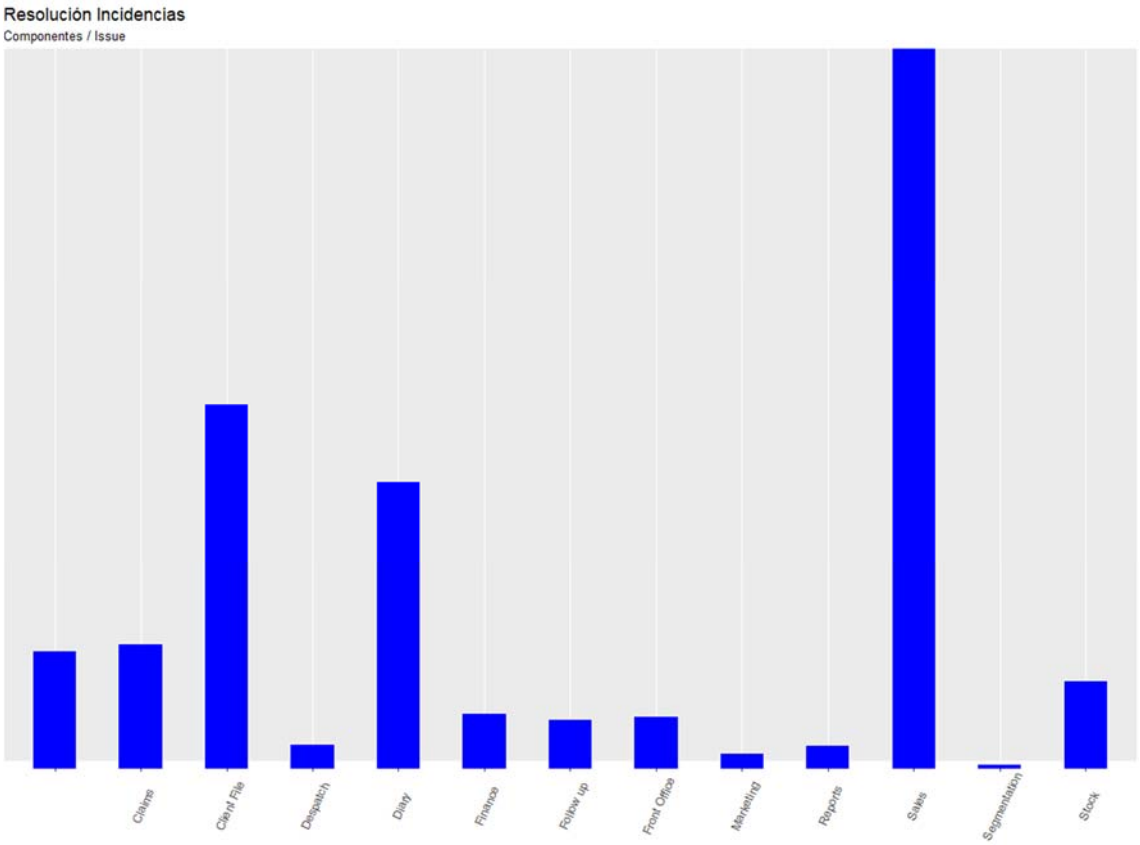
```
//#define TEST_PHASE 1
```

```
  params.set(gaNpopulationSize, 80);      // population size
  params.set(gaNpCrossover, 0.6);   // probability of crossover
  params.set(gaNpMutation, 0.04);   // probability of mutation
  params.set(gaNnGenerations, 100); // number of generations
  params.set(gaNpReplacement, 0.6); // how much of pop to replace each
gen
  params.set(gaNscoreFrequency, 10);      // how often to record scores
  params.set(gaNnReplacement, 60);  // how much of pop to replace each
gen
  params.set(gaNflushFrequency, 10);      // how often to dump scores
to file
```

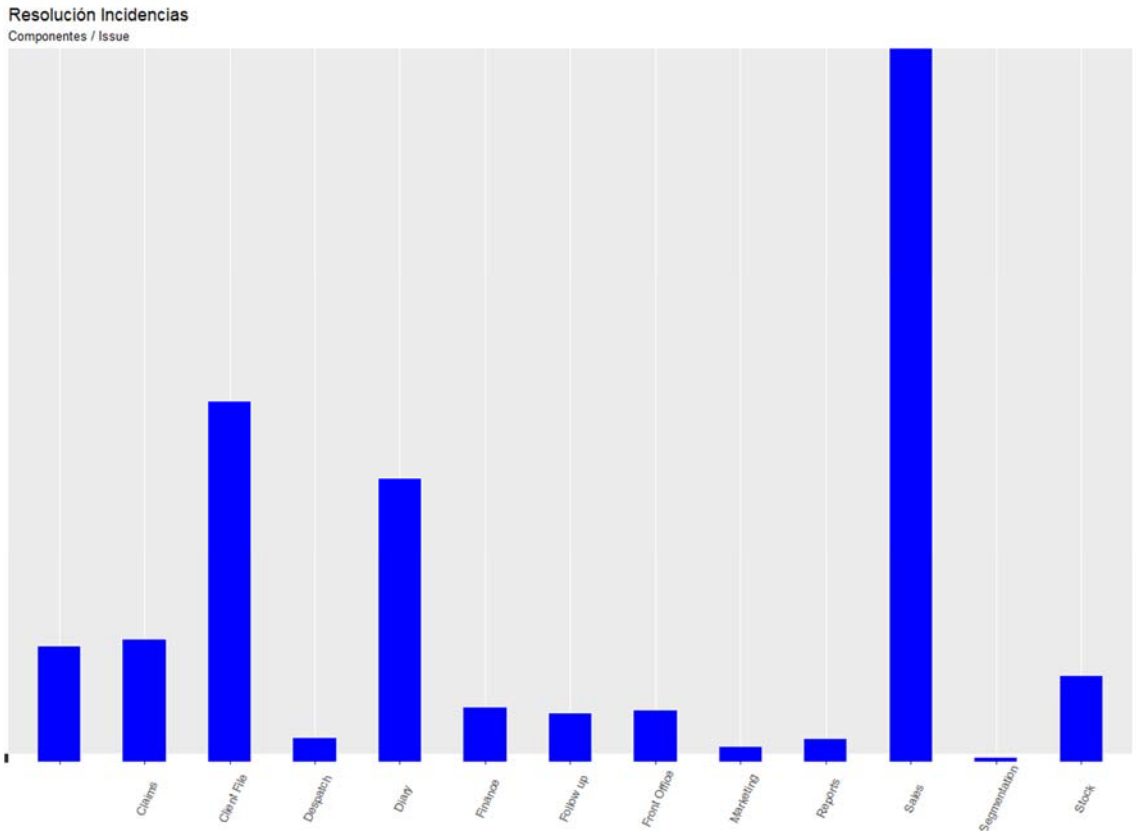
6.1.2 Graficas

6.1.2.1 [0-500] Incidencias

Captura.png



6.1.2.2 [0-500] Incidencias



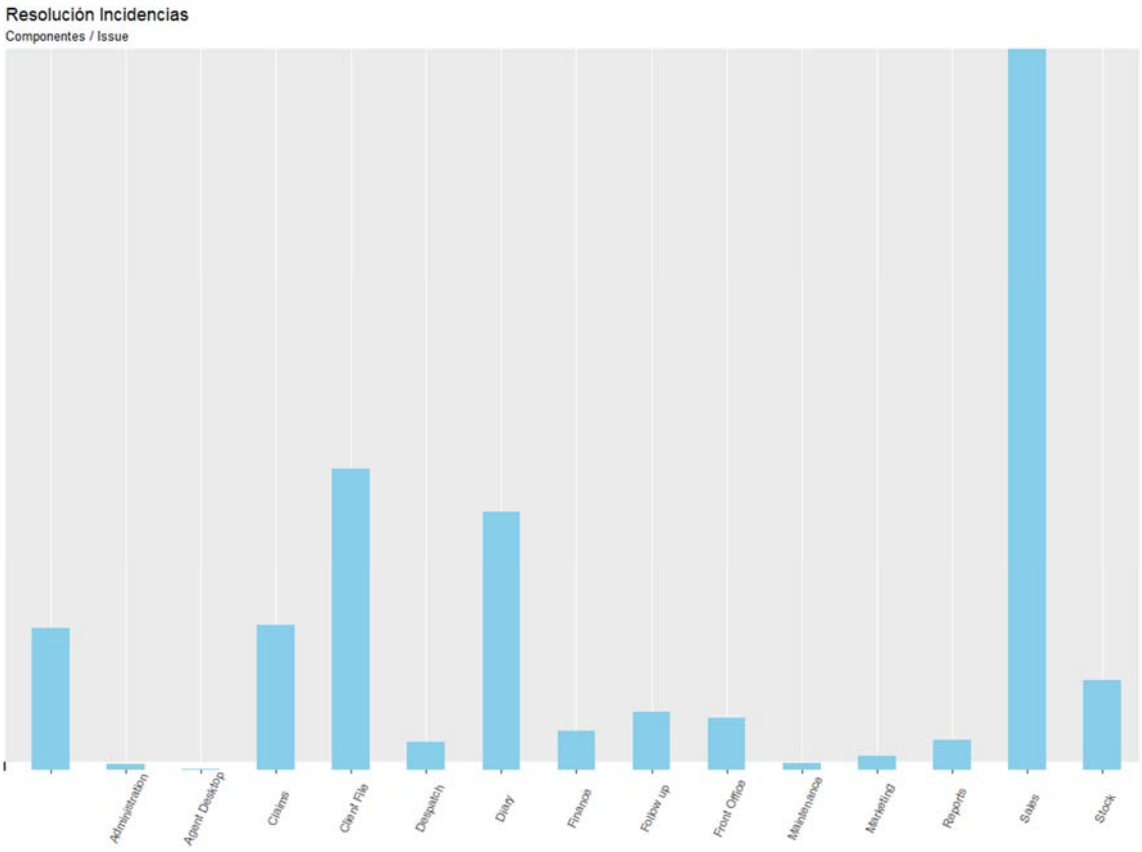
6.2 Caso I

6.2.1 Readme I.txt

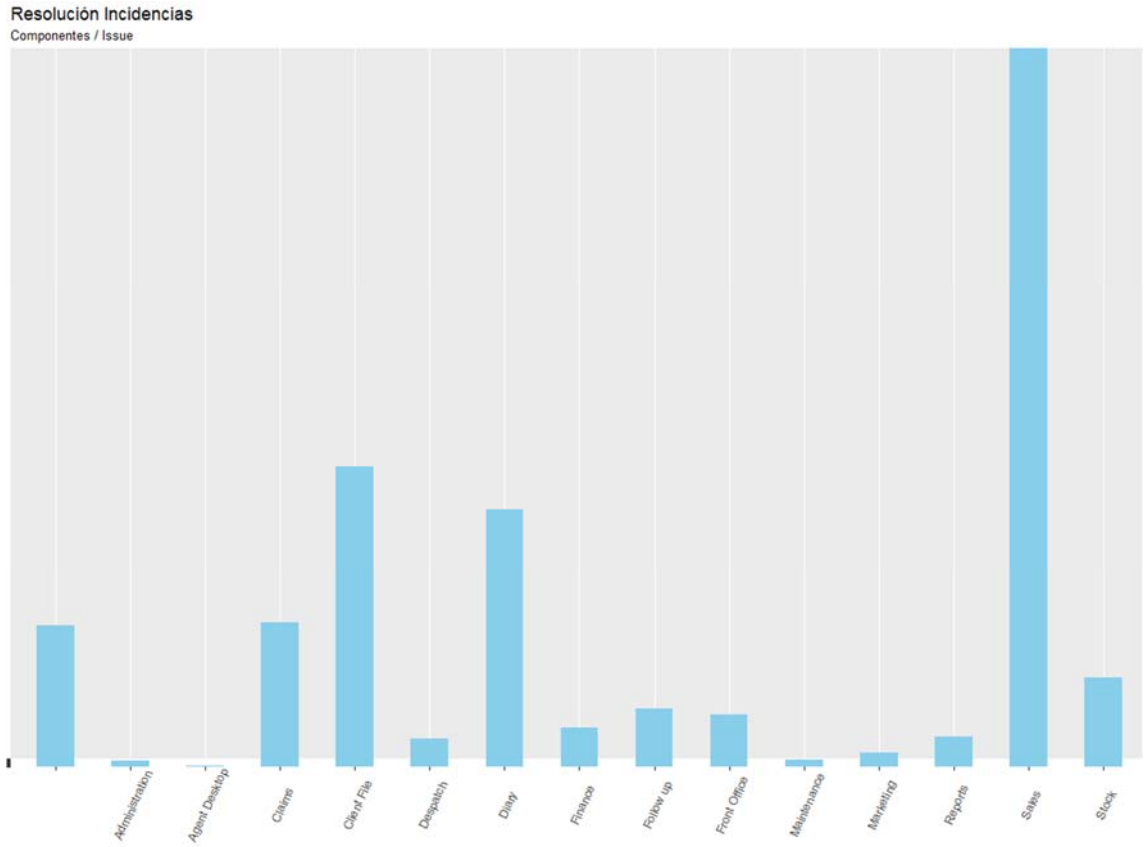
```
#define MAX_ISSUES 50
#define TIME_SPLIT 80
#define DATE_FIX 30
#define WINDOW_WORK_BASED_COMPLEXITY 50
#define DEFAULT_COMPLEX 1
#define FIX_HIGH    4
#define FIX_MEDIUM  2
#define FIX_LOW     1

#define MONEY_TALK 1
//#define TEST_PHASE 1
  params.set(gaNpopulationSize, 80);      // population size
  params.set(gaNpCrossover, 0.6);   // probability of crossover
  params.set(gaNpMutation, 0.04);   // probability of mutation
  params.set(gaNnGenerations, 100); // number of generations
  params.set(gaNpReplacement, 0.6); // how much of pop to replace each
gen
  params.set(gaNscoreFrequency, 10);      // how often to record scores
  params.set(gaNnReplacement, 60);  // how much of pop to replace each
gen
  params.set(gaNflushFrequency, 10);      // how often to dump scores
to file
```

6.2.2 Graficas

6.2.2.1 [0-500] Incidencias



6.2.2.2 [500-1000] Incidencias



6.3 Caso II

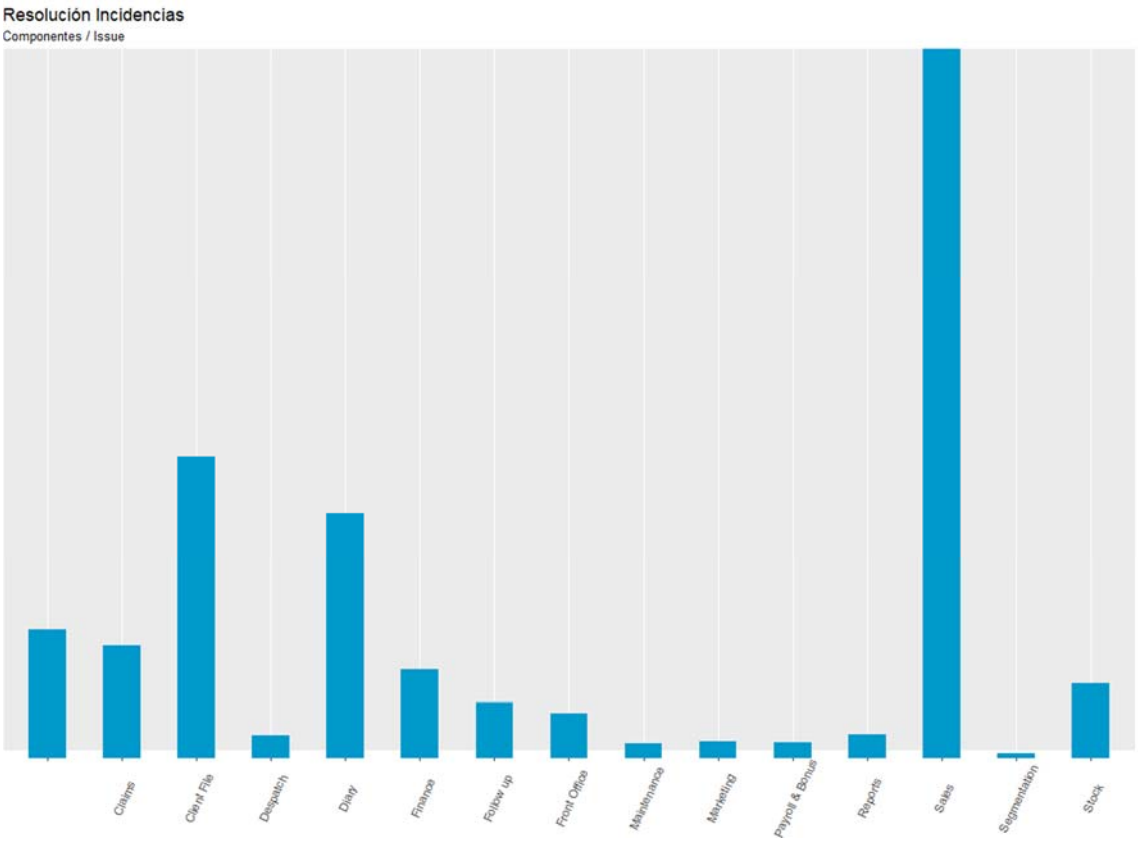
6.3.1 Readme II.txt

```
#define MAX_ISSUES 50
#define TIME_SPLIT 80
#define DATE_FIX 30

#define WINDOW_WORK_BASED_COMPLEXITY 50
#define DEFAULT_COMPLEX 1

#define FIX_HIGH    4
#define FIX_MEDIUM  2
#define FIX_LOW     1

#define MONEY_TALK 1
//#define TEST_PHASE 1


  params.set(gaNpopulationSize, 80);      // population size
  params.set(gaNpCrossover, 0.6);   // probability of crossover
  params.set(gaNpMutation, 0.04);   // probability of mutation
  params.set(gaNnGenerations, 500); // number of generations
  params.set(gaNpReplacement, 0.6); // how much of pop to replace each
gen
  params.set(gaNscoreFrequency, 10);      // how often to record scores
  params.set(gaNnReplacement, 60);  // how much of pop to replace each
gen
```

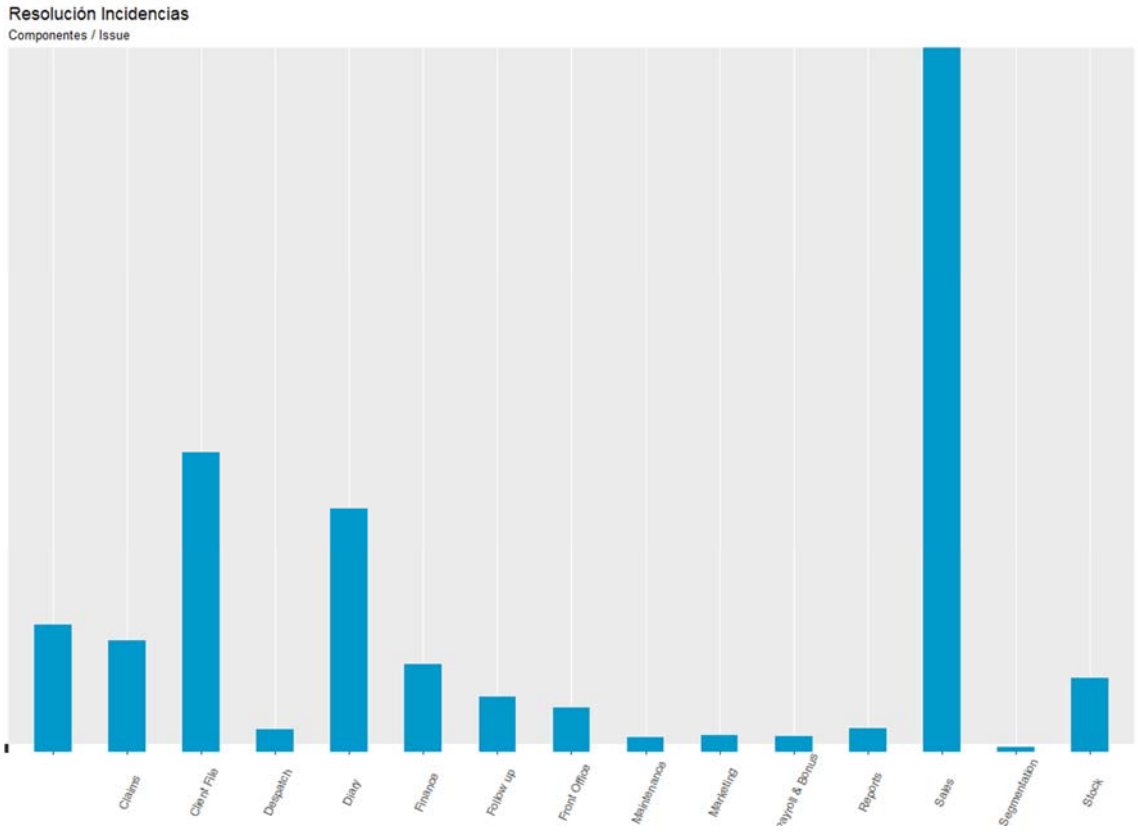
```
  params.set(gaNflushFrequency, 10);      // how often to dump scores
to file
```

6.3.2 Graficas

6.3.2.1 [0-500] Incidencias



6.3.2.2 [500-1000] Incidencias



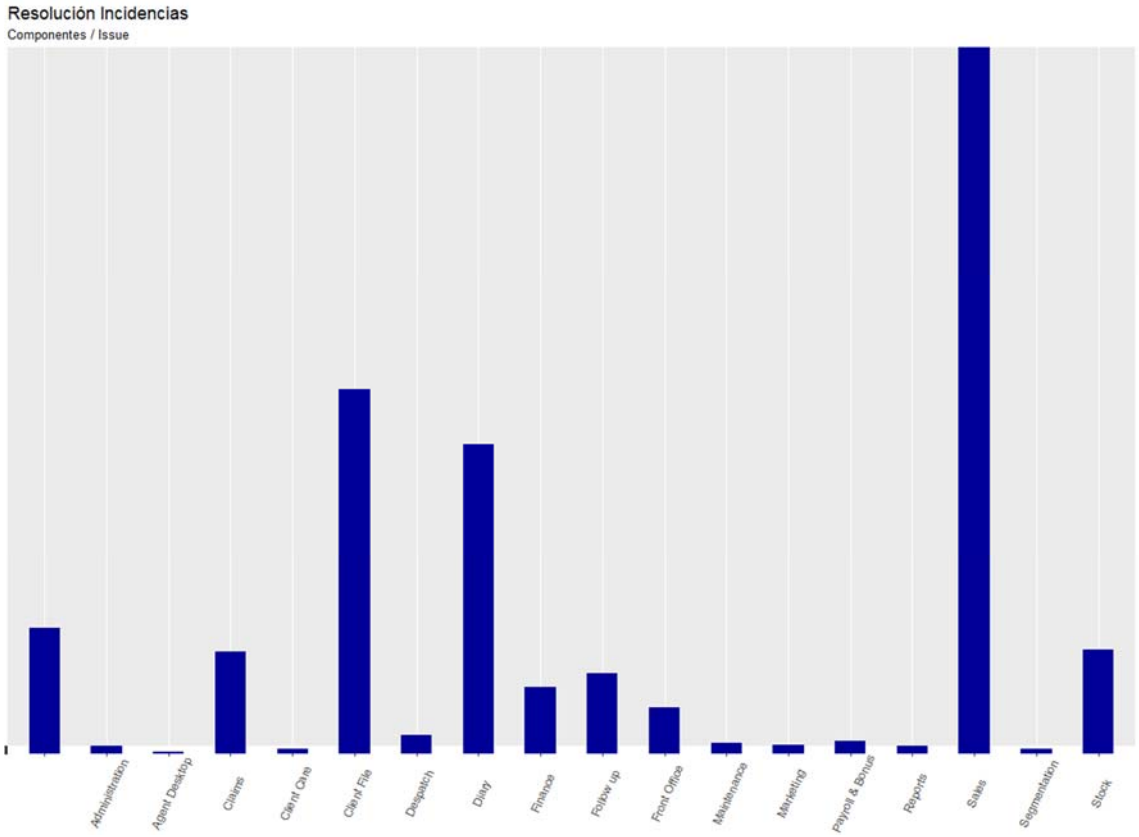
6.4 Caso III

6.4.1 Readme III.txt

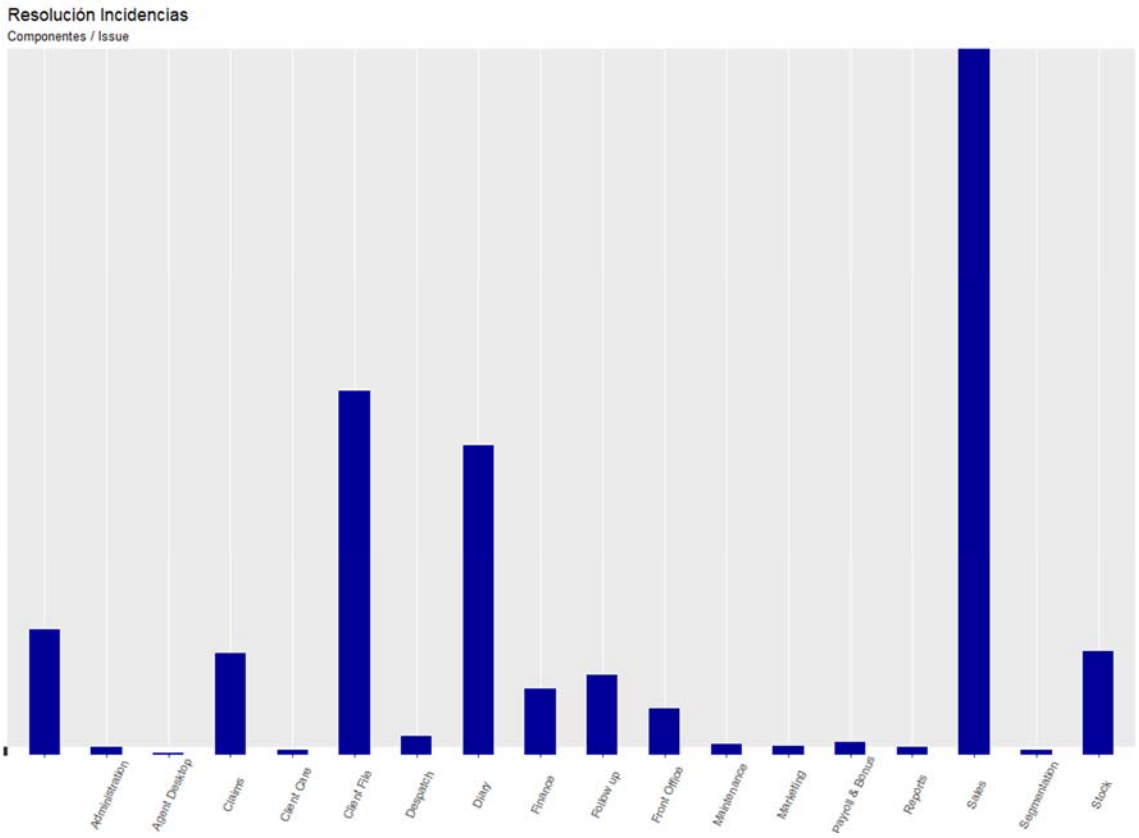
```
#define MAX_ISSUES 50
#define TIME_SPLIT 80
#define DATE_FIX 30

#define WINDOW_WORK_BASED_COMPLEXITY 50
#define DEFAULT_COMPLEX 1

#define FIX_HIGH    4
#define FIX_MEDIUM  2
#define FIX_LOW     1

//#define MONEY_TALK 1
#define TEST_PHASE 1

 params.set(gaNpopulationSize, 80); // population size
  params.set(gaNpCrossover, 0.6);   // probability of crossover
  params.set(gaNpMutation, 0.04);   // probability of mutation
  params.set(gaNnGenerations, 500); // number of generations
  params.set(gaNpReplacement, 0.6); // how much of pop to replace each
gen
  params.set(gaNscoreFrequency, 10);      // how often to record scores
  params.set(gaNnReplacement, 60);  // how much of pop to replace each
gen
  params.set(gaNflushFrequency, 10);      // how often to dump scores
to file
```

6.4.2 Graficas

6.4.2.1 [0-500] Incidencias



6.4.2.2 [500-1000] Incidencias



6.5 Caso VI

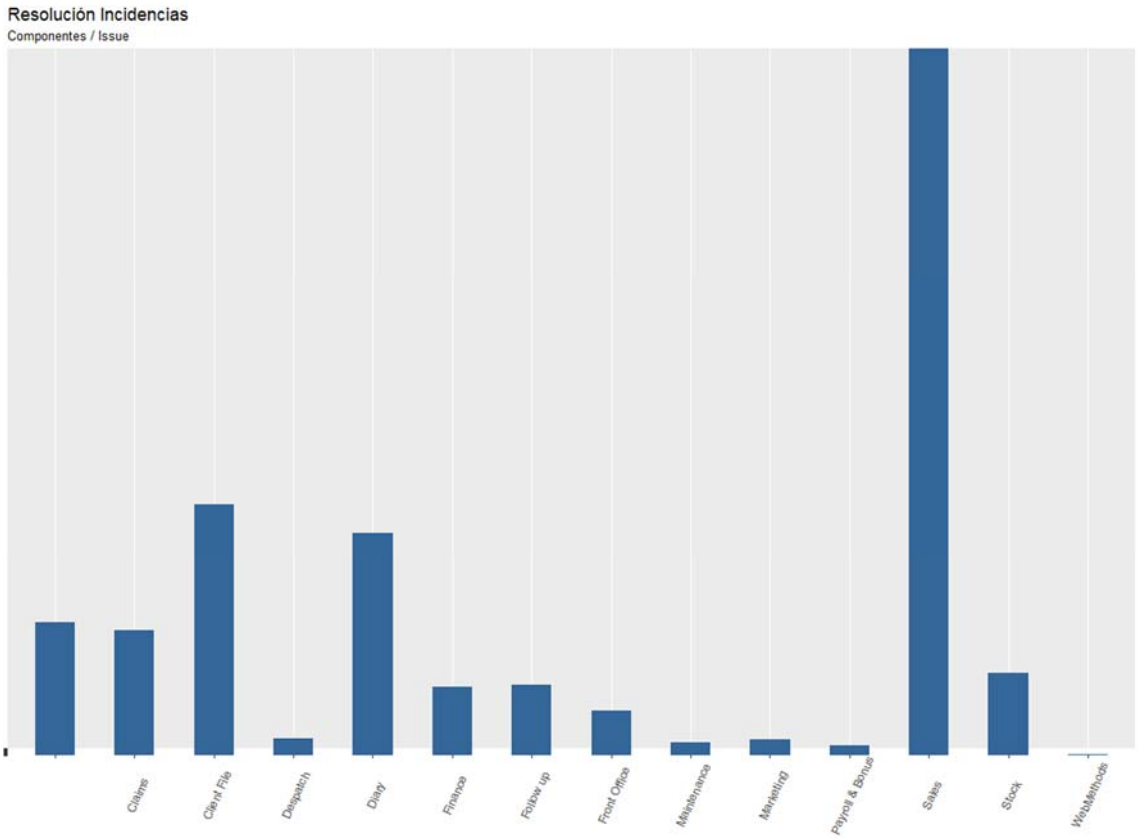
6.5.1 Readme IV.txt

```
#define MAX_ISSUES 50
#define TIME_SPLIT 80
#define DATE_FIX 30

#define WINDOW_WORK_BASED_COMPLEXITY 50
#define DEFAULT_COMPLEX 1

#define FIX_HIGH    4
#define FIX_MEDIUM  2
#define FIX_LOW     1

//#define MONEY_TALK 1
#define TEST_PHASE 1

  params.set(gaNpopulationSize, 80);      // population size
  params.set(gaNpCrossover, 0.6);   // probability of crossover
  params.set(gaNpMutation, 0.04);   // probability of mutation
  params.set(gaNnGenerations, 100); // number of generations
  params.set(gaNpReplacement, 0.6); // how much of pop to replace each
gen
  params.set(gaNscoreFrequency, 10);      // how often to record scores
```

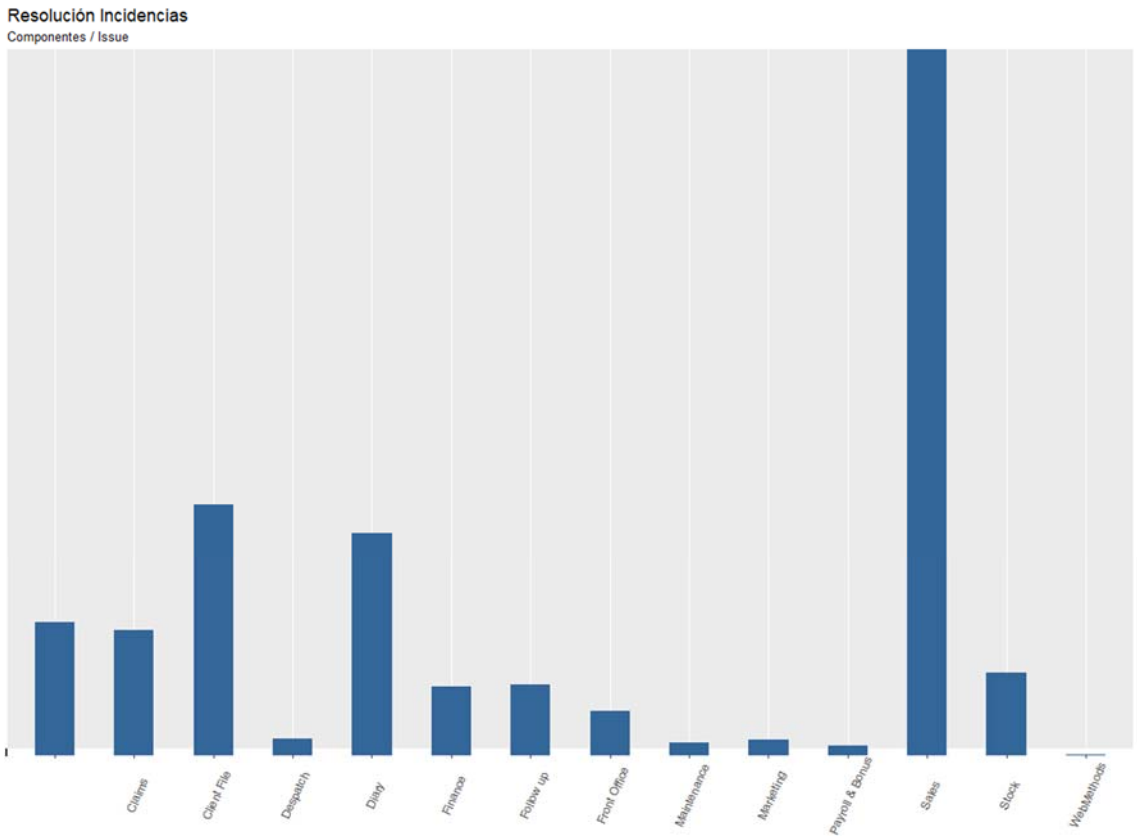
```
  params.set(gaNnReplacement, 60); // how much of pop to replace each
gen
  params.set(gaNflushFrequency, 10);     // how often to dump scores
to file
```

6.5.2 Gráficas

6.5.2.1 [0-500] Incidencias



6.5.2.2 [500-1000] Incidencias



7 Conclusiones

La librería galib centrarse en la forma de crear una función de aptitud adecuada a los individuos de la población, que por debajo de 100 individuos es muy eficiente, de 100-150 individuos tarda algo más. En caso de necesitar poblaciones de más de 100 individuos, usaría otro tipo de representación o listas fraccionadas.

Se que se requiere un análisis minucioso de las tareas que no se quieren dejar en el olvido, y el algoritmo genético permite eso, no crear una "muerte" por inhibición a ninguna incidencia, ya sea la casualidad en la ordenación inicial de las incidencias o que casos puntuales con alta puntuación en la función de aptitud, permiten conservar casos no "perfectos".

Esto es muy importante, porque al intentar aproximarse a Pareto, también se encuentran incidencias en módulos que parecían libres de error.

El modelo de calidad por objetivos marca unas directrices muy claras de cara al ejercicio de las pruebas software en todo momento para mejorar la calidad de producto. En ese sentido crea unas métricas muy claras y concisas a nivel de proceso, en las características físicas del software (líneas de código, complejidad…)

La complejidad de Halstead, es una forma de abordar Pareto y la métrica de los componentes, en cuanto a predicción de defectos.

8 Futura línea de investigación

Sería fundamental el desarrollo para metodologías ágiles, iterativas o espiral y prototipos.

Para ello es necesario incluir un módulo de "Factibilidad" de la solución, en el caso de la aplicación del Planificador genético basándose en los datos creados con NormalizeTablesTask.linq, una matriz de "precedencia de las actividades" con tiempo de creación, el problema sería las actividades cíclicas, que fueran generadoras de incidencias y después volvieran a dar requerir asistencia.

Es cierto que hay muchos datos cíclicos, porque se crea la dimensión Reopened, pero puede ser interesante también el desarrollo de la aplicación como un grafo unidireccional, tipo DAG, de esta forma la ejecución en paralelo puede ser mucho más acertada, tal como se hace con Apache Spark[1].

El análisis posterior a través de máquina de aprendizaje que hubiera hecho clustering de la información en primer lugar, para sacar conclusiones más certeras de

[1] https://spark.apache.org/

“cuánto” optimiza mi programa de Planificador Genético, para futuros proyectos en que se reutilicen módulos o metodologías similares.

9 Bibliografía

Adams, J., Balas, E., & Zawack, D. (1988). The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science*, *34*(3), 391-401.

Arapidis, C. (2012). *Sonar Code Quality Testing Essentials.* Birmingham: Packt Publishing, Limited.

Basili, V. R., & Rombach, H. D. (1987). Tailoring the software process to project goals and environments. En *Proceedings of the 9th international conference on Software Engineering* (pp. 345–357). IEEE Computer Society Press.

Bagchi, T. P. (1999). *Multiobjective Scheduling by Genetic Algorithms*. Boston, MA: Springer US.

Blazewicz, J., Lenstra, J. K., & Kan, A. R. (1983). Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, *5*(1), 11–24.

Bean, J. C. (1994). Genetic Algorithms and Random Keys for Sequencing and Optimization. *ORSA Journal on Computing*, *6*(2), 154-160.

Bierwirth, C. (1995). A generalized permutation approach to job shop scheduling with genetic algorithms. *OR Spektrum*, *17*(2-3), 87-92.

Bierwirth, C., Mattfeld, D. C., & Kopfer, H. (1996). On permutation representations for scheduling problems. In H.-M. Voigt, W. Ebeling, I. Rechenberg, & H.-P. Schwefel (Eds.), *Parallel Problem Solving from Nature — PPSN IV* (Vol. 1141, pp. 310–318). Berlin, Heidelberg: Springer Berlin Heidelberg.

Błażewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., & Węglarz, J. (2001). *Scheduling computer and manufacturing processes*. Berlin; New York: Springer. Retrieved from http://public.eblib.com/choice/publicfullrecord.aspx?p=3097406

Boehm, B., & Basili, V. (2005). Software Defect Reduction Top-10 List. En B. Boehm, H. D. Rombach, & M. V. Zelkowitz (Eds.), *Foundations of Empirical Software Engineering* (pp. 426-431). Springer Berlin Heidelberg

Boehm, B. W. (1988). Understanding and controlling software costs. *Journal of Parametrics*, *8*(1), 32–68.

Cano, E. L., Moguerza, J. M., & Prieto Corcoba, M. (2015). *Quality control with R: an ISO standards approach*.

Cheng, R., Gen, M., & Tsujimura, Y. (1999). A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. *Computers & Industrial Engineering*, *36*(2), 343–364.

Dahal, K. P., Cowling, P. I., & Tan, K. C. (2007). *Evolutionary Scheduling*. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg.

Desharnais, J.-M., Abran, A., & Suryn, W. (2011). Identification and analysis of attributes and base measures within ISO 9126. *Software Quality Journal*, *19*(2), 447-460.

Della Croce, F., Tadei, R., & Volta, G. (1995). A genetic algorithm for the job shop problem. *Computers & Operations Research*, *22*(1), 15-24.

Ebert, C. (1997). Experiences with criticality predictions in software development. En M. Jazayeri & H. Schauer (Eds.), *Software Engineering ? ESEC/FSE'97* (Vol. 1301, pp. 278-293). Berlin, Heidelberg: Springer Berlin Heidelberg.

Ebert, C. (Ed.). (2005). *Best practices in software measurement: how to use metrics to improve project and process performance*. Berlin ; New York: Springer.

Eiselt, H. A., & Sandblom, C.-L. (2004). *Decision analysis, location models, and scheduling problems: with 48 tables*. Berlin: Springer.

Elberzhager, F., & Münch, J. (2013). Using Early Quality Assurance Metrics to Focus Testing Activities.

Falkenauer, E., & Bouffouix, S. (1991). A genetic algorithm for job shop. En *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on* (pp. 824–829). IEEE.

Fang, H.-L., Ross, P., & Corne, D. (1993). *A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems*. University of Edinburgh, Department of Artificial Intelligence.

D. Goldberg and R. Lingle, "Alleles, Loci and the Traveling Salesman Problem," Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications, Los Angeles, 1985, pp. 154-159.

Jones, C. (2014). *The technical and social history of software engineering*. Upper Saddle River, NJ: Addison-Wesley.

Kanet, J., & Sridharan, V. (1991). *PROGENITOR: A Genetic Algorithm for Production Scheduling* (Vol. Wirtschaftsinformatik).

Kachitvichyanukul, V., & Sitthitham, S. (2011). A two-stage genetic algorithm for multi-objective job shop scheduling problems. *Journal of Intelligent Manufacturing*, *22*(3), 355-365.

Kan, S. H. (2003). *Metrics and models in software quality engineering* (2nd ed). Boston: Addison-Wesley.

Garey, M. R., & Johnson, D. S. (1975). Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, *4*(4), 397–411.

Gen, M., Tsujimura, Y., & Kubota, E. (1994). Solving job-shop scheduling problems by genetic algorithm (Vol. 2, pp. 1577-1582). IEEE.

Gen, M., & Cheng, R. (1997). *Genetic algorithms and engineering design*. New York: Wiley.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, Mass: Addison-Wesley Pub. Co.

International Conference on Genetic Algorithms and Their Applications, Grefenstette, J. J., & Texas Instruments Incorporated (Eds.). (1988). *Proceedings of the First International Conference on Genetic Algorithms and Their Applications: July 24- 26, 1985 at the Carnegie-Mellon University, Pittsburgh, Pa*. Hillsdale, N.J. u.a: Erlbaum.

https://www.linqpad.net/

Mori, M., & Tseng, C. C. (1997). A genetic algorithm for multi-mode resource constrained project scheduling problem. *European Journal of Operational Research*, *100*(1), 134–141.

Myers, G. J., Sandler, C., & Badgett, T. (2012). *The art of software testing* (3rd ed). Hoboken, N.J: John Wiley & Sons.

National Research Council (U.S.), Committee on Certifiably Dependable Software Systems, Jackson, D., Millett, L. I., & Thomas, M. (2007). *Software for dependable systems: sufficient evidence?* Washington, D.C.: National Academies Press.

Nakano, R., & Yamada, T. (1991). *Conventional Genetic Algorithm for Job Shop Problems.* (Vol. ICGA).

Nakano, R., & Yamada, T. (1992). *A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems*.

Paredis, J. (1992). Exploiting Constraints as Background Knowledge for Genetic Algorithms: A Case-Study for Scheduling. En *PPSN* (pp. 231–240).

Pfleeger, S. L., & Quiroga, E. (2002). *Ingeniería de software: teoría y práctica*. Argentina; México: Prentice Hall : Pearson Educación.

Pinedo, M. (2009). *Planning and scheduling in manufacturing and services* (2nd ed). Dordrecht ; New York: Springer.

Project Management Institute (Ed.). (2013). *A guide to the project management body of knowledge (PMBOK guide)* (Fifth edition). Newtown Square, Pennsylvania: Project Management Institute, Inc.

Putnam, L. H., & Myers, W. (1992). *Measures for excellence: reliable software on time, within budget*. Englewood Cliffs, N.J: Yourdon Press.

Royce, W. (2005). Successful software management style: Steering and balance. *IEEE software*, *22*(5), 40–47.

Roth, T. (2016). Working with the qualityTools package. Retrieved from

Schach, S. R., Fernández, E., Guerrero, E., Trejo Ramírez, R. A., & Juárez Betancourt, S. T. (2006). *Ingeniería de software clásica y orientada a objetos*. México: McGraw-Hill.

St-Louis, D., & Suryn, W. (2012). Enhancing ISO/IEC 25021 quality measure elements for wider application within ISO 25000 series. En *IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society* (pp. 3120–3125).

Sprecher, A., Hartmann, S., & Drexl, A. (1997). An exact algorithm for project scheduling with multiple modes. *OR Spectrum*, *19*(3), 195–203.

Takeshi Yamada, R. N. (n.d.). *Conventional Genetic Algorithm for Job Shop Problems.* (Vol. ICGA 1991).

Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*. Hoboken, N.J: John Wiley & Sons.

Uyar, A. S., Ozcan, E., & Urquhart, N. (Eds.). (2013). *Automated Scheduling and Planning* (Vol. 505). Berlin, Heidelberg: Springer Berlin Heidelberg.

Vance, S. (2013). *Quality code: software testing principles, practices, and patterns*. Upper Saddle River, NJ: Addison-Wesley.

Wall, M. B. (1996). *A genetic algorithm for resource-constrained scheduling*. Massachusetts Institute of Technology.

Zhang, R., & Wu, C. (2008). Decomposition and immune genetic algorithm for scheduling large job shops. En *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on* (pp. 33–39). IEEE.

Zhang, C., Li, P., Rao, Y., & Li, S. (2005). A New Hybrid GA/SA Algorithm for the Job Shop Scheduling Problem. In G. R. Raidl & J. Gottlieb (Eds.), *Evolutionary Computation in Combinatorial Optimization: 5th European Conference, EvoCOP 2005, Lausanne, Switzerland, March 30 - April 1, 2005. Proceedings* (pp. 246–259). Berlin, Heidelberg: Springer Berlin Heidelberg.