308917

UNIVERSIDAD PANAMERICANA

ESCUELA DE INGENIERIA

CON ESTUDIOS INCORPORADOS A LA
UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

48
2ej.

USO DEL METODO DE IVAR JACOBSON EN LA ESPECIFICACION Y ANALISIS DE UNA HERRAMIENTA DE COMPUTO AUXILIAR EN EL DISEÑO DE EXPERIMENTOS DE FALLA A FATIGA

T E S I S

QUE PARA OBTENER EL TITULO DE:
INGENIERO MECANICO ELECTRICISTA
AREA: INGENIERIA MECANICA
P R E S E N T A :

JAVIER SOLORZANO ZAVALA

DIRECTOR: ING. EDMUNDO MARROQUIN TOVAR

MEXICO, D. F. 1998

263836

TESIS CON
FALLA DE ORIGEN

Universidad Nacional
Autónoma de México

Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Quiero dedicar esta tesis a mis padres, Manuel Enrique y Ana María, quienes me han apoyado y alentado a descubrir el mundo, y a mí mismo, mediante el estudio, la preparación académica y la formación humana. En especial le debo esta tesis a mi padre, cuyo ejemplo continuo de trabajo y su interminable paciencia hacia mi lograron que concluyera esta etapa de mi formación.

Quiero dar gracias especiales a Sofía, que me ha dado la energía e inspiración final que requerí para concluir esta tesis.

A ellos les debo todo mi amor.

Junio, 1998

1 INTRODUCCIÓN 1

2 EL MÉTODO DE CASOS DE USO DE IVAR JACOBSON 6

2.1 EL DESARROLLO DE SISTEMAS ES LA CONSTRUCCIÓN DE MODELOS 6
2.2 EL MODELO DE REQUERIMIENTOS 8
2.3 EL MODELO DE ANÁLISIS 12
2.4 EL MODELO DE DISEÑO 18
2.5 EL MODELO DE IMPLEMENTACIÓN 23
2.6 EL MODELO DE PRUEBAS 23

3 PROGRAMACIÓN ORIENTADA A OBJETOS 25

3.1 LAS GENERACIONES EN LOS LENGUAJES DE PROGRAMACIÓN 25
3.2 EL LENGUAJE *JAVA* Y LA PROGRAMACIÓN ORIENTADA A OBJETOS. 32
3.2.1 Encapsulación 36
3.2.2 Herencia 38
3.2.3 Polimorfismo 39
3.2.4 Java y C++ 40

4 EL FENÓMENO DE LA FATIGA 45

LA FALLA A FATIGA 45
4.1.1 Introducción a la fatiga 45
4.1.2 Modos de Carga de Fatiga 46
4.1.3 Definición y Descripción 50
4.1.4 Teorías del Fenómeno de la Fatiga 61
PRUEBAS DE FATIGA EN LABORATORIO 65
4.1.5 Introducción 65
4.1.6 Las curvas S-N-P 66
4.1.7 La influencia de un promedio de esfuerzo diferente a cero 68
4.1.8 Procedimientos de Prueba 70
4.1.9 Teoría de Soderberg 90

5 ANÁLISIS DE LA FUNCIONALIDAD DE UNA PEQUEÑA HERRAMIENTA AUXILIAR EN EL ANÁLISIS DE LA FATIGA 94

5.1 EL MODELO DE REQUERIMIENTOS 94

5.1.1 Actores *94*
5.1.2 Casos de uso e interfases *95*
5.1.3 Modelo en el dominio del problema *116*
5.2 EL MODELO DE OBJETOS 118
6 CONCLUSIONES **128**
7 BIBLIOGRAFÍA **132**

1 Introducción

Las tecnologías de la información han cambiado paulatinamente la forma en la que vivimos y trabajamos cotidianamente. Las computadoras se encuentran hoy en día en todos los ámbitos desde los electrodomésticos hasta los sistemas de control de las aeronaves. Sin embargo una computadora no es útil sin el programa que le da instrucciones sobre lo que debe hacer y cómo debe hacerse. Desarrollar estos programas no es una tarea simple. Sin embargo cuando se diseña todo un sistema, tradicionalmente se le ha dado poca importancia al software y por tanto ha quedado relegado a un papel secundario.

Es poco admitido el hecho de que la técnica en la ingeniería de software está todavía en estado muy primitivo. Desde sus inicios y todavia en la actualidad se puede asegurar que se encuentra en estado artesanal. Recientemente han surgido importantes avances en esta área. La mayor parte de los avances han surgido con el intento de estandarizar los procesos requeridos para producir software, haciendo la actividad más controlable, predecible y por tanto más confiable. Algunos ejemplos de estos avances los podemos ver en la adecuación de la norma ISO 9000 a productos de software en su versión ISO 9000-3 y en la recientemente publicada norma ISO 15504 que describe los procesos de producción de software.

Otros avances importantes se han logrado gracias a la investigación que han desarrollado centros como el *Software Engineering Institute* quienes recientemente publicaron una "disciplina" para la ingeniería de software

tomando como base los estudios de Watts Humphrey[1].

Uno de los mayores problemas que esta disciplina ha sufrido es que escribir programas de computadora es divertido. Es por esto que en el ámbito técnico de la programación ha habido una gran efervescencia en la publicación de metodologías de trabajo, especialmente en el área de la programación. El ejemplo más exitoso fue la metodología estructurada de Jordan. Durante algunos años se inventaron lenguajes de programación casi a diario.

Este énfasis tan grande en la programación tuvo como consecuencia un descuido en otras áreas de la creación de programas de cómputo que son el análisis del sistema en el que funcionará el programa y el diseño previo a la programación.

Un programa de computadora nunca funciona por sí mismo. Siempre está incrustado en un sistema mayor. Es decir, es siempre parte y nunca el todo. El estar integrado dentro de un sistema mayor implica que deberá interactuar con otros elementos del sistema, a veces físicos, y en otras veces humanos u otros programas de cómputo. Dependiendo del tamaño de los sistemas, las interacciones posibles entre los elementos del sistema crecen en cantidades que no son fácilmente asimilables por el ser humano. Es en este contexto que los programas de computadora fallan, poniendo la

---

[1] Humphrey, Watts. A Discipline for Software Engineering. Addison Wesley. Reading, Massachusetts. Primera Edición, 1995. ISBN 0-201-54610-8

vida humana en riesgo. Ivars Peterson en su libro *Fatal Defect*[2] hace una recopilación de numerosas fallas como esta que han puesto en riesgo la vida de seres humanos.

El descuido en el diseño de programas de cómputo trae como consecuencia resultados que en general son menos catastróficos pero bastante más caros. Durante muchos años se ha hablado de la crisis del software. Básicamente ésta se refiere a que, contrario a lo que inicialmente se esperaba, los costos de producción de software y el tiempo y esfuerzo invertidos en ellos ha aumentado vertiginosamente. Esto se debe tanto a que el tamaño y complejidad de los sistemas de cómputo se ha incrementado notablemente como a que la técnica se ha quedado rezagada de los requerimientos. Se habla, por ejemplo, de que el porcentaje de la productividad en los Estados Unidos de Norte América se ha incrementado menos que el porcentaje de inversión en tecnologías de la información. De hecho, el incremento en productividad medida en dólares de principios de siglo fue bastante mayor a la de finales de siglo. Enormes cantidades de dinero han sido invertidas en desarrollar programas de cómputo que nunca fueron terminados. La principal causa, el exceso de recursos requeridos en comparación con el presupuestado.

En la gran mayoría de los casos de fracaso estudiados se puede atribuir a un deficiente análisis y diseño del problema a resolver. En otras ocasiones la implementación técnica consideraba un plazo de recuperación menor al que el costo de cambio o mejora justificaba, resultando en soluciones que

---

[2] Peterson, Ivars. Fatal Defect. Times Books. Primera Edición, 1995. ISBN 0-8129-2023-6

con el paso del tiempo resultaron muy pobres o que incrementaron el costo total de poseer el sistema.

Este ejemplo lo tenemos muy patente hoy en dia con el llamado problema del año 2000. El problema del año 2000 resulta básicamente de sistemas viejos, pensados para durar tan sólo unos pocos años que han sido utilizados por encima de su vida planeada, por razones de recuperación del costo, y que debido a que se intentó ahorrar en el diseño en su momento, ahora es necesario invertir trillones de dólares a nivel mundial para solucionar este problema. Costo que en muchos casos es superior a la erogación que fue necesaria para la programación del sistema original.

En esta tesis describo un método que recientemente ha tenido mucha aceptación a nivel mundial para el análisis y diseño de sistemas llamado el método de Casos de Uso, desarrollado por Ivar Jacobson. La idea central atrás de la metodología de Jacobson es que el mayor costo de un sistema está no en la programación inicial, sino en las modificaciones que sufrirá posteriormente. Es por esto que si se desea lograr un sistema de cómputo robusto y con el menor costo, se debe diseñar para permitir modificaciones posteriores fácilmente. Si bien el modelo de casos de uso de Ivar Jacobson fue desarrollado en un ambiente de programación orientado a objetos, la técnica es adecuada para cualquier tecnología, ya que su fundamento está en el análisis y no en la programación.

Después de describir el método de Casos de Uso, describo la programación orientada a objetos, técnica en la que se basó el desarrollo de la metodología de Jacobson. Como un ejemplo de la programación orientada a objetos presento el lenguaje *Java*. Este lenguaje, de reciente creación, ha

tomado gran empuje debido a que se presenta no sólo como un lenguaje robusto, sino también como la primera posibilidad real de un lenguaje que produce programas que pueden funcionar en varias plataformas con pocas, o inclusive ninguna modificación.

Todos estos conceptos son ejemplificados con el análisis de una pequeña herramienta útil para el diseño de pruebas de fatiga en el laboratorio que podría ser incluida en la computadora que acompaña a las máquinas de prueba de fatiga de control numérico, como auxiliar en la preparación, documentación y análisis de pruebas de fatiga.

El diseño de bajo nivel y el código del programa no se incluyen ya que el alcance propuesto de esta tesis es ejemplificar el modelo de análisis de la metodología de Ivar Jacobson con un ejemplo práctico.

Debido a que el primer paso para desarrollar cualquier programa de cómputo es conocer a fondo el problema que se desea resolver incluyo un análisis muy detallado del fenómeno de la fatiga.

El análisis en el modelo de Jacobson es independiente del ambiente de implementación. El diseño, que es dependiente del ambiente, es desarrollado en el lenguaje *Java*.

2 El Método de Casos de Uso de Ivar Jacobson

2.1 El Desarrollo de Sistemas es la Construcción de Modelos

El desarrollo de un sistema es una tarea complicada. Diversos aspectos deben ser tomados en cuenta. Lo que se desea obtener es un programa de computadora que es confiable y que desempeña su tarea de manera apropiada. El desarrollo de sistemas es una tarea tan compleja que típicamente no se puede trabajar con todos los requerimientos simultáneamente. Pude darse el caso de que para programas muy pequeños se puedan tomar los requerimientos y escribir el programa directamente, pero esto es completamente imposible para los sistemas grandes. Lo que se debe hacer es administrar la complejidad de manera ordenada. Esto se logra al trabajar con diferentes modelos, cada uno enfocándose en algún aspecto del sistema. Al introducir la complejidad gradualmente y en un orden específico en modelos sucesivos se puede atacar la complejidad. La metodología de Ivar Jacobson reconoce cinco diferentes modelos[3]:

- El modelo de requerimientos intenta capturar los requerimientos funcionales del sistema.

- El modelo de análisis intenta dar al sistema una estructura robusta y modificable de objetos.

- El modelo de diseño intenta adoptar y refinar la estructura de objetos que la implementación específica utilizará

- El modelo de implementación implementa el sistema en un ambiente de implementación específico.

- El modelo de pruebas intenta verificar el sistema.

Cada modelo intenta capturar alguna parte o aspecto del sistema a construir. Estos modelos son el resultado de las actividades de análisis, construcción y pruebas.

La idea básica de estos modelos es capturar desde el principio todos los requerimientos funcionales desde el punto de vista del usuario. Esto se logra con el modelo de requerimientos. Aquí se describe cómo un usuario potencial utilizará el sistema. Este modelo es desarrollado frecuentemente con la participación cercana de usuario final del sistema. Cuando este modelo está terminado el sistema es estructurado desde un punto de vista lógico en una forma que es robusta y sobre todo modificable durante el ciclo de vida. Esto es logrado en el modelo de análisis. Aquí se asume un ambiente de implementación ideal; es decir, no se toman en cuenta consideraciones específicas como la base de datos a utilizar, el equipo, si el sistema será o no distribuido, requerimientos de funcionamiento en tiempo real, ... Existen dos razones para esto. La primera es que es mucho más fácil trabajar con circunstancias ideales, esto permite reducir la

---

[3] Jacobson Ivar. Object-Oriented Software Engineering, A Use Case Driven Approach. Addison-Wesley. 1992. pág. 113. ISBN 0-201-54435-0

complejidad y los esfuerzos requeridos para lograr una estructura lógica robusta y estable. La segunda razón es que el ambiente de implementación cambiará durante el ciclo de vida del sistema y no se desea que estos cambios afecten la estructura lógica del sistema.

Sin embargo, el mundo no es ideal. Cuando se ha adoptado una estructura ideal se adopta una estructura con los mínimos cambios posibles en el modelo de diseño. La razón es que se desea mantener una estructura que acepte cambios en el modelo de diseño. En este modelo se toman en cuenta todas las decisiones de implementación. Mientras que en el modelo de análisis casi nada puede ser implementado directamente, esto sí se puede lograr del modelo del diseño. Cuando todas las decisiones se han hecho y la aplicación se ha refinado y formalizado se trabajó el modelo de implementación. Este es el código comentado de la aplicación. Finalmente el modelo de pruebas es desarrollado para ayudar en la verificación del sistema. El sistema es refinado gradualmente al pasar de modelo en modelo.

Las relaciones existentes entre los modelos son por supuesto importantes. La transición entre modelos es automática. Esto significa que es posible prever cómo cambiar un objeto de un modelo a otro. Esto es absolutamente crucial para un proceso de desarrollo industrializado ya que las transformaciones deben ser repetibles. La habilidad de mantener el sistema es también indispensable. Para lograrlo, es necesario poder rastrear objetos entre modelos.

2.2 El Modelo de Requerimientos

La primera transformación hecha es desde la especificación de requerimientos hacia el modelo de requerimientos. El modelo de requerimientos consiste de:

- un modelo de casos de uso,
- descripciones de la interfase,
- un modelo del dominio del problema.

El modelo de casos de uso utiliza actores y casos de uso. Estos conceptos son una simple ayuda para definir lo que existe fuera del sistema (actores) y lo que el sistema ha de hacer (casos de uso).

Los actores representan lo que interactúa con el sistema. Un actor no es necesariamente una persona; puede ser otro programa, o en general cualquier otro dispositivo físico o lógico que interactúa en el sistema. Ellos representan todo lo se necesita para intercambiar información con el sistema. Ya que los actores representan lo que existe fuera del sistema, no se describen en detalle. Los actores no son iguales a otros objetos en la medida que sus acciones no son deterministas. Existen diferencias entre actores y usuarios. El usuario es la persona que utiliza el sistema, mientras que un actor representa un rol específico que la persona toma. Se puede considerar a un actor como una clase (en el sentido de la programación orientada a objetos) y los usuarios como instancias de esta clase. Estas instancias existen sólo cuando el usuario interactúa con el sistema. La misma persona puede aparecer como instancias de diferentes actores.

Una instancia de un actor lleva a cabo un número de operaciones diferen-

tes. Cuando el usuario utiliza el sistema, el o ella ejecuta una secuencia de transacciones en un diálogo con el sistema. Esta secuencia especial se llama caso de uso. Cada caso de uso es específico a la manera de utilizar el sistema y cada vez que se ejecuta ese caso de uso se puede pensar como una instancia del caso de uso. Cuando un usuario envía un estímulo, la instancia del caso de uso se ejecuta y comienza una transacción perteneciente a ese caso de uso. Esta transacción consiste en una serie de diferentes acciones a ejecutar. Una transacción está terminada cuando la instancia del caso de uso de nuevo se encuentra esperando un estímulo de la instancia de actor. La instancia de caso de uso existe mientras el caso se encuentra operando.

Estas instancias de casos de uso siguen, como todas las instancias de un sistema orientado a objetos, una clase específica. Cuando un caso de uso es ejecutado, podemos verlo como una instancia de la clase del caso de uso. Una clase de caso de uso es una descripción. Esta descripción especifica las transacciones del caso de uso. El conjunto de todas las descripciones de caso de uso especifica la funcionalidad completa del sistema.

Por lo tanto el modelo de casos de uso es descrito por un número de actores y de casos de uso. Para los casos de uso, se desarrollan descripciones detalladas. Cuando el sistema se encuentra en operación, se crean instancias de las descripciones de este modelo.

Cuando se desea cambiar el comportamiento del sistema, se remodelan los actores y casos de uso apropiados. Toda la arquitectura del sistema será controlada por lo que los usuarios desean hacer con el sistema. Si se

mantiene la rastreabilidad a través de los modelos, será fácil modificar el sistema conforme existan nuevos requerimientos.

Otra característica importante del modelo de requerimientos es que se puede discutir directamente con los usuarios y descubrir fácilmente sus requerimientos y preferencias. Este modelo es fácil de formular desde la perspectiva del usuario, por lo que fácilmente se puede hablar con ellos y descubrir si se está construyendo el sistema adecuado a los requerimientos. Ya que este es el primer modelo a desarrollar, se puede evaluar si los usuarios están satisfechos con lo que se está a punto de diseñar antes de construir el sistema.

Para soportar el modelo de casos de uso, es apropiado desarrollar las interfases de los casos de uso. Un prototipo de la interfase con el usuario es una herramienta perfecta. De esta manera se pueden simular los casos de uso para los usuarios mostrándoles las vistas con las que interactuará cuando el sistema esté en operación. Adicionalmente para comunicarse con los usuarios potenciales es comúnmente apropiado desarrollar un modelo lógico de objetos del dominio del problema. Este modelo consiste en objetos que tienen una contraparte directa en el dominio del problema bajo consideración y sirve para ayudar en el desarrollo del modelo de requerimientos.

El modelo de requerimientos pude considerarse como la formulación de las especificaciones de requerimientos funcionales basadas en las necesidades de los usuarios del sistema.

El modelo de casos de uso controlará la formación de todos los otros

modelos, se desarrolla en cooperación del modelo de objetos del dominio. La funcionalidad especificada por los casos de uso es estructurada en un modelo robusto, lógico e independiente de ambiente de operacíón y que es estable hacia los cambios llamado modelo de análisis. Este modelo de análisis se adapta a la implementación específica mediante refinaciones sucesivas del modelo de diseño utilizando los casos de uso para describir cómo las instancias de uso fluyen sobre los objetos del diseño. Los casos de uso se implementan en código a través del modelo de implementación. Finalmente los casos de uso dan una herramienta para probar el sistema, especialmente durante las pruebas de integración. Los casos de uso dan soporte adicional para escribir manuales de uso e instrucciones de operación.

2.3 El modelo de análisis

El modelo de requerimientos define las limitaciones del sistema y especifica su comportamiento. Cuando el modelo de requerimientos ha sido desarrollado y aprobado por los usuarios del sistema, se puede empezar a desarrollar el sistema en sí. Ese proceso comienza con el desarrollo de un modelo de análisis. Este modelo sirve para estructurar el sistema independientemente del ambiente de implementación. Esto significa que se enfoca la atención en la estructura lógica del sistema. Es aquí que se define la estructura estable, robusta y modificable que también es extensible.

Es posible utilizar el modelo desarrollado a partir de los requerimientos como base para construcción del sistema. De hecho, muchas metodologías orientadas a objetos utilizan modelos de requerimientos como única

entrada al proceso de construcción. Sin embargo, esto no resulta en la estructura más robusta para cambios futuros. El modelo de análisis representa en la experiencia de Ivar Jacobson una estructura del sistema más robusta y propensa a aceptar cambios que perdura a través de todo el ciclo de vida.

Debido a que muchos cambios futuros provendrán de cambios en el ambiente de implementación, estos cambios no afectan la estructuración lógica del modelo de análisis.

En el espacio de información se capturaran tres dimensiones: la información, el comportamiento y la presentación. La dimensión de la información especifica la información que tiene el sistema, tanto de corto como de largo plazo. En esta dimensión se describe el estado interno del sistema. En la dimensión del comportamiento se especifica la funcionalidad del sistema. Aquí se especifica cómo y cuándo debe cambiar de estado el sistema. La dimensión de la presentación provee de los detalles requeridos para presentar la información hacia afuera del sistema.

El modelo de análisis se construye especificando objetos en el espacio de información. Una posibilidad es la de tener objetos en los que sólo se expresa una dimensión. Este el caso de los métodos estructurados en funcionalidad y datos. La funcionalidad se coloca sobre la dimensión del comportamiento y los datos sobre la dimensión de información. Si se diseña de esta manera se obtiene un sistema que es muy sensible a las modificaciones ya que frecuentemente se debe modificar el comportamiento cuando se modifica la estructura de la información. Ya que esta situación no se desea, los objetos de los casos de uso deben situarse en cuando

menos dos dimensiones: la información y comportamiento y la de presentación cuando sea requerido.

Muchos métodos de análisis orientados a objetos escogen utilizar sólo un tipo de objetos que pueden ser utilizados en cualquier parte del espacio de información. La metodología de Ivar Jacobson sin embargo ha escogido utilizar tres tipos de objetos. La razón es simple: mantener una estructura que será más adaptable a los cambios. Los tres tipos de objetos utilizados son los objetos de entidad, objetos de interfase y objetos de control. Cada uno de estos tipos de objetos capturan cuando menos dos de las tres dimensiones del espacio, sin embargo, cada tipo tiene una inclinación particular a cada dimensión.

Se tienen tres propósitos diferentes para cada uno de los tres tipos de objetos. El objeto de entidad modela información del sistema que debe ser almacenado por un tiempo largo y generalmente debe subsistir después de que una instancia de caso de uso ha terminado. Todo el comportamiento asociado de manera natural a esta información también debe ser abstraído en el objeto de entidad. El objeto de interfase modela el comportamiento e información que es dependiente de la interfase con el exterior del sistema. Por lo tanto todo lo concerniente a la interfase es colocado en objetos de interfase. La interfase del sistema con el usuario es un ejemplo del tipo de comportamiento que se podría incluir en este tipo de objetos. El objeto de control modela la funcionalidad que no se encuentra relacionada de manera natural a los otros objetos, por ejemplo el comportamiento consistente en operar sobre diferentes tipos de objetos de entidad, hacer cálculos y después dar el resultado a un objeto de interfase.

La pregunta natural es ¿por qué estos tres tipos de objetos dan como resultado un sistema estable? La suposición básica es que todos los sistemas sufrirán cambios durante su ciclo de vida. Por lo tanto la estabilidad ocurrirá cuando todos los cambios que se realicen sean locales, esto es, que preferiblemente sólo afectan a un objeto del sistema. Los cambios más comunes a un sistema son los de funcionalidad y los de interfase. Los cambios a la interfase deben impactar sólo los objetos de interfase. Los cambios a la funcionalidad son más complejos. La funcionalidad pude ser colocada sobre cualquiera de los tres tipos de objetos. ¿Cómo puede limitarse la localización de los cambios? Si se trata de funcionalidad que está relacionada con la información del sistema, entonces los cambios sólo afectan objetos de entidad. Si los cambios son sobre la funcionalidad de la forma de representar la información, entonces los cambios sólo afectan objetos de interfase. Los cambios de funcionalidad de la interoperación de objetos se encuentran localizados en objetos de control.

Los diseños más estables no son aquellos que están desarrollados sobre objetos que corresponden a entidades de la vida real, característica que varias metodologías de diseño orientado a objetos enfatizan. La funcionalidad que se coloca en los objetos de control, pues, estará distribuida en varios objetos "de la vida real" haciendo que los cambios sean más dificiles de realizar.

Si la razón para utilizar tres tipos de objetos es la de localizar los cambios, entonces ¿cómo podemos saber si éstos son los objetos correctos? Desgraciadamente no se pueden hacer conclusiones sobre esto hasta que el sistema ha sufrido un número de cambios.

Existen otros modelos similares al presentado por Ivar Jacobson. El paradigma de Smaltalk llamado MVC[4] (*Model, View, Control*) es parecido, excepto que distingue entre las presentaciones de entrada y salida. Davis y Morgan[5] presentan un modelo de categorías separadas por niveles, cada una de ellas representa la funcionalidad, el proceso y los objetos de negocios que se pueden mapear a interfase, control, entidad en el modelo Jacobson.

El modelo de análisis está basado en el modelo de casos de uso. Cada caso de uso está dividido completamente en objetos de los tres tipos mencionados antes. En el modelo de requerimientos se especifica la funcionalidad completa del sistema. Esta funcionalidad debe ahora ser estructurada en un modelo robusto y susceptible a recibir cambios. En esta forma los casos de uso se parten en objetos de análisis. En la práctica esto significa que la funcionalidad especificada en los casos de uso debe ser asignada a diferentes objetos. De esta manera cada caso de uso será especificado por objetos en el modelo de análisis. Cada objeto, por supuesto, puede ser parte de dos o más casos de uso. De hecho esto es deseable, ya que genera objetos que son reutilizables que no sólo dan economías de escala, sino también facilitan los cambios posteriores. La transformación de casos de uso a objetos de análisis dan los cimientos

---

[4] Véase manual de referencia Smaltalk

[5] Davis J., Morgan T. Object-oriented development at Brooklyn Union Gas. IEEE Software, Enero 1993, pp 67-74.

para la arquitectura del sistema. Básicamente se particiona al caso de uso siguiendo tres principios:

1. Aquellas funcionalidades del caso de uso que dependen directamente del ambiente del sistema se colocan en objetos de interfase.

2. Aquellas funcionalidades que tratan sobre el manejo y almacenamiento de información que no se asocia naturalmente a objetos de interfase se coloca en objetos de entidad.

3. Aquellas funcionalidades específicas a uno o mas casos de uso y que no se asocian naturalmente a ninguno de los otros dos tipos de objetos se coloca en los objetos de control.

Al realizar esta división se obtiene una estructura que ayuda a entender al sistema desde un punto de vista lógico y da una estructura que da gran localización de los cambios y por tanto menos sensibilidad a modificaciones. Por ejemplo, cambios a la interfase, una modificación bastante común, afecta tan sólo a un objeto, el de interfase. Cambios al formato de datos afectan sólo a los objetos de entidad.

En la práctica la asignación y partición es bastante difícil. En esta etapa es cuando se debe decidir sobre la robustez contra la facilidad de cambio. El principio básico a seguir es la localización de los cambios. Generalmente es bastante difícil definir una frontera clara entre las funcionalidades de un sistema. En la práctica se fuerza a hacer juicios sobre dónde partir la funcionalidad entre objetos. Frecuentemente se debe pensar en términos

de cambios potenciales y cómo éstos pueden alterar la estructura.

El modelo de análisis está desarrollado directamente sobre el modelo de requerimientos. Este modelo de análisis formará las bases de una estructura específica del sistema pero sin hacer ninguna consideración sobre factores de la implementación.

2.4 El modelo de diseño

En el proceso de construcción se parte tanto del modelo de análisis como del modelo de requerimientos. Primero creamos un modelo de diseño que es una refinación y formalización del modelo de análisis. El trabajo inicial al desarrollar el modelo de diseño es adaptarlo al ambiente de implantación. El modelo de análisis que se desarrolló asume condiciones ideales; ahora debe ser adaptado a condiciones reales. Existen dos razones de peso para no introducir factores de implementación en el modelo de análisis:

No se desea que la implementación afecte la estructura básica del sistema ya que las circunstancias actuales seguramente cambiarán de alguna manera durante el ciclo de vida.

No se desea introducir demasiada complejidad al problema. De esta manera se pueden concentrar los esfuerzos en la parte esencial del problema: la estructura básica.

Por lo tanto, se puede considerar al modelo de diseño como la formalización del espacio de análisis, donde se adapta el modelo de análisis para que se ajuste al ambiente de implementación. El espacio de diseño es una

ampliación del espacio de análisis al que se le agrega una dimensión: el ambiente de implementación. Esta nueva dimensión significa que se introducen nuevos conceptos a los que el modelo debe adaptarse. El objetivo es refinar el modelo de análisis hasta que sea posible escribir código a partir de él. Debido a que el modelo de análisis tiene todas las propiedades que debe tener el sistema, se quiere conservar esta estructura como base del modelo de diseño. Sin embargo, posiblemente existan modificaciones cuando se introduzcan detalles de implementación como por ejemplo una base de datos relacional, requerimientos de desempeño, un lenguaje de programación específico, control de procesos concurrentes,... Esta es la razón por la que se desarrolla un nuevo modelo y no simplemente se continúa con el anterior.

La pregunta clásica a responder es ¿cuánto se debe trabajar en el modelo de análisis antes de pasar al de diseño?; en otras palabras, ¿cuándo está listo el análisis? No existe una respuesta universal a esta pregunta. Por un lado, se desea hacer cuanto trabajo sea posible en el modelo de análisis donde se centra en los problemas esenciales, pero por otro lado no deseamos hacer tanto trabajo que tenga que ser descartado cuando se introduzcan los factores de implementación. Lo que realmente se desea es una continuidad de refinamientos en los modelos y se hace el cambio del modelo de análisis al modelo de diseño cuando es imprescindible introducir factores específicos de la implementación.

Esto causa que se considere la pregunta de cuándo se debe cambiar el modelo de análisis mientras se trabaja en el modelo de diseño. Si el cambio en el modelo de diseño es causado por un cambio en la estructura lógica del sistema, tal como dos objetos que deben estar lógicamente

relacionados, entonces el cambio también debe ser hecho en el modelo de análisis. Si el cambio es consecuencia de la implementación, por ejemplo dos objetos que no deben comunicarse directamente debido a la estructura de proceso seleccionada, entonces esos cambios no deben incorporarse al modelo de análisis.

Las estructuras con las que se trabaja en el modelo de diseño son esencialmente las mismas que las del modelo de análisis. Sin embargo, ahora el punto de vista ha cambiado ya que este es un paso con vistas a la implementación. Por tanto se usa el concepto de bloque para describir la intención de producir código. Los bloques son los objetos del diseño. Un bloque normalmente intenta implementar un solo objeto del análisis. Aquí podría ser posible utilizar diferentes tipos de bloque (entidad, interfase, control) para ayudar a mantener rastreabilidad entre los modelos.

Es importante hacer notar que los bloques no son los mismos objetos que los del análisis. En el diseño se introducirán cambios, por ejemplo el partir un objeto en dos. Estos cambios no deben afectar al modelo de análisis ya que no se desea que las decisiones de implementación tengan efecto sobre el modelo ideal del análisis.

Otra diferencia entre los modelos de análisis y diseño es que el modelo de análisis debe ser visto como un modelo lógico y conceptual del sistema, mientras que el modelo de diseño debe acercarse más a la implementación física en código; se puede ver como un mapa hacia el código a escribir. Esto significa que se cambia la vista del modelo de diseño en una abstracción del código a escribir posteriormente. Por lo tanto el modelo de diseño debe ser un esquema de cómo debe estructurarse el código. Ya que

se desea una rastreabilidad fuerte y fácil de mantener desde los requerimientos hasta el código se trata de asignar los objetos del modelo de diseño al concepto de módulo del lenguaje con el que se esté trabajando.

Se ve a los bloques como una abstracción de la implementación del sistema. Los bloques agruparán el código. Para saber cómo implementar los bloques, se refina el modelo. Esto se logra describiendo cómo se comunican los bloques durante la ejecución. Para describir la comunicación entre los bloques se utiliza el concepto de estímulo. Un estímulo se manda desde un bloque a otro para gatillar la ejecución de ese bloque. Esta ejecución, por su parte puede generar estímulos a otros bloques.

Para describir la secuencia de estímulos se utilizan diagramas de interacción. En éstos, se describen como varios bloques de comunicación entre sí. Como una base para los diagramas de interacción se utiliza el modelo de casos de uso de nuevo. Se describe en detalle para cada caso de uso cómo y qué estímulos se enviarán y en qué orden. Se describe por lo tanto el caso de uso como una secuencia de estímulos enviados entre bloques.

Cuando se han descrito todas las secuencias, esto es, todos los casos de uso, incluyendo flujos alternativos y flujos de error, se han descrito todas las comunicaciones externas entre bloques. Desde aquí se ha obtenido una descripción completa de las interfases de todos los bloques. La notación[6] para describir los estímulos debe ser por tanto la misma que la utilizada

---

[6] Para un ejemplo de esta notación véase la obra de Jacobson *The Object Advantage*.

para el lenguaje de programación escogido. El modelo de diseño, por tanto, consiste en una descripción completa de todos los bloques con sus interfases.

Se mencionó que los bloques deben ser vistos como abstracciones del código a escribir y que es deseable que exista rastreabilidad entre los bloques y el código. Un problema en el proceso de construcción es que la complejidad se incrementa enormemente. Para dominar la construcción de sistemas es necesario manejar esta complejidad. Por tanto, se deben tener conceptos en el espacio de información que permitan manejar la complejidad de los sistemas construidos.

Para administrar esta complejidad se utiliza el concepto de subsistema. Un subsistema agrupa varios objetos. Los subsistemas pueden ser utilizados tanto en el modelo de diseño como en el modelo de análisis. Cada subsistema puede incluir otros subsistemas; el concepto es recursivo. De esta manera se tiene una estructura jerárquica de subsistemas, siendo el nivel más alto en la estructura el sistema. El sistema es la aplicación en si que se está construyendo. El sistema también describe las fronteras de la aplicación.

Durante la construcción se parte del modelo de análisis. Para cada objeto en el modelo de análisis se asigna un bloque en el modelo de diseño. Esta transformación ocurre de manera totalmente mecánica. Dependiendo del ambiente de implementación se deben hacer modificaciones tales que esta relación uno a uno se debe romper. En el modelo de diseño se formaliza el modelo de análisis y se adapta de modo que satisfaga el ambiente de implementación escogido.

Cuando se ha creado la estructura de bloques se dibujan diagramas de interacción para mostrar cómo estos bloques se comunican. Normalmente se dibujan diagramas de interacción para cada caso de uso. En la realidad el modelo de casos de uso prevalece y por tanto se puede garantizar que el sistema construido satisface los requerimientos del usuario.

2.5 El modelo de implementación

El modelo de implementación consiste en el código comentado. El espacio de información es el que el lenguaje utiliza. Es importante hacer notar que no se requiere un lenguaje orientado a objetos; la metodología de Ivar Jacobson puede ser utilizada con cualquier lenguaje ya que sólo da una estructura orientada a objetos. Sin embargo es recomendable utilizar un lenguaje orientado a objetos ya que los conceptos se codifican de manera más natural.

La base de la implementación es el modelo de diseño. Aquí se ha especificado la interfase para cada bloque y también se ha especificado su comportamiento de lo que existe detrás de la interfase.

2.6 El modelo de pruebas

El modelo de pruebas es el último modelo desarrollado durante el desarrollo del sistema. Dicho de manera simple, describe el resultado de probar el sistema. Los conceptos fundamentales en el modelo de pruebas son la especificación de las pruebas y el resultado de las pruebas.

Inicialmente, los niveles más bajos como los objetos y módulos son

probados de manera independiente. Estas pruebas son hechas por los codificadores y por los diseñadores. Después se prueban los subsistemas. Las pruebas de integración resultan de probar subsistemas cada vez más complejos. Una herramienta para las pruebas de integración es el uso de los casos de uso. Esto es generalmente realizado por un equipo independiente de pruebas. El modelo de requerimientos de nuevo es una herramienta poderosa. Cuando se prueban los casos de uso de hecho se prueba que los objetos se comuniquen de manera correcta. De manera similar se revisan las interfases descritas en el modelo de requerimientos. Por tanto se ve que el modelo de requerimientos es verificado por el proceso de pruebas.

También se pueden ver a las pruebas como objetos tal como se vieron los casos de uso como objetos. Al hacer esto se puede ver la especificación de pruebas como la clase de las pruebas, y por tanto podemos heredar partes comunes entre pruebas resultado en la reutilización de las especificaciones. Las instancias tienen por tanto comportamiento y estado. El resultado de ejecutar tal instancia es resultado de la prueba.

3 Programación Orientada a Objetos

3.1 Las Generaciones en los Lenguajes de Programación

Si miramos hacia atrás en la corta historia de la ingeniería de software, no se puede evitar observar dos grandes tendencias:

- el cambio de orientación de programación en pequeña escala a programación masiva
- la evolución de los lenguajes de programación.

La mayoría de los sistemas de cómputo de grado industrial actuales son más complicados que sus predecesores de tan sólo hace unos pocos años. El incremento en complejidad ha propiciado la investigación en la ingeniería de software, particularmente en los temas de la descomposición, la abstracción y la jerarquisación. El desarrollo de los lenguajes ha seguido la tendencia de aquellos programas que le dicen a la computadora qué hacer hacia lenguajes que describen las abstracciones clave en el dominio del problema.

Wegner[7] ha clasificado algunos de los lenguajes más conocidos en generaciones, tomando como base las características que poseen. La clasificación de Wegner fue publicada en la década de los 80.

Primera generación (1954-1958)

- *Fortran* - expresiones matemáticas
- *Algol* 58 expresiones matemáticas
- *Flowmatic* expresiones matemáticas
- IPL V expresiones matemáticas

Segunda Generación (1959-1961)

- *Fortran* II - Subrutinas, compilación separada
- *Algol* 60 Estructura de bloques, tipos de datos
- *COBOL* - Descripción de información, manejo de archivos
- *Lisp* - Procesamiento de listas, apuntadores, colectores de basura

Tercera generación (1962-1970)

- PL/1 *FORTRAN* + *ALGOL* + *COBOL*

---

[7] Wegner, P. 1980 Research Directions in Software Technology. Cambridge, MA: The MIT Press - July 1984. Capital-intensive Software Technology. IEEE Software vol. 1(3)

- *Algol* 68 Sucesor de *Algol* 60
- *Pascal* Sucesor de *Algol* 60
- *Simula* Clases, abstracción de datos

La clasificación de la mal llamada cuarta generación no puede ser atribuida a la clasificación original de Wegner. Se trata de un mote asignado a los lenguajes que se orientan a la descripción de la información y la interfase con el usuario. Como se verá en los siguientes párrafos, si algún tipo de lenguajes deben ser llamados de cuarta generación, éstos deben ser los basados en objetos y los orientados a objetos.

En el estudio de Wegner los lenguajes de las generaciones posteriores tienen mecanismos más avanzados de abstracción. Los lenguajes de la primera generación se utilizaron principalmente para aplicaciones científicas e ingenieriles. El vocabulario del dominio del problema es casi exclusivamente matemático. Los lenguajes como FORTRAN I se desarrollaron principalmente para liberar a los programadores de las dificultades del lenguaje ensamblador o del código de máquina. La primera generación representó la primera aproximación de los lenguajes hacia el dominio del problema y menos hacia el dominio de la computadora. En la segunda generación el énfasis fue hacia la abstracción de algoritmos. En estas fechas las computadoras eran más poderosas que las de la primera generación y por tanto, más problemas eran susceptibles de ser automatizadas, especialmente las aplicaciones de negocios. El enfoque era hacia decirle a la computadora qué hacer. Los lenguajes de la tercera generación como Algol 60 y Pascal se desarrollaron para obtener mayor abstracción de información. El programador podía describir el significado de tipos de

datos relacionados y dejar que el lenguaje de programación obligue a que se cumpla las decisiones de diseño.

En los años setenta existió una gran actividad en la investigación de los lenguajes de programación, lo que resultó en la invención de cerca de dos mil diferentes lenguajes y dialectos. La necesidad de escribir programas cada vez más grandes, mostraba los problemas de los lenguajes anteriores y se idearon mecanismos para solucionar los problemas encontrados. Muy pocos lenguajes sobrevivieron, sin embargo la mayor parte de los mecanismos nuevos sirvieron para desarrollar nuevos lenguajes. Surgió Smaltalk como un sucesor revolucionario de Simula, Ada como un sucesor de Algol 68 y Pascal.

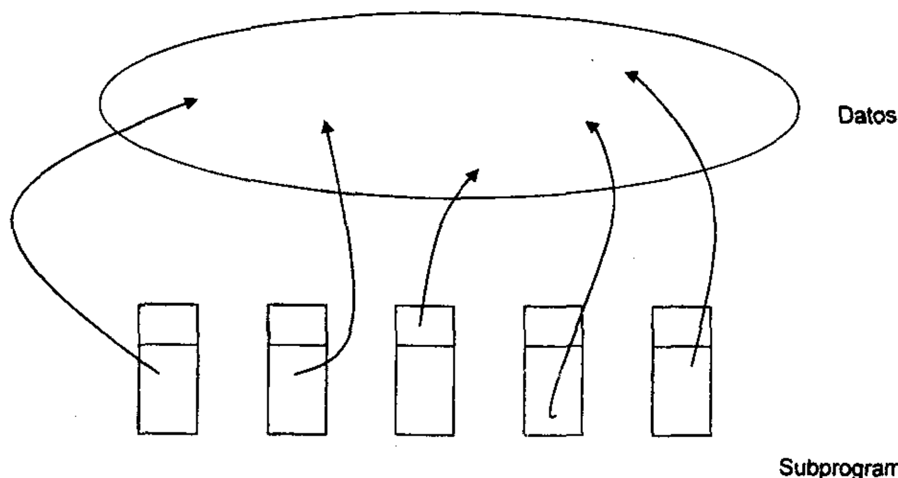


Figura 1. Esquema conceptual de la topología de los lenguajes de primera generación[8]

La figura uno muestra la topología de los lenguajes de la primera generación e inicios de la segunda. Por topología se entiende los elementos que provee el lenguaje y cómo estos elementos pueden ser conectados. En la figura se observa cómo en lenguajes del tipo de FORTRAN y COBOL el bloque fundamental de programación es el subprograma (o párrafo en COBOL). Los programas escritos en estos lenguajes tienen una estructura plana que consiste sólo de datos globales y subprogramas. Las flechas de la figura representan la dependencia de los programas hacia los datos. Durante el diseño es posible separar diferentes datos unos de otros, pero el lenguaje en sí no prevee mecanismos para hacer válidas las decisiones de diseño. Una falla en una parte del programa puede tener efectos

[8] Grady Booch. Object Analysis and Design with Applications. Benjamin Cummings. Second Edition. 1994. Redwood City, California. Pág. 30. ISBN 0-8053-5340-2

devastadores en el resto del código ya que los datos están expuestos a todo el programa. Cuando se hacen cambios a un sistema grande, es difícil asegurar la integridad del programa. Generalmente el desorden prevalece y después de un periodo de mantenimiento, el programa tiene una tremenda cantidad de referencias cruzadas propiciadas por el compartir de datos entre subprogramas poniendo en riesgo la claridad del programa y de las soluciones propuestas.

En los años sesenta se comenzó a reconocer a los programas como elementos importantes entre la computadora y el problema. La primera abstracción de software, ahora llamada la abstracción procedural nació debido a la manera pragmática de ver al software. Los subprogramas se inventaron antes de 1950 pero no fueron apreciados como abstracciones útiles sino que se veían como mecanismos para ahorrar trabajo. Poco a poco los subprogramas fueron apreciados como formas para abstraer la funcionalidad de los programas. Esta apreciación tuvo varias consecuencias: la invención de lenguajes que soportan medios para pasar parámetros, y más importante, se dieron las bases para el desarrollo de la programación estructurada. Los métodos de programación estructurada surgieron como solución al problema del desarrollo de grandes sistemas teniendo como bloque básico de construcción los subprogramas. No es sorprendente ver que la topología de los lenguajes de la segunda generación sea tan similar a la topología de la primera generación. Esta nueva topología resuelve el problema de la abstracción algorítmica pero no resuelve el problema de la programación a escala industrial.

A partir de FORTRAN II y después en los demás lenguajes de la tercera generación surgió otro mecanismo inventado para solucionar el problema

de la programación en gran escala. Los programas más grandes significan equipos de programación más grandes y por tanto, la necesidad de desarrollar diferentes partes del mismo programa de manera independiente y paralela. La solución a este problema fue el módulo compilado por separado, que surgió como un simple contenedor de datos y funciones. Los módulos rara vez fueron vistos como una abstracción importante. En la práctica fueron utilizados como simples agrupaciones de subprogramas que se esperaba cambiaran juntos. La mayoría de los lenguajes de la tercera generación proporcionan estructura modular, pero con pocos o ningún mecanismo para asegurar la consistencia en las interfases de los módulos.

La naturaleza de las abstracciones que pueden lograrse mediante el uso de procedimientos es adecuada para la descripción de algoritmos, pero no es adecuada para la abstracción de objetos. Esta es una desventaja importante ya que en la mayoría de las aplicaciones la complejidad de los objetos de datos constituyen una buena parte de la complejidad total del sistema. Como consecuencia surgieron los métodos para diseñar basados en los datos que proporcionaron una manera ordenada de lograr la abstracción de datos en lenguajes algorítmicos. Estas ideas aparecieron primero en el lenguaje Simula y se perfeccionaron en otros lenguajes como Smaltalk, Ada, CLOS y otros lenguajes híbridos como Object Pascal y C++. Estos lenguajes son llamados basados en objetos y orientados a objetos según sea el grado de abstracción logrado en el lenguaje. El bloque básico de construcción es el módulo, que representa un conjunto lógico de clases y objetos en vez de subprogramas. La programación orientada a objetos es un método de implementación en el que los programas están organizados como conjuntos de objetos que cooperan entre ellos, cada uno de los

cuales es una instancia específica de una clase que a su vez pertenece a una jerarquía. Si algún conjunto de lenguajes debe ser llamado de cuarta generación, éstos serían aquellos lenguajes que han logrado incorporar abstracción de datos, encapsulación, modularidad y jerarquía.

Existe una nueva abastracción que se está gestando hoy en día. Se trata de la abstracción del ambiente de operación. Si bien los lenguajes de programación de la primera generación lograron la abstracción del lenguaje del dominio del problema sobre el lenguaje de máquina. Sin embargo los programas desarrollados en estos lenguajes son dependientes del ambiente operativo que proporciona servicios adicionales, como la manipulación de archivos, almacenamiento externo, interfase con el usuario,... Una nueva generación de lenguajes abstraen el ambiente operativo, haciéndolos verdaderamente independientes de la plataforma de operación. Si bien hoy en día se percibe esta nueva abstracción como tan sólo un mecanismo para ahorrar trabajo en verdad representa una nueva abstracción. Uno de los primeros lenguajes que poseen esta nueva abstracción es *Java*. Si alguna categoría de lenguajes debe ser llamado quinta generación se trata de ésta.

3.2 El Lenguaje *Java* y la Programación Orientada a Objetos.

Los programas de computadora son simulaciones digitales de modelos conceptuales y físicos. Estos modelos son generalmente complejos por naturaleza. El objetivo del análisis y diseño es convertir esta complejidad a una forma comprensible tanto a los usuarios del sistema como a las

computadoras. Un estudio psicológico bien conocido[9] indica que el ser humano puede comprender a lo más de cinco a nueve secciones de información simultáneamente. Como resultado de esto, muchas interfases modernas abstraen la manera de presentar la información en conceptos como menús, escritorios o folders para poder aprovechar al máximo la atención y la posibilidad de compresión del ser humano. Esto es cierto tanto más para la programación. Cuando un programador escribe un programa, el número de variables e interrelaciones que debe manejar crece rápidamente con el tamaño del programa llegando a cantidades inmanejables por el cerebro humano.

Como se discutió en la sección anterior, este problema se observa con mayor prontitud en los lenguajes de primera y segunda generación, donde el programa se conceptualiza como una serie de pasos. Este modelo orientado al algoritmo es una manera de pensar en los pasos individuales de un problema sin modelar la abstracción del problema. Los lenguajes procedurales como PASCAL, C y FORTRAN han empleado el modelo con considerable éxito desde 1960. Aunque este modelo es muy adecuado para programas pequeños, se ha utilizado cada vez más para programas más grandes. La carencia de interfases e interacciones definidas entre elementos, combinada con la gran cantidad de elementos hace que un programa sea inmanejable y la probabilidad de cometer errores aumenta considerablemente.

---

[9] William James. Principios de Psicología, Biblioteca de Psicología y Psicoanálisis. Fondo de Cultura Económico. Pág. 322. México 1989. ISBN 968-16-2617-6

Para poder limitar el número de elementos simultáneos que el programador debe mantener en la mente simultáneamente, se inventó el modelo de objetos. El cerebro humano maneja la complejidad del mundo mediante la abstracción. El cerebro no mantiene el concepto de automóvil como un conjunto de miles de propiedades, si no que tan sólo se piensa en el objeto con sus características únicas. Esta abstracción permite poder subir a un automóvil y manejar hacia el destino deseado sin la necesidad de entender la complejidad del automóvil. Los detalles de operación del motor, la suspensión, el sistema de frenos, ..., pueden ser ignorados, mientras que la atención se concentra en un modelo abstracto del automóvil como un todo. Se posee una poderosa habilidad de manejar la abstracción en un esquema jerárquico que permite manejar la complejidad por niveles según sea necesario. Así, por fuera el automóvil es tan sólo un automóvil, mientras que por dentro se puede operar el sistema de impulso, de freno, de dirección, de acondicionamiento de la temperatura, el radio, el teléfono,... simultáneamente aunque la concurrencia está limitada de cinco a nueve conceptos. Por eso se debe tener cuidado: si demasiados conceptos comienzan a ocupar la atención, cuando se desea aumentar la funcionalidad, como cambiar de carril o de estación se debe cancelar la atención en otro concepto, o se debe afrontar la posibilidad de tener una falla catastrófica.

La idea de abstracciones jerárquicas de objetos y conceptos se puede aplicar de manera natural a los programas de computadora. Los programas algorítmicos tradicionales pueden ser abstraídos en componentes, objetos, que los procesos operan. Una secuencia de pasos de un proceso se convierte en una colección de mensajes entre objetos autónomos. Cada uno de estos objetos posee su propio comportamiento.

Se trata a estos objetos, aún a los abstractos, como entidades concretas con su correspondiente en el mundo físico, que responden a mensajes que implican instrucciones para hacer algo. Esta es la esencia de la programación orientada a objetos.

El mayor poder de la programación orientada a objetos radica en la habilidad de supervivencia de un programa ante los cambios que invariablemente sufre a lo largo de su vida.

Esta es la razón de existencia de la programación orientada a objetos: el manejo de la complejidad, especialmente las causadas por modificaciones posteriores, en los programas de computadora de tamaño industrial. De hecho, la programación orientada a objetos acarrea complejidades e ineficiencias innecesarias para programas extremadamente pequeños, que pueden ser mejor resueltos con lenguajes de tercera generación como C.

Debido a que se puede entender cabalmente la función de cada objeto dentro de una jerarquía, sin necesidad de preocuparse por los detalles de operación, y se tienen interfases claras y confiables entre objetos, se puede modificar parte de un sistema sin alterar el resto del sistema.

La academia ha enseñado a pensar en términos de objetos aún años antes de que existieran lenguajes orientados a objetos con la técnica llamada diseño de arriba a abajo. Se enseñaba a conceptualizar el problema de la manera más abstracta posible y definir posteriormente subrutinas que resuelven problemas concretos. Se enseñaba a evitar las variables globales y a pasar apuntadores a estructuras de datos como parámetros a

subprogramas.

Sin embargo, después de enseñar esto, se dota al aprendiz con lenguajes como C que le permiten al programador romper todas las reglas y por tanto es difícil estar seguro de los posibles efectos colaterales cuando se modifica una parte del programa. Los lenguajes orientados a objetos proveen mecanismos que fuerzan el cumplimiento de las reglas de objetos. Los mecanismos fundamentales son el encapsulamiento, la herencia y el polimorfismo. Si un lenguaje cumple estrictamente con la definición se llama lenguaje orientado a objetos. Si un lenguaje sólo soporta un subconjunto de estos conceptos o si soporta todo pero no los enforza estrictamente se llama un lenguaje basado en objetos. Si un lenguaje permite trabajar tanto con el modelo de objetos como con el modelo de la tercera generación (simultáneamente o no) se le llama lenguaje híbrido.

A manera de ejemplo *Simula* y CLOS son lenguajes basados en objetos; *Smaltalk* y *Java* son lenguajes orientados a objetos; C++ y *Object Pascal* son lenguajes híbridos

3.2.1 Encapsulación

Todos los programas, en su nivel más bajo constan de dos cosas: código y datos. En el modelo tradicional de programar, los datos son posicionados en memoria y son manipulados por subrutinas o funciones. El encapsulamiento del código que manipula los datos junto con la declaración y almacenamiento de los datos es la clave del diseño orientado a objetos.

La encapsulación es el empaquetamiento de código y los datos que manipula. Este empaquetamiento define el comportamiento y protege tanto al código como a los datos de ser accesado arbitrariamente por otra sección de código. El poder de la encapsulación radica en el conocimiento público sobre el comportamiento del objeto, sin exponer los detalles de implementación.

En *Java*, la base del encapsulamiento es la clase. Se crean clases que representan la abstracción de un conjunto de objetos que comparten la misma estructura y comportamiento. Un objeto es una instancia específica de una clase que posee la estructura y comportamiento definido por la clase como si estuviera estampada en un molde, sin embargo cada objeto es un ente independiente de los demás. Los objetos son en ocasiones conocidos como instancias de clases. La estructura individual de datos está definida por un conjunto de variables. Estas variables contienen el estado dinámico de cada instancia de la clase. El comportamiento y la interfase de una clase está definida por los métodos que operan en los datos de cada instancia. Un método es un mensaje que ordena a algún objeto a realizar una acción sobre sí mismo. Los mensajes se parecen a llamadas a subrutinas de los lenguajes procedurales.

Ya que el objetivo es encapsular la complejidad, *Java* provee de mecanismos para ocultar la implementación dentro de la clase. Cada método o variable en una clase puede ser identificado como público o privado. Una interfase pública de una clase representa todo lo que los usuarios externos de la clase necesitan saber. Se pueden declarar métodos o variables como privados si no deben ser accesibles desde afuera de la clase. Las interfases públicas deben ser escogidas con gran cuidado para

evitar exponer el funcionamiento interno de la clase. Ya que la encapsulación evita que se conozca el detalle de operación, se evitan efectos secundarios cuando se decide modificar el detalle de operación de un objeto.

3.2.2 Herencia

El ser humano percibe el mundo que le rodea como un conjunto de objetos que están relacionados unos con otros en una forma jerárquica parecida a la clasificación biológica de los seres vivos. Si se desea describir a un animal de una manera abstracta, se podría decir que tiene ciertos atributos como tamaño, inteligencia o tipo de esqueleto, por ejemplo. Los animales también tienen cierto comportamiento: comen, respiran, duermen... La descripción completa constituye la definición completa de la clase abstracta del animal.

Si se desea describir un tipo de animal más específico, tal como un mamífero se podría añadir atributos tales como tipo de dientes o el período digestivo. Esto es conocido como una subclase de la clase abstracta de animales y la clase de animales es conocida como la superclase de los mamíferos.

Ya que los mamíferos son tan sólo una especialización de los animales, la clase mamífera tan sólo necesita describir la interfase diferencial ya que hereda la descripción de la superclase. Una clase siempre hereda toda la funcionalidad y variables de todas sus superclases en la estructura jerárquica.

Cualquier clase puede tener cualquier cantidad de subclases. La estructura jerárquica de un lenguaje orientado a objetos puede permitir la herencia simple o la herencia múltiple. La herencia múltiple le permite a una clase tener dos o más líneas jerárquicas ascendentes. Los lenguajes orientados a objetos más puros como Smaltalk o *Java* proveen sólo herencia sencilla; otros lenguajes como C++ proveen herencia múltiple. Aún cuando la herencia múltiple aparenta tener poderosos beneficios, la implementación se puede convertir en un tema extremadamente complejo y por tanto propenso a errores. *Java* provee de un mecanismo simple y por tanto menos propenso a errores que permite simular los beneficios de la multiherencia.

3.2.3 Polimorfismo

Los métodos de los objetos son activados junto con un conjunto, posiblemente vacío, de datos llamados parámetros. Los parámetros son datos con los que se espera que el método realice su función.

En los lenguajes anteriores a la orientación a objetos si se deseaba realizar dos tareas diferentes se requería de dos funciones con nombres distintos. El polimorfismo permite que múltiples métodos compartan un mismo nombre. El método correcto a utilizar es seleccionado en tiempo de compilación mediante los parámetros del método. Además una clase puede redefinir un método heredado con los mismos parámetros que el original. A esto se llama sobrecarga de métodos. La determinación de qué método ejecutar en el caso de la sobrecarga se delega al tiempo de ejecución y por tanto permanece indeterminado en tiempo de compilación.

Algunos lenguajes permiten la sobrecarga de operadores. Si bien la sobrecarga de operadores. permite escribir código muy compacto, es un mecanismo muy complejo y propenso a errores. Además, la simple inspección del código no permite conocer la funcionalidad del método que utiliza al operador. *Java* no permite la sobrecarga de operadores, pero C++ si.

3.2.4 *Java* y C++

El problema con la mayoria de los lenguajes es que permiten escribir programitas rápidamente, con resultados satisfactorios pero a expensas de una estructura poco rigurosa. Los lenguajes como C, Pascal y FORTRAN han sido utilizados para crear algunos de los programas más grandes y complejos de la práctica moderna. La mayoría de estos programas escritos en los "lenguajes estructurados" se han convertido en entes que mejor pueden ser descritos como platos de espagueti por lo rebuscado y revuelto de la secuencia de instrucciones.

No sólo se necesitaba un nuevo lenguaje para solucionar el problema de la complejidad, sino que ese nuevo lenguaje debía implementar un nuevo estilo de programación: una mayor abstracción del problema.

En 1980, las universidades y centros de investigación de software estaban firmemente comprometidos con el lenguaje C debido al éxito obtenido con

el sistema operativo UNIX. Brian Kernighan y Dennis Ritchie[10] utilizaron C para escribir la mayor parte del sistema operativo y la totalidad de las herramientas de UNIX. Lo que el mundo necesitaba era una manera de migrar a los nuevos conceptos, pero sin perder la compatibilidad con la herencia de C. Bjarne Stroustrup[11] dio con la solución al inventar el lenguaje C con Clases, que después recibió el nombre más oscuro, si bien más ingenioso, de C++.

Una de las mayores críticas de C fue la cantidad tan enorme de funciones que se debían conocer para lograr resultados. La extraordinaria cantidad de funciones que se requerían en programas desde tamaño mediano, aumentaban innecesariamente la complejidad de los programas introduciendo la posibilidad de cometer errores. La programación orientada a objetos resuelve este problema y C++ fue un intento para implementar la solución, mientras que se conservaban las características del lenguaje C. Uno de los objetivos de diseño de C++ fue mantenerse estrictamente como un superconjunto de C. Todo compilador de C++ debería ser capaz de compilar código C. Es decir, se debían conjugar dos estilos antagónicos en un solo lenguaje.

---

[10] Kernighan Brian, Ritchie Dennis. El Lenguaje de Programación C. Segunda Edición. 1991. Prentice Hall Hispanoamericana. ISBN 968-880-205-0

[11] Stroustrup Bjarne. El Lenguaje de Programación C++. Addison-Wesley. Segunda Edición. 1993. ISBN 0-201-60104-4

Todos los promotores de C++ reconocen en la programación orientada a objetos la solución a los problemas de la tercera generación y por supuesto proclaman a C++ como la solución. Los promotores de *Java* argumentan que C++ no cumplió su promesa. La pregunta resultante es ¿qué es lo que está tan enteramente equivocado en C++ que causó que *Java* fuera inventado? La respuesta está en la compatibilidad con C.

La compatibilidad de C++ con C fue considerada como una ventaja. Un programa C es un programa C++ perfectamente válido; lo cual permitió que una gran cantidad de código C probado fuera utilizado en programas C++ nuevos, asegurando el éxito inmediato de C++. Sin embargo esto también aseguró la mezcla de dos estilos de programación incompatibles.

Java es mejor que C++ simplemente porque no requirió mantener compatibilidad con C y así logró una definición más pura del lenguaje. Los problemas de C que *C++* heredó se resumen en variables globales, *goto*, apuntadores, asignación de memoria, tipos de datos frágiles, conversión insegura de tipos, listas de parámetros inseguras, archivos de encabezados separados, estructuras inseguras y un preprocesador separado.

Las variables globales son herencia del tiempo de la programación en ensamblador. La idea básica de las variables globales es que cualquier subprograma puede modificar el estado global del sistema. El problema es que es posible escribir un subprograma que espera que alguna variable global permanezca constante durante algún periodo en el cual esto no sea cierto. Las variables globales de C++ muestran que el lenguaje no logra una encapsulación total del programa. En *Java* no existen ni las variables globales ni procedimientos fuera de alguna clase.

El uso de *goto* para controlar situaciones de error fue difundido, especialmente para termiar ciclos en condiciones de error, en C++. *Goto* permite la programación de código espagueti, que en ocasiones es necesario en C++ para tratar estas situaciones de error. *Java* no tiene la instrucción *goto*. El manejo de excepciones de *Java* es un mecanismo extraordinariamente poderoso para atender las situaciones de error.

Los apuntadores son la característica más poderosa y a la vez más peligrosa de C y C++. El uso inadecuado de la aritmética de apuntadores es la causa de la mayor parte de los errores en los programas de C y C++. El uso de apuntadores es la mejor manera para darle la vuelta a los mecanismos de control de tipos del lenguaje. Virtualmente todos los virus de computadoras aprovechan la posibilidad de cambiar arbitrariamente el contenido de la memoria a través de los apuntadores. *Java* no tiene apuntadores.

El manejo de la asignación de memoria, de la mano con los apuntadores, es una fuente importante de riesgos en los lenguajes C y C++. El manejo de la memoria de *Java* es superior al de C y C++ gracias a la incorporación del colector de basura, que se encarga automáticamente de liberar la memoria que ya no es utilizada.

Los tipos de datos de C y C++ están ligados directamente al tipo de máquina principal de la plataforma en la que se compila, haciendo que un mismo programa tenga diferente semántica en diferentes plataformas. Los tipo de datos de *Java* están fijos independientemente de la plataforma.

C y C++ permiten convertir cualquier tipo de dato en cualquier otro sin

importar si la información contenida es compatible o no. Así mismo los parámetros enviados a una función pueden ser convertidos a cualquier tipo. *Java* es un lenguaje que refuerza estrictamente los tipos de datos y sólo permite conversiones compatibles.

Aún cuando puede parecer que *Java* tiene menos características que C++, ninguna de estas características es en esencia siquiera deseable. De ninguna manera es *Java* un lenguaje menos poderoso que C++ y por el contrario sí es un lenguaje mucho más seguro.

4 El fenómeno de la fatiga

La Falla a Fatiga

4.1.1 Introducción a la fatiga

Desde hace ya muchos siglos, el hombre ha estado consciente de que puede romper la madera o el metal mediante dobleces repetidos de gran amplitud. Sin embargo, fue una verdadera sorpresa cuando se encontró que aún con cargas dentro del rango elástico también se produciría la fractura. Parece que las primeras investigaciones sobre la fatiga las realizó un ingeniero en minas alemán llamado W.A.S Albert[12], quien en 1829 realizó pruebas de cargas repetitivas en cadenas de hierro. Sin embargo, de las primeras fallas de fatiga que se encontraron fueron en los ejes de las carretas. Después, a mediados del siglo pasado, cuando los ferrocarriles se expandieron rápidamente, la falla de los ejes de los vagones se convirtió en un problema muy serio. Esta fue la primera vez que partes mecánicas muy similares fueron sujetas a millones de ciclos y a niveles de esfuerzo muy por debajo de su límite de cedencia. Los reportes de falla en servicio aparecían con una regularidad que empezó a preocupar. Como es frecuente en los casos de fallas no explicables, se llevaron a cabo experimentos que intentaban reproducir las fallas en el laboratorio. Entre

---

[12] Collins Jack. Failure of Materials in Mechanical Design, Analysis, Prediction, Prevention. Wiley Inter-Science. 1993. Pág. 179. ISBN 0-471-55891-5

1852 y 1870 el ingeniero de ferrocarriles alemán August Wöhler condujo el primer estudio sistemático de la fatiga[13]. Realizó pruebas con ejes de ferrocarriles y pruebas con modelos a escala bajo torsión, carga axial y flexión para varios materiales. Los resultados los presentó en un diagrama que relaciona el número de ciclos para la falla y el esfuerzo. Este diagrama, de Wöhler, comúnmente llamado diagrama S-N, se usa en la actualidad para mostrar la resistencia a la fatiga.

Casi al mismo tiempo, otros ingenieros se preocupaban con el problema de la falla debida a cargas repetitivas. Debido al rápido desarrollo de los ferrocarriles, los puentes de acero empezaban a remplazar a los de ladrillo y piedra, lo cual provocó que se pusiera en duda su seguridad. Debido a ello, se llevaron experimentos a escala natural de elementos remachados.

A medida que la tecnología avanzó a principios de siglo, se construyeron máquinas con elementos que trabajaban a gran velocidad como las turbinas de vapor. Posteriormente, la industria de la aviación requirió de avances en el estudio de la fatiga. Cerca de los años veinte, la fatiga era un tema de investigación a nivel mundial tanto micro como macroscópicamente. Algunos de los adelantos que han beneficiado mucho al avance del estudio de la fatiga son el microscopio electrónico y la teoría de las dislocaciones.

4.1.2 Modos de Carga de Fatiga

---

[13] Wöhler, Agust. *Versuche über die Festigkeit der Eisenbahnwagen-Achsen. Zeitschrift für Bauwesen*, 1860.

La carga estática, o cuasiestática es muy rara en la ingeniería moderna, por eso es esencial que el diseñador conozca las implicaciones que tienen las cargas repetitivas o fluctuantes. La mayoria de los proyectos de diseño en ingeniería mecánica involucran piezas que están sujetas a cargas fluctuantes o cíclicas. Estas cargas inducen estados de esfuerzo ciclicos que frecuentemente llevan a la falla por fatiga. Es de observarse que el término "fatiga", que fue acuñado hace más de un siglo, no es el mejor pues, muchas de las características del fenómeno físico no corresponden al fenómeno biológico. Por ejemplo, es difícil detectar los cambios progresivos en el material que ocurren debido a la fatiga, y por tanto, la falla se presenta con poca o ninguna anticipación. Además, los períodos de "descanso" no tienen ningún efecto sobre el material. Por lo tanto, el daño del proceso de fatiga es acumulativo y en términos generales irreparable.

La fatiga, aún cuando es un tema muy complejo, no ha sido descuidada por los investigadores. Nada más estar al tanto de toda la literatura publicada podría ser una tarea bastante larga. Sin embargo, el diseñador está siempre presionado para obtener más eficiencia, desempeño, velocidad, temperatura, menor peso, más vida, confiabilidad, ..., todo esto al menor precio y con un tiempo de producción razonable. Para lograr todas estas características se debe prevenir la falla por fatiga. Algunos de los problemas se pueden resumir como:

- Los cálculos de vida debida a la fatiga son menos exactos que los de resistencia mecánica. Es común encontrar errores de hasta varias órdenes de magnitud.

- Las características de fatiga de un material no pueden ser deducidas de otras características, sino que deben ser medidas directamente.

- Es común requerir de prototipos de tamaño real para pruebas y así poder asegurar una vida aceptable.

- Los resultados de diversas pruebas "idénticas" pueden variar mucho, por lo que se requiere de una interpretación estadística.

- Los materiales y el diseño deben ser seleccionados para dar una propagación lenta de las grietas, y si es posible, la detección de las grietas antes de que sean peligrosas.

- El concepto de "falla segura" debe ser incluido para lograr suficiente confiabilidad. Esto es que si un elemento de un sistema falla, el resto del sistema debe permanecer intacto y capaz de soportar la carga durante el corto plazo de la emergencia. Por lo tanto, la capacidad de inspección es una característica importante en el diseño de falla segura.

La investigación de la fatiga ha llevado a la observación de que el proceso de la fatiga abarca dos mecanismos diferentes. Un mecanismo de carga cíclica es en el que una deformación plástica significativa ocurre durante cada ciclo. Este mecanismo está asociado con cargas altas y vidas bajas, o pocos ciclos. Se le denomina fatiga de pocos ciclos. El otro mecanismo es en el que los esfuerzos están dentro del rango elástico. En este mecanismo las vidas son largas y se llama fatiga de muchos ciclos. El umbral entre ambos ciclos se encuentra alrededor de los 10 mil ciclos.

4.1.2.1 El espectro de la carga de fatiga

Cuando se diseña un elemento de máquina que está sometido a esfuerzos fatigantes, se debe tomar en cuenta la respuesta que puede tener a

diversas cargas que puedan sucederse durante la vida de la pieza, es decir el espectro de carga y de esfuerzo.

El espectro de esfuerzo más simple al que se puede someter una pieza es el patrón sinusoidal de promedio cero, con amplitud constante y frecuencia fija. Este tipo de patrón es frecuentemente llamado esfuerzo cíclico completamente revertido. Para definir estos esfuerzos se definen los siguientes símbolos:

σmax= esfuerzo máximo en el ciclo

σm= esfuerzo promedio = $\frac{\sigma_{max} + \sigma_{min}}{2}$

σmin= esfuerzo mínimo en el ciclo

σa= amplitud alternante de esfuerzo = $\frac{\sigma_{max} - \sigma_{min}}{2}$

Δσ= rango del esfuerzo = σmax - σmin

R = razón de esfuerzo = $\frac{\sigma_{min}}{\sigma_{max}}$

A = razón de amplitud = $\frac{\sigma_a}{\sigma_m} = \frac{1-R}{1+R}$

Cualquier combinación de cantidades definidas arriba, excepto la combinación de A y R o σa con $\Delta\sigma$ son suficientes para describir el patrón en discusión.

Un segundo tipo de patrón frecuentemente encontrado es el patrón de promedio diferente a cero. Un patrón como estos puede ser conceptualizado como la suma de un patrón completamente revertido superimpuesto a un esfuerzo estático con magnitud constante.

Un caso especial es cuando el esfuerzo mínimo σmin es igual a cero. Este tipo de esfuerzo generalmente es llamado de tensión liberada. El caso inverso es llamado compresión liberada.

4.1.3 Definición y Descripción

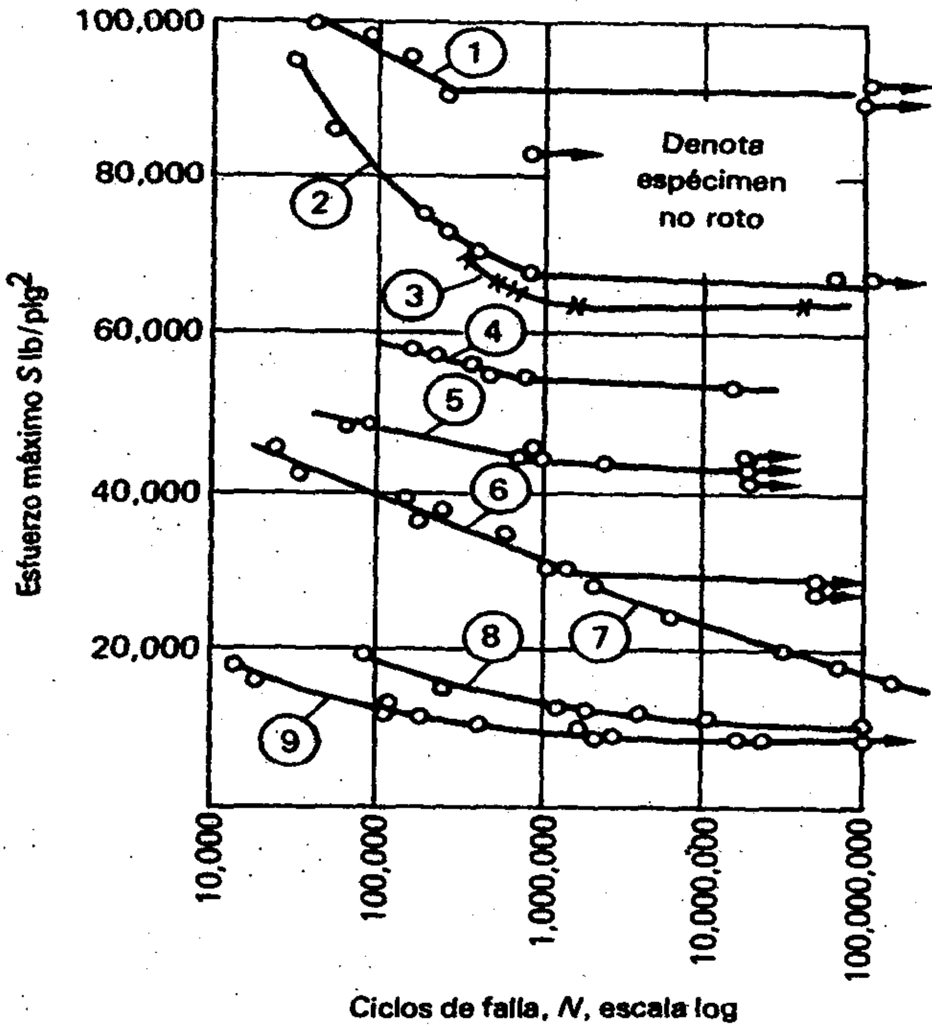
El límite de resistencia debida a carga cíclica es el esfuerzo máximo originado por carga completamente cíclica para la cual un material falla después de que el esfuerzo se ha repetido un número determinado de ciclos. Las fallas por fatiga son peligrosas porque tales fallas ocurren cuando no se está prevenido, son súbitas, sin "aviso" previo y a un nivel de esfuerzo mucho menor que el esfuerzo de cedencia.

Generalmente, la falla por fatiga tiene su origen en imperfecciones superficiales, cuyos orígenes pueden ser tan variados como: los métodos de fabricación, tratamiento térmico, condiciones ambientales, producción de la materia prima, esfuerzos residuales y recubrimiento superficial. En el proceso de la fatiga se pueden identificar tres etapas: nucleación de las

microgrietas, crecimiento de las grietas y la falla súbita.

El esfuerzo límite de fatiga es el esfuerzo máximo por completo reversible para el cual se supone que el material nunca fallaría, independientemente del número de ciclos aplicados. Generalmente se acepta y se está haciendo norma que los materiales ferrosos que por lo menos sobrevivan diez millones de ciclos de esfuerzo reversible tendrán vida infinita. Los materiales no ferrosos tales como el latón, cobre, y aleaciones de cobre, aluminio y magnesio no tienen definido un valor del esfuerzo último por carga reversible: en consecuencia, sólo se proporciona un valor de resistencia por carga reversible para estos materiales.

Se tienen tres tipos de pruebas de carga: cíclica, alternada o reversible. Estas son para torsión, tensión, flexión y combinaciones de las mismas. La prueba más común es la de flexión reversible, la cual se efectúa en la máquina R. R. Moore con una viga en rotación. Para realizar la prueba, la viga es cargada con un peso seleccionado. El espécimen gira junto con el eje del motor, pero la carga no gira. Las fibras longitudinales del espécimen cambian alternativamente de esfuerzo de tensión a esfuerzo de compresión en cada rotación. La rotación continúa hasta que ocurra la ruptura, anotándose para dicho tiempo el número de ciclos efectuados hasta la ruptura.



1. Acero 1.20% de carbón templado y estirado a 860°F 2. SAE 3420, templado y estirado a 1200°F. 3. Acero estructural de aleación. 4. SAE 1050, templado y estirado a 1200°F. 5. SAE 4130, normalizado y recocido. 6. Acero estructural ordinario. 7. Duraluminio. 8. Cobre de recocido alternado. 9. Hierro vaciado.

Figura 2 Tomada de Aaron Deutschman pág 120 (véase la bibliografía)

En la figura se observa una gráfica S-N en la cual se muestran los resultados de diferentes materiales probados. Se puede ver que la curva para el acero muestra un quiebre muy pronunciado (un codo) y muchos otros materiales (pero no todos) presentan estas regiones antes (o cerca) de los diez millones de ciclos. Esta es la razón por la que se considera de base a este número en el caso de esfuerzo último de fatiga para los materiales ferrosos (y el titanio) aún cuando muchos aceros tienen valores cercanos a los dos millones de ciclos.

Hay que notar que no todos los materiales presentan el codo. De hecho sólo los materiales ferrosos lo presentan, por lo cual no puede especificarse un esfuerzo último a la fatiga. Sin embargo, por cuestiones prácticas quinientos millones o mil millones de ciclos podrían considerarse como límites. De todas maneras, este criterio depende de la aplicación.

Se ha observado que existe una correlación pequeña entre el esfuerzo último de fatiga y algunas propiedades mecánicas tales como el esfuerzo de cedencia y la ductilidad. Existen pues, algunas relaciones empíricas entre el esfuerzo último de fatiga y el esfuerzo último a tensión para especímenes pulidos y sin muescas en prueba a flexión. Llamemos S_f al esfuerzo último de fatiga, para una probabilidad de supervivencia del 50% tenemos:

- S_f=0.5 S_{ut} para acero forjado con S_{ut} < 1.38 GPa y BHN < 400
- S_f=0.7 GPa para aceros forjados con Sut > 1.38 GPa
- S_f=0.4 S_{ut} para aceros y hierro vaciado
- S_f=0.38 S_{ut} para aleaciones de magnesio vaciado, y forjado (ciclos

de vida)

- S_f=0.45 S_{ut} para aleaciones de niquel base y para aleaciones de cobre

- S_f=0.38 S_{ut} para aleaciones de aluminio forjado con S_{ut} < 275 MPa (vida de 50^8 ciclos)

- para aleaciones de aluminio vaciado con S_{ut} < 345 MPa (vida de 50^8 ciclos)

Todo el análisis previo de fatiga está basado en pruebas de laboratorio con especimenes pulidos de tamaño y forma geométrica definida. Como consecuencia de esto, el diseñador deberá multiplicar su diseño por un valor que involucre los factores que afectan adversamente los resultados obtenidos bajo condiciones de laboratorio. De hecho la resistencia a la fatiga de un elemento basado en esfuerzos alternantes reales excede raras veces al 70% del valor indicado en los manuales. A continuación mencionaré los factores más relevantes.

La influencia del tamaño sobre los valores de la resistencia última de fatiga puede ser un factor importante. Aunque quizá no resulte obvio que deba haber algún efecto debido al tamaño, considerando que el diámetro en la sección más pequeña de una viga sujeta a la prueba de rotación es de 0.3 plg podemos dislumbrar la razón de que sí exista tal variación. Aunque se dispone de pocos datos se ha encontrado que los esfuerzos últimos de fatiga para un mismo material y dureza tienden a disminuir al aumentar el tamaño de los especimenes sujetos a flexión y torsión. La explicación de la disminución del esfuerzo último de fatiga es que al ser la pieza de mayor

tamaño es más factible que tenga defectos internos (por ejemplo inclusiones) que favorecerán a la aparición de microgrietas.

Para especímenes sujetos a flexión y torsión con diámetro mayor a 0.3 plg puede tenerse una reducción en su resistencia última a fatiga del 15% o más en especímenes de hasta 0.5 plg de diámetro. Para piezas mayores, la reducción permanece relativamente constante hasta para los diámetros cercanos a 2 plg después de lo cual empieza a aumentar. Con respecto a la reducción tenida en piezas mayores, no se pueden hacer generalizaciones, pero éstas pueden ser tan altas hasta de 30%. Las pruebas de laboratorio, curiosamente, indican que para carga axial no es necesario hacer la corrección por tamaño.

El método de fabricación tiene un efecto muy importante en las propiedades de fatiga de los materiales. Las diferentes técnicas de fabricación, tales como vaciado, forjado en caliente, formación en frío, torneado, rectificado, soldado, pulido, remachado, ..., contribuyen a alterar la resistencia de fatiga del material. Algunos de los efectos más marcados corresponden a la condición de acabado superficial o a la condición de un máquina o algún componente estructural.

Los trabajos de investigación sobre el efecto que tiene el acabado superficial con respecto a la fatiga y el acabado superficial, mostrados por Noll y Lipson,[14] dan curvas promedio de resistencia de fatiga contra resistencia a

---

[14] Tomada de Deutschman, pág. 126

la tensión para cada tipo de superficie estudiada; los resultados se utilizaron para trazar una serie de curvas que relacionan el esfuerzo de tensión para cada tipo de superficie como una función del porcentaje del esfuerzo último por cargas reversibles de un acero.

La soldadura, que constituye un método muy popular de fabricación particularmente con el empleo de máquinas automáticas para soldar, puede causar una gran reducción en las propiedades de fatiga de la pieza. La razón básica de esto es que la soldadura produce un cambio geométrico de la forma en la unión y por lo mismo actúa como un factor geométrico de concentración de esfuerzos.

Las causas de las concentraciones de esfuerzo son muy variadas. Son principalmente microscópicamente debidas a acabado superficial, inclusiones no metálicas; macroscópicamente se observan en los cambios de geometría debidas al diseño.

Una concentración de esfuerzo es cualquier condición que cause que el esfuerzo local sea mayor que el esfuerzo nominal. En otras ciertas condiciones es posible determinar un factor de reducción de la resistencia de fatiga y para otros sólo es posible confiar en la experiencia. La geometría o forma de la pieza es uno de los factores más importantes que contribuyen a la concentración del esfuerzo con bases muy racionales. En los casos en que no es posible el uso de técnicas analíticas se usan métodos experimentales (por ejemplo con fotoelasticidad o galgas extensométricas).

La definición matemática del factor de concentración de esfuerzos es K_f=(esfuerzo último por carga reversible en espécimen sin mues-

ca)/(esfuerzo último por carga reversible en espécimen con muesca) donde K_f es el factor de concentración de esfuerzos teórico.

Los factores mencionados anteriormente son de alguna manera u otra cuantificables. Existe una variedad de factores no cuantificables que tienen efecto sobre la resistencia a la fatiga.

Un ejemplo es la fatiga por rozamiento que ocurre cuando dos piezas apareadas, que están en contacto directo, sufren rozamiento entre ellas debido a vibraciones o a cargas repetidas. El rozamiento generalmente está confinado a áreas locales y contribuyen al deterioro de las superficies en contacto. Este deterioro puede ocasionar que las superficies dañadas empiecen a resquebrajarse. Esto conduce a una reducción de la resistencia por fatiga, y quizá a una fractura eventual. Las aplicaciones en las cuales se presenta el problema de fatiga por rozamiento y corrosión debidas a rozamiento son las referentes a ajustes por presión de diferentes elementos en flechas o pernos, flechas y juntas acuñadas, uniones atornilladas o remachadas, uniones con resortes, uniones acanaladas, baleros, ... Obsérvese que los ejemplos representan casos tales como ajustes por interferencia, uniones remachadas y atornilladas y baleros tanto de balines como de rodillos, en los cuales se tiene poco o algo de presión, pero que desarrollan presiones altas de contacto debido a cargas repetidas que conducen a fallas por rozamiento.

El tratamiento térmico adecuado puede ser muy benéfico en cuanto al mejoramiento de las propiedades de fatiga de un metal. En particular esto es una de las diferentes formas de introducir esfuerzos residuales los cuales, usados adecuadamente en el diseño, podrán tener efectos benéfi-

cos al reducir los daños debidos a la fatiga. Los esfuerzos residuales se clasifican en dos categorías: macroesfuerzos y microesfuerzos. Los macroesfuerzos son esfuerzos que pueden obtenerse cuantitativamente y están basados en el análisis elasto-plástico. Los microesfuerzos son aquellos esfuerzos que se relacionan con la estructura granular, su carga y deformación o desalineamiento. El proceso de templado usado en el tratamiento térmico puede introducir esfuerzos residuales en la superficie, ya sean de tensión o de compresión. De estos dos esfuerzos el que finalmente se tendrá dependerá de si el templado fue térmico o si fue tanto térmico como metalúrgico. Por ejmplo, suponiendo que la barra circular de acero mostrada en la figura ha sido calentada por encima de su temperatura de austenización. Entonces la pieza es templada y forma una estructura metalúrgica de acuerdo al templado (perlita, bainita o martensita[15]).

Analizando secuencialmente este proceso rápido, encontraremos que debido al enfriamiento, la superficie de la barra tiende a contraerse. Sin embargo, esto encuentra dificultades para efectuarse por lo caliente del núcleo. Se tiene entonces que momentáneamente la capa superficial se encuentra a tensión mientras que el núcleo está a compresión. Después, a medida que se endurecen las capas exteriores y se forman cristales, el esfuerzo es reducido un poco debido a que la transformación a martensita causa una ligera expansión en volumen. Para entonces, la superficie estará más fría; sin embargo, el núcleo sigue caliente pero continúa

---

[15] Véase Avner, Introducción a la Metalurgia Física. McGraw Hill. 1991. México. pág. 660. ISBN 968-6046-01-1

enfriándose, y por tanto, habrá contracción. Esta contracción del núcleo causa nuevamente contracción en la parte externa y tratará de reducir la tensión mientras que el núcleo esté en tensión. Por último, a medida que el núcleo adquiere la temperatura necesaria para la formación de martensita, éste se expande. Esta expansión actúa ahora sobre la capa externa, la cual está bastante dura y no se mueve fácilmente con el núcleo. Como resultado de esto, el esfuerzo en la capa exterior de la barra es cambiado a tensión o bien trata de reducir el esfuerzo de compresión resultante.

Por otra parte, si la barra ha sido calentada por debajo de su temperatura de austenitización[16], no se tendrá una transformación de fase asociada con un incremento de volumen al haber enfriamiento. Las fibras exteriores de la barra mostrada se enfriarán y tenderán a contraerse antes que el núcleo, este tiende a contraerse y hace que la capa exterior se comprima, porque esta capa ya se ha enfriado y está a temperatura ambiente. En consecuencia, el núcleo estará tenso.

En general, el templado independientemente de la temperatura inicial, debe efectuarse con mucho cuidado ya que de otra manera se tendrá una superficie alabeada o agrietada, lo cual sería muy perjudicial para el material cuando se le sujete a cargas de fatiga. También debe tenerse en cuenta que los esfuerzos nunca deben sobrepasar el límite de plasticidad, ya que si lo hacen el material se deformará y no quedarán esfuerzos residuales. Por último hay que tener en cuenta, especialmente para piezas

---

[16] Ibídem

diseñadas para mucha vida, que los efectos de esfuerzos residuales siempre son temporales. Diversos mecanismos metalúrgicos harán que los esfuerzos residuales desaparezcan eventualmente: el creep y el cambio de fase de baja velocidad.

Hay mucha evidencia que demuestra que el esmerilaje produce esfuerzos residuales de tensión desfavorables en la superficie de los aceros que hacen disminuir la resistencia a la fatiga.

El esmerilaje produce esfuerzos residuales de tensión desfavorables en la superficie de los aceros que hacen disminuir la resistencia a la fatiga. Sin embargo, mediante un esmerilaje cuidadoso en la dirección longitudinal de piezas redondas y planas no afecta adversamente las propiedades de fatiga. Para casos en los que el esmerilaje no se realizó con cuidado, el esfuerzo último de fatiga fue reducido casi en 25%. Sin embargo, el limpiado con perdigones logró recuperar la resistencia de fatiga original. Un pulido a mano en dirección paralela a la carga puede aumentar la resistencia a la fatiga, por la pequeña deformación en frío causada.

Hay varios procesos de trabajo en frío que se emplean para producir esfuerzos residuales de compresión en la superficie. Estos métodos, además del limpiado con perdigones y el pulido mencionados anteriormente, son: rolado superficial, martilleo neumático y estirado. Todos los métodos causan estiramiento de las fibras superficiales del material más allá del punto de cedencia pasándose a la región plástica. Las capas del material que están por debajo de la superficie, permanecen en la región elástica y regresan a su longitud original cuando termina la operación del trabajo en frío. Esto da esfuerzo de compresión a las fibras exteriores.

El perdigonado es realizado por dos tipos de máquinas. Una máquina arroja perdigones a alta velocidad, lo hace desde una rueda centrífuga y es capaz de arrojar un gran volumen de perdigones sobre la superficie que está siendo tratada. La otra máquina usa aire comprimido para arrojar los perdigones a alta velocidad. Este método es empleado con frecuencia para la limpieza de partes difíciles, como agujeros, y para piezas de fundición.

La roladura de la superficie mejora las propiedades de la fatiga y logra esfuerzos compresivos a profundidades mayores que el perdigonado.

La carburización es un proceso de endurecimiento superficial usado principalmente para resistir el desgaste o la abrasión y es producida en aceros de bajo carbón. Sin embargo, las propiedades de fatiga se benefician grandemente con la carburización. La nitruración produce algunos resultados similares a la carburización pero no produce la distorsión causada por un templado severo de la carburización porque no requiere tratamiento térmico después que el acero es calentado por abajo de su temperatura crítica. Sin embargo, la nitruración induce esfuerzos residuales altos junto con un incremento marcado de resistencia y es más caro que la carburización. Además no todos los aceros pueden ser nitrurados. Otros métodos de endurecimiento superficial son el cianurado, el carboniturado y el endurecimiento por flama o por inducción. Todos ellos mejoran las propiedades de fatiga.

4.1.4 Teorías del Fenómeno de la Fatiga

La fatiga se puede describir como un fenómeno progresivo de falla que tiene dos etapas: la iniciación o nucleación de microgrietas y la propaga-

ción hasta un tamaño inestable de las grietas. Cabe hacer notar que a la fecha no existe una teoría universalmente aceptado sobre la nucleación y propagación de las grietas, pero a continuación presento algunas de las teorías más aceptadas.

Se cree que el movimiento de las dislocaciones[17], que produce finas bandas de deslizamiento en la superficie de los cristales, es el mecanismo que permite la nucleación de las microgrietas. La aplicación de una carga estable que produce un esfuerzo cortante produce escalones en la frontera de grano debido a los deslizamientos en el orden de 10^{-6} a 10^{-7} m de altura. Estos escalones son denominados anchos. Los escalones que producen los deslizamientos generados por cargas cíclicas se denominan finos, pues tienen alturas típicas de 10^{-9} m, que son las regiones donde se inician las grietas de la fatiga. Las bandas de deslizamiento de fatiga dan origen a irregularidades superficiales llamadas protuberancias y grietas o ranuras como resultado del deslizamiento de planos de deslizamiento adyacentes debidos al revertimiento cíclico de la carga aplicada. Las protuberancias y ranuras pueden ser tanto afiladas, como dientes de sierra, o suaves que asemejan corrugaciones redondeadas. Si muchos planos se deslizan, las estrías resultantes son profundas y ondulantes, mientras que si sólo unos pocos planos cercanos se deslizan se forman paredes y grietas claramente definidas. En algunas ocasiones se puede encontrar extrusiones de material, que bien pudieron ser causadas por el deslizamiento cíclico. Una vez formadas, las grietas crecen en profundidad,

---

[17] Dieter, George. *Mechanical Metallurgy*. McGraw Hill *Materials Science & Metallurgy*. 1988. pp. 145. ISBN 0-07-100406-8

por el proceso de ciclos de deslizamiento y su crecimiento probablemente constituye la razón principal de la vida por fatiga del material.

Otra explicación para la iniciación del núcleo de fatiga está basada en la observación de que muchos circuitos de dislocaciones son producidas por cargas cíclicas. Se propone que la interacción de estos circuitos produce muchos huecos que se condensan en planos de deslizamiento en suficientes cantidades como para formar huecos estables en la estructura atómica del material. Aunque se han observado huecos en las bandas de deslizamiento y en las fronteras de grano, su mecanismo de formación se ha explicado solamente a nivel teórico. Estos huecos constituyen los núcleos a partir de los que se generan las grietas de la fatiga.

El crecimiento de los núcleos por carga cíclica ocurre generalmente a lo largo de los planos de deslizamiento colineales con la dirección del esfuerzo cortante máximo. Mientras la grieta crezca sobre un plano activo de deslizamiento no se observan cambios en este mecanismo de crecimiento. Esta etapa de crecimiento ha sido llamada, de forma arbitraria, etapa I de crecimiento. La etapa de crecimiento I, que puede ocupar tanto una pequeña parte de la vida como una larga parte, se ve favorecida por la aplicación de esfuerzos pequeños. Si existen periodos de alto esfuerzo o condiciones que den lugar a una relación de esfuerzo normal a esfuerzo cortante entonces se generan las condiciones de la etapa II de crecimiento.

La etapa II de crecimiento de las grietas no es gobernada por los esfuerzos cortantes locales, sino por el esfuerzo principal normal en la vecindad del vértice de la grieta. Debido a esto, el vértice de la grieta es desviado del plano de deslizamiento y se propaga en una dirección aproximadamente

perpendicular a la dirección del esfuerzo normal máximo. El crecimiento de la grieta en la etapa II se caracteriza por estrías que son llamadas marcas de playa o *beach marks*, que pueden ser relacionadas en su densidad y anchura con el nivel de esfuerzo aplicado. La superficie de fractura producida por la etapa II es relativamente suave.

Finalmente el tamaño de la grieta llega a una dimensión crítica y un ciclo adicional causa que la pieza falle. La fractura final típicamente muestra evidencia de la deformación plástica que se produce justo antes de la separación final: el área de la sección transversal se ve reducida debido al hueco producido por la grieta aumentando el esfuerzo que la pieza sufre más allá del límite del material.

Aunque el fenómeno microscópico es importante, el fenómeno macroscópico debe ser considerado en el diseño mecánico de una pieza. Los elementos que típicamente incluye el fenómeno macroscópico son:

- los efectos de un esfuerzo simple y cíclico completamente revertido en la resistencia y propiedades del material;
- los efectos de un esfuerzo constante cuando le es superimpuesto un componente alternante, es decir, los efectos de un esfuerzo cíclico con promedio diferente de cero;
- los efectos de un esfuerzo alternante en una situación multiaxial;
- los efectos de esfuerzos residuales, como los generados por el rolado en frío, por ejemplo;
- los efectos de los concentradores de esfuerzo, como filetes, juntas,

muescas,...;

- los efectos del acabado superficial;
- los efectos de la temperatura en el comportamiento de la fatiga;
- los efectos de la acumulación de ciclos con diferentes niveles de esfuerzo y la permanencia de estos efectos;
- la variación de la resistencia a la fatiga real de un material con respecto al promedio;
- los efectos de la humedad, un medio corrosivo y otros efectos ambientales;
- los efectos de interacción entre la fatiga y otros modos de falla, como el *creep* - deformación plástica acumulada causada por temperatura y esfuerzo -, la corrosión y el *fretting* - desgaste superficial de dos cuerpos en contacto.

Pruebas de fatiga en laboratorio

4.1.5 Introducción

El diseño de piezas para máquinas o estructuras que están sujetas a cargas fatigantes está basado en el resultado experimental de laboratorio donde se prueban especímenes del material a utilizar. Las máquinas de laboratorio para experimentos de fatiga pueden ser clasificados de la siguiente manera:

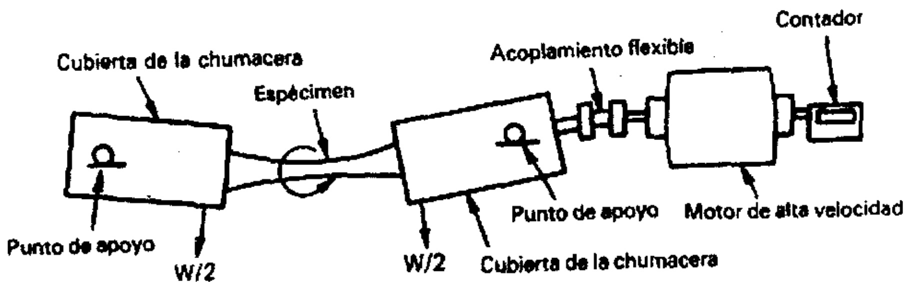


Figura 3. Esquema de la máquina de prueba R. R. Moore.

- *Máquinas rotatorias* de *flexión*, tanto de momento flexionante constante como de cantiliver;
- Máquinas de flexión reciprocantes;
- Máquinas de esfuerzo axial directo, tanto resonantes, como de fuerza bruta;
- Máquinas vibratorias, tanto mecánicas como electromagnéticas;
- Máquinas de torsión repetida
- Máquinas de control numérico
- Sistemas de escala completa

4.1.6 Las curvas S-N-P

Los datos de vida de fatiga pueden ser *mostrados* de manera muy práctica en una gráfica de nivel de esfuerzo contra el logaritmo de la vida en ciclos. Estas gráficas, llamadas curvas S-N, constituyen la información de

diseño para fatiga fundamental. Debido a que sólo se puede hablar de experimentos discretos de vida, no se puede hablar de una única curva S-N, sino de una familia de curvas cada una asociada a una probabilidad particular de que la falla ocurra. Estas curvas son llamadas S-N-P, o curvas S-N de probabilidad constante.

Las gráficas S-N distinguen dos tipos de comportamiento diferente, dependiendo del tipo de material analizado. Las aleaciones ferrosas y el titanio tienen una rama muy pronunciada en un rango de vida relativamente corto, y una segunda rama que se aproxima asintóticamente a un nivel de esfuerzo, conforme la vida se incrementa. Este nivel de esfuerzo es llamado límite de fatiga, y se define como el nivel de esfuerzo debajo del cual un elemento puede perdurar un número infinito de ciclos sin sufrir falla. Los compuestos no ferrosos no presentan esta asíntota y la curva de esfuerzo contra vida continúa cayendo indefinidamente. Para tales materiales no existe un límite de fatiga, y si se aplican suficientes ciclos, siempre existirá falla. Todos los materiales, sin embargo, presentan una curva de pendiente muy pequeña para el rango de vida larga.

Para caracterizar la respuesta a la falla de los materiales, tanto ferrosos como no ferrosos, a un nivel de vida finito, se usa el término esfuerzo de fatiga a una vida específica, SN. El término "esfuerzo de fatiga" identifica el nivel de esfuerzo al cual ocurrirá (con una cierta probabilidad) la falla a una vida específica. La especificación de un esfuerzo de fatiga sin especificar la vida no tiene ningún sentido. La especificación de un límite de fatiga siempre implica una vida infinita. Para efectos prácticos, vidas arriba de 10^9 ciclos se consideran vida infinita.

4.1.7 La influencia de un promedio de esfuerzo diferente a cero

La información básica recolectada en el laboratorio de pruebas de fatiga es para el caso de esfuerzos cíclicos de promedio cero. Sin embargo casi todas las aplicaciones requieren que los elementos sean diseñados para trabajar con cargas con promedio diferente de cero. Por lo tanto conocer la influencia que tiene la presencia de un esfuerzo estático sobre uno cíclico es de suma importancia, si se pretende utilizar información básica.

La siguiente gráfica muestra datos obtenidos experimentalmente donde se relaciona el promedio del esfuerzo contra el esfuerzo límite a fatiga. La influencia de un promedio de esfuerzo diferente de cero es distinto según se trate de un promedio de tensión o de compresión. Con un esfuerzo promedio de tensión el esfuerzo límite a fatiga es mucho más sensible que con un promedio de compresión.

Cuando se ha realizado experimentación con diferentes niveles de esfuerzo compresivo o tensil, esta información se presenta en diagramas maestros para el material (como el de la figura 5). Si la información no está disponible en un diagrama maestro, entonces se debe estimar la influencia del promedio distinto de cero mediante relaciones empíricas. Históricamente se han intentado diversas maneras de relacionar σ_a con σ_m. Los intentos más exitosos han resultado en cuatro diferentes relaciones[18]:

---

[18] Collins, pág. 230

- Relación lineal de Goodman
- Relación parabólica de Gerber
- Relación lineal de Soderberg
- Relación elíptica

Las relaciones se expresan con las siguientes ecuaciones.

- Relación lineal de Goodman: $\frac{\sigma_a}{\sigma_N}+\frac{\sigma_m}{\sigma_u}=1$
- Relación parabólica de Gerber: $\frac{\sigma_a}{\sigma_N}+\left(\frac{\sigma_m}{\sigma_u}\right)^2=1$
- Relación lineal de Soderberg: $\frac{\sigma_a}{\sigma_N}+\frac{\sigma_m}{\sigma_{yp}}=1$
- Relación elíptica: $\left(\frac{\sigma_a}{\sigma_N}\right)^2+\left(\frac{\sigma_m}{\sigma_u}\right)^2=1$

El criterio más conservador es la relación de Soderberg, mientras que el criterio más laxo es la relación elíptica; sin embargo ninguna de estas relaciones modela exactamente la realidad. Existen otras relaciones no lineales, sin embargo, además de ser más complejas, algunas requieren de constantes de material, por lo que en la práctica la relación más utilizada es la lineal de Soderberg. Collins[19] indica que el mejor criterio para uso

---

[19] Ibidem

general es la relación modificada de Goodman. El uso del esfuerzo último en vez del esfuerzo de cedencia, es a consideración del que escribe, demasiado laxo, ya que en la mayoría de las aplicaciones una pieza ha fallado cuando se excede su límite de cedencia y se deforma plásticamente, aún cuando no haya todavía fallado a fatiga o a tensión.

4.1.8 Procedimientos de Prueba[20]

Se han desarrollado diversos procedimientos de prueba de fatiga con el objeto de proveer información utilizable para diversos objetivos. Por ejemplo, si se desea obtener información sobre la distribución de la vida a un nivel de esfuerzo constante; o quizá la distribución del límite de fatiga a una vida determinada.

Los métodos que a continuación describo son apropiados para todos los tipos de prueba, tanto de laboratorio como industriales, sin importar el tipo de máquina utilizada.

[20] Las figuras de esta sección fueron tomadas de Collins Jack. Failure of Materials in Mechanical Design, Analysis, Prediction, Prevention. Wiley Inter-Science. 1993. ISBN 0-471-55891-5 Capítulo 3.

4.1.8.1 Método estándar

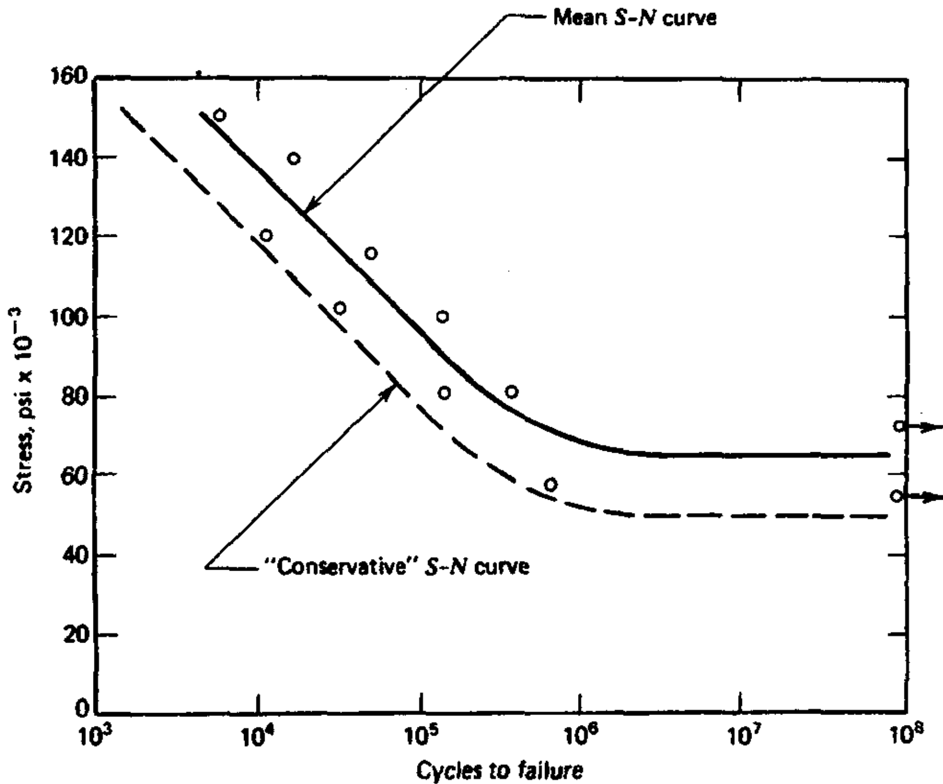


Figura 4. Curvas S-N resultantes de la prueba de fatiga "estándar".

El método llamado estándar es útil cuando sólo se dispone de unos cuantos especímenes y se requiere un estimado de toda la curva S-N. Este método propone el probar uno o dos especímenes a varios niveles de carga cíclica, midiendo la amplitud de la carga y el número de ciclos en la falla. Si un espécimen sobrevive a la prueba, entonces puede reutilizarse en otra prueba con un esfuerzo superior. Sin embargo en estos casos se debe

tomar en cuenta que el espécimen puede contener daños internos debido a la prueba a la que sobrevivió por lo que el resultado final puede verse afectado. Los datos después son graficados en un plano S-N convencional. Es de esperarse obtener sólo datos dispersos, por lo que surge la pregunta de cómo construir una curva útil a partir de ellos. Existen básicamente dos formas. Una es construir una curva promedio y la otra es construir una *curva conservadora*, que va por debajo de los puntos. La curva promedio generalmente representa un estimado razonable de la probabilidad real de que una pieza sobreviva en 50% de las veces. Si se estima una desviación estándar, se puede construir la familia de curvas S-N-P. Sin embargo debido a la escasez de datos no se puede obtener buena información estadística. La curva conservadora no tiene interpretación estadística.

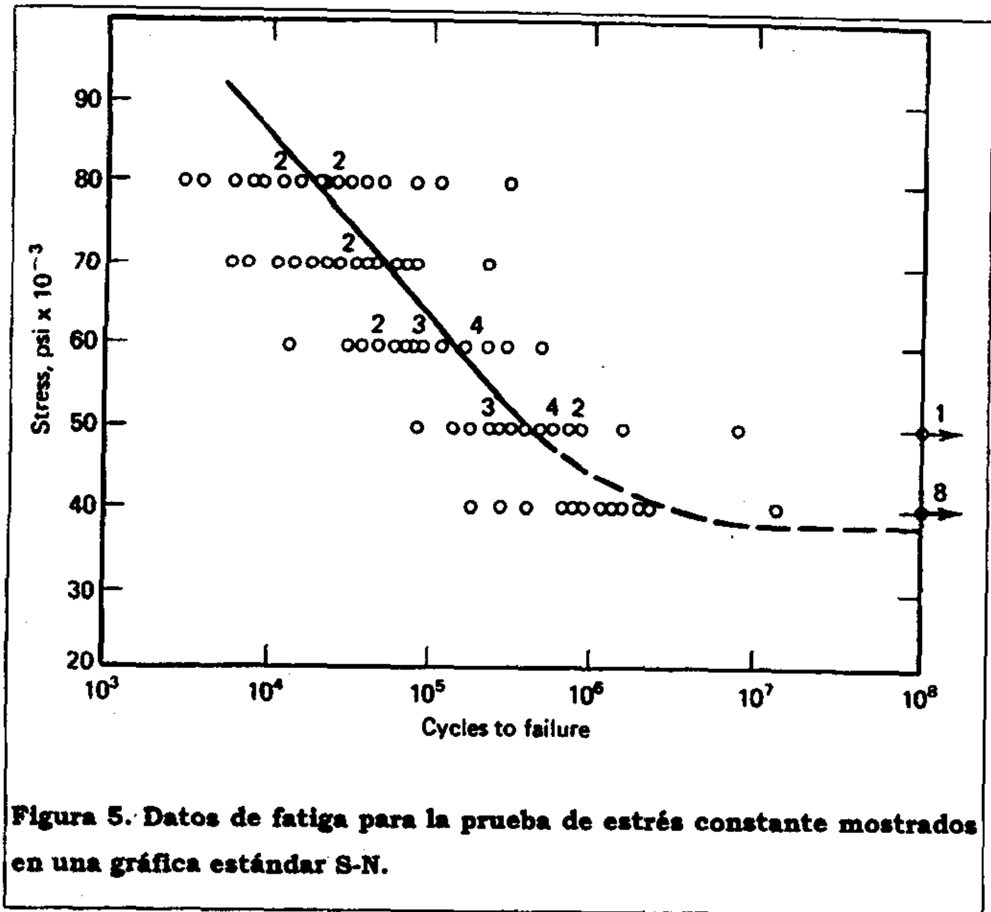


Figura 5. Datos de fatiga para la prueba de estrés constante mostrados en una gráfica estándar S-N.

4.1.8.2 Método de la prueba a esfuerzo constante

Para realizar la prueba del esfuerzo constante se someten 15 o más especímenes a cuatro o más niveles constantes de esfuerzo que están dentro del límite de fatiga y el límite de cedencia del material registrando la vida del espécimen en cada prueba. Heurísticamente se sabe que una buena aproximación a la distribución de la vida a fatiga a esfuerzo constante

superior al límite de fatiga es semi-logarítmica. Por lo tanto, se grafican los datos obtenidos en un plano semilogarítmico de probabilidad para verificar la distribución y para determinar el promedio y la varianza a un nivel de esfuerzo determinado. Esta aproximación es más exacta según se aumente el nivel de esfuerzo aplicado y es completamente inválido cerca del nivel de esfuerzo correspondiente a la resistencia a fatiga. Esto es así debido a que cerca del esfuerzo de resistencia a fatiga los datos no son homogéneos, pues se presentan fallas y supervivencias, y por tanto este método no es recomendable para usarse cerca del límite de fatiga. Sin embargo a niveles más altos de esfuerzo este es un buen método para generar la familia de curvas S-N-P.

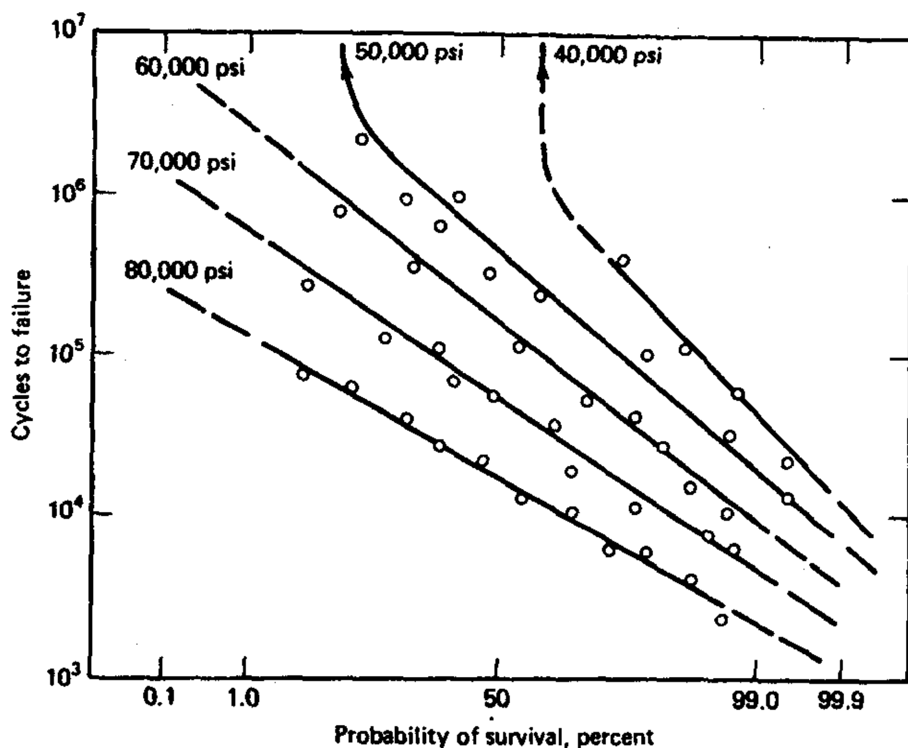


Figura 6. Datos de fatiga para la prueba de esfuerzo constante mostrados en una gráfica de probabilidad log-normal.

4.1.8.3 *Método Probit*

Anteriormente se aseveró que el método de esfuerzo constante no es bueno si lo que se desea es un método estadístico para establecer el límite de fatiga. El método Probit, por el contrario es adecuado para estos propósitos. Este método es llamado también método de supervivencia. Esta técnica puede ser utilizada para determinar el promedio y la varianza tanto del límite de fatiga como la resistencia de fatiga para alguna vida

predeterminada.

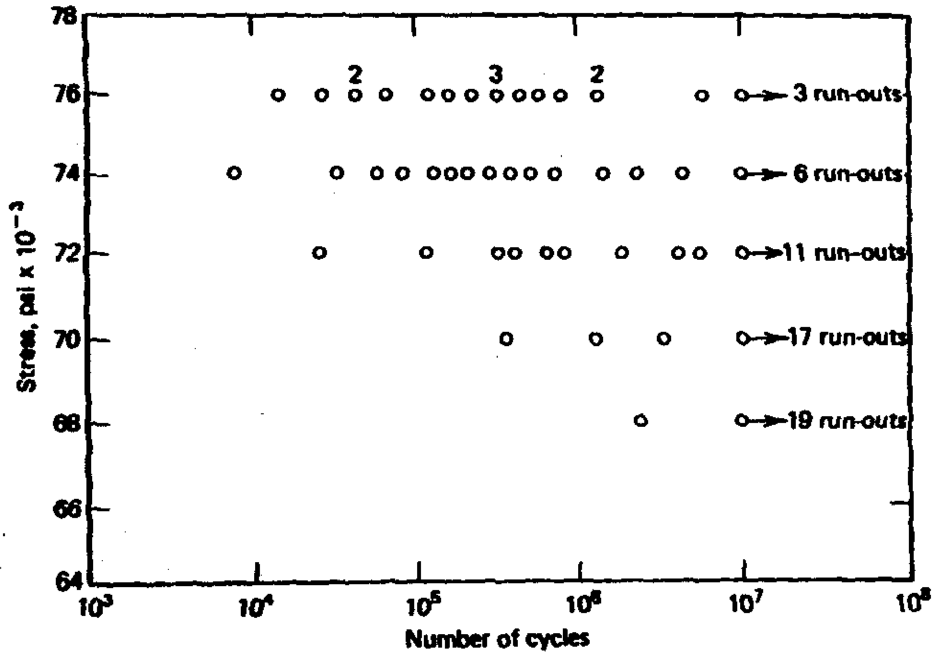


Figura 7. Gráfica S-N mostrando los resultados de una prueba de supervivencia para determinar el límite de fatiga.

El método Probit se basa en probar varios grupos de especímenes en un rango de esfuerzos que va desde aproximadamente dos desviaciones estándar por arriba y por abajo del límite de fatiga. Por ejemplo, para una pieza determinada una prueba (probablemente a esfuerzo constante) revela que el límite de fatiga está alrededor de 496 MPa. Una vez conocida la primera aproximación a la resistencia a fatiga se podrían escoger, por ejemplo, cinco niveles de esfuerzo que van desde 468 MPa a 524 MPa en intervalos de 14 MPa. Si se prueban 20 especímenes en cada nivel de

esfuerzo seleccionado se podrían dibujar los resultados en una gráfica S-N. Es de esperarse que a los niveles más bajos varios especímenes sobrevivan a la prueba, mientras que en los niveles más altos, sólo algunos sobrevivan a la prueba. Huelga decir que la elección de los límites y del rango es de suma importancia. Si son seleccionados correctamente la prueba arrojará buenos resultados. Si los intervalos son muy grandes, muchos grupos tendrán solamente supervivencias, mientras otros grupos tendrán sólo fallas.

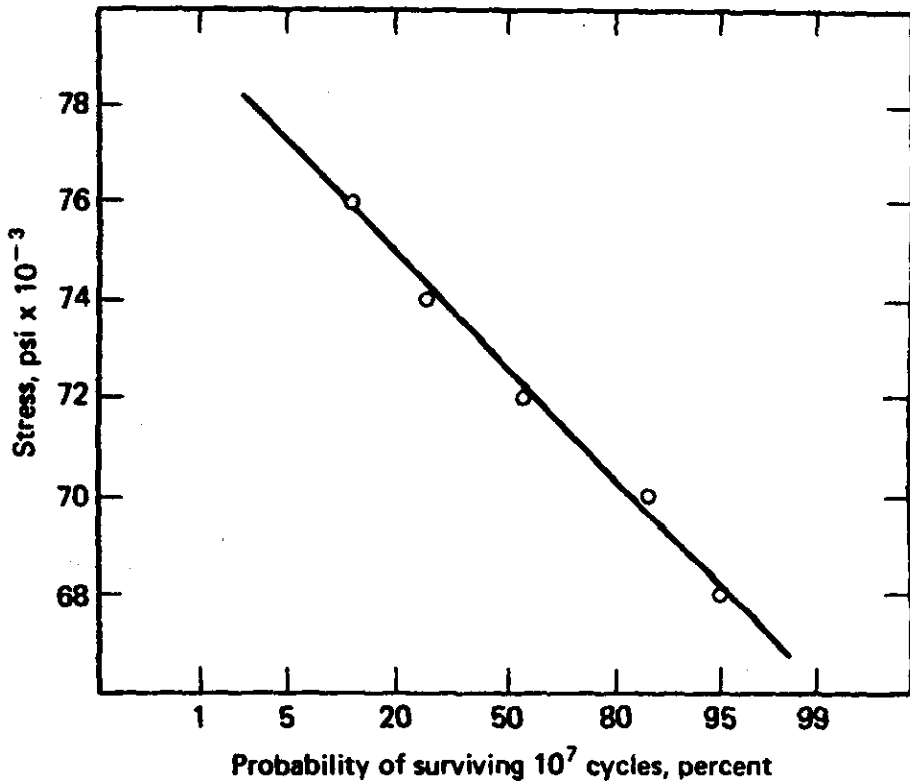


Figura 8. Datos de la prueba de supervivencia graficados en una gráfica normal de probabilidad.

Los resultados pueden correlacionar la probabilidad de supervivencia con el nivel de esfuerzo, quizá con una recta de mínimos cuadrados. De esta interpolación se puede obtener la media y la desviación estándar.

4.1.8.4 Método de Prueba a Pasos

Una clara desventaja del método Probit es que requiere una gran cantidad

de especímenes, típicamente entre 60 y 100. Esto hace que el método Probit sea muy costoso y requiera de mucho tiempo.

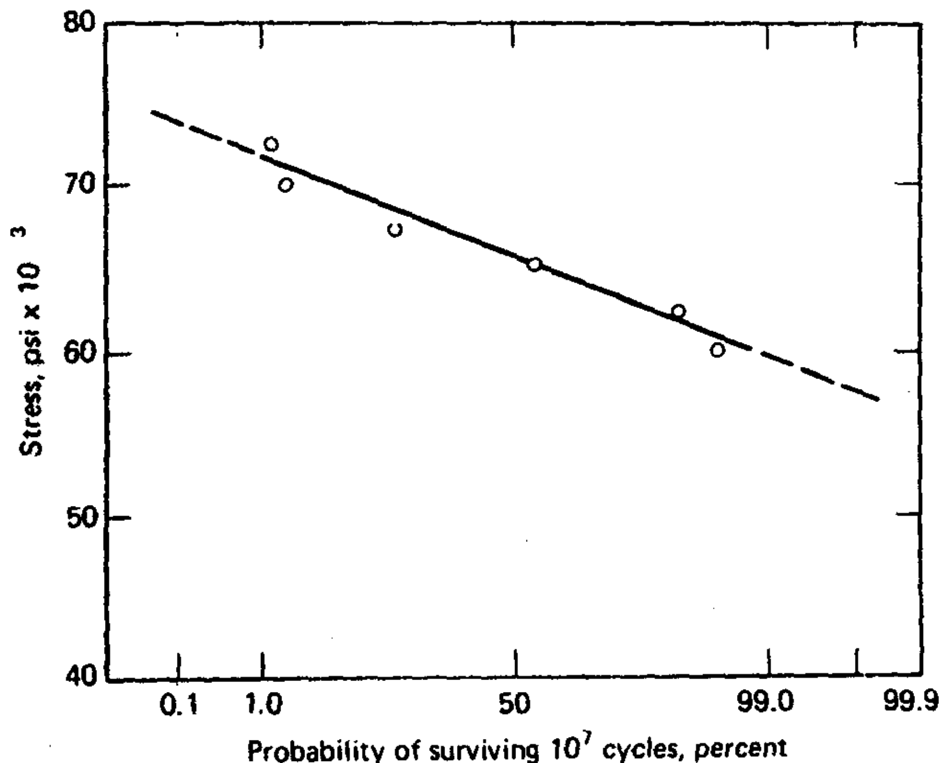


Figura 9. Resultados de la prueba de paso a paso graficado en una gráfica de probabilidad normal.

Un método alternativo, llamado el método de prueba a pasos es una prueba que fuerza a todos los especímenes a fallar. El método se basa en probar cada espécimen en un grupo de niveles de esfuerzo una cantidad predeterminada de ciclos en una serie progresiva de esfuerzos hasta que falle. El riesgo de este método es el desconocimiento de los efectos de daño

ESTA TESIS NO DEBE
SALIR DE LA BIBLIOTECA

latente que se incurre al hacer pasar el espécimen por varios niveles de esfuerzo.

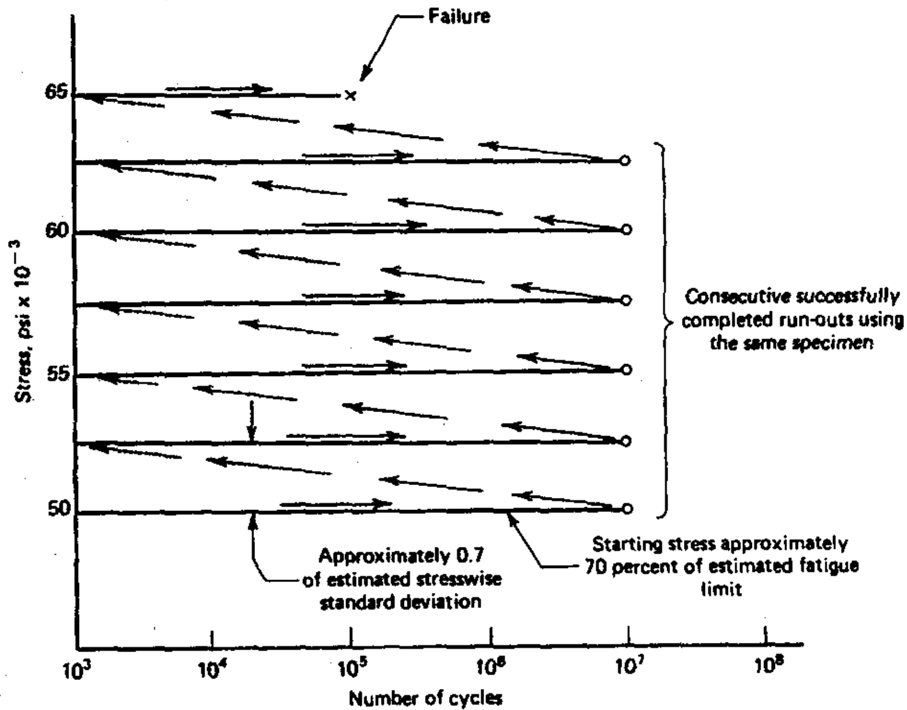


Figura 10. Gráfica que muestra la naturaleza progresiva de la prueba de paso a paso.

Para comenzar una prueba a pasos, se selecciona un nivel de esfuerzo que sea aproximadamente el 70% del límite de fatiga estimado. El espécimen se prueba a ese nivel de esfuerzo hasta que falle o sobreviva un número predeterminado de ciclos, por ejemplo 10^7 ciclos. Si ocurriese la falla antes del número predeterminado, se registra el nivel de esfuerzo y el número de ciclos para la falla. Si el espécimen sobrevive, entonces se eleva el nivel de

esfuerzo aproximadamente 0.7 de la desviación estándar estimada y el mismo espécimen vuelve a probarse en este nuevo nivel de esfuerzo. El procedimiento se repite hasta que el espécimen falla. Se requieren de 10 a 15 especimenes como mínimo para que la prueba dé resultados útiles.

Cuando la prueba ha concluido se pueden correlacionar los resultados de manera normal en el esfuerzo *contra* la variable probabilidad de falla. De esta correlación es posible obtener la resistencia a la fatiga promedio y la desviación estándar.

4.1.8.5 Método Prot

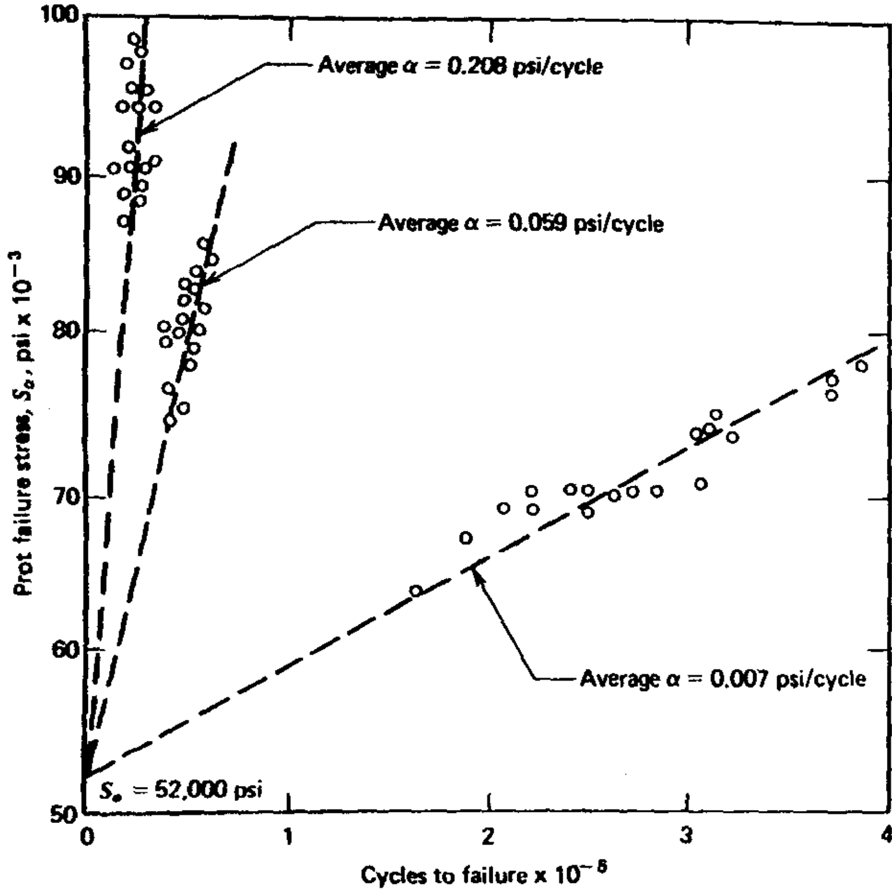


Figura 11. Prueba de prot que muestra el esfuerzo de falla contra los ciclos de falla para tres diferentes valores Prot.

En 1948 Marcel Prot propuso un método especialmente rápido para determinar el límite de fatiga[21]. El método Prot requiere la aplicación de un esfuerzo que se incrementa a razón constante durante los ciclos de fatiga hasta que la pieza falle. El límite de fatiga se obtiene relacionando el esfuerzo aplicado en el momento de la falla y dos constantes dependientes del material. Este método ha sido utilizado exitosamente con aleaciones de acero, titanio y aluminio, sin embargo, para poder utilizarse es necesario conocer o ignorar el efecto acumulativo.

---

[21] Collins, pág. 381

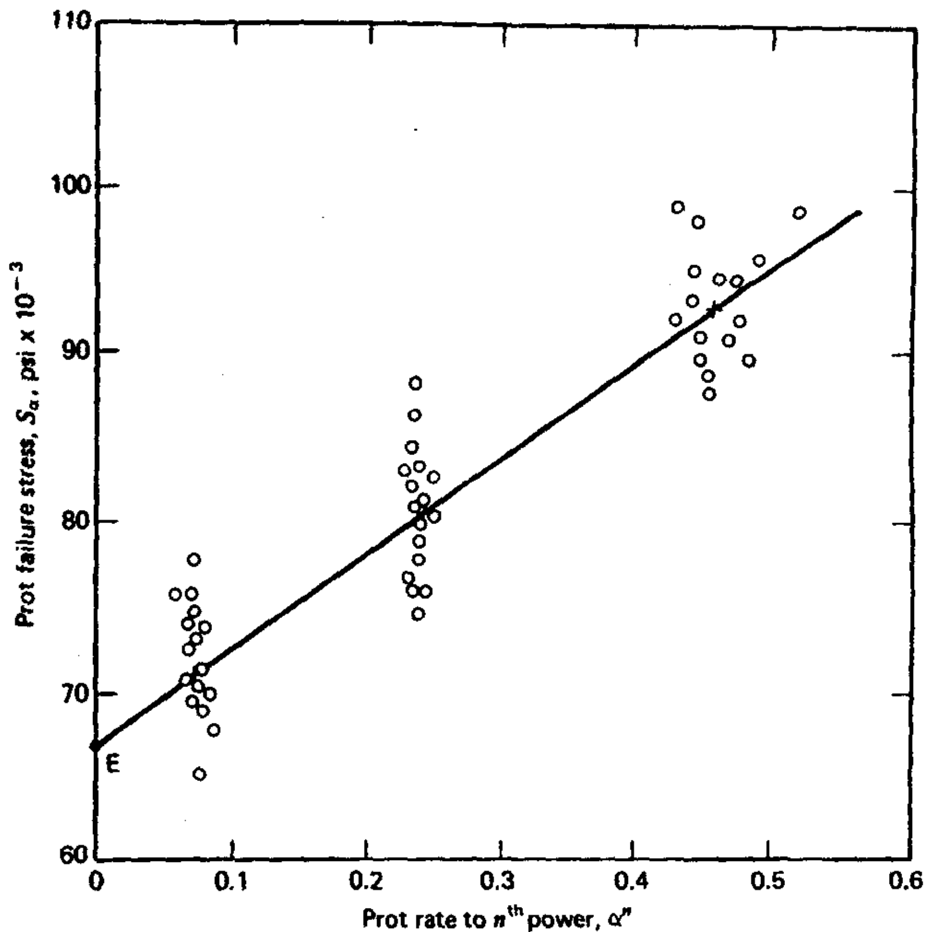


Figura 12. Resultados de la prueba Prot que muestra la determinación del límite de fatiga "E" al graficar el esfuerzo de falla contra el valor Prot.

El procedimiento Prot se basa en colocar un espécimen a prueba con un nivel de esfuerzo muy por debajo del límite de fatiga esperado, usualmente

en el rango de cero al 70% del límite de fatiga. Cuando la prueba comienza y se acumulan ciclos, el nivel de esfuerzo aplicado se incrementa sistemáticamente con los ciclos, de modo que en promedio el incremento en el esfuerzo es lineal con respecto al número de ciclos. El nivel de esfuerzo puede incrementarse ya sea de manera continua o discreta y la prueba continúa hasta que el espécimen falla, por lo que no hay especímenes que sobrevivan en el método Prot. Típicamente se requieren de 15 a 20 especímenes para esta prueba en la que se someten todos a la misma tasa de incremento de esfuerzo. Esta tasa de incremento se llama el valor Prot α y tiene dimensiones de presión por ciclo (Newton o psi por ciclo, por ejemplo). Un segundo grupo, o más grupos de especímenes se prueban después a valores Prot diferentes para lograr resultados más exactos. La información relevante que se obtiene de cada espécimen es el nivel de esfuerzo inicial S_0, el valor Prot α, el número de ciclos antes de la falla N_α y el esfuerzo de falla S_α. Prot propuso que la resistencia a fatiga se relaciona como: $S_\alpha = E + K\alpha^n$ donde K y n son las constantes del material. Si se relaciona S_α contra α^n se encuentra que se puede ajustar una relación lineal para algún valor de n, que se puede encontrar con algún método matemático o graficando los puntos.

4.1.8.6 El método de escalera o arriba - abajo.

Un método muy útil para determinar el esfuerzo de fatiga promedio y su desviación para una vida determinada es el método arriba - abajo. Este método es útil también para determinar el límite de fatiga, si se utiliza una vida lo suficientemente larga.

El método indica que se debe seleccionar un grupo de especímenes lo suficientemente grande (al menos 15 especímenes). El primer espécimen se prueba a un esfuerzo ligeramente superior al esfuerzo de fatiga esperado para la vida requerida hasta que falle o sobrepase el número de ciclos de interés. Si el espécimen falla antes de llegar al número de ciclos de interés, el nivel de esfuerzo se disminuye una cantidad predeterminada y se prueba un segundo espécimen a este nuevo nivel de esfuerzo. Si el espécimen sobrepasa el número de ciclos de interés, el nivel de esfuerzo se incrementa una cantidad predeterminada y un segundo espécimen se prueba a este nuevo nivel de esfuerzo. La prueba se repite para cada espécimen del grupo. Debido a la naturaleza secuencial de la prueba, ésta tiende a centrarse en el valor promedio del esfuerzo de fatiga para la vida de interés. Para efectos prácticos no se considera que la prueba haya comenzado sino hasta que se produzca el primer cambio entre supervivencia y falla; por tanto, el nivel de esfuerzo inicial es arbitrario y no tiene influencia sobre el resultado de la prueba. El nivel de esfuerzo inicial debe, sin embargo, ser considerado con mucho cuidado para disminuir el número de especímenes a utilizar.

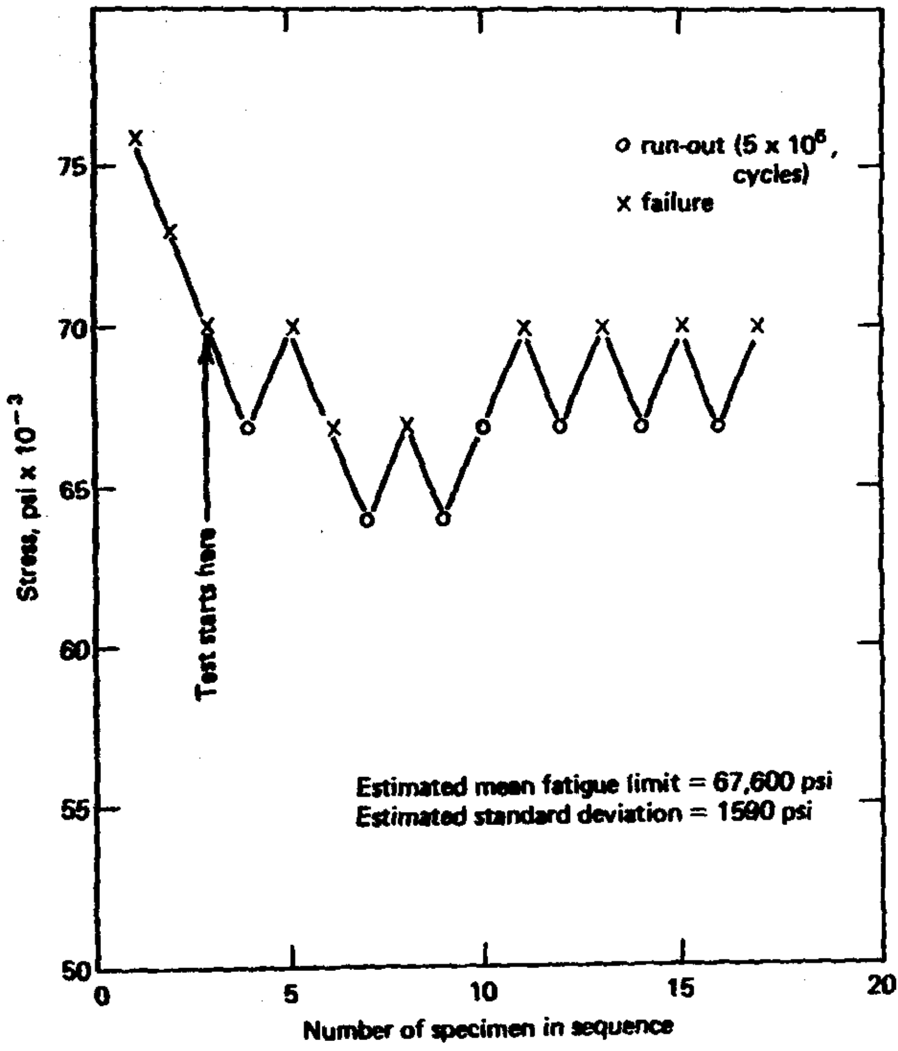


Figura 13. Prueba de fatiga arriba abajo utilizada para determinar el esfuerzo medio a 5 millones de ciclos para la aleación de acero 4340.

4.1.8.7 Método del valor extremo

El método del valor extremo es utilizado para encontrar el valor extremo de las curvas de probabilidad S-N. El método indica que se seleccione un grupo de n especimenes para ser probados simultáneamente en máquinas de fatiga idénticas. En la prueba se someten todos los especímenes del grupo al mismo esfuerzo, que permanece constante. Cuando el primer espécimen del grupo falla, se registra el esfuerzo aplicado y el número de ciclos a los que ocurrió la falla; se detienen las demás máquinas y se descartan los especimenes sobrantes.

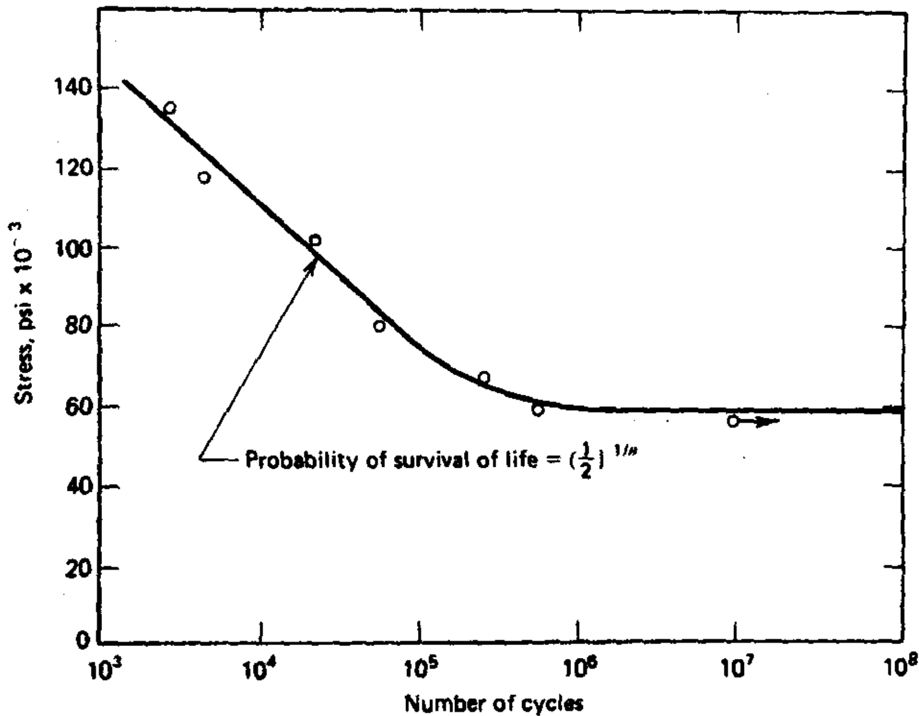


Figura 14. Probabilidad de supervivencia para la prueba del valor extremo en una gráfica S-N.

Después, se prueba un segundo grupo de n especímenes a un nivel de esfuerzo diferente. Las pruebas se repiten a varios niveles de esfuerzo desde el límite de fatiga en adelante, con el objeto de generar una curva SN. La curva que mejor se ajusta a los puntos corresponde a la curva de probabilidad dada por $P(\text{supervivencia}) = \left(\frac{1}{2}\right)^{1/n}$ donde n es el número de especímenes en cada grupo (permanece constante a lo largo de la prueba).

4.1.9 Teoría de Soderberg

Para muchos materiales de ingeniería, un esfuerzo que fluctúe entre dos valores dados α_{min} y $\alpha_{máx}$, es más probable que cause falla para el caso de que el esfuerzo fuera estable e igual a $\alpha_{máx}$. Debido a que las pruebas de fatiga son caras y consumen mucho tiempo, casi todos los datos disponibles están basados en flexión cíclica obtenida de un espécimen en rotación.

Si el esfuerzo de tensión en un punto está dado por el esfuerzo variable α, y el esfuerzo medio α_m, entonces ambos esfuerzos contribuyen a la falla. La línea Soderberg de falla es una representación aproximada de este efecto. Es una línea trazada entre el esfuerzo del punto de cedencia y el esfuerzo límite de fatiga en el sistema de coordenada esfuerzo medio - esfuerzo variable. Se ha observado que casi todas las fallas en los especímenes probados debido a las combinaciones de las cargas media y variable pudieran quedar representados como puntos por arriba o por abajo de esta línea. Si aplicamos un factor de seguridad N al esfuerzo de cedencia y al límite de fatiga, obtendremos la línea de esfuerzo seguro, que es una línea paralela a la línea de falla. Un punto de coordenadas σ_m, σ_r, trazado en o abajo de esta línea de esfuerzo representará cargado seguro.

Si consideramos un punto en un cuerpo para el cual hay concentración de esfuerzo, entonces debe multiplicarse el valor del esfuerzo variable por el factor de concentración de esfuerzo. Para este caso, el estado de esfuerzo para un punto estaría dado por σ_m, $k_f\sigma_r$, el cual debe quedar en o abajo de la línea de esfuerzo seguro. Cuando se trabaja con materiales dúctiles, el esfuerzo medio no se multiplica por el factor de concentración de

esfuerzo.

Supóngase que se carga un material con combinación de esfuerzo medio y variable hasta la condición límite segura. El estado de esfuerzo pudiera estar representado por el punto σ_m, $k_f\sigma_r$, sobre la línea de esfuerzo seguro tal como se muestra en la figura. Observado que los triángulos AOB y CDB son semejantes, obtenemos la relación

$$\frac{S_{yp}/N - \sigma_m}{K_f\sigma_r} = \frac{S_{yp}}{S_e}$$

la cual puede rearreglarse en la forma

$$\frac{S_{yp}}{N} = \sigma_m + K_f\sigma_r\frac{S_{yp}}{S_e}$$

El lado derecho de esta ecuación puede considerarse como un estado de esfuerzo estático equivalente. Si tentativamente diseñamos una pieza, el factor de seguridad real en un punto cualquiera está dado por

$$N = \frac{S_{yp}}{\sigma_{eq}}$$

Si este valor de N es menor que el factor de seguridad del diseño de conjunto que se ha seleccionado, se debe rediseñar la pieza.

En algunos problemas se requiere el diseño de una pieza para resistir una

relación específica de carga variable a carga media. Se puede obtener una relación adecuada si la relación esfuerzo variable a esfuerzo medio puede obtenerse a través de una relación conocida de esfuerzo-carga. En tales casos es conveniente rearreglar la ecuación en la forma

$$\sigma_m = \frac{S_{yp}/N}{K_f \dfrac{\sigma_r}{\sigma_m} \dfrac{S_{yp}}{S_e} + 1}$$

Si en la pieza no se tiene concentración de esfuerzos en el punto analizado, el valor de K_f es de uno. Una vez que se calcula α_m se utiliza la relación esfuerzo-carga para obtener la dimensión requerida en la pieza.

El criterio de Soderberg se aplica particularmente para el acero dúctil. Sin embargo puede usársele con seguridad para casi todos los materiales cuyos esfuerzos límite de fatiga y de cedencia sean conocidos. Para vida finita (un número dado de ciclos) la resistencia de cedencia basada en el método de desplazamiento podrá sustituirse por el esfuerzo del punto de cedencia. Si no se obtienen para ninguno de los casos anteriores, podrá usarse el valor de la resistencia última en lugar del esfuerzo del punto de cedencia, en cuyo caso sería prudente aumentar el factor de seguridad.

Si los esfuerzos principales son distintos a cero en dos o más direcciones, pudiera considerarse la utilización de la teoría de corte máximo o la teoría de energía de distorsión. Un procedimiento de diseño podría ser como sigue:

1. Asegurarse que la magnitud del esfuerzo de compresión no sea mayor que la resistencia última de cedencia a compresión o al esfuerzo de cedencia a compresión dividido entre el factor de seguridad

$$|\sigma_c|_{máx} \leq min\left(S_{u(c)}, \frac{S_{yp(c)}}{N}\right)$$

2. Asegurarse que la amplitud del esfuerzo variable no sea mayor que el esfuerzo límite de fatiga dividido entre el factor de seguridad

$$\sigma_r \leq \frac{S_e}{N}$$

5 Análisis de la funcionalidad de una pequeña herramienta auxiliar en el análisis de la fatiga

5.1 El modelo de requerimientos

Siguiendo la metodología de Jacobson, primero debe desarrollarse el modelo de requerimientos. Tomaré los capítulos teóricos y la introducción como especificación de requerimientos. De este modo los pasos a seguir son:

- identificar a los actores
- elaborar un modelo de casos de uso
- elaborar la descripción de interfases
- elaborar un modelo en el dominio del problema

5.1.1 Actores

Para identificar a los actores, es indispensable tomar en cuenta que un actor no es una persona, sino el rol que toma una persona. La única persona que interactuará con el programa, en este caso, es el ingeniero que desea elaborar un diseño de prueba a fatiga. En este sentido podemos identificar los siguientes roles:

Administrador de datos. El administrador de datos es el rol encargado de

asegurarse de la integridad de la información, así como de la adquisición de datos cuando no provienen de experimentación.

Diseñador de pruebas. El diseñador de pruebas a fatiga es el rol encargado de especificar el tipo de prueba, establecer los parámetros de la prueba seleccionada y recopilar los resultados de la prueba.

Analizador de datos. El analizador de datos es el rol encargado de tomar decisiones sobre la vida probable segura de un material dadas las condiciones de carga y los datos de vida cargados en el programa.

5.1.2 Casos de uso e interfases

Una vez identificados los actores se deben detallar los casos de uso. Un *caso de uso* es un tipo de transacción completa realizada por un actor. Los casos de uso deben ser una especificación completa de la funcionalidad del programa desde el punto de vista del usuario. Esto es, un curso completo e independiente de eventos relacionados que un actor realiza al dialogar con el programa en el contexto del sistema.

A continuación describo todas las transacciones que un usuario puede ejecutar utilizando el programa. Cada transacción o secuencia completa es llamada un *caso de uso* y es ejecutada por un usuario en alguno de los roles identificados, es decir, por un actor. Junto a la descripción del caso presento una descripción gráfica de todas las interfases que el programa presenta a los actores. Éstas servirán como guía adicional en la construcción del sistema (modelo de implementación).

5.1.2.1 Archivo de características de materiales

El objetivo de este *caso de uso* es permitir el dar mantenimiento a la base de datos de materiales y capturar información de fatiga que no proviene de experimentación. El administrador de datos selecciona en el programa la opción de "archivo de características de materiales".

Figura 15. Pantalla principal.

El *caso de uso* comienza con la identificación del usuario, quien deberá estar autorizado para acceder a esta opción que permite almacenar en memoria permanente o consultar en ella características del material importantes en el análisis de fatiga. Además esta opción debe permitir el añadir o eliminar materiales a la base de datos.

- nombre del material,
- temperatura de la prueba (grados Kelvin, celcius o farenheit),
- tratamiento térmico,
- dureza (Brinell, Rockwell, Vickers),

- esfuerzo último de cedencia (Pascal, kg/cm^2, psi),

- esfuerzo último de tensión (Pascal, kg/cm^2, psi) y

- resistencia a fatiga (esfuerzo máximo de fatiga para vida superior a 10 millones de ciclos, medido en Pascal, kg/cm^2 o psi) a una probabilidad de supervivencia especificada.

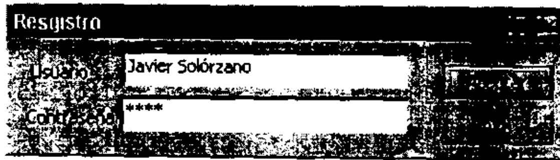


Figura 16

Los datos serán almacenados en unidades del sistema internacional aún cuando éstos pueden ser especificados y mostrados en cualquiera de las unidades especificadas.

Para las conversiones de temperatura se utilizan las siguientes fórmulas:

- de grados Celsius a Kelvin: C+273.15

- de grados Farenheit a grados Kelvin: (F+459.67)/1.8

Para las conversiones de unidades de esfuerzo:

- de kg/m^2 a Pascal: $kg/m^2 * 9.80665$

- de kg/cm^2 a Pascal: kg/cm^2 * 98066.5
- de psi a Pascal: psi * 6894.757

Las unidades de dureza no se convierten ya que la relación que existe entre las diversas escalas no es exacta. Esto es debido a que no hay dos métodos que midan exactamente la misma clase de dureza[22].

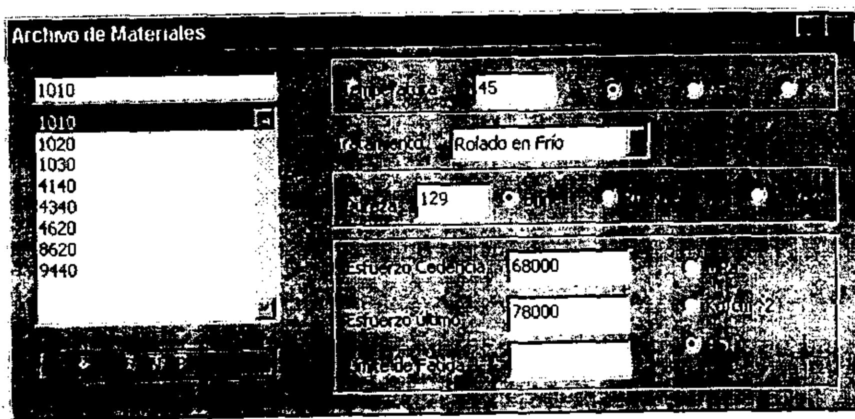


Figura 17

Para los datos de resistencia a fatiga, esta opción puede mostrar tanto datos capturados por separado como obtenidos por experimentación. En ambos casos mostrará los puntos discretos como pares ordenados de vida y esfuerzo de falla agrupados por probabilidad. Opcionalmente se puede ver la gráfica S-N-P correspondiente.

---

[22]Marks, cap. 5, pág. 14

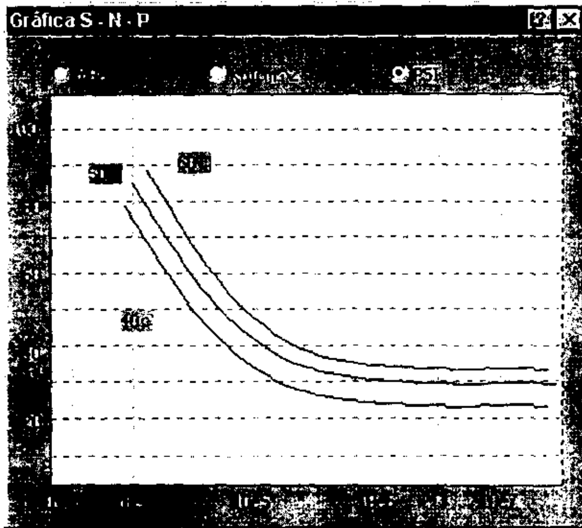


Figura 18

Es posible capturar los datos de un material sin capturar información de fatiga, en cuyo caso la información se utilizará para ser asociada posteriormente a datos de fatiga. La unidad de datos es diferenciada por el nombre del material y el tratamiento térmico. En el caso de que se capturen datos de prueba de fatiga, se incluirá también la temperatura de la prueba. En este caso la unidad de datos será la prueba misma especificada por el tipo de prueba, el material, la temperatura, el acabado térmico y un número de serie que permitirá almacenar resultados de pruebas bajo las mismas condiciones con la finalidad de una posterior comparación.

El programa deberá validar todos los datos y confirmar antes de hacer modificaciones permanentes. En particular se deberán evitar temperaturas Kelvin, esfuerzos y vidas negativas; probabilidades de supervivencia fuera

del rango 0 - 100% y durezas fuera del rango de la escala. Los rangos de las escalas de dureza son[23]:

- Brinell, para cargas de 3000 kg., 1500 kg. Y 500 kg. son 160 a 600, 80 a 300, 26 a 100 respectivamente.

- Rockwell, escalas A, B, C números positivos menores a 100.

- Vickers, números positivos menores a 1,000.

5.1.2.2 Diseño de experimentos de falla a fatiga

El objetivo de este *caso de uso* es auxiliar al ingeniero de laboratorio, al diseñar un experimento para conocer la resistencia a la fatiga de un espécimen. El diseñador de pruebas selecciona en el programa la opción de "diseño de experimentos".

Esta opción permite al diseñador seleccionar la prueba adecuada a las necesidades de información así como las posibilidades del laboratorio tomando en cuanta costo, tiempo involucrado y el número de especimenes del que se dispone.

El *caso de uso* comienza con la identificación del usuario, quien deberá estar autorizado para acceder a esta opción. Una vez identificado el usuario, se selecciona el tipo de prueba que se desea. La selección puede

---

[23] Deutschman, pág.. 107

ser directa (el diseñador de pruebas ya conoce qué prueba desea hacer y simplemente la selecciona) o mediante una selección del método de pruebas asistida.

Si se utiliza la opción de asistente automatizado, el sistema hará las siguientes preguntas:

- Objetivo de la prueba: Curvas S-N-P, límite de fatiga o resistencia a fatiga para una vida determinada.

- Número de especímenes disponibles.

Las siguientes reglas se utilizan para decidir qué prueba usar.

Si se desea conocer la resistencia a fatiga para una vida determinada se sugiere el método de escalera.

Si se desea conocer el límite de fatiga, se elige entre el método Probit y el método Prot dependiendo del número de especímenes. Si se tienen menos de 60 se sugiere Prot, si se tienen más de 60 se sugiere Probit.

Si se desean encontrar las curvas S-N-P, se selecciona de entre el método estándar, esfuerzo constante y prueba a pasos dependiendo del número de especímenes que se tengan. Si se tienen menos de 15 se sugerirá la prueba estándar, si se tienen de 20 a 60 se sugerirá la prueba a pasos y si se tienen más de 60 se sugerirá el método a esfuerzo constante.

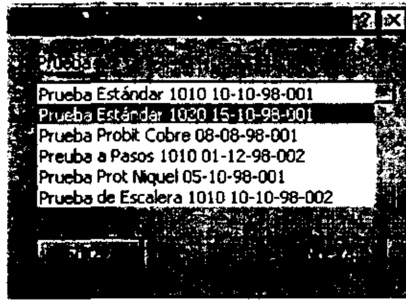


Figura 19

El método del valor extremo no se considera ya que requiere de varias máquinas idénticas funcionando simultáneamente.

Una vez que se ha seleccionado el método de la prueba a utilizar, se permite al diseñador cambiar la selección, y se le pide seleccione el material del espécimen, el tratamiento térmico, la temperatura de la prueba y la dureza del material. También se asignará un número de serie a la prueba.

Un procedimiento alterno a la selección de la prueba será tomar alguna ya registrada (seleccionable por el número de serie y los parámetros).

Una vez que se ha especificado correctamente toda la información se puede proceder a registrar los datos de la prueba.

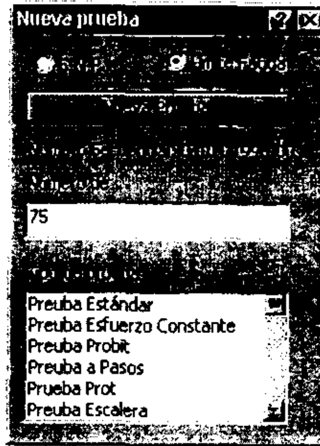


Figura 20

5.1.2.2.1 *Método de la prueba estándar*

Para el método estándar de prueba simplemente se registra para cada corrida de la máquina el nivel de esfuerzo al que se sometió el espécimen y el número de ciclos que sobrevivió. Si la prueba se detuvo por haber sobrepasado 10^7 ciclos, se registra como sobreviviente a la prueba. El diseñador de pruebas tendrá la opción de volver a utilizarlo en otra prueba o descartarlo. En caso de volver a utilizarlo en otra prueba, se tendrán más observaciones que especimenes. Cuando se ha concluido con todos los especimenes se muestra una tabulación de los datos obtenidos y opcionalmente se muestra una gráfica S-N con la curva promedio y los datos individuales. La curva puede ser aproximada de dos formas: dos líneas rectas, una horizontal al límite de fatiga (si es que se encontró) y una inclinada que se encuentre por mínimos cuadrados. Tanto la línea inclinada como la horizontal terminan donde se encuentran. Una segunda

forma podría redondear la esquina utilizando una curva de segundo grado que enlace ambas rectas.

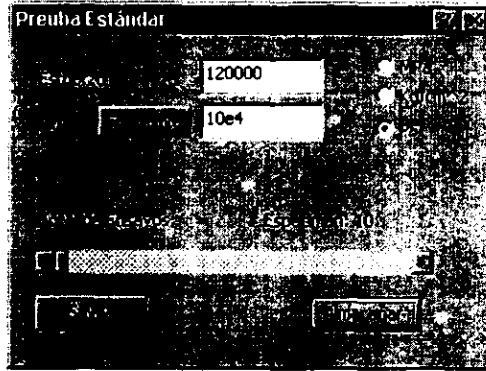


Figura 21

El método de mínimos cuadrados deberá utilizar los logaritmos de las vidas, ya que las gráficas S-N son semilogarítmicas.

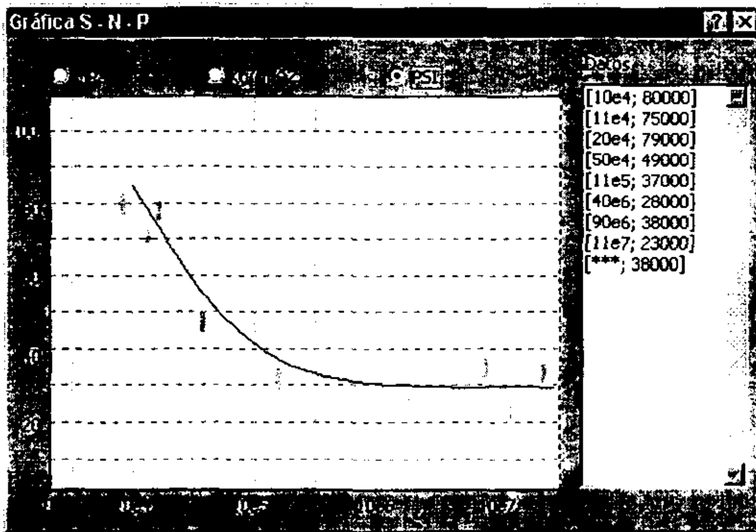


Figura 22

5.1.2.2.2 Método de la prueba a esfuerzo constante

Para el método de la prueba a esfuerzo constante se define el número de niveles de esfuerzo. Si el número de especímenes no es divisible entre el número de niveles de esfuerzo, se debe indicar el hecho al diseñador y se debe permitir modificar ambos parámetros. Se debe capturar un dato de vida para cada espécimen que es probado en cada nivel de esfuerzo. Una vez terminada la captura de todos los datos, se podrán mostrar tabularmente o mediante una gráfica S-N-P. La familia de curvas puede ser aproximada mediante una sencilla fórmula, ordenando de manera ascendente la vida a un determinado *nivel de esfuerzo*. La probabilidad de falla en ese punto en particular se calcula como $\frac{100q}{n+1}$ en porcentaje donde q es la *posición* y n es *el número de especímenes*. Por lo tanto se tendrán tantas curvas S-N-P como especímenes por nivel de esfuerzo. Es

posible aproximar una curva para cada probabilidad como en el método estándar mediante dos rectas y un codo. *La horizontal que representa* la asíntota al límite de fatiga se calcula con la moda de los valores de vida de los especímenes que sobrevivieron (más de 10^7 ciclos).

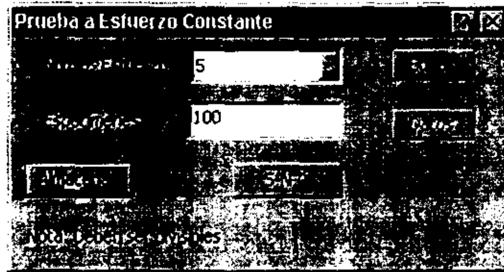


Figura 23

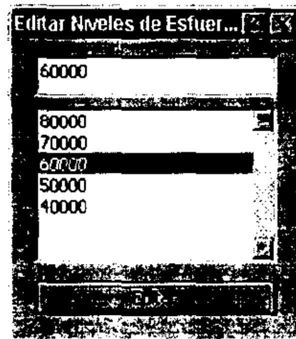


Figura 24

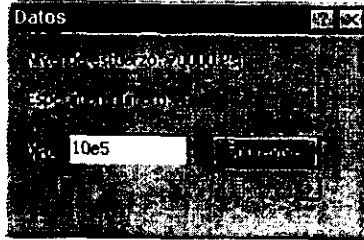


Figura 25

5.1.2.2.3 Método Probit

Para el método Probit, se requiere conocer los niveles de esfuerzo mínimo y máximo de la prueba. El rango cubierto por estos niveles debe cubrir dos desviaciones estándar por arriba y por debajo de la media aritmética de ellos. Esto se puede estimar por el diseñador, considerando dos niveles de esfuerzos en los que espera que a la vida deseada sobrevivan y fallen la mayoría de los especímenes. Se debe conocer el número de especímenes disponibles y el número de especímenes por nivel de esfuerzo que se desea usar. Típicamente se utilizarán 100 especímenes y 20 por cada nivel de esfuerzo. Éstos serán los valores que se sugerirán pero podrán ser modificados libremente. Si el número de especímenes no es divisible entre el número de especímenes por nivel de esfuerzo, el programa simplemente sustituirá el número de especímenes por el entero más próximo que divida exactamente al número de especímenes.

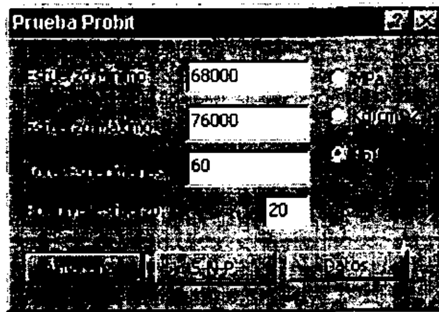


Figura 26

Una vez que se han ingresado los datos, el sistema divide el rango entre el número de niveles de esfuerzo resultantes y permitirá registrar para cada nivel de esfuerzo y espécimen la vida o si exceden los 10^7 ciclos.

Cuando se han concluido los experimentos, el sistema aproximará una familia de líneas rectas en el plano vida vs. probabilidad. Cada línea de la familia representa la probabilidad de falla del material dependiendo del nivel de esfuerzo a una vida determinada.

Cada línea es calculada mediante el método de los mínimos cuadrados. Los pares ordenados de cada recta se encuentran como el número de especímenes que sobrevivieron un nivel de esfuerzo entre el total de especímenes por nivel y el nivel de esfuerzo. Se puede calcular una línea para cualquier valor de vida dentro del rango de vida de los puntos (por ejemplo 10^7 y la penúltima vida, ya que para especificar la recta se requieren al menos dos puntos). También se debe poder mostrar la ecuación de cada recta.

5.1.2.2.4 Método a pasos

Para el método a pasos se requiere el número de especímenes disponibles y el nivel de esfuerzo al que se desea comenzar la prueba. Este nivel de esfuerzo deberá ser aproximadamente el 70% del límite de fatiga para la vida deseada (10^7 ciclos, por ejemplo). El sistema sugiere que el incremento en el esfuerzo entre pasos sea el 30% del esfuerzo inicial. Se registra para cada espécimen la vida y el esfuerzo de falla. Cuando se ha terminado la prueba se puede mostrar un diagrama esfuerzo vs. probabilidad y su ecuación calculada por el método de mínimos cuadrados. Cada par ordenado se obtiene con el porcentaje de los especímenes que sobrevivieron a un nivel de esfuerzo y el nivel de esfuerzo.

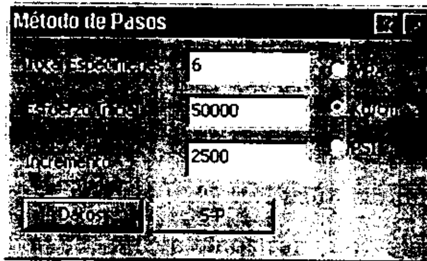


Figura 27

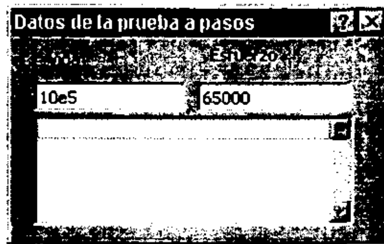


Figura 28

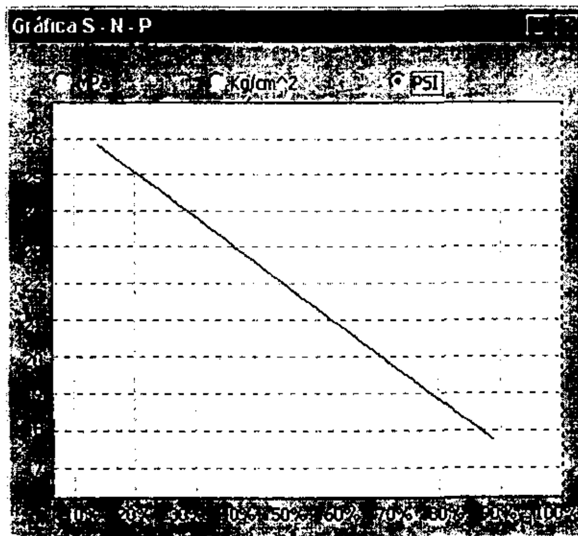


Figura 29

5.1.2.2.5 Método Prot

Para el método Prot se requiere conocer el número de especímenes a utilizar por tasa Prot (mínimo 10), la constante n del material (que deberá estar en el rango $0 < n < 1$), el esfuerzo inicial de la prueba y el número de tasas Prot α que se utilizarán (mínimo 2). El esfuerzo inicial de la prueba

deberá ser aproximadamente el 70% de la resistencia a fatiga.

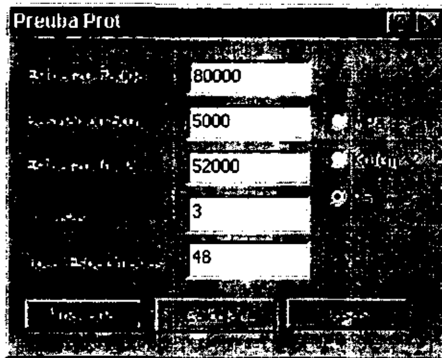


Figura 30

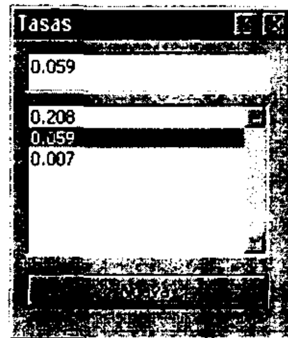


Figura 31

Para cada espécimen se registra la tasa Prot α, el número de ciclos para la falla N, y el esfuerzo de falla S. Para calcular la constante K, se calcula para dos tasas Prot, el esfuerzo promedio y se encuentra de: $k = \frac{S_1 - S_2}{\alpha_1^n - \alpha_2^n}$.

Cuando se ha calculado K, es posible encontrar el esfuerzo de falla para cada espécimen de $E = S_a - K\alpha^n$, después se calcula el esfuerzo

promedio y la desviación estándar de las E's. Este método no presenta ni gráficas ni ecuaciones, tan sólo el esfuerzo de falla promedio y su desviación estándar.

5.1.2.2.6 Método escalera

Para el método de la escalera se requiere saber el número de especímenes con los que se cuenta (mínimo 15), el valor esperado del esfuerzo de fatiga, su desviación estándar estimada y la vida a analizar (dato informativo).

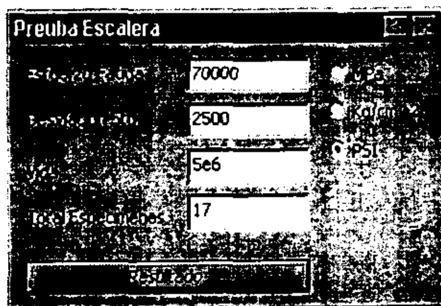
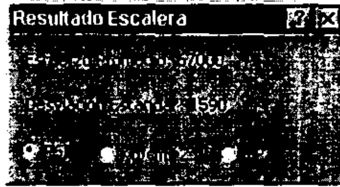


Figura 32

Una vez que se han capturado los datos, el programa indica el esfuerzo al que se debe someter el espécimen, como la media más una desviación estándar, y preguntará si el espécimen sobrevivió a la prueba o si falló antes del número de ciclos.



Si el espécimen falló, se indicará probar el siguiente a un esfuerzo menor en una desviación estándar.

Si el espécimen sobrevivió, se indicará probar el siguiente a un esfuerzo mayor en una desviación estándar.

Se repite el proceso hasta que se termine con los especímenes.

Cuando la prueba se ha terminado se calcula el esfuerzo de falla a fatiga promedio y su desviación estándar para la vida especificada siguiendo el siguiente procedimiento.

Se selecciona el evento menos frecuente (sobrevivir o fallar).

I	II	III	IV	V
64,000	0	2	0	0
67,000	1	5	5	5
70,000	2	0	0	0

Figura 33. Ejemplo de tabulación.

Se tabulan los niveles de esfuerzo del evento menos frecuente de menor a mayor; el número secuencial, comenzando por cero; el número de veces que ocurrió el evento a cada nivel de esfuerzo; el producto de la segunda y

tercer columna; el producto del cuadrado de la columna dos por la columna tres.

La sumatoria de la columna cuatro la llamaremos A. La sumatoria de la columna cinco la llamaremos B. El esfuerzo de fatiga promedio para la vida especificada se calcula como: $\hat{S}_m = S_0 + d\left(\frac{A}{N} \pm \frac{1}{2}\right)$ donde S_m es el estimado del esfuerzo de falla por fatiga a una vida determinada. S_0 es el nivel de esfuerzo menor que ocurrió en el evento menos frecuente, d es el tamaño de paso utilizado (la desviación estándar estimada), N es el número total de los eventos menos frecuentes. El signo + se utiliza si el evento menos frecuente es sobrevivir y - se utiliza si el evento menos frecuente es la falla.

La desviación estándar se calcula como: $\hat{\sigma} = 1.62d\left(\frac{NB - A^2}{N^2} + 0.029\right)$ si

$\frac{NB - A^2}{N^2} \geq 0.3$ ó

$\hat{\sigma} = 0.53d$ si $\frac{NB - A^2}{N^2} < 0.3$.

5.1.2.3 Análisis de resultados de experimentos de falla a fatiga

El *objetivo de este caso de uso* es explotar la información de fatiga almacenada en el programa. El analizador de datos selecciona en el programa la opción de "análisis de resultados".

Este *caso de uso* permite al analizador de resultados obtener para un material en una condición particular conocer las condiciones de falla y, opcionalmente, utilizar el criterio de Soderberg.

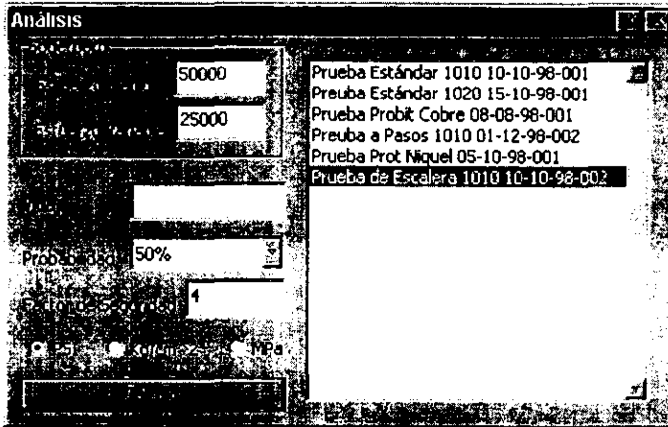


Figura 34

El *caso de uso* comienza pidiendo al analizador que seleccione de las pruebas almacenadas la que desea utilizar. Para tal efecto se presenta una lista de todos los materiales y tratamientos térmicos disponibles y si existen para el mismo caso más de una prueba, el diseñador de la prueba que la realizó. Además se debe informar si se cuenta con información de límite de fatiga o curvas S-N-P.

Si se seleccionó una prueba con datos de límite de fatiga, el programa deberá pedir el factor de seguridad deseado, y si se aplicará el criterio de Soderberg para encontrar un esfuerzo equivalente. Dependiendo de la prueba, se pide la probabilidad deseada (cuando se trata de curvas S-N-P) de la lista de probabilidades disponible, y el esfuerzo aplicado o la vida

deseada. El programa dará como resultado el dato no proporcionado, esfuerzo o vida. Para el caso de esfuerzo, se comparará contra el esfuerzo del criterio de Soderberg y se proporciona el factor de seguridad.

5.1.3 Modelo en el dominio del problema

La metodología de Jacobson indica que elaborar un modelo de objetos en el dominio del problema puede ser muy útil para mostrar la funcionalidad del sistema a usuarios. No es conveniente elaborar mucho en este modelo ya que sólo es útil para comunicar el concepto general. De lo contrario se estaría demasiado tentado a traducir este modelo al modelo de diseño. Sin embargo, como he explicado anteriormente, el modelo de diseño se elabora buscando la robustez que permita cambios ulteriores. Este no es el caso del modelo del dominio del problema.

Una buena manera de iniciar el modelo del dominio del problema es preguntarse ¿con qué se está trabajando? En este caso la respuesta es "datos de pruebas de fatiga". El objeto datos de pruebas de fatiga, abstraerá el almacenaje e interpretación de los datos de una prueba de fatiga. Como hemos visto, una prueba de fatiga no es un concepto final, sino que debe especificarse el tipo de prueba de fatiga, ya que cada clase de prueba requiere que se le dé un tratamiento distinto a los especímenes. Aún así podemos encontrar características comunes en los diferentes tipos de pruebas. Es por esto, que podemos definir el objeto pruebas de fatiga como una super-clase abstracta de la cual se especifican las pruebas:

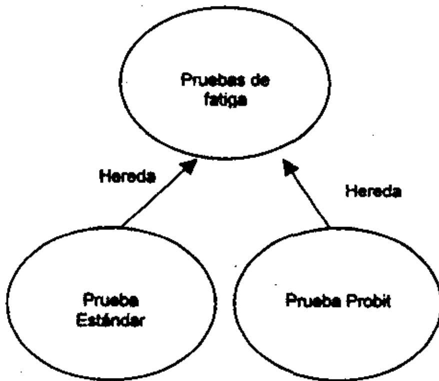


Figura 35

Estrictamente hablando, deberíamos introducir dos niveles de herencia, ya que se pueden distinguir dos tipos de pruebas: las que dan por resultado un límite de fatiga a una vida especificada y las que dan por resultado una curva de vida a diferentes cargas. Si bien, esto es cierto, la funcionalidad que las pruebas de laboratorio poseen bien puede ser abstraída en tan sólo dos niveles, haciendo el modelo más simple.

A su vez, una prueba de laboratorio contiene datos de experimentación. La clase Conjunto de datos abstrae la funcionalidad del almacenaje y manipulación de los datos. De nuevo, la funcionalidad específica varía según el tipo de prueba. De modo que podemos crear una super-clase que abstrae la funcionalidad general y sub-clases, una por cada tipo de prueba, de la que se hereda la funcionalidad específica asociada a cada prueba. Existe una asociación dinámica entre las subclases derivadas de las pruebas de fatigas y los objetos derivados de conjuntos de datos. La

asociación dinámica se indica a nivel superclase.

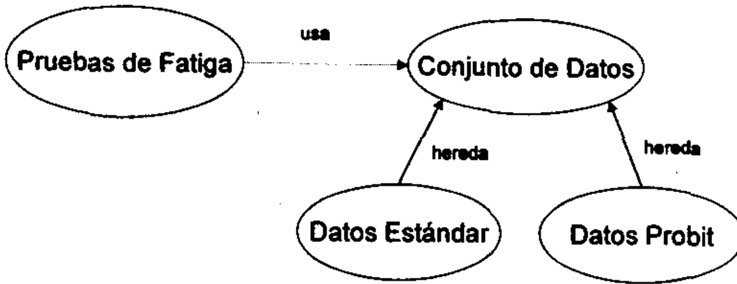


Figura 36

Por último cada instancia de prueba de fatiga está intimamente ligada a un conjunto de características que definen un experimento. El experimento guarda información como la relativa al diseñador de la prueba y la fecha de realización y el material utilizado.

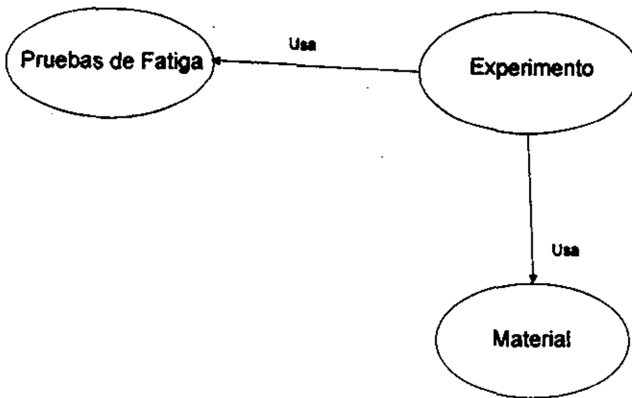


Figura 37

5.2 El modelo de objetos

El primer paso para elaborar el modelo de objetos, es separar la funcionalidad del sistema en tres dimensiones mutuamente exclusivas como se muestra en la figura:

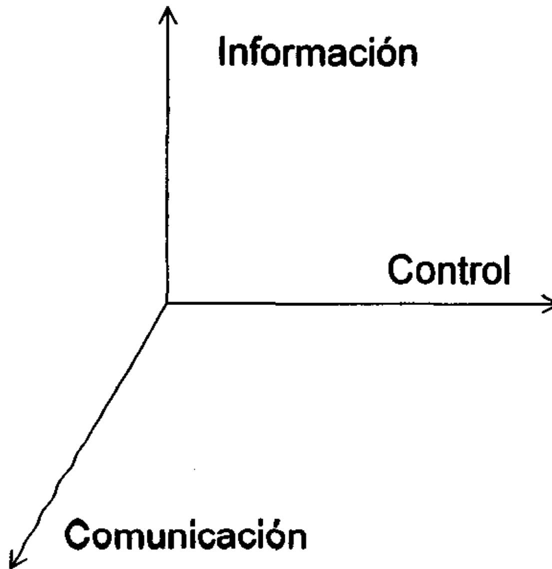


Figura 38

La funcionalidad de comunicación, ya sea con el usuario (interfase) o con otros sistemas debe ser agrupada por separado de la funcionalidad de control y de los requerimientos de almacenamiento de información permanente. En el modelo de objetos que presento, utilizo la simbología *presentada originalmente* por Jacobson.

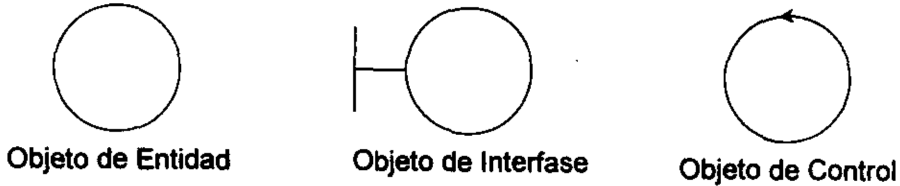


Figura 39

Idealmente la separación en estos tres grupos debe ser perfecta, lo que significa que nunca habrá mezcla alguna de funcionalidad entre los grupos. En la práctica si bien es fácil en términos generales separar la funcionalidad básica en estos tres grupos, podría haber dificultad en el detalle. Aún así, entre mejor se logre la separación más robusto será el sistema y por tanto cualquier esfuerzo adicional en esta etapa redituará en un mantenimiento más económico.

En el ejemplo en particular que presento, la funcionalidad se puede separar sin ningún problema. Las interfases descritas en las secciones anteriores son agrupadas en los objetos de interfase. La funcionalidad en sí es descrita en los objetos de control y toda la información que es almacenada es agrupada en objetos de entidad.

Los objetos de control los podemos identificar como el núcleo del proceso de operación del programa. A un nivel muy general, siempre se puede ver a un objeto de control como la lógica que controla a un *caso de uso*. Sin embargo, si tan sólo se dijera que un objeto de control es un *caso de uso*, estos objetos podrían tornarse muy complejos, ya que, como hemos visto, un *caso de uso* es una secuencia completa de interacción con el sistema. Es por esto que los objetos de control pueden y deben agruparse dentro del

caso de uso, pero abstrayendo funcionalidad más discreta. Otra poderosa razón para definir a los objetos de control como entes más simples que todo un *caso de uso* es la reutilización. Frecuentemente nos encontraremos con funcionalidad que se repite entre casos de uso. En estos casos es muy conveniente agrupar esta funcionalidad en *objetos* de control que serán *reutilizados*.

Cada objeto de interfase es en general una pantalla cuando se trata de un usuario, o un protocolo electrónico de comunicación, cuando el sistema se comunica con otro sistema. En el caso que nos ocupa todas las interfases son con usuarios. Cada pantalla debe ser representada por un objeto de interfase. Nos encontraremos frecuentemente con pequeñas pantallas de diálogo cuya vida es bastante corta y se utilizan principalmente para validaciones, obtener información adicional y que en general no existen por sí mismas, sino siempre ligadas a otra. Este *tipo de pantallas* (varias de este *tipo están descritas en el ejemplo*), no se consideran objetos de diseño de interfase sino que se incluyen dentro del objeto de interfase principal, sin perjuicio de ser construidas como un objeto del lenguaje de programación elegido.

Por último, los objetos de interfase, son aquellos que agrupan la información y funcionalidad que debe ser almacenada de forma permanente.

El siguiente diagrama muestra un modelo de *objetos* para la funcionalidad inicial del sistema.

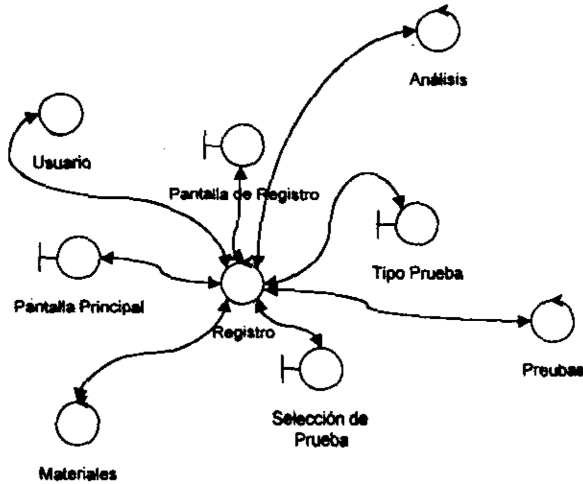


Figura 40

Observamos cómo el centro de todo el sistema está en el objeto de control de registro, quien controla todas los objetos de interfase y que cede el control a otros objetos de control (materiales, pruebas y análisis) según sea necesario. Ya que Registro es el primer objeto de control y el que despacha los casos de uso ese será el objeto principal, que en *Java* se llama "*main*". Las líneas indican relación dinámica y flujo de control.

Cada objeto de control a su vez debe ser expandido para mostrar los objetos a los que afecta. Vemos cómo los diagramas para los objetos de control de Materiales y análisis son más simples y similares.

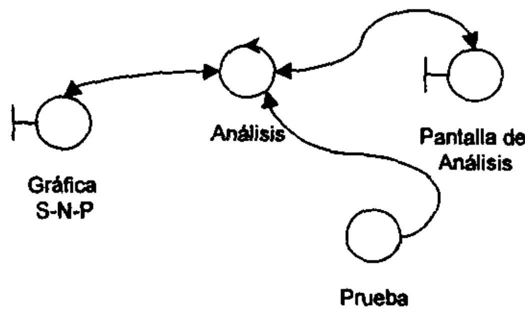


Figura 41

En estas gráficas podemos ver cómo algunos objetos pueden ser reutilizados entre casos de uso, como es el caso del objeto Gráfica S-N-P.

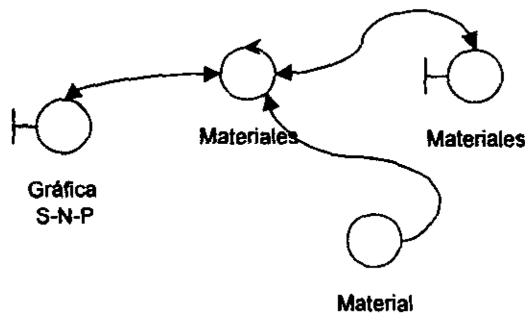


Figura 42

El diagrama de *caso de uso* de pruebas, correspondiente al objeto de control de pruebas es notoriamente más complejo y aparece un nuevo tipo de línea, la punteada, que representa relación estática o jerárquica de herencia.

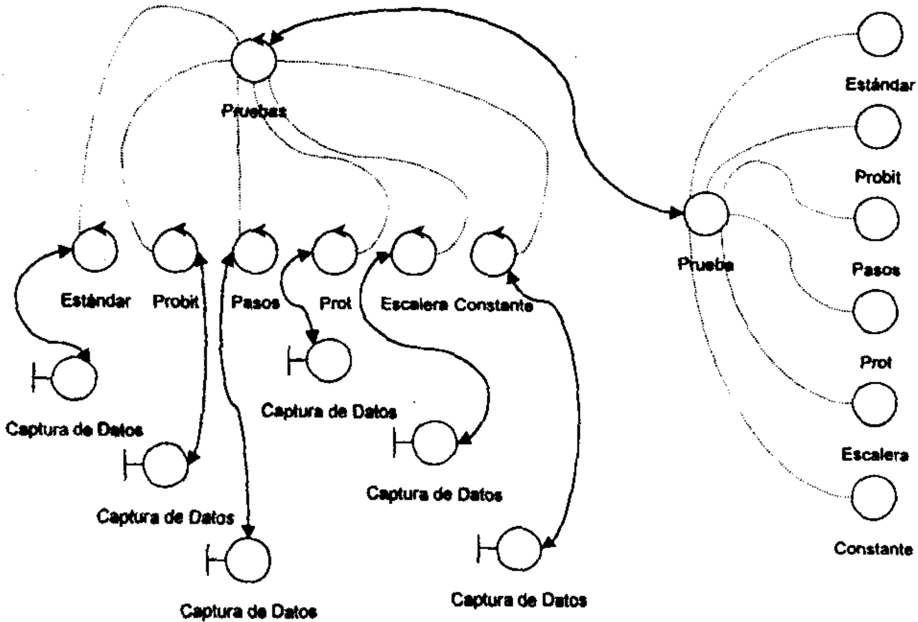


Figura 43

En este caso el modelo de objetos se alineará sin ningún problema al modelo de construcción si éste se realiza en *Java*, ya que resulta de manera natural que cada tipo específico de prueba sea un objeto especializado de una super clase de pruebas. O que cada objeto de entidad de prueba, que almacena los resultados de una prueba, sea un descendiente de una super clase de pruebas. Sin embargo, debe entenderse que este concepto de servir de igual forma si se tratase de un lenguaje estructurado, o no donde no existen los conceptos de herencia y objetos.

En el caso orientado a objetos, en tiempo de ejecución el programa decidirá qué clase de objetos, descendiente de la superclase, se llamará

gracias al polimorfismo. Sin embargo, en el caso estructurado o no estructurado seguramente se requerirá de lógica del tipo "case" para seleccionar entre las distintas clases de pruebas a realizar. Cualquier cambio que se decida hacer quedará estrictamente localizado en la unidad de compilación referente a la clase del objeto en cuestión, sin necesidad de alterar ninguna otra unidad de compilación. En el caso de *Java* ni siquiera es necesario religar los módulos, como podría ser el caso de C++.

A manera de ejemplo de cómo se utilizaría *Java* en el modelo de construcción, presento algunos de los encabezados de algunos de los objetos que deberían ser utilizados. Si bien, esto corresponde estrictamente al modelo de construcción y por tanto no deben aparecer en un documento de diseño y mucho menos de análisis, son útiles en este momento para ligar los conceptos de programación orientada a objetos, con el modelo de objetos que presento en esta tésis. Al final de los encabezados presento a manera de referencia el diagrama del modelo de objetos integrado.

En primer lugar está el objeto de control principal, llamado *Registro*. Dentro del objeto tendremos métodos que implementarán la funcionalidad del sistema. El método *main* es el primero en ser llamado a ejecutarse en una aplicación *Java* y siempre deberá existir uno y sólo un método *main* que es el corazón del sistema.

```
public class Registro {
public static void main ( String args[ ] )
        /* Controla el inicio de la aplicación */
public VentanaPrincipal ( argumentos )
/* Crea y maneja el objeto de interfase de la pantalla principal */
public ValidaUsuario ( argumentos )
        /* Extrae la información del usuario de la base de datos */
```

```
.
.
}

public class Usuario extends ISAMBD {
        /* los datos del usuario podrían estar almacenados en tablas de texto */
.
.
.
}

public class PantallaRegistro extends Applet {
/* Los objetos de interfase pueden adaptarse perfectamente bien a cualquier jerarquía
de objetos que se decida utilizar. En este caso podría ser AWT de Sun Microsystems,
pero bien podría ser MFC, por ejemplo */
.
.
.
}

abstract public class Pruebas {
/* la clase de Pruebas se declara abstracta ya que no tiene sentido instanciar un objeto
de pruebas sin especificar de qué tipo de prueba se trata, sin embargo la clase abstracta
incluye el comportamiento genérico */
public CapturaDatos ( argumentos )
public EjecutaPrueba ( argumentos )
public AlmacenaDatos ( argumentos )
.
.
.
}

public class PruebaEstandar extends Preubas {
        .
        .
        .
}
public class PruebaProbit extends Pruebas {
        .
        .
        .
}
```

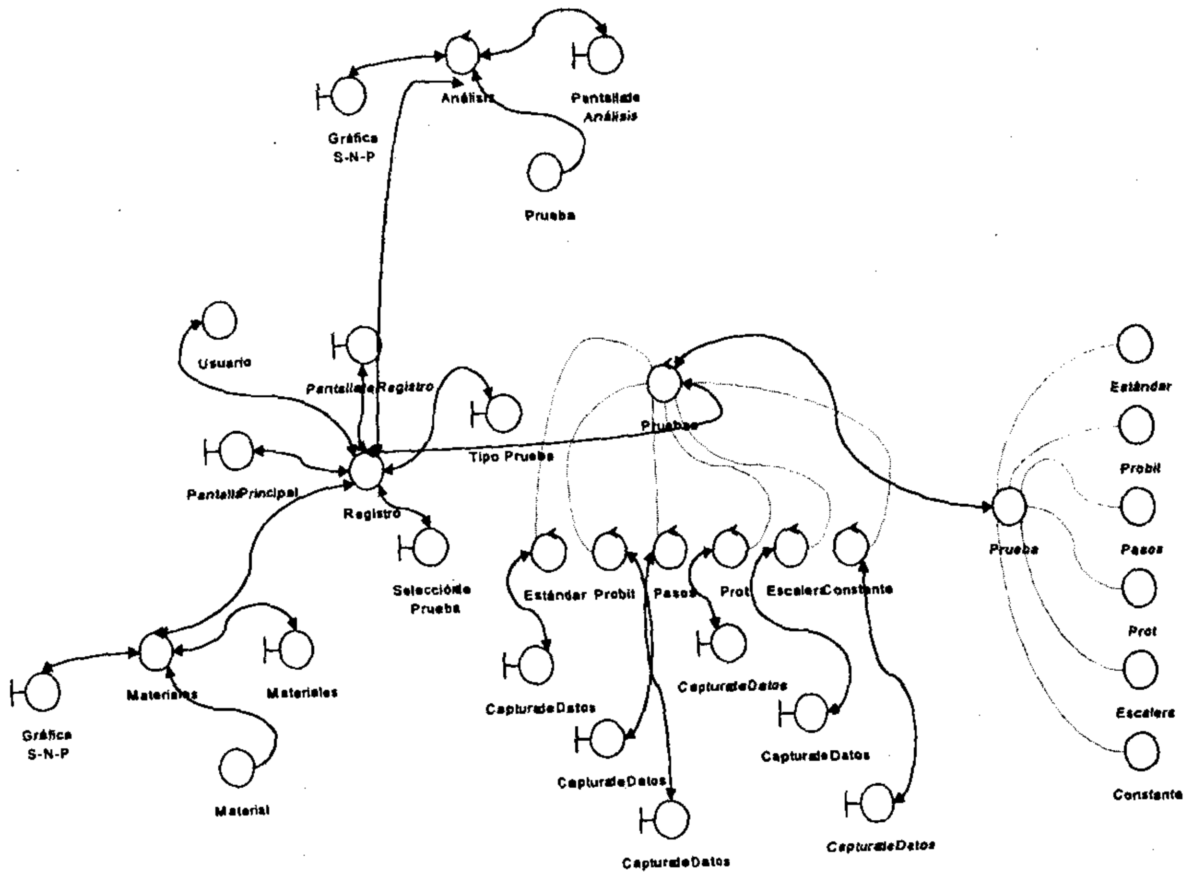


Figura 44

6 Conclusiones

El modelo de Ivar Jacobson de casos de uso puede ser utilizado tanto para sistemas de gestión como para cualquier otro tipo. El concepto más valioso de la metodología de Jacobson es el de costo total del sistema durante toda su vida. Las técnicas de ingeniería de software tradicionales terminan el análisis y diseño en el modelo de objetos en el dominio del problema. En general, estas metodologías profundizan en este modelo mucho más de lo que en esta tesis lo he hecho, especificando no sólo todas las clases y superclases, sus atributos lógicos, las asociaciones estáticas de clases, la herencia y las asociaciones dinámicas de objetos. El problema es que al seguir esta alternativa, si bien se puede construir bien un sistema, se termina mezclando en cada objeto peligrosamente la funcionalidad esencial junto con funcionalidad accesoria, por ejemplo el cálculo o las operaciones junto con el desplegado de la información, especialmente dado que la mayoría de las bibliotecas de clases que se suministran con los lenguajes de programación están orientadas a la interfase gráfica con el usuario.

En la metodología Jacobson no se insiste mucho en este modelo, ya que tan sólo es un auxiliar, que es útil generalmente cuando la funcionalidad es muy extensa, para comunicarse con los usuarios o patrocinadores del sistema. En caso de haber continuado con el diseño sobre el modelo del dominio del problema inevitablemente se hubiera mezclado la funcionalidad de la interfase del usuario en los objetos pruebas y de conjunto de datos. El problema se hubiera complicado al mezclar la

estructura de objetos del análisis con la estructura de objetos de la interfase gráfica. Generalmente se termina con una poca elegante solución de herencia múltiple, cuando el lenguaje lo permite, mezclando ambas jerarquías de objetos. Esto presenta dos claras desventajas:

Primero, la complejidad que una jerarquía de objetos con herencia múltiple presenta. Debido a que las funcionalidades están mezcladas, al alterar la funcionalidad gráfica u operativa, necesariamente se debe alterar la otra funcionalidad.

Segundo, los lenguajes que presentan la posibilidad de herencia múltiple, como C++, son generalmente complejos y muy propensos a errores. Las llamadas a objetos con herencia múltiple son generalmente poco eficientes, ya que se resuelven durante la ejecución. Cuando el lenguaje no soporta herencia múltiple, debido a la complejidad de las interfases gráficas, su jerarquía prevalece encima de la del dominio del problema, ajustándose esta segunda en la medida de lo posible. Con las obvias desventajas que esto conlleva. El mismo efecto se tiene con la funcionalidad de almacenamiento de la información.

El modelo de casos de uso separa de manera muy limpia la funcionalidad en tres ámbitos: la entidad, la interfase y el control. Cualquier cambio en alguno de estos ámbitos queda completamente aislado del resto del código, por lo que las modificaciones ulteriores que un sistema siempre sufrirá quedan confinadas haciendo el mantenimiento mucho más barato y eficiente.

Una desventaja del método de casos de uso que no queda muy clara con el

ejemplo escogido para ilustrarlo, y es que requiere que se tomen muchas decisiones en el *modelo de construcción*. Una mejora muy sustancial al modelo de Jacobson exigiría que se detallara *en el modelo de diseño* los diagramas de interacción y de transición, especialmente útiles en sistemas complejos. De esta manera toda la funcionalidad quedaría completamente contenida antes del modelo de construcción. Esta desventaja de hecho no lo es cuando el equipo que analiza, diseña y construye es el mismo. Sin embargo, hoy en día la globalización requiere que la construcción de sistemas sea desarrollada en localidades remotas, donde pueda ser más conveniente que en un lugar cercano al usuario o patrocinador del sistema.

El centrar la atención durante el análisis a los casos de uso permite elaborar una especificación inicial que puede ser compartida y entendida por los usuarios y no requiere interpretación por un experto de sistemas. Esto es particularmente útil cuando existe dificultad para transmitir las ideas entre los equipos de usuarios y de desarrolladores.

El ejemplo de modelo de objetos puede ser contrastado con el modelo de objetos del dominio de problema. Encontramos cómo aún con similitudes muy fuertes en los objetos, el modelo es esencialmente distinto. El modelo del dominio del problema, es útil nada más como herramienta de comunicación entre los diseñadores y los usuarios, mientras que el modelo de objetos de Jacobson proporciona un marco poderoso de construcción que tendrá como resultado un sistema robusto y tolerante a cambios.

El método de análisis de casos de uso hace énfasis en la orientación a objetos para el diseño. Sin embargo, el concepto de objetos no es exclusivo

para la programación ni incompatible con el análisis. Esto significa que es completamente posible hacer un análisis orientado a objetos de Ivar Jacobson, pera realizar la construcción en un lenguaje estructurado e inclusive no estructurado. Esto significa que el método de casos de uso puede bien ser utilizado para llevar a cabo todo el ciclo de desarrollo de un programa de control numérico incrustado en alguna maquinaria y programado en lenguaje de máquina. Sin perjuicio de lo dicho anteriormente, el método es mejor aprovechado cuando se utiliza un lenguaje puro orientado a objetos, como *Java*. En resumen, un análisis de casos de uso no implica un diseño orientado a objetos y un diseño orientado a objetos no implica una construcción en un lenguaje orientado a objetos. Aún así, un lenguaje orientado a objetos poderoso como *Java* se adaptará naturalmente al modelo de Ivar Jacobson.

Por último, en la sección del modelo de objetos se ilustró claramente porque es más robusto un sistema programado en un lenguaje orientado a objetos que uno estructurado. En el ejemplo se mostró cómo tan sólo es necesario crear un nuevo tipo de objeto para expandir el sistema, mientras que en una solución estructurada seguramente se tendría que modificar varias unidades de compilación.

7 Bibliografía

Baumeister, Avallone, Baumeister III. *Marks, Manual del Ingeniero Mecánico.* McGraw Hill. Octava Edición, 1984. ISBN 968-451-640-1

Collins Jack. *Failure of Materials in Mechanical Design, Analysis, Prediction, Prevention.* Wiley Inter-Science. 1993. ISBN 0-471-55891-5

Deutschman Aaron, Michels Walter, Wilson Charles. *Diseño de Máquinas, Teoría y Práctica.* CECSA. 1985. ISBN 968-26-0600-4

Grady Booch. *Object Analysis and Design with Applications.* Benjamin Cummings. Second Edition. 1994. Redwood City, California. ISBN 0-8053-5340-2

Humphrey, Watts. *A Discipline for Software Engineering.* Addison Wesley. Reading, Massachusetts. 1995. ISBN 0-201-54610-8

Jacobson Ivar. *Object-Oriented Software Engineering, A Use Case Driven Approach.* Addison-Wesley. 1992. ISBN 0-201-54435-0

Jacobson Ivar. *The Object Advantage.* Addison-Wesley. 1994. ISBN 0-201-42289-1

Kernighan Brian, Ritchie Dennis. *El Lenguaje de Programación C.* Segunda

Edición. 1991. Prentice Hall Hispanoamericana. ISBN 968-880-205-0

Mendenhall, Scheaffer, Wackerly. *Estadística Matemática con Aplicaciones.* Grupo Editorial Iberoamericana. Tercera Edición. 1986. ISBN 968-7270-17-9

Naughton Patrick. *Java Handbook.* Osborne McGraw Hill. 1996. Berkeley, California. ISBN 0-07-882199-1

Peterson, Ivars. *Fatal Defect.* Times Books. 1995. ISBN 0-8129-2023-6

Oberg Erick. Machinery´s Handbook. 24 edición. 1992. Industrial Press. ISBN 0-8311-2492-X

Shigley Joseph, Mitchell Larry. *Diseño en Ingeniería Mecánica. McGraw Hill. Cuarta Edición.* 1989. ISBN 968-451-607-X

Stroustrup Bjarne. *El Lenguaje de Programación C++.* Addison-Wesley. Segunda Edición. 1993. ISBN 0-201-60104-4