Capítulo 8

Trabajos futuros

A pesar de que la aplicación funciona de forma correcta (consistente y eficiente), existen determinados aspecto que pueden mejorarse, tanto desde el punto de vista informático como desde el mecánico.

Este capítulo, sirve de colofón a todo el trabajo realizado en el Proyecto, y por tanto constituye el mejor momento para sacar conclusiones respecto de la aplicación. Por ello, se establece una división de las posibles modificaciones o mejoras que puedan llevarse a cabo, en virtud de la rama de la Ingeniería a la que pertenezcan. En primer lugar se describen algunas mejoras de tipo informático, y posteriormente, las mejoras que puede precisar para otros cálculos mecánicos.

8.1 Las mejoras informáticas

El mundo de la informática es un mundo muy dinámico, en constante cambio y evolución; surgen nuevas versiones de aplicaciones antes de que las anteriores se hayan consolidado definitivamente. Por otro lado, en el caso de la programación, el cambio es algo más pausado, sin embargo todavía hoy siguen surgiendo nuevos lenguajes OOP o modificaciones de los existentes, nuevas bibliotecas de clases que facilitan la tarea al programador y otros muchos casos.

A pesar de que las técnicas de programación que se utilizan hoy día no son demasiado modernas (en informática hablar en términos de años suele ser considerado como "antigüedad"), su aplicación a determinadas circunstancias no siempre es posible, de ahí la constante evolución. A esto hay que añadir que con las aplicaciones de hoy día no es suficiente con que funcionen bien, el usuario quiere que funcione "bien y bonito"; es decir, que dispongan de un entorno atractivo y actual, que facilite su uso y que permita personalizarla (cambiando colores, formas, etc.). Este hecho, aunque nada tiene que ver con el aspecto técnico de la aplicación, favorece la distribución de la aplicación y su extensión a otros campos (no sólo al caso de contacto esférico).

En lo referente a las condiciones técnicas de la aplicación, se describen a continuación una serie de mejoras desde el punto de vista del autor que serían conveniente realizar para posteriores versiones, no con el propósito de que otra persona las realice, sino con la idea de que la programación llevada a cabo tiene cierto carácter personal y puede realizarse de otras muchas formas.

8.1.1 La modularización

El propósito de modular el código es facilitar la revisión y las posibles mejoras futuras, sin hacer lo mismo con la programación restante. El ejemplo que siempre se muestra en las escuelas informáticas en el de un mueble con cajones para almacenar productos varios, cuantos más cajones (mayor modularización) de mayor poder de clasificación se dispone y por tanto más fácil sería encontrar un determinado producto.

Pero del mismo modo que conlleva una serie de ventajas, arrastra algunos inconvenientes. El repartir el módulo en demasiados métodos (modulares) hace que para seguir un proceso, las partes que lo componen estén muy dispersas, con lo que se dificulta la observación del proceso.

En virtud de lo anterior, lo deseable sería una situación en la que el código estuviera modulado pero en la medida justa. Como puede comprender el lector, no es tarea fácil, constituyendo uno de los mayores inconvenientes de la programación: la solución de compromiso por la que se opte finalmente, debe reunir estas cualidades.

Con respecto a la programación de *Fretting Fatigue* en su versión 1.1, conviene observar que el código podría ser más modular, pero conlleva que el reparto del cálculo no pueda observarse con un "golpe de vista".

8.1.2 Las librerías

Otra forma de modular es agrupar las clases que no requieran de formularios gráficos en librerías externar a la aplicación. Aunque esta idea no es nueva (se propuso ya en el Capítulo 5), sigue constituyen la principal fuente de mejora para las aplicaciones modernas. Hoy día, las actualizaciones del S.O., de las aplicaciones más comunes y de los programas más usuales, se realizan a través de ficheros DLL, que son librerías al fin y al cabo.

Se propone al programador que retome la aplicación con la descripción, utilidad y forma realizada en este trabajo, y adopte la postura adecuada para permitir este tipo de actualizaciones.

Como ejemplo, en la aplicación se realiza una integración numérica por un método específico (método de los trapecios); esta tarea, externa totalmente a la aplicación, puede incluirse en una librería DLL. De esta forma, para tratar otro tipo de datos puede hacerse uso de ella o bien modificar el método para que se haga de otra forma sin modificar la aplicación, simplemente cambiando una librería por otra.

8.1.3 La base de datos de material

Este es uno de los aspectos que más me ha preocupado desde el día que se incluyó en la aplicación. Sin duda representa una mejora bastante considerable por el hecho de automatizar la adquisición de datos de los materiales; pero conlleva otros aspectos no tan buenos: el tiempo de conexión es elevado (comparado con el resto de procesos de la aplicación) y requiere de otras aplicaciones para un correcto funcionamiento (Microsoft Access).

Por ello, la alternativa que se presenta, es que, para un número reducido de materiales (una veintena materiales por ejemplo) se realice la adquisición por ficheros

planos como los manejados en el resto de la aplicación (ficheros tabulados). Esta solución sería factible para un número reducido de materiales, cuando éste número aumente (mil materiales por ejemplo) la gestión de la base de datos daría su frutos.

Otra modificación sobre la propuesta sería hacer posible la lectura de un fichero de datos de materiales determinado por el usuario; con ello podría disponerse de datos para distintos materiales agrupados en diferentes archivos. Por ejemplo, un fichero para las aleaciones de aluminio, otro para los aceros, etc.

8.1.4 La representación gráfica

La clase FormGrafico ya ha sido mencionada como la más costosa de desarrollar; pero a cambio puede afirmarse que es producto 100% propio. En Internet pueden localizarse librerías que realizan tratamientos de datos para su representación gráfica en una aplicación adicional; pero uno de los valores añadidos de este proyecto es el aprendizaje del lenguaje informático C++.NET, y por tanto la realización de esta clase conlleva el aprendizaje de parte del lenguaje.

Por otro lado, las aplicaciones de representación gráfica tienen un carácter bastante general, en el caso de este Proyecto era interesante disponer de una herramienta muy específica y con unas características que no ha sido posible localizar.

De otra forma, si el lenguaje de programación hubiera sido Basic (Visual Basic), la incorporación de un objeto Ole permitiría disponer de todas las herramientas de otras aplicaciones (como Microsoft Excel) dentro de la propia aplicación *Fretting Fatigue*, lo cual representaría una gran ventaja. En el caso de C++.NET, esta situación no es inalcanzable, pero conlleva una dificultad muy elevada para la información que se obtendría.

Se propone por tanto algunas mejoras en la clase FormGrafico para que su aspecto sea más atractivo. En particular, pueden incluirse marcas sobre los ejes para que sirvan de referencia, así como otras líneas más fina (paralelas a los ejes) que tengan la misma función.

8.2 Las mejoras del modelo mecánico

En esta apartado se incluyen tres tipos de mejoras: algunas que se tenían presente incluso antes de comenzar la aplicación (el caso del contacto cilíndrico); otras que se pueden implementar fácilmente puesto que lo permite la aplicación (importar geometría procedente de un programa de MEF); y un tercer grupo que representa a todas las demás.

8.2.1 El contacto cilíndrico

Como se ha contemplado en el desarrollo de los capítulos anteriores, las clase de cálculo están bastante distribuidas: las clase que proporciona las tensiones, es distintas a la que gestiona el cálculo (en una línea, por ejemplo), y por supuesto distinta de las que evalúan los criterios y las leyes de crecimiento. Esto posibilita que modificando la clase Tensiones a la del contacto cilíndrico, y leves modificaciones el la gestión de la misma (calculo en una línea), la aplicación esté preparada para trabajar dicho problema.

8.2.2 Una geometría desde MEF (Método de los Elementos Finitos)

Los programas de Elementos Finitos (EF) pueden proporcionar unos resultados que en la mayoría de los casos deben ser contrastados con otros modelos más simples. En este sentido, la capacidad de calcular en los mismos puntos que el programa de EF representa una gran ventaja, puesto que permite realiza la comparación de forma directa.

8.2.3 Las mejoras no contempladas

Con respecto a ésta hay poco que decir; sí procede orientar al programador acerca de las características que debe reunir la aplicación, tal y como se ha hecho en los documentos Manual de Usuario y Manual de Programador (el actual), así como animarle a intentarlo (y felicitarle en caso de conseguirlo).