1

Introducción

1.1. ¿Qué es y para qué sirve este libro?

Este libro propone un ***Ideario*** que ayude a ***gestionar tus proyectos software.*** Está destinado, en especial, a los "responsables de proyectos software de gestión". No obstante, espero que pueda ser una lectura útil y amena para cualquier persona relacionada con la informática.

Podrás encontrar multitud de referencias bibliográficas relacionadas con lo que debe ser la gestión "ortodoxa" de proyectos y todos sus temas colindantes como la ingeniería del software y la gestión de equipos humanos. Me refiero, por ejemplo, a:

- Metodologías específicas de gestión de proyectos como Prince2 y PMBOK.
- Metodologías genéricas como Métrica y Rational Unified Process.
- Modelos de referencia como CMMi e ISO/IEC 15504 SPICE.
- Modelos de gestión de equipos como el Team Software Process del SEI (Software Engineering Institute).
- "Enciclopedias" de ingeniería del software, como las de Pressmann y Sommerville.

Este libro no es ningún extracto de toda esta bibliografía; de hecho, no encontrarás en el mismo "procesos", ni "formularios", ni "listas de tareas", ni "entregables", ni nada similar.

Aquí sólo encontrarás ideas, principios y buenas prácticas; esto es, una filosofía de trabajo basada en la experiencia y en el sentido común.

Siempre que aludo al sentido común, recuerdo la siguiente anécdota:

> *Hace unos años, un político español criticó a otro aduciendo que era muy inculto y de escasísimo bagaje cultural.*
>
> *El político aludido replicó de la siguiente forma:*
>
> *"Es cierto, mi bagaje cultural es quizás escaso, pero prefiero ser algo inculto a tener una formación tan amplia que no me quede lugar para el sentido común.".*

Parafraseando a Lou Marinoff ("Más Platón y menos Prozac"), propongo ***un poco más de ideas y sentido común, un poco menos de metodologías***.

He optado, como modo de exposición de mi Ideario, por un lenguaje imperativo y de "tú a tú". Lo he elegido porque me parece la forma más directa y amena de transmitirte mi mensaje. Esto no significa; ni mucho menos, que me considere en posesión de la verdad. De hecho, "mi verdad" ha ido cambiando con el transcurso del tiempo y seguro que seguirá cambiando conforme sigo trabajando en nuestra nada fácil profesión.

1.2. Empecemos con el "desaprendizaje"

Es posible que para mejorar tu capacidad de gestión de proyectos software, una de las primeras cosas que te conviene hacer es "***desaprender***" algunas ***ideas y comportamientos*** que te lastran y te restan energía. Te enumero a continuación una serie de frases que *generalmente* son síntomas de una forma equivocada de enfocar los proyectos:

> *"...No puedo continuar porque no dispongo de suficientes especificaciones".*

"... Necesito más información para estimar plazos".

"...Necesito 2 semanas más para lo que tenía previsto acabar mañana".

"...He hecho una planificación mediante un diagrama de Gannt con el producto Open Project afinando a la hora".

"…Pero, !si tú me dijiste que lo hiciera así! "

Algunas de estas frases se retoman y comentan posteriormente. Para las otras, espero que si consigues acabar la lectura de este libro, tú mismo seas capaz de comprender mi rechazo a las mismas.

En el instituto IESE (Instituto de Estudios Superiores de la Empresa) se solía decir que las personas se pueden clasificar profesionalmente en dos grandes perfiles: aquellos que buscan la verdad (éstos son los filósofos) y aquellos que tratan de funcionar con la realidad (éstos son los gestores).

Los desarrollos software necesitan más de gestores que de filósofos.

Muchos de los libros de **gestión de proyectos** clásicos te proponen formas de trabajo basadas en una realidad inexistente y que "debería ser pero no es", algo así como lo que se cuenta en la siguiente "historia" (que muy seguramente es aplicable a muchos de los supuestos "gurús" de la ingeniería del software):

"Se cuenta de un joven chino que dedicó toda su vida a aprender el arte de cazar dragones, hasta que estuvo seguro que ya dominaba todas las técnicas de cómo cazar dragones.

En ese momento se percató de que no había en el mundo dragones que pudieran ser cazados y el joven se dedicó a enseñar cómo cazar dragones..."

Te ruego que estés abierto a "desaprender", a "vaciar tu taza", tal y como tuvo que hacer el aprendiz de Budismo Zen que te comento a continuación:

Nan-in, un maestro japonés del período Meiji (1868-1912), recibió a un profesor universitario, quien vino a preguntarle acerca del Zen.

Nan-in sirvió el té. Llenó la taza de su visitante y continuó vertiéndolo.

El profesor observó cómo la taza rebosaba, hasta que no pudo contenerse más y gritó:

"La taza se desborda. ¡Ya no cabe más!".

Nan-in replicó:

"Al igual que esta taza, usted rebosa de sus propias opiniones y especulaciones. ¿Cómo puedo enseñarle Zen a menos que primero la vacíe?".

Lo primero que has de comprender es que **la informática es todavía una disciplina joven e inmadura**, apenas tiene 50 años de vida, y hay que lidiar constantemente con:

- Unas especificaciones imprecisas, cambiantes o a veces casi inexistentes.
- Unos equipos humanos volátiles y/o con formación insuficiente y heterogénea.
- Unos interlocutores, usuarios y expertos en el "negocio" que tienen dificultades para concretar sus necesidades con el suficiente detalle y claridad.
- Unos jefes que a veces no comprenden nuestras dificultades.
- Unos plazos que suelen ser excesivamente cortos.

En un contexto como el anterior, los "informáticos" nos sentimos a menudo presionados, infravalorados, incomprendidos o frustrados. No hemos de atormentarnos por estas sensaciones y, por el contrario, tenemos que sentirnos como "exploradores" de una disciplina que tiene todavía muchísimo camino por recorrer.

Piensa en **la actividad de desarrollo de software como en el proceso de aprendizaje que describe la "metáfora de la carta y la papelera"** (figura 1.1).

El proceso de desarrollo de proyectos informáticos se aproxima más a la forma de escribir una carta, a la que subyace una serie de ciclos de prueba-error, que a un cuidadoso proceso lineal de diseño, planificación y ejecución.

Figura 1.1

En este contexto, quizás lo más apropiado no es aferrarse con rigidez a una metodología pautada, sino afrontar el reto pertrechado de ideas flexibles que, aplicadas adecuadamente, nos ayuden al obtener un producto que satisfaga las necesidades de nuestros clientes a un coste razonable.

En el libro denominado "Fábula de los seis ciegos y el elefante (la maestría en la gestión de proyectos)" el consultor de gestión de proyectos David A. Schmaltz dice, entre otras cosas:

> *Al principio yo también encontré una metodología que, con la ayuda de mis colegas, infligimos a nuestros clientes. Los resultados fueron decepcionantes para todos.*
>
> *Después de más de un cuarto de siglo gestionando proyectos llegué a la conclusión de que la teoría de la gestión de proyectos informáticos carecía totalmente de utilidad en el mundo real. La denomino 'el placebo más popular que existe hoy en día'.*
>
> *No creo para nada en las técnicas de moda, tan decepcionantes en la práctica, que ofrecen el Project Management Institute (PMI), el Software Engineering Institute (SEI) y las diferentes consultoras que trabajan con estos modelos..."*

Con lo anterior no pretendo que entones un ¡Muera la metodología!, ¡Muera el proceso!, sino justificar la conveniencia de complementar el proceso y la metodología con un ideario abierto y flexible que incremente tus opciones y te dé más posibilidades de culminar tus proyectos con éxito (figura 1.2).

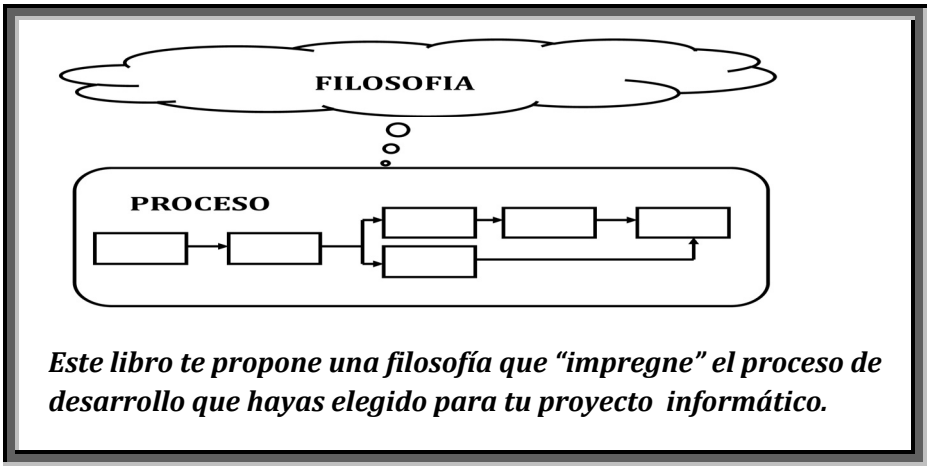

Figura 1.2

1.3. Otra forma de ver los desarrollos informáticos

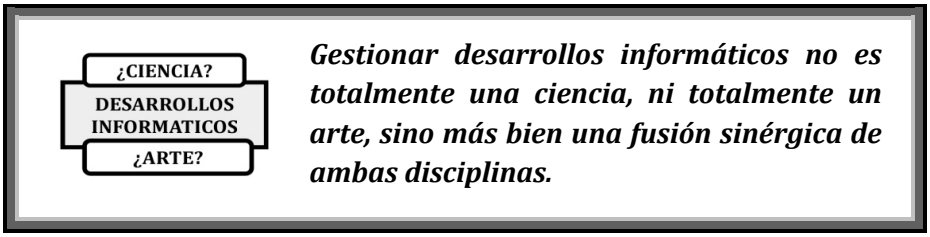

Figura 1.3

Un proyecto informático no es una actividad rutinaria y sistemática a la que basta aplicar una serie de técnicas descritas en libros al uso para alcanzar el éxito.

Cada proyecto es, lo queramos o no, ***una pequeña aventura*** en la que se presentan dificultades e imprevistos que hay que ir solventando sobre la marcha.

A esta pequeña aventura uno ha de ir bien armado de "ideas" y ser lo suficientemente flexible y ágil para tomar decisiones correctas.

Piensa que la informática no es, hoy por hoy, una disciplina ingenieril más:

> Cuando un ingeniero naval, por ejemplo, acomete la construcción un barco, el porcentaje de incertidumbre de que dicho barco sea un total fracaso es muy bajo. Sin embargo, cuando un "ingeniero" informático acomete un proyecto, la posibilidad de que dicho proyecto fracase es significativa.

Suelo decir que nuestro trabajo tiene que ver más con la "***albañilería del software***" que con la ingeniería. La albañilería es una actividad también noble pero más "pegada al terreno" y menos científica que la ingeniería. Requiere más pragmatismo. Esto no lo debes interpretar como un deshonor, pues también hay buenos y malos albañiles.

He sido aficionado al ajedrez y me viene a la memoria lo que me solía decir un contrincante habitual:

> *"Tengo el tablero sembrado de trampitas, y simplemente estoy esperando, a que caigas en alguna de ellas."*

A menudo, ***los desarrollos informáticos esconden "celadas"*** que hay que saber identificar y superar para no quedarse "enredado" en las mismas.

En cualquier caso, no te engañes, cuanto más ***bagaje teórico*** tengas, mejor pertrechado irás a la aventura. Pero has de tener claro que la teoría de la ingeniería del software y de la gestión de proyectos no son "Biblias" en las que has de creer y seguir a "pies juntillas". Son, simplemente, ***un recurso más, pero no el único,*** para conseguir llevar a buen fin tu proyecto.

La cultura de gestión que aquí se propone es coherente con el estudio que IBM realizó sobre el uso de herramientas CASE en el que se extrajeron las siguientes conclusiones:

- *Los equipos exitosos enfatizaban que no habían seguido métodos formales ni herramientas CASE y que habían estimulado la comunicación y las pruebas.*
- *Los equipos con problemas no entendían sus fallos, si habían cumplido con los métodos formales.*

- *La experiencia se repitió durante toda una década por todo el mundo y con distintas herramientas.*

Te recomiendo que interiorices que un desarrollo de software es muy parecido a lo que entiende la metodología liviana de desarrollo "SCRUM":

> *"SCRUM entiende el desarrollo de software como algo empírico y no determinista. Esto significa que SCRUM comprende que crear software es algo muy complejo, sujeto a multitud de variaciones e incertidumbres, sobre todo al comienzo. Además cada problema es diferente y no existe una solución mágica o bala de plata que se pueda aplicar a cualquier proyecto.*
>
> *Desarrollar software es, todavía hoy en día, una tarea ardua, poco madura y artística. La experiencia y el voluntarismo son, hoy por hoy, definitivos para aumentar las posibilidades de éxito del proyecto.*

1.4. Organización del Ideario

No me ha resultado fácil organizar las ideas que intento transmitirte. Muchas de ellas se solapan y realimentan. Finalmente me he decidido por estructurarlas en 6 secciones (ver figura 1.4).

La primera sección son las ideas para tu equipo de trabajo; esto es, el Ideario para ***todas y cada una de las personas de tu equipo***. Como tú formas partes del equipo, aplícatelas también.

La segunda es un Ideario suplementario que también debes aplicarte a ***ti mismo*** en tu rol de responsable del proyecto.

La tercera es el Ideario de ***cómo has de gestionar a las personas de tu equipo***. Esta sección, a diferencia de los anteriores, en vez de tratar de ideas aplicables a uno mismo, te propone ideas para tus ***relaciones*** con las personas de tu equipo.

La cuarta se centra en ***tus relaciones con otras personas*** distintas a tus colaboradores: tu jefe, tus clientes y el resto de las personas de tu organización.

La quinta trata de la gestión del ***proyecto y del producto*** (la aplicación).

La sexta contiene reflexiones sobre la ***documentación, la tecnología y el proceso de desarrollo.***

PROYECTICA
es un ideario compuesto de 6 secciones.

1. **Tu equipo.**
2. **Tú mismo.**
3. **La gestión de tu equipo.**
4. **Tus relaciones con otras personas.**
5. **La gestión del proyecto y del producto.**
6. **Documentación, tecnología y proceso.**

Figura 1.4