CHAPTER 7

Estimaciones

7.1 Lo que deseamos estimar

Una planificación adecuada de un proyecto de desarrollo de *software* requiere conocer tempranamente dos magnitudes: el tiempo necesario para realizarlo, medido por ejemplo en semanas, y el esfuerzo involucrado en ello, medido por ejemplo en semanas-hombre. Lo primero porque pueden haber *deadlines* duros de entrega, compromisos de entrega e incluso multas en caso de atrasos. Lo segundo porque está directamente relacionado con los costos.

A pesar de que el foco principal está en estimar los números para esas dos magnitudes, es muy común que la primera estimación que hagamos corresponda al tamaño o envergadura del producto de *software* a desarrollar. Esto porque esta magnitud va a incidir directamente en las dos anteriores y es más fácil partir por allí.

Es importante hacer notar también que se hacen estimaciones desde antes que el proyecto parta hasta que el proyecto termina. Por ejemplo, las estimaciones preliminares deben ser inmediatamente corregidas al completar el primer sprint, al cambiar la composición del equipo o por alguna situación excepcional.

7.2 Las Magnitudes y sus Métricas

La primera magnitud a considerar es el tiempo. Es lo más sencillo y se suelen utilizar meses, semanas, días y horas. Es importante destacar que la sola elección de la unidad de tiempo va a transmitir en forma implícita un sentido de precisión en la estimación. Así, por ejemplo, si se plantea que un desarrollo durará 14 días, uno esperaría un error de más menos algunos días. Si,

en cambio, uno dice que serán 2 semanas, uno tiende a entender que puede haber un error menor.

La segunda magnitud es el esfuerzo. Las métricas usadas aquí son el hombre mes, hombre semana u hombre hora y que corresponde al producto del tiempo empleado por el número de personas que trabaja en ello. Por ejemplo, si una determinada pieza de *software* se construyó en 4 meses por un equipo de 5 personas, el esfuerzo asociado fue de 20 hombres mes.

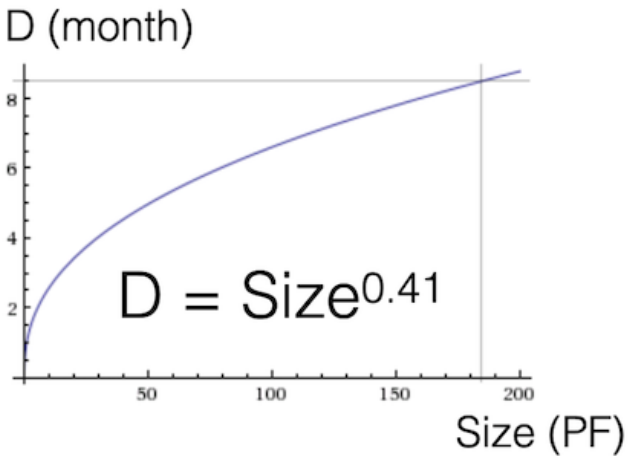
Es extremadamente importante al considerar el esfuerzo lo que Fred Brooks alertó en su célebre libro El Mítico Hombre Mes en el sentido que en el caso del *software* personas y tiempos no son intercambiables. Tomando el caso del ejemplo anterior en que el esfuerzo fue de 20 hombres mes, no necesariamente un equipo de 10 personas habría tardado solo dos meses en construirlo.

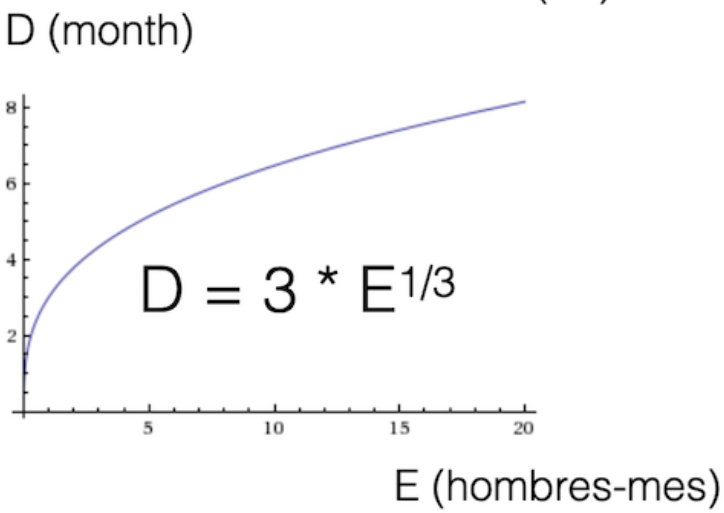
Tanto el esfuerzo como el tiempo dependen directamente del tamaño de la pieza de *software* o envergadura del proyecto. El tamaño se puede medir con distintas métricas, pero la más utilizada es la de líneas de código fuente (*LOC*). Esta métrica considera simplemente el número de líneas de código (posiblemente eliminando líneas vacías y comentarios) en el programa final. Más adelante retomaremos este tema y examinaremos otras métricas para medir el tamaño.

7.3 Del tamaño a la Duración y al Esfuerzo

Uno puede utilizar información empírica para construir modelos que permitan relacionar estas magnitudes. Lo ideal es tener modelos propios, pero si no se tienen pueden usarse los de la industria. A continuación se muestra la esencia de esta idea con un modelo sacado de la industria en USA.

El modelo relaciona la duración en meses con el tamaño en puntos de función (más adelante lo explicaremos) y la duración en meses en función del esfuerzo de desarrollo (ver figura).





Supongamos que hemos realizado una estimación de tamaño de 100 puntos de función. Entonces"

S = 100

D = 100 ** 0.41 = 6.6 meses

6.6 = 3 * E ** (1/3) => E = (6.6/3) ** 3

E = 10.6 hombres mes

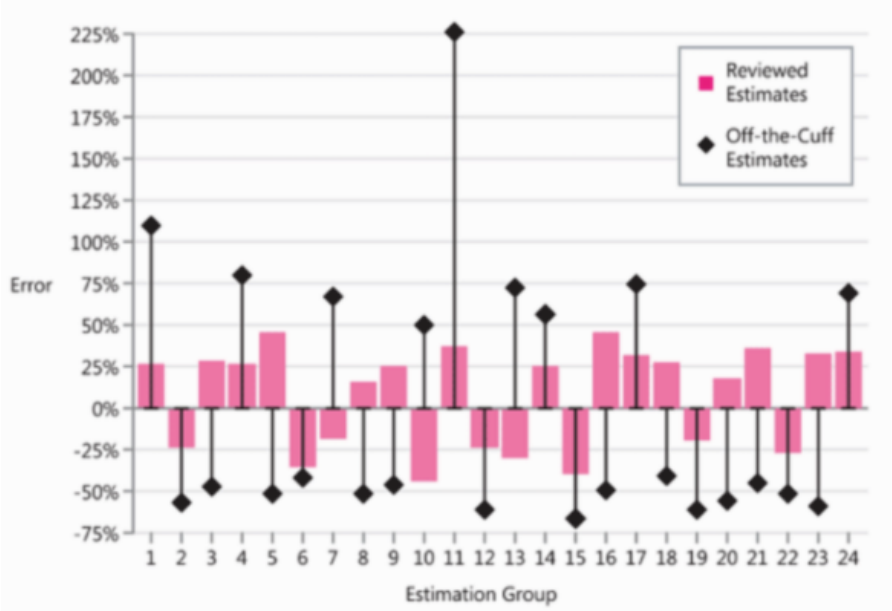
10.6 hombres mes con una duración de 6.6 meses sugiere usar a dos personas para este proyecto.

7.4 Evitar la respuesta improvisada

Este es un error en que incurren muchas veces los ingenieros poco experimentados. De pronto alguien pide una opinión del tipo "cuanto crees que tomaría hacer algo como ..." y, en lugar de callar, el ingeniero responde "e imagino que una semana". La respuesta correcta debería ser "lo estudiaré y

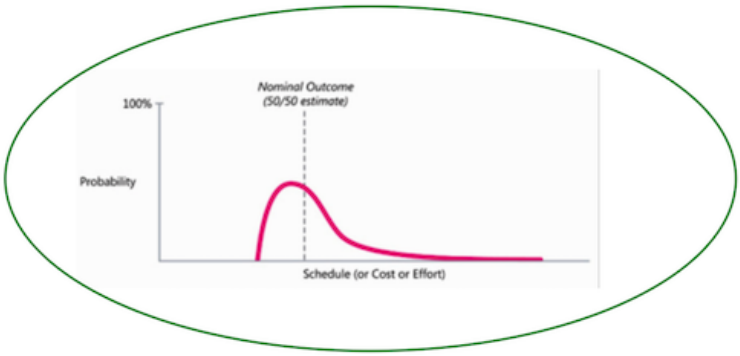
te contesto en un rato más". Incluso un análisis superficial, pero con calma, va a entregar números mucho mas cercanos a la verdad.

El gráfico siguiente muestra lo alejada que está la estimación rápida (rombos negros) de la estimación realizada en forma metódica más tarde.



Nunca hay que olvidar que una estimación siempre es una afirmación probabilística. Es decir, cuando uno afirma que el desarrollo tomará, por ejemplo, 14 semanas, lo que se está en realidad afirmando en que 14 es lo más probable, pero en realidad hay una distribución de probabilidad alrededor de ese valor (ver figura)

En realidad la curva será más bien como muestra la siguiente figura, ya que es mucho mas común que uno se quede corto con la estimación y no al revés.



Esto nos lleva a comparar el problema de subestimar con el de sobreestimar. Sobreestimar significa, por ejemplo, estimar el tiempo de desarrollo en 8 semanas cuando en realidad solo se requerían 6 semanas. Subestimar es lo contrario: se estimaron 6 semanas y en realidad se requerían 8.

7.5 Problemas de Subestimar y Sobreestimar

Obviamente que no puede ser bueno ninguna de las dos cosas, pero los problemas que trae el subestimar son mucho mayores que el de sobreestimar.

El sobreestimar el tiempo de desarrollo está asociado a dos problemas conocidos como la **ley de Parkinson y la ley de Goldratt**.

La ley de Parkinson establece que el trabajo se expandirá hasta usar todo el tiempo disponible.

La ley de Goldratt, también conocida como el síndrome de estudiante, dice que si hay demasiado tiempo se va a malgastar el tiempo al comienzo

A pesar de estos problemas, la penalización por sobreestimar es mucho menor a la de subestimar. En el primer caso, la penalización crece linealmente con la magnitud de la sobreestimación, en cambio, en el segundo caso, la penalización puede crecer en forma exponencial por las razones que veremos a continuación.

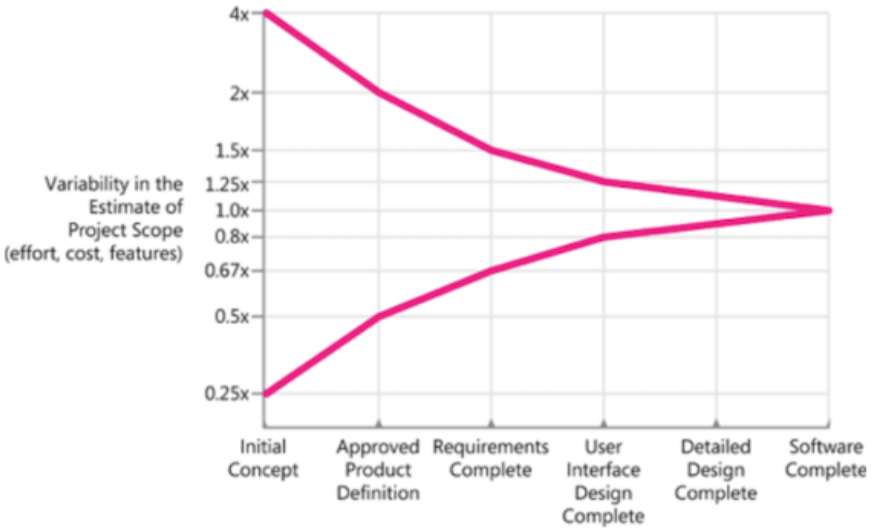
La figura muestra como la penalización crece mucho más rápido cuando hay una subestimación en comparación con el caso contrario.

¿Por qué es tan problemático subestimar el tiempo (o el esfuerzo) de desarrollo? Hay muchas razones, pero básicamente trabajar en un modo en que el proyecto está permanentemente atrasado genera un enorme nivel de estrés en el equipo que puede traducirse en una baja de rendimiento general sin contar problemas de calidad o el riesgo de perder a parte del equipo a medio camino.

A veces se produce una suerte de efecto dominó. Se deben reducir tiempos de análisis y diseño, lo que a la larga genera mas atrasos. Hay que reagendar reuniones, lo que consume más tiempo del originalmente planeado, hay que ajustar los planes, etc.

7.6 El Cono de la Incertidumbre

Ya hemos dicho antes que se hacen estimaciones a lo largo de toda la duración de proyecto y que los errores en las estimaciones disminuyen a medida que el proyecto avanza. Inicialmente, puede haber un error de 4 veces hacia arriba o hacia abajo de la magnitud que termina siendo pero a medida que el proyecto avanza esta incertidumbre disminuye generando una curva que se conoce como el cono de incertidumbre. La figura muestra el cono para un modelo de proceso clásico de cascada pero es igualmente válido en el caso de desarrollos ágiles, solo que en lugar de tener etapas en el eje X tendremos Sprint 1, Sprint 2, etc.



Hay dos mecanismos importantes para ir reduciendo este cono:

- el desarrollo iterativo e incremental
- la descomposición en unidades pequeñas

Al primero ya nos hemos referido, pero volveremos más adelante. Lo segundo es porque es mucho más fácil estimar una tarea pequeña que una

grande. Por ejemplo, cuanto tiempo me toma construir un sitio web para una empresa de *delivery* puede ser complejo, pero estimar el tiempo para implementar la autenticación del usuario es mas tratable En este sentido, la división de la funcionalidad en base a relatos de usuario nos será de gran utilidad.

7.7 Fuentes de Error en las Estimaciones

Existen cuatro fuentes de error posibles relacionados con las estimaciones:

Información imprecisa sobre el proyecto

No se entendió bien lo que se quería hacer o el análisis no fue lo suficientemente profundo. Se dijo que era una página web simple y resultó que se trataba de un *ecommerce* completo.

Información imprecisa sobre las capacidades del equipo

Si los integrantes del equipo nunca han trabajado juntos o si lo han hecho, pero en otro tipo de proyecto es muy difícil estimar su productividad. ¿Podrán sacar 2, 3 o 6 relatos en un sprint? Es sabido que hay diferencias muy grandes de productividad entre programadores individuales y mucho más si incorporamos aspectos de trabajo en equipo (revisar la clase anterior)

Imprecisiones del proceso mismo de estimación

A estas alturas debería estar claro que el proceso de estimación puede tener muchas limitaciones: datos incompletos o que corresponde a información de la industria y no del equipo, modelos aproximados, uso de productividades constantes, no considerar el tipo de proyecto específico, etc.

El Proyecto es del tipo “blanco móvil”

Un proyecto de tipo blanco móvil es aquel que cambia permanentemente según como haya amanecido el encargado del proyecto. Se espera que ocurran cambios, pero a veces se está en presencia de una periodicidad en los cambios que hace casi imposible estimar nada.

7.8 Precisión, exactitud y uso de las métricas correctas

El ejemplo que comúnmente se da para comparar precisión con exactitud es el valor de PI. Si yo digo que el valor de PI es 3 no es muy preciso. Un valor de 3.14 es más preciso y un valor de 3.14159 es aún más preciso. Si yo digo que

el valor de PI es 5.123233341 estoy también comunicando precisión, pero el valor no es exacto.

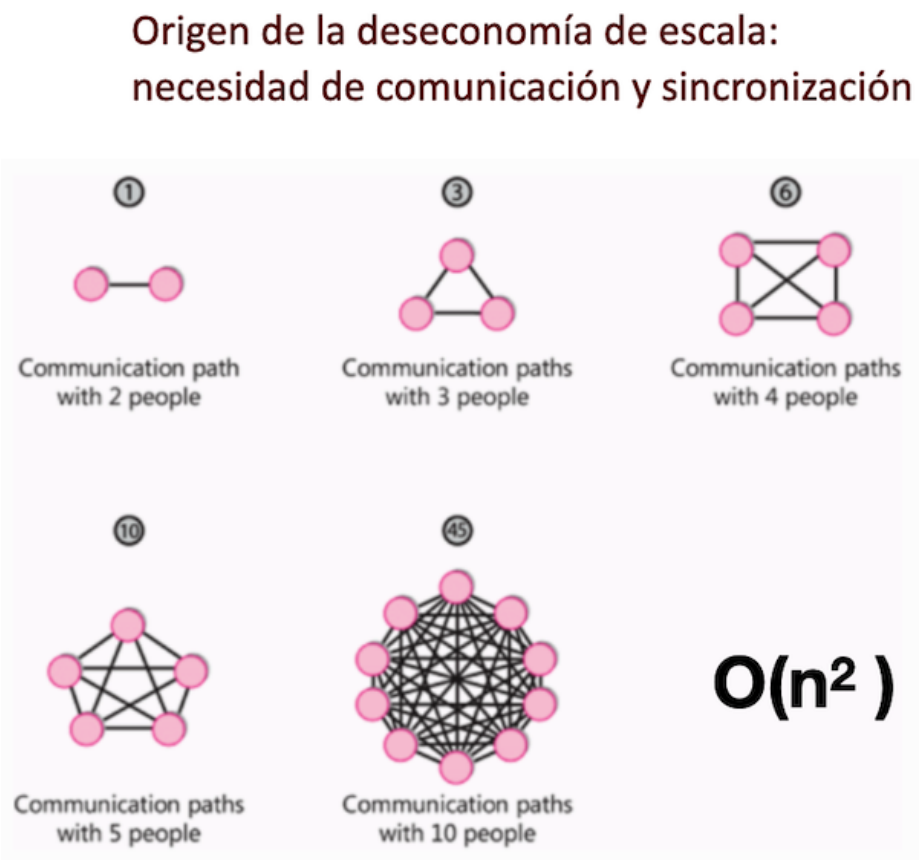
En el caso de una estimación, la métrica o unidad utilizada transmite un sentido de precisión, por lo que hay que ser cuidadoso con ello. Si uno ha hecho una estimación de 3 meses para la duración del proyecto, pero lo comunica diciendo que tomará 90 días, un error de 7 días será un problema. Si, en cambio, dice que tomará 12 semanas y termina siendo 13, es mucho mejor tolerado. Si lo que se dijo son 3 meses, incluso un atraso de 2 semanas estará dentro de lo aceptable.

7.9 Todo parte con el tamaño

Es lo primero que se tiene a mano para llegar finalmente a tiempo de desarrollo y esfuerzo. Se sabe que a mayor tamaño mayor esfuerzo y mayor tiempo de desarrollo. Sin embargo, en el mundo del *software* ocurre algo inusual: la deseconomía de escala. Esto hace que mientras más grande el proyecto vamos a tener una menor productividad. Esto significa que un producto de 100.000 LOC probablemente requerirá más de 10 veces el tiempo que toma hacer uno de 10.000 LOC.

2020-09-29_06-46-32_PM.png

La razón de esta deseconomía de escala es que a medida que el proyecto crece necesitaremos más personas en los equipos, lo que se traducirá en mucho más tiempo de comunicación y coordinación entre ellos. Si tenemos dos nodos en un grafo, solo hay un arco posible que los conecte. Si hay 3 nodos son 3 arcos y si hay 5 son 10 arcos. En general, el número de arcos conectando a N nodos es N(N-1)/2 y este número crece con el cuadrado de N.



Hablemos ahora de las unidades. Hemos dicho que las líneas de código fuente (LOC) son ampliamente utilizadas como una métrica del tamaño. Aparte de ser muy simple, esta métrica tiene otra gran ventaja: que una vez terminada la construcción del *software* se puede saber en forma muy precisa el tamaño. Basta contar las líneas de código. Sin embargo, la métrica tiene muchos inconvenientes también.

La primera dificultad es que si bien es fácil saber el valor al terminar el proyecto, es extremadamente difícil hacer una estimación del número de líneas de código al comenzar el proyecto. Adicionalmente, esta métrica es dependiente de demasiados factores:

- el lenguaje de programación que se esté usando (hay lenguajes mas verbosos que otros)
- el estilo de escritura del código (indentación, posición de las llaves, saltos de línea, etc)
- calidad del código (`i++` vs `i = i + 1`)
- uso de comentarios y líneas en blanco

Necesitamos entonces métricas de tamaño que

- no dependan del lenguaje que se usará
- faciliten la estimación temprana

Puntos de Función

Esta es una métrica que fue extremadamente popular y de amplio uso en los 80's y 90's, pero lentamente ha ido siendo dejado de lado por los tremendos cambios que han ocurrido en el tipo de software y la forma en que se desarrolla. Esta métrica reúne los dos requisitos anteriores: puede ser usada en forma temprana y no depende del lenguaje que será utilizado.

Para entender esta métrica hay que entender el tipo de programa típico de esa época. Los elementos principales que caracterizaban el tamaño resultante eran cinco factores que eran fácilmente estimables:

- número y complejidad de entradas (inputs)
- número y complejidad de salidas (outputs)
- número y complejidad de consultas
- número y complejidad de archivos (o tablas) internos
- número y complejidad de archivos (o tablas) externos

La complejidad de cada uno de estos elementos se estimaba en tres niveles: baja, mediana y alta y de acuerdo a ello producía un puntaje de acuerdo a la siguiente tabla.

Tipo	Baja	Mediana	Alta
Entradas	x3	x4	x6
Salidas	x4	x5	x7
Consultas	x3	x4	x6
Archivos Internos (tablas)	x7	x10	x15
Archivos Externos (tablas)	x5	x7	x10

Así, 2 entradas de baja complejidad producen 2 x 3 = 6 puntos y 3 consultas de alta complejidad contribuyen con 3 x 6 = 18 puntos. Sumando todo se obtiene un número que se conoce como puntos de función no ajustados (PFNA) que serán reducidos o incrementados en hasta un 35% dependiendo de un factor de complejidad del programa en general.

Este factor de complejidad considera 14 áreas en que se asigna un puntaje de 0 a 5:

1. Comunicaciones
2. Funciones distribuidas
3. Objetivos de desempeño
4. Configuración sobrecargada
5. Tasa de transacciones
6. Entrada de datos *online*
7. Eficiencia para usuario
8. Actualización en línea
9. Proceso complejo
10. Reuso
11. Facilidad de instalación

12. Facilidad de operación
13. Varios sitios
14. Facilidad de mantención

lo que produce un número entero N que va desde 0 a 70. A partir de este número, el factor de complejidad se calcula como:

FC = 0.65 + N/100

De allí que

0.65 ≤ FC ≤ 1.35

El número final de puntos de función es entonces

PF = PFNA * FC

Relatos y Puntos de Relato (*storypoints*)

Una métrica que comparte con la de puntos de función, el que no depende del lenguaje y sirve para estimaciones tempranas es la de puntos de relato (*storypoints*). Sabemos que el número de relatos de usuario es un buen indicador de la magnitud de lo que queremos construir. Lo que agrega la métrica de puntos de relato es tomar en consideración que hay relatos más complejos de implementar que otros.

La idea entonces es ponderar la complejidad del relato incorporando un multiplicador. Estos multiplicadores crecen con la complejidad del relato en una escala de Fibonacci o Exponencial.

Al usar una escala de Fibonacci, un relato simple cuenta por 1 punto, uno mas complejo 2, el siguiente 3, luego 5, etc. Con una escala exponencial, los puntos de relato crecen más rápido: 1, 2, 4, 8, ...

Así, por ejemplo, supongamos que clasificamos los relatos en solo 4 tipos, podríamos usar solo multiplicadores 1 para el más simple y 5 para el más complicado.

Estimación Colaborativa

La idea es hacer que cada miembro del equipo haga una estimación personal y secreta del puntaje de un relato. Luego se descubren simultáneamente las “apuestas” y se genera una pequeña discusión tratando de llegar a un acuerdo. Por ejemplo, alguien que asignó un puntaje 1 puede no haber visto determinada complejidad que sí vio quien le puso 5, pero al conversar cambia de opinión. Se pueden hacer varias rondas hasta converger en un determinado valor.

Esta dinámica es tan popular que se venden cartas con los números (Fibonacci o Exponencial) para hacer las apuestas. Cada jugador (miembro del equipo) necesita un mazo de cartas.

2020-09-29_07-51-39_PM.png

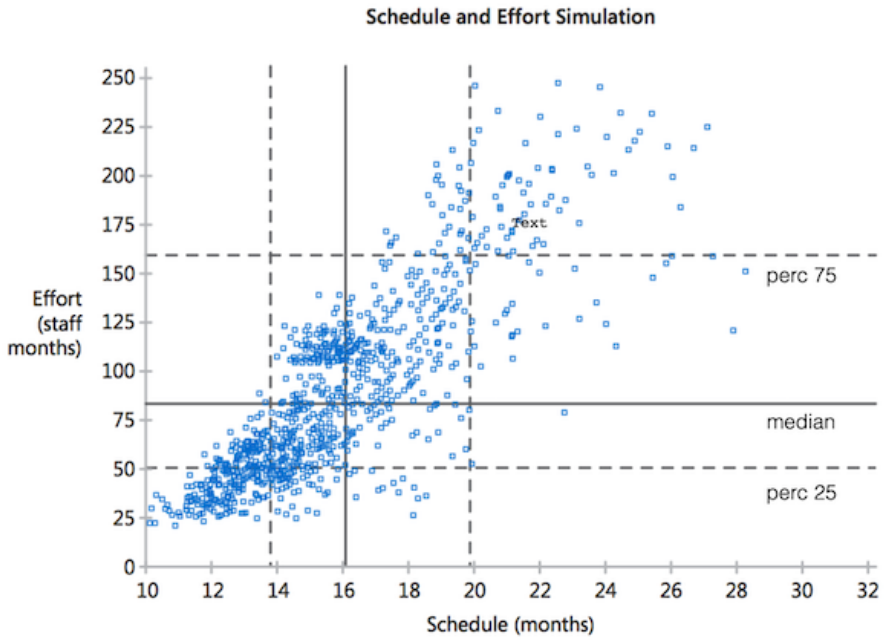
7.10 Del tamaño al esfuerzo

Sabemos que el esfuerzo no crece linealmente con el tamaño, pero cuando los números son relativamente pequeños se puede usar esa aproximación. Ello es equivalente a considerar un número de productividad, ya sea individual o del equipo medido en LOC por hombre-semana o puntos de relato por hombre-semana.

Esta cifra de productividad depende de muchos factores, incluyendo el desempeño individual de cada miembro del equipo y el funcionamiento como equipo, además de tipo de proyecto, plataforma de desarrollo, etc. Debido a esto, no resulta muy confiable usar parámetros de la industria en general y menos aún si es la industria norteamericana (se suele encontrar en estudios).

Lo mejor es utilizar cifras basadas en la información histórica del mismo proyecto en desarrollo (como han sido los sprints anteriores). Si no disponemos de ellas (proyecto no ha iniciado) se puede usar la información histórica de la organización (en otros proyectos similares). Solo en último término hay que recurrir a datos generales de la industria del tipo que muestra el estudio de la figura.

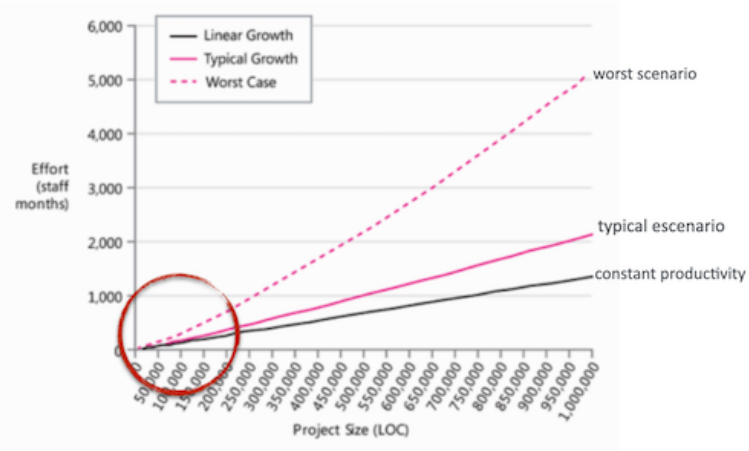
Los datos que se deben recolectar incluyen por supuesto líneas de código, tiempo de desarrollo, número de personas que participan. También conviene recolectar información sobre la calidad: número y tipo de defectos encontrados en relación a las líneas de código, etc.



Esta información debería recolectarse durante el proceso mismo en forma automatizada o si no es posible inmediatamente completado el proyecto. Si no se hace en ese momento, lo más probable es que no va a quedar registro alguno que pueda ser utilizado en el futuro.

Aun cuando sabemos que las cifras de productividad dependen mucho del equipo, muchas veces los equipos se mantienen o cambian solo ligeramente (uno o dos integrantes) por lo que el usar cifras anteriores de la organización es bastante útil.

En cuanto al supuesto de crecimiento lineal del esfuerzo en relación al tamaño, para tamaños relativamente pequeños no se induce un error demasiado grande. La figura muestra como la curva comienza a alejarse de la linealidad para programas de 250.000 LOC o más. El círculo muestra la zona donde normalmente estaremos operando.



Si se quiere mas presición puede usarse una tabla con productividades crecientes del tamaño como la siguiente:

Team Size	Average Stories Delivered per Calendar Month
1	5
2–3	12
4–5	22
6–7	31
8	No data for projects of this size

A continuación se muestra un ejemplo de cómo se ajustan las estimaciones en base a la información obtenida en el mismo proyecto.

- Inicialmente, se inició con las estimaciones de puntos de relato para cada uno de los 60 relatos del sistema. Se usó un *scrum poker* con escala exponencial y el total de puntos fue de 180.
- Nos encontramos a punto de iniciar el segundo *sprint* (3 semanas) y, por lo tanto, ya sabemos lo que ocurrió en el primero. Estas son las

cifras: trabajaron 4 personas en estas primeras 3 semanas, quienes entregaron 27 puntos de relato.

  - Esfuerzo = 3 x 4 = 12 semanas hombre
  - Personas = 4
  - Se produjeron 2.25 puntos por semana hombre
  - Tiempo = 3 semanas
  - Se produjeron 9 puntos por semana

- Con estas 2 cifras de productividad volvemos a nuestros 180 puntos del proyecto y hacemos las estimaciones globales
  - Esfuerzo = 180/2.25 = 80 semanas hombre
  - Duración = 180/9 = 20 semanas

Estimación por Analogía Simple

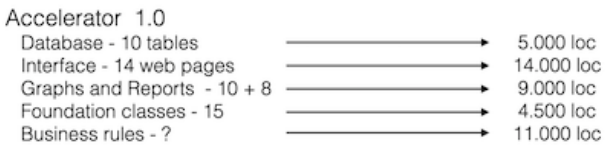
A veces se hace una estimación haciendo una directa analogía con un proyecto similar anterior. El proceso es como sigue:

- Etapa 1: Obtener cifras de tamaño, esfuerzo para proyecto similar
- Etapa 2: Poner las cifras del nuevo proyecto en términos relativos al anterior
- Etapa 3: Construir la estimación de tamaño para el proyecto nuevo
- Etapa 4: Construir estimación de esfuerzo basada en tamaño en comparación al antiguo
- Etapa 5: Asegurarse que son en verdad comparables /envergadura, tecnología, equipo, etc)

Un ejemplo se muestra a continuación.

Se pide desarrollar la versión 1.0 de una nueva aplicación Web llamada Triad. Se ha estimado que el modelo de datos incluye 15 clases de dominio y 14 tablas y comprende una interfaz compuesta por 19 páginas Web. Hay además que generar 14 gráficos y 16 reportes.

Se acaba de desarrollar una aplicación previa llamada AccSellerator, para la cual se requirió un esfuerzo de 30 meses hombre de la que se dispone de la información que muestra la tabla siguiente



Hacemos la analogía entre lo que conocemos y lo que no conocemos extrayendo un factor multiplicativo

Subsystem	Actual Size of AccSellerator 1.0	Estimated Size of Triad 1.0	Multiplication Factor
Database	10 tables	14 tables	1.4
User interface	14 Web pages	19 Web pages	1.4
Graphs and reports	10 graphs + 8 reports	14 graphs + 16 reports	1.7
Foundation classes	15 classes	15 classes	1.0
Business rules	???	???	1.5

Aplicamos los factores a los tamaños conocidos de AccSellerator para obtener la estimación de tamaño de la nueva aplicación

Subsystem	Code Size of AccSellerator 1.0	Multiplication Factor	Estimated Code Size of Triad 1.0
Database	5,000	1.4	7,000
User interface	14,000	1.4	19,600
Graphs and reports	9,000	1.7	15,300
Foundation classes	4,500	1.0	4,500
Business rules	11,000	1.5	16,500
TOTAL	**43,500**	**-**	**62,900**

En definitiva, el tamaño de Triad será de 62.900 LOC, es decir, 1.45 veces el tamaño de AccSellerator para el cual se invirtieron 30 meses hombre. Si asumimos una relación lineal, se esperaría entonces que el desarrollo de Triad ocupe 30 x 1.45 = 44 meses hombre.