

Estimaciones

- ▶ ¿Qué es lo que queremos estimar ?
 - ▶ tiempo necesario para el desarrollo
 - ▶ esfuerzo necesario para el desarrollo
 - ▶ tamaño del problema
- ▶ ¿Cuando queremos hacer la estimación?
 - ▶ antes de comenzar
 - ▶ cada vez que planificamos un sprint



Tiempo

- ▶ Es lo que más interesa saber antes de comenzar porque pueden firmarse compromisos o incorporarse multas
- ▶ Métricas
 - ▶ horas
 - ▶ días
 - ▶ semanas
 - ▶ meses
 - ▶ trimestres



Esfuerzo

- ▶ Esto es importante para tener una primera aproximación del costo de desarrollo
- ▶ Métricas
 - ▶ meses hombre
 - ▶ semanas hombre
- ▶ Si el proyecto va a requerir a dos ingenieros durante 3 meses serían 6 meses hombre

Recordar "El Mítico Hombre mes"

- ▶ El esfuerzo debe ser estimado en forma independiente
- ▶ Error muy común:

Hemos estimado el tiempo de desarrollo en 3 meses

Como es un equipo de 4 personas el esfuerzo es 12 meses hombre

- ▶ Tanto el tiempo como el esfuerzo dependen del tamaño o envergadura del proyecto

Tamaño

- ▶ Es lógico pensar que un producto mas "grande" requerirá más tiempo y mas esfuerzo de desarrollo
- ▶ Métricas
 - ▶ líneas de código fuente (LOC)
 - ▶ número de relatos de usuario
 - ▶ puntos de relatos de usuario
 - ▶ puntos de función

Ejemplo de Modelo basados en Datos de la Industria

Se estimó S en 100 PF

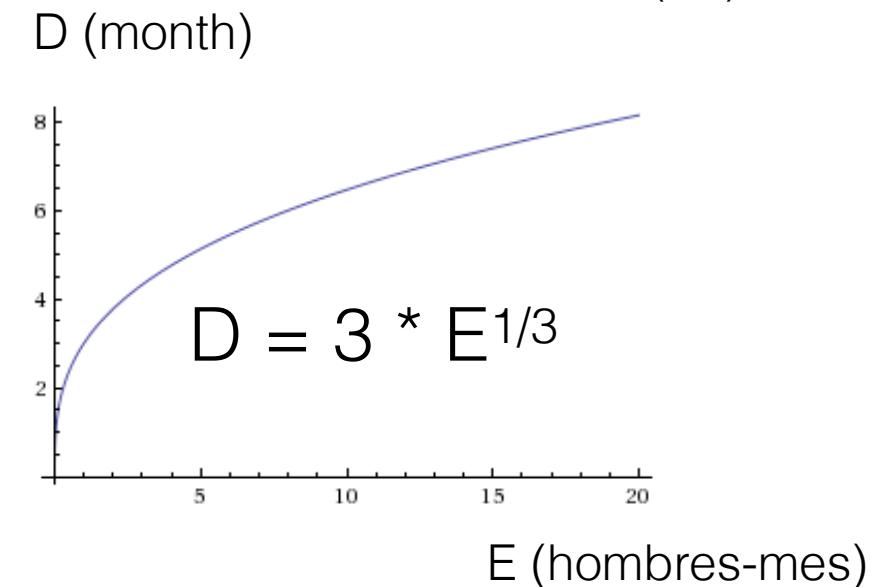
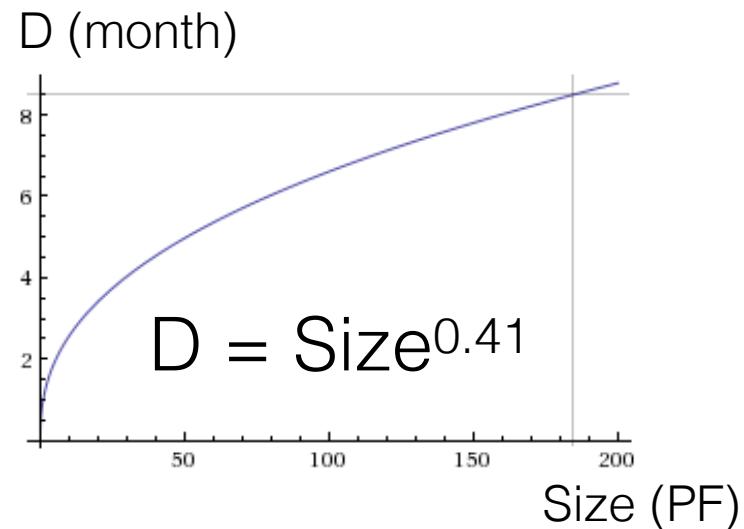
$$D = 100^{0.41} = 6.6 \text{ meses}$$

$$6.6 = 3 * E^{1/3}$$

$$E = (6.6/3)^3$$

$$E = 10.6 \text{ hombres mes}$$

Se requieren 2 personas

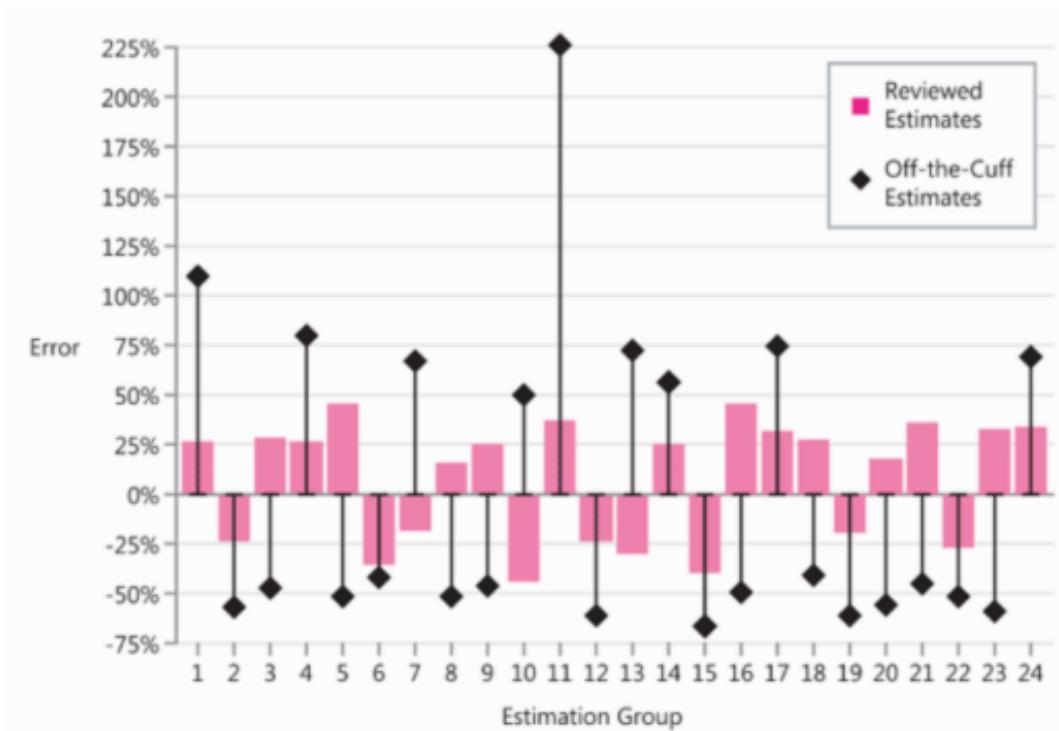


Algunas consideraciones generales

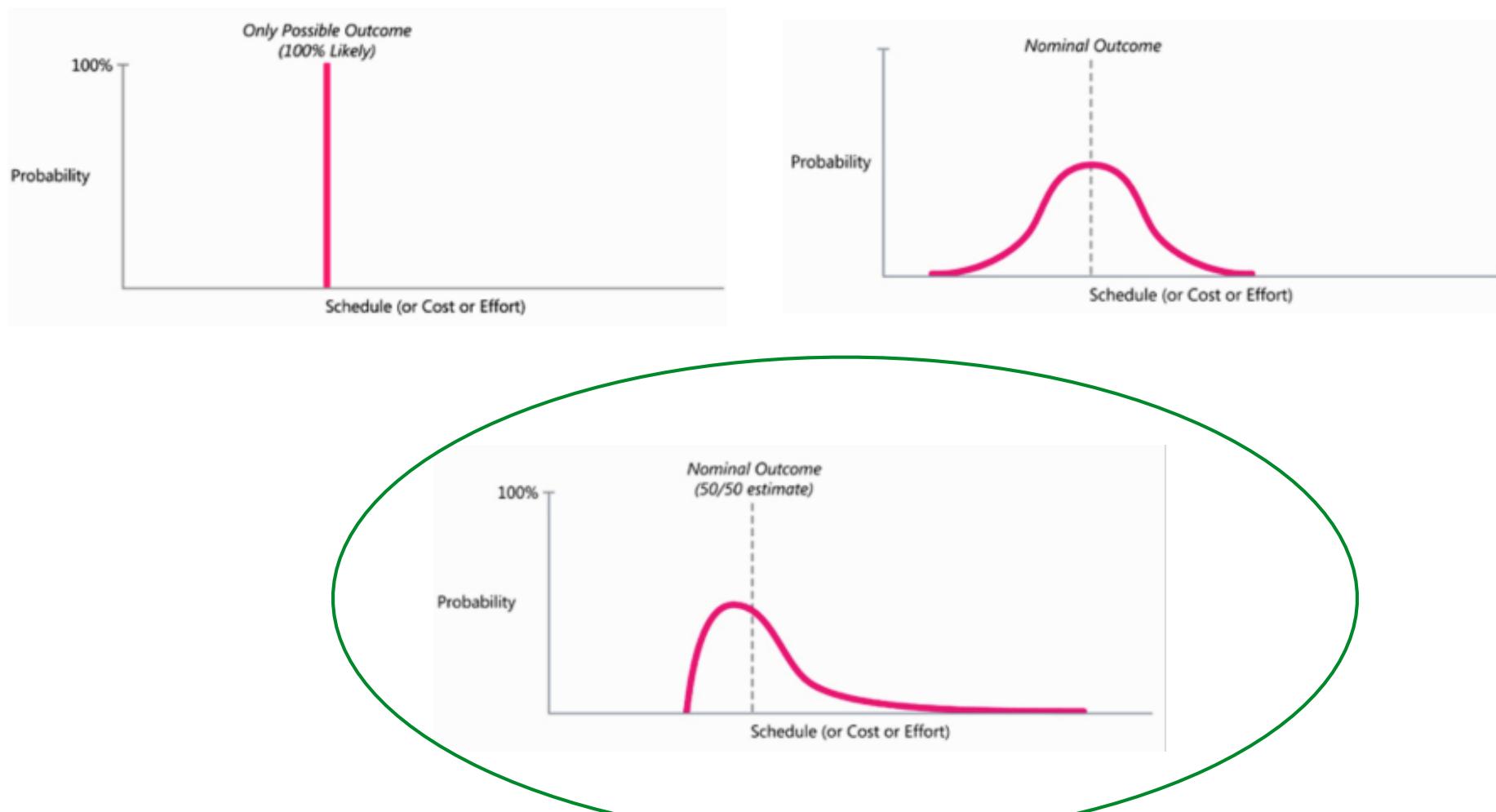
- ▶ Evitar a toda costa entregar una estimación a la rápida sin ningún fundamento
 - ▶ Cuanto crees que tomaría ...
 - ▶ No sé pero supongo que un par de semanas
- ▶ Siempre es una afirmación probabilística por lo que debe usarse escala que de una indicación de ello
 - ▶ Estará listo en 56 días
 - ▶ Estará listo en dos meses
 - ▶ Estará listo antes de fin de año

La respuesta improvisada suele ser pésima

Es mejor esperar aunque sean 20 minutos y hacer un pequeño análisis y NO DAR estimaciones "al tufo"



Estimaciones son afirmaciones probabilísticas



Sobreestimar vs Subestimar

Problemas de sobreestimar el tiempo

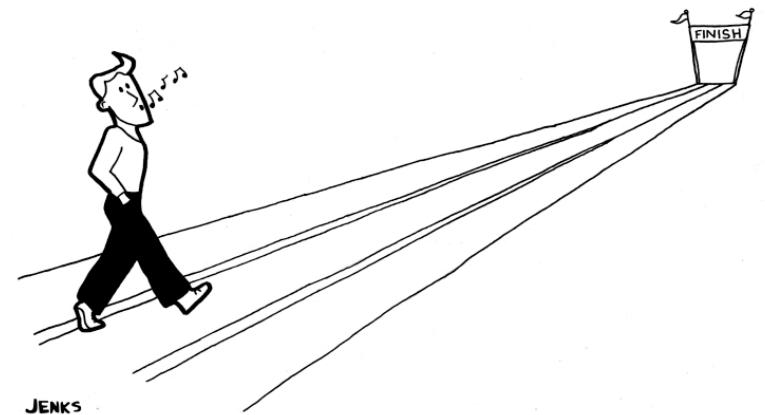
- ▶ ley de Parkinson - el trabajo se expandirá hasta usar todo el tiempo disponible
- ▶ ley de Goldratt (síndrome del estudiante) - si hay demasiado tiempo se malgasta al comienzo del proyecto

● Time given
for task

● Time task
will take

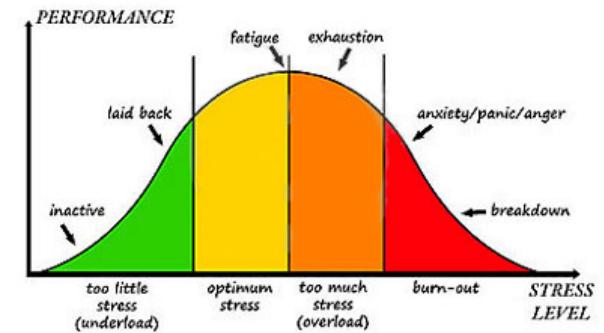


Work expands to fit the time given



- ▶ Problemas de Subestimar el tiempo
 - ▶ errores de planeación, errores de coordinación
 - ▶ reducción de tiempos en análisis y diseño puede generar mucho mas atraso
 - ▶ dinámica de proyecto atrasado: reuniones para reagendar, para recuperar tiempo, etc
 - ▶ problemas con el cliente
 - ▶ arreglos parches (no hay tiempo)

STRESS CURVE



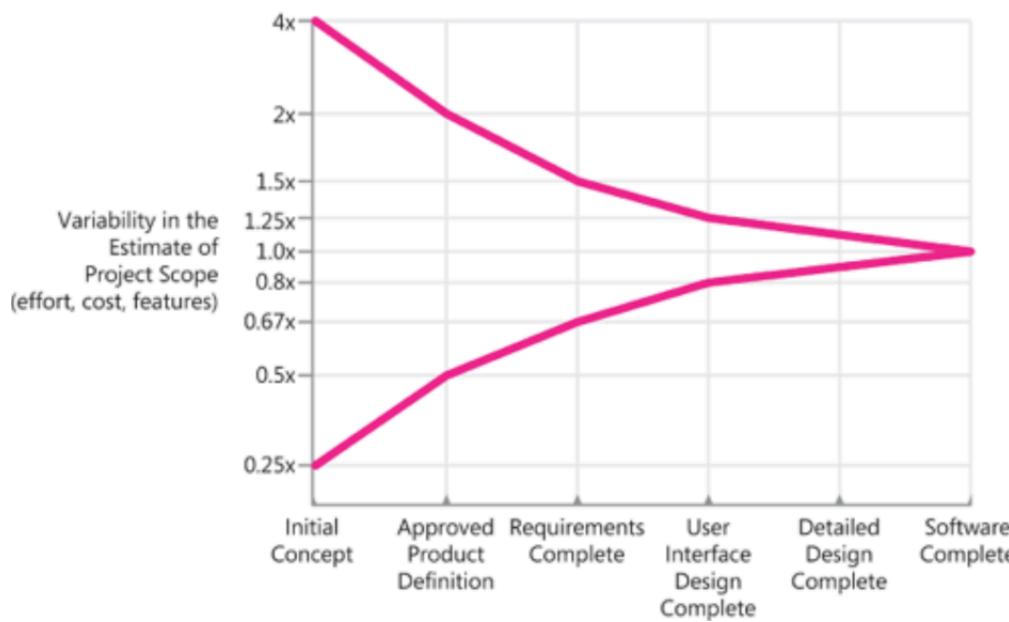
La penalización por sobreestimar es mucho menos que la penalización por subestimar



Cuando Estimar

▶ El cono de incertidumbre

Mientras mas temprano en el proyecto mayor es el error que podemos cometer en la estimación



Reducción de la Incertidumbre

- ▶ Desarrollo iterativo e incremental
- ▶ Después de cada iteración debemos ajustar estimaciones
- ▶ Descomponer en unidades mas pequeñas (ley de los grandes números)
 - ▶ tiempo para un sitio web (10 semanas)
 - ▶ tiempo para un relato de usuario (3 días)
 - ▶ tiempo para una parte del relato (6 horas)

Las 4 fuentes de error en estimaciones

- ▶ Información imprecisa sobre el proyecto
- ▶ Información imprecisa sobre las capacidades del equipo de desarrollo
- ▶ Proyecto demasiado caótico (blanco móvil)
- ▶ Imprecisiones del proceso de estimación mismo

Información del Proyecto

- ▶ Al comienzo del proyecto no se tiene claridad respecto a lo se debe construir
- ▶ Cada pieza de software tiene muchas formas de ser implementada (performance, usabilidad, etc)
- ▶ Ello produce una enorme gama de posibilidades

Capacidades del Equipo

- ▶ Variaciones de productividad individual 1-5
- ▶ Variaciones de productividad como equipo (recordar lo que estudiamos sobre equipo de alto desempeño)
- ▶ Es más fácil si los equipos se mantienen de un proyecto a otro
- ▶ Es más fácil si el tipo de proyecto y la plataforma es la misma
- ▶ Se sabe más en cada iteración

Precisión

- ▶ Si indicas que el proyecto tomará 90.5 días los clientes asumen que estimación podría variar tal vez en medio día
- ▶ Mejor indicar 3 meses o 1 trimestre

Precisión - número de dígitos significativos

Exactitud - cuan cerca del valor real

1. Accurate
Precise
2. Not Accurate
Precise
3. Accurate
Not Precise



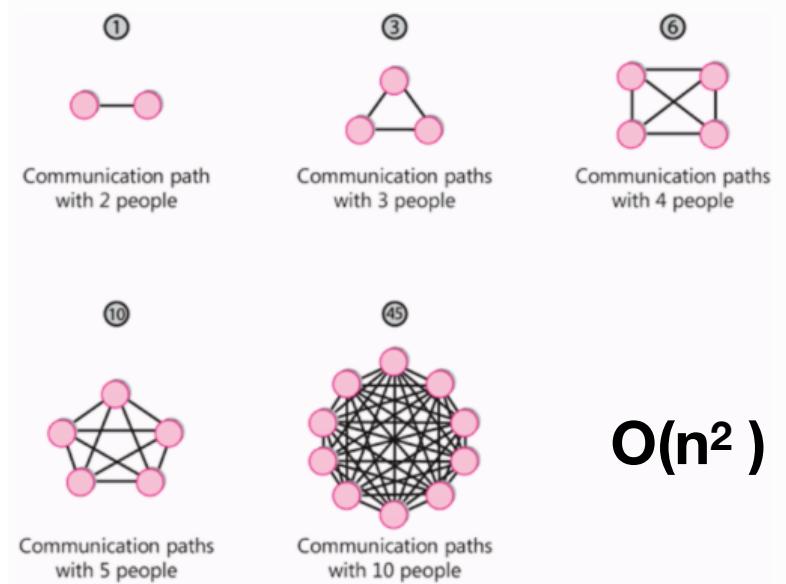
Se parte con estimación de tamaño

- ▶ El tamaño es el factor mas relevante en determinar el tiempo y el esfuerzo de un proyecto
- ▶ Considerar que aquí rige la **deseconomía** de escala
 - ▶ mientras mas grande menor productividad
 - ▶ un producto de 100.000 LOC tomará más de 10 veces lo que toma hacer uno de 10.000 LOC

El tamaño sí importa

- ▶ Ya hemos dicho que es el tamaño lo que va a determinar el tiempo y el esfuerzo de desarrollo
- ▶ Rigen las deseconomías de escala
 - ▶ a mayor tamaño menos eficiencia (productividad)
 - ▶ algo de 100.000 loc tomará más del doble de algo de 50.000 loc

Origen de la deseconomía de escala:
necesidad de comunicación y sincronización



$O(n^2)$

¿ Y que demonios es una loc ?

- ▶ loc - line of code
- ▶ número de líneas de código fuente
- ▶ es la métrica mas utilizada para tamaño
 - ▶ simple
 - ▶ gran ventaja es que una vez construido se puede medir en forma exacta
- ▶ puede incluirse algunos ajustes como
 - ▶ no contar las líneas de comentario
 - ▶ no contar las líneas en blanco

Ventajas y desventajas de usar locs

- ▶ Ventaja: una vez construido se puede medir en forma exacta
- ▶ Desventajas:
 - ▶ no se conoce antes de construir el producto
 - ▶ depende de
 - ▶ lenguaje de programación usado
 - ▶ estilo de escritura del código
 - ▶ calidad del código (mejor tiende a ser más corto)
 - ▶ comentarios y líneas en blanco

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. HELLO-WORLD.  
* simple hello world program  
PROCEDURE DIVISION.  
    DISPLAY 'Hello world!'.  
    STOP RUN.
```

COBOL

```
print('Hello, world!')
```

Python

```
#include <stdio.h>  
int main()  
{  
    // printf() displays the string inside quotation  
    printf("Hello, World!");  
    return 0;  
}
```

C

```
public class HelloWorld {  
  
    public static void main(String[] args) {  
        // Prints "Hello, World" to the terminal window.  
        System.out.println("Hello, World");  
    }  
}
```

JAVA

```
puts "Hello World!"
```

Ruby

```
console.log('Hello world!');
```

JavaScript

Hello world

Otras métricas que no tienen esas desventajas

- ▶ pueden usarse en forma temprana y no dependen del lenguaje ni del estilo de programación
 - ▶ puntos de función (muy antigua pero aún en uso)
 - ▶ relatos de usuario (user stories)
 - ▶ puntos de relato (user stories points)

Puntos de Función

Independiente de lenguaje o estilo de programación

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

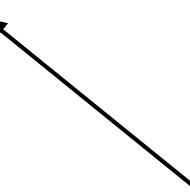
Puntos de Función no Ajustados: PFNA

Factor de Complejidad (14 factores) : 0.65 - 1.35

Puntos de Función Ajustados = PFNA * FC

Factor de Complejidad

- 01 comunicaciones
 - 02 funciones distribuidas
 - 03 objetivos de desempeño
 - 04 configuración sobrecargada
 - 05 tasa de transacciones
 - 06 entrada de datos on line
 - 07 eficiencia para usuario
 - 08 actualización en línea
 - 09 proceso complejo
 - 10 reuso
 - 11 facilidad de instalación
 - 12 facilidad de operación
 - 14 varios sitios
 - 14 acilidad de mantención
- Mín 14 * 0 = 0
Máx 14 * 5 = 70
- $$FC = 0.65 + N/100$$



User stories y storypoints

- ▶ Número de relatos de usuario es una buena indicación del tamaño (complejidad) del software
- ▶ Si los relatos varían mucho en complejidad es mejor incluir ponderadores lo que da origen a los storypoints
- ▶ Se asignan ponderadores en escalas no lineales porque esfuerzo aumenta en forma no lineal
 - ▶ potencias de 2 : 1, 2, 4, 8, 16
 - ▶ secuencia de fibonacci: 1, 2, 3, 5, 8

Asignación simple de Puntos de Relato

- ▶ Clasificar los relatos en un número pequeño de categorías (por ejemplo 4)
 - ▶ simple
 - ▶ intermedio
 - ▶ complejo
 - ▶ muy complejo
- ▶ Asignar por escala
 - 1 - simple
 - 2 - intermedio
 - 3 - complejo
 - 5 - muy complejo

Estimación Colaborativa

- ▶ Planning poker o Scrum poker para asignar puntos de relato
- ▶ Variante del método delphi
- ▶ Para cada relato varios participantes "apuestan" después de analizar el trabajo
- ▶ Las apuestas son con cartas hacia abajo
- ▶ Se levantan las cartas simultáneamente
- ▶ Si hay mucha dispersión se conversa y justifica y se repite hasta llegar a un consenso



- ▶ Escala de cartas puede variar (Fibonacci, Fibonacci + 1/2, etc)
- ▶ usar un timer para limitar la discusión

Del tamaño al esfuerzo

- ▶ Una vez hecha una estimación de tamaño por ejemplo en puntos de relato es necesario pasar a esfuerzo
- ▶ Sabemos que esfuerzo no crece linealmente con el tamaño pero en números pequeños puede aproximarse de este modo
- ▶ Se usa la cifra de productividad (individual o del equipo)
 - ▶ loc/hombre-mes
 - ▶ puntos relato/hombre-semana
- ▶ Productividad puede variar bastante y va a depender de
 - ▶ productividad individual, funcionamiento como equipo, tipo de proyecto
- ▶ Pueden usarse cifras de productividad histórica

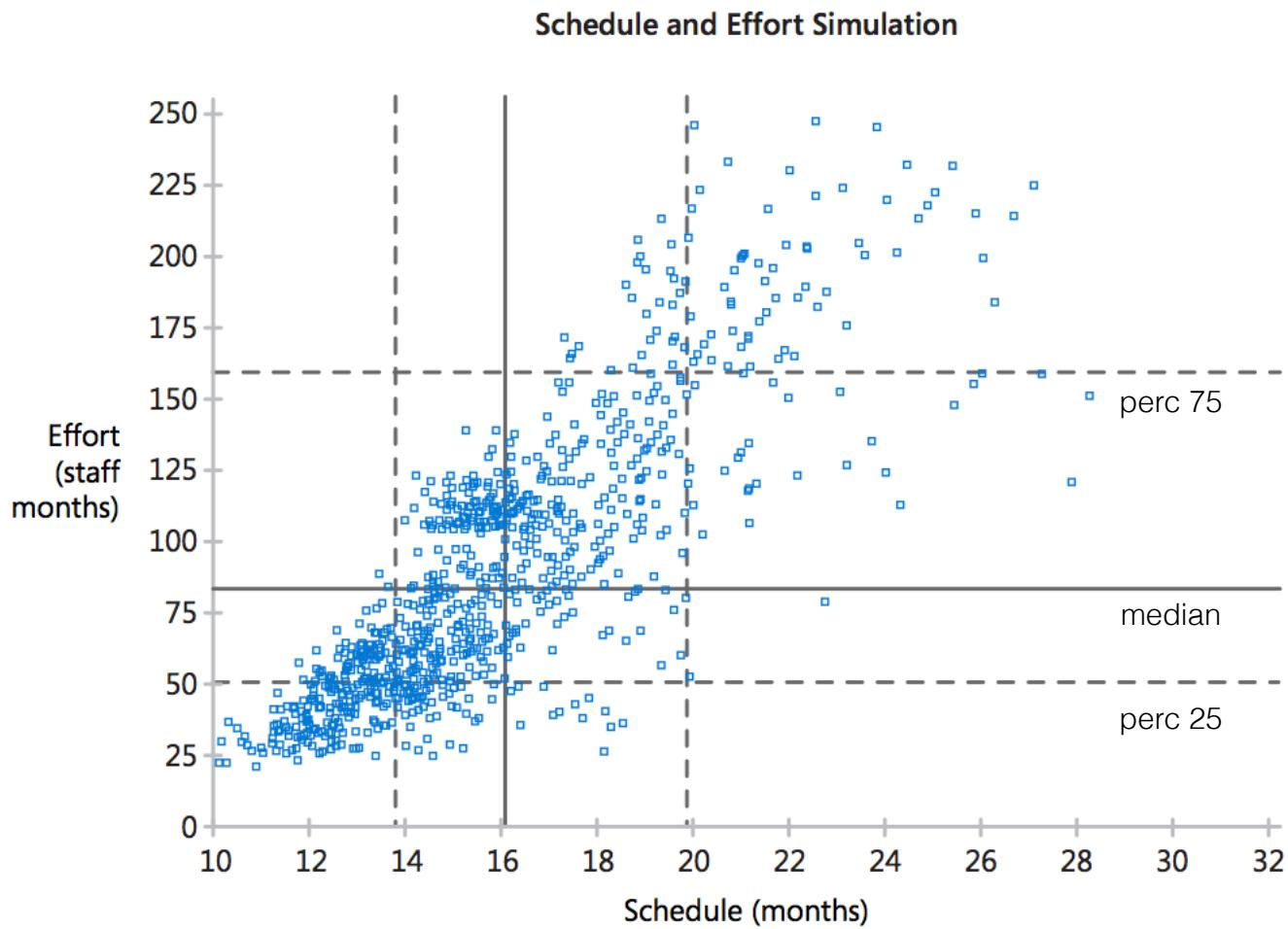
Productividades Típicas (USA)

Project Size (in Lines of Code)	Lines of Code per Staff Year (Cocomo II Nominal in Parentheses)	LOC/Staff Month Low-High (Nominal)		
		10,000- LOC Project	100,000- LOC Project	250,000- LOC Project
10K	2,000–25,000 (3,200)			
100K	1,000–20,000 (2,600)			
1M	700–10,000 (2,000)			
10M	300–5,000 (1,600)			
Kind of Software		10,000- LOC Project	100,000- LOC Project	250,000- LOC Project
Avionics		100–1,000 (200)	20–300 (50)	20–200 (40)
Business Systems		800–18,000 (3,000)	200–7,000 (600)	100–5,000 (500)
Command and Control		200–3,000 (500)	50–600 (100)	40–500 (80)
Embedded Systems		100–2,000 (300)	30–500 (70)	20–400 (60)
Internet Systems (public)		600–10,000 (1,500)	100–2,000 (300)	100–1,500 (200)

La crucial información histórica

- ▶ Permite hacer la calibración de algo que se puede contar en el valor de lo que queremos estimar
 - ▶ Lo mejor es usar información histórica del mismo proyecto
 - ▶ Lo segundo mejor es información histórica de la organización
 - ▶ Finalmente usar información de la industria (ojalá del país)
- ▶ Es muy deseable preocuparse de recolectar toda esta información en forma automatizada
 - ▶ ojalá durante el proyecto
 - ▶ en el peor caso inmediatamente después de terminado

Uso de Datos de la Industria solo como último recurso



Qué datos recolectar

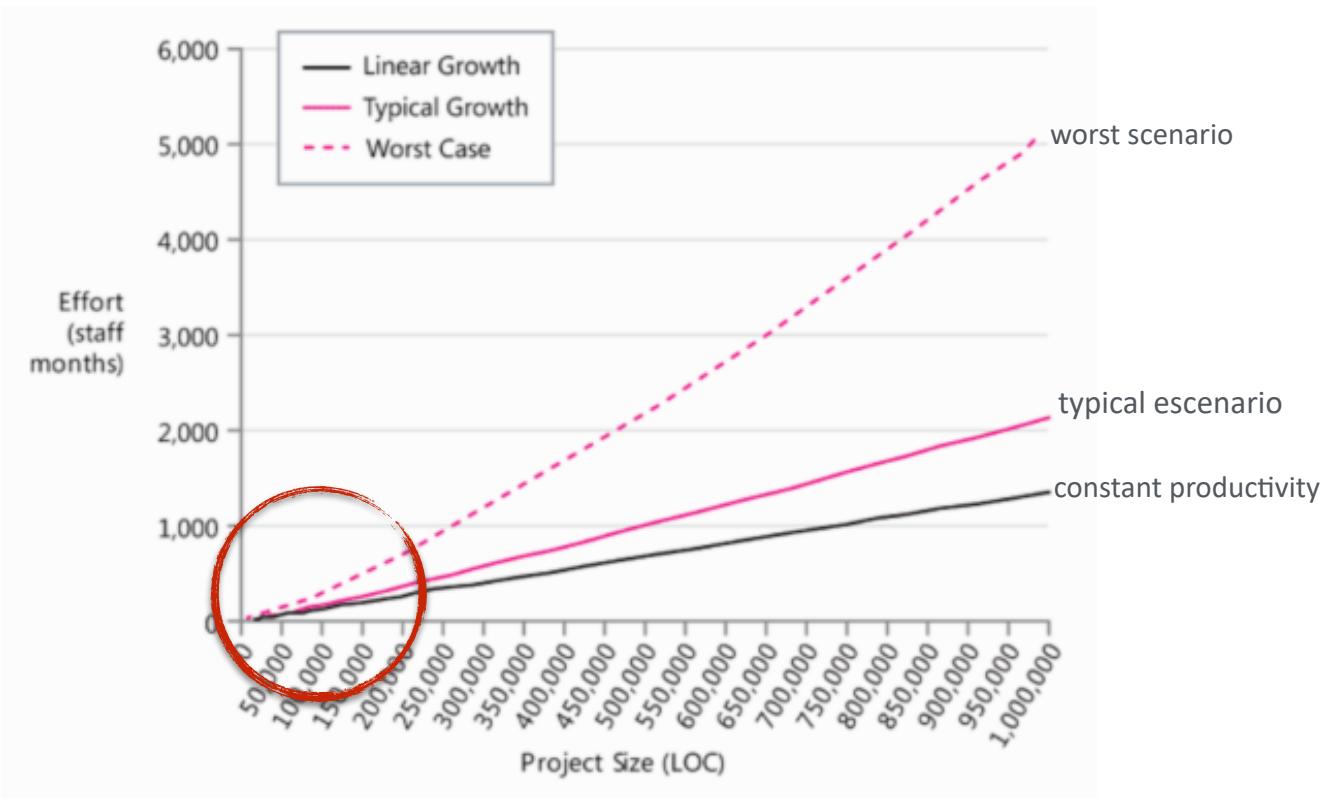
- ▶ Comenzar con lo más simple
 - ▶ líneas de código una vez completado el proyecto
 - ▶ esfuerzo en personas y meses
 - ▶ tiempo en meses
 - ▶ defectos (calibrados por severidad)
- ▶ Debe recolectarse durante ejecución o inmediatamente terminado (después es difícil)

La buena noticia ...

- ▶ Muchas veces un grupo desarrolla proyectos de tamaño similar (hasta 2 o 3 veces)
- ▶ Equipo no cambia demasiado
- ▶ Si se mantienen ambas cosas, lo mas probable es que la productividad se mantenga y puede ser usada para la estimación

La mala noticia

La productividad no es constante (depende del tamaño)



Sin embargo para tamaños pequeños funciona

Uso de calibración lineal

- ▶ Lo más simple, muchas veces suficiente
 - ▶ equipo (conocido) puede entregar X relatos implementadas por mes calendario en este tipo de proyecto
 - ▶ Se puede manejar no linealidad con tablas construidas en base a información histórica

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

Ejemplo de Calibración durante el Desarrollo

Story	Points
Story 1	2
Story 2	1
Story 3	4
Story 4	8
...	
Story 60	2
TOTAL	180

Data for Iteration 1

27 story points delivered

12 staff weeks expended

3 calendar weeks expended

Preliminary Calibration

Effort = 27 story points ÷ 12 staff weeks = 2.25 story points/staff week

Schedule = 27 story points ÷ 3 calendar weeks = 9 story points/calender week

Data for Iteration 1

Assumptions (from Preliminary Calibration)

Effort = 2.25 story points/staff week

Schedule = 9 story points/calender week

Project size = 180 story points

Preliminary Whole-Project Estimate

Effort = 180 story points ÷ 2.25 story points/staff week = 80 staff weeks

Schedule = 180 story points ÷ 9 story points/calender week = 20 calendar weeks

Estimación por Analogía

- ▶ 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)

Ejemplo

Accelerator 1.0

Database - 10 tables	→	5.000 loc
Interface - 14 web pages	→	14.000 loc
Graphs and Reports - 10 + 8	→	9.000 loc
Foundation classes - 15	→	4.500 loc
Business rules - ?	→	11.000 loc

Nueva Aplicación (Web) a Desarrollar: Triad 1.0

Database- 14 tablas
Interface - 19 páginas Web
Graphs and Reports - 14 + 16
Foundation classes - 15

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

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

Term	Value
Size of Triad 1.0	62,900 LOC
Size of AccSellerator 1.0	÷ 43,500 LOC
Size ratio	= 1.45
Effort for AccSellerator 1.0	× 30 staff months
Estimated effort for Triad 1.0	= 44 staff months