Ingeniería de software^2

Yadran Eterovic S. (yadran@uc.cl)

Estimaciones en proyectos de ingeniería de software

Frente a la necesidad de llevar adelante un proyecto de desarrollo de software

… nos hacemos dos preguntas principales:

¿Cuánto me va a costar?

¿Cuánto se va a demorar?

De la clase “Planificación y estimaciones en proyectos de software”

1. Una estimación es un juicio o cálculo aproximado del valor, número, cantidad o extensión de algo —p.ej., del esfuerzo y tiempo necesarios para realizar el proyecto

2. El esfuerzo y el tiempo necesarios para realizar el proyecto dependen principalmente de la magnitud del software —invirtamos una cantidad apropiada de trabajo para evaluar la magnitud del software que se va a desarrollar (en sí, una estimación)

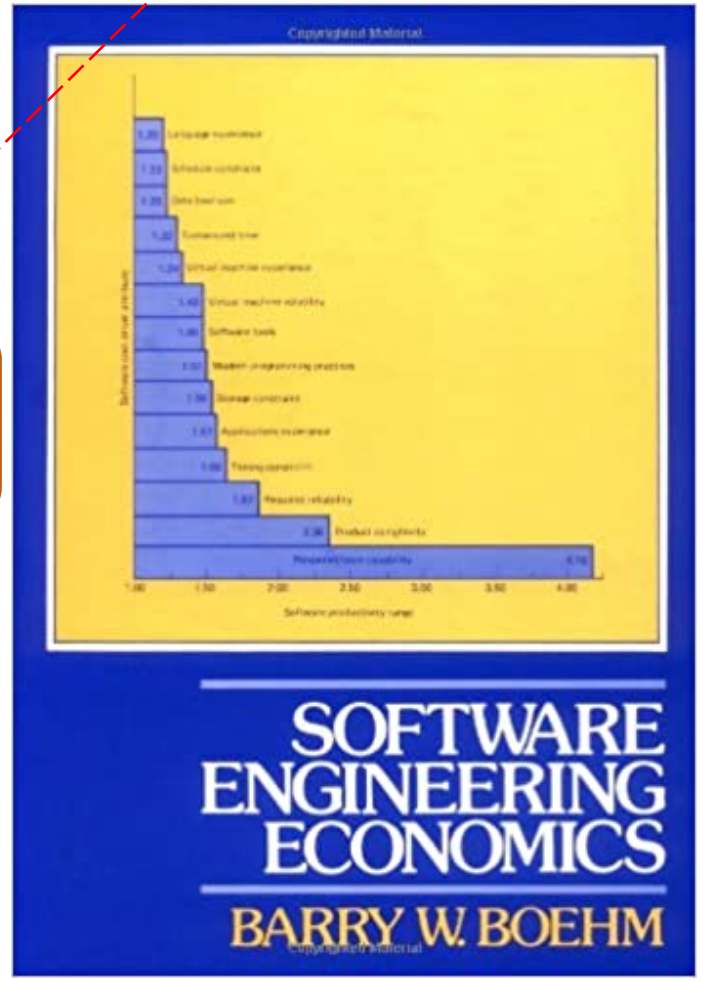
¿Cómo estimamos la magnitud del software?

Primero, ¿cuál es la unidad de medida? (¿cuál es el “metro cuadrado” de software?)

Segundo, ¿cómo calculamos—aproximadamente (es una *estimación*)—el número de estas unidades para una aplicación que aún no ha sido desarrollada?

En 1981, B. Boehm propuso usar el número de miles de líneas de código fuente (KSLOC) como medida de la magnitud del software, para su modelo de estimación COCOMO

en el lenguaje de programación de alto nivel original



Con respecto al número de líneas de código …

¿Cómo estimar el número de líneas de código?

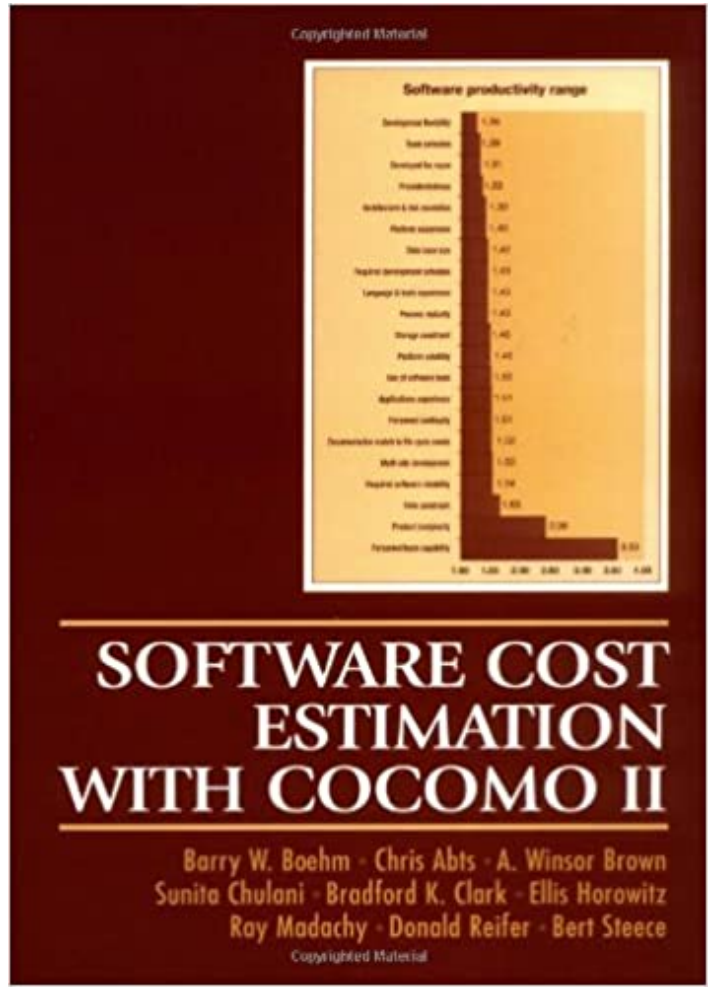
- datos históricos a partir de información disponible temprano en un proyecto
- opinión de expertos

Dificultades:

- diferencias debido a los lenguajes de programación
- diferencias en estilos de escritura de código
- diferencias en calidad de código

En COCOMO II se usa la sentencia fuente lógica (oficialmente definida por el SEI)

En 2001, B. Boehm et al. propusieron usar ya sea el número de KSLOC o bien el número de puntos de función no ajustados (UFP) como medida de la magnitud del software, para la nueva versión del modelo, COCOMO II



Análisis de puntos de función:
un método para medir la magnitud de un sistema

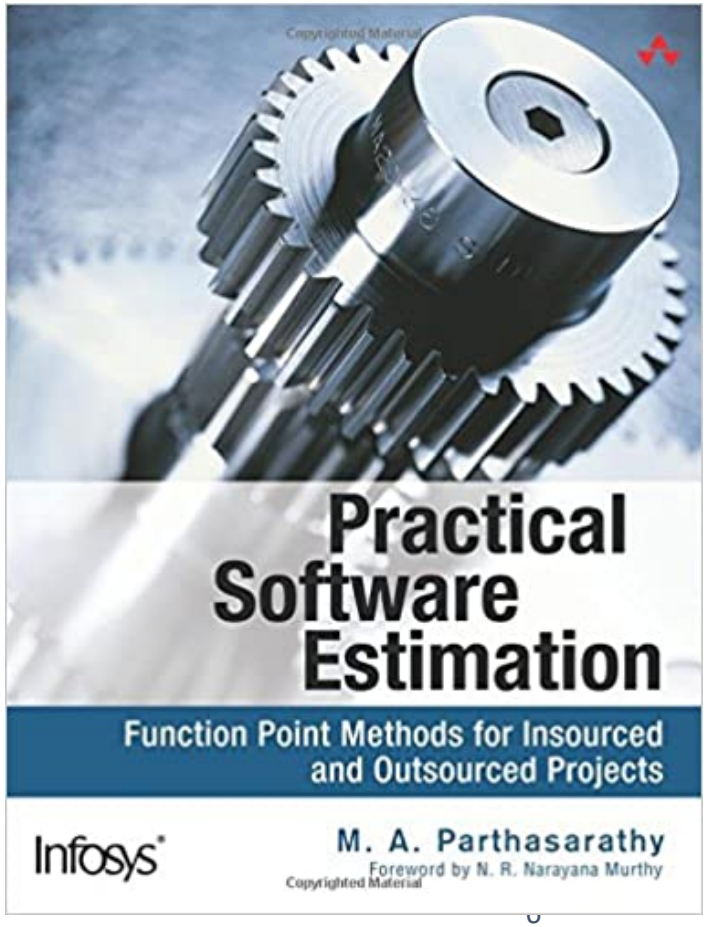
Un punto de función es una unidad de medida sintética del tamaño o magnitud de una aplicación de software

Los puntos de función de una aplicación debieran ser una medida de la funcionalidad que el usuario solicita (y la aplicación ofrece):

- es decir, puntaje por funcionalidad según el usuario final, personal de marketing, analista de negocios, comprador, etc.

Los puntos de función son más fáciles de calcular —a partir de una especificación de requisitos funcionales— que las líneas de código

… y ofrecen una base para calcular la magnitud en líneas de código



A medida que avanza el desarrollo, mejora la información disponible para contar puntos de función

Propuesta del proyecto

Documento de requisitos

Diagramas del sistema y de entidad-relación

Especificaciones funcionales y del sistema

Modelos lógicos de datos y de procesos

Especificaciones de programas y módulos

Manual del usuario

Material de capacitación

Contar puntos de función toma tres pasos principales:

1) Identificar los archivos lógicos internos (ILF) y los archivos de interfaz externos (EIF), y sus niveles de complejidad

2) Identificar los inputs (EI), los outputs (EO) y las consultas (EQ), y sus niveles de complejidad

3) Calcular los puntos de función no ajustados, sumando los resultados de 1) y 2) (próx. diapo.)

Contar puntos de función toma tres pasos principales:

1) Identificar los archivos lógicos internos (ILF) y los archivos de interfaz externos (EIF), y sus niveles de complejidad

2) Identificar los inputs (EI), los outputs (EO) y las consultas (EQ), y sus niveles de complejidad

3) Calcular los puntos de función no ajustados, sumando los resultados de 1) y 2) (próx. diapo.)

ILF: Cada grupo principal de datos o de información de control relacionada lógicamente e identificable por el usuario, que es controlado (generado, usado o mantenido) completamen-te por la aplicación

EIF: Cada archivo (grupo lógico de datos o de infor-mación de control) con el cual la aplicación interac-túa, pero que es controlado (mantenido) por otra aplicación

EI: Cada proceso elemental de la aplicación sobre datos o información de control que entra a la aplicación

EO: Cada proceso elemental de la aplicación que genera datos o información de control que sale de la aplicación

EQ: Cada proceso elemental de la aplicación compuesto por una combinación de input-output en la que el input (informa-ción de control que define la consulta) produce un output simple inmediato (datos obtenidos de uno o más ILF's o EIF's)

Contar puntos de función toma tres pasos principales:

1) Identificar los archivos lógicos internos (ILF) y los archivos de interfaz externos (EIF), y sus niveles de complejidad

2) Identificar los inputs (EI), los outputs (EO) y las consultas (EQ), y sus niveles de complejidad

3) Calcular los puntos de función no ajustados, sumando los resultados de 1) y 2) (próx. diapo.)

ILF: Cada grupo principal de datos o de información de control relacionada lógicamente e identificable por el usuario, que es controlado (generado, usado o mantenido) completamen-te por la aplicación

EIF: Cada archivo (grupo lógico de datos o de infor-mación de control) con el cual la aplicación interac-túa, pero que es controlado (mantenido) por otra aplicación

EI: Cada proceso elemental de la aplicación sobre datos o información de control que entra a la aplicación

EO: Cada proceso elemental de la aplicación que genera datos o información de control que sale de la aplicación

EQ: Cada proceso elemental de la aplicación compuesto por una combinación de input-output en la que el input (informa-ción de control que define la consulta) produce un output simple inmediato (datos obtenidos de uno o más ILF's o EIF's)

3) Calcular los puntos de función no ajustados, sumando los resultados de 1) y 2); es decir, completar la siguiente tabla:

Característica de la aplicación	Complejidad baja	Complejidad promedio	Complejidad alta
EI's	..... × 3 = ___	..... × 4 = ___	..... × 6 = ___
EO's	..... × 4 = ___	..... × 5 = ___	..... × 7 = ___
EQ's	..... × 3 = ___	..... × 4 = ___	..... × 6 = ___
ILF's	..... × 7 = ___	..... × 10 = ___	..... × 15 = ___
EIF's	..... × 5 = ___	..... × 7 = ___	..... × 10 = ___

... y sumar los 15 valores en los ___ para obtener los puntos de función

¿Cómo pasar de una estimación de magnitud a una de esfuerzo y tiempo?

El modelo **COCOMO II** de estimaciones de esfuerzo (PM) y tiempo (TDEV):

$$PM = 2.94 \times Size^{E} \times (EM_1 \times EM_2 \times \cdots \times EM_n)$$

$$TDEV = 3.67 \times (PM)^{F}$$

El modelo **COCOMO II** de estimaciones de esfuerzo (PM) y tiempo (TDEV):

$0.91 + 0.01 \times (SF_1 + SF_2 + \cdots + SF_5)$

factores de escala
próx. diap.

Esfuerzo necesario para desarrollar la aplicación, en *personas-meses*

Magnitud de la aplicación a ser desarrollada

Multiplicadores del esfuerzo:
$n = 7$ (diseño preliminar)
$n = 17$ (post arquitectura)

$$PM = 2.94 \times Size^{E} \times (EM_1 \times EM_2 \times \cdots \times EM_n)$$

$$TDEV = 3.67 \times (PM)^{F}$$

$0.28 + 0.2 \times (E - 0.91)$

Tiempo que tomará el desarrollo, en *meses*

Las constantes **2.94** y **3.67**, especialmente, pero también 0.91 y 0.28, se recomienda que sean recal-culadas de acuerdo con el entorno de desarrollo local

En la clase “Planificación y estimaciones en proyectos de software” decimos que el esfuerzo (*PM*) depende no sólo de la magnitud de la aplicación a ser desarrollada

Los 5 *factores de escala* (SF_i)

… y los 7 (o 17) *multiplicadores del esfuerzo* (EM_i) toman en cuenta factores adicionales

En la clase “Planificación y estimaciones en proyectos de software” decimos que el esfuerzo (*PM*) depende no sólo de la magnitud de la aplicación a ser desarrollada

Los 5 *factores de escala* (SF_i)

… y los 7 (o 17) *multiplicadores del esfuerzo* (EM_i) toman en cuenta factores adicionales

similitud del producto con respecto a proyectos desarrollados previamente

flexibildad de desarrollo

resolución de riesgos y de decisiones arquitectónicas

cohesión del equipo de *stakeholders* (usuarios, clientes, desarrolladores, mantenedores, …)

madurez del proceso

Los valores de estos ***5 factores de escala*** varían desde 0, para los casos más favorables, hasta entre 5 y 8 para los casos más desfavorables

En la clase “Planificación y estimaciones en proyectos de software” decimos que el esfuerzo (*PM*) depende no sólo de la magnitud de la aplicación a ser desarrollada

Los 5 *factores de escala* (SF_i)

… y los 7 (o 17) *multiplicadores del esfuerzo* (EM_i) toman en cuenta factores adicionales

similitud del producto con respecto a proyectos desarrollados previamente

flexibildad de desarrollo

resolución de riesgos y de decisiones arquitectónicas

cohesión del equipo de *stakeholders* (usuarios, clientes, desarrolladores, mantenedores, …)

madurez del proceso

Los valores de estos ***5 factores de escala*** varían desde 0, para los casos más favorables, hasta entre 5 y 8 para los casos más desfavorables

confiabilidad requerida y complejidad del producto

desarrollo para reusabilidad

exigencias de tiempo y memoria, y volatilidad de la plataforma

capacidad de analistas, programadores y personal

experiencia del personal en la aplicación, lenguaje y otras herramientas, y plataforma

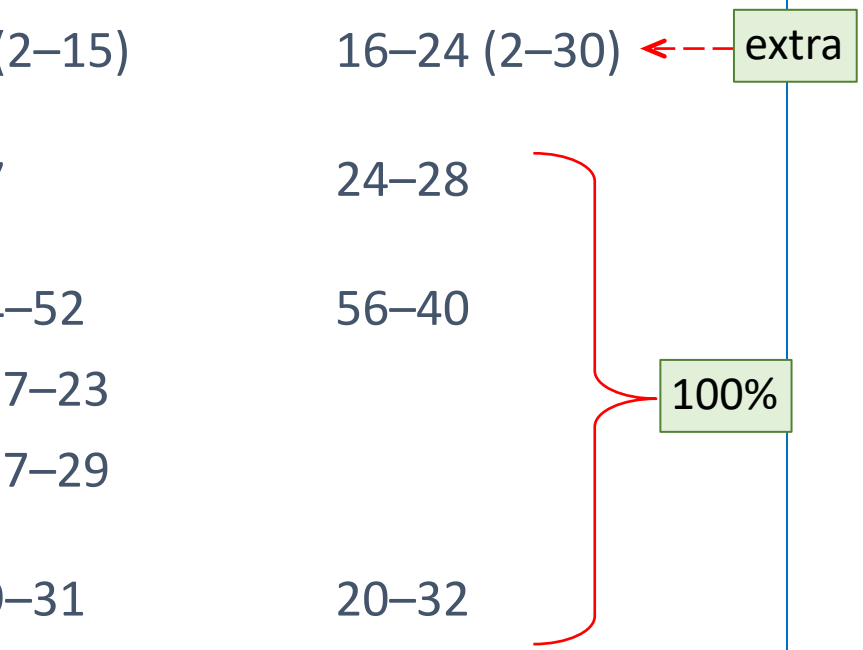
uso de herramientas y facilidades para desarrollo multisitios

exigencias con respecto a tiempo de desarrollo

Los valores de estos ***7 multiplicadores del esfuerzo*** varían desde 0.5, para los casos más favorables, hasta entre 1.5 y 2.5+ para los casos más desfavorables

¿Cuánto esfuerzo y cuánto tiempo demanda cada fase del desarrollo de una aplicación?

Fase	Esfuerzo %	Tiempo %
Planes y requisitos	7 (2–15)	16–24 (2–30)
Diseño del producto	17	24–28
Programación	64–52	56–40
Diseño detallado	27–23	
Código y tests unitarios	37–29	
Integración y pruebas	19–31	20–32



Para proyectos que siguen procesos de desarrollo ágiles, hay otra posibilidad

Para cada relato de usuario,

… queremos estimar **cuántos días** nos vamos a demorar en desarrollarlo

Para ello, jugamos *planning poker*, como se explica a continuación

Todos los que van a hacer la estimación—principalmente, los desarrolladores—se sientan alrededor de una mesa

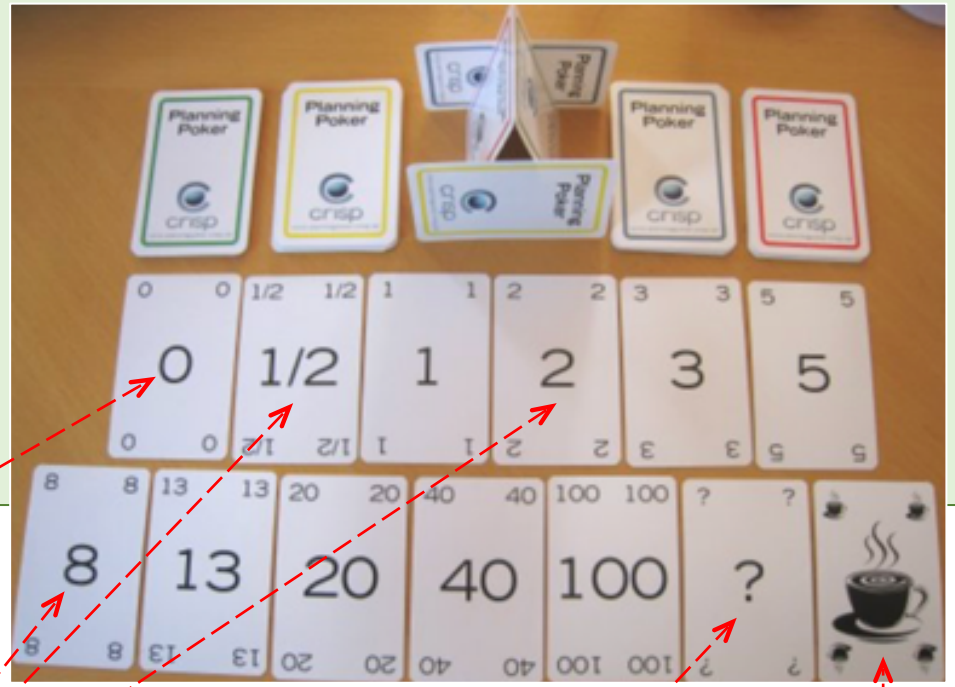
Al centro de la mesa se coloca **un** relato de usuario

Cada persona recibe una baraja de 13 naipes:

… hace individualmente su estimación para el relato de usuario

… y coloca el naipe correspondiente cara abajo sobre la mesa

Finalmente, todos dan vuelta sus naipes al mismo tiempo



Ya está hecho

Cada naipe representa una estimación en *días-desarrollador*

"No tengo suficiente informa-ción para hacer la estimación"

"Necesito un descanso"

Con respecto a nuestras estimaciones

1) Debe ser para todo lo que implica desarrollar el relato, incluyendo:

diseño, programación, pruebas, entrega

… no sólo para una parte

2) No debemos discutirla con nadie, para no dejarnos influenciar—por eso, colocamos el naipe cara abajo

Las estimaciones individuales casi nunca coinciden unas con otras

Si al dar vuelta los naipes varias estimaciones están en un mismo rango,

p.ej., 1/2 a 2 días, 3 a 8 días, 8 a 20 días

… es más o menos seguro deducir que una estimación correcta también va a estar en ese mismo rango

Por otra parte, no ignoremos una estimación que es muy diferente de las otras:

preguntemos al que la propuso qué está pensando, qué suposiciones está haciendo

Rangos amplios pueden reflejar malentendidos

P.ej., parte del equipo de desarrollo …

- entendió mal el relato de usuario
- no está seguro de algo que para la otra parte está OK

Hay que mirar las suposiciones que estamos haciendo

… y decidir si necesitamos ir a hablar con el cliente

Incluso si las estimaciones coinciden, vale la pena preguntar por las suposiciones, para asegurarnos de que no estemos todos haciendo la misma suposición equivocada

Cuando se trata de requisitos, las suposiciones hay que aclararlas

El objetivo durante las estimaciones es eliminar tantas suposiciones como sea posible:

> no hagamos suposiciones sobre las suposiciones

… clarificándolas con el cliente:

> incluso si todo el equipo comparte una suposición, hay que considerarla equivocada hasta que sea clarificada por el cliente

Las que persistan, se convierten en riesgos:

> aunque, por lo menos, ahora sabemos qué es lo que no sabemos

Una estimación muy grande no es muy confiable

Si todos estamos de acuerdo en que este relato de usuario tomará 40 días desarrollarlo

esto es, dos meses de trabajo de un desarrollador

… quiere decir que es un relato muy grande:

- en general, las estimaciones de más de 15 días tienen una probabilidad mucho menor de ser correctas que las estimaciones de menos de 15 días
- hay mucho espacio para errores

¿Qué hacemos frente a una estimación de más de 15 días?

Dividamos el relato en relatos más pequeños, más fáciles de estimar:

p.ej., apliquemos la regla del “y” —cualquier relato que tenga un “y” en su título o descripción, probablemente puede ser dividido en dos o más relatos más pequeños

Hablemos con el cliente … de nuevo:

- tal vez hay suposiciones que están aumentando la estimación

  … si el cliente pudiera clarificar las cosas, podríamos deshacernos de esas suposiciones

  … y reducir la estimación

El propósito de *planning poker* es converger

Después de una rueda de *planning poker*

en que hemos eliminado tantas suposiciones como ha sido posible

… uno debiera no sólo tener estimaciones para cada relato:

p.ej., si las estimaciones para un relato son 3, 5 y 8 días, es decir, están cerca unas de otras

… sino también tener confianza en esas estimaciones:

… entonces podemos llegar a consenso con una estimación de, p.ej., 5 días

En resumen, sigamos estos pasos

1) Hablar con el cliente

2) Jugar *planning poker* para cada relato de usuario

3) Clarificar las suposiciones:

- volver al paso 1 si identificamos suposiciones que sólo el cliente puede clarificar

4) Llegar a consenso para la estimación de cada relato:

- cuando tengamos confianza en la estimación
- … y nos sintamos cómodos con las suposiciones que hicimos

Finalmente, podemos estimar el proyecto completo

Tenemos relatos de usuario cortos y focalizados:

- hemos jugado *planning poker* para cada relato
- hemos aclarado todas las suposiciones que estábamos haciendo en las estimaciones

… y tenemos estimaciones en las cuales todos creemos

Sumemos todas las estimaciones (consensuadas) para obtener una estimación total de cuánto demorará el proyecto en entregar el software que se necesita

La interpretación de *puntos de relato*

Una versión alternativa de esta técnica propone que los valores de los naipes—1, 2, 3, 5, 8, 13, …—no representan necesariamente días de desarrollo,

… sino simplemente la *magnitud de la dificultad* de desarrollar el relato correspondiente—esta unidad se la llama **puntos de relato** (*story points*)

Al final de la primera iteración, se calcula cuántos puntos de relato fueron efectivamente implementados, con qué esfuerzo (personas-semanas) y en qué tiempo

… y así se calibra, *inicialmente*, la relación entre puntos de relato, esfuerzo y tiempo para el resto del proyecto