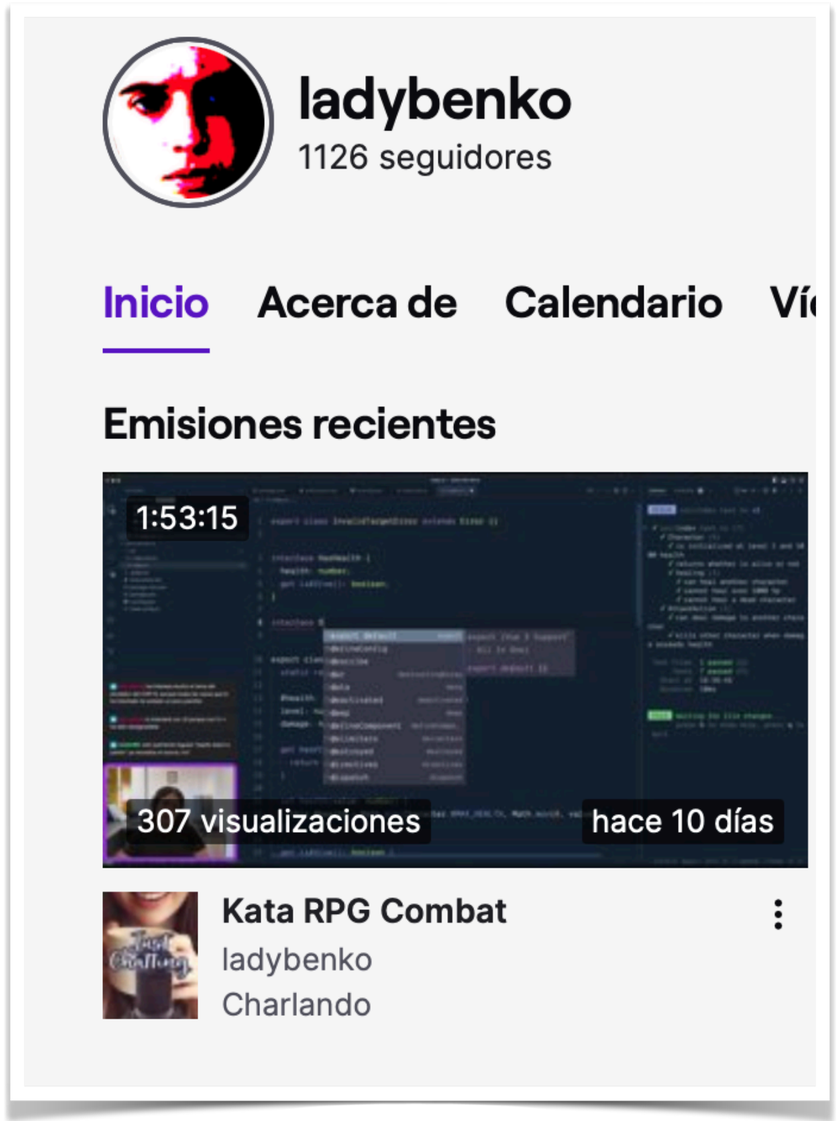
# MADS
# S2: Desarrollo de software

# El post de la semana




*Emisión de Twitch de Belén Albeza:*
*Kata RPG Combat*

*@ladybenko*

# ¿Son necesarios los desarrolladores en la era de la IA?

- Repasamos el artículo de Henrik Kniberg.
- Antes del debate, vamos a hacer una **pequeña demo de cómo GPT-4 resolvería la práctica 1**.
- Una vez vista la demo:


- ¿El uso de la IA va a permitir acortar las iteraciones en los proyectos ágiles?
- ¿Es exagerada la postura de Kniberg de que no van a ser necesarios?¿Qué contra-argumentos se pueden ofrecer?
- ¿Para qué serán necesarios los/las desarrolladores/as dentro de 10 años?

# Índice

# 1. Software

BASIC PROGRAMMING

UNIVAC I
Data Automation System

Donald Knuth
IBM 650 - 1958

Programmer's Primer for
**FORTRAN**
Automatic Coding System
for the IBM 704

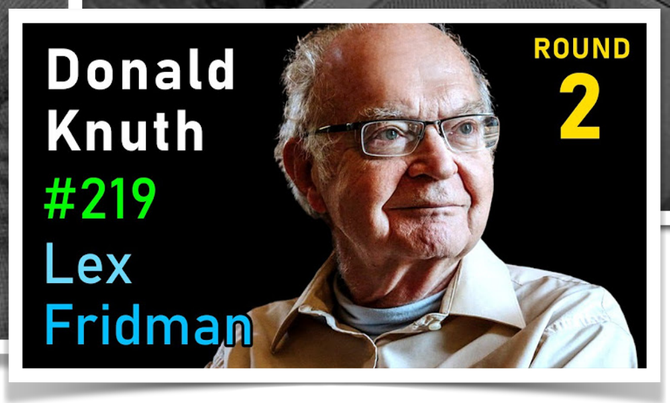
© 1957 by Inte

LISP I

PROGRAMMER'S MANUAL

March 1, 1960

COMPUTATION CENTER
and
RESEARCH LABORATORY OF ELECTRONICS
Massachusetts Institute of Technology
Cambridge, Massachusetts




EXIT


android


9:41
Clock
Camera
375/500CAL
19/30MIN
4/12HRS
TV
Stocks
Activity
FRIENDS
Sep 10, 2019
Mail
Files
Photos
App Store
Weather
Health
Home
Chromatica
Lady Gaga
News
Photos
Music


Radar
COVID


IJ


AMBULANCIA
M-375

- CartoDB. Software español para representación visual de datos geográficos.
- Guice. Framework de inyección de dependencias en Java.
- swift-nio. Framework asíncrono de entrada-salida en Swift.
- Spring Boot. Framework web en Java
- Swift. Compilador y librería stándar de Swift. Escrito en C++ y Swift.

# 2. Metáforas

# metáfora.

(Del lat. *metaphŏra,* y este del gr. μεταφορά, traslación).

**1.** f. *Ret.* Tropo que consiste en trasladar el sentido recto de las voces a otro figurado, en virtud de una comparación tácita; p. ej., *Las perlas del rocío. La primavera de la vida. Refrenar las pasiones.*

**2.** f. Aplicación de una palabra o de una expresión a un objeto o a un concepto, al cual no denota literalmente, con el fin de sugerir una comparación (con otro objeto o concepto) y facilitar su comprensión; p. ej., *el átomo es un sistema solar en miniatura.*

## ~ continuada.

**1.** f. *Ret.* Alegoría en que unas palabras se toman en sentido recto y otras en sentido figurado.

---

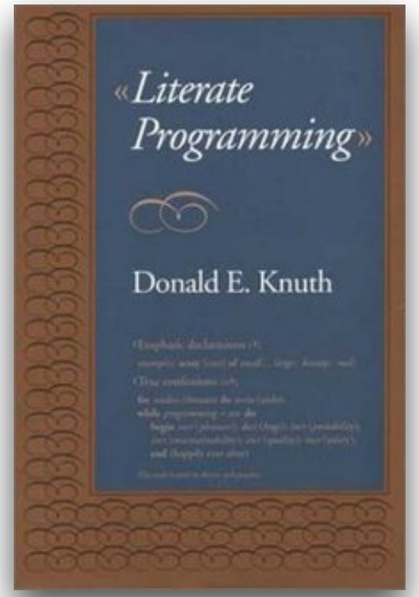
"Hemos ganado este combate"
"Nos lo jugamos todo en esta campaña"
"Es una oportunidad de vida o muerte"

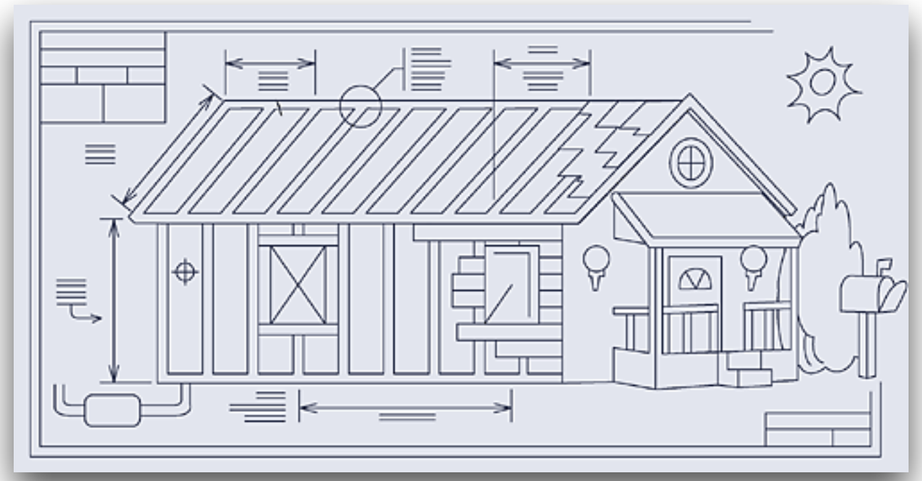
Vs.

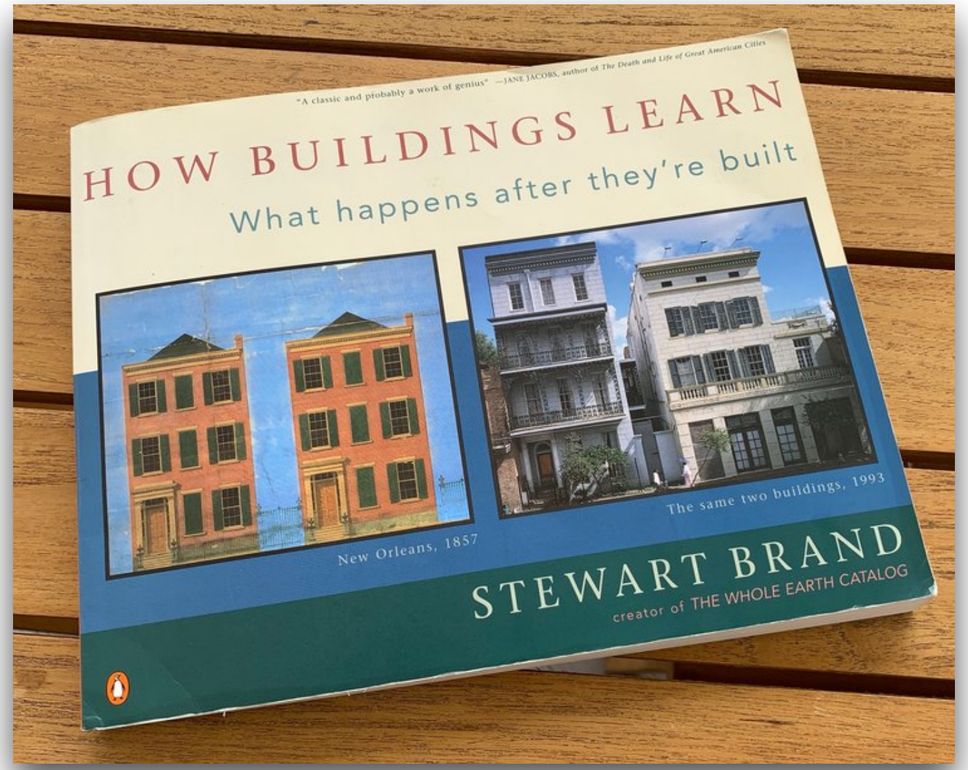
"Hemos coreografiado perfectamente la puesta en marcha del producto"
"Todos debemos participar en este viaje"
"Nuestro ecosistema premia la lealtad de nuestros clientes"

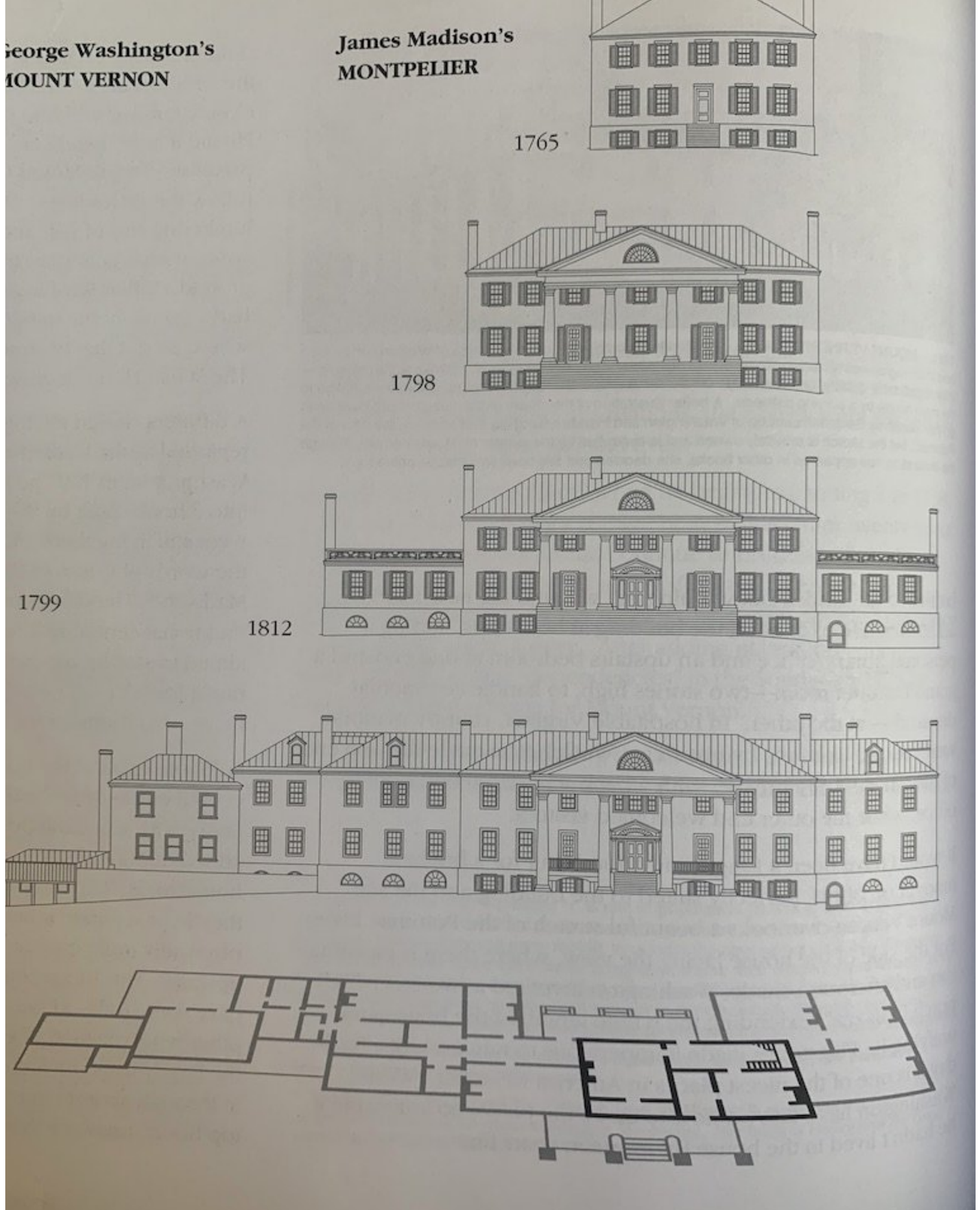
# Metáforas del desarrollo de software

- El desarrollo de software es como ...

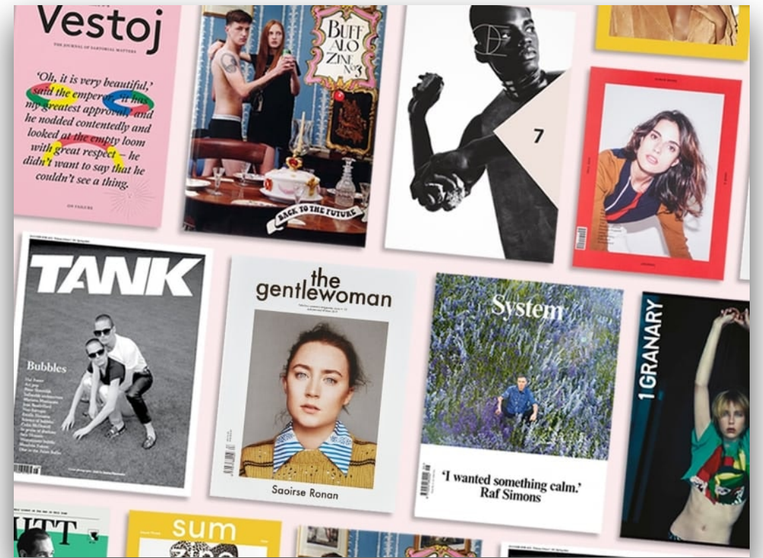


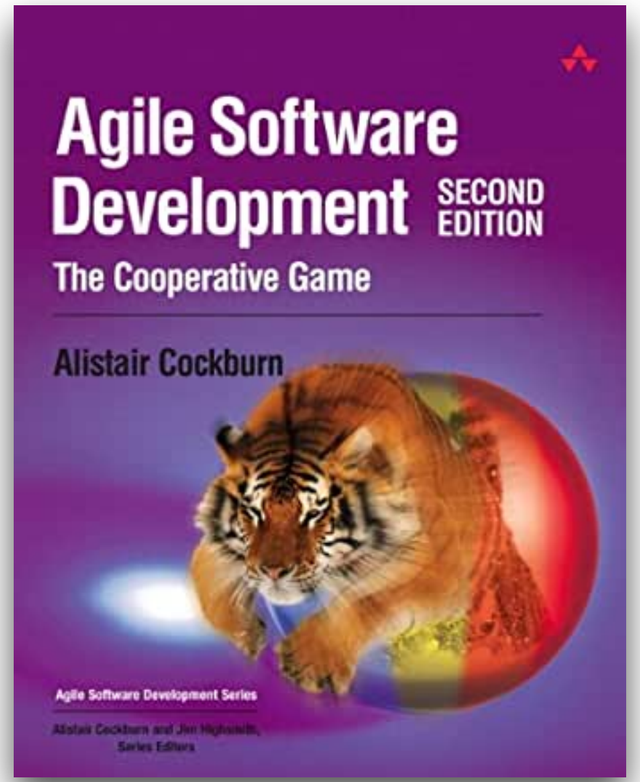




"A classic and probably a work of genius" —JANE JACOBS, author of The Death and Life of Great American Cities
HOW BUILDINGS LEARN
What happens after they're built
New Orleans, 1857
The same two buildings, 1993
STEWART BRAND
creator of THE WHOLE EARTH CATALOG

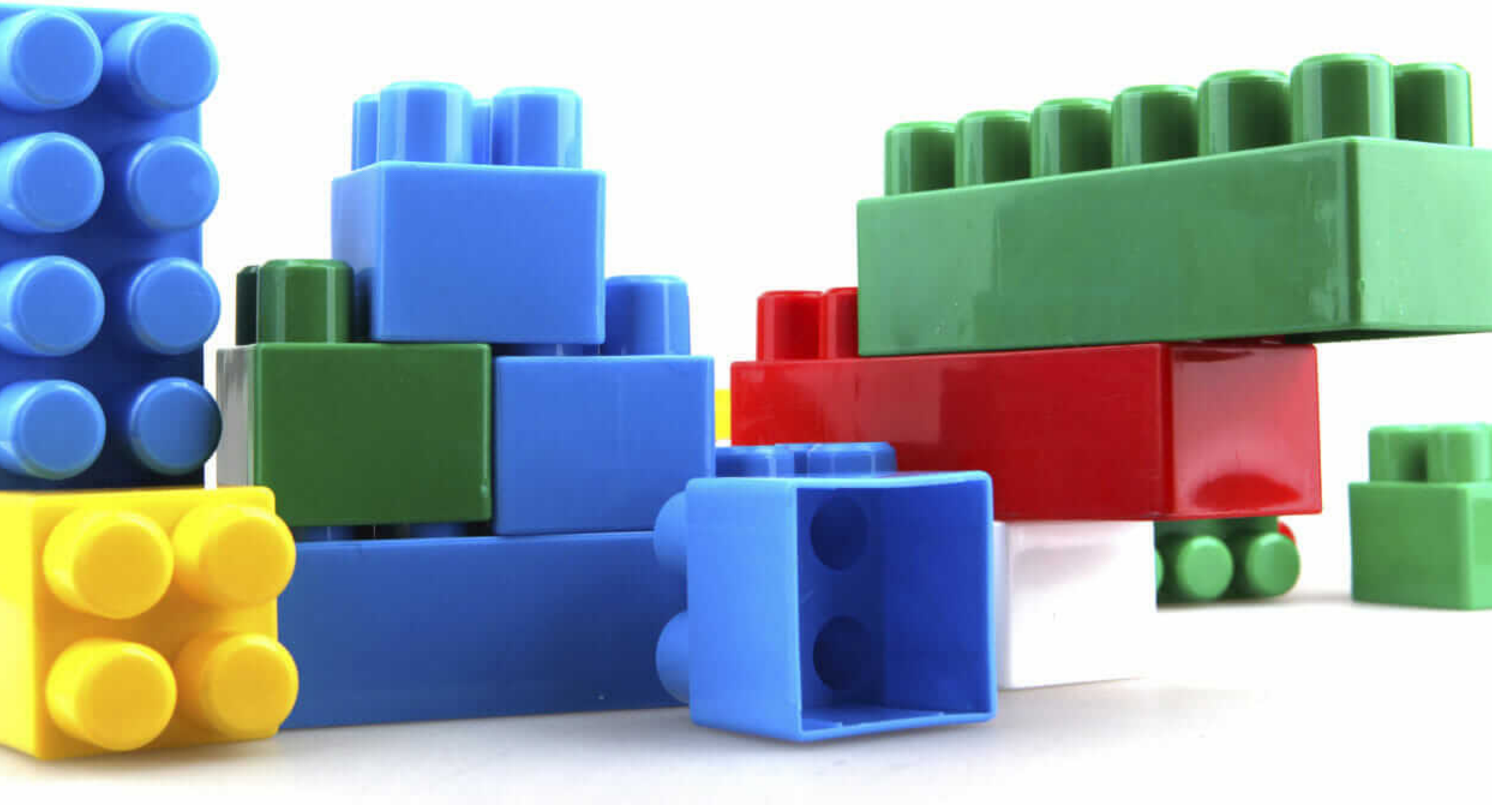

George Washington's
MOUNT VERNON
James Madison's
MONTPELIER
1765
1798
1799
1812

- El desarrollo de software es como ...

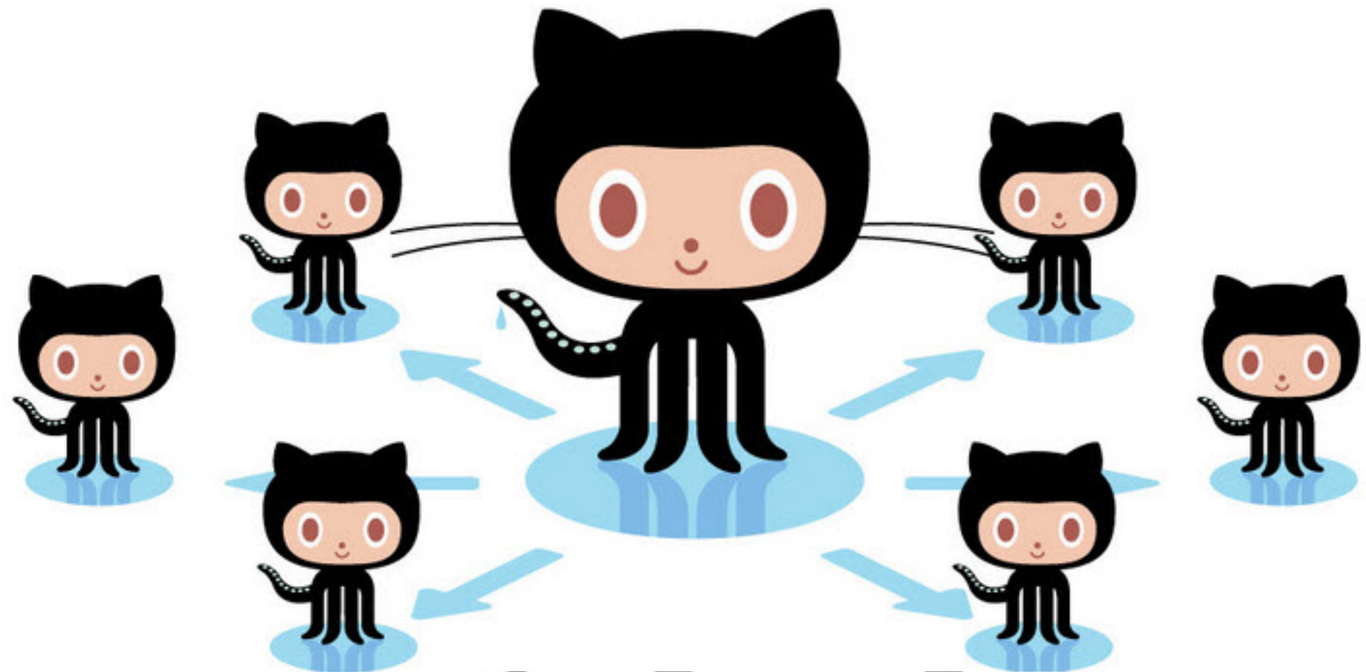


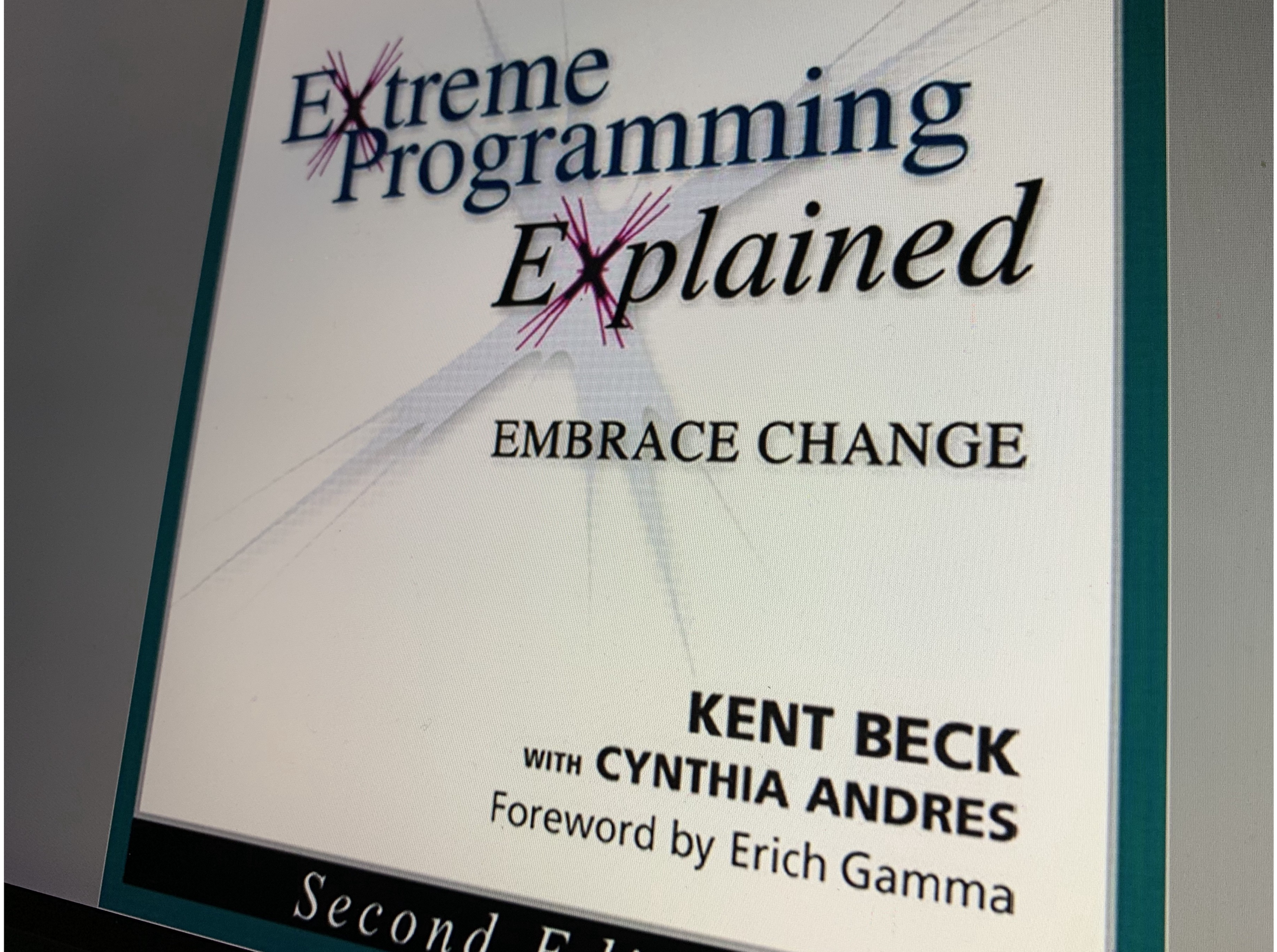
# El software es único



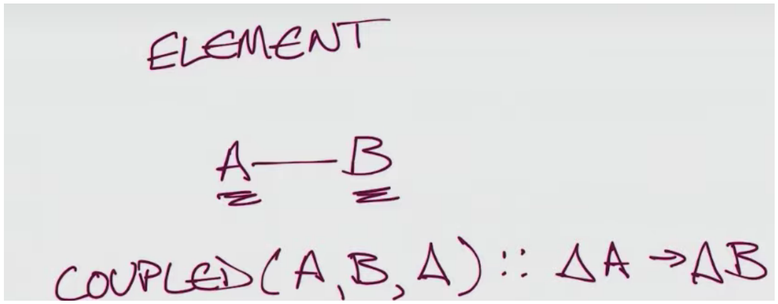

We

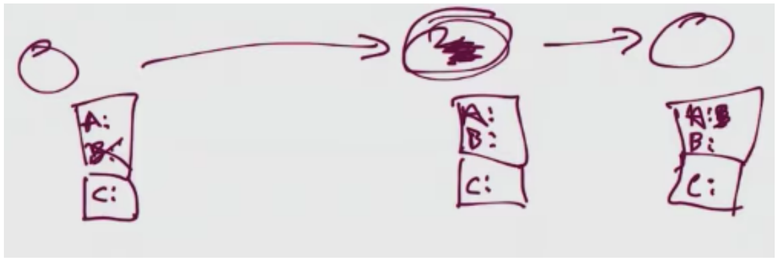


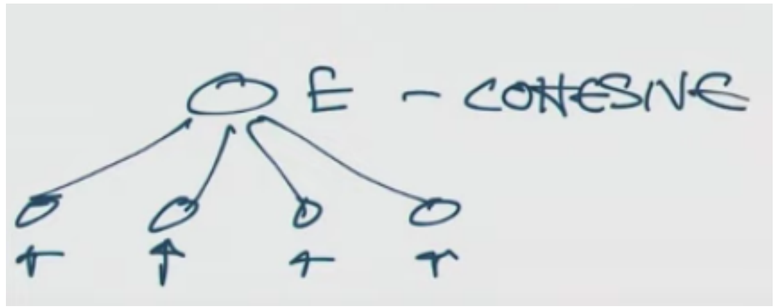
# github
## SOCIAL CODING

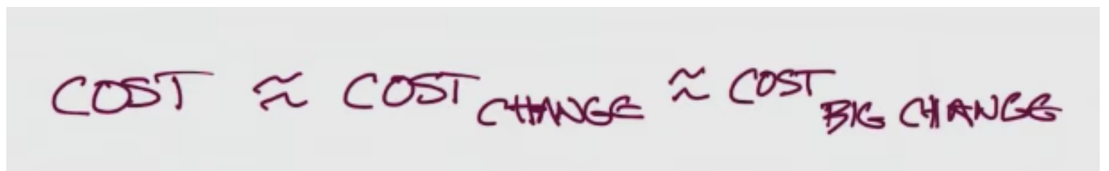

Extreme Programming Explained
EMBRACE CHANGE
KENT BECK
WITH CYNTHIA ANDRES
Foreword by Erich Gamma

# Continued Learning: The Beauty of Maintenance
# Kent Beck - DDD Europe 2020

# 3. El desarrollo de software no es una ingeniería tradicional

# The New Methodology

*In the past few years there's been a blossoming of a new style of software methodology - referred to as agile methods. Alternatively characterized as an antidote to bureaucracy or a license to hack they've stirred up interest all over the software landscape. In this essay I explore the reasons for agile methods, focusing not so much on their weight but on their adaptive nature and their people-first orientation.*

13 December 2005

Martin Fowler

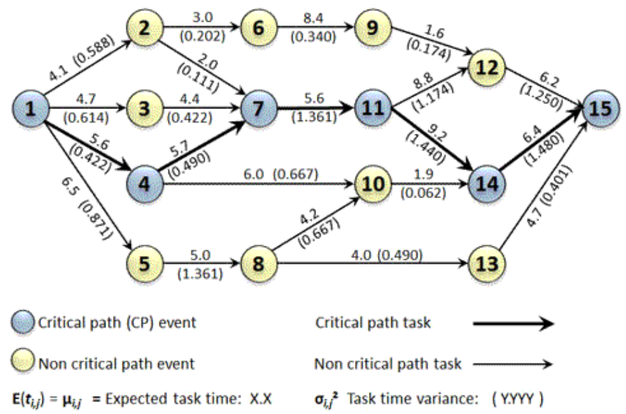
AGILE

PROCESS THEORY

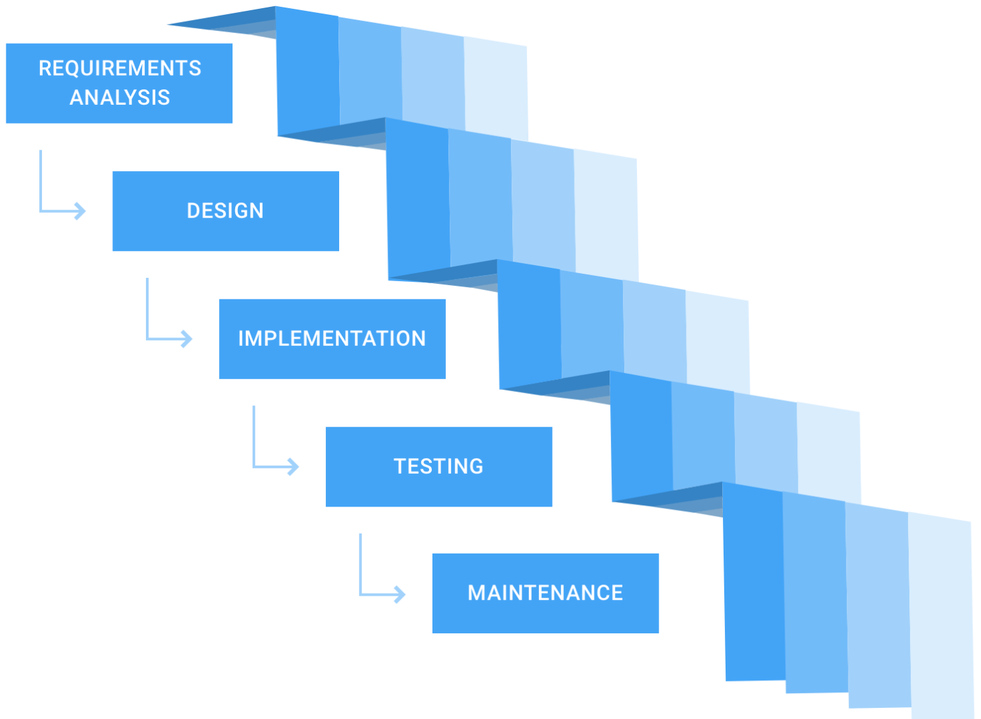
**CONTENTS**

## Resource Usage for Black Sea Port Construction Project

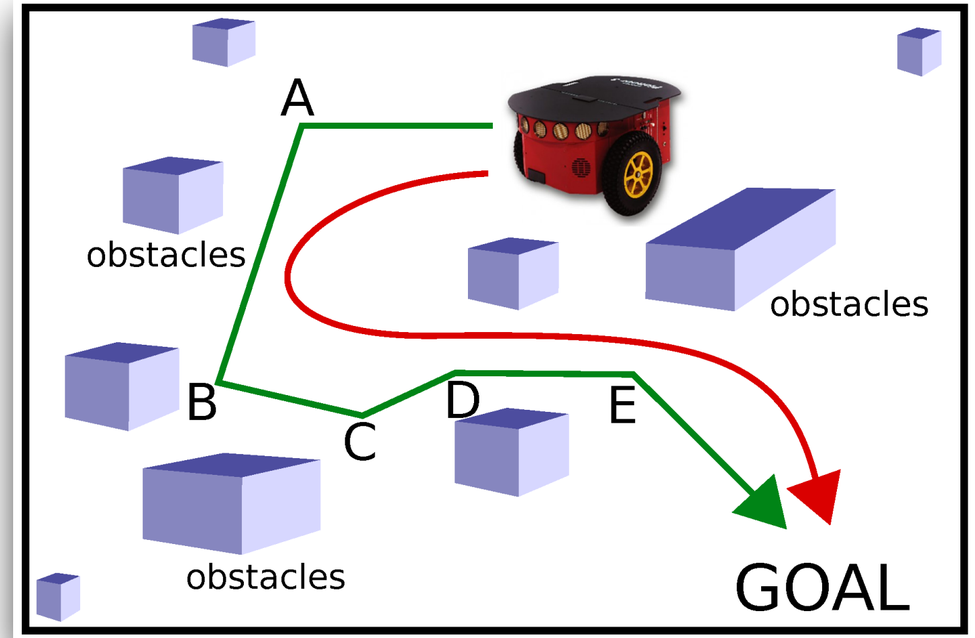
| # | Resource Name | Overallocated, h | Allocated, h |
|---|---|---|---|
| 1 | Andrew Anderson | 0.0 | 160.0 |
| 2 | Barbara Taylor | 4.0 | 168.0 |
| 3 | Charles Lewis | 0.0 | 396.0 |
| 4 | David Harris | 0.0 | 72.0 |
| 5 | Donna Hall | 0.0 | 324.0 |
| 6 | Helen Clark | 48.0 | 704.0 |
| 7 | James Smith | 38.9 | 236.6 |
| 8 | John Brown | 334.8 | 924.6 |
| 9 | Joseph Allen | 120.0 | 608.0 |
| 10 | Karen Martin | 457.6 | 781.2 |
| 11 | Laura Rodriguez | 0.0 | 88.0 |
| 12 | Linda Davis | 86.9 | 764.6 |
| 13 | Mark Robinson | 120.0 | 608.0 |
| 14 | Mary Williams | 554.1 | 915.5 |
| 15 | Michael Smith | 64.0 | 152.0 |
| 16 | Nancy Garcia | 0.0 | 96.0 |
| 17 | Patricia Jones | 62.9 | 212.5 |
| 18 | Paul King | 0.0 | 296.0 |

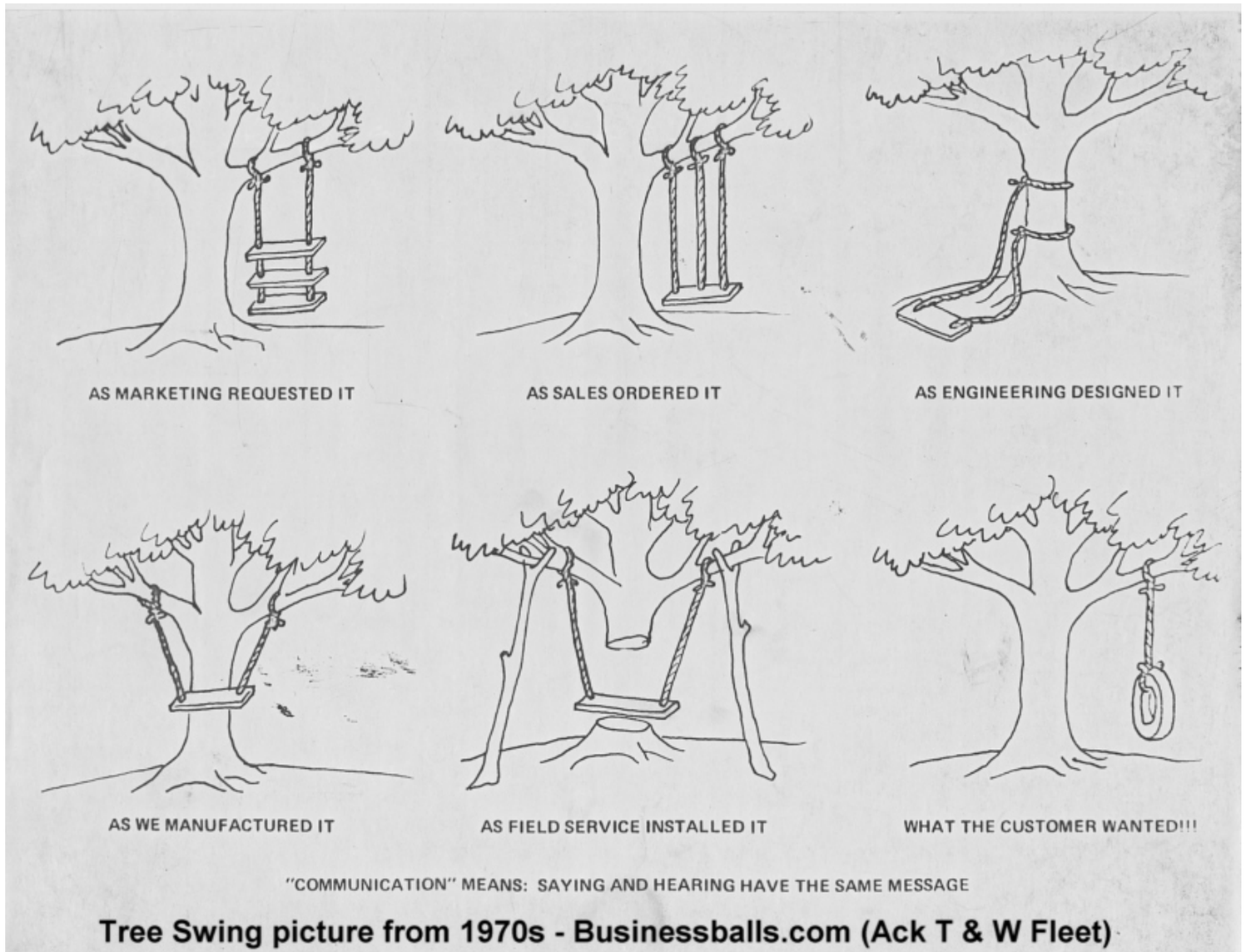



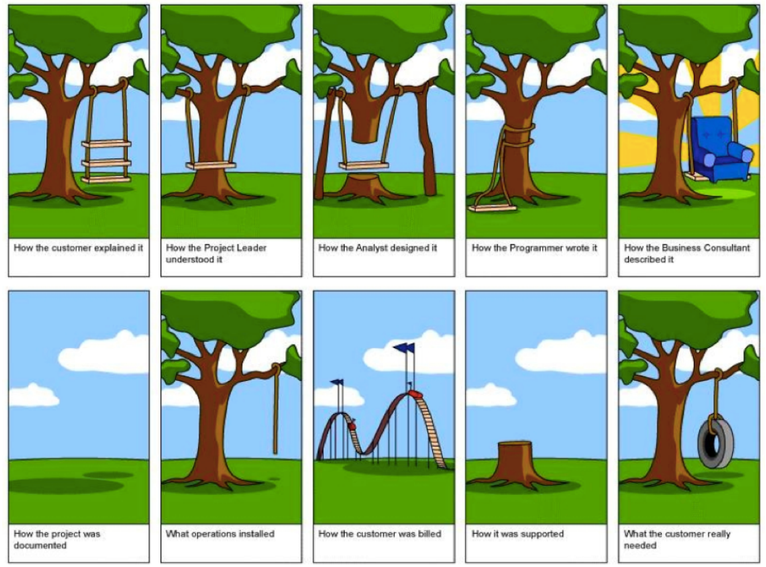
## Waterfall Model

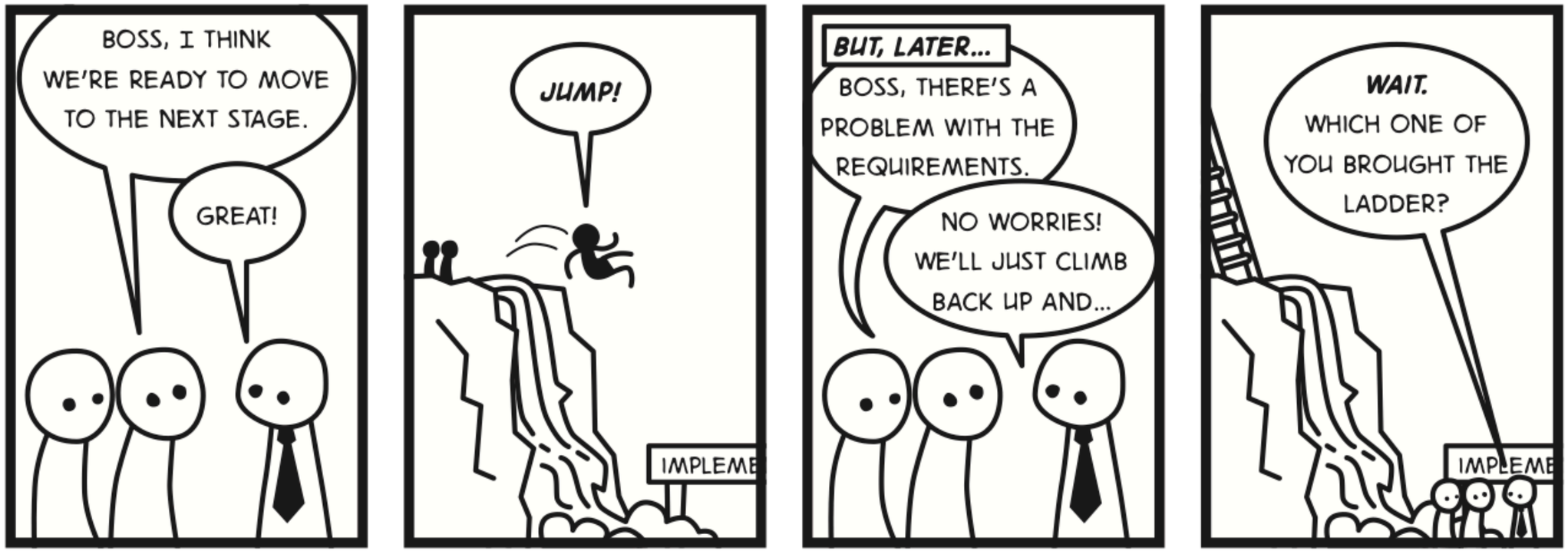







AS MARKETING REQUESTED IT
AS SALES ORDERED IT
AS ENGINEERING DESIGNED IT
AS WE MANUFACTURED IT
AS FIELD SERVICE INSTALLED IT
WHAT THE CUSTOMER WANTED!!!
"COMMUNICATION" MEANS: SAYING AND HEARING HAVE THE SAME MESSAGE
Tree Swing picture from 1970s - Businessballs.com (Ack T & W Fleet)

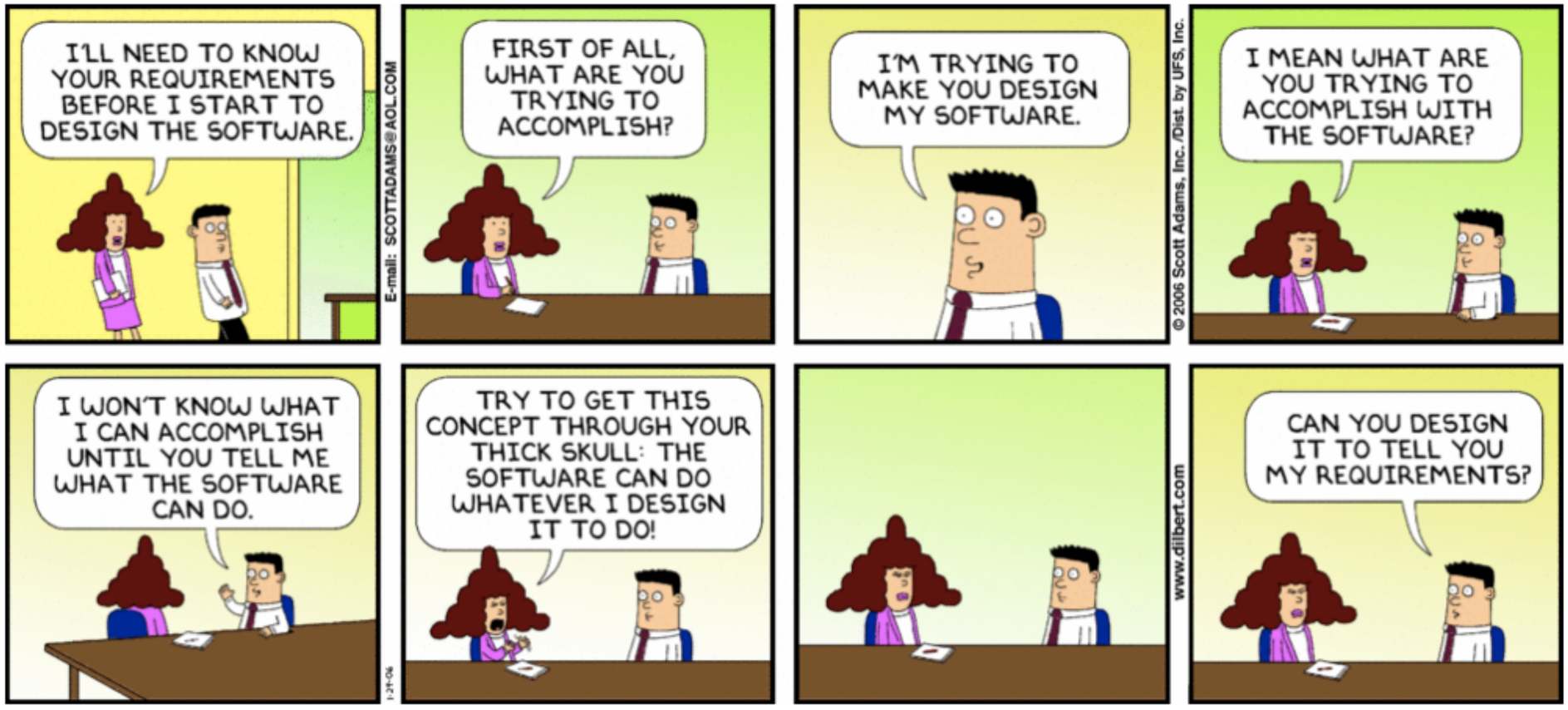

How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

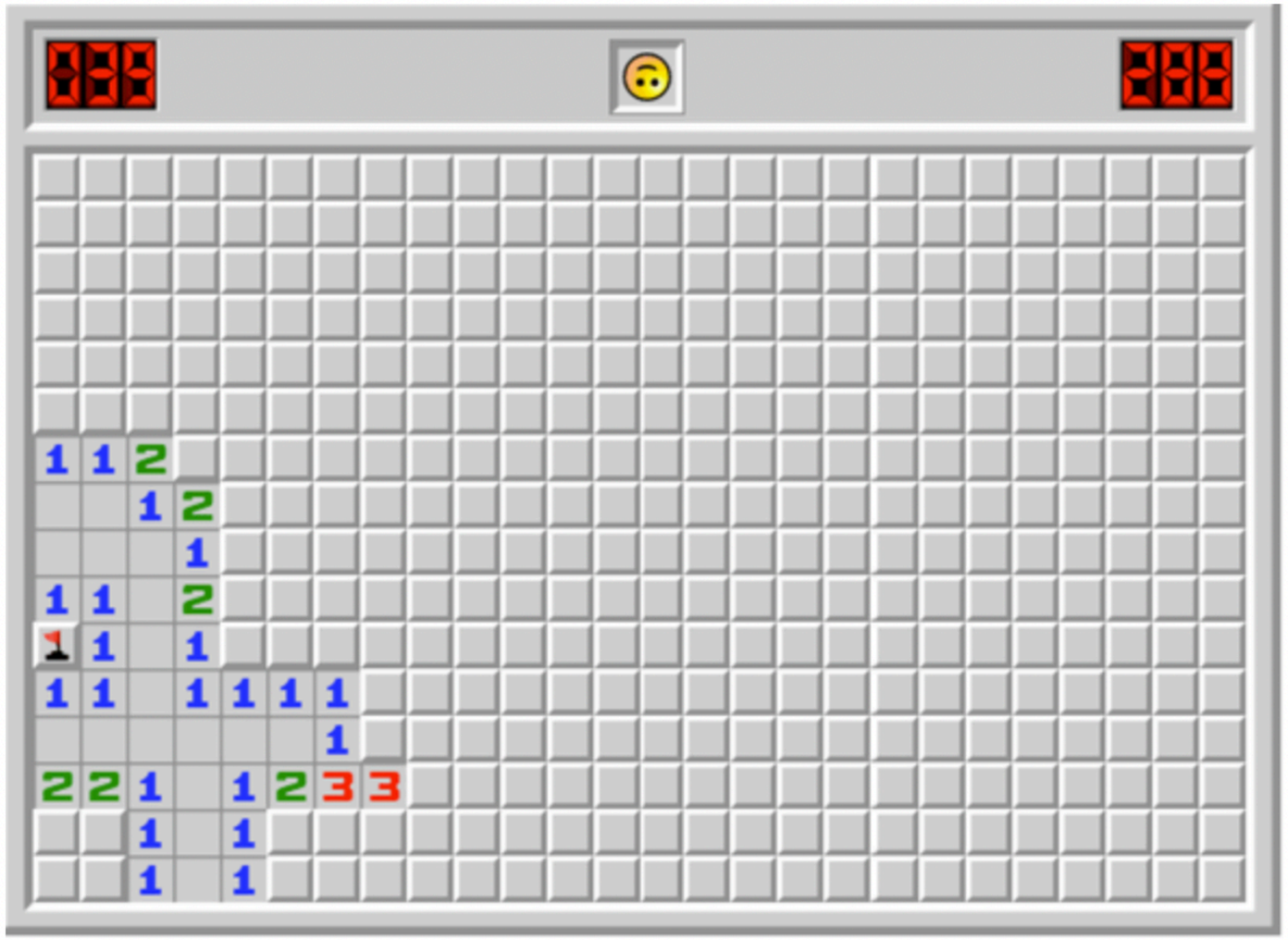

BOSS, I THINK WE'RE READY TO MOVE TO THE NEXT STAGE.
GREAT!
JUMP!
IMPLEME
BUT, LATER...
BOSS, THERE'S A PROBLEM WITH THE REQUIREMENTS.
NO WORRIES! WE'LL JUST CLIMB BACK UP AND...
WAIT.
WHICH ONE OF YOU BROUGHT THE LADDER?
IMPLEME

# No es una ingeniería tradicional

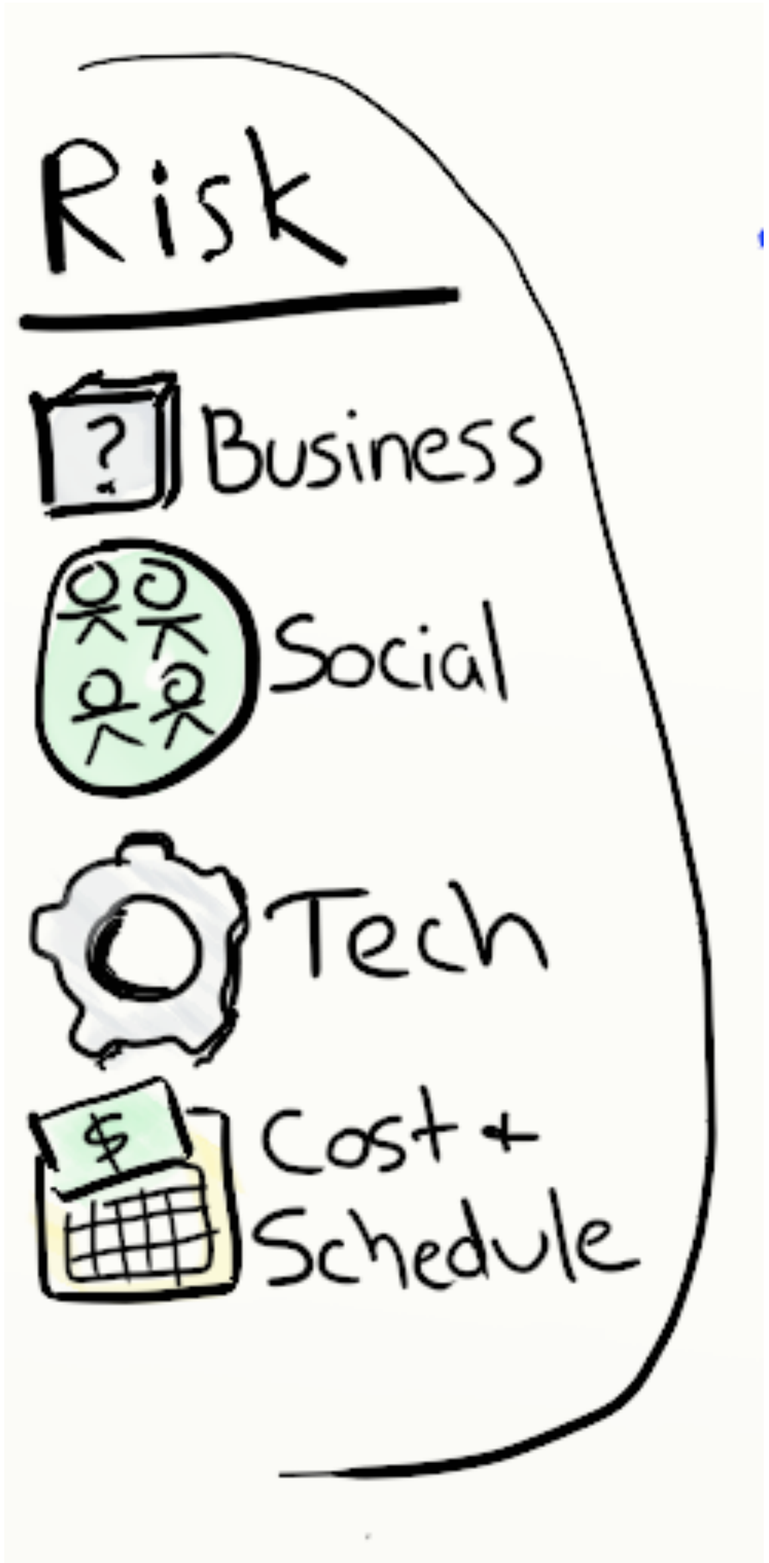
- En el software la parte de construcción es muy barata.
- En el software todo el esfuerzo es diseño y requiere, por tanto, de personas creativas y con talento.
- Los procesos creativos no se pueden planificar fácilmente, por lo que la predictividad es un objetivo imposible.

# ¿Los requisitos son predecibles?

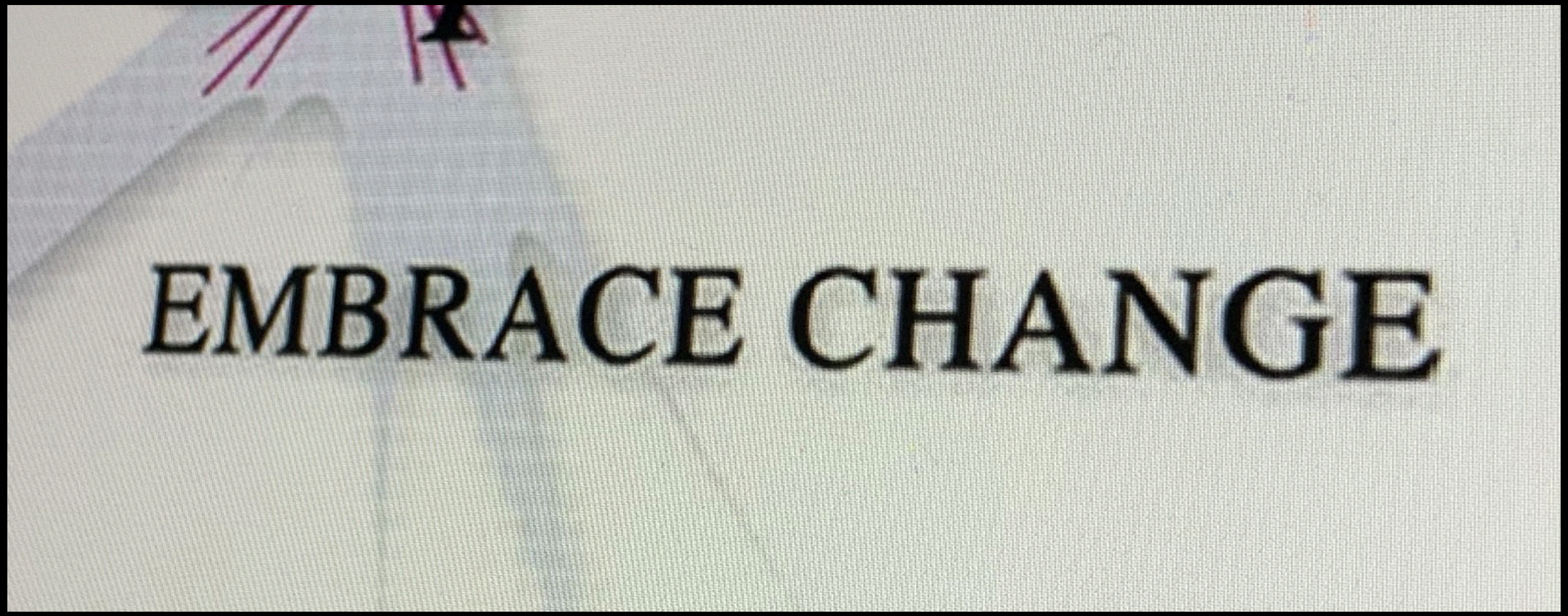




- **Lo que nos gustaría**
  - Los clientes saben lo que quieren
  - El equipo sabe cómo construirlo
  - Nada cambiará en el camino
  - Tenemos mucho tiempo y dinero para hacerlo
- **La realidad**
  - Los clientes descubren lo que necesitan
  - Los desarrolladores descubren cómo hacerlo
  - Muchas cosas cambian en el camino
  - Siempre hay más cosas qué hacer que tiempo y dinero disponible

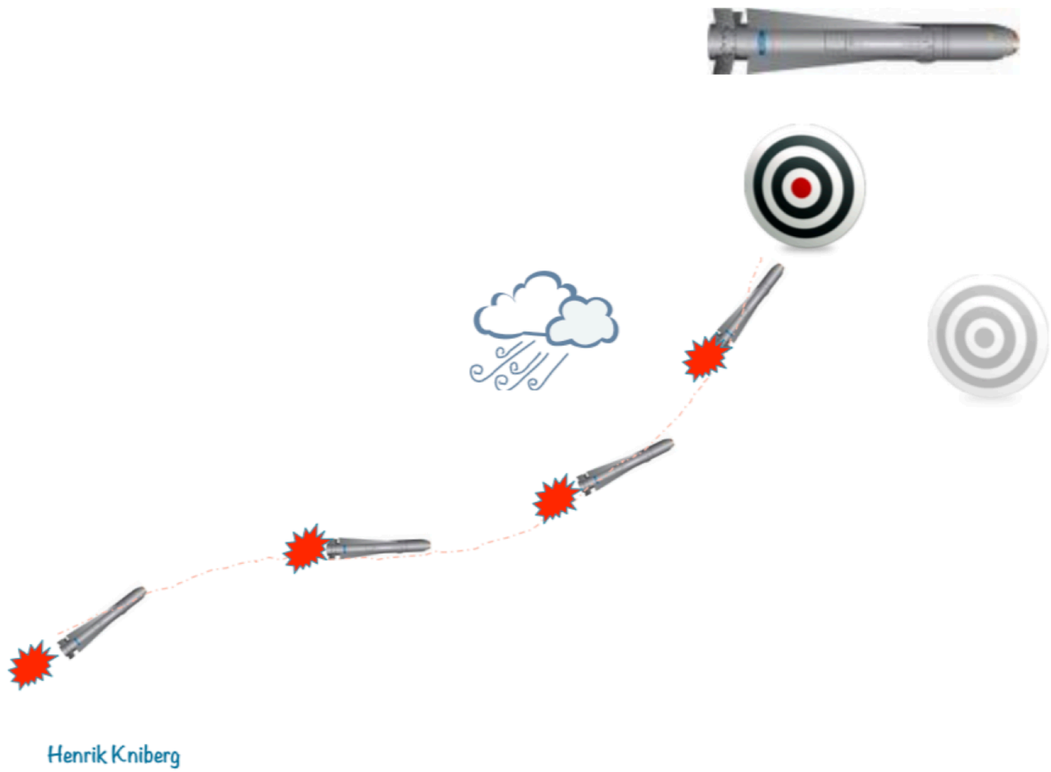

Risk
Business
Social
Tech
Cost +
Schedule

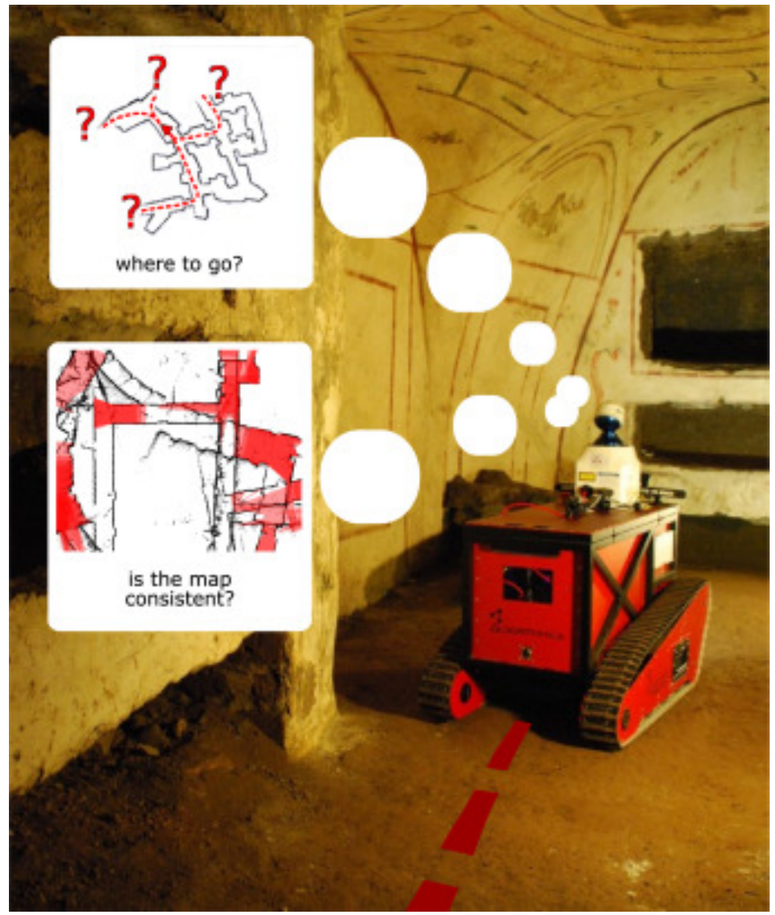

# NEW LAWS

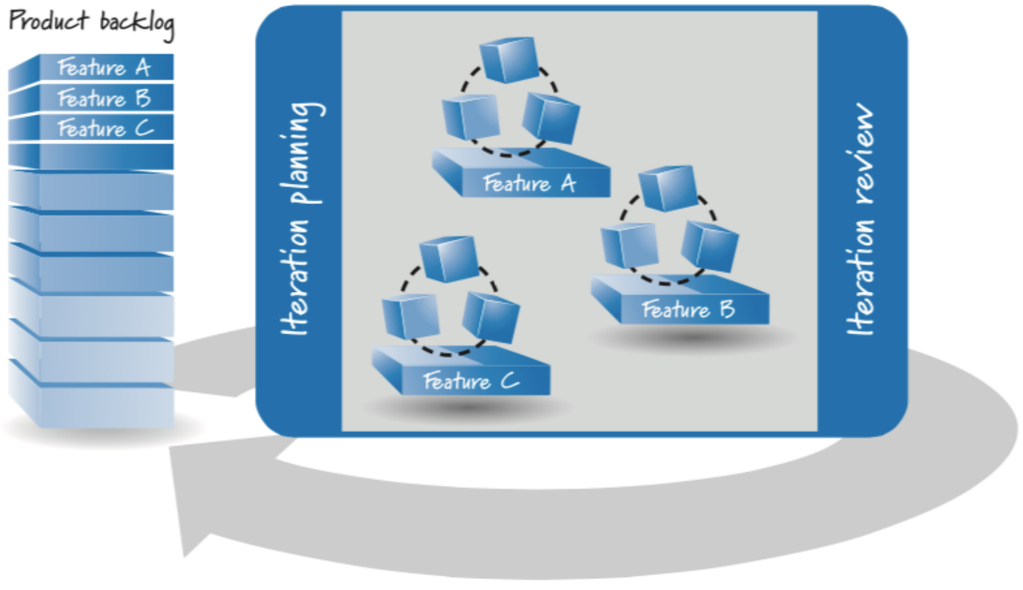

EMBRACE CHANGE

No es asumible que añadir nuevos campos a un modelo obligue a revisar y modificar las consultas SQL ya desarrolladas.

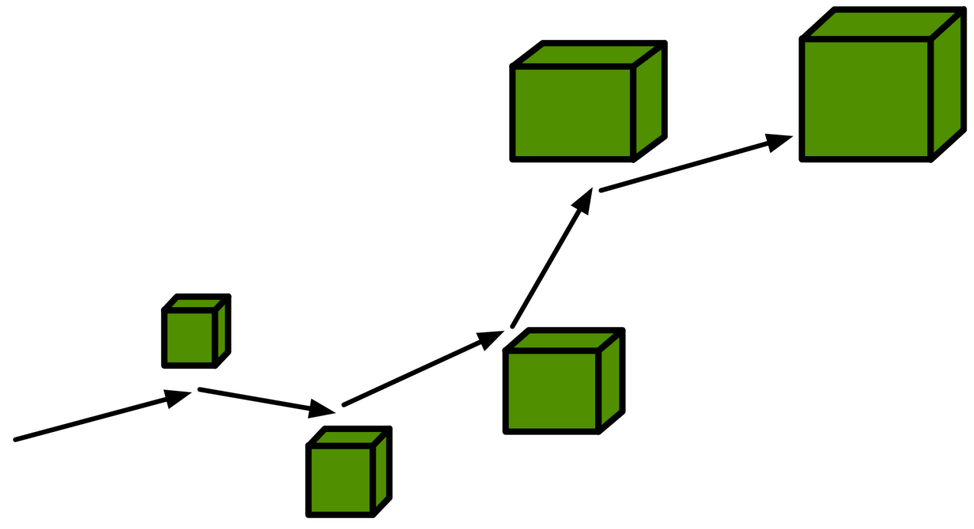
# ¿Cómo controlar un proceso no predecible?

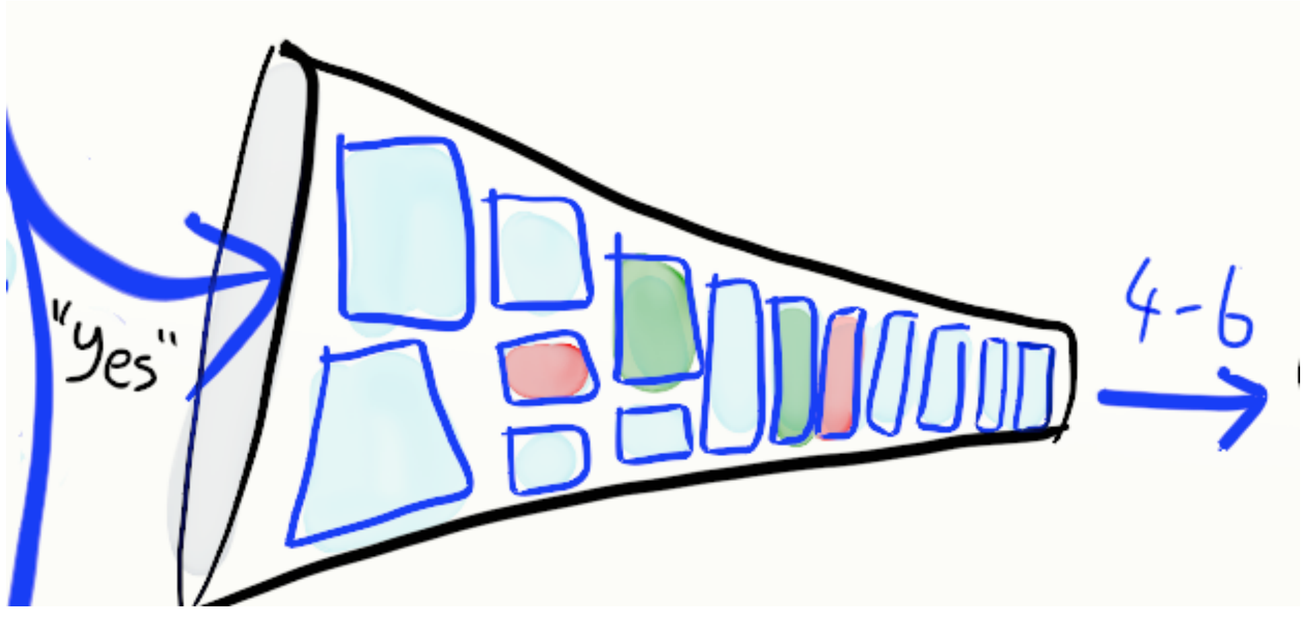


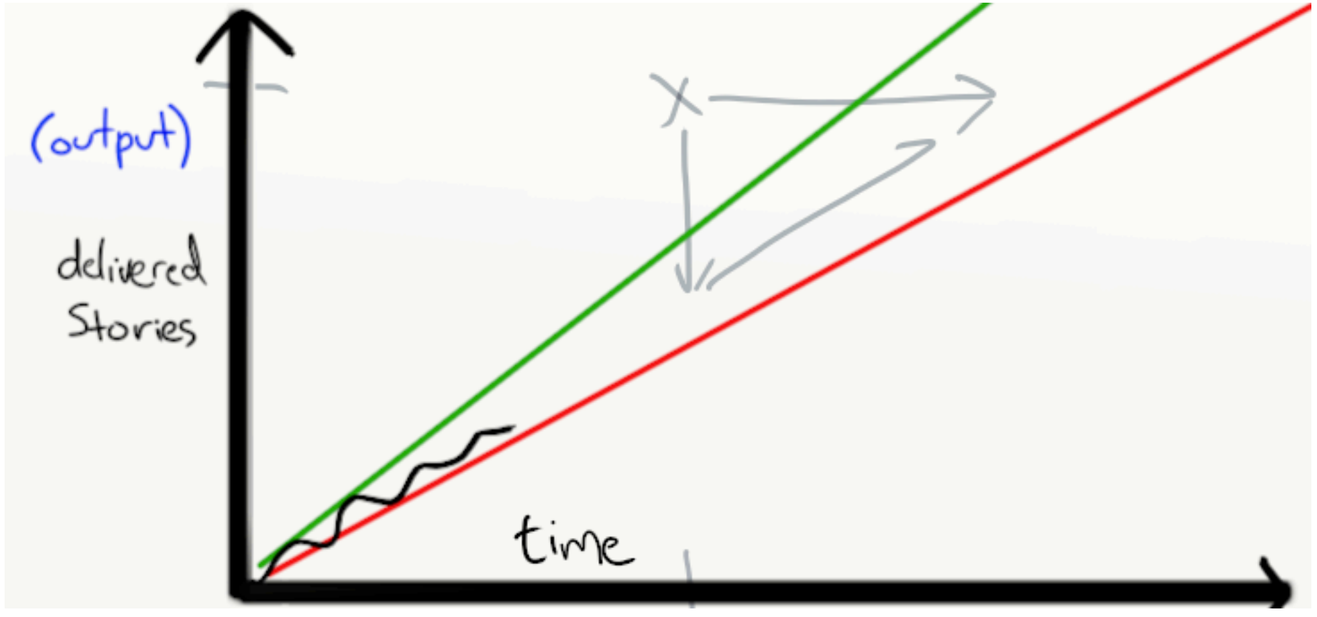






Iteration (1 week to 1 calendar month)



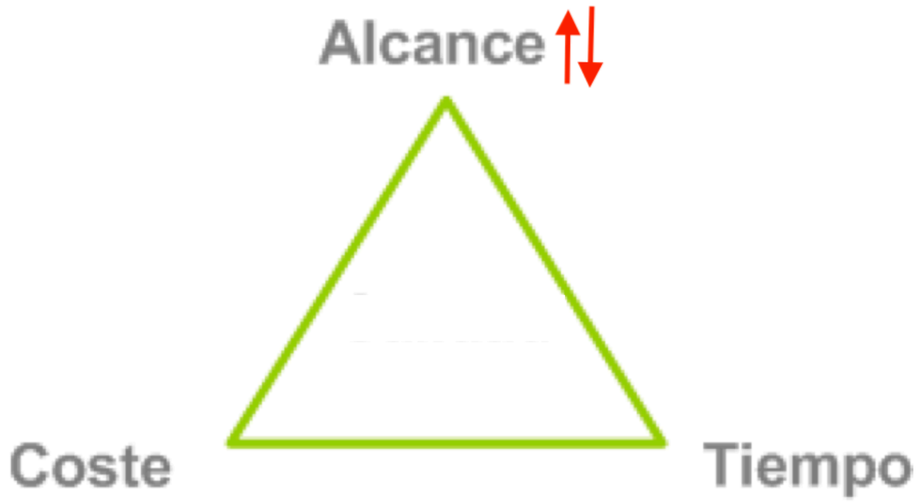

"yes"
4-6

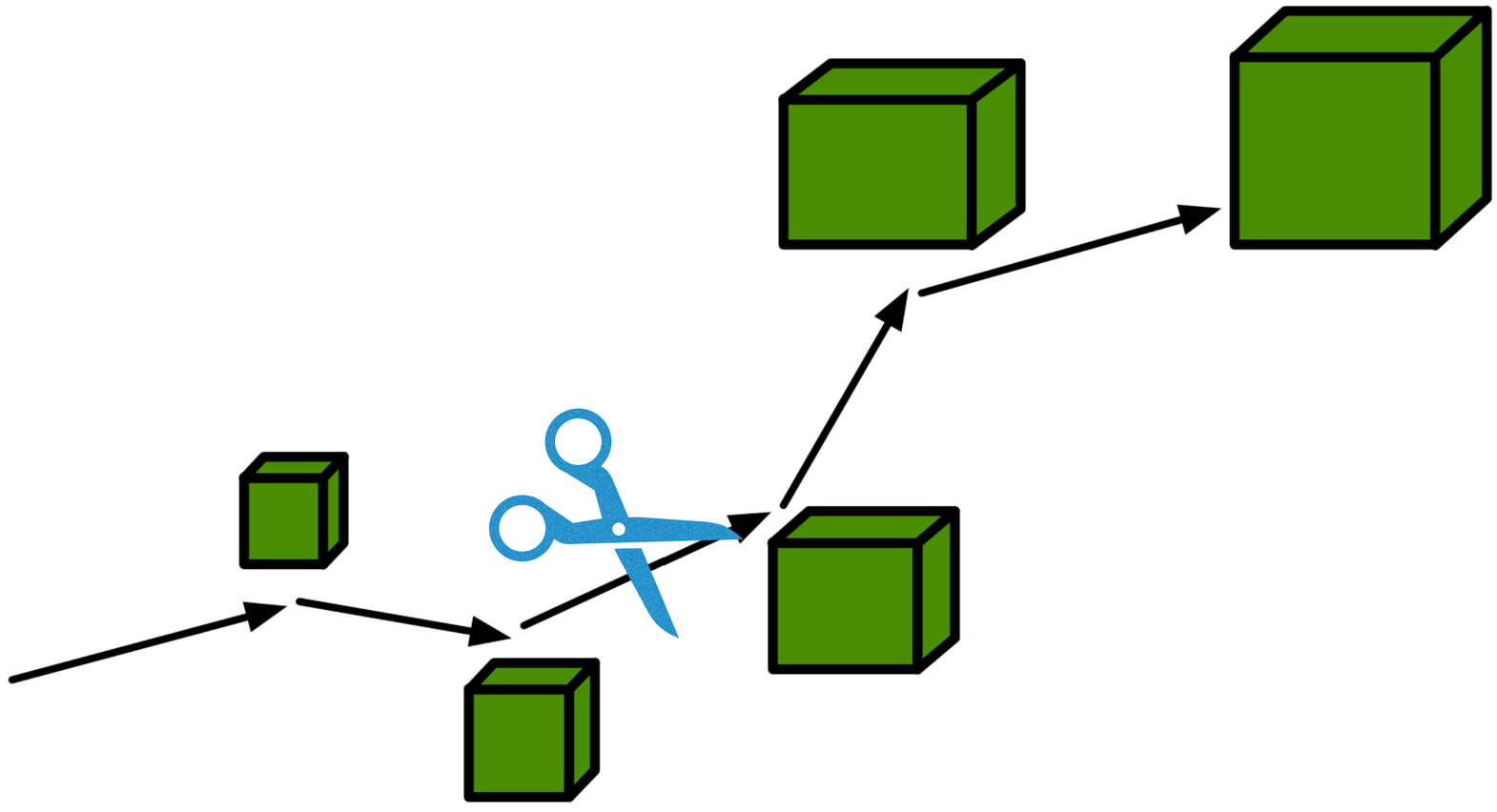

(output)
delivered
Stories
time

# Contratos ágiles

- **Contrato tradicional**: precio fijo, el cliente paga al final por unos requisitos en un plazo. Si se incumple el plazo o los requisitos el cliente puede no pagar y la empresa de software no entrega nada.

- **Contrato ágil**: presupuesto fijo, se paga a intervalos establecidos y la empresa de software entrega periódicamente el producto. Cualquiera de los dos puede cancelar el contrato en cualquier momento.

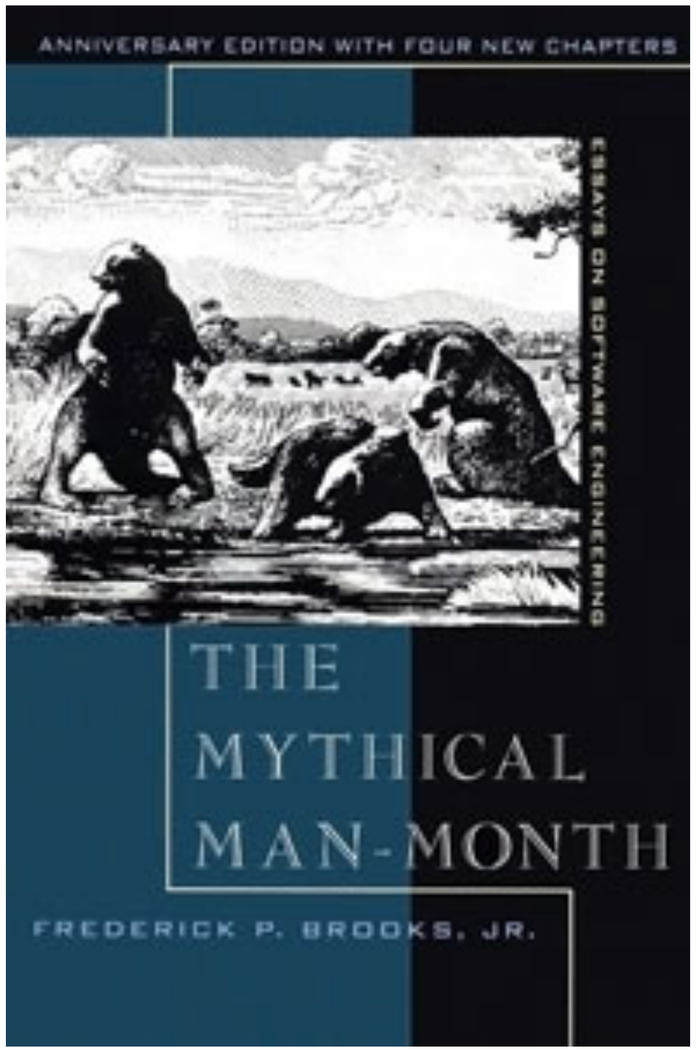




*"Un cambio de última hora en los requerimientos es una ventaja competitiva."*

Mary Poppendieck

# 4. El desarrollo de software es una actividad creativa




1975

"Adding manpower to a late software project makes it later."

Fred Brooks

# No Silver Bullet

## Essence and Accidents of Software Engineering

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill

**Fashioning complex conceptual constructs is the *essence;* *accidental* tasks arise in representing the constructs in language. Past progress has so reduced the accidental tasks that future progress now depends upon addressing the essence.**

Of all the monsters that fill the nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors. For these, one seeks bullets of silver that can magically lay them to rest.

The familiar software project, at least as seen by the nontechnical manager, has something of this character; it is usually innocent and straightforward, but is capable of becoming a monster of missed schedules, blown budgets, and flawed products. So we hear desperate cries for a silver bullet—something to make software costs drop as rapidly as computer hardware costs do.

But, as we look to the horizon of a decade hence, we see no silver bullet. There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity. In this article, I shall try to show why, by examining both the nature of the software problem and the properties of the bullets proposed.

Skepticism is not pessimism, however. Although we see no startling breakthroughs—and indeed, I believe such to be inconsistent with the nature of software—many encouraging innovations are under way. A disciplined, consistent effort to develop, propagate, and exploit these innovations should indeed yield an order-of-magnitude improvement. There is no royal road, but there is a road.

The first step toward the management of disease was replacement of demon theories and humours theories by the germ theory. That very step, the beginning of hope, in itself dashed all hopes of magical solutions. It told workers that progress would be made stepwise, at great effort, and that a persistent, unremitting care would have to be paid to a discipline of cleanliness. So it is with software engineering today.

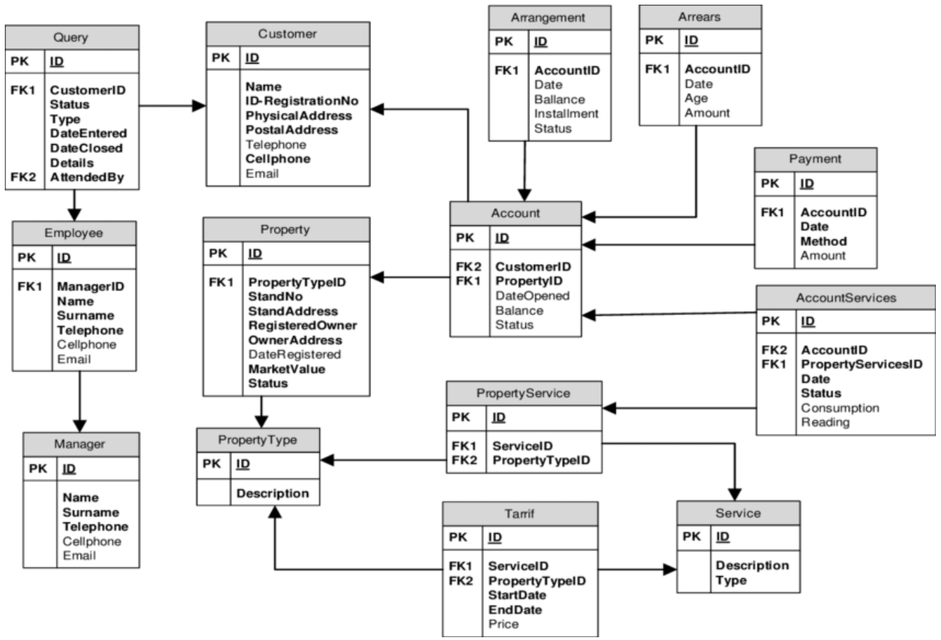
This article was first published in *Information Processing '86,* ISBN No. 0-444-70077-3, H.-J. Kugler, ed., Elsevier Science Publishers B.V. (North-Holland) © IFIP 1986.

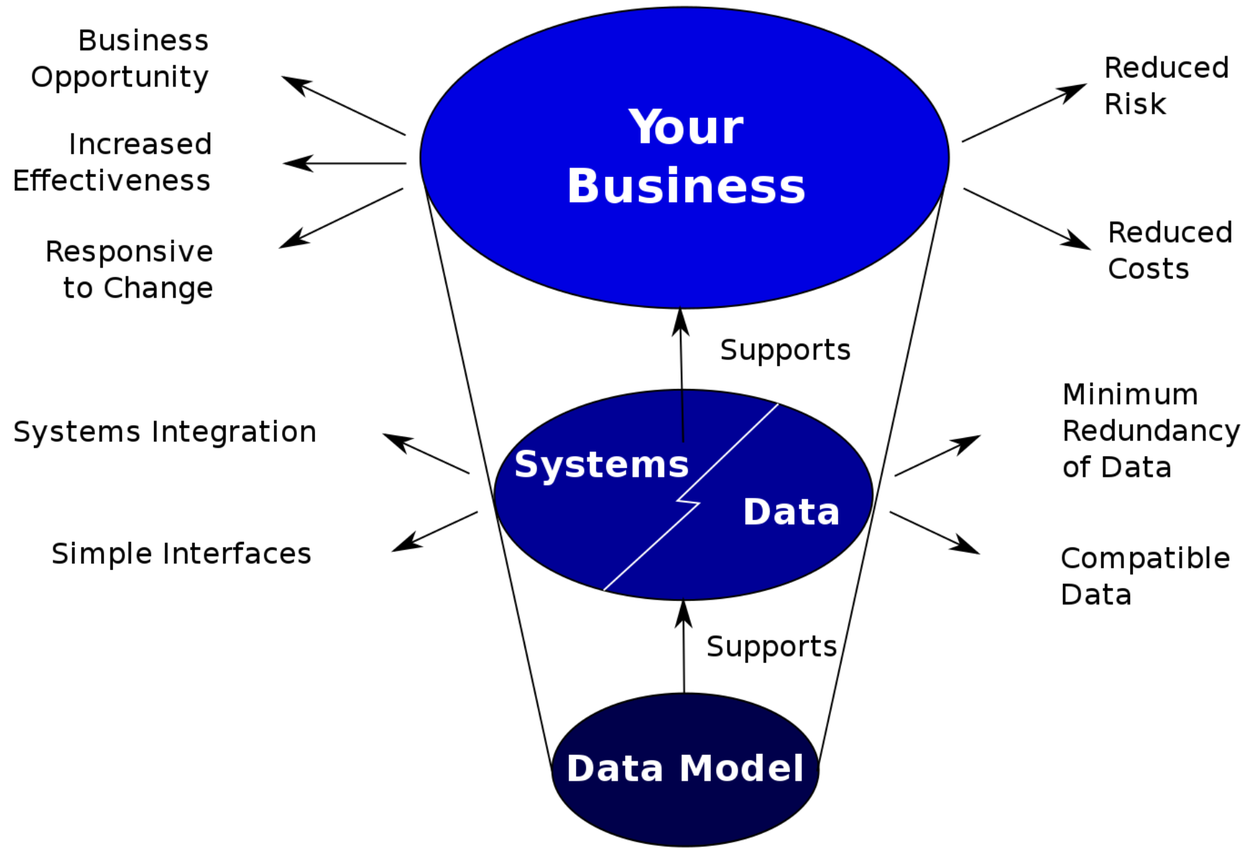
1986

## Does it have to be hard?—Essential difficulties

Not only are there no silver bullets now in view, the very nature of software makes it unlikely that there will be any—no inventions that will do for software productivity, reliability, and simplicity what electronics, transistors, and large-scale integration did for computer hardware.

# Tareas esenciales

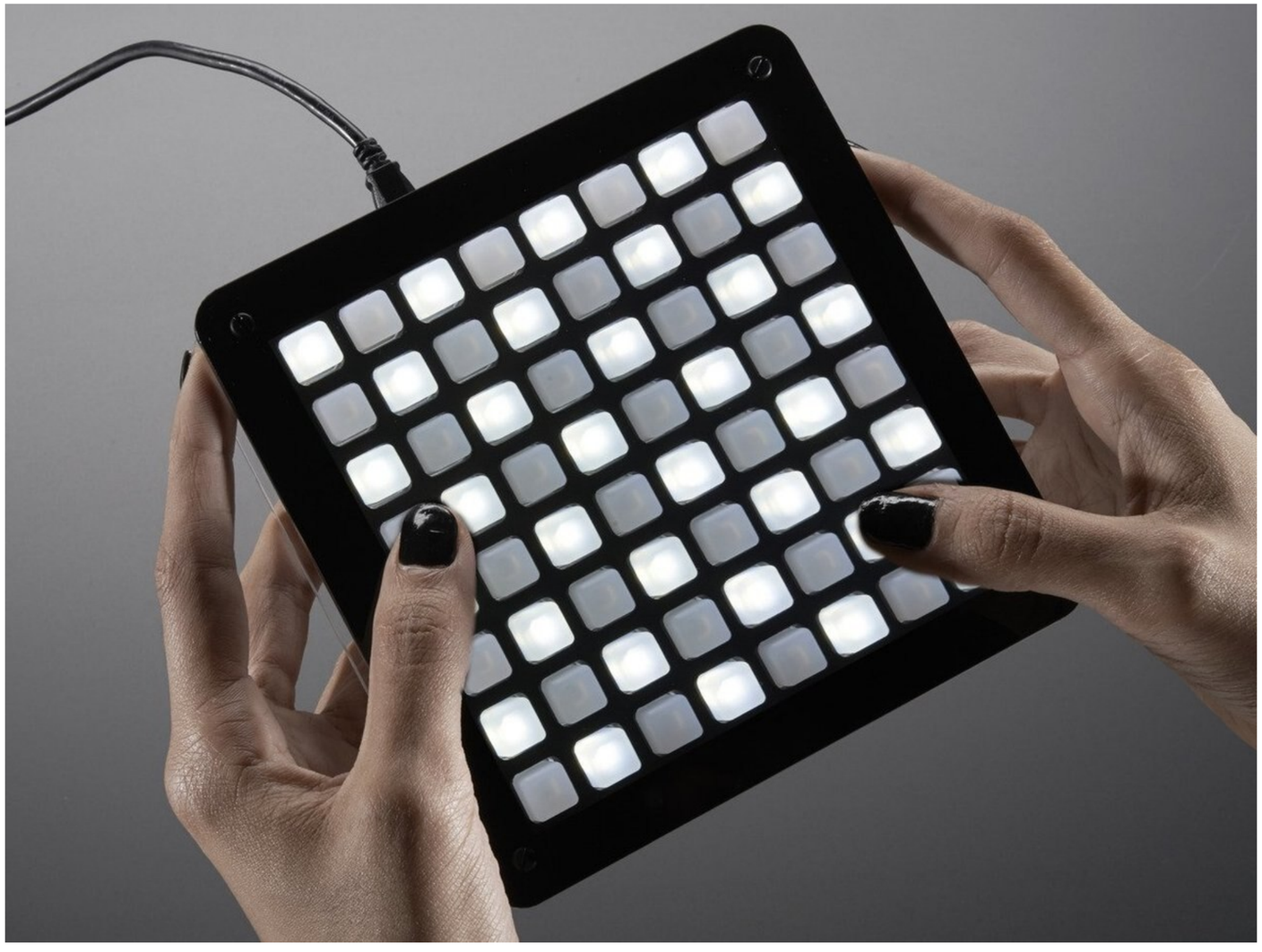


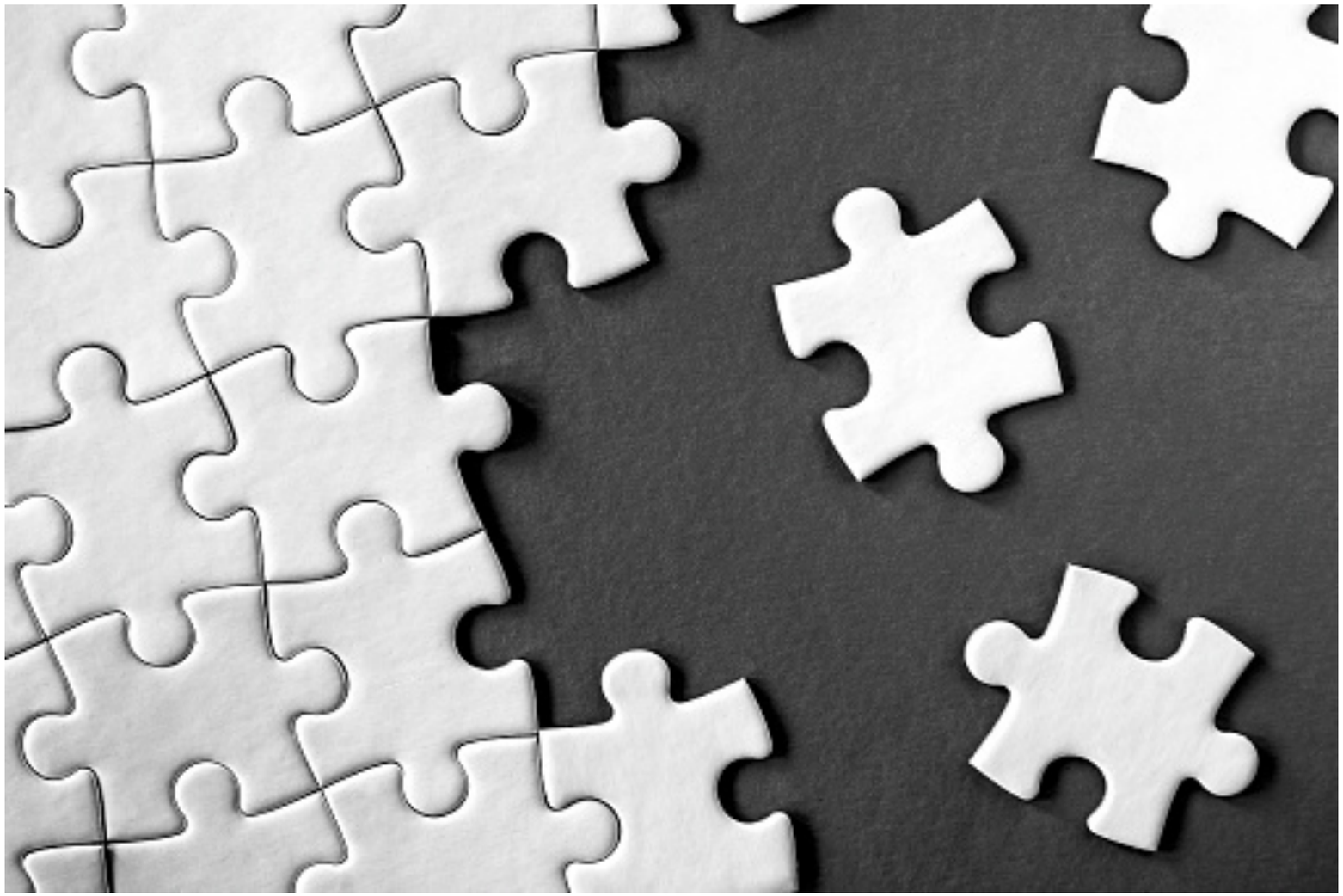
# Tareas accidentales

# Las tareas esenciales son difícilmente optimizables

# Complejidad

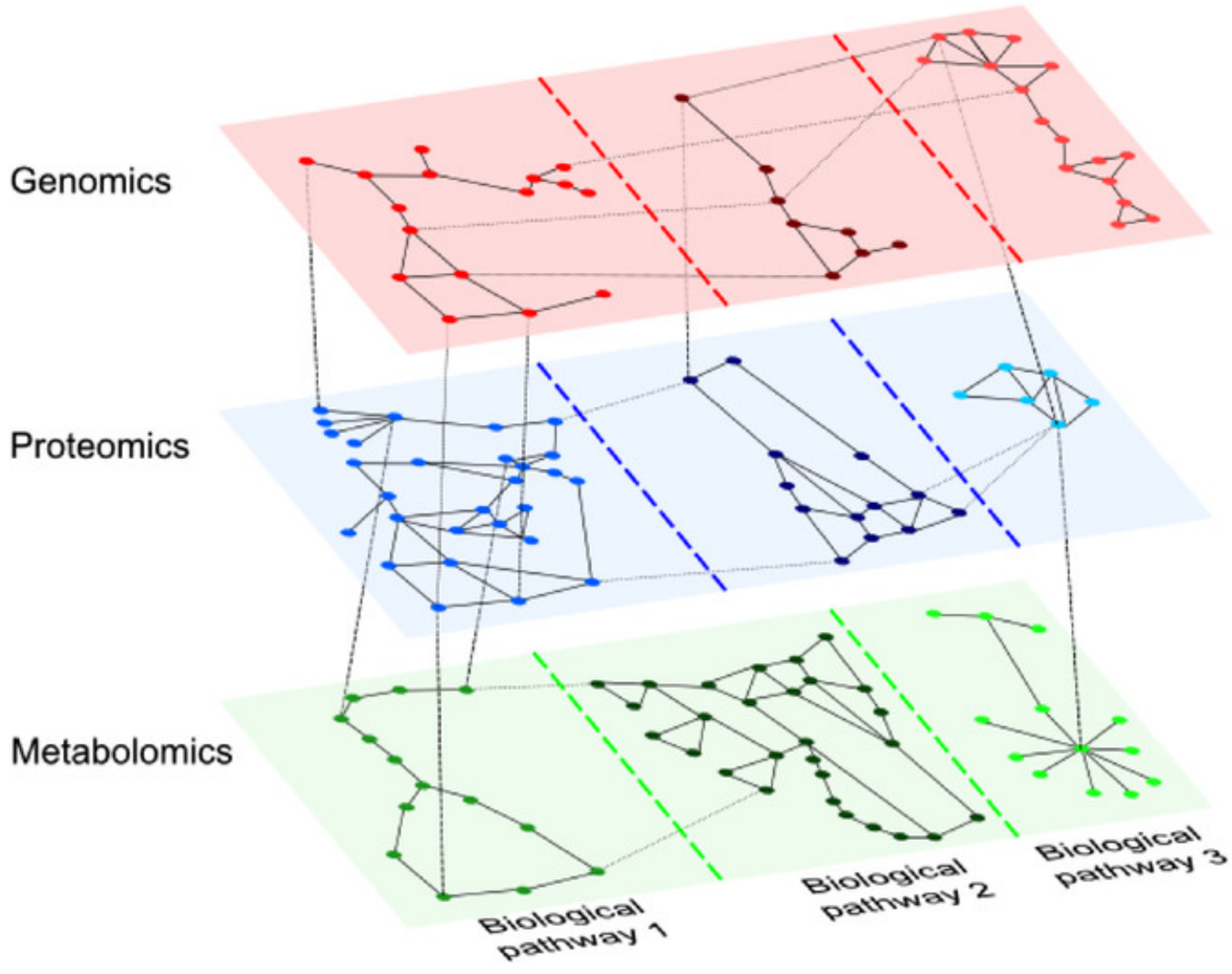


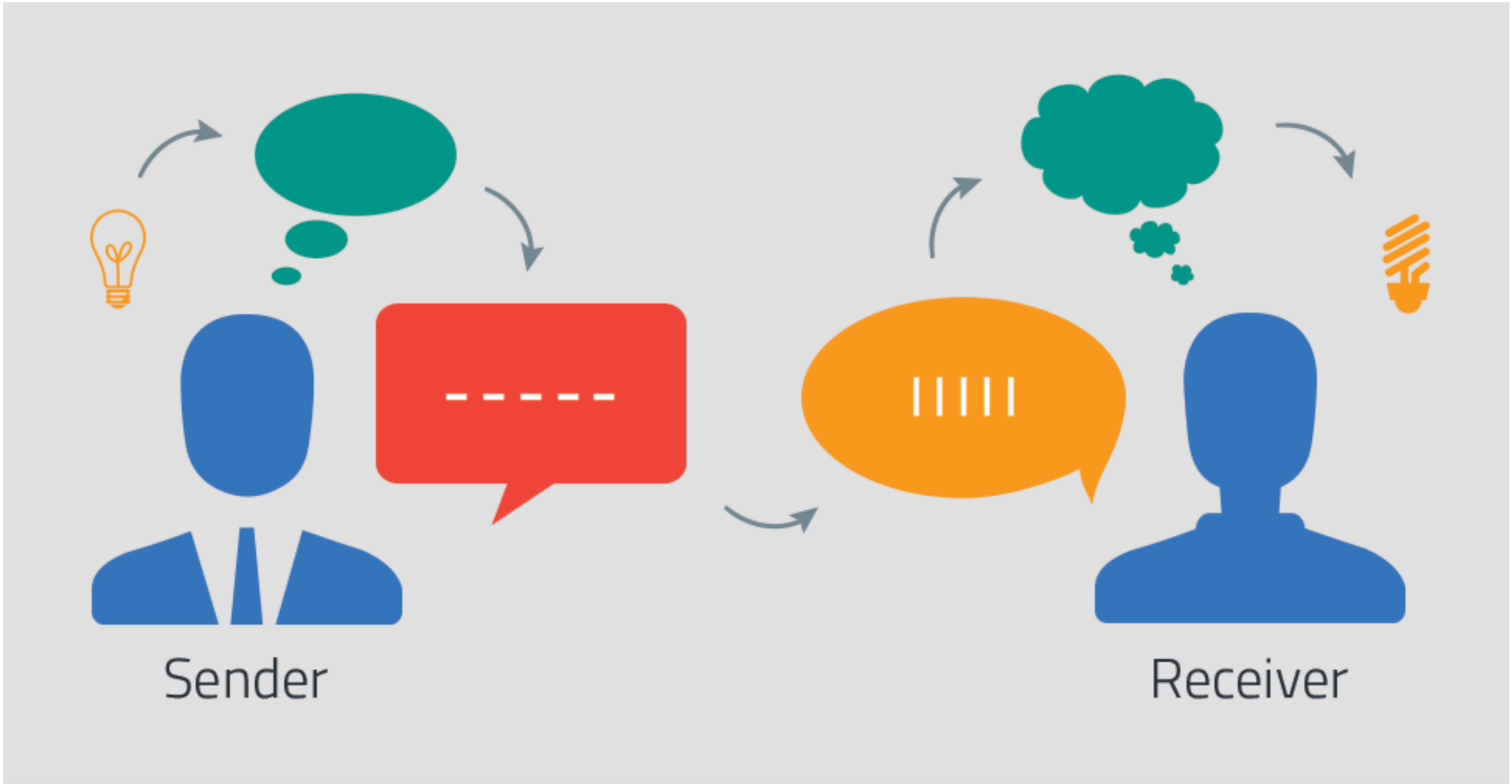
# Conformidad



# Cambiabilidad

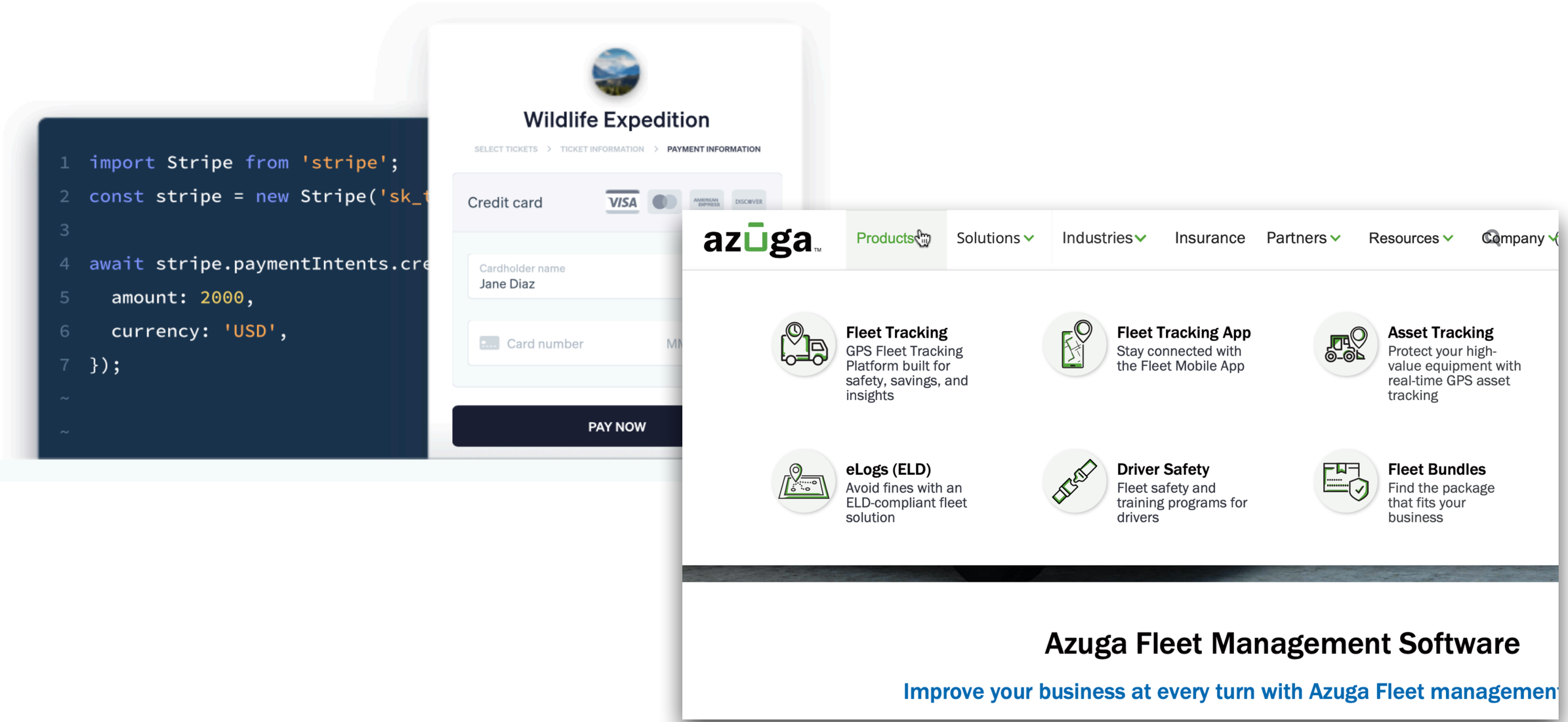
# Invisibilidad



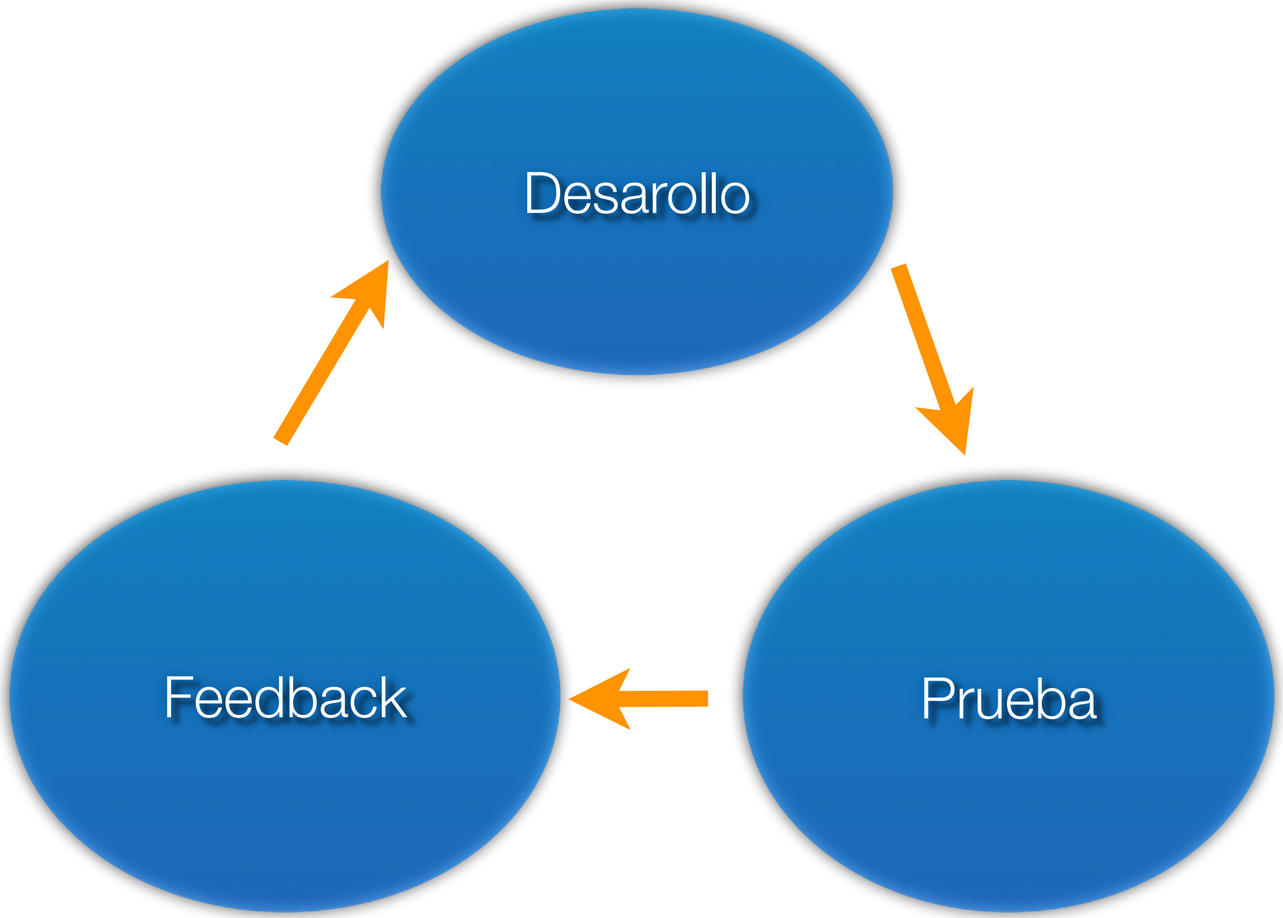

Sender
Receiver

# Algunas posibles optimizaciones

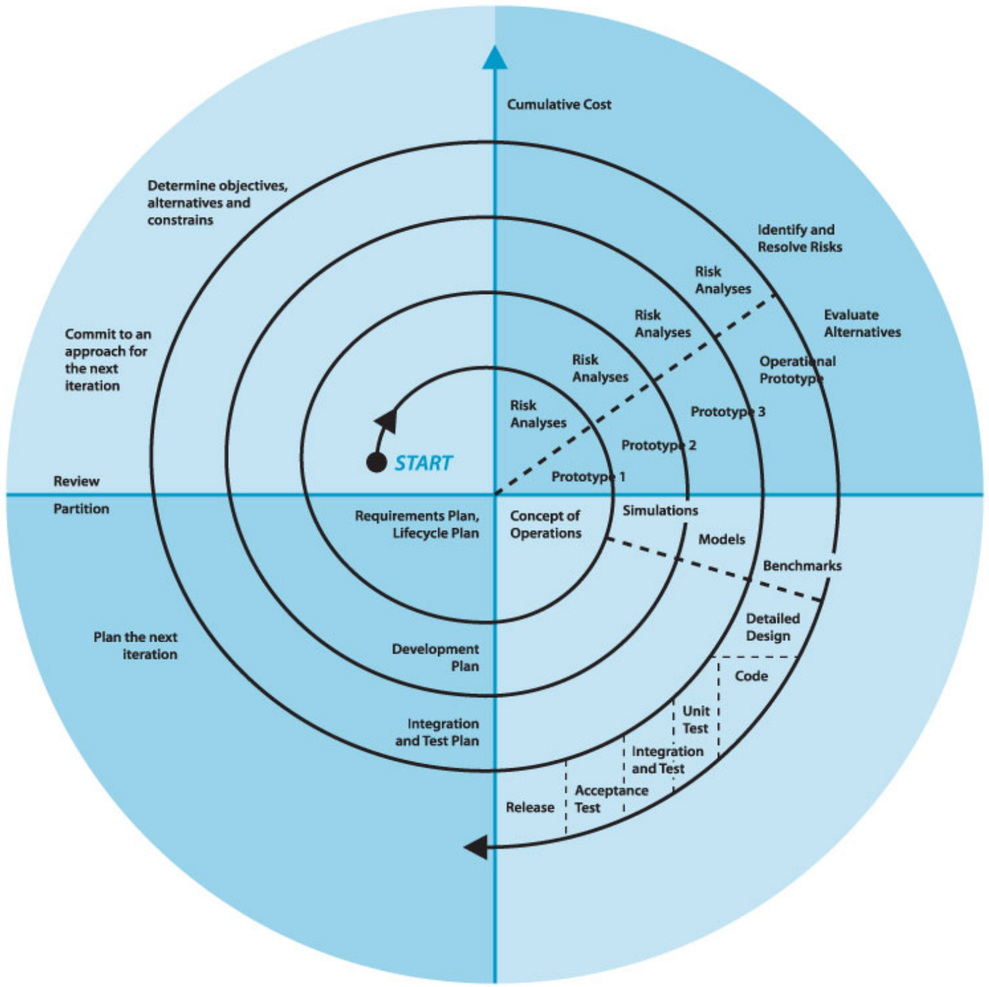
# Comprar siempre que sea posible



# Prototipado rápido y refinamiento de los requisitos



# El software debe crecer orgánicamente, no ser construido



# Desarrollo de software

Veremos en este apartado las características propias del software y de la forma de desarrollarlo q
comparar con ingenierías tradicionales como la construcción.

## Software

El software es un invento muy reciente de la humanidad. Fue a mitad del siglo XX cuando se empe
computadores electrónicos programables en organismos oficiales y grandes empresas. Y los prim