Software Redesign

Contents

Contents

Introduction

SOFTWARE REDESIGN

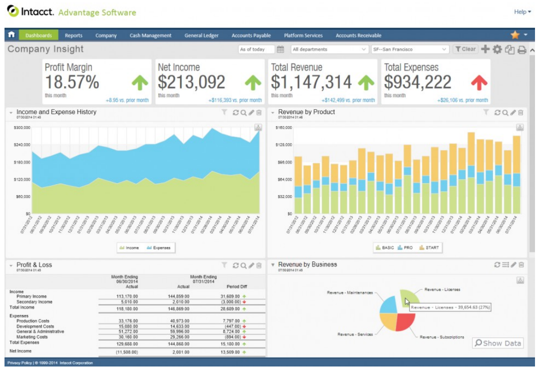
Software is Eating the World

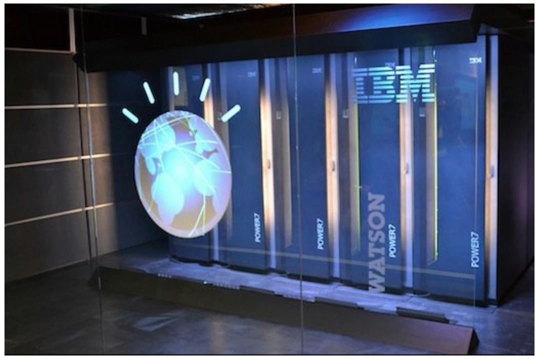
- “every company needs to be a software company”
- software eating a traditional business

Marc Andreessen, The Wall Street Journal on August 20, 2011

Software is Eating the World

- "every company needs to be a software company"
- industries primarily exists in physical world





Marc Andreessen, The Wall Street Journal on August 20, 2011

Software is Eating the World

“No one should expect building a new high-growth, software-powered company in an established industry to be easy. It’s **brutally difficult**.”

Marc Andreessen, The Wall Street Journal on August 20, 2011

Successful Software Development

- deliver software product:

high quality

+

on time

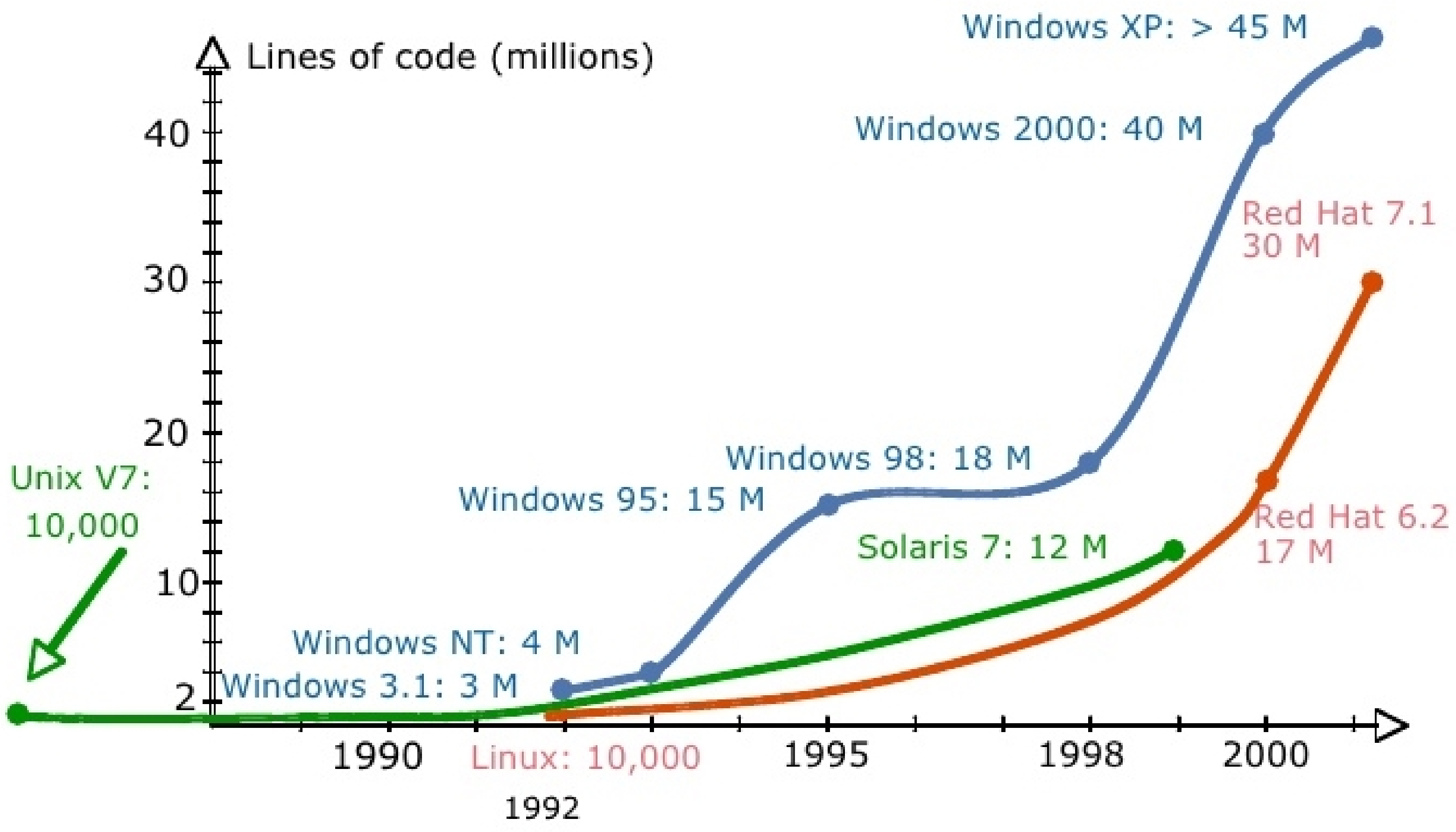
+

on budget

+

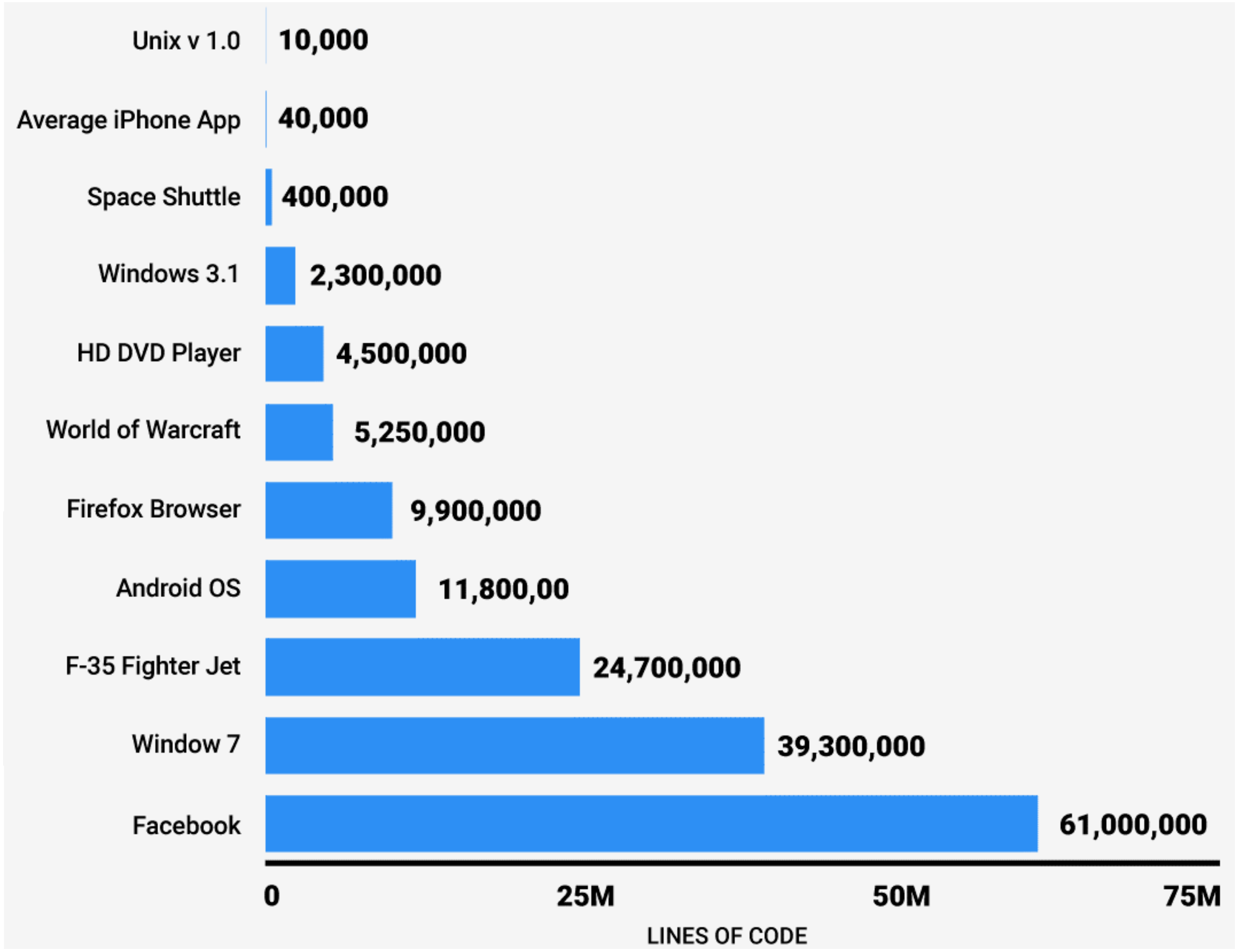
satisfies stakeholders' expectation or goal

Software Size



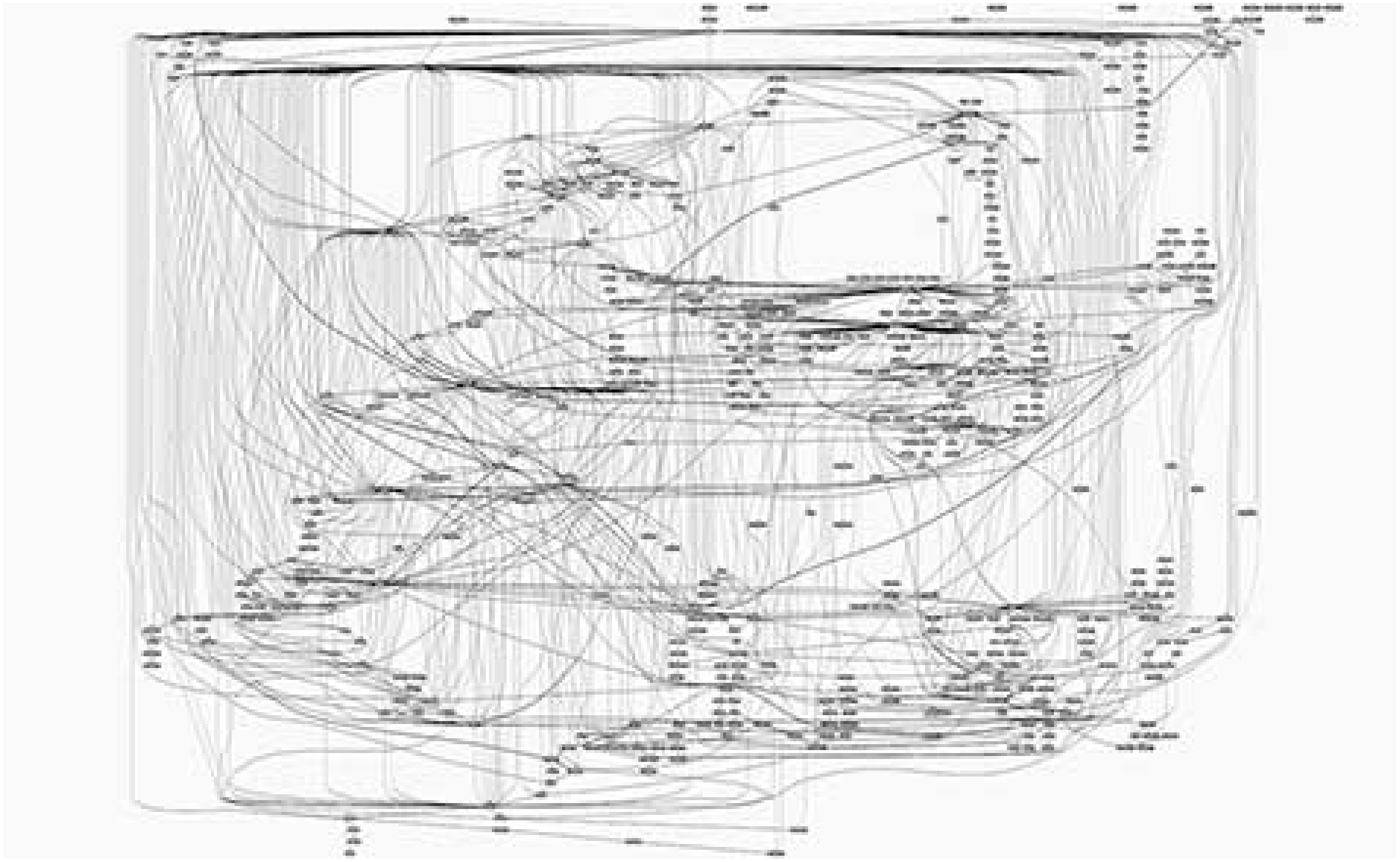
Michele Lanza, Software Evolution, 2008

Software Size

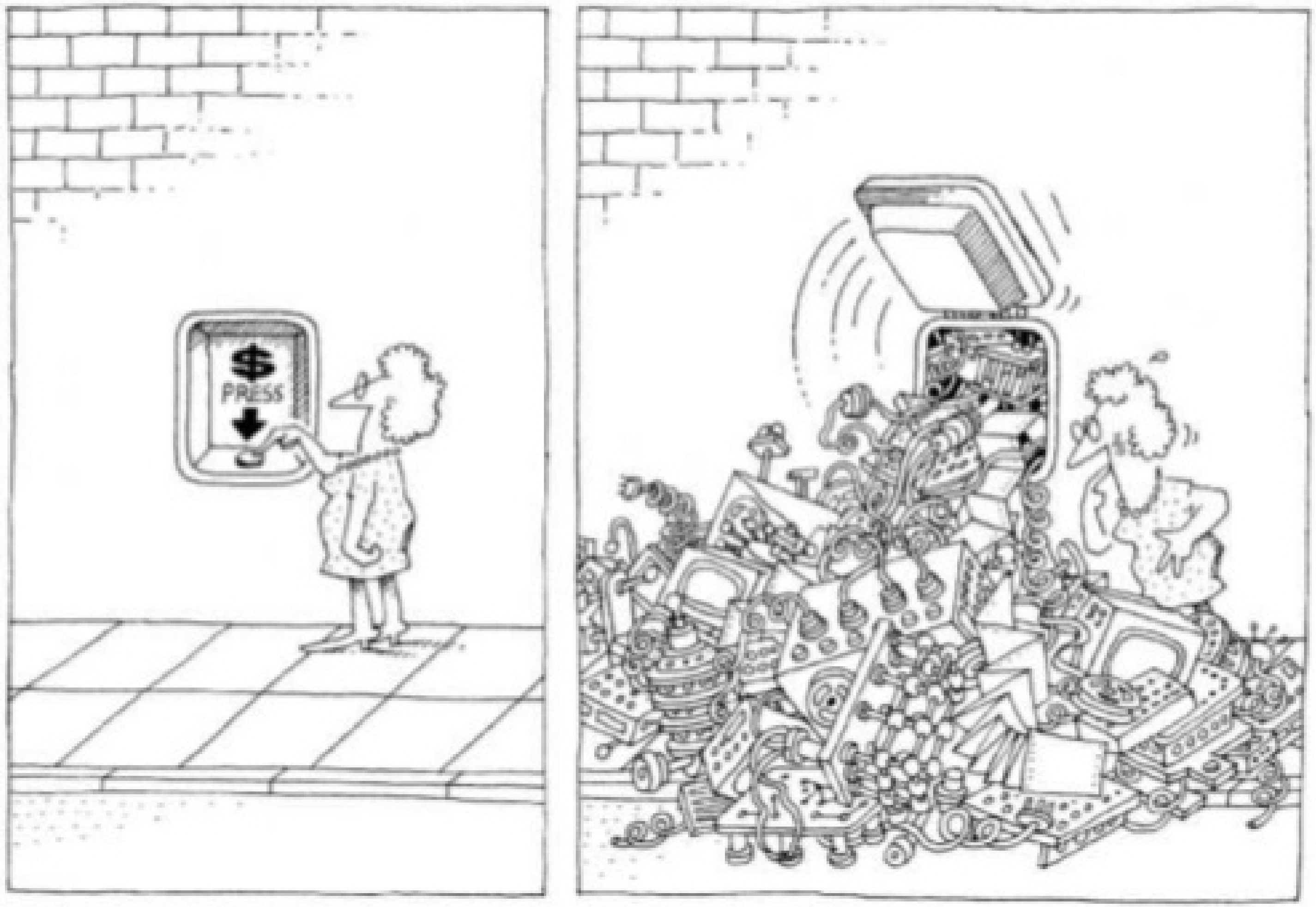


Dragan Radovanovic, Business Insider, 2016

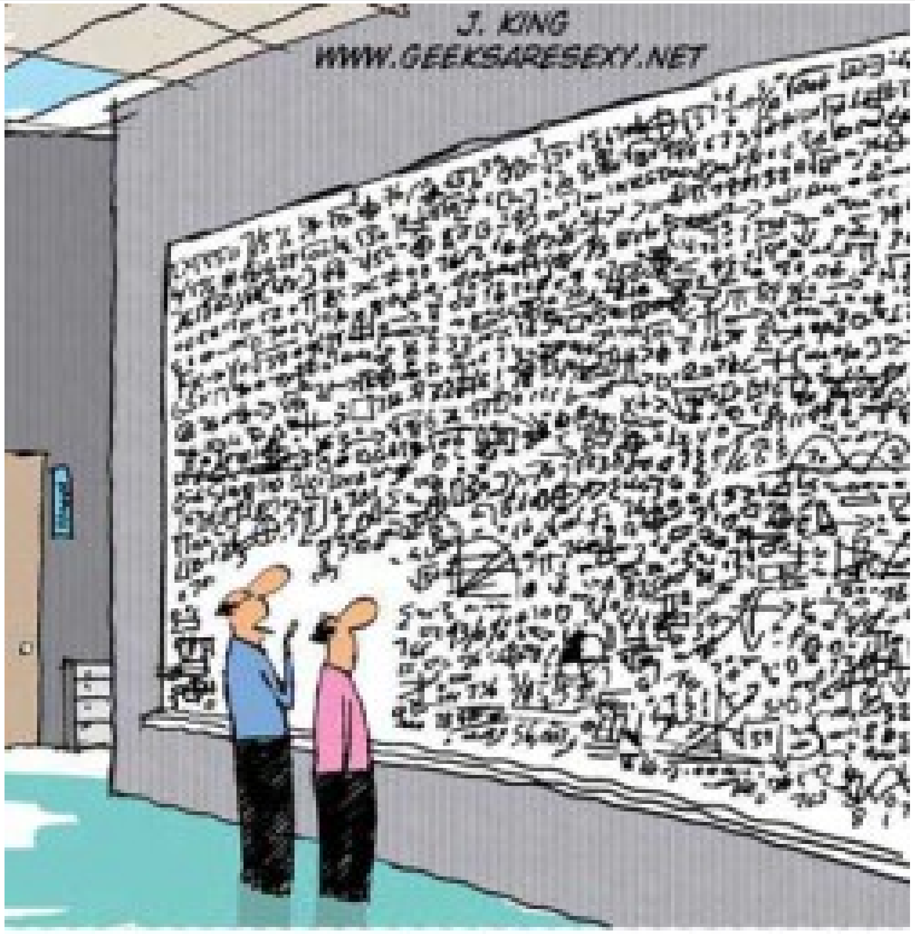
Software Complexity



Software Complexity

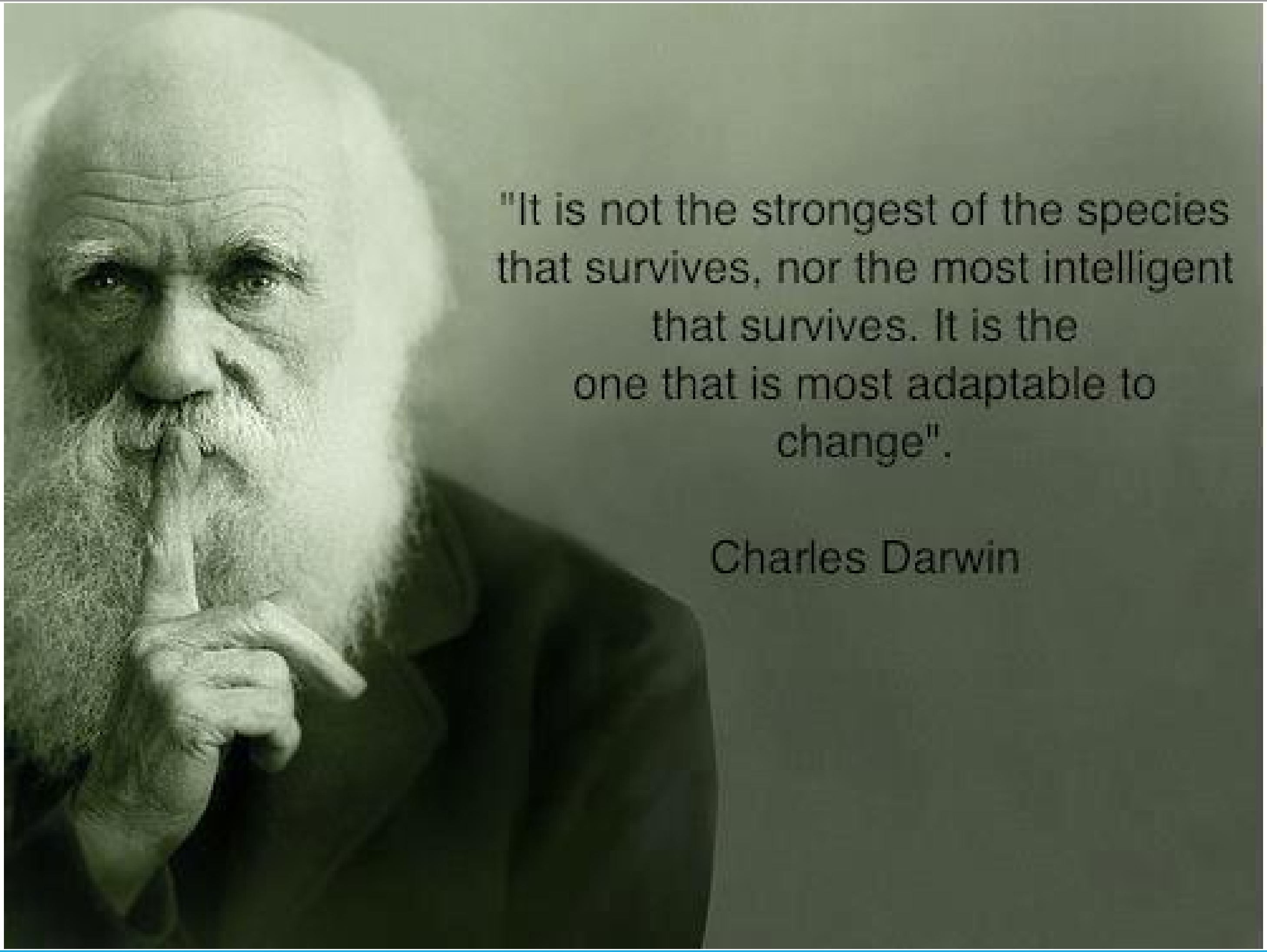


Software Complexity



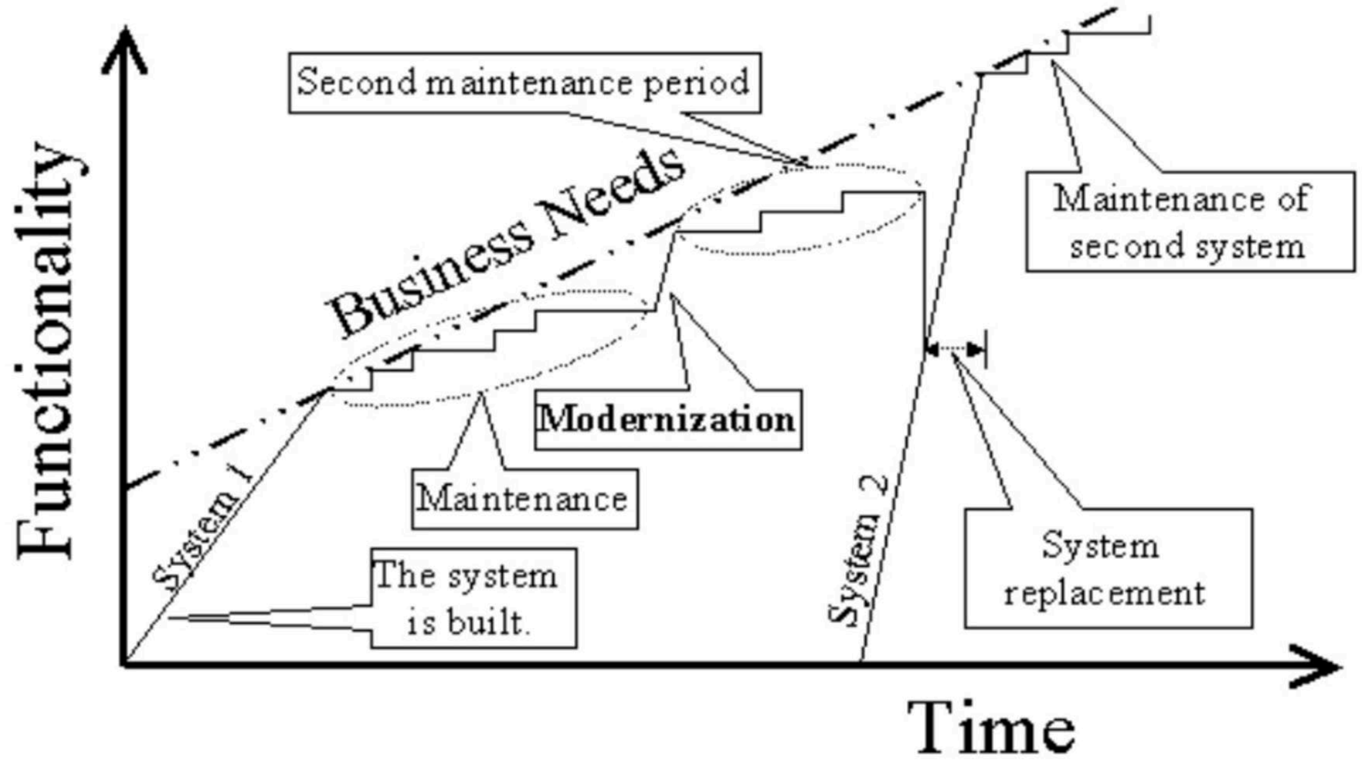
"...And that, in simple terms, is what's wrong with your software design."

Software Evolution


"It is not the strongest of the species
that survives, nor the most intelligent
that survives. It is the
one that is most adaptable to
change".
Charles Darwin

Software Evolution

- making changes to software over time
- comprises:
  - development & maintenance & reengineering



Analysis of Software Evolution

- "Nevertheless, the industrial track record raises the question, why, despite so many advances, …
- satisfactory functionality, performance and quality is only achieved over a lengthy **evolutionary process**
- software maintenance **never ceases** until a system is scrapped
- software is still generally regarded as the **weakest link** in the development of computer based systems."

-Lehman et. Al, 1997

Lehman's Laws of Software Evolution

1. Continuing Change (1974)
   - E-type program must be continually adapted, else they become progressively less satisfactory
2. Increasing Complexity (1974)
   - As an E-type program evolves, its complexity increases unless work is done to maintain or reduce it

6. Continuing Growth (1991)
   - the functional content of an E-type system must be continually increased to maintain user satisfaction over its lifetime
7. Declining Quality (1996)
   - the quality of an E-type system will appear to be declining unless it is rigorously maintained and adapted to operational environment changes

* E-type systems:

"monolithic systems produced by a team within an organization that solve a real world problem and have human users."

Analysis of Software Evolution

- objective
  - investigate the evolution of a software system to identify potential shortcomings in its architecture or logical structure
- structural shortcomings can be subjected to reengineering or restructuring
  - prerequisite: **Reverse Engineering**

-Lehman et. Al, 1997

Legacy Systems

- what is legacy ?
  - old
  - unstable
  - unsupported
  - not maintained
  - supplanted
  - monolithic
  - complex
  - obsolete
  - bad

Legacy Systems

- legacy system is an older software system that remain **vital** to an organization
  - have a long lifetime
  - still in use
  - developed many years ago
  - using obsolete technologies
  - still business critical

Legacy System - Common Issues

- no documentation
- no overarching design
- lost knowledge
- hidden knowledge
- unused functionality
- fragility
- coupling & cohesion
- zombie technologies
- politics

Legacy Systems Evolution

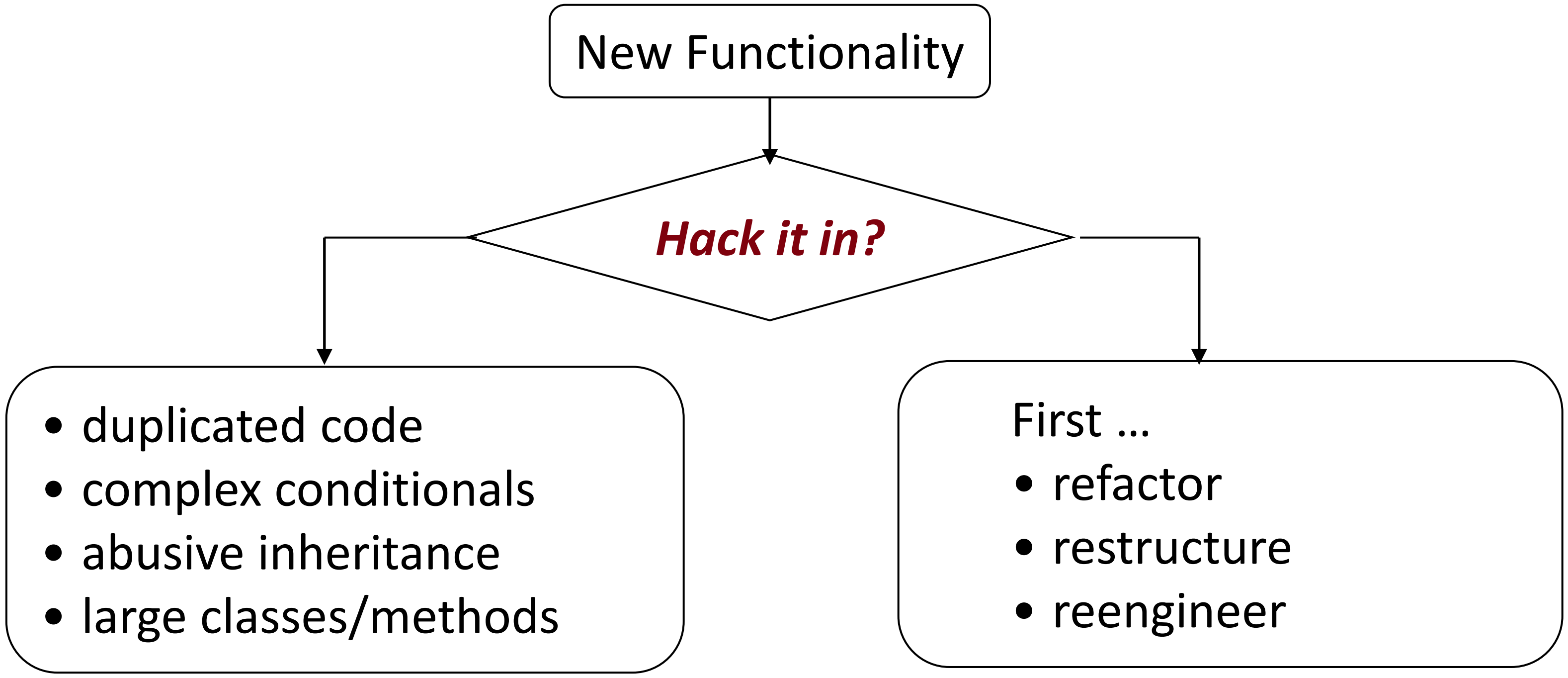
- it is expensive and risky to replace legacy system
  - no documentation
    - rarely have complete specification
    - have undergone major changes without documentation
  - hidden knowledge
    - may embed business rules that are not formally documented
  - business processes are reliant on
  - new software is risky and may fail

Legacy Systems Maintenance

- it is expensive to change(maintain) legacy system
  - no documentation
    - documentation is missing or out-of-date
  - no overarching design
    - Initial good design may not be maintained
    - different part implemented by different team
    - no consistent design / programming style
  - may use obsolete programming language
  - structure may be corrupted by many years of maintenance

Legacy Systems Maintenance

- new or changing requirements will gradually degrade original design … unless extra development effort is spent to adapt the structure



Take a loan on your software
⇒ pay back via reengineering

Investment for the future
⇒ paid back during maintenance