Software and Security Engineering

Lecture 7

Anil Madhavapeddy

avsm2@cam.ac.uk

With many thanks to Ross Anderson and Alastair R. Beresford

Warm up: What mistakes were made in the LAS system?

- Specification
- Project management
- Operational

Specification mistakes

- LAS ignored advice on cost and timescale
- Procurers insufficiently qualified and experienced
- No systems view
- Specification was inflexible but incomplete: it was drawn up without adequate consultation with staff
- Attempt to change organisation through technical system
- Ignored established work practices and staff skills

Project management mistakes

- Confusion over who was managing it all
- Poor change control, no independent QA, suppliers misled on progress
- Inadequate software development tools
- Ditto data comms, with effects not foreseen
- Poor interface for ambulance crews
- Poor control room interface

Operational mistakes

- System went live with known serious faults
  - slow response times
  - workstation lockup
  - loss of voice comms
- Software not tested under realistic loads or as an integrated system
- Inadequate staff training
- No effective back-up system in place

NHS National Programme for IT

Idea: computerise and centralise all record keeping for every visit to every NHS establishment

- Like LAS, an attempt to centralise power and change working practices
- Earlier failed attempt in the 1990s
- The February 2002 Blair meeting
- Five LSPs plus national contracts: £12bn
- Most systems years late or never worked
- By 2012 & coalition government: NPfIT ‘abolished’

Universal Credit: fix poverty trap

Idea: Hundreds of welfare benefits which means there is often little incentive to get a job.

- Initial plan was to go live in October 2013
- A significant problem: big systems take seven years not three; doesn’t align with political cycle
- Complexity was huge, e.g. depended on real-time feed of tax data from HMRC, which in turn depended on firms
- See https://cpag.org.uk/news/digital-universal-credit-system-breaches-principles-law-and-stops-claimants-accessing-support in 2023, a decade on

NAO: poor value for money, not paying 1 in 5 on time

https://www.youtube.com/watch?v=qE2fpNSrrpc

Smart meters: more centralisation

Idea: expose consumers to market prices, get peak demand shaving, make use salient

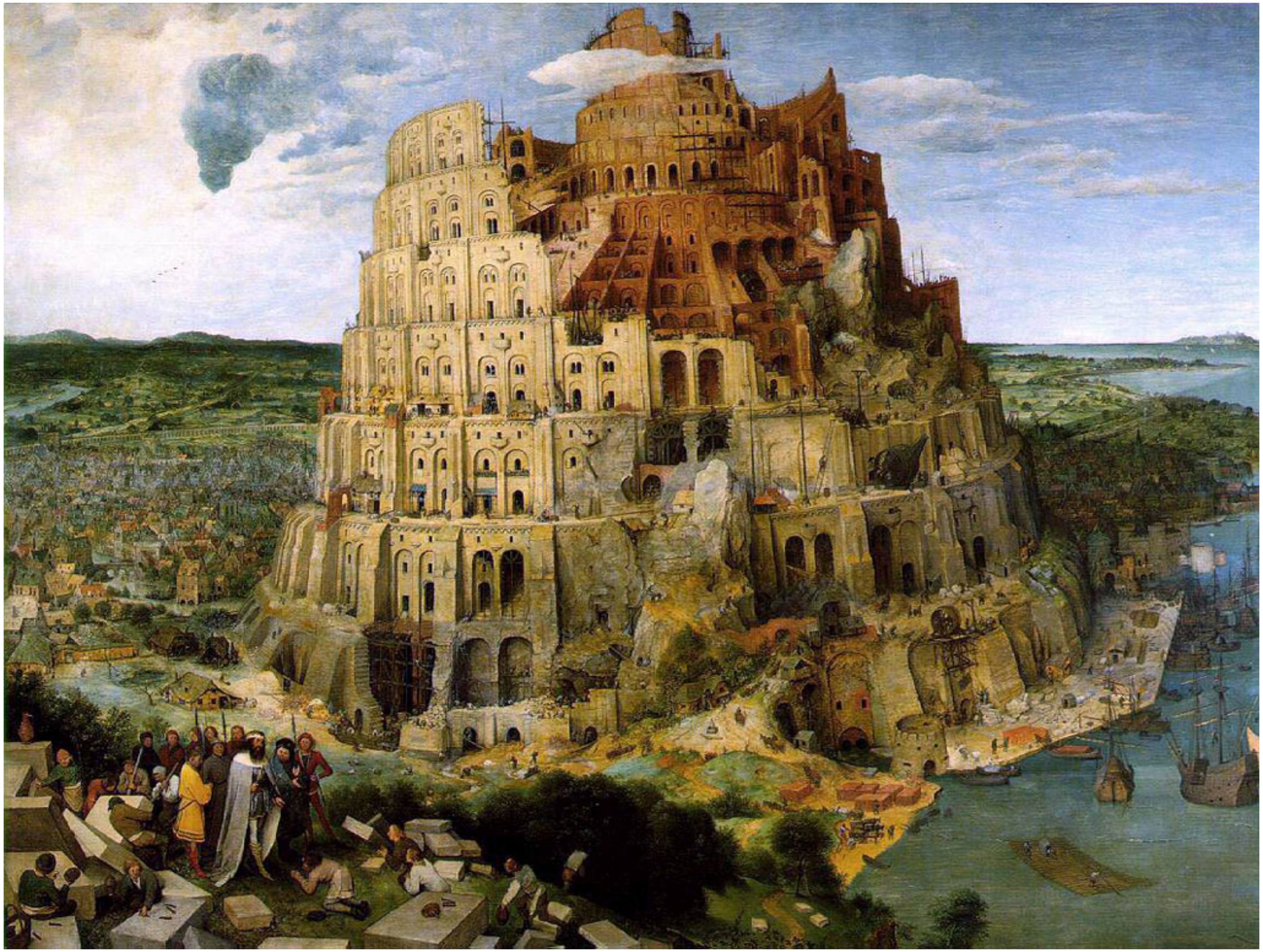
- 2009: EU Electricity Directive for 80% by 2020
- 2009: Labour £10bn centralised project to save the planet and help fix supply crunch in 2017
- 2010: Experts said we just can't change 47m meters in 6 years. So excluded from spec
- Coalition government: wanted deployment by 2015 election! Planned to build central system Mar–Sep 2013 (then: Sep 2014 …)
- Spec still fluid, tech getting obsolete, despair …
- 2023: https://publications.parliament.uk/pa/cm5803/cmselect/cmpubacc/1332/summary.html

Software engineering is about managing complexity at many levels

- Bugs arise at micro level in challenging components
- As programs get bigger, interactions between components grow at $O(n^2)$ or even $O(2^n)$
- The ‘system’ isn’t just the code: complex socio-technical interactions mean we can’t predict reactions to new functionality

Most failures of really large systems are due to wrong, changing, or contested requirements

Project failure, circa 1500 BCE



On contriving machinery

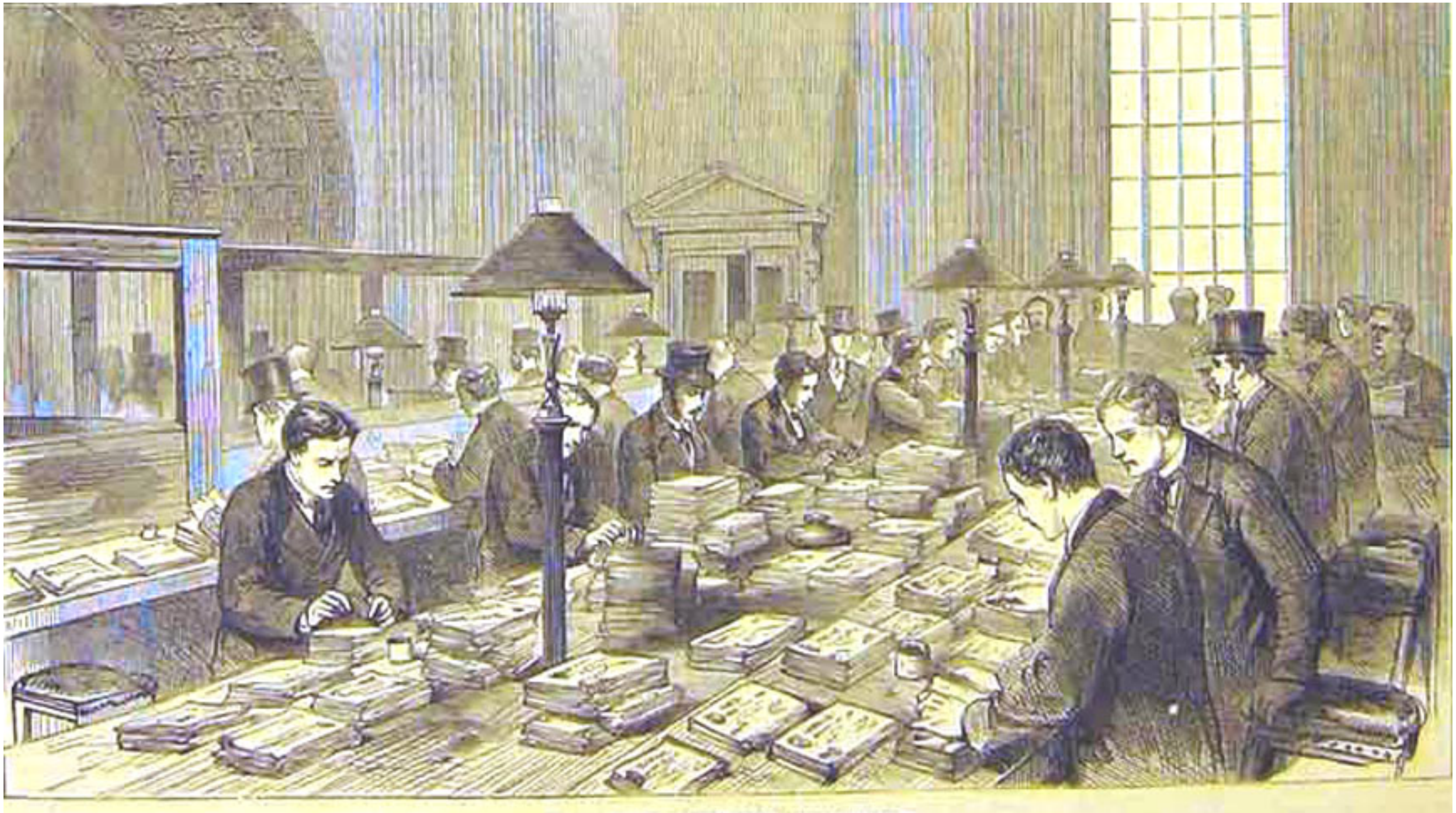
"It can never be too strongly impressed upon the minds of those who are devising new machines, that to make the most perfect drawings of every part tends essentially both to the success of the trial, and to economy in arriving at the result"

Charles Babbage

[The Analytical Engine] might act upon other things besides number, were objects found whose mutual fundamental relations could be expressed by those of the abstract science of operations, and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the engine...Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent.

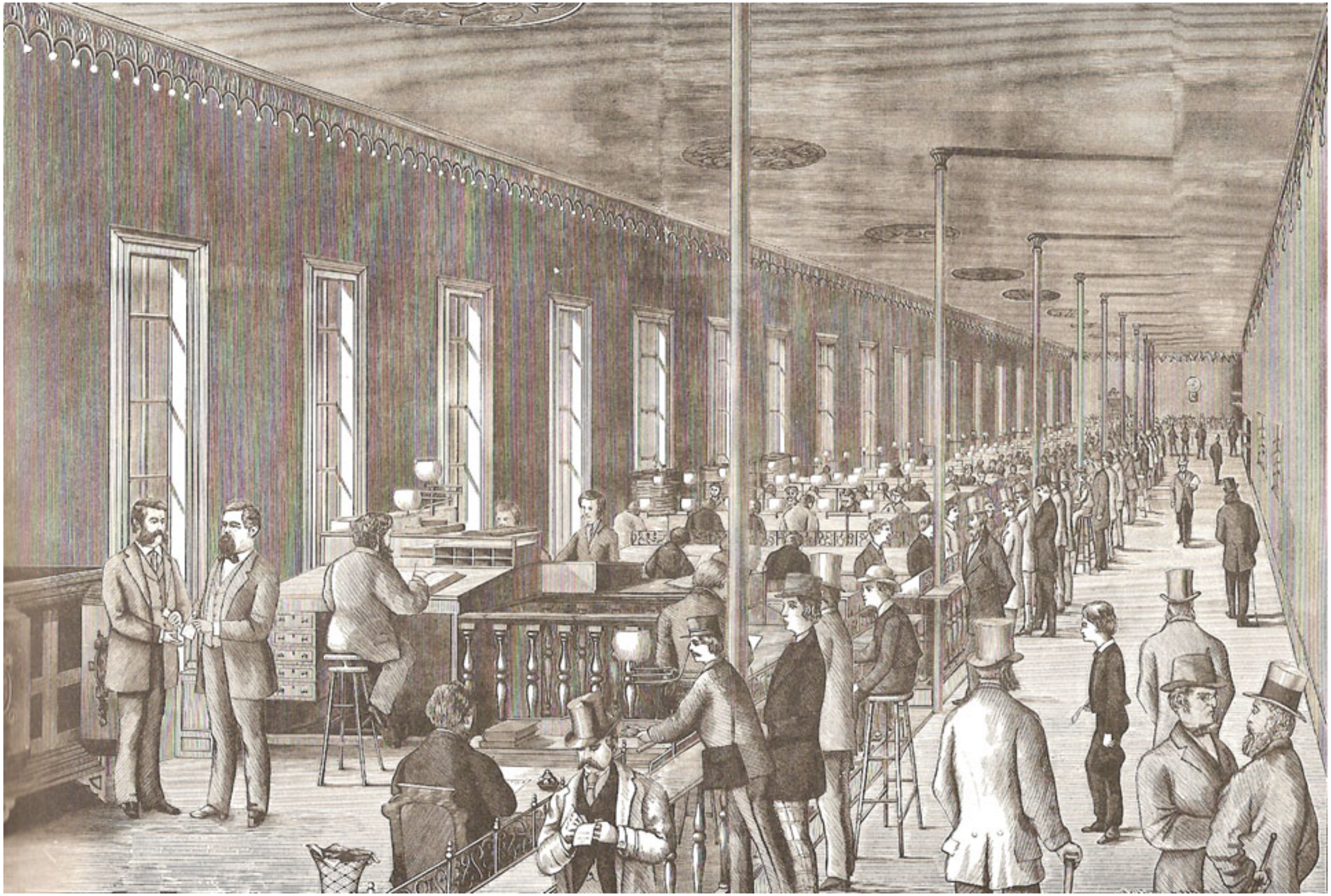
Ada Lovelace (1842)

Bank of England, 1870



THE ACCOUNTANTS' BANK NOTE OFFICE

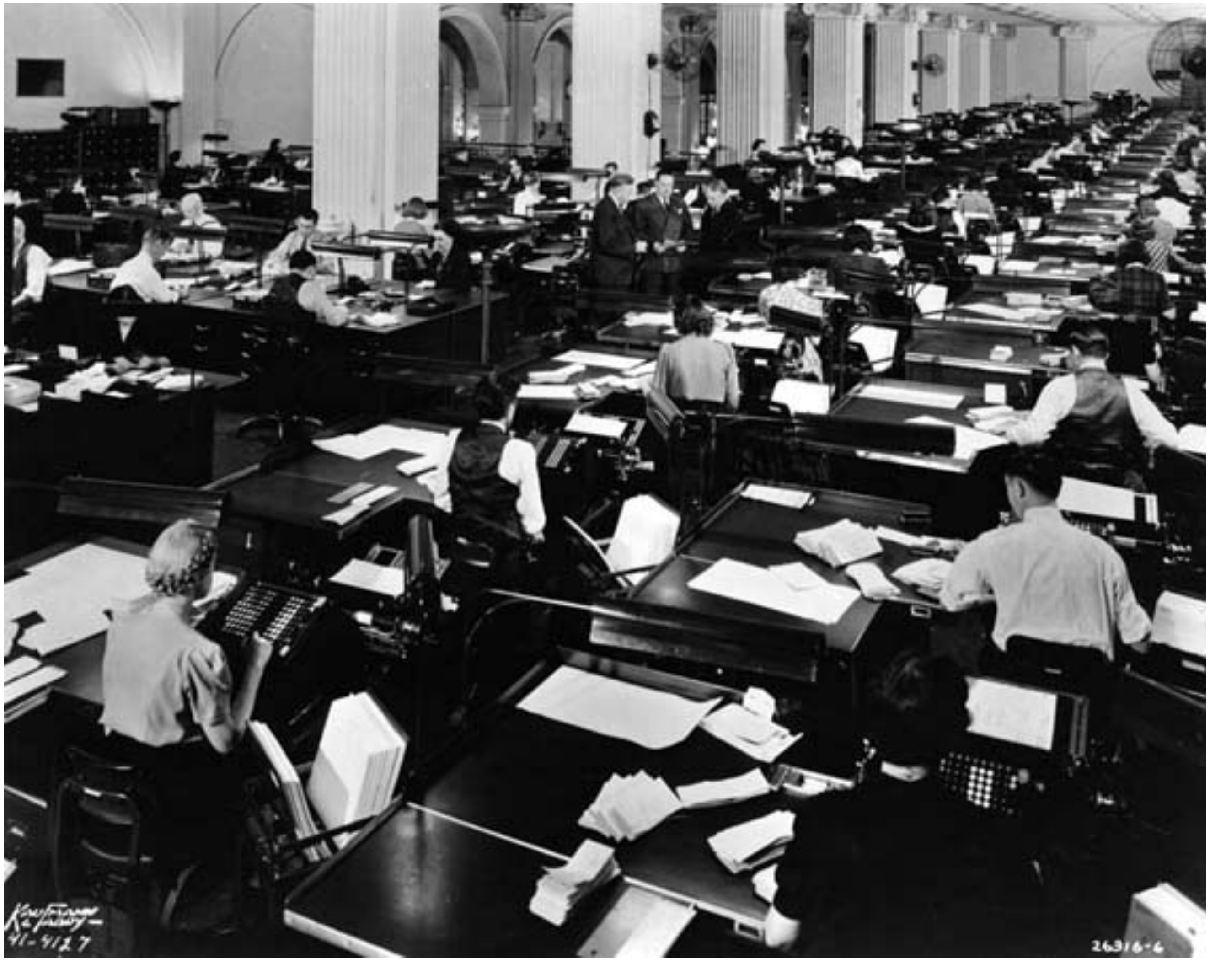
Dun, Barlow & Co, 1876



Sears, Roebuck and Company, 1906

- Continental-scale mail order meant specialization
- Big departments for single bookkeeping functions
- Beginnings of automation

First National Bank of Chicago, 1940



The software crisis, 1960s

- Large, powerful mainframes made complex systems possible
- People started asking why project overruns and failures were so much more common than in mechanical engineering, shipbuilding, etc.
- The term *software engineering* coined in 1968
- The hope was that we could things under control by using disciplines such as project planning, documentation and testing

Those things which make writing software fun also make it complex

- Joy of solving puzzles and building things from interlocking parts
- Stimulation of a non-repeating task with continuous learning
- Pleasure of working with a tractable medium, ‘pure thought stuff’
- Complete flexibility – you can base the output on the inputs in any way you can imagine
- Satisfaction of making stuff that’s useful to others

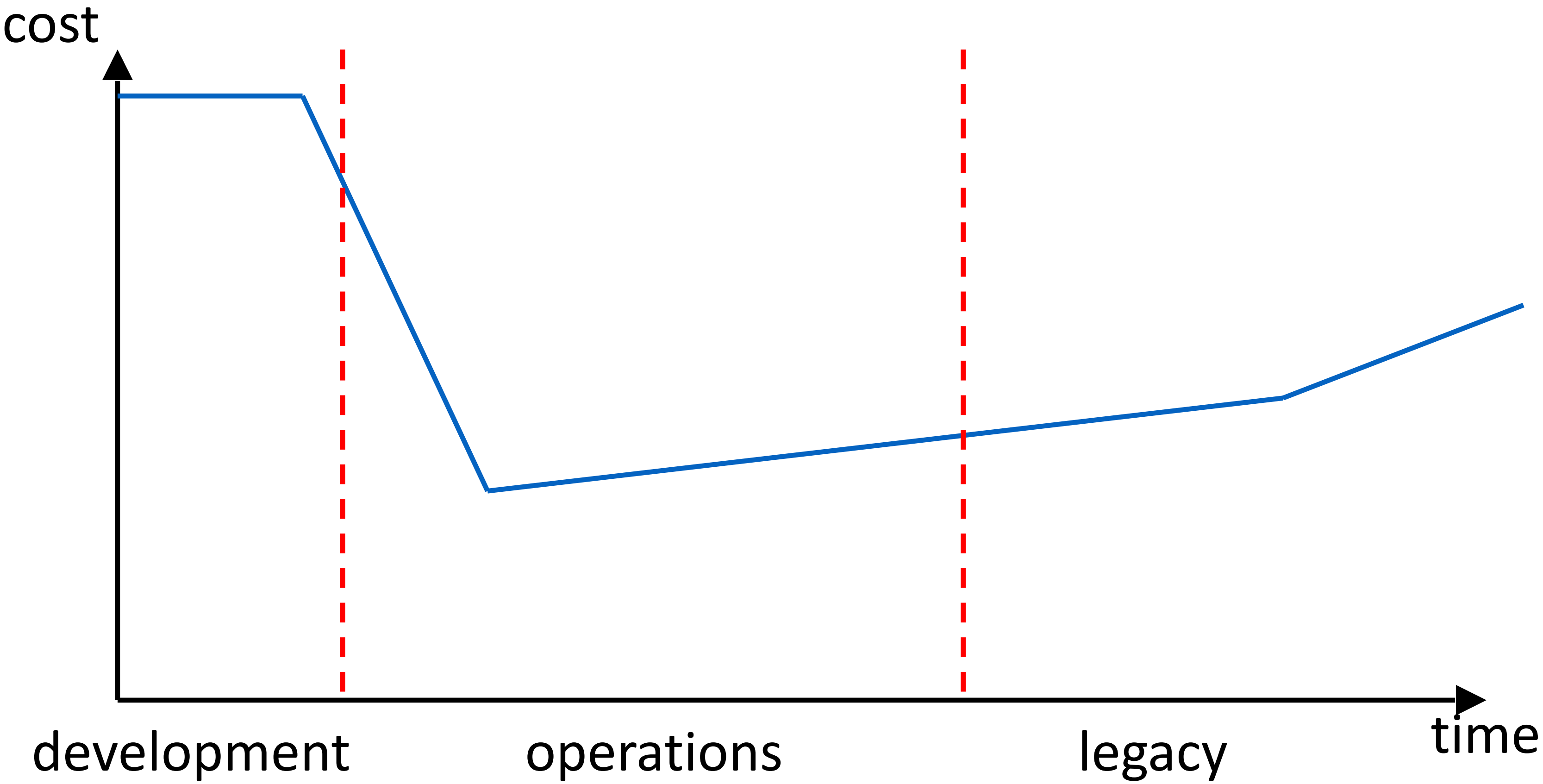
How is software different?

- Large computer systems become **qualitatively more complex**, unlike big ships or long bridges
- The tractability of software leads customers to **demand flexibility** and **frequent changes**
- This makes systems **more complex to use over time** as features accumulate, and interactions have odd effects
- The structure can be **hard to visualise** or model
- The hard slog of debugging and testing piles up at the end, when the excitement's past, the budget's spent and the **deadline's looming**

Software economics can be nasty

- Consumers buy on sticker price
- Businesses buy based on total cost of ownership
- Vendors use lock-in tactics
- Complex outsourcing

Cost of software: development 10%, maintenance 90%



Measuring cost of code is hard

First IBM measures (1960s)

- 1.5 KLOC per developer-year (operating system)
- 5 KLOC per developer-year (compiler)
- 10 KLOC per developer-year (app)

AT&T measures

- 0.6 KLOC per developer-year (compiler)
- 2.2 KLOC per developer-year (switch)

KLOC is a poor measure

1.

```
//Print out hello
for (int i = 0; i < 4; i++) {
  System.out.println(“Hello, world”);
}
```

2.

```
for (int i = 0; i < 4; i++) { System.out.println(“Hello, world”);}
```

3.

```
System.out.println(“Hello, world”);
System.out.println(“Hello, world”);
System.out.println(“Hello, world”);
System.out.println(“Hello, world”);
```

Alternatives:

- Halstead (entropy of operators/operands)
- McCabe (graph entropy of control structures)
- Function point analysis

Early lessons: productivity varies, use a high-level language

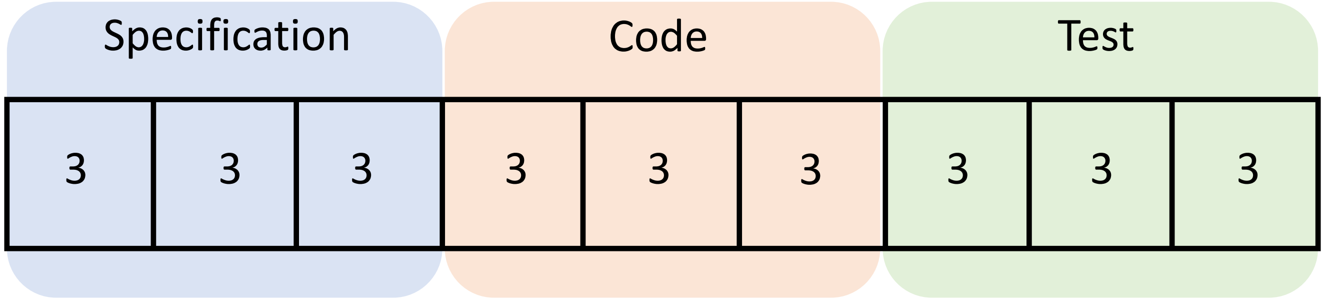
- Huge variations in productivity between individuals
- The main systematic gains come from using an appropriate high-level language since they reduce accidental complexity; programmer focuses on intrinsic complexity
- Get the specification right: it more than pays for itself by reducing the time spent on coding and testing

Barry Boehm surveyed relative costs of software development (1975)

	Spec	Code	Test
C3I	46%	20%	34%
Space	34%	20%	46%
Scientific	44%	26%	30%
Business	44%	28%	28%

- All stages of software development require good tools

Mythical Man-Month: *"adding manpower to a late project makes it later"*

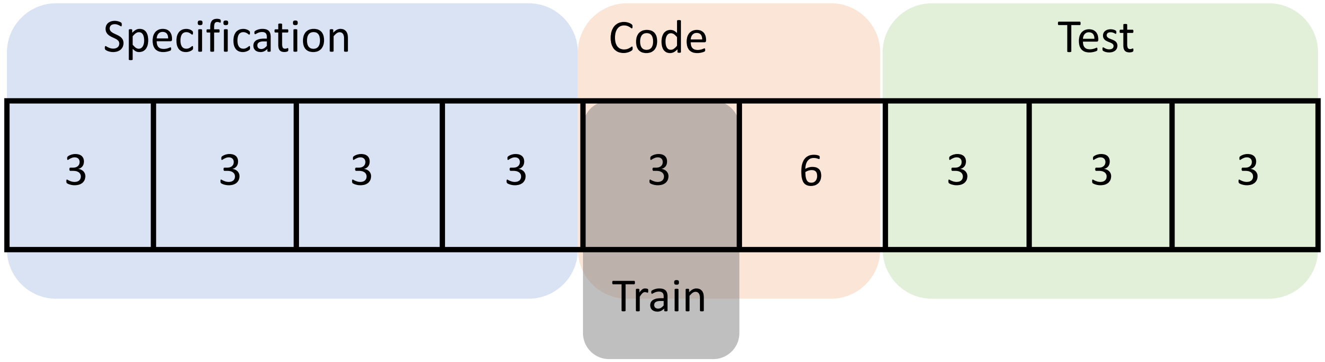


Specification			Code			Test		
3	3	3	3	3	3	3	3	3

Example project with 3 developers and 9 months. Initial estimate is 9 person-months each for spec, code and test.

- But spec ends up taking 12 PMs. What do you do?

Mythical Man-Month: *"adding manpower to a late project makes it later"*



We try to catch up:

- Train 3 more developers in the first month, then use all 6 developers in the next month
- But: work of 3 developers in 2 months can't be done by 6 developers in 1 – interaction costs maybe O(n^2)

Time to first shipment is cube root of developer-months (Boehm, 1984)

$$T = 2.5\sqrt[3]{\mathrm{d}}$$

where T is time to first shipment and d is developer months

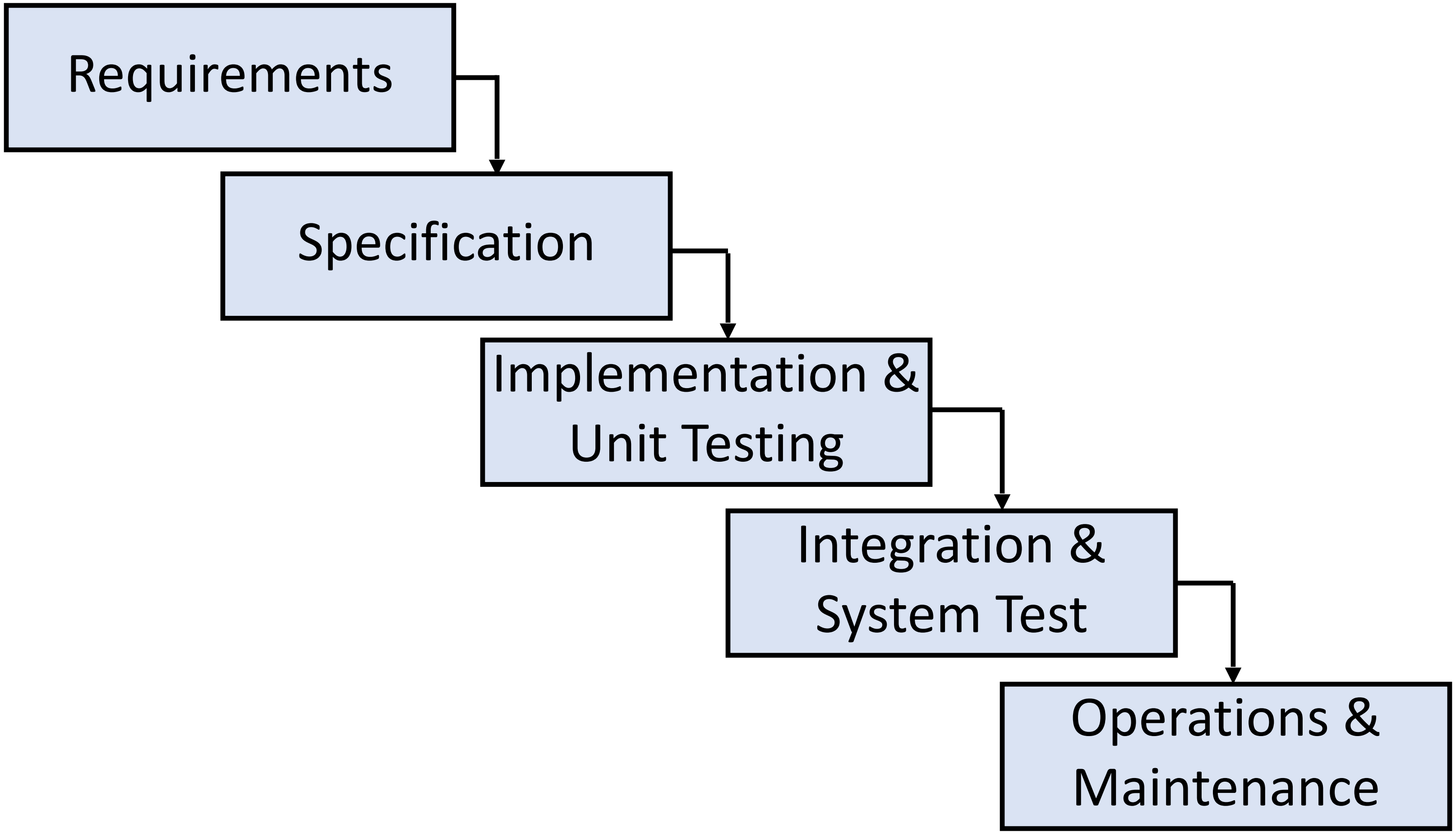
- With more time, costs rise slowly
- With less time, costs rise sharply
- Hardly any projects succeed at $\frac{3}{4}T$
- Some projects still fail

The Software Tar Pit

Take a structured, modular approach

- Only practical way forward is *modularisation*
- Divide a complex system into small components
- Define clear APIs between them
- Lots of methodologies based on this idea:
  - SSDM
  - Jackson
  - Yourdon,
  - UML,
  - …

The Waterfall Model (1970)



Requirements

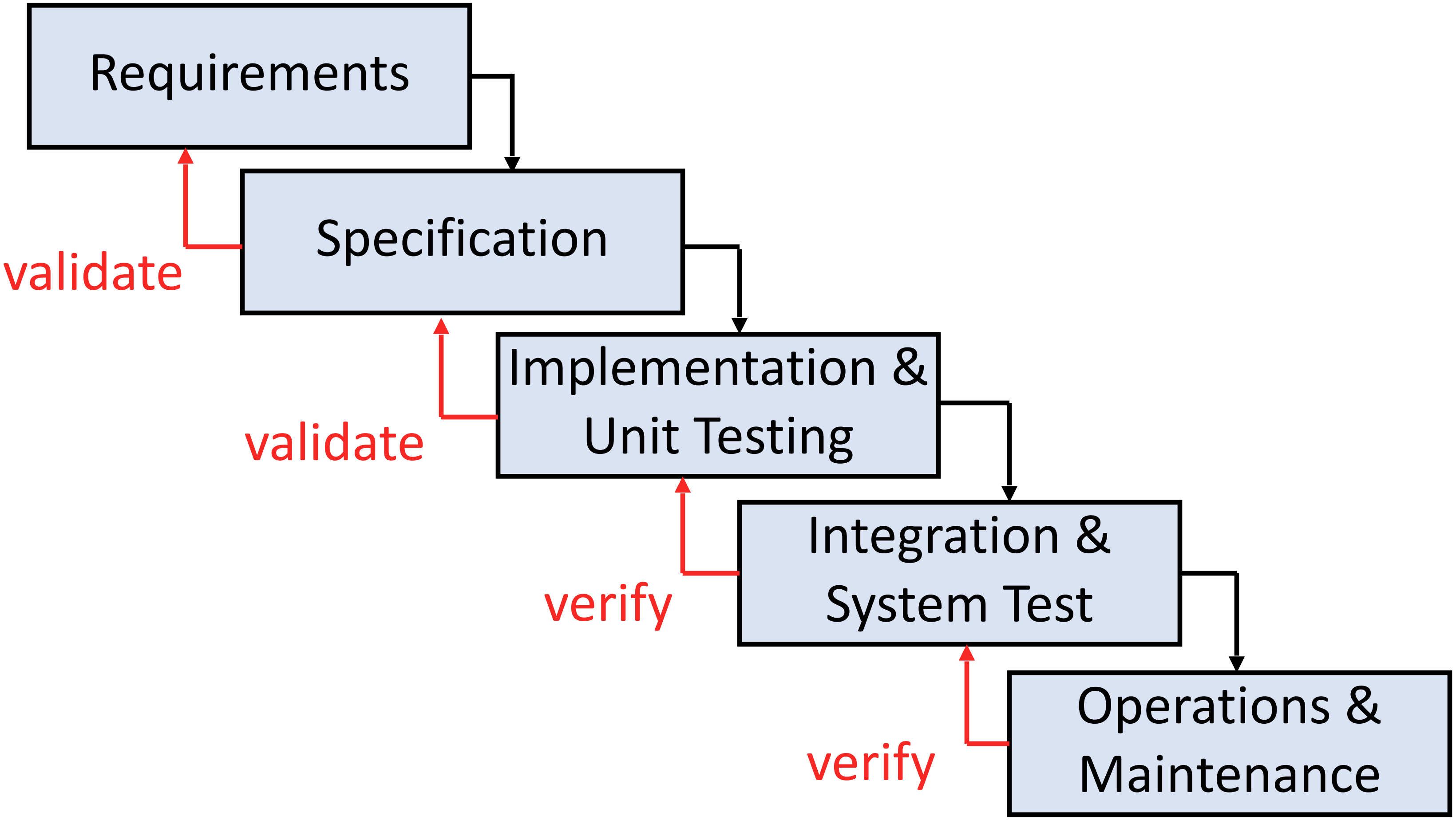
Specification

Implementation & Unit Testing

Integration & System Test

Operations & Maintenance

The Waterfall Model (1970)



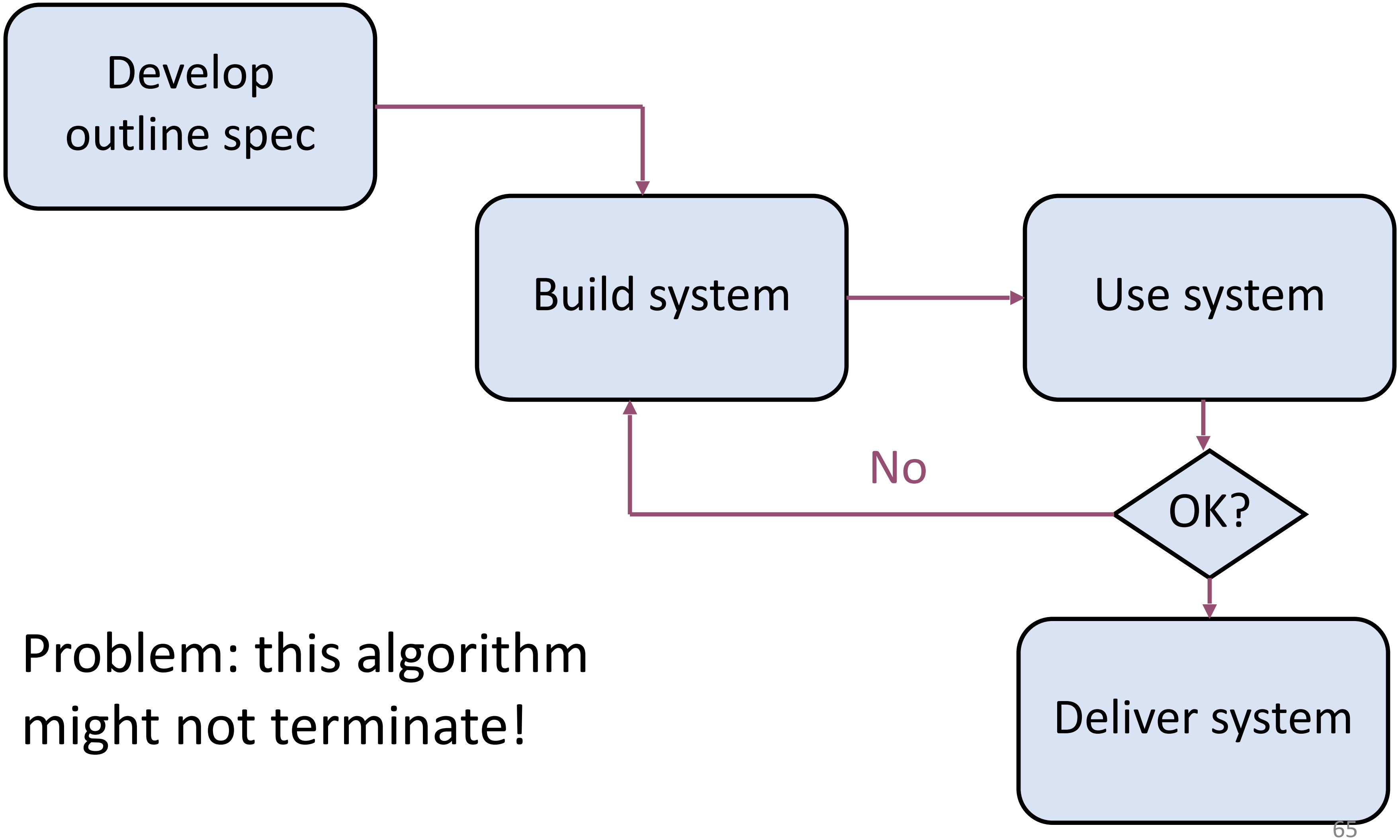
Waterfall Model has advantages

- Compels early clarification of system goals
- Supports charging for changes to the requirements
- Works well with many management and tech tools
- Where it’s viable it’s usually the best approach
- The really critical factor is whether you can define the requirements in detail in advance. Sometimes you can (Y2K bugfix); sometimes you can’t (HCI)

Waterfall fails where iteration is required, such as:

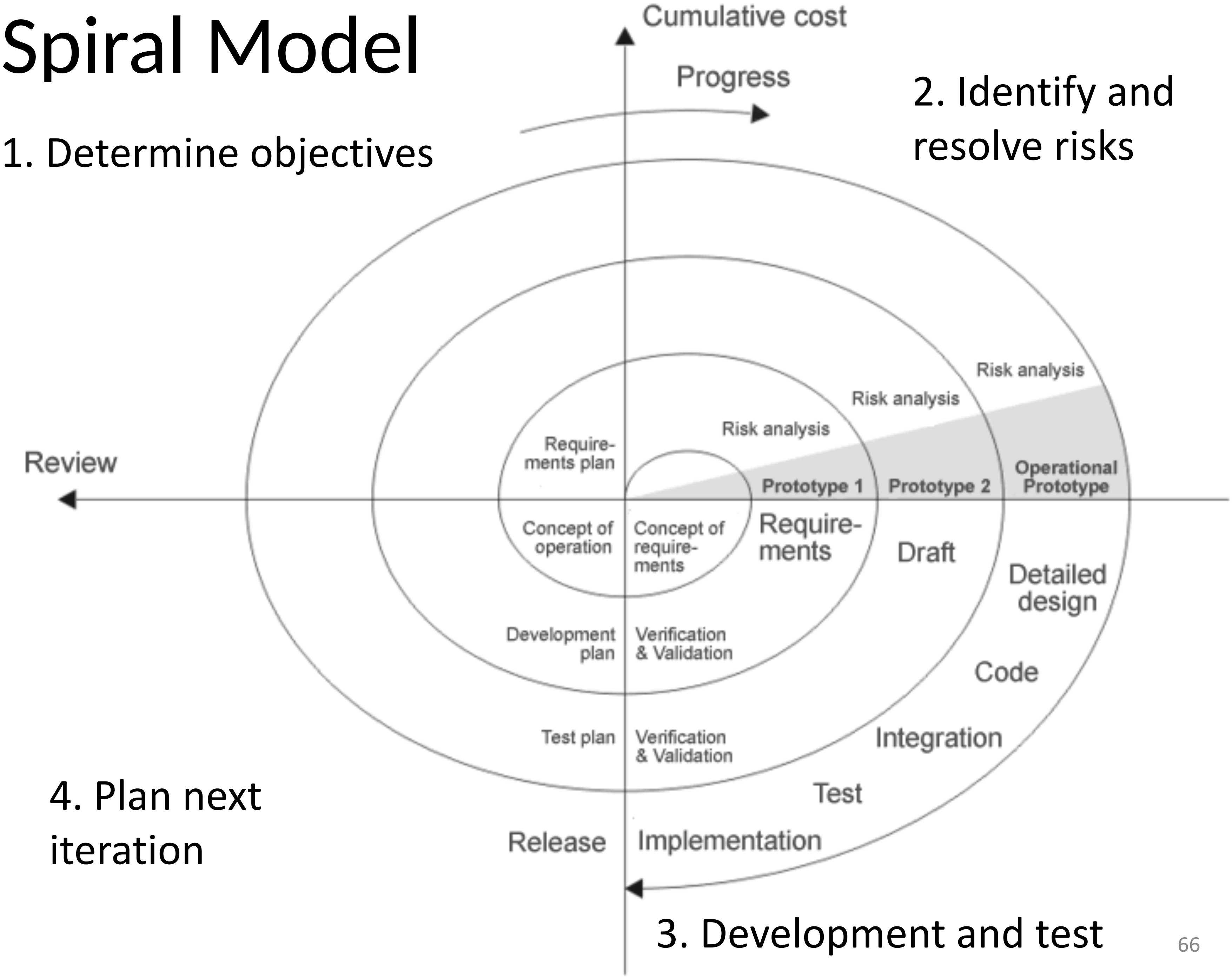
- Requirements not yet understood by developers
- Not yet understood by the customer
- The technology is changing
- The environment (legal, competitive) is changing
- …

Iterative development



Problem: this algorithm might not terminate!

Spiral Model



1. Determine objectives

2. Identify and resolve risks

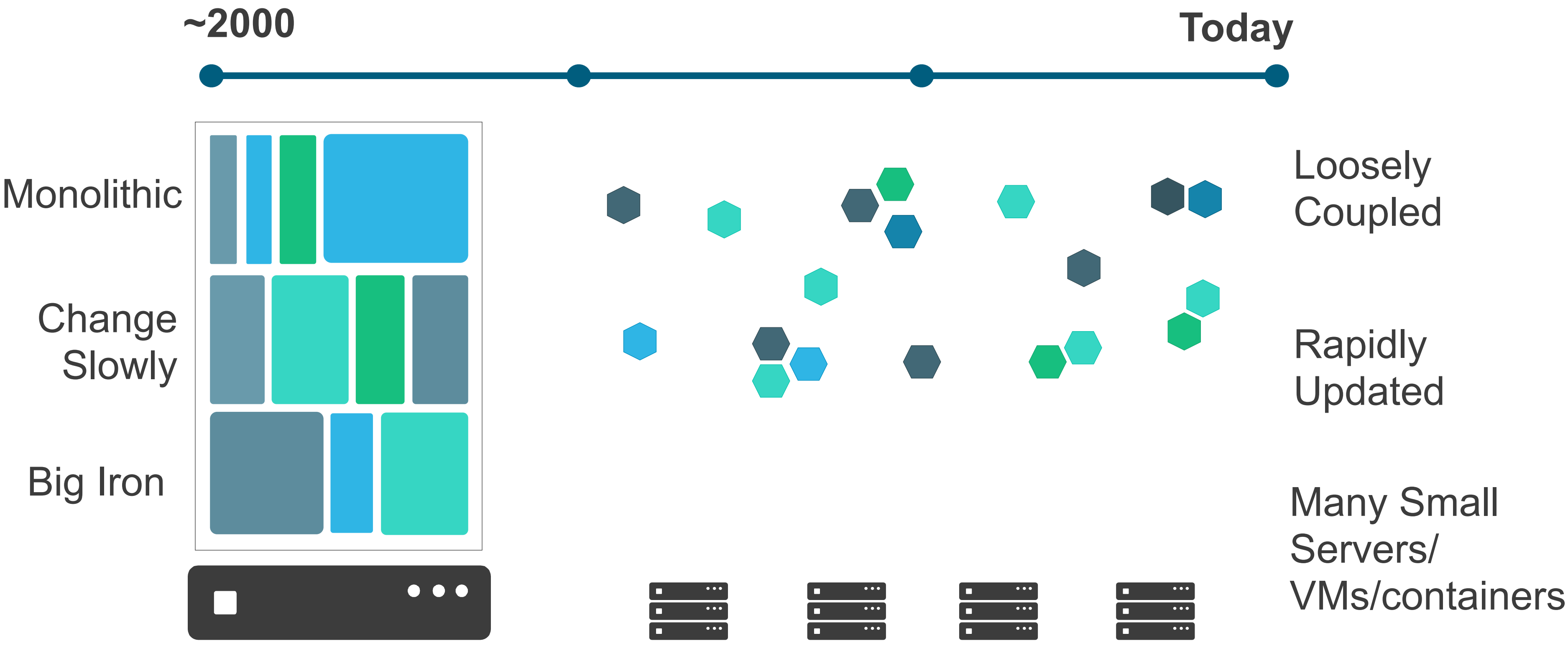
3. Development and test

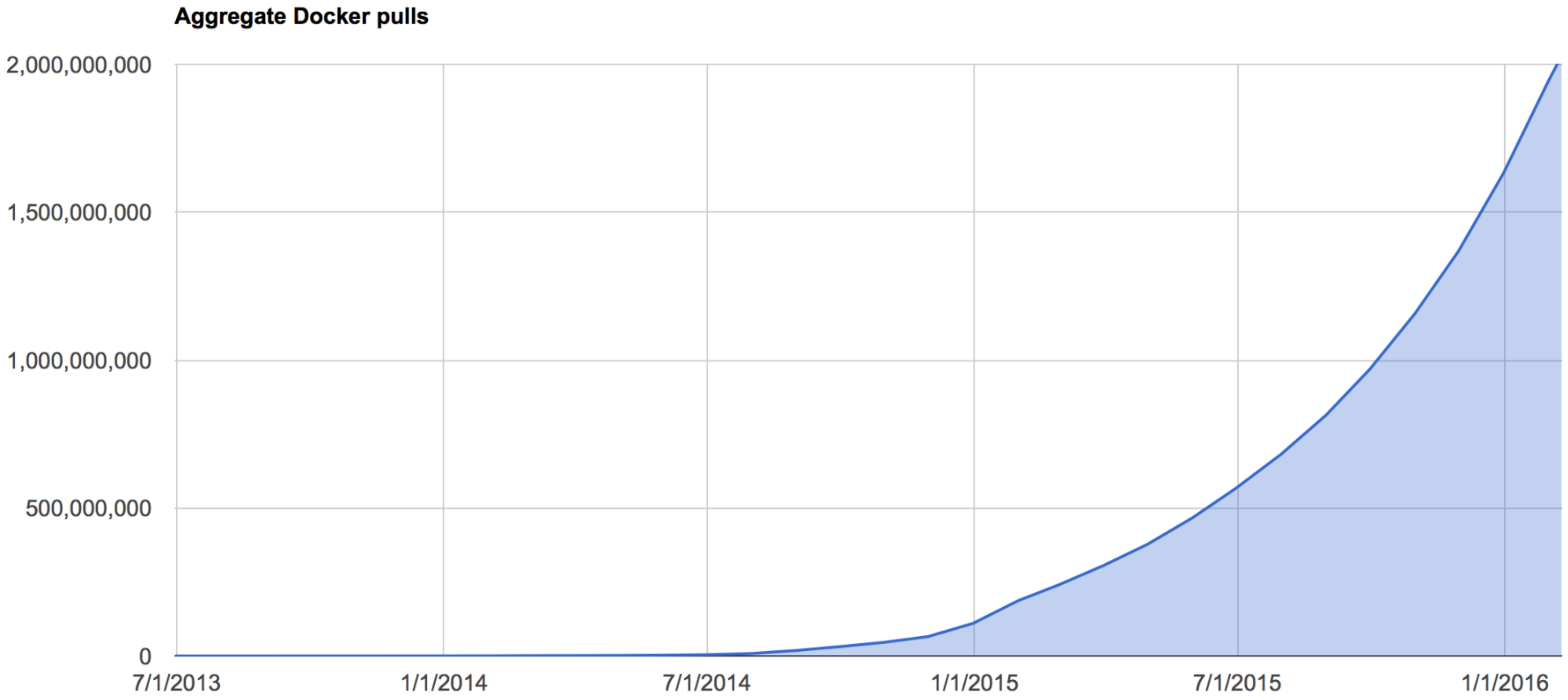
4. Plan next iteration

Spiral model invariants

- Decide in advance on a fixed number of iterations
- Each iteration is done top-down
- Driven by risk management (i.e. prototype bits you don’t yet understand)

Docker: Transformed the Development Landscape





Aggregate Docker pulls
2,000,000,000
1,500,000,000
1,000,000,000
500,000,000
0
7/1/2013
1/1/2014
7/1/2014
1/1/2015
7/1/2015
1/1/2016